Access Control for Large Collections

H. M. GLADNEY
IBM Almaden Research Center

Efforts to place vast information resources at the fingertips of each individual in large user populations must be balanced by commensurate attention to information protection. For centralized operational systems in controlled environments, external administrative controls may suffice. For distributed systems with less-structured tasks, more-diversified information, and a heterogeneous user set, the computing system must administer enterprise-chosen access control policies. One kind of resource is a digital library that emulates massive collections of paper and other physical media for clerical, engineering, and cultural applications. This article considers the security requirements for such libraries and proposes an access control method that mimics organizational practice by combining a subject tree with ad hoc role granting that controls privileges for many operations independently, that treats (all but one) privileged roles (e.g., auditor, security officer) like every other individual authorization, and that binds access control information to objects indirectly for scaling, flexibility, and reflexive protection. We sketch a realization and show that it will perform well, generalizes many deployed proposed access control policies, and permits individual data centers to implement other models economically and without disruption.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications; distributed databases;* D.2.0 [**Software Engineering**]: General—*protection mechanisms;* D.4.6 [**Operating Systems**]: Security and Protection; H.2.0 [**Database Management**]: General; H.2.8 [**Database Management**]: Database Applications; H.3.5 [**Information Storage and Retrieval**]: Online Information Services; H.3.6 [**Information Storage and Retrieval**]: Library Automation—*large text archives*

General Terms: Management, Security

Additional Key Words and Phrases: Access control, digital library, document, electronic library, information security

1. INTRODUCTION

Many projects are striving to make more information readily accessible. Increased access must be balanced by constraining users to authorized data [Hoffman and Moran 1986]. Adequate data security can be delivered by knitting together components that either exist today or are under consideration, except that no extant access control method combines everything

Author's address: IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120-6099; gladney@almaden.ibm.com.

needed. Existing, widely deployed access control tools are based on models devised 15–20 years ago. These are inflexible, do not provide all the functionality needed, and do not scale well to the collection sizes and user populations wanted. The research literature has not yet included attempts to cover all the requirements of practical systems (e.g., in Sections 3.1 and 4.2 below); in particular, common delegation patterns among office workers are not fully solved.

One kind of information resource is a digital library to emulate a document collection on paper and other physical media. ("Digital Library" is suddenly much discussed [Fox et al. 1994; 1995]; a digital library is a higher-level abstraction than a database, but might be implemented using relational or OO databases.) For about six years we have been developing a document storage subsystem, called **DocSS,** intended for clerical, engineering, and university libraries [Gladney 1993]. To make the access control discussion-specific, we will cast it in the context of this library service; we believe the approach, referred to as **DACM** (*Document Access Control Method*) below, is suitable for other kinds of data collection.

1.1 Overview

Our objective is to define access control data structures that can scale from small to very large collections, that permit decentralized administration of privileges, that can accommodate different rule sets controlling a single collection, that can model delegation patterns common in various organizations, that are as close to what users expect as possible (including conforming to some standards), and that permit efficient implementations. We have found a scheme that will be intelligible to end-users of document services.

Scaling to many objects is managed by having each object point to its access control rule set represented as another object, just as object-oriented languages and databases do to represent containment. Scaling to many users is handled by emulating vertical delegation in organizational hierarchies, extended to permit privilege delegation from any to any other node—up, down, or across the organization tree; this provides a way to represent special administrative roles like "security officer." Scaling to many yes/no privilege decision points is handled by assigning positions in long bitvectors; permission calculations become boolean operations on bitvectors. These choices yield the other properties wanted; for instance, access control objects point to their interpreters, allowing different rule sets to protect objects within a single collection.

The core of the article is the semiformal model in Section 2, which includes careful definitions of *library, privilege, role,* and other key terms. This is first sketched and then built up from primitive concepts in Sections 2.2–2.5. An objective is that every privilege is traceable as a sequence of delegations from a custodial user; the preferred model of delegation in the organizational tree includes the following rules:

—a custodian has all privileges on all objects;

—creating new subordinate users is itself a privilege whose user can grant only those privileges he himself has;
—any user can grant a subset of his or her privileges to another user; this is a role which is only effective while the second user claims to be acting for the first; the domain of a role is limited to the subtree rooted in the role grantor node;
—a user's privileges are the union of static privileges granted during user creation and the current role privileges;
—a user's privileges on an object are the intersection of his or her operator privileges and the privileges granted in the access control object pointed to by the object;
—ownership of an object is the privilege of binding an access control object; and
—search for a privilege set is upward in the organizational tree until the first hit.

This basic construction is implicitly biased toward discretionary access control; an extension deals with mandatory access control and shows how additional rule interpreters can be added to a basic implementation. Section 3 makes the core discussion concrete; because access control data are held in library objects, not much needs to be added to a digital library to provide control.

As requirements are carefully described elsewhere, the key ones are tabulated only very concisely before we plunge into the core description. A more careful treatment occurs later in the article when it is shown how well our model accommodates requirements specified by other workers. A discussion of costs focuses on what users have to learn and do to specify controls; administrative chores are as little as possible for the granularity of control wanted for the data at hand. The mechanisms for handling scaling—particularly that any number of objects can share an access control rule—create this economy. Economy of execution comes from the bitvector representation of privileges and because searches are only upward in a tree.

1.2 What Access Control Services Are Needed?

In principle, object access control is simply conformance to a rules array which records the privileges allowed to each subject for each object. Such an array is, however, impractical for even a small library because of the human effort to manage it. We want to replace the array by much smaller data structures which work well for both small (one user and a few hundred objects) and large systems (thousands of users and many millions of objects, which is what customers in insurance, banking, and taxation operations tell us they need). It must support heterogeneous applications: what a public service library needs (almost nothing) differs from what governmental oversight of toxic waste disposal might demand, and both differ from what an aircraft manufacturer needs.

Thus our objective is not to address the kind of requirements statement that might originate from a single application class, but rather to devise a broadly applicable model and implementation. The model will be deemed successful if the differences between applications can be reflected in tables without modifying tricky programs. In addition to well-known needs [Gladney et al. 1975], a comprehensive scheme must provide the following:

—decentralized administration of resource pools, because for large pools no single individual or department can know what controls are appropriate for everything;
—certifiable behavior at resource pool boundaries, so that service offerers can confidently enter explicit or implicit contracts to protect other peoples' data;
—smooth synthesis of mandatory and discretionary access control [Chokhani 1992], as might be needed by a company with both military and commercial work;
—support for generally accepted accounting principles, which require that each person is limited to resources needed to discharge his or her responsibilities (the principle of least privilege), that sensitive resources are modifiable only in partial steps by independent users (separation of authority as used for money management), and that outside auditors can easily review user actions both retrospectively and prospectively;
—freedom to define what it means to be a specially privileged user, such as an auditor; an enterprise might require its own definition of the operations associated with each such role and limit each instance to a limited data scope;
—differentiation of user roles from individual user instances (e.g., "payments office manager" instead of "Jane Doe");
—proxy support, in which a human user acting for another human temporarily gets partial privileges of the principal;
—fine granularity relative to operations, possibly with every operator separately controllable; for instance, a library administrator should be able to keep some users from destroying any objects, even those they themselves created;
—as little clerical burden as possible, as experience shows that users bypass controls which are tedious to administer; more precisely, proportional burden in which owners who are content to protect most resources similarly are not forced to specify control details needed by those wanting differentiated protection;
—the possibility of high-performance implementation, e.g., adding less than 5% to server elapsed time, compared to what uncontrolled service uses; and
—upward compatibility from prior methods such as RACF [IBM 1985], industry conventions such as OSF-DCE security [OSF 1991], and formal standards such as POSIX security [ISO 1991].

1.3 Bounds of the Topic at Hand

This article is about access control; security is a bigger topic. We assume that improved communication security, identity authentication, operating system basics, and the like come from a combination of well-known and emerging techniques. We further assume that certain key programs do precisely what they are specified to do, no more and no less, and that system components not discussed, working together with the access control mechanism, ensure that these key programs are not altered or bypassed except in approved ways. Our scope and jargon conform to the ISO security framework [ISO 1992] which articulates the role and limitations of access control.

We assume the protected objects are well-defined clumps of data; each clump might have many internal links but has relatively few external links significant to access control. Relational databases accessed by SQL are outside this scope because query sets do not partition the information. Document collections meet such restrictions imperfectly.

We limit ourselves to protection of the contents of widely shared data servers. A library might be the archive for smaller object-oriented databases, with each OO database acting as an archive cache, but we do not consider the security implications.

Sensitivity to environmental factors—time of day, workstation identity, and so on—is well understood. In a network whose data servers might run different operating systems than clients, the interpretation context for such factors is different from clients' contexts. This creates a modeling and programming language challenge, which we treat elsewhere [Gladney 1994].

A complete solution to distributed authorization management would allow subject descriptors to be shared in the network so that each subject would be described only once. Since to treat the topic would encounter interprocess authentication and protocol problems not yet worked out, we defer it.

We do not intend the current article to specify programming interfaces, syntactic details, or much about implementation method. That will be most evident in the discussion of access control records and the operators that modify them. It also figures in scant attention to whether expert systems technology can be used for the interpretation of access control lists (it can) and in the very limited discussion of protection for persistent data created by object-oriented languages. We carry the discussion just far enough that the semantics for library protection and the substitution alternatives for other applications are easily inferred.

2. A DOCUMENT ACCESS CONTROL MODEL

Security is conformance to proper authorizations for the movement of data out of a store into other stores—the rest of the world in Figure 1—and for changes made in this store responsive to instructions originating in other stores; the distinguished store is sometimes called the *protected resource(s).*

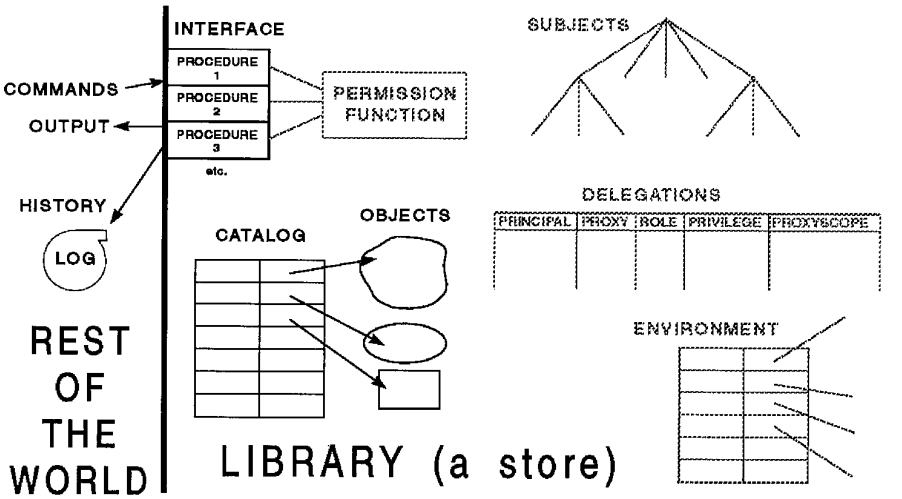

Fig. 1. A protected resource and the rest of the world. The black portions describe any object store; the grey portions are specific to Section 2.3 and to later sections of this article.

Procedures which define the sole external interface to what is stored—collectively called the *resource manager*—are part of the protected resource and are responsive to *commands*—bit strings passed from outside. Since distinct programs can effect the same state change set, we allude to each equivalent procedure set as an *operation* which might be subject to access control.

A *library* is a protected resource containing data sets called objects, documents, or items below (depending on the context) and a catalog which locates and describes each item in one or more records. Thus a library is a specialized form of protected resource, and **DocSS** is a specific resource manager.

Access control is a custodial contract[1] governing the relationship of the store with the rest of the world. Specifically, it has to do with the execution of operations which deliver information, or which change the state in a way that potentially affects future information delivery. *Access control* is said to be in effect if the store state permits any past action or future permission to be traced to proper authority and if such permissions faithfully reflect an articulated policy; in this the output stream is considered part of the state. The store is said to have *integrity* if its state conforms to articulated consistency rules. A subsystem which has integrity and access control is said to be *secure*.

A *reference monitor* is a subsystem which records who may do what, i.e., what is authorized, and provides conforming yes/no decisions responsive to queries by active system components. An access control mechanism necessarily has portions in the resource manager as well as in the reference

[1]The contract is usually implicit, with the custodian offering "If you store your data with me, I promise not to lose it and to ensure certain additional protections, viz., ..."

monitor, with the resource manager implementing the constraints defined by the reference monitor. An ISO access control framework [ISO 1992] delineates how to partition the functionality. The reference monitor for a particular store can be embedded within the store itself or be part of another protected resource; this distinction is unimportant in the current chapter.

Access control calculations are evaluations of a boolean function $B(u, r, c, o, \mathrm{S}, \mathrm{O}, \mathrm{E})$—the permission function—on the identity u and role r claimed by a user who asks for execution of some operation c on an object o, an object directory and object set O, and a subject graph S, all in the protected resource environment E. (The notation used below is shown in Table I, in which the domains of privilege sets, procedures, objects, subjects, roles, enumerations, and booleans are respectively denoted by **p, P, o, s, r, e,** and **b.**) Each tabulated row of this function is an *access control rule.* In theory, evaluation is trivial. However, tabulating $B(\ldots)$ is impractical, partly because of its size and partly because not all choices of the function express reasonable policies. Particular choices of $B(\ldots)$ will be acceptable if they allow all desired policies, will be efficient if some form for $B(\ldots)$ is concise, and will help enterprises achieve resource integrity if they embody policies wanted by every custodian.

2.1 Plan of Action

Everything that follows in this section deals with these problems. The size is handled by aggregating subjects and, separately, objects into classes; the constraint to useful policies is handled primarily by limiting interrelationships of subjects. An objective of the overall scheme is easily understood delegation rules in an implementation, for instance as in Section 3.4.

Before proceeding with the main business, we dispose of a technicality that would otherwise clutter what follows. Before anything of value is granted, a user identity check is made, possibly by calling an authentication server running *Kerberos* [Kohl 1991]. If this "logon" test succeeds, the connect operation binds the session to a subject, i.e., creates a 1-to-1 mapping from the user u to some subject s. Thus we can drop the user denotation u below and use the subject s it binds instead.

A *privilege* can be access to an operation or omission of some normally required validity check or audit trail addition. Even though *privilege* and *permission* are synonyms in normal usage, to help the reader distinguish what is tabulated from what is calculated we use *privilege* for an express access grant which is relatively persistent, i.e., a datum which is tabulated as part of the access control information, and *permission* for a value calculated by combining privileges, other resource state elements, and transitory circumstances. Thus, a privilege is a durable part of the resource state (in S or O) communicated by a resource manager command, and a permission is an ephemeral value used more or less directly to control execution.

The rules for deciding whether any privilege can be granted and for combining privilege sets to determine whether to permit an action are the

Table I. Definitions of Symbols Used in Equations and Text and Their Domains

Function or Object	Data Domain	Meaning or Intention
C	{**p**}	The set of all possible operation execution and other privileges.
O	{**o**}	The set of all library objects, considered together with all pertinent descriptors of those objects.
s_i	**s**	The ith subject, with s_c denoting the custodian.
S	{**s**}	The set of all subjects, i.e., S ≡ {s_i| $i \ni s_i$ is defined}.
E	{?}	The protected resource environment, comprising everything not explicitly mentioned but available for decision evaluation.
$g(s)$	**s** → **s**	The parent (group) of the subject s in the static grant hierarchy, typically associated with the subject's manager. The function g is complete, except that $g(s_c)$ is not defined.
$f(s)$	**s** → **s**	The administrator who creates a subject s. The function f is complete, except that $f(s_c)$ is not defined.
$m(s, r)$	**s** × **r** → **s**	The subject who grants the role r to s. The function m is partial.
$g(s, r)$	**s** × **r** → **s**	The root of the subject subtree whose owned objects are granted for a proxy to s for role r. The function g is partial.
$o(o)$	**o** → **s**	The owner of the object o.
$a(o)$	**o** → **o**	The access control object bound to the object o.
$r(o)$	**o** → **P**	The reference monitor bound to the object o.
$d(o)$	**o** → **b**	A flag which indicates whether the owner is permitted to change which access controls $a(o)$ and $r(o)$ are bound to the object o.
R(s)	**s** → {**p**}	The privilege set granted a subject s; s may use operation k if and only if $r_k \in$ R(s), i.e., C ⊃ R(s) ≡ {r_k|$f(s)$ granted k to s}.
D(s, s', s'', r)	**s** × **s** × **s** × **r** → {**p**}	The set of privileges that subject s grants to subject s' for the role called r. The grant is limited to objects owned by s'' and by s'' subordinates. D is complete, taking the value {} for arguments for which ProxyDefine has not been executed. D(s, s', s'', r) ⊂ C.
P(s, o)	**s** × **o** → {**p**}	The permission set of a subject s on an object o. P(s, o) ⊂ C.
P(s, r, o)	**s** × **r** × **o** → {**p**}	The permission set of a subject s exercising the role r on an object o. P(s, r, o) ⊂ C.
A(s, e, a)	**s** × **e** × **o** → {**p**}	The privilege set for the subject s in the access control object a; the enumeration e indicates whether the grant is to the subject itself, to the group of the subject, to the owner, . . . A(s, e, a) is partial, i.e., values are undefined for some subjects in some access control lists. A(s, e, a) ⊂ C.

same for most privileges. Changing object ownership (Section 2.6) is an exception, as is the privilege of bypassing tests dependent on object identifiers and contents, which is needed for maintenance activities such as data backup.

It is always possible to choose a function $B'(\ldots)$ which is insensitive to objects and an auxiliary function $B''(\ldots)$, described below, so that

$$B(s, r, c, o, \mathrm{S}, \mathrm{O}, \mathrm{E}) \equiv B'(s, r, c, \mathrm{S}, \mathrm{E}) \wedge B''(s, r, c, o, \mathrm{S}, \mathrm{O}, \mathrm{E}) \wedge B(s, r, c, o, \mathrm{S}, \mathrm{O}, \mathrm{E}). \quad (1)$$

$B'(s, r, c, \mathrm{S}, \mathrm{E})$ describes a user's permission to operations. $B''(s, r, c, o, \mathrm{S}, \mathrm{O}, \mathrm{E})$ can be any function which satisfies Eq. (1). Separating $B'(s, r, c, \mathrm{S}, \mathrm{E})$ is useful because whenever it evaluates to *FALSE* there is no need to evaluate the other factors on the right; for instance, if a certain user is not permitted to update anything, there is no need to prevent him or her from updating any particular object.

It is concise and efficient to handle all operations simultaneously by working with functions evaluating to permission sets, such as $\mathrm{B}(s, r, o, \mathrm{S}, \mathrm{O}, \mathrm{E}) \equiv \{B(s, r, c, o, \mathrm{S}, \mathrm{O}, \mathrm{E}) \mid c \in \mathrm{C}\}$. We replace Eq. (1) with the following:

$$\mathrm{B}(s, r, o, \mathrm{S}, \mathrm{O}, \mathrm{E}) \equiv \mathrm{B}'(s, r, \mathrm{S}, \mathrm{E}) \cap \mathrm{B}''(s, r, c, o, \mathrm{S}, \mathrm{O}, \mathrm{E}) \cap \mathrm{B}(s, r, o, \mathrm{S}, \mathrm{O}, \mathrm{E}) \quad (2)$$

The rightmost array explodes as the numbers of subjects and objects get very large. We show how to choose an economical $B''(\ldots)$ which makes it unnecessary to compute the rightmost factor.

To accomplish this, we collect objects and subjects into classes;[2] in fact, collecting subjects into groups is done in many access list schemes and is essential for POSIX compliance, as this security standard externalizes the concept of user groups [ISO 1991]. Each access control list is part of $B''(\ldots)$. A tabulation of access control rules would look something like the following:

OBJECT	USER	PRIVILEGES
id1	charlie	0001011111001...
id1	robert	0010011111001...
...	...	...
id2	charlie	1111111111001...
id9	james	0001010111001...
...	...	...

This relates objects and subjects to boolean vectors representing privileges. (We leave out a column needed to express constraints dependent on the

[2]This is similar to how symmetry is used in physics. Symmetries in a function allow it to be described more concisely than would otherwise be possible and are also closely related to what we mean when we say we "understand" some behavior.

environment and subject and object attributes [Gladney 1994].) Aggregating subjects and objects into equivalence sets allows us to reexpress this as a table with far fewer rows:

OBJCLASS	USERGROUP	USERKIND	PRIVILEGES
class1id	charlie	user	0001011111001...
class1id	robert	group	0010011111001...
class1id		owner	0010011111001...
class1id		public	0000001110000...
...	...	...	...
class2id	charlie	group	1111111111001...
class9id	james	dept.	0001010111001...
...	...	...	...

Here each first-column entry identifies an *access equivalence set*—a set of objects that have the same access control information; the set of rows with a common first column value is the *access control list* for the object class identified. The second column variously identifies the user, the group in which the user is a member, the owner of the object in question, etc. Which interpretation is intended is indicated by the value in the third column. **public** in the third column means that the privileges tabulated in the fourth column are for anyone at all.

In some implementations and in external views presented to users for inspection and editing, it is helpful to treat each access control list as an attribute of the object identified in the first column, which we then call an *access control object.* An access control object can be edited and otherwise manipulated as other objects.

In summary, access control rules are compactly represented by collecting them into equivalence sets, and portions are made externally accessible as portions of ordinary objects; how the rules are represented internally is left as an implementation choice (but Sections 3.3 and 4.3.2 make suggestions). A *permission function* interprets the tabulated information, adding logic which implements a widely accepted model of delegation.

2.2 Subjects, Roles, and Proxies

A subject is a potential resource user.[3] Formally, a *subject* is a privilege set for the store's operation and data resources. Privileges are granted by other subjects; each subject (except for one) has a parent and may have any

[3]Terms descriptive of human subjects, such as *user, administrator,* and *auditor,* are mapped by data blocks in computing systems—data blocks which become bound to executing processes. Relationships, such as *manager of* and *auditor of,* are mapped by cross-references among these data blocks. Delegation, considered in the abstract, is realized as a directed graph. Because the mappings are 1-to-1, the exposition can ignore the abstract-to-realization dichotomy, as is customarily done. However, the reader should keep in mind that every concept that follows is realized as a data structure whose correctness can be checked.

number of children, collectively called the *group*[4] of that subject. The subject graph is a delegation or grant hierarchy; we constrain it to a tree to avoid ambiguities that might otherwise arise in calculating permissions.[5] Rules which we will presently articulate constrain each subject to granting only privileges which he or she has.

The root of the subject tree, called the *custodian,* is a surrogate for the person[6] offering storage services and committing to users the integrity and security of data held. The custodian may be considered either to be acting as an enterprise agent (the authoritarian view) or to be offering a service for consideration (the contractual view[7]). The custodian has unconditional access to everything in the store, i.e., no access control checks are performed for a user logged on as the custodian, who is therefore analogous to a UNIX superuser or a SQL database administrator.

Roles are task-oriented relationships which recur within a community; examples are "is secretary to" and "is auditor for department." A *role* is a set of privileges required to accomplish a related set of tasks and is represented by a named bitvector. For instance, a *store administrator* allows subjects to connect to the store, defines their privileges, and administers some basic access control information; the custodian grants the privileges needed to meet these responsibilities to each store administrator, who is distinguished from other subjects only by having certain privileges not commonly granted.

Roles are often assumed for limited durations to accomplish well-defined task sets on circumscribed resources. Since users often find it convenient to share role designations, their named bitvectors are tabulated by a store administrator. However, an assumed role's effective domain must be limited to resources controlled by the subject who grants the role; for instance, the manager-secretary relationship may be homogeneous, but each manager can grant resources only from his or her own pool. Thus, a *proxy* is a subject's authorization that another subject may use a specific subset of the grantor's privileges.

A *role* is the exercise of a proxy, limited to the duration of a session; a user accesses a role by asking for it during logon. Connecting to a store is establishing an association between a user and a subject and optionally a role; for the period in which such a binding exists, the user gets the union of the privileges of the bound subject and role. In line with the principle of granting users only what they need to do their jobs (Section 1.2), we restrict each to at most one role at any moment and provide no recursive

---

[4]Here the distinction between an individual subject and a group is minimized. The ability to give privilege by virtue of being a group member is not lost. The structure can model the relationships among managers and departments.

[5]Other access control systems work differently. For instance, RACF [IBM 1985] permits user membership in several groups, but the user must declare which group he or she wants to connect to at any moment. Certain restricted DACM structures and modes give the same effect.

[6]See footnote 3.

[7]See footnote 1.

inheritance of privileges, because we find this sufficient to model the requirements (Section 4.2).

2.3 Structure of Access Control Information

To articulate the class of functions B″(. . .) that are expressive enough to satisfy Section 1.2, we define particular structure within the store of Figure 1, doing so for the class of protected resources called libraries. A library consists of a subject set S, an object set O, a proxy set D which records role grants from subject to subject, a directory which is the sole place recording the object location (making it a point of control), an exported history L which is a chronological sequence of records, and procedures which can be invoked from outside to update the library and export information. (L embeds a security log. Since we have nothing novel to say about logging or audit, L is not referred to again.) Collectively the procedures embed a set C of decision points which are resolved by permission function calls. An environment E, a map from identifiers to values, makes available everything else used in access control decisions.

An access control mechanism is part of such a library, consisting of one or more reference monitors. Each reference monitor consists of part of the description of subjects S, an object subset A called access control objects, a role grant array D, a permission function, and procedures to maintain the subject descriptions. Some reference monitor provides a binary value at each of the decision points C. Any library action is caused by a user u who may have claimed a role r; actions are permitted only if, at the time of the request, u is bound to some member of ⟨S, D⟩, i.e., u is "logged on" as some subject s possibly using some role r.

Permission calculations are evaluations of the set function B(s, r, o, S, O, E) on the directory, the subject tree, access control objects, the environment, and the identity s and role r of the subject who asks for execution of some decision point $c \in$ C, i.e., a requested calculation ⟨s, r, c, o⟩ proceeds if and only if $B(s, r, c, o,$ S, O, E) evaluates to *TRUE*. It is convenient to evaluate $B(. . .)$ all at once for all $c \in$ C; the set of privileges B(s, r, o, S, O, E) available to s on o is the subset of C for which $B(. . .)$ evaluates to *TRUE,* i.e., B(s, r, o, S, O, E) ≡ $\{c | B(s, r, c, o,$ S, O, E) = *TRUE*$\}$.

Subjects $s_i \in$ S form a tree with a root subject s_c representing the custodian. The model allows us to constrain subjects to be objects, i.e., require that S ⊂ O. Part of what describes subjects is a role grant relationship D (see Section 2.4): proxies are a partial function $m(s, r)$ from subjects and roles onto subjects whose privileges are delegated:

$$m(s, r) \quad \begin{array}{ll} = s' & \text{if } \exists\, (s', s, s'', r) \ni \mathrm{D}(s', s, s'', r) \text{ is defined} \\ \text{is undefined} & \text{otherwise} \end{array} \qquad (3)$$

Objects $o \in$ O are accessible only via the library directory. They are aggregated into sets with identical access constraints; the function $a(o)$

selects the access control object $a \in$ O bound to the object O. An access control object can describe itself, i.e., $a(o)=o$ is permissible, and $a(o)$ is partial, i.e., an object need not choose an access control object.

An access control object a contains a tabulation of privileges granted to various subjects—an *access control list.* The function A(s, e, a) which describes all the access control lists in a library is partial, i.e., values are undefined for some subjects in some access control lists. We partition A(s, e, a) among access control objects because doing so provides fast access to the pertinent parts of B″(s, r, o, S, O, E) and because access control objects are themselves convenient units of ownership. Privileges on objects do not imply privileges on their access control objects.

2.4 Subjects' Operation Privileges and Delegations via Roles

The protected object set, O, will be created by normal library operators. Subjects and relationships among subjects are needed mainly for resource management and must be created by operators provided just for this purpose. The following description of constructors for a subject hierarchy and a role delegation graph leaves out practical but well-known details of authentication and conveniences unrelated to access control.

Part of each subject descriptor is a definition of the set of operations it is privileged to use. Each subject is added to the access control information by a function **SubjectDefine:** {**s**} × **s** × **s** × **p** → {**s**} from a subject tree to a subject tree. Its other arguments denote, respectively, the new subject *s,* its parent subject $g(s)$, and a candidate operation privilege set O. The subject creating *s,* denoted $f(s)$, must be authorized to use the **SubjectDefine** operation; this is what we mean by *library administrator.*

The privileges granted to s are not necessarily related to those of its parent $g(s)$ in the subject tree, but are limited to those of the library administrator $f(s)$. Suppose that $s' = f(s)$ is the currently executing subject, that r' is the current role of the executing subject, and that $m(s', r')$ is the subject whose proxy is being used by s'. Then **SubjectDefine** fails if the parent $g(s)$ of the subject s being created is not in the pair of trees rooted in s' and $m(s', r')$, i.e., a user can create a new subject only downward in the subject tree, but this rule applies not only to his or her own subordinate tree, but also to the subordinate tree of his or her current proxy grantor. This can be used to enable upward administration of subjects without granting any other privileges high in the tree. For instance, the owner of the SALES node in Figure 2 could authorize his or her subordinate BILL to manage the SALES subject tree.

Given a candidate set of privileges Q, the operation privileges granted to s are defined by

$$\mathrm{R}(s) = \mathit{if}(\, f(s) \equiv s_{\mathrm{c}})\ \mathit{then}\ (\mathrm{C} \cap \mathrm{Q})\ \mathit{else}\ (\mathrm{R}(\, f(s)) \cap \mathrm{Q}), \tag{4}$$

i.e., the custodian can authorize all operation privileges; any library administrator can grant only the privileges he or she has. Here it is the statically defined privilege R($f(s)$) that determines R(s), not the calculated

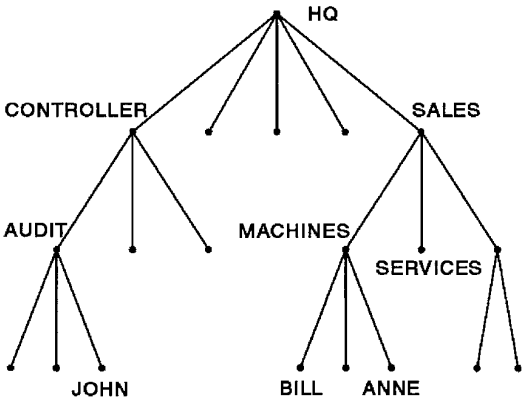

Fig. 2. Subject graph and delegation. For instance, the ordinary grant might be within the SALES department, and the proxy grant might be assigned to an internal auditor, JOHN, who is a member of an audit group temporarily assigned by the sales manager to audit the machines group.

(in Eq. (6)) administrator permission $P'(f(s), r)$; an administrator is not allowed to pass on any proxy powers that might enlarge the privileges of the new subject beyond what the administrator himself has been granted directly.

The delegation array $D(s, s', s'', r)$ records the privileges each subject s' wants to grant some other subject s and the data domain within which this grant is to be effective. **ProxyDefine: {p}** $\times$ **s** $\times$ **s** $\times$ **s** $\times$ **r** $\times$ **p** $\rightarrow$ {**p**} updates a privilege set in D. Its other arguments denote, respectively, the receiving subject *s,* the donor subject $s' = m(s, r)$, the domain $s'' = g(s, r)$ which is a subtree within which the delegation is valid, a name for the role r being granted, and the candidate privilege set Q. Both $g(s, r)$ and $m(s, r)$ must exist for this operation to succeed. $g(s, r)$ is provided so that the scope of delegation, a subset of the object space O, is conveyed. This space is all objects owned by s' and descendants of s', a subordinate subject—providing the means for a proxy to be limited to a subtree of the tree of the grantor.

Granting can enlarge the portion of an organizational hierarchy (subtree of the subject tree) to whose objects the grantee has access; it can include tree portions that are subordinate to s' but not already accessible to s. This permits a custodian to grant administrative privileges to subjects low in the tree, when these subjects are to administer objects high in the tree. For instance, a security auditor, JOHN (Figure 2), could receive a proxy to inspect but not change objects in an entire library even though this individual, acting for himself, has broad functions only on a more limited set of objects.

2.5 The Basic Permission Decision

The set of privileges $P(s, r, o)$ available to s and o is the subset of C for which $B(\ldots)$ evaluates to *TRUE,* i.e., $P(s, r, o) \equiv \{c | b(s, r, c, o, S, O, E) =$

TRUE}. Since B(s, r, o, S, O, E) is evaluated in the environment of fixed S, O, and E, these arguments can be taken as understood, writing explicitly only what passes across the library interface and evaluating a function P(s, r, o), defined below.

Permission $p_c(s, r, o)$ for subject s with role r to execute operation c on item o is available only if the subject is permitted to the operation and separately to the object, i.e., we rewrite Eqs. (1) and (2):

$$B(s, r, c, o, \mathrm{S}, \mathrm{O}, \mathrm{E}) \equiv p_c(s, r, o) \in \mathrm{P}(s, r, o) \tag{5}$$

$$\mathrm{P}(s, r, o) = \mathrm{P}'(s, r) \cap \mathrm{P}'(s, r, o)$$

$$\text{where } \mathrm{P}'(s, r) \equiv \mathrm{B}'(s, r, \mathrm{S}, \mathrm{E})$$

$$\text{and } \mathrm{P}'(s, r, o) \equiv \mathrm{B}''(s, r, o, \mathrm{S}, \mathrm{O}, \mathrm{E})$$

This says that the overall set of permissions is the intersection of this subject's operation permissions and his or her object permissions. The operation permissions are

$$\mathrm{P}'(s, r) = \mathrm{R}(s) \cup (\mathrm{D}(m(s, r), s, g(s, r), r) \cap \mathrm{R}(m(s, r)), \tag{6}$$

i.e., s has the union of the privileges granted when he or she was defined to the library and those granted him or her in the assumed role r granted by the principal $m(s, r)$. What is inherited from $m(s, r)$ is limited to privileges that $m(s, r)$ currently has. If a user s grants privileges to a user s', the effective grant is no greater than what s has when s' exploits the grant—rather than what s had at the time s granted to s'. Thus if the privilege set of s is curtailed, the grant to s' is also curtailed when it is next recalculated. For the case in which either the proxy parent function $m(s, r)$ or the role domain function $g(s, r)$ does not exist, the delegation array D is defined to confer no privileges, i.e.,

$$\begin{aligned} \mathrm{D}(s, s', s'', r) &= \mathrm{D}(m(s', r), s', s'', r) && \text{if } s = m(s', r) \ \& \ m(s', r) \text{ is defined} \\ & && \quad \& \ g(s', r) \text{ is defined} \\ &= \{\} && \text{otherwise} \end{aligned} \tag{7}$$

In the object-sensitive part $P'(s, r, o)$ of Eq. (5), a subject gets access to an object by being mentioned in that object's access control object or by having a parent, grandparent, . . ., whose group is mentioned, with the lowest superior node in the delegation graph being the effective one. Let us consider P'(s, r, o) from the bottom up.

Each access control object a contains a set of triples $\{\langle s, e, \mathrm{A}(s, e, a)\rangle\}$ defining the privileges of some set s of subjects. How the value s is to be interpreted is indicated by the enumeration argument e; this denotes any of $e_s \equiv$ "the subject s itself," $e_a \equiv$ "anyone at all," $e_p \equiv$ "the set of subjects whose parents are the parent of s (the group of s)," $e_o \equiv$ "the owner of the

object being controlled," $e_g \equiv$ "the group of the owner of the object being controlled," and so on.[8] What we want is a function that evaluates to all privileges if s is the custodian s_c, to the owner's privileges if the current subject is the owner $o(o)$ of the object o in question, to what is mentioned explicitly for s if $\langle s, e_s \rangle$ is found, and recursively up the group hierarchy otherwise. An auxiliary function $R(s, a, o)$ on a subject s, an access control object a, and an object o expresses the desired behavior; if neither s nor an ancestor of s is mentioned in a, $R(s, a, o)$ evaluates to the empty set, i.e., nothing is granted to s.

$$\begin{aligned} R(s,a,o) &= C && \text{if } s = s_c \\ &= A(s, e_o, a) \cup A_{all}(a) && \text{if } s \neq s_c \ \& \ s = o(o) \ \& \ A(s, e_o, a) \text{ is defined} \\ &= A(s, e_s, a) \cup A_{all}(a) && \text{if } s \neq s_c \ \& \ s \neq o(o) \ \& \ A(s, e_s, a) \text{ is defined} \\ &= G(s, a) \cup A_{all}(a) && \text{otherwise} \end{aligned} \tag{8}$$

where the effect of public access privileges is reflected by

$$\begin{aligned} A_{all}(a) &= A(e_a, a) && \text{if } A(e_a, a) \text{ is defined} \\ &= \{\ \} && \text{otherwise.} \end{aligned} \tag{9}$$

The auxiliary function $G(s, a)$ covers the possibility that s is not explicitly mentioned in a, but that some ancestor of s is mentioned; it is defined by the recursion

$$\begin{aligned} G(s, a) &= \{\ \} && \text{if } s = s_c \\ &= A(s, e_p, a) && \text{if } s \neq s_c \ \& \ A(s, e_p, a) \text{ is defined} \\ &= G(g(s), o) && \text{otherwise.} \end{aligned} \tag{10}$$

Eqs. (8), (9), and (10) show how to evaluate $R(s, a, o)$, which is the privilege set that access control object a awards to subject s for any object o that a happens to protect. The conditions on the right-hand side ensure that the expression for $R(s, a, o)$ defines a partial function; they become a trivial **case** statement in the permission evaluation program. The access control list entry of any subject explicitly mentioned is selected preferentially over any subject group entry that implies this subject, and among groups the entry for the lowest one in the hierarchy is the one selected; these preferences make it possible to define access control lists that deny privileges selectively within a group. If there is a public privilege entry $A(e_a, a)$, every subject gets at least these privileges.

Now a protects an object o if the catalog entry for o says that a is its access control object, i.e., if $a = a(o)$. Combining this with Eq. (9), we find

[8]The expressions beyond this point in the article will be kept concise by limiting them to the first four values of this enumeration; the extension to other values is obvious.

the permissions on o awarded to s because of the position of s in the subject tree S to be

$$\mathrm{P}'(s, o) = \mathrm{R}(s, a(o), o). \tag{11}$$

This value is the direct permissions s has to use o; s also derives permissions by virtue of acting in a role r if the grantor of r, or some parent of the grantor of r, is mentioned in the access control list $a = a(o)$. This contribution is expressed by

$$\begin{aligned} \mathrm{P}''(s, r, o) &= \mathrm{D}(m(s, r), s, g(s, r), r) \cap \mathrm{R}(m(s, r), a(o), o) && \text{if } q(s, r) \\ &= \{\,\} && \text{otherwise} \end{aligned} \tag{12}$$

in which the first factor on the right evaluates the privilege subset granted by the delegator $m(s, r)$ on o and where the qualification $q(s, r)$ is

$$\begin{aligned} q(s, r) = (&m(s, r) \text{ is defined}) \ \&\ (g(s, r) \text{ is defined}) \ \& \\ &(\exists\, (s', n \in \mathrm{N}) \ni (s' = o(o)) \ \&\ (g^{n}(s') = g(s, r)). \end{aligned} \tag{13}$$

The interpretation of the qualification $q(s, r)$ is that permission is awarded by a proxy only if there is a subject $m(s, r)$ who has granted a proxy, if the proxy grant defined a subject subtree within which it was valid, and if some ancestor $g^{n}(s')$ of the owner $s' = o(o)$ is the subject group $g(s, r)$ to which privileges were granted. N is the set of natural numbers.

Finally, the permission of s on o, ready for substitution in Eq. (5), is

$$\mathrm{P}'(s, r, o) = \mathrm{P}'(s, o) \cup \mathrm{P}''(s, r, o). \tag{14}$$

Eqs. (5)–(14) express a complete function, so that evaluation order in a reference monitor is unimportant. These superficially formidable functions in fact do little more than express ordinary managerial delegation down an enterprise hierarchy, extended by the temporary proxies, as illustrated in Figure 2.

2.6 DACM with Mandatory Controls and Enterprise-Chosen Alternatives

Section 2.5 presumes that the single permission function described will satisfy everyone; in fact, it assumes a discretionary control regime, even though the distinction between this and what a mandatory control scheme demands is not mentioned. Some enterprises may want to implement special policies. Some operating systems implement different models which their users will want to continue even if they are otherwise induced to exploit **DACM**. This is accommodated by allowing any object o to select not only an access control object $a(o)$, but also a permission function $r(o)$.

We therefore take up extensions needed to allow a single resource manager to hold objects controlled by different rule classes. For some control rules, only part of what is needed can be implemented within the reference monitor; further restrictions are needed within the operations of the resource manager. For instance, mandatory rules typically require that a secret document be copied only into a document labeled as secret, top secret, or some higher classification. In addition, mandatory control rules require [Department of Defense 1985] labeling each item with a hierarchical status (e.g., "Secret") and with an administrative domain (e.g., "Department of the Navy") and associating similar tokens with subjects; such labels are supported (Section 3.3).

The fundamental *mandatory access control* (*MAC*) restriction is that unprivileged subjects should not be able to cause information labeled at some sensitivity $\mathbf{L}_1$ to become accessible to subjects labeled as having $\mathbf{L}_2$ permission unless $\mathbf{L}_2$ dominates $\mathbf{L}_1$ [ISO 1991]. One consequence is that, under MAC rules, fixing who can do what to each object must be limited to individuals other than the users of the objects—individuals commonly called security officers; a *discretionary access control* (*DAC*) regime does not demand such separation of authority. In **DACM**, this distinction between MAC and DAC affects how object ownership is handled. Under DAC, an object owner is understood to be a subject who is authorized to change the object's access controls. Under MAC, an object owner is merely the subject on whose behalf an object is held.

DACM calculates permission to an object using a table and a procedure bound by the object's catalog entry—a level of indirection needed to allow different rules for different objects, aggregation, and custodian-chosen policies. If the access control object used as part of a MAC rule were to allow anyone other than a security officer to update it, users could evade the intention of mandatory control. Thus each object's catalog record must indicate whether DAC or MAC applies, and any storage manager operation which changes ownership or access control bindings must enforce the indicated rule.

The other aspects of MAC are particular limits on data changes and disclosures [Bell 1975]—limits which must be implemented in the resource manager. These include (1) binding security parameters closely to data objects so that, when objects are copied, the security information becomes part of the updated object and (2) limiting users to labeling objects into certain security classifications. Since **DACM** neither adds to nor alters such well-known practices, we do not discuss them further, but restrict ourselves to what must be included in the reference monitor for ownership and security officer privileges.

To what has already been defined, we add a boolean function $d(o)$ which indicates whether an object o is subject to mandatory (MAC) or discretionary access control (DAC); this is a function $d(o)$ on the object itself, rather than part of an access control object. We represent $d(o)$ by a value tabulated within each object's primary catalog and do the same for $a(o)$, which identifies the access control object protecting o, for $o(o)$, which

Table II. Protection Depends on Existence of Function Portions

	/ r(o)	$r(o)$
/ $a(o)$	Unprotected	Protected if the catalog record for o names a "special rule" in $r(o)$
$a(o) \wedge (r(a(o))$	Unprotected	Protected using $a(o)$ and $r(o)$
$a(o) \wedge \quad (r(a(o))$	Protected using $a(o)$ and $r(a(o))$	Protected using $a(o)$ and $r(a(o))$, ignoring $r(o)$

identifies the owner of o, and for $r(o)$, which identifies the permission function to be used to interpret the access control information. Only o's owner or a security officer is permitted to change $d(o)$, $r(o)$, $a(o)$, or $o(o)$ (the owner function). For objects subject to MAC, only a security officer may change these bindings.

This leads us to reconsider the meaning of ownership, extending the model slightly to accommodate what we think people expect. Under DAC, since an object owner can change the access list to give himself or herself any privilege (effectively limited to operations he or she has been allowed by a library administrator), we allow everything directly; under MAC, this shortcut is not permitted. The DAC shortcut (which can be used to enhance performance) is expressed by replacing Eq. (5) by

$$\begin{aligned} \mathrm{P}(s, r, o) &= \mathrm{C} && \text{if } s = o(o) \ \& \ d(o) = TRUE \\ &= \mathrm{P}'(s, r) \cap \mathrm{P}'(s, r, o) && \text{otherwise.} \end{aligned} \tag{15}$$

Protected objects do not always need access control objects. For instance, an object which should be equally accessible to everyone can bind a privilege set and permission function to itself directly. This is actually more useful than it might seem, because it can be used in conjunction with limiting which operators each subject may use. This feature enables custodian-defined fast paths; for instance, it could be used to implement the semantics of UNIX file permission bits.

In summary, **DACM** behavior depends on whether or not a permission function and/or an access control object is defined for the object o, as shown in Table II. The access paths for information used in permission decisions are collected in Figure 3, which suggests a relationship of **DACM** to object-oriented computing models.

2.7 Embedding Access Control in Subsystems

A popular line of thought, exemplified by OSF-DCE [Kumar 1991], emphasizes a network of mutually supportive resource managers, each providing a specialized service to multiple concurrent clients. Figure 1 is redrawn in Figure 4 to emphasize how such resource managers embed themselves in networks and interact. Each resource manager avoids reproducing what it can get from siblings, calling them instead. Each resource manager distributes itself, hiding whether and how it uses the network from its callers.

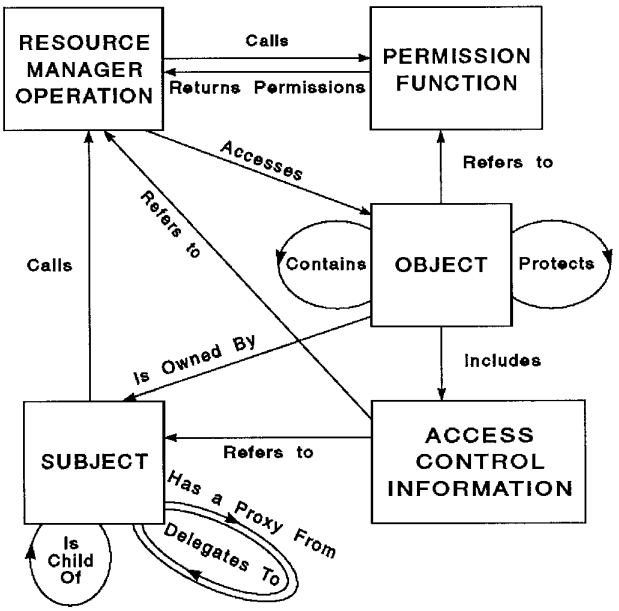

Fig. 3. Entity-relationship diagram describing access control data structures. An access control object is a special case of an ordinary object.

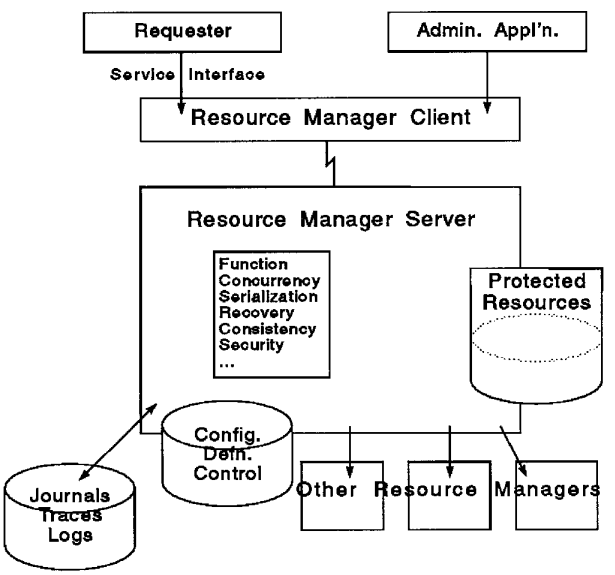

Fig. 4. Client-server structure for a protected resource, being one way of providing isolation demanded by Figure 1.

Each resource manager encapsulates a protected resource, providing a certified approximation to data quality management encompassed in the ACID (**A**tomicity, **C**onsistency, **I**ntegrity, **D**urability) properties [Gray and Reuter 1993].

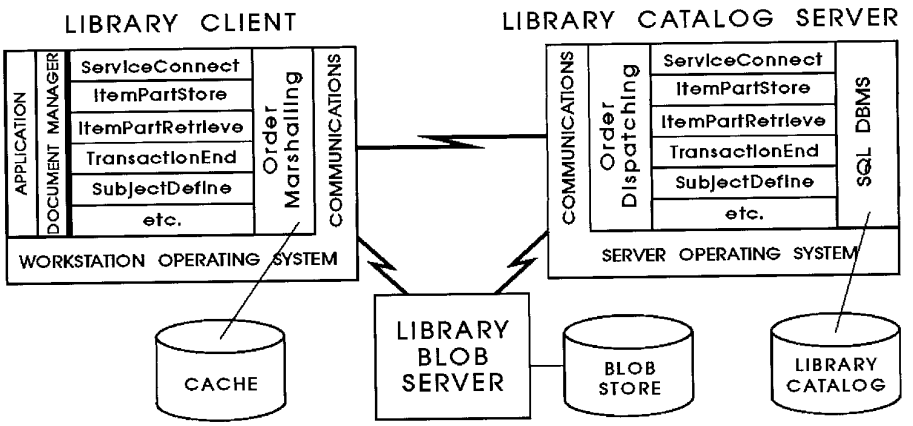

Fig. 5. Structure of a library service instance. The permission function is called by the catalog server subroutines. Programs which maintain subject descriptions (SubjectDefine and ProxyDefine) are primitive operations similar to ItemPartStore.

Since a library can catalog objects it does not itself hold, the figure suggests that a **DACM** implementation could be used as an external reference monitor for some other protected resource **D.** For this, a **DACM** instance needs access to **D**'s subject and object descriptions. **D** must invoke the reference monitor to create a surrogate object for each controlled resource, to build access control objects, subject descriptors, and proxy records, and to inquire whenever a permission is sought.

A **DACM**-protected library has been built with this structure, using SQL database and file resource managers as the implementation vehicle. The client-server separation provides the world/library isolation needed. If **DACM** is to be used to protect the resources of an operating system (e.g., the file system) rather than the contents of a server subsystem, the world/library isolation must be provided by executing critical portions of the resource manager and the reference monitor with hardware protection. As this is a well-known practice, it merits no further discussion.

3. IMPLEMENTATION WITHIN A DOCUMENT STORAGE SUBSYSTEM

DocSS mimics the simplest aspects of a conventional library, storing and cataloging objects for the benefit of workstation clients. It controls where library data are stored and manages distribution over local and wide-area networks, encapsulating everything needed to implement custodial responsibilities for enterprise data resources.

3.1 A Document Storage Subsystem

The library paradigm splits the world into the domain of the librarian and the domain of each user. The librarian assumes custodial responsibility for objects held for users' benefits, making specific commitments about data integrity and confidentiality. Access control rules are the articulation of

these commitments; since such rules must be protected similarly to the library contents, we made them part of the library catalog.

In a layered software system (Figure 5), the document storage subsystem layer avoids modeling, but is part of every access path to library data. Client-server logic hides communications from other programs. A document manager realizes a conceptual document and information web model—the digital analog of a collection of reading materials. Typical document managers—folder managers, computer-aided design tools, software library systems, and the like—interpret scanned and keyed data to create catalog entries automatically, manage interrelationships among documents, facilitate the most common search methods, and help move information among workers.

Catalog servers control access to each library, to library operations, and to each item within a library. For instance, a subject who wants to discard something needs permission to use the containing library, permission to use. . . .**Discard** in that library, and discard permission on the item selected. No privilege implies any other privilege, e.g., discard permission does not imply retrieval permission. Library catalog queries are compiled SQL programs for which access is managed exactly as for items.

A library *item* is a set of blobs associated for common administration. A *blob* is a bit sequence which can represent the content of a page, a picture, or any other data collection. All blobs in an item get the same protection. Whenever a library catalog server routine attempts to touch an item, it calls the permission function and implements its decision about granting or denying the requested action. As every blob server access is mediated by the catalog server, blob servers do not need their own access control service. The parts of an access control implementation, described below, are as follows:

—one or more permission functions, which are called as subroutines of ItemPartStore, ItemPartRetrieve, . . ., depicted in Figure 5;
—tables describing subjects and proxies;
—client-server subroutines SubjectDefine and ProxyDefine to maintain these tables;
—a few fields in each item's primary catalog entry;
—a single table collecting access control lists; and
—facilities which permit custodians to replace or extend these parts.

3.2 Defining and Binding Subjects and Roles

Access to a library is by means of a **ServiceConnect** call, which claims a subject identifier and a role and which is validated by a conventional identification and authentication process. **ServiceConnect** returns a **LibSession** handle which binds the subject and role to subsequent library service calls.

SubjectDefine: LibSession TableAct SQLuserid Password SubjectID SubjectID PrivilegeList ItemID StoreID ItemID Days allows a

library administrator to define a new user and his or her privileges, or to modify them, and to assign subjects to subject-groups, e.g., to reflect an organizational hierarchy. To the arguments essential for access control, it adds conveniences such as defaults for obligatory catalog fields for new-object creation. One of these indicates which access control object is attached whenever the new subject creates a new object.

ProxyDefine: LibSession TableAct SubjectID SubjectID Privilege-List SubjectID allows any user to grant a subset of his or her own privileges to another subject, possibly limiting the effect to objects whose owners are within a portion of the organizational hierarchy. A role name is associated with this delegation. For instance, a manager might direct, "Give Michelle, acting as SECRETARY (the role name), all my read privileges for data owned by department XYZ."

3.3 Library Catalog Schema

Only library catalog tuples that describe items, queries, subjects, and proxies are pertinent for what follows. ITEMS is a view with a single record for each item in the library; its fields pertinent to access control are

ITEM ItemID	CONTAINER ItemID	OWNER SubjectID	ITEMFLAGS Byte	SECURITY SecCode	ACCESSRULE ItemID	... ...

—when prefixed by the network identifier of the library, the key ITEM is unique worldwide and in eternity;

—CONTAINER indicates which item "contains" the current one; cycles are prohibited;

—OWNER identifies the subject which controls access to this item; however, see Section 2.6;

—two bits in ITEMFLAGS indicate how the ACCESSRULE field is to be interpreted:

00—this is a privilege vector;
01—the field holds the identifier of an access control item;
10—the field identifies a special rule implemented by the reference monitor;
11—the access control rule is found in the ITEMS record of the CONTAINER object.

—two bits in ITEMFLAGS select which reference monitor to use in deciding access, with 00b choosing the reference monitor described in this article;

—ITEMFLAGS indicates whether the item owner may change the OWNER, SECURITY, ITEMFLAGS, and/or ACCESSRULE fields or whether this may be done only by a security officer;

—ITEMFLAGS indicates whether the permission calculated for the item itself is to be ANDed with the permission calculated for its container to arrive at a final permission; this is recursive up the container hierarchy; a folder manager can use this to derive document permissions from folder permissions and to limit access to annotations to a subset of access on the annotated object (this feature is outside the **DACM** model—a weakness);

—SECURITY is a data security label, e.g., "IBM Confidential."

—ACCESSRULE identifies an access control object, a special ADF rule, or a null.

Access control information in the table describing compiled library queries is similar to that in ITEMS; since creating a query can expose any part of the library catalog, this is a privilege which should be granted only to administrators.

SUBJECTS describes subjects and subject groups:

SUBJECT SubjectID	MANAGER Subjectid	PRIVILEGE PrivilegeList	SUBJECTDESC ItemID	ACCESSRULE ItemID	SECURITY SecCode	GRANTOR SubjectID	... ...

—SUBJECT identifies whose library privileges are recorded;

—MANAGER identifies the group to which this subject belongs;

—PRIVILEGE defines the operations this subject may use and other privileges, e.g., to bypass logging required of most subjects;

—SUBJECTDESC identifies an object which describes this subject;

—ACCESSRULE identifies the access control object to be automatically attached to items this subject creates;

—SECURITY records the security clearance(s) of this subject;

—GRANTOR identifies, for audit purposes, which library administrator created this subject.

The DELEGATION view defines roles as subsets of subject privileges and the subjects which may exercise these roles. A principal may grant named roles to several subjects (e.g., the custodian may authorize several different subjects to act as auditors), and several principals may grant similarly named roles to different subordinates (e.g., each manager may delegate to several secretaries). When a subject claims a role, it is derived unambiguously from a single principal (e.g., "agent for the purchasing manager").

PRINCIPAL SubjectID	PROXY SubjectID	ROLE RoleName	PRIVILEGE PrivilegeList	PROXYSCOPE SubjectID	GRANTOR SubjectID

—PRINCIPAL identifies the subject whose privileges are partially granted to the PROXY;
—PROXY identifies the subject who may adopt the role;
—ROLE identifies the role defined in this record;
—PRIVILEGE defines the subset of the principal's privileges granted to the proxy;
—PROXYSCOPE defines the scope granted—the resources of a subject subtree;
—GRANTOR identifies which subject created this role.

A PrivilegeList represents a privilege set, encoded as a bitvector with each position assigned to a library operator or other privilege. Thus privilege set union and intersection are calculated by bitwise OR and AND respectively.

3.4 Delegation and Access Control Decisions

The rules for delegation are summarized by the following:

(1) The custodian is permitted every operation on every object. The custodian can grant any privilege to any subject, including bypassing privilege checks.
(2) The *privileges* of a subject are what is granted as part of admitting him or her to library service with SubjectDefine.
(3) The *permissions* of a subject are the union of his or her privileges and those of the role he or she successfully claims when connecting to the library.
(4) When a subject grants a role, this is to another specific subject and is limited to the objects owned by the subjects in a subtree.
(5) What is actually permitted when a role is exercised is a subset of the privileges of the subject who granted it.

Items are grouped into sets with common access control lists by the ACCESSRULE field in the ITEMS table; values in this field are the item identifiers of access control objects. Each access control object is interpreted by a permission function which it selects (Section 2.6) by fields in its catalog entry (Section 3.3) and must conform to rules specified by the author of this permission function.

For the most part, permission to an item does not depend on other items (apart from access control objects). However, if an item has a container, and an ITEMS flag is set, no permission is given unless it is also given on the container. A folder manager can use this to control access to documents within folders and to annotations on documents.

An otherwise unrelated resource manager could use a **DocSS** library instance as an external reference monitor. To enable **DocSS** for this, what needs to be added to what is already defined (Section 3.2) is a **PermissionSetGet: LibSession ItemID → PrivilegeList** operator which evaluates permissions.

4. DISCUSSION

DACM will be seen to be extremely flexible. The proposed standard implementation creates a decision logic more by the contents of tables than by fixed programs. Standard **DACM** can be extended by modules that realize different policy classes, with the possibility of using different policies for portions of a single library. **DACM** will also be seen to be efficient, both in new software needed and in execution overhead.

The client-server split, natural to library services, helps achieve protection. The programs and data that express and enforce library policies are in storage pools inaccessible to users except by way of library interface software, and thus themselves easily protected.

4.1 Flexibility, Extensibility, and Custom Reference Monitors

DACM treats subject relationships, subject-to-object relationships, and operation relationships differently. It provides as much support as possible for subject relationships, particularly aggregation, presuming that we know enough about generic delegation rules to represent them in fixed data structures and programs. Subject-to-object relationships are left to users, who are provided only means for tabulating them—the collection of access control lists A(s, e, a)—with aggregation over objects distributing administration to whomever is best qualified to govern each object set. Object aggregation is not new [Gladney 1975], but is achieved differently (each object points at its control object), a small change which confers critical advantages (e.g., conformance to OO structuring).

In contrast to what it does for subjects, **DACM** avoids built-in operation relationships. Such relationships built explicitly or implicitly into other access control systems have invariably been unwelcome in some applications; for instance, the frequent assumption that write privilege should imply read privilege is unsatisfactory for an audit trail. **DACM** implementations represent privilege sets by bitvectors easily managed in a resource manager or higher-level software. An enterprise which wants to enforce certain relationships can limit which privilege vector patterns are used.

Each stored object can bind both an access control object and a permission function and can choose its permission function either directly or via its access control object. This allows us to enable different rules for disjoint object sets. Thus a single object collection can mix access control policies, combining old methods with new ones and MAC with DAC, to meet the needs of different departments in a complex enterprise.

Access control sensitive to object contents is incomplete. Object and subject attributes stored as part of library catalogs can be handled efficiently. Constraints dependent on interrelationships of current object and subject with the environment E and arbitrary properties of other objects and subjects can be expressed as predicates stored in additional library catalog columns. Because the resource manager, reference monitor, and application programs are likely to be written by different people at differ-

ent times, how to represent and interpret such predicates is a new problem which can be handled by a **DACM** extension [Gladney 1994].

DACM makes its decisions based solely on library catalog records and primarily using only object identifiers. Content-sensitive access control needs a more powerful mechanism than this or any prior work provides. Needed controls are quite different for different enterprises, for different kinds of objects, and for different object representations (e.g., how a SmallTalk system might represent a tax return is different from any likely SGML representation). One recourse is to have the standard reference monitor implementation pass to an exit routine everything it receives, but this would be no more than an evasion for something we do not know how to handle.

4.2 Analysis of Delegation and Special Requirements

DACM proposes a quite specific delegation method; whether or not it is sufficiently flexible for its intended domain will not be settled without extensive peer review and consideration of many real-world situations. In the meantime, two tests make functional acceptability plausible. We first show **DACM** definitions of a few common administrative roles as specific permissions. We then describe how **DACM** can implement somebody else's statement of delegation needs.

DACM roles emulate job descriptions. Roles carry names like "secretary" because their instances are similar for all donor/donee pairs; here, "similar" means "have (approximately) the same privilege vector mask." Some examples of common meanings are shown here:

(1) *Library administrator:* permitted to administer subject definitions and to bypass checks dependent on object identifiers, in order to execute database backup and recovery.
(2) *Departmental administrator:* having the same privileges as a library administrator, but only for objects owned by a particular department, e.g., owned by subjects in a subtree of Figure 2.
(3) *Auditor:* permitted to inspect data owned by a subject subset, but prevented from changing any data in the inspection domain.
(4) *Secretary:* permitted to a subset of the privileges of some other single user.

The specific privileges of each role are likely to differ in different enterprises, but be uniform within each enterprise. Sharing privilege vector templates will be common, with graphic screen interfaces to define templates (by way of a checklist) and to manage arcs in the delegation graph. Long bitvectors are unacceptable for end-users; in our implementation [IBM 1994] most users see only role names.

Moffett [1988] provides an independent opinion of the delegation patterns needed (Table III). He also asks that access rules refer to organizational positions rather than to people. This is readily modeled by giving over a subject identifier whenever an office incumbent changes, with the

Table III. Compliance to Delegation Needs

Behavior Called for by Moffett	How to Model with DACM and DocSS
"give access control administration to security administrators"	Shortly after a document is created, mark its catalog record for MAC.
"give an access right if and only if the recipient occupies a position in the administrator's organizational domain and the resource falls in the administrator's resource domain"; for instance, "give access control administration of the marketing department of XYZ Co. to the security administrator in the data processing department."	Have the custodian give a proxy with security officer privileges to a subordinate of the data processing manager, naming the marketing manager node as the top of the tree defining this proxy's domain.
"allow particular users to particular resources"	Use **SubjectDefine** as defined in Section 2.4.
"giver (must have) authority over the resources to which he is giving access (and) authority to give access to the recipient"	This is the essence of Eqs. (6) and (12).
"designate a security administrator with no line authority, but with the authority to administer data on behalf of line management . . . without access to the data"	Create a user with the security officer attribute and permitted to read catalog entries, but not to retrieve objects or to make any updates apart from changing which access control object is bound to objects, limiting this security officer to a departmental domain.
"distinguish among *ownership* (total control), *grant-right* (permission to give access right to a third party), and *access-right* to perform a specific operation on a particular resource instance"	Section 2.6 defines what is meant by *ownership* under MAC and DAC rules; *access-right* is permission to an ordinary object; *grant-right* is permission to update an access control object.
"a manager should be able to specify the people under his or her control who form an organizational domain and grant access control authority over this domain to a security administrator"	Have a library administrator give the departmental manager a proxy, limited to the domain of his department and containing the **SubjectDefine** privilege.
"(a mandatory policy is that) only specified operations (transactions) can be invoked on authorized resources"	The essence of this is that subjects should be restricted to domains and be restricted to certain operations; both are supported.
Moffett implicitly makes use of the common practice that resource domains are realized by a hierarchical file directory which parallels the organizational structure.	Use the **DocSS** container relationship to create a structure isomorphic to the subject graph, or alternatively use the owner field, with some owners being groups.
"A library custodian should be able to control access as a function of document type. For example, (s)he should be able to limit the creation of a class of items to a particular department in an enterprise."	With an object-oriented resource manager, this can be achieved by controlling the class object; for other resources, it requires extensions [Gladney 1994].

simple flourish of a password-change ceremony. In this, we take Moffett literally—that there are *no* privileges granted to individuals in their own right, since they are allowed to use information only as enterprise agents. If, instead of this, subjects are to draw on resources both in their own rights and as enterprise agents, an extra level of indirection must be added to how a subject is bound to a library session.

Another needs analysis appears in an unpublished specification prepared by a Swiss bank in 1988; from this we extract and answer points not yet addressed in this section (see Table IV).

4.3 What Does DACM Cost?

In order of decreasing importance, we consider the cost in user time to exploit **DACM**, the overhead **DACM** adds to a storage subsystem without access control, and the development cost. (The cost of implementation is a vendor's issue only.)

Ordinary computer users mostly become aware of security machinery indirectly. They are commonly aware of access control because they have to understand it and to do something to take advantage of it. We must demonstrate that the **DACM** model can easily be taught to end-users, that managing controls requires a modest effort and can be delegated to the users who best know what protection each resource pool deserves, and that **DACM** imposes next to no new database administrator duties.

Run-time overhead and responsiveness are more important for clerical applications (such as handling tax returns and welfare paperwork) than for engineering and knowledge worker applications. Some proposed digital library applications require 10,000 object accesses/hour or more and may require 100,000 evaluations of Eq. (14) to achieve that; banking applications may require 10× this pace.

4.3.1 *Economy of Administration.* How easy end-users find **DACM** will, of course, depend on the quality of screen interfaces, including how well they display the subject graph. That is outside the scope of this article, which limits itself to the external effect of the underlying semantics and recommended implementation. We note the following:

—library administration can be as centralized or decentralized as is wanted;
—each user community and each individual user can choose as little or as much control differentiation as is wanted; access control objects are readily attached to objects and reflect subject groupings which parallel typical organizations;
—delegation, both hierarchical and proxy/role, mimics patterns familiar to users; access control lists are understood by many users and easily explained to others;
—access control data are managed similarly to other library data; application programmers need to understand only three new operations;

Table IV. Compliance to a Banker's Needs

Behavior Wanted by Swiss Bank	How to Model with DACM and DocSS
Document Ownership in the Library: "authorization . . . must dynamically react to . . . corporate organization changes. . . . (permit) an end-user to create, archive, and retrieve protocols of his or her branch, but not on a division level or for other branches . . ."	Changes are either SQL database updates to ITEM table fields or updates of access control objects. The kind of delegation called for can be represented by **DACM** structures.
Different Types of End-Users: "to let access to shared corporate documents (from outside), e.g., from foreign branches or customers, (might require) . . . both technical checks (e.g., network identity of request originator) and policy or legal checks (e.g., data import/export legislation) . . ."	The checks called for are neither addressed nor precluded by **DACM.** They are likely to require extensions of **DocSS** along anticipated lines [Gladney 1994].
Authorization Checking: "authorization of deputies is changed to that of the delegating end-user for the duration of the delegated task. This change of authorization is limited to the documents associated with the notification."	This requires some elaboration of the access control objects and permission function discussed and tight time-of-change to time-of-use constraints.
Delegating the Delegation: "if permitted by the original delegating end-user, further delegation to another end-user should be allowed. If so, the original delegating end-user should be informed about this if requested."	**DACM** provides this for hierarchical delegation, but precludes propagation of a proxy delegation because this feature is not known to be safe.
Authorization Checking Algorithm: ". . . deny document access if no match occurs between the document properties (or the properties of its group) associated with a qualifying document and the user properties and his or her workplace(s) properties . . ."	With added attributes in the SUBJECTS table (Section 3.3) and possibly other library catalog tables, this is possible with proposed **DACM** extensions [Gladney 1994].
Deputy Authorization for Private Drawer: "protect 'strictly personal' documents."	It is not clear that the strictest interpretation of this is either possible or, in fact, appropriate in a corporate database.
Exclude Lists: support "'exclude lists' . . . in which any document which matches . . . any of the properties of the exclude lists is subject to special treatment. (There exists a need for a very fast interrupt mechanism to *immediately overrule* all existing access authorizations granted.) . . . It is not feasible to mark all the involved documents, for two reasons: volumes (i.e., time to find out) in the library and—more important—more documents of this type may continue to flow into the corporation or are generated in the business processes. . . ."	This is neither provided nor precluded by **DACM** as defined.

Table IV—*continued*

Behavior Wanted by Swiss Bank	How to Model with DACM and DocSS
Differentiation in Staple Management Authority: "'staple' and 'unstaple' must be separately authorizable, i.e., they must be assignable to different end-users, because end-users with the authority to add (staple) commentary texts to documents may very well *not* be authorized to remove and delete (unstaple) commentary texts from documents."	Presuming that the resource manager defines staple and unstaple operations, they can be separately controlled. The **DocSS** LinkItem operation and LINKS table [Gladney 1993] could be used to implement staple support.
Limit Users by Data Type: limit the creation of a class of items to a particular department in an enterprise, or more generally, limit a user's access as a function of document attributes.	This is supported in the product implementation of **DocSS** [IBM 1994].
Redaction and Differential Control for Annotations: force placing an opaque mask over portions of a page when it is displayed, to provide a measure of confidentiality with regard to the underlying information. Control who can annotate an object.	These needs are not met by what is described in this article.
Changing Rules: conditions/policies for access control may change over time, but such changes must not lead to changes in the contents of stored objects or in their catalog entries (e.g., by reclassification).	This is similar to unanswered questions about versions in object-oriented databases: what to do about old instances when methods in the underlying class definition change. It has been addressed neither in this article nor anywhere we know.

—a library service can define any number of distinct privileges and can leave enforcing implied relationships to higher-level software; people readily understand boolean vectors, and commonly used patterns can be provided as named templates;

—changing an access control list is similar to any other document-editing task;

—administrators can arrange that most users get access controls attached automatically to the objects they create and can leave other administration to technicians.

—users can readily determine why any particular action was or will be denied or allowed; since delegation patterns are explicit (Figure 3), audits can be effected with simple utility programs acting on the security log L;

—in our **DocSS**-embedded implementation, no new computer operation or database administration tasks are imposed by **DACM.** Neither new

databases, nor new backup and recovery tasks, nor additional manuals are needed.

In summary, the **DACM** model mimics conventional office patterns so closely that external interfaces and jargon can readily be designed for computer novices; users can choose different behavior for different circumstances and can change specific rules whenever they want without unexpected side effects; and access management is so similar to other operations that it imposes few extra administrative chores.

4.3.2 *Economy of Execution.* Evaluation of the access decision ($B(s, r, c, o, S, O, E)$ in Section 2.5) is almost trivial. Equations (11) and (12) each express a short walk from a tree branch toward its root; the rest is merely some existence checks and boolean vector arithmetic. The cost of locating access control information is negligible because the root linkage is in each object's main catalog record. The potential performance degradation is the I/O overhead required to fetch access lists.

There follows a plausibility argument that a **DACM** implementation can make execution overhead small in all practical circumstances and imperceptible in many interesting applications. Its essence is that **DACM** data structures will always be small and heavily reused so that caching is feasible and effective. This will be the case because the size and number of access control lists will be determined by the human time needed to create them; **DACM** helps users by minimizing redundancy in these data.

—Each user's operation privileges are fetched as part of library session creation and held for the duration of the session; they are checked before accessing objects, saving time in the case of failed checks and costing next to nothing otherwise.

—At the option of the library custodian, any subject is permitted without further check any operation he or she has on objects he or she owns.

—Many objects will be protected by generic rules, e.g., in a public library, every object will have read privileges for most people and update privileges only for librarians. In such a library, fewer than 10 access control objects will describe the protection pattern for most of the collection.

—The worst case would be an application in which every user created his or her own access control lists. We believe that most users will create fewer than 10 lists with fewer than 10 subjects or groups in each, i.e., a large access control object collection will comprise $100 \times N_u$ table entries for N_u users, using less than 1MB for 1000 users.

Caching will be very effective, particularly caching of evaluated permissions [Gladney 1994] which has been implemented in an extension to the user description block in each library session [IBM 1994]. We find that retrieving the access control list for the first item in an access equivalence set contributes 10–15% to the library server delay to retrieve item catalog information. Within a library session, subsequent access to equivalence set members imposes no perceptible overhead.

4.3.3 *Economy of Implementation.* The **DocSS** framework can be used to create a different kind of library by changing catalog schema and a few subroutines. Implementing **DACM** this way costs us less than 7000 lines of C program and a few SQL table additions. Everything else needed already existed in an earlier version of the library.

This implementation runs on OS/MVS under the CICS transaction management subsystem and on OS/2. Porting to UNIX is under way; the cost of porting has been found to be imperceptible as **DACM** is less than 10% of the library catalog server and has no I/O interfaces apart from ANSI-standard SQL calls.

4.4 Prior Work and Future Possibilities

As far as we know, no prior practical implementations target the kind of information resources and scales some customers are requesting—10^8 objects and 10^4 users for a library with 10^4 object accesses per hour. (The largest current application still has under 10^6 objects; library size is limited by maximal relational DB size, but we do not know what will limit practical size.) Little fundamentally new work has affected products since the more popular access control schemes were devised over a decade ago. Those which are widely deployed on IBM machinery, or likely to be widely deployed—RACF, built-in SQL access control, AS/400 object-oriented controls, and the POSIX and DCE evolution of UNIX file controls—are implemented so that extraction for broader contexts would be impractical. What can be learned from these subsystems is not significantly extended by products in other environments. None of these tools has been prepared to bridge from commercial security practices to military ones.

It is nevertheless instructive to consider the semantics of several prior schemes relative to the problem at hand, not only to answer "Why not just extend scheme XYZ?", but also to show it plausible that **DACM** coverage could be broad.

4.4.1 *Access Control for File Systems.* RACF [Gladney et al. 1975] is the earliest commercially successful and surviving tool based on access control lists. The access control list for a protected object is in a special catalog and is found via the data set name; object grouping is accomplished by wildcards in the names which bind access control lists. This name orientation apparently has unfortunate consequences; the behavior under copying and moving objects can be unexpected by users. Specifically, we have been told by IBM customers that the generic data set profile scheme implemented by wildcards confuses users and has unexpected effects. For instance, suppose the generic profile ABC.DE*.TEXT was put into effect by one user and that another, without checking, puts into effect a profile named ABC.DEF*.TEXT and denying access to the first user; the first user is likely to be surprised and annoyed when jobs which previously ran are suddenly rejected as violating security. While **DACM** could emulate RACF, we deem it undesirable to carry forward its name orientation.

OS/400 file access control is different: controls are bound to objects rather than to object names. Each object and each subject have associated tokens used as tickets [Saltzer and Schroeder 1975]; the access decision is made by comparing tokens, looking for matches. (This rule is somewhat different in mandatory control systems, which require [Bell and Padulo 1975] that subjects' organizational domains contain objects' domains and that subjects' authorization levels exceed document sensitivity levels.) When an object is copied or moved, its tickets accompany it—a much more desirable behavior than RACF provides. We intend to take up such ticket-based control in the future.

POSIX and OSF-DCE access control directions call for [ISO 1991] modest extensions of UNIX file permission bits, to which they add control lists based on users and groups, without nesting of groups. Aggregation into access equivalence sets is limited to the aggregation into directories. They support neither user-to-user delegation nor fine-grained control of administrative privileges.

POSIX edicts only the traditional read/write/execute permissions, but permits an implementation any number of other control bits. The commentary in the draft standard discloses uneasy compromises to maintain compatibility with old systems, e.g., a difficult-to-understand purpose and semantics for ACL_MASK_OBJ entries [ISO 1991]. We find "the expectation that POSIX conforming systems will wish to extend the functionality defined in this standard to meet particular, specialized needs. For these reasons, flexibility in the POSIX_MAC requirements while still conforming . . . is an important objective" [ISO 1991]. This leaves enough freedom to permit vendors to make mutually incompatible extensions.

The obligatory part of POSIX is a subset of **DACM,** i.e., the contents and bindings of **DACM** access lists can be chosen to conform to a minimal implementation. Alternatively, any **DACM** data structure without proxy/role exploitation conforms to the more general POSIX definition.

4.4.2 *Access Control in Relational Databases.* SQL access control [Griffiths and Wade 1976] does not seem extensible to controlling documents and similar objects because it is deliberately constrained to conform to the relational model. Implementations are integrated into database management systems to an extent that reuse is practical only by superposing a new layer of software, more or less as **DACM** does.

One aspect of SQL access control needing discussion is how users propagate authority. The SQL rule is much simpler than the **DACM** rule: the owner of an object can grant to other subjects not only the basic relational operations but also the privilege of granting any received permission to further subjects. However, access revocation seems to present problems [Fagin 1978], and the delegation patterns described in Section 4.2 are not modeled. It has been carefully considered [Gagliardi et al. 1989] relative to GRANT with GRANT support. One objective has been to represent the delegation graph relationally with only additive propagation (no denial of a privilege). Allowing only the sole owner of an object to grant

privilege is easy to control but too restrictive. The alternative of allowing each grantee to propagate privileges received causes owners to lose control so that they cannot be held responsible for enterprise resources. To solve that, Leiss [1983] proposed to bound privilege propagation horizontally and vertically.

Fagin [1978] proposed that revocation of a grant should leave the DB in a state equivalent to the one in which the grant was never made; this requires a grant timestamp record because a revoked subject might have received the privilege from different sources and granted it further on. If he or she granted the privilege before receiving it from any other source, then the grantee should be revoked, and only in this case. This leads to situations in which the revoker needs to be fully aware of side effects, depending on definitions entered by other users. A variation [Gagliardi et al. 1989] lets a revokee's grant persist if he or she received the privilege from at least one other source. This risks grant cycles in which the source of privilege becomes obscure. **DACM** can thus not span all delegation policies and avoids some of the pitfalls by narrower rules.

4.4.3 *Access Control in Object-Oriented Databases.* "Object" in the current article is a primitive concept, being a set of bitstrings associated with a single identifier. Sophisticated object models are represented by several database implementations, but recent reports [Butterworth et al. 1991] hardly mention access control.

DACM structure follows object-oriented practice in that access control information is bound to an object by having the object point to an access control object and by having the access control object point to its interpretation method. We believe this makes **DACM** applicable along the following lines:

—one or more classes would define access control information management; an instance of such a class would be what we have called an access control object;

—any object needing protection would bind an access control object as an attribute, by calling a polymorphic ObjectBindAccessInformation operator during its creation; such a binding would persist until explicitly changed;

—the object would deny sensitive actions until a subject and pertinent environmental objects were bound with an ObjectGetCurrentUser operation invoked in an ObjectOpen method; this binding would last only for the current user-database connection;

—an access check would be a message passing the current subject and an integer to the associated access control object; such messages would be issued by the protected object's sensitive methods, i.e., implemented conventionally as part of the class definition of the protected object;

—the integer mentioned would select a PrivilegeList position (Sections 2.1 and 3.3), i.e., the meaning of PrivilegeList slots would depend on the applicable object class.

The author of every class would define its controls, with the same flexibility and responsibility for access control as he or she has for any other part of the class definition. Whether or not such an approach could be fleshed out to a complete and coherent solution is yet to be examined.

Rabitti [Rabitti et al. 1991] has started to address object-oriented authorization by proposing a scheme (called **RACM** below) which propagates implicit authorizations along class- and instance-nesting hierarchies. The objective is to make it "unnecessary to store all authorizations explicitly; the authorization mechanism can compute authorizations from a minimal set of explicitly stored authorizations . . ." Propagation forces distinctions between strong and weak authorizations (the weak ones can be overridden) and between unstated and negative authorizations—distinctions unneeded in other access control schemes. Because it exploits class and object hierarchies, **RACM** is currently limited to object-oriented systems; in contrast, **DACM** does not assume interobject relationships.

Comparing these two approaches raises interesting questions. For instance, **RACM** Setof-X-Instances objects hold authorizations for all X instances except those holding overriding specifications; however for a class like *memorandum* it is unlikely that a majority of instances will share a single authorization list, i.e., it is doubtful that **RACM** offers space economy over **DACM**. Rabitti makes clear how class authorizations might relate to instance authorizations for engineering databases, in which the individual users who define classes commonly work in some proximity to those who exploit the classes, but does not touch on what seems more likely for massive clerical applications, viz., that class authors have no direct communication with end-users, and end-users manipulate only instances, not class objects.

Possibly **RACM** and **DACM** ideas can be combined for OO databases, but doing so is beyond the scope of the current article. However, it seems plausible that **DACM** can be used for object-oriented databases with implementation following the usual style of object-oriented programming.

4.4.4 *Other Kinds of Delegation and Composite Subjects.* The delegation scheme described does not show how to control a process working for some user other than the one who started it and accessing resources authorized to the former user, e.g., a server commanded to print named contents of a database. This problem was not addressed because it did not appear among the topics raised by our users and because the **DocSS** design allows a secure channel (based on cryptographic protocols [Janson 1992] for third-party authentication and digital signature certification) from each library client to the library catalog server. (In our digital-library work, such a secure channel has neither been cost-justified nor implemented, so in fact end-users and data custodians must assume that the library client code has not been tampered with; no one has raised this as a practical concern for the current applications and user sets.)

Delegation in **DACM** is from user to user and relatively static. Abadi [Abadi et al. 1993] has recently analyzed delegation formally, concentrating

on user-to-network-node delegation and node-to-node delegation, pointing out that the meaning of *delegation* results from the combination of the contents of access control lists and the behavior of delegators and delegates and concluding that the meaning of *delegation* is a matter of policy or convention. In general, delegation can be expressed as subject composition and becomes computationally complex. In this context, delegation in what we term the standard **DACM** model is a particular policy which is interesting because it captures common business needs efficiently both from a user perspective and for computation.

Starting with Karger [1985], the topic of delegation has received careful attention [Erdos and Pato 1993]. Although an authorization management system pretending completeness must embody this, we can defer doing so until we extend beyond user-to-user delegation with secure channels from users to trusted servers.

4.5 What Makes the DACM Design Economical and Flexible?

DACM combines independent simple measures, applying four well-known design principles: let data structures follow the users' external world model; put rules into tables rather than into procedures; use recursion, with self-reference for termination; add a level of indirection whenever information is too tightly bound.

For a distributed-resource manager it defines the database and procedure kernel for the authorization component of a protection system. From the most specialized and functionally complete level to the most general framework, it consists of the following:

—embedded **DACM** to be part of and protect a digital library service;
—standard **DACM**, with tables and a permission function which satisfy a wide variety of commercial and some military requirements; the basic version depends only on subject and object identifiers; an extended version will be sensitive to subject, object, and environmental attributes;
—structure to replace the standard reference monitor with custom rules which can ignore, use, or extend the standard subject and object tables; distinct reference monitors can coexist, even within and applied to a single data collection;
—**DocSS** infrastructure to provide all the linkages needed to employ the former components in remote authorization servers.

To understand the essential structure, scaling properties, and extensibility of standard **DACM**, the reader is encouraged to focus on the object relationships depicted in Figure 3 and the partitioning that allows distributed administration without loss of accountability. Economy comes from the following elements:

—permissions are unions of static privileges and role privileges; operation privileges are granted down a subject tree, with any grant being a subset of the grantor's privileges at the time of grant; proxy privileges may be

granted across the subject tree, with such a grant being a subset of the grantor's privileges at the time when the role is claimed;

—the delegation tree is rooted in a custodian, who has all privileges; there is no distinction between users and groups;

—administrative operations, such as subject creation, are ordinary library operations; access control information is controlled and administered as other information;

—a cumulative effect of these three measures is identical semantics for all kinds of global administrators, departmental administrators, and individual delegations;

—grouping of objects, subjects, and operations is by table entries rather than by code or by isomorphism with operating system structure like UNIX file directories;

—fields in each object select its access control object and its permission function; the syntax and interpretation of access control information are fixed by the author of the handling permission function; and

—privileges are represented as bitvectors, with interpretation by the service routines of the resource manager for the object(s) protected.

5. CONCLUSIONS

The users of almost every computer network want access to more information. Broad access to huge collections must be tempered by prudent controls. The interesting information services have diverse applications and users ranging from clerks with routine tasks to professional staff with wide-ranging needs; authorization services must have unprecedented flexibility. Asset administration must be distributed, even for individual collections, as close to automatic as possible and comprehensible and convenient for ordinary users.

We have described a scheme for digital libraries. **DACM** complements other essential protection tools and comes closer to meeting known requirements (Section 1.2) than prior work; we have identified shortfalls explicitly and are optimistic that upward-compatible extensions will solve them. The other tools needed—communication security, authentication of process identities, and basic storage and operating system measures—will come from a combination of well-known and emerging techniques. **DACM** implementations can readily comply with all pertinent standards and can emulate prior methods. The scheme for privilege delegation is novel, and the ability to mix different control models within a single protected resource is unique.

DACM scales well, allowing as little or as much control differentiation as is wanted, up to the ridiculous extreme of indicating which of several hundred privileges each of many thousand users has on each of many million objects. Librarians with simple requirements can specify them quickly, possibly without consulting with end-users. Users can mix fine differentiation for critical resources with uniform controls for most of a

collection. Lightly protected resources do not suffer any performance or administrative burden because sensitive resources happen to exist. At the low extreme, the sole user and custodian of a private library needs to specify nothing and will experience no perceptible **DACM** overhead.

The abstract problem solved is finding symmetries in a permission function $B(s, r, c, o, \mathrm{S}, \mathrm{O}, \mathrm{E})$ to allow its concise expression without losing any desired distinctions. We teach how to represent important differences among subjects, objects, operations, and relationships for economical description, storage, and calculation of $B(s, r, c, o, \mathrm{S}, \mathrm{O}, \mathrm{E})$.

A **DACM** subset is implemented in the library portion of a distributed image/document service [IBM 1994]. As no affordable experiment can test and demonstrate everything claimed, we must wait for user communities and document collections to grow sufficiently to validate complete achievement of our scale, usability, and performance objectives.

Massive data collections are not only extremely valuable assets, but also at risk from deliberate and accidental misuses; they will present tempting targets. We have no wish to exaggerate the importance of computer crime. However, whatever one estimates the risks to be, it seems prudent and socially positive to minimize temptation by raising barriers, provided that these do not impede legitimate access and are not a nuisance to administer. We believe that **DACM** improves safety without being obtrusive.

ACKNOWLEDGMENTS

The **DACM** design would not have been possible without correspondence and conversations with Charlie Cree, Rene Furegati, Paul Hudecek, Stan Kurzban, Anne Lescher, Marcel Schlatter, Curt Symes, Adrian Walker, and Eldon Worley. That these communications occurred at widely separated intervals over six years has not made them less effective.

The implementation embedded within **DocSS** was started by Tom Burket, John DiClemente, and Mike Vitale with the support and encouragement of Ken Fisher and Karl Schubert. They contributed to the practical design, together with Arjun Mendhiratta and Kevin McBride, who also built the latest version and provided information about it for this article.

REFERENCES

ABADI, M., BURROWS, M., LAMPSON, B., AND PLOTKIN, G. 1993. A calculus for access control in distributed systems. *ACM Trans. Program. Lang. Syst. 15,* 4, 706–734.

BELL, D. E. AND LAPADULA, L. J. 1975. Secure computer system: Unified exposition and multics interpretation. MIRTRE Corp., Bedford, Mass. Also available as NTIS AD-A023588. National Technical Information Service, Springfield, Va. (1976).

BUTTERWORTH, P., OTIS, A., AND STEIN, J. 1991. The GemStone object database management system. *Commun. ACM 34,* 10 (Oct.), 50–63.

CHOKHANI, S. 1992. Trusted products evaluation. *Commun. ACM 35,* 7 (July), 64–76.

CLARK, D. C. AND WILSON, D. R. 1987. A comparison of commercial and military security policies. In *Proceedings of the IEEE Security and Privacy Symposium.* IEEE, New York.

DEUX, O. 1991. The O_2 system. *Commun. ACM 34,* 10 (Oct.), 34–49.

DEPARTMENT OF DEFENSE. 1985. Trusted computer system evaluation criteria. DOD 5200.28 STD, National Computer Security Center, U.S. Department of Defense, Washington, D.C.

ERDOS, M. E. AND PATO, J. N. 1993. Extending the OSF DCE authorization system to support practical delegation. In *Proceedings of the PSRD Workshop on Network and Distributed System Security.* 93–100.

FAGIN, R. 1978. On an authorization mechanism. *ACM Trans. Database Sys. 3,* 3, 310–375.

FOX, E. A., AKSCYN, R. M., FURUTA, R. K., AND LEGGETT, J. J., Eds. 1994. *Proceedings of Digital Libraries '94.* Springer-Verlag, Berlin.

FOX, E. A., AKSCYN, R. M., FURUTA, R. K., AND LEGGETT, J. J. 1995. Digital libraries. *Commun. ACM 38,* 4 (Apr.), 22–28.

GAGLIARDI, R., LAPIS, G., AND LINDSAY, B. G. 1989. A flexible and efficient database authorization facility. IBM Res. Rep. RJ 6826, IBM, San Jose, Calif.

GLADNEY, H. M. 1978. Administrative control of computing service. *IBM Syst. J.* 17, 151.

GLADNEY, H. M. 1993. A storage subsystem for image and records management. *IBM Syst. J. 32,* 3, 512–540.

GLADNEY, H. M. 1994. Condition tests in data server access control. IBM Res. Rep. RJ 9244, IBM, San Jose, Calif.

GLADNEY, H. M., WORLEY, E. L., AND MYERS, J. J. 1975. An access control mechanism for computer resources. *IBM Syst. J. 14,* 212.

GRAY, J. AND REUTER, A. 1993. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, San Mateo, Calif.

GRIFFITHS, P. P. AND WADE, B. 1976. An authorization mechanism for a relational database system. *ACM Trans. Database Syst. 1,* 3, 242–255.

HOFFMAN, L. J. AND MORAN, L. M. 1986. *Societal Vulnerability to Computer System Failures.* Vol. 5, *Computers and Security.* Elsevier Science, Amsterdam, 211–217.

IBM. 1985. Resource Access Control Facility (RACF) general information manual. Systems Ref. Lib. GC28-0722, IBM, San Jose, Calif.

IBM. 1991. Image and Records Management (IRM) general information guide. IBM Systems Ref. Lib. GC22-0027, IBM, San Jose, Calif.

IBM. 1994. IBM ImagePlus VisualInfo general information and planning guide. IBM Systems Ref. Lib. GK2T-1709, IBM, San Jose, Calif.

ISO. 1991. Information technology—Portable Operating System Interface (POSIX)—security interface. ISO/IEC JTC 1/SC22/WG15 N046R1 P1003.6 Draft 12, International Standards Organization, Geneva, Switzerland.

ISO. 1992. Information retrieval, transfer and management for OSI: Access control framework. ISO/IEC JTC 1/SC 21/WG 1 N6947 Second CD 10181-3. International Standards Organization, Geneva, Switzerland.

JANSON, P. 1992. Security and management services in open networks and distributed systems. IBM Res. Rep. RZ 2274, IBM, San Jose, Calif.

KARGER, P. A. 1985. Authentication and discretionary access control in computer networks. *Comput. Networks ISDN Syst. 10,* 1, 27–37.

KOHL, J. T. 1991. The evolution of the Kerberos authentication service. In *Proceedings of the EurOpen Conference, Unix Open Systems in Perspective.* IEEE Computer Society Press, Los Alamitos, Calif., 295–313.

KUMAR, R. 1991. OSF's distributed computing environment. *IBM AIXpert* 2, 22–29.

LAMB, C., LANDIS, G., ORENSTIEN, J., AND WEINREB, D. 1991. The ObjectStore database system. *Commun. ACM 34,* 10 (Oct.), 50–63.

LAMPSON, B., ABADI, M., BURROWS, M., AND WOBBER, W. E. 1991. Authentication in distributed systems: Theory and practice. *Oper. Syst. Rev. 25,* 5, 165–182.

LAMPSON, B., ABADI, M., BURROWS, M., AND WOBBER, W. E. 1992. Authentication in distributed systems: Theory and practice. *ACM Trans. Comput. Syst. 10,* 4 (Nov.), 265–308.

LEBOWITZ, G. 1992. An overview of the OSF DCE distributed file system. *IBM AIXpert* 3, (Feb.), 55–64.

LEE, T. M. P. 1988. Using mandatory integrity to enforce "commercial" security. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy.* IEEE, New York, 140–146.

LEISS, E. 1983. Authorization systems with grantor-controlled propagation of privileges. In *Proceedings of Spring COMPCON '83.* IEEE Computer Society Press, Los Alamitos, Calif.

LINN, L. 1990. Practical authentication for distributed computing. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*. IEEE, New York, 31–40.

LOHMAN, G. M., LINDSAY, B., PIRAHESH, H., AND SCHIEFER, K. B. 1991. Extensions to Starburst: Objects, types, functions, and rules. *Commun. ACM 34,* 10 (Oct.), 94–109.

MOFFETT, J. D. AND SLOMAN, M. S. 1988. The source of authority for commercial access control. *Computer 21*, 2 (Feb.), 59–69.

OSF. 1991. *Distributed Computer Environment (DCE) Version 1.0: Application Development Reference*. Open Software Foundation, Cambridge, Mass.

RABITTI, F., BERTINO, B., KIM, K., AND WOELK, D. 1991. A model of authorization for next-generation database systems. *ACM Trans. Database Syst. 16,* 1, 88–131.

RICHARDSON, J., SCHWARZ, P., AND CABRERA, L.-F. 1992. CACL: Efficient fine-grained protection for objects. IBM Res. Rep. RJ 8894, IBM, San Jose, Calif.

SALTZER, J. H. AND SCHROEDER, M. D. 1975. The protection of information in computer systems. *Proc. IEEE 63,* 9, 1278–1308.

SILBERSCHAT, A., STONEBRAKER, M., AND ULLMAN, M., Eds. 1991. Database systems: Achievements and opportunities. *Commun. ACM 34,* 10 (Oct.), 110–120.

STONEBRAKER, M. AND KEMNITZ, G. 1991. The PostGres next generation database management system. *Commun. ACM 34,* 10 (Oct.), 78–92.

VARADHARAJAN, V., ALLEN, P., AND BLACK, S. 1991. An analysis of the proxy problem in distributed systems. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*. IEEE, New York, 255–275.

Received March 1993; revised January 1994; accepted December 1994