Authorization in Data Management Systems

Darrell Raymond
Alternative Output Inc.
Waterloo, Ontario, Canada

Abstract Computer systems for data management are increasingly concerned with the authorization problem: the problem of creating and managing the matrix of users, objects, and allowable operations. This paper discusses the authorization problem and explains why the problem is likely to become more complex in the future. We then present the use of database normalization theory to formalize and address the authorization problem. A variety of issues related to authorization that arise in practical data management deployments are discussed. Finally, recommendations are made for further research on authorization.

Introduction

Engineering and manufacturing companies need to manage drawings, documents, bills of materials, engineering change orders, and other data about their products. In the past this information was managed as paper. The trend today, however, is to store data electronically, using data management software from commercial vendors.[1] Data management systems are application suits built on relational databases that provide the following capabilities:

- Storage of large numbers of documents in an electronic 'vault'
- Change management for documents, typically through the use of check-in/check-out and revision control
- Storage and change management of product data and product structure (component-assembly or 'bill of material' relationships)
- Storage and management of relationships between product data and documents
- Definition and execution of structured processes or *workflow*

A data management system of this kind can provide the following benefits to an organization:

- Timely access to documents and drawings
- Better configuration control
- Reduced cycle time for engineering change
- Reduced need for data management personnel
- Reduced time spent in data management

In general, an engineering data management system is an accelerator—it speeds the delivery of information and increases the volume of information that can be transferred in a given unit of time. This advantage has a downside, however—if the security of the system is not sufficient, then the loss of information to unauthorized parties can also be greatly accelerated. The security requirements of an engineering data management system are significant when the system contains a large fraction of a company's intellectual property.

The authorization problem

Security requirements include the following:

- Authentication
- Authorization
- Data integrity
- Data privacy
- Reliable service

[1] Vendors of data management software include companies such as Agile, Auto-trol, Enovia, FileNet, Documentum, SDRC, Unigraphics Solutions, MatrixOne, Parametric Technologies, and Eigner + Partner. Their products fall into categories such as electronic document management (EDM), product data management (PDM), configuration management (CM and CM II), collaborative product commerce (CPC), and knowledge management (KM).

An engineering data management system's needs for authentication, integrity, privacy, and reliability are not dramatically different from typical IT systems. The requirement for authorization, however, far exceeds the typical case. This is because an engineering data management system tends to have a more sophisticated notion of people, operations, and data. The authorization subcomponent of an engineering data management system is therefore an important functional element.

We shall define an *authorization specification* as a n-dimensional matrix of elements that are involved in authorization. A simple authorization specification would be three-dimensional, and deal with people, objects, and operations: if the value of *(x, y, z)* is 1, then person *x* can apply operation *z* to object *y*. The *authorization problem* is the problem of populating the authorization matrix so that we have reasonable guarantees that the matrix is correctly and completely populated, and so that the matrix can be maintained in a reasonably efficient manner.

In a typical engineering or manufacturing company, this matrix is potentially very large. The number of objects *y* can easily be in the millions:

- A typical manufacturing firm maintains data on at least 10^5 parts and 10^6 component-assembly relationships
- Each part may have several associated documents (specification, geometric drawing, parts list, schematic, gerber file, application drawing, etc.)
- Parts typically exist in multiple revisions, each of which may have a drawing and usually one or more engineering change orders
- Some companies also maintain information on serialized parts—thus, *y* is potentially as large as the total number of units ever produced

The number of operations *z* is larger than the standard read-write-execute of common file systems. A typical data management system may support as many as 30 different operations, including:

- Create—generate a new object
- Read—read the attributes of an object
- View—read the files associated with an object
- Modify—edit the attributes of an object
- Revise—generate a revision of an object
- Print—allow the files associated with the object to be printed
- Check-out (check-in)—export (import) files associated with the object
- Delete—delete an object
- Lock (unlock)—disallow (allow) file checkout
- Promote (demote)—move the object to the next (previous) stage in its lifecycle
- Grant—allow other users privileges
- Set privileges—change any of the authorizations for an object
- Link (unlink)—create (delete) a certain kind of relationship to another object
- Execute—execute a program or method associated with an object

Although *x*, the number of users of a data management system is moderate, each of the users can belong to multiple groups and can participate in multiple roles. A 'role' or a 'group' is essentially a set of persons that can be treated as identical for some business purpose. Many data management systems treat a group or a role as a substitute for a person. In other words, *x* can be set-valued, with sets chosen from the set of persons.[2]

Workflow adds another dimension to the authorization matrix, that of *state*. In workflow systems, it is typical to define processes where (for example) person *x* can apply operation *z* to object *y* only when that object is in state α. States typically include such concepts as 'draft', 'in work', 'review', 'release', 'superseded', and 'obsolete', but others are possible. It is not unusual to have different sets of states for different kinds of objects; thus, parts and drawings will be subject to formal release control and go through several states, but documentation may be handled more informally and have only one or two states.

In addition to adding another dimension, the notion of state also increases *z*, because the system requires operations to change the state of an object, and these operations are subject to authorization control.

[2] Some data management systems allow full boolean combinations of person and role sets.

Given the foregoing issues, it is clear that in a typical situation the authorization matrix may easily exceed 10^{12} entries. This makes the authorization problem quite significant.

Stresses on authorization

Until recently, companies have made the authorization problem tractable through the use of simple rules that dramatically reduce the size of the authorization matrix:

- All employees have read access to everything
- Non-employees have no access to everything
- All other operations are permitted only to a small group that does all data entry

Under these rules, y is reduced to 1 (because all objects are treated the same); z is reduced to 2 ('read' and all other operations); a is reduced to 1 (there is no state dependency) and x is reduced to 3 (internal, public, and data entry).

This simple scenario will shortly become untenable. Companies are focused on the 'new economy', joint ventures, downsizing, e-commerce, Web portals, and other trends whose effect is to increase the size of the authorization matrix.

The rule 'all employees read access to everything' is increasingly less useful as the notion of 'the company' becomes more complex. Manufacturing companies are becoming less vertically integrated and are moving towards outsourcing, collaboration, core competency, and joint venture as their principal mode of business. This change increases the complexity of the authorization problem.

Contracting out (outsourcing) involves subdividing a project among a number of contractors. Contractors should be shown only the data they need to do their job, but not other data. It is important to limit contractor access to information for many reasons. One is that contractors may also work for competitors, and hence, could accidentally or intentionally transmit information the competitor should not see. Another reason is that access to unrelated information enables the contractor to see other projects and needs, and hence to bid more effectively on new work—thus costing more than they would otherwise. Contractors need more than just read access to data, because they are often actively generating new engineering and manufacturing information.[3] Workflow also becomes more important because contractors may not be physically on-site, and may even be in different time zones; in these situations workflow is useful to ensure synchronization of tasks.

Collaborative ventures are joint projects undertaken with other firms in a peer relationship (rather than a subcontracting relationship). Collaborators may include current and future competitors. While collaborators should have free access to data related to the project on which they collaborate[4], it is usually important to ensure that competitors do not gain access to other projects. Collaborators, like contractors, typically need the ability to update information, not just to read it.

Joint ventures are semi-autonomous organizations that are formed by otherwise competitive firms. A joint venture may have its own IT department with its own notions about how to do authentication and authorization; for example, it may see less need for authorization that do its parent companies, because the joint venture's information is wholly open to either parent.

E-commerce brings with it an expansion of the authorization problem. Customers are beginning to expect online access to information about products both past and present; in some cases, customers want and companies to maintain information on as-installed and as-maintained configurations. Customers of course should only have access to their own configurations, and not those of other customers. Customers themselves are complex entities; engineering personnel at a given customer site should have different kinds of access than accounts payable personnel, for example. As an additional complexity, the rapid pace of mergers, consolidations, and divestitures means that what counts as a single customer changes over time.

All these factors increase the size and complexity of x and y as well as increase their rate of change. It is not unusual for a merger, a joint venture, a collaboration, or some other significant corporate event to occur during the lifetime of a data management system.

---

[3] In some cases, even the management of engineering data itself may be outsourced.

[4] Although almost certainly not all of it; for example, a company will keep its margin and cost information hidden from its collaborators.

Many companies are interested in workflow, but not all have deployed it. Workflow will soon not be a matter of choice, but a matter of survival. Configuration management departments that once had 40 people to manage engineering change are now reduced to 4. Shuffling paper change forms is no longer a viable option—particularly when so much of the change is happening 'outside' the firm. This means significant increases to z and α.

The demands of the new economic climate simultaneously increase the importance and the difficulty of the authorization problem.

Categorization and roles

One method proposed to ameliorate the authorization problem is the use of categories and roles. A *category* is a set of documents or objects; a *role* is a set of people who are authorized to act in a certain way on a given category. Generally we allow objects to belong to more than one category, and people to belong to more than one role. The authorization matrix might then be expressed as *(X, Y, z)*, where X is a set of roles and Y is a set of categories. The use of roles and categories can reduce the size of the authorization matrix if it is the case that $|X| << |x|$ and $|Y| << |y|$. Role-based authorization [1] is founded on this assumption.

Role-based authorization attempts to avoid redundancy in the authorization matrix. Instead of specifying as independent constraints that individuals $x_1, x_2, x_3,\ldots x_n$ can each perform some operation on documents $d_1, d_2, d_3,\ldots d_m$, we simply define two sets X and Y, and establish a single authorization constraint between them. This redundancy reduction also simplifies updates; in the example given above, adding a single user to the set X automatically gives the user access to all the documents Y. The set-based constraint provides a multiplicative benefit under update. However, there is a corresponding downside: any error in defining the sets also results in a multiplicative defect. If person x_k is mistakenly assigned to a role, then that person gains much greater access than would be the case if a single error occurred in a pairwise constraint system. Thus, role-based authorization can reduce the cost of populating the authorization matrix, but increases the requirement for assuring the matrix. Role-based authorization also increases z, because it introduces the need for operations to create and maintain roles and categories.

Designing a matrix

The size of the authorization matrix and the number of dimensions it contains make for a complex design problem. The standard literature provides no formal method for representing and solving such design problems. We propose the use of database normalization techniques to construct a proper authorization matrix.

A *database design* is a set of relations and associated constraints, such as keys and dependencies. A good database design is a set of relations that controls redundancy and ensures that updates will not result in inconsistent data. *Data normalization* is a technique for arriving at a good database design by constructing relations according to known data dependencies. There is a considerable body of work on normalization, including definitions of normal forms, algorithms for achieving normal forms, and complexity results [4][5]. A full exploration of data normalization is beyond the scope of this paper, but the following gives some idea of the flavor of the approach.

In order to apply database design techniques to the authorization problem, we must recast the problem in terms of attributes, dependencies, and relations. We can then apply known normalization techniques to arrive at a normalized schema that captures the data elements in as concise a manner as possible, and minimizes the update problems that might arise. We next shall see an informal description of how this is done.

Attributes

The previous statement of the authorization problem—a matrix of persons, objects, and operations—does not capture the full generality of the problem. A more complete specification, which assumes a role-based system, takes into account the following factors:

- People take on one or more Roles in their work activities, and Object access can be based on a Person's Role. For example, an Author has permission to write while a Reviewer may only have read permission.
- People belong to Groups, and Object access can be based on Group membership. For example, an Employee may be able to read some Objects a Contractor may not.
- An Object may comprise multiple Files, and these may have different access rules. For

example, an editable file may be edited, while an archive format file may not.

- An Object exists in different Revisions, and different Operations can be applied to these Revisions. For example, an engineer may only be able to read the current Revision, but can both read and write a new (unreleased) Revision.
- An Object can be in one of several States, and the State may determine the Operations that can apply to it. For example, a released Object may be read-only, but an in-process Object can be written.
- Objects belong to one or more Categories. Access rules may be defined in terms of Categories instead of individual objects.
- Operations belong to one or more sets we shall call Capabilities. Access rules may be defined in terms of Capabilities rather than individual operations.
- Work is organized as a set of Transactions. Each Transaction applies one or more Operations to one or more Objects.

The following attributes will be considered in our design:

- `Person`
- `Operation`
- `Object`
- `Role`
- `Group`
- `Revision`
- `File`
- `State`
- `Category`
- `Capability`
- `Transaction`

Functional dependencies

Once we have identified attributes, we next look at the dependencies between them. A *functional dependency* $X \rightarrow Y$ exists between two sets of attributes X and Y if for every value of X there exists only one value of Y. Identifying the functional dependencies is a key step in constructing a good database design, because the dependencies capture the sets of attributes which must be updated simultaneously if consistency is to be preserved.

The dependencies that apply to the authorization problem depend to a large extent on the business rules that one wants to enforce. Thus, we cannot derive a set of dependencies that apply to all situations. The following dependencies may be regarded as typical.

An Object of a given Revision can be in only one State. Different Revisions of an Object may be in different States.

FD1 `Object, Revision → State`

An Object of a given Revision can be modified by only one Transaction at a time[5]. A Transaction can modify more than one Object at a time. A given Object of a given Revision need not be modified by any Transaction.

FD2 `Object, Revision → Transaction`

A Person can only take one Role per Transaction[6]. A Person may be in different Roles in different Transactions.

FD3 `Person, Transaction → Role`

A given Person can apply a set of Operations to an Object of a given Revision.

FD4 `Person, Object, Revision → Capability`

Often it is desirable to define access rules in terms of sets of Persons (that is Groups or Roles), sets of Objects (that is, Categories), and sets of Operations (that is Capabilities) rather than in terms of individual mem-

[5] Note that in practice, Objects may be associated with Transactions without being modified by them (for example, Objects may be associated for reference purposes). In such cases **FD2** should be `Object, Revision -> Modifying_Transaction`. For simplification, we restrict our notion of Transaction to include only Objects that are to be modified.

[6] This restriction stops a Person from effecting a change and then approving it.

bers of these sets[7]. Where this is the case, we have one or both of the following additional dependencies:

All Objects within a given Category in a given State permit a specific set of Operations to a Specific Group or Role. Different Groups or Roles may have the same set of Operations on a given Category in a given State.

FD5	`Category, State, Group → Capability`
FD6	`Category, State, Role → Capability`

Given a set of functional dependencies, we can derive other implied dependencies through the use of closure axioms[8]. Among the non-trivial dependencies that we derive are the following:

Union of FD1 and FD2:

FD7	`Object, Revision → State, Transaction`

Augmentation of FD7:

FD8	`Person, Object, Revision → State, Transaction`

Union of FD8 and FD4:

FD9	`Person, Object, Revision → State, Transaction, Capability`

Reflexivity of F8 with `Person`

FD10	`Person, Object, Revision → State, Transaction, Person`

Union of FD3 with `State`:

FD11	`Person, Transaction, State → Role`

Transitivity of FD10 with FD11, augmentation with FD8, union with FD4;

FD12	`Person, Object, Revision → State, Transaction, Role, Capability`

Pseudotransitivity of FD1 and FD5 (eliminating `State`):

FD13	`Object, Revision, Category, Group → Capability`

Pseudotransitivity of FD1 and FD6 (eliminating `State`):

FD14	`Object, Revision, Category, Role → Capability`

Pseudotransitivity of FD2 and FD3 (eliminating `Transaction`):

FD15	`Object, Revision, Person → Role`

Generation of a good database design

Given attributes and dependencies, we now attempt to develop a set of relations that will have good update and redundancy properties. Consider first of all the design that would result if we simply put all attributes in one table (attribute names shortened here for presentation), thus directly representing what we have called the authorization matrix:

Pn	Op	Ob	Ro	Gr	Re	Fi	St	Ct	Cp	Tr

This would be a valid relation, but suffers from the following problems:

1. We do not know which set of attributes to use as the key (that is, the unique identifier for a row)
2. We prefer avoiding null values in the table[9], but this means we must provide information on every attribute every time we add a row to the table. We cannot, for example, easily add a row to show that Object O belongs to Category C, because to add a row we must provide information for every attribute.
3. There is a lot of redundancy in the table. The fact that Object O belongs to Category C is represented for each row that appears. This redundancy costs space and presents the possibility that an update can change the correspondence between O and C in some of the rows without changing others, thus leading to inconsistency.

Database designers will recognize the (unnormalized) authorization matrix as an instance of the *universal relation*—that is, a single table with as many columns as there are attributes. The universal relation contains

[7] Note that in general, a Person may belong to more than one Group, an Operation may belong to more than one Capability, and an Object may belong to more than one Category.

[8] Defined in the Appendix.

[9] Writing correct queries in the presence of null values is difficult.

a significant amount of data duplication and some update problems. The goal of database design is to normalize this table into multiple tables that eliminate the redundancy and avoid the update problems.

While the development of a normalized schema is beyond the scope of this paper, but we will show one better-normalized schema and discuss its characteristic. The database schema shown below consists of a number of relations **Ri**, each consisting of a set of attributes. The underlined attributes are the primary key of each relation.

R1	Object, Category
R2	Person, Group
R3	Operation, Capability
R4	Object, File

The first four relations allow us to express the containment of Persons in Groups, Operations in Capabilities, Objects in Categories, and Files in Objects. The key in each case is the entire row, since no set has exclusive ownership of any of its members. Since these are binary relations, they are normalized.

The next set of relations are derived from the functional dependencies we previously identified.

R5	Object, Revision, State
R6	Object, Revision, Transaction
R7	Person, Object, Revision, Role, Capability
R8	Category, State, Group, Capability
R9	Category, State, Role, Capability

The next five relations add the attributes of State, Role, Revision, and Transaction. The key for each relation is based on an appropriate functional dependency. Because all of these relations except for **R7** have only one non-key attribute, the design is easily shown to be in third normal form, a benchmark for good database design[10].

[10] The design is in first normal form because there are no repeating groups; it is in second normal form because every non-key attribute is dependent on the entire primary key; it is in third normal form because there are no transitive dependencies. The details behind these criteria can be found in Dutka and Hanson [4].

The use of normalization results in a database design that has several nice properties.

- By reducing the sizes of tables, we have enabled updates that do not need as much data as is needed for the universal relation. For example, we can add information about a Person's membership in a Group without adding any other information.
- By adhering to the functional dependencies, we ensure that updates to any single table will leave the database in a consistent state. For example, removing a (Person, Group) tuple still leaves the Person involved in all the pending Transactions for which that Person had a Role.
- We have guarantees that the database has minimal redundancy.

Our schema is a relatively simple translation of functional dependencies into relations. A more involved solution might capture other dependencies that typically exist in a realistic authorization problem. Groups, for example, are often organized in a containment hierarchy so that supergroups can be defined as the union of subgroups (similarly for Categories and Capabilities).

As the dimensions of the authorization matrix increase and the dependencies proliferate, the use of the formal techniques known to database design theory become more necessary for generating a valid design. Normalization theory provides the following benefits:

1. Well-understood normal forms, including their relationships
2. Algorithms for achieving normal forms
3. Complexity results for the algorithms, including known intractable problems

These benefits are essential if we are to produce a valid authorization system and to assure its correctness and reliability.

Practical problems related to authorization

The abstract design of relational tables is important, but authorization in realistic situations involves a number of practical complexities. In this section we discuss a few of these.

Unintentional 'back doors'

One typical problem in data management systems is that not all parts of the system employ the authorization mechanism. So-called 'back doors' allow users to access information to which they do not have authorization. One common back door is found in data management systems that have recently had a search engine 'bolted on the side' to provide Web-like search capability. When the search engine is not integrated with the authorization model, it is sometimes possible to know that data exists and to view some of the content of data even if one is not officially authorized to access it. Thus, it is possible to use the search engine to find one's own name in a file named `DOWNSIZE.DOC`, and also know that this document is owned by one's manager, without having read access to the document.

Intelligent devices

Another typical problem area is in printing and scanning. Data management systems generally do not come with software for batch printing, application of watermarks and banners, and other output-device-specific software that is essential to user's notion of what a document management system ought to do. This software is typically:

- customized or locally developed; hence its security is suspect
- arcane; hence, it is difficult to evaluate its impact on security
- produced late in the deployment cycle, so there is pressure to get the software 'out the door'; hence, security is put on hold in order to complete the project
- full of interactions with intelligent devices that can be points of entry for a determined hacker.

The need for a printer or scanner to input and output information is often not explicitly represented in the authorization model (as it was not represented, for example, in our database design), and so constitutes a security hole.

MRP/ERP systems

A third area for problems is in interactions with MRP/ERP systems. Engineering systems maintain information such as bills of materials that are also needed in production. The transfer of bills to MRP/ERP systems is usually a task requiring third-party or custom software. Since this data transfer usually updates data, it requires substantial authorization privileges. The data transfer is often unprotected against spoofing.

Intelligent objects

Objects managed in a data management system may have authorization capabilities of their own. Adobe's Portable Document Format (PDF), for example, supports password-protected authorization controls on certain operations such as modification, printing, content selection, and annotation. These authorizations are controlled through Adobe Acrobat, rather than through an authorization layer in a system environment. More recently, systems have been designed which implement online authorization requests from distributed objects; examples of such systems include IBM's Cryptolopes, Xerox's ContentGuard, and Authentica's PageRecall. There are also new systems proposed to control access to electronic entertainment media [3]. A comprehensive security requirements effort must specify how these kinds of technologies should be used in concert with data management authorization.

Discussion

Authorization as important as authentication

Although authentication and encryption get more attention, many security breaches are really problems in authorization. The notorious case of Wen Ho Lee at Los Alamos is a recent example. Lee was not charged with an authentication violation (that is, pretending to be someone he was not, or attempting to gain access to information for which he did not have rights), but with transferring material from a secure computer to a non-secure one—that is, with carrying out an unauthorized operation [9]. At least part of the problem is that unauthorized operations were permitted by the computing systems.

A key difference between authentication and authorization is that the former is not a direct responsibility of business process owners[11], as is the latter. Business process owners must come to grips with authorization, because they will make authorization decisions on a day-to-day basis. Deciding who gets access to what information is a business decision, not a legal or tech-

[11] This is not to say they are unimportant—only that they will be managed by the IT group, rather than business process owners.

nical one. When access to information is itself a product, the authorization system becomes a necessary part of the production and distribution system for information.

Authorization, unlike authentication, can't be purchased as a turn-key system—vendors can sell you an empty matrix, but you have to populate it and assure it. There is little information available at a general level to assist companies in learning how to populate the matrix. Engineering data management systems are, in many cases, the most elaborate example of the authorization problem that companies have faced, and their manual precursors do not provide a very useful guide to how to manage automatic authorization.[12] Moreover, the authorization problems that companies will soon encounter could not be solved by manual methods in any case.

The problems of grouping

Role-based authorization involves specifying sets of persons and objects. We have already alluded to the multiplicative effect that errors in the specification will have. Mistakes in assigning persons to roles or documents to categories are common. The meaning and use of classifications is fundamental to security, but organizations do a remarkably poor job of establishing workable classifications. Consider the ease with which managers might confuse such categories as 'unclassified controlled nuclear information' and 'sensitive but unclassified nuclear information'[6]. Or consider the distinction between 'classifying' data and 'categorizing' it: 'PARD: Protect As Restricted Data' is not a classification level per se but 'a handling method for computer-generated numerical data or related information, which is not readily recognized as classified or unclassified because of the high volume of output and low density of potentially classified data,' and ranks between unclassified and confidential, the lowest level of classification'[9]. In general, categorization for security purposes is considered one of the most problematic areas of governmental security [11]. It is much easier to define categories than it is to ensure that managers apply them consistently and correctly. I know of one Fortune 50 company in which there are only four possible security classifications for documents, but whose managers consistently gave me different definitions of those classifications. Categorization for authorization is subject to many of the same problems that arise in other categorization contexts.

Role-based authorization is not compelling in situations where a role doesn't completely define authorization across all objects. 'Contractor' appears to be a role, and it is usually true that there is some information that no contractor should see, but on the other hand, each contractor needs read and modify access to distinctly different sets of data. Hence, in addition to requiring a role definition, it appears that we still need individual specifications for each contractor. In such circumstances, there may be as many roles as there are individuals, and so role-based authorization would not reduce the size of the authorization matrix.

A common notion of things

A desirable authorization system is one that can be applied to a wide spectrum of data. One barrier to a wide-spectrum authorization system is the lack of a standard notion of *things*. All systems share a notion of persons, expecting them to be unique and stable entities, and so authentication systems at least have a common base to work with (facilitating, for example, the idea of 'single sign-on').[13] But authorization deals with computer objects, and what constitutes an 'object' depends largely on which software package you use: operating systems deal with a universe of files, relational databases deal with a universe of tables and rows, object request brokers deal with a universe of CORBA-compliant objects, intelligent objects deal with pages or other subelements, Web servers deal with a universe of URLs, and data management systems control many of these elements and add workflow as well. The 'impedance mismatch' between these systems makes it difficult to have an authorization mechanism that spans them all.

One way to reduce impedance mismatch is to funnel all data through a single namespace. This is a fairly old technique: consider for example mapping objects such as sockets into a file system namespace. Today, more attention is given to the URL namespace. Netegrity

[12] In most cases, the manual authorization mechanism is based entirely on personal knowledge: the person authorizing an operation knows both the object and the user and makes an independent one-time decision about that specific combination of object, user, and object.

[13] Looming on the horizon are issues such as how to separate human clones, how to safeguard bio-identification techniques against advanced surgery and genetic techniques, and so on.

and Dascom are two companies with products that provide access control to a URL namespace; if you can assign URL 'names' to all your objects, then authorization for these objects can be defined through Netegrity or Dascom. This kind of approach is useful for data access, but is less satisfactory for workflow and other operations that require update. Another namespace possibility is an object-based namespace, possibly with something like the CORBA security standard [2]. There are main problem with this solution is that most organizations aren't CORBA-centric.

Architecture of an authorization system

From a system management point of view, it seems reasonable to have a single centralized authorization matrix, because this reduces administrative overhead. From a security point of view, however, there is an argument for *not* centralizing authorization. A centralized authorization system suffers the problems of any centralized system: it is less robust to failure and error. Redundancy is a common approach to increasing robustness. In manual authorization contexts, the classical form of redundancy is the *two-man* rule [8]. An ICBM launch, for example, requires the coordinated activities of a number of distinct people; this redundancy in decisions tends to make the system as a whole more robust to failure in any single component (such as, for example, error or malicious intent on the part of one of the persons). Research should be done on authorization systems that employ redundancy to improve their reliability.

It is worth rethinking the distinction between authorization and authentication. One proposal is for merged authentication and authorization systems, by incorporating authorization information in a certificate used for authentication [7]. One problem with this approach is that authorization is trending towards a time-sensitive, rule-based property, rather than an unchanging attribute of a single individual. An alternative approach that works in some contexts is to relinquish the need to authenticate a person's identity, and instead authenticate their access permissions, as is done in trust management systems. This kind of approach is unknown to most vendors and consumers of data management products today.

Conclusions

Authorization is an important problem, and one whose impact is likely to increase in the near future. Reasoning about the design and effectiveness of such systems is currently done informally. We need to consider more formal solutions to developing and assuring authorization systems. Database normalization techniques can help in this regard. Work needs to be done on developing more robust ways to categorize people and objects, on system for identifying computer objects, and on the architecture of authorization systems.

About the author

Darrell Raymond assists companies in acquiring and deploying systems for document and product data management. Raymond does extensive work with the Gateway Group, an independent consultancy that specializes in data management systems. Raymond's clients include Amtrak, Cummins Engine, Eastman Kodak, General Electric, Los Alamos National Laboratory, Oxford University Press, and Siemens. Raymond holds a Ph.D. in Computer Science from the University of Waterloo. He is presently an adjunct assistant professor in UW's Department of Computer Science and is a member of the editorial board of the Springer-Verlag journal *Markup Languages: Theory & Practice*.

Appendix: inference axioms for functional dependencies

Let A, B, C and D represent sets of attributes. Then the following rules can be used to derive the closure of a set of functional dependencies on A, B, C, and D:

Rule name	Description
Reflexivity	A → A
Augmentation	If A → B, then AC → B
Union	If A → B and A → C, then A → BC
Decomposition	If A → B, then A → C where C is a subset of B
Transitivity	If A → B and B → C, then A → C
Pseudotransitivity	If A → B and BC → D, then AC → D

References

[1] Barkley, John F., Cincottta, Anthony V., Ferraiolo, David F., Gavrilla, Serban, Kuhn, D. Richard, 'Role Based Access Control for the World Wide Web', NIST (April 8, 1997).

[2] CORBA Security Service Specification V1.2 CORBA Services Specification Chapter 15 (December 1998).

[3] Content Protection System Architecture: A Comprehensive Framework for Content Protection, Intel Corporation, International Business Machines Corporation, Matsushita Electric Industrial Co., Ltd., Toshiba Corporation (February 17, 2000).

[4] Dutka, Alan F. and Hanson, Howard H. *Fundamentals of Data Normalization*, Addison-Wesley (1989).

[5] Maier, David, *The Theory of Relational Databases*, Computer Science Press, Rockville, Maryland (1983).

[6] 'Science at its Best, Security at its Worst: A Report on Security Problems at the US Department of Energy', Report of the Special Investigative Panel of the President's Foreign Intelligence Advisory Board (June 1999).

[7] Rubin, Aviel D., Geer, Rubin, Ranum, Marcus J. *Web Security Sourcebook*, John Wiley & Sons p. 317 (1997).

[8] Sagan, Scott D. *The Limits of Safety: Organizations, Accidents, and Nuclear Weapons*, Princeton University Press (1993).

[9] Schwartz, Stephen I. 'Scientist, Fisherman, Gardener…Spy? Bulletin of the Atomic Scientists' **56**(6) pp. 24-30. (November—December 2000)

[10] Schneider, Fred B. 'Enforceable Security Policies' *Information and System Security* **3**(1) pp 30-50 (2000).

[11] Smith, Jeffrey H. 'Redefining Security: A Report to the Secretary of Defense and the Director of Central Intelligence', Joint Security Commission (February 28, 1994).