Role-Based Security, Object Oriented Databases & Separation of Duty *†

Matunda Nyanchama & Sylvia Osborn ‡
email:{matunda,sylvia}@csd.uwo.ca

October 11, 1993

ABSTRACT

In this paper we combine concepts of role-based protection and object oriented (O-O) databases to specify and enforce separation of duty as required for commercial database integrity [5, 23, 24]. Roles essentially *partition* database information into access contexts. Methods (from the O-O world) associated with a database object, also partition the object interface to provide *windowed* access to object information. By specifying that all database information is held in database objects and authorizing methods to roles, we achieve object *interface distribution* across roles. For processing in the commercial world we can design objects and distribute their associated methods to different roles. By authorizing different users to the different roles, we can enforce both the order of execution on the objects and separation of duty constraints on method execution.

Keywords: Roles, role-based protection, access control, context, least privilege, separation of duty, object oriented databases, methods, objects, classes.

1 Introduction

Roles group system privileges into units that can then be authorized to users as single units. Role-based protection eases the task of managing large numbers of users or user groups and/or large numbers of system privileges which might overlap [17, 25]. Viewed this way, roles *partition* system/database information into *contexts* or classes of information. Authorization to a role facilitates access to the information associated with (accessible via) the role. A role can thus be seen as a *window* into some database for authorized users.

Roles implement *least privilege*, ensuring that authorized users access only the information necessary for performing desired tasks. In specifying user role authorization, it must be ensured that users are authorized only to non-conflicting roles [10, 15].

Object oriented (O-O) principles require that information access be done via method invocations. Methods in turn access database objects which are the information bearing receptacles. Using some appropriate method design and access control, information encapsulated in an object can be *windowed* such that only authorized information is visible to a user. The rest of object information remains hidden. Ting et. al. [25] have employed this approach to provide differentiated access to object information in an O-O design environment.

Given the foregoing, we argue that just as roles partition the database into contexts of information, so also does method access to objects partition the object interface to provide *windowed* access to object information. This paper uses roles and O-O principles to specify the enforcement of *separation of duty*.

The requirement for separation of duty [5, 23, 24] is found in commercial security applications (see [5, 23, 24, 10, 15]) where in processing, a user who has participated in one step in an execution process is barred from executing further steps in the same process. Barring collusion, separation of duty ensures that the rules specifying the manner of accomplishing a task are adhered to. We combine role-based protection and the O-O approach and show how to realize separation of duty. The main idea is to keep track of an *object history* within each object.

In the next section we formally define roles and give a brief outline of role-based protection. Section 3 briefly outlines the O-O concepts such as classes, objects (as instances of classes) and methods. More details on O-O approaches can be found in [8, 1, 2, 4, 6, 12, 14, 19]. In section 4 is a summary of commercial database security, in particularly, the concept of separation of duty. In section 5 we combine role-based protection and O-O principles to enforce separation of duty. We offer a summary in section 6.

2 Role-Based Protection: An Overview

2.1 Role-Based Protection Basics

Role-based approaches use differentiated access to system privileges to realize system protection. A privilege, in this context, determines a subject's access rights with

*This work was supported in part by a grant from the National Science & Engineering Research Council of Canada.

†Submitted to SIGMOD RECORD, June 1993

‡The Department of Computer Science, The University of Western Ontario, London Ontario N6A 5B7 Canada, FAX: (519) 661-3515 email:{matunda,sylvia}@csd.uwo.ca

respect to the associated data item, system resource, etc. Thus a privilege can be viewed as a token whose possession confers access rights to the subject (user or process acting on user's behalf) possessing it. A privilege is specified by its name and a set of access modalities to the associated object(s).

Definition 1 *A privilege is a pair (x, m) where x represents (name, identifier, etc.) an object (data item, resource, etc.) and m is a non-empty list of access modalities for object x.*

In practice, x can refer to an object (such as a protected data item, an O-O class definition or its extent, etc.) or a resource (e.g. printer). In systems with simple access modes such as read, write, execute, etc. m, the list of modalities, is a subset of these access modes. In more complex systems such as O-O environments, this list of access modalities is a list of methods. Indeed, m can be a list of transactions involving x. When x is some resource, m is a list of modes that facilitate its access and/or use. The exact nature of x and m is a matter of the application and the associated security policy.

Definition 2 *A role is a named collection of privileges. It is a pair (rname, plist) where rname is the role name and plist is the privilege list.*

Roles are named groups of related privileges pertaining to protection objects, resources and/or the system in an information system. The privileges encapsulated in a role are administered as a single unit so that granting a user/group access to a role authorizes such a user/group to exercise the privileges in the role [17].

Definition 2 only captures the *functional* [7] component of a role. The other important component is the *structure* which should capture a role's relationship with other roles in the system [7]. The nature of role relationships, hence role structure, is an aspect of role organization. Structures such as hierarchies [25], lattices [20] and Ntrees [21, 22] have been proposed for role organization. In general, both the functional and structural components of a role are necessary to *completely* specify the role. We do not address role structuring but assume that such structural specification exists within or outside the role specification.

The main advantage of role-based protection is that it eases the administration of a large number of system privileges. This can be enhanced further should users themselves be grouped such that authorizations to roles are given to user groups instead of individuals. Roles offer flexibility in the granting and revoking of privileges by alteration of a role's privilege list or user/group authorization to the role. They facilitate the implementation of *least privilege* in which a role list contains the fewest privileges necessary to perform associated duties. As well, roles can be designed at the application level

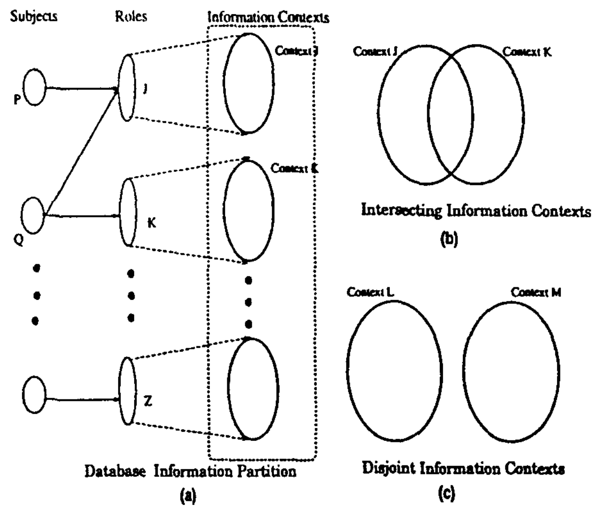

Figure 1: Role Database Partition

which allows for integration of security related application semantics.

The main disadvantage of roles is that the analysis of user privileges and their distribution to various users/user groups can be a very complex process.

2.2 Roles & Information Partition

Since roles facilitate access to information based on the privileges they encapsulate, they offer access to different pieces of protected information/resources. In essence, a role groups system privileges into a unit that is then authorized to users as a single unit. For a user authorized for a given role(s), only the information accessible via the role(s) is available to the user. Viewed this way, and for a protected database, a role is a *window* into the database. The information visible/accessible via such a window is a context[1] by itself. In general, roles *partition* database information into *contexts* with each context of information/resources accessible via the associated role (see figure 1a). Depending on the application, these contexts may or may not have overlapping information (see figures 1a & 1b). The intersection (or lack of it) of information contexts is matter of the security policy.

3 O-O Databases

3.1 Some O-O Basics

O-O Database Systems have evolved in an attempt to approximate real world entity modeling. They capture more real world semantics, a fact that makes them better at modeling complex entities than their relational

[1]This context can be seen as a class of information. However, we use the term context to avoid the confusion that may arise with classes in the O-O paradigm.

counterparts. Therefore, they find applications in complex modeling environments such as computer aided design/manufacturing (CAD/CAM), geographic information systems (GIS), very large scale integration (VLSI), etc. In this section we emphasize only those aspects that are of relevance to our current formulation.

O-O databases support conventional database functionalities such as persistence, concurrency control, recovery, some form of storage management that includes indexing, an *ad hoc* query facility, a provision for schema definition and evolution, etc. As well, an O-O database must incorporate concepts from the O-O paradigm including concepts of complex objects and aggregation, encapsulation, polymorphism, classes, extensibility, class hierarchies and inheritance [2, 6, 12].

3.2 An O-O Model

A *class* defines the structure and behaviour of its instances. Class structure is defined by instance variables (attributes) and their types. Instances can be simple or complex since the domains of attributes can be simple or complex. *Behaviour* is determined by the methods defined in the class. *Methods* operate on the instances of the class on being invoked by corresponding messages. The set of messages that a class responds to is its *interface*. O-O databases allow for *extensibility* which facilitates the introduction of user-defined classes from existing ones. Such classes are handled in the same manner as system-defined ones, i.e. *seamlessly*.

Definition 3 *An object (o) in an O-O database is a triple [14, 11]: o = (oid, class, state) where* oid *is the unique* object *dentifier,* class *is the class of which the object is an instance and* state *represents the value of the object.*

oid and class are drawn from the countably infinite universes $\mathcal{OID}$ and $\mathcal{CN}$ for object identifiers and class names, respectively.

Definition 4 *The state of an object O_state= H_state ∪ NH_state where H_state=(H_attr,H_attrval) and NH_state={(NH_attr,NH_attrval)} represent the attribute-value pairs for attributes whose history is kept, and not kept, respectively.*

The value of the H_state of an object carries its *history*. A history is an *ordered sequence* of events where:

Definition 5 *An event is a quadruple e=(evname,act,uid,time) where* evname *is the event name,* act *is the nature of the action,* uid *is the identity of the subject executing the event,* time *is a chronological indicator of the time of the event.*

Consequently:

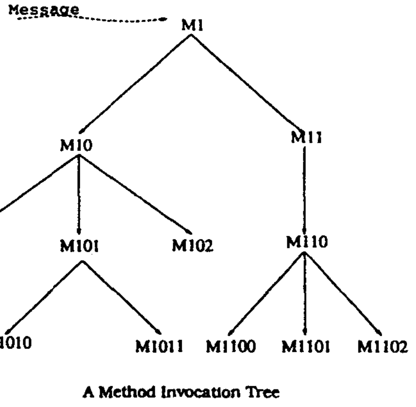

Figure 2: A Method Invocation Tree

Definition 6 *A history $H = \langle e_1, e_2, e_3, \cdots \rangle$, where each e_i is an event. A finite history is of the form $H = \langle e_1, e_2, e_3, \cdots, e_n \rangle$ where n is finite. It is infinite otherwise. H and n are related via the size function, i.e. $n = size(H)$. Given two events e_i, e_j we say e_i precedes ($\preceq$) e_j if and only if $e_i.time \le e_j.time$.*

Each event stores necessary audit information resulting from the occurrence of the event. The exact nature of this information is application dependent. Given all object histories, one can construct the system audit record by ordering the events according to their time (**e.time**) of occurrence and appending the object name or oid.

Definition 7 *A class c is a triple:* (**n**, **s**, **m**) *where* **n** *is the name of the class,* **s**, *is its structure and* **m** *is the method list applicable to the class.*

All communication between objects in O-O databases is via *messages*. Messages invoke methods which manipulate the objects as defined in the class hence providing *encapsulation*: we cannot access the representation other than through the class interface.

Definition 8 *Given an object o and the set of database messages (regarded as the message universe in the system) $\mathcal{MS}$, the interface $OI(o) \subseteq \mathcal{MS}$ is the set of messages understood by o.*

In this work $OI(o) = Methods(o)$.

A Method invocation can take various forms. It can access (read and/or update) an object's attributes; it can invoke other methods (associated with the same object); it can send messages to other objects; it can create new objects. In general, method invocation is in the form of a tree (see figure 2 and also [9]).

To ensure *Polymorphism*, instances of different classes can receive the same message but respond differently depending on the class of the receiver of the message.

3.3 O-O Objects, Methods & Interface Windowing

In O-O databases, an object's information is captured by its state which is determined by the values of its attributes. Method invocations form the only means of access to this object information. Hence methods manipulate the object state and can either read or update this state.

Methods can be structured in such a manner that different portions of the object information are available via different methods to facilitate access control. We term this differentiated access to object information via its interface *object interface windowing.* A subset of an object's interface is an *interface partition* or *window.* Indeed, any subset of the messages an object responds to is a *window* into object information.

Definition 9 *An interface partition, part(OI(o)) of some object o, is any subset (or nil) of the object interface OI, i.e.* $part(OI(o)) \subseteq OI(o)$,. *Given some interface* $OI(o)$, $part(OI(o)) \in 2^{OI(o)}$.

In specifying and enforcing access control, this windowing effect can be exploited by authorizing different users to access different portions of object information via the associated methods. By explicitly (or implicitly) authorizing different users to execute different methods, we realize differentiated access to object information. We define a privilege based on interface partitions as:

Definition 10 *An o-privilege is a pair (x,part(OI(x))) where x is the object (resource) name and part(OI(x)) is some partition of the interface of x.*

One can enforce the principle of *least privilege* by ensuring that only the necessary methods, that avail sufficient information to a user, are authorized to the user.

Example 1 Consider an automated cheque issuing process in which two "signatures", of a clerk and supervisor, are required to be appended onto a cheque, and where the clerk's must come before that of supervisor. The cheque object is an instance of a class **CHEQUE** with two methods, **clerk** and **supervisor**, which append (update) the clerk and supervisor signatures to the cheque object, respectively. The class definition (using the syntax of [18]) will be of the form:

```
Name:       CHEQUE;
Structure: { PAYEE: String,
             PAYEE_ID: String,
             AMOUNT: Currency,
             SIGN_1: String,
             SIGN_2: String };
Methods:   { clerk,supervisor }
```

Let method **clerk** be implemented to update the PAYEE, PAYEE_ID, AMOUNT and SIGN_1 attributes with the payee name, payee identifier, the amount of the cheque and signature, respectively. Then audit trail is updated with the appropriate information and the cheque is "dispatched".[2] On "receipt", the supervisor invokes method **supervisor** which, among other things, updates SIGN_2, the audit trail is updated, and the cheque is dispatched for payment. A security system will specify authorization to the appropriate methods such that subjects in the clerk and supervisor roles can execute the **clerk** and **supervisor** methods, respectively.

Audit trail information would be necessary if execution depends on past history.

4 Separation of Duty & The O-O Paradigm

Separation of duty is applied where several people (or processes acting on their behalf) are required to perform a given task. Such a task would be broken into subparts which are then assigned to different people. Every individual is then required to perform (say) only one of the subtasks with the restriction that none of the individuals can perform more than one subtask. From example 1, separation of duty will bar a single individual from updating both SIGN_1 and SIGN_2.

The main idea of separation of duty is to ensure that no individual can initiate action, approve the same action and (possibly) benefit from the action. Separation of duty aims to spread the responsibility for various processing steps across different individuals (or their proxies) and achieve dispersion of of authority across individuals that access database information.

Separation of duty is a major requirement for commercial database integrity [5, 10, 15, 23, 24]. The Clark and Wilson model for commercial security [5] proposes to model the security requirements of commercial environments, which stress integrity more than secrecy. Thus, apart from separation of duty, there are requirements such as all database items being constrained data items (CDIs), that only certified transformation procedures (TPs) manipulate the CDIs, that the TPs are certified to take CDIs from one correct state to another, that there are verification procedures (IVPs) that assure the integrity of code which manipulates the CDIs, etc. In general, all data transformations are required to be designed as well-formed transactions (WFTs) where a WFT is a program that has been certified to maintain the integrity of the data it manipulates [24].

Separation of duty requires the association of TPs with users and the associated CDIs which leads to relationships of the form $(UID, (TP_i(CDI_1, CDI_2, \cdots)), \cdots)$ where UID is the user identity. It is important to specify what are the constrained data items and transformation procedures. In this work, we make the following constraint:

Constraint 1 *All data items whose history is neces-*

[2] Assume there is some ordering mechanism.

sary for their processing must be handled as CDIs. The associated access procedures must be well formed transactions.

The enforcement of these policies must be mandatory, i.e. system behaviour is defined *a priori* and cannot be altered while the system is running. Subjects' and objects' attributes must be the only basis for granting authorization according to specified rules. Moreover, subjects can neither alter nor transfer security relevant information to third parties. Further, neither ordinary subjects nor system security officers can alter information pertaining to their own authorizations. As well, any modification of authorization information must be subjected to separation of duty.

5 Roles, O-O Objects & Separation of Duty

5.1 *Roles & The O-O Paradigm*

Roles as seen in section 2 offer differentiated access to database information based on their privileges. In the O-O paradigm, information is held in the state of objects and is accessible via methods as seen in section 3. In this section we combine the two concepts to exploit the advantages of role-based protection and those of the O-O paradigm.

O-O database objects are defined using O-O principles and made accessible via associated methods only. The authorization to object information is realized via role "authorization" to the associated method according to the definition of privilege (see definition 10) which leads to role definition of the form:

Definition 11 *An o-role is a named collection of o-privileges i.e.* $o\text{-}role = (o\text{-}rname, \{\cdots, (o_i, int_{i,j}), \cdots\})$ *where* o_i *is some object and* $int_{i,j}$ *is the associated interface to the object and is of the form* $part_j(OI(o_i))$.

For the rest of this paper we do not make a distinction between definitions 2 and 11.

User access to database information comes via user role authorization specified in a role's access control list which is a finite set of subject/group identifiers:

Definition 12 *A role access control list (racl) is of the form:* $\{id_1, \cdots, id_n\}$ *where* $id_i \in ID$ *is a user* ($uid \in \mathcal{UID}$) *or group identifier* $gid \in \mathcal{GID}$)*. In general* $\mathcal{ID} = \mathcal{UID} \cup \mathcal{GID}$.

Let $\mathcal{R}$ and $\mathcal{M}$ be the universal sets of roles and methods, respectively. Then $\forall r \in \mathcal{R}, \exists$ r.racl=$\{\cdots, id_i, \cdots\}$.[3] A secure role is one with an associated access control list.

[3] We use the dot notation to refer to role name and role privilege list as r.name and r.plist, respectively.

Definition 13 *A secure role is a named collection of privileges along with its access list. It is a triple (rname,plist,racl) where rname is the role name, plist is its privilege list and racl is its access control list.*

Definition 14 *An access strategy* Φ *is of the form:* $\Phi : \mathcal{ID} \times \mathcal{R} \times \mathcal{O} \times \mathcal{M} \mapsto \{true, false\}$.

With $id \in \mathcal{ID}, r \in \mathcal{R}, o \in \mathcal{O}$ and $m \in \mathcal{M}$ we have:

$$\Phi_1(id, r, o, m) = \begin{cases} \text{true} \iff & \text{id} \in \text{r.acl} \\ & \wedge \ (\text{o, m}) \in \text{r.plist} \\ \text{false} & \text{Otherwise} \end{cases}$$

The condition id $\in$ r.acl (user-role authorization) ensures that the current user is authorized to execute the role while (o, m) $\in$ r.plist ensures that there is an associated privilege defined in the role.[4]

The effect of user role authorization is to generate relations of the form: $(rname, \{(o_i, m_j), \cdots\}, \{\cdots, id_p, \cdots\})$ with rname being the role name, o_i the object, m_j is a method and $\{\cdots, id_p, \cdots\}$ is the access control list.

Definition 15 *A user's privilege authorization scope is the set of all privileges accessible to the user.*

Given an authorization scheme and some $id \in ID$ we can generate a user's *authorization scope* of the form: $(id, (rname_a, rname_b, \cdots))$. Substituting each role with its definition of the privilege list and rearranging the result yields a relation of the form $(id, (m_j, (o_1, o_2, \cdots)), \cdots)$. Since we regard methods as TPs and the objects o_is as CDIs, we have a similar relationship as that of section 4.

The O-O paradigm is suitable (almost natural) for this scheme of role-based protection in which we have roles authorized to execute specific methods associated with some objects (see figure 3). The resulting effect is the distribution of an object's interface across roles.

Example 2 Consider the cheque process of example 1 and let the cheque object have the same methods **clerk** and **supervisor**. To associate these methods with roles, define two roles, **CLRK** and **SPV**, corresponding to clerks and supervisors, respectively. **CLRK** and **SPV** are then authorized to execute methods **clerk** and **supervisor**, respectively. This leads to role definitions of the form: (SPV, {(cheque,supervisor)}) and (CLRK, {(cheque,clerk)}) which effectively distributes the cheque object interface to two roles.

Next, individuals are authorized to execute the roles. For instance (say) John and Margaret are authorized for the **CLRK** and **SPV**, respectively. The roles with their access control lists now look like: (SPV,{(cheque,supervisor)},{Margaret}) and (CLRK, {(cheque,clerk)},{John}).

[4] (o, m) can be defined as any subset of the authorized interface in the role.

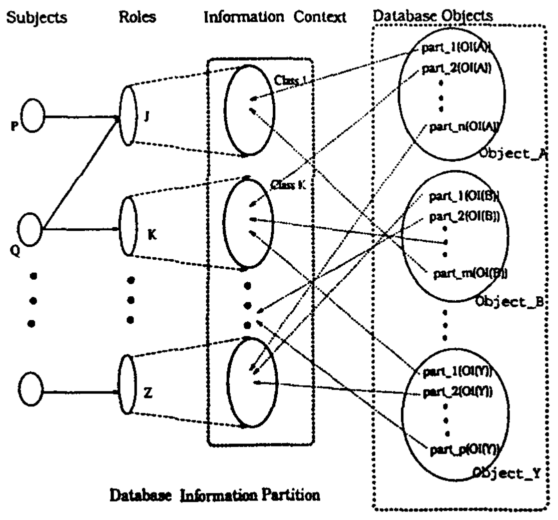

Figure 3: Object Interface Distribution Over Roles

5.2 *Enforcing Separation of Duty*

Separation of duty requires audit information to ensure that before subjects are allowed execution, they have not participated in the processing before. However, as Karger [10] observes, searching for such information in the audit record can be very costly. Hence we use object history, which is part of the object state hence part of the object itself. This enables each object (at least those that require separation of duty in their processing) in the database to keep track of its own audit information. We introduce a history attribute of the object to record audit information.

The class structure must be defined to reflect the desired object structure to ensure that objects (at least those that require separation of duty) keep track of their histories. The history has a value which is the audit information required for its processing. In defining a class, then, we not only specify that there be a history attribute but also its nature, i.e. the domain of its value. This history provides no more information than can be found in the audit trail; nor does it preclude the storage of the same information in the system audit record. It merely avails the same information in a form that supports performance improvement.

To enforce separation of duty requires non-participation in the current history which is necessary, but not sufficient, to guarantee access at any execution stage [10]. The final decision on whether or not to allow access must depend on authorization and any constraints imposed on access that may take into account both the history and any other required information to make such a decision. Karger [10] makes similar observations regarding token capabilities for control of object access.

Our refined access strategy retains definition 14 but imposes a separation of duty criterion.

With $id \in \mathcal{ID}, r \in \mathcal{R}, o \in \mathcal{O}$ and $m \in \mathcal{M}$ we have:

$$\Phi_2(id, r, o, m) = \begin{cases} \text{true} & \iff \text{id} \in \text{r.acl} \\ & \wedge \forall e_i \in H,\ \text{id} \neq e_i.uid \\ & \wedge\ (\text{o, m}) \in \text{r.plist} \\ \text{false} & \text{Otherwise} \end{cases}$$

The condition id $\neq e_i.uid, \forall e_i \in H$ ensures prior non-participation for the current user in any previous event.

For this history to be useful, method executions must either update the history attribute or be part of some transaction whose execution updates the attribute. Processing constraints must ensure that each permitted (or attempted) execution on the object utilizes the history and updates it.

Example 3 Consider the cheque object of examples 1 and 2 which, as defined, do not keep track of execution history. We introduce another attribute (**HIST**), to record audit information associated with the object. The redefined class structure of **CHEQUE** is:

```
Name:       CHEQUE;
Structure: { PAYEE: String,
             PAYEE_ID: String,
             AMOUNT: Currency,
             SIGN_1: String,
             SIGN_2: String
             HIST: SequenceofEvents };
Methods: {clerk,supervisor }
```

Further, method executions must be redesigned to update this attribute on attempted execution.

```
on invocation of method (m)
check:= $\Phi_2(id, r, o, m)$;
if check then
    begin
        execute method;
        update(o.HIST)
    end
else update(o.HIST)
```

In example 3, method execution is part of a transactional process that reads history information, uses it along with authorization information and updates the history. This illustration is similar to what Ravi Sandhu [23] terms *transactional expressions*. We do not address the manner in which these executions are structured and processed. It suffices (for now) to say that it must be transactional in nature.

Notice also that our formulation realizes *dynamic* separation of duty [16] in that all we care about is that the current access attempt is authorized and that the said user's participation is not in the object history.

6 Summary

We have discussed the enforcement of separation of duty using both role-based protection and O-O database principles. Roles offer a flexible means of managing system privileges for different numbers of users/user groups with varying information access requirements. They can be employed at the application level, thus incorporating application level semantics. They offer a flexible means of administering system privileges in that access rights can be conferred and/or revoked via user authorization to a role or role privilege assignment. Roles can employ the principle of least privilege and effectively partition database information into contexts which could or could not overlap.

In the O-O paradigm, methods, the only means of access to object information, provide a windowing effect on this information. By authorizing different roles to execute different methods of an object, we effectively distribute the object interface across roles and hence the individuals authorized for the associated roles.

To realize separation of duty we must keep track of an object's history and use it, along with access control information, to determine whether or not to allow access. Using O-O principles we can incorporate this audit information in the object structure and impose conditions on method execution that must access the history (before) execution and update it on completion of execution. Making audit information part of the object facilitates ease of processing as searching for the same information in a common audit trail would be too expensive.

Methods themselves can be made transactional or be part of some transactional execution where object history is used and updated on any access or attempted access to object information.

References

[1] T. Andrews and C. Harris. Combining Language and Database Advances in an Object Oriented Database Environment. In S. B. Zdonik and D. Maier, editors, *Readings in Object Oriented Database Systems*. Morgan Kaufmann, 1990.

[2] M. Atkinson, F. Bancilhon, D. DeWitt, K. R. Dittrich, D. Maier, and S. Zdonik. The Object Oriented Manifesto. In *ACM SIGMOD '90 Proceedings*, page 395, May 1990.

[3] R. W. Baldwin. Naming & Grouping Privileges to Simplify Security Management in Large Databases. In *Proc. 1990 IEEE Symposium on Research in Security and Privacy*, pages 116–132. IEEE Computer Society Press, May 1990.

[4] J. Banerjee, H. T. Chou, J. F. Garza, W. Kim, D. Woelk, N. Ballou, H. J Kim, F. Manola, and U. Dayal. Data Model Issues for Object Oriented Applications. *ACM Trans. Office Information Systems*, 5(1):3–26, Jan. 1987.

[5] D. D. Clark and D. R. Wilson. A Comparison of Commercial and Military Security Policies. In *Proc. 1987 IEEE Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society Press, April 1987.

[6] K. R. Dittrich. Object Oriented Database Management Systems: The Next Miles of the Marathon. *Information Systems*, 15(1):161–167, Mar 1990.

[7] J. E. Dobson and J. A. McDermid. Security Models and Enterprise Models. In C.E. Landwehr, editor, *Database Security II: Status & Prospects*, pages 1–39. North-Holland, 1989.

[8] The Object Oriented Database Task Group. Final Report of the Object Oriented DataBase Task Group–OODBTG. Sept 1991.

[9] S. Jajodia and B. Kogan. Integrating an Object-Oriented Data Model with Multilevel Security. In *Proc. 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 76–85. IEEE Computer Society Press, May 1990.

[10] P. A. Karger. Implementing Commercial Data Integrity with Secure Capabilities. In *Proc. 1988 IEEE Symposium on Security and Privacy*, pages 130–139. IEEE Computer Society Press, April 1988.

[11] S.N. Khoshafian and G.P. Copeland. Object Identity. In *OOPSLA '86 Proceedings*, pages 406–416, Nov 1986.

[12] Won Kim. Object Oriented Databases: Definitions and Research Directions. *IEEE Trans. on Knowledge and Data Engineering*, 2(3):327–341, Sept 1990.

[13] L. G. Lawrence. The Role of Roles. *Computers & Security*, 12(1):15–21, Feb 1993.

[14] C. Lecluse, P. Velez, and F. Velez. O2 an Object Oriented Data model. In *Proc. ACM SIGMOD Int'l Conference on Management of Data*, 1988.

[15] T. M. P. Lee. Using Mandatory Integrity to Enforce "Commercial" Security. In *Proc. 1988 IEEE Symposium on Security and Privacy*, pages 140–146. IEEE Computer Society Press, April 1988.

[16] M. J. Nash and K. R. Poland. Some Conundrums Concerning Separation of Duty. In *Proc. 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 201–207. IEEE Computer Society Press, May 1990.

[17] M. Nyanchama and S. L. Osborn. Role-Based Security: Pros, Cons & Some Research Directions. *ACM SIGSAC Review*, 2(2):11–17, June 1993.

[18] S.L. Osborn. Algebraic Query Optimization for an Object Algebra. Tech. Report #251, Department of Computer Science, University of Western Ontario, London Canada, 1989.

[19] S.L. Osborn. The Role of Polymorphism in Schema Evolution in an Object-Oriented Database. *IEEE Transactions on Knowledge and Data Engineering*, pages 310–317, Sept. 1989.

[20] F. Rabitti, E. Bertino, D. Woelk, and W. Kim. A Model of Authorization for Next Generation Databases Systems. *ACM TODS*, 16(1):88–131, March 1991.

[21] Ravi Sandhu. The NTree: A Two Dimensional Partial Order for Protection Groups. *ACM Trans. on Computer Syst.*, 6(2):197–222, May 1988.

[22] Ravi Sandhu. Recognizing Immediacy in an N-Tree Hierarchy and its Applications to Protection Groups. *IEEE Trans. on Software Engineering*, 15(12):1518–1525, Dec 1989.

[23] Ravi Sandhu. Separation of Duties in Computerized Information Systems. In S. Jajodia and C. E. Landwehr, editors, *Database Security, IV: Status and Prospects*, pages 179–189. North-Holland, 1991.

[24] D. J. Thomsen. Role-Based Application Design and Enforcement. In S. Jajodia and C. E. Landwehr, editors, *Database Security, IV: Status and Prospects*, pages 151–168. North-Holland, 1991.

[25] T. C. Ting, S. A. Dermurjan, and M. Y. Hu. Requirements Capabilities and Functionalities of User-Role Based Security for an Object-Oriented Design Model. In C. E. Landwehr and S. Jajodia, editors, *Database Security V: Status & Prospects*, pages 275–296. North-Holland, 1992.