Proc. of the 16th NIST-NCSC National Computer Security Conference, Baltimore, MD, September 20-23 1993, pages 63-74.

DISCRETIONARY ACCESS CONTROL IN OBJECT-ORIENTED DATABASES: ISSUES AND RESEARCH DIRECTIONS

Roshan K. Thomas and Ravi S. Sandhu[1]

Center for Secure Information Systems
&
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444

ABSTRACT In recent years we have witnessed considerable efforts in the research and development of object-oriented database management systems. As object-oriented database technology matures, the availability of adequate access control mechanisms will be crucial to its commercial acceptance. In this paper we discuss discretionary access control issues in object-oriented databases. Our objective is two-fold. One objective is to survey the state of the art in access control concepts and mechanisms as reported in the relevant literature. To do this, we develop a framework to categorize access control issues. The categories include subject to object, inter-object, and intra-object access control. We cover structural and behavioral approaches to access control. Another objective is to identify several research directions and access control issues that are beyond the scope of existing mechanisms. These include authorizations based on separation of duties and multiple approvals, the incorporation of temporal semantics, and transaction based authorization.

Keywords: Object-oriented databases, discretionary access control, integrity, authorization, protection groups, separation of duties, composite objects

1 INTRODUCTION

In recent years we have witnessed considerable efforts in research and development of object-oriented databases. The driving force behind these efforts have come from emerging application domains such as computer-aided design (CAD/CAM), software development, office automation, to name a few. These domains call for modeling capabilities that are beyond the scope of record-based data models. The main attraction of the object-oriented paradigm is its ability to model entities with complex structure and behavior.

With the ever-increasing threats to the security of computing systems, the maturing and commercial acceptance of object-oriented database technology depends to a large degree on the provision of adequate security and integrity mechanisms. In this paper, we survey discretionary access control issues and mechanisms that have been reported in the current literature, and further identify some promising research directions.

In order to fully exploit the benefits of the object-oriented paradigm, it is important that we consider the data model impacts of object-orientation on access control mechanisms. In particular,

[1]The work of both authors is partially supported by a grant from the National Security Agency, contract No: MDA904-92-C-5140. We are grateful to Pete Sell, Howard Stainer, and Mike Ware for their support and encouragement.

the data elements and units of access, as well as the different operation types (that need to be supported by the access control mechanism), are all heavily influenced by the underlying data model. At the same time, we must recognize that there are general principles and mechanisms that are unaffected by the object-oriented data model and thus still applicable. An example of this would be the idea of grouping users into access control/protection groups. This would offer the obvious capability of granting (and revoking) privileges to an entire group, thereby eliminating the burden of providing such privileges individually to every member of the group. The harmonious marriage of data model dependent and general access control mechanisms is the key to building a flexible and yet general purpose access control facility for object-oriented databases.

Dittrich [4] has provided a useful taxonomy of object-oriented databases. Structurally object-oriented database systems provide support for the modeling and manipulation of complex (nested) object structures. Behaviorally object-oriented database systems model the behavior of real world entities by allowing the user to define type-specific operators (methods) that make up object interfaces. An object is thus essentially an instance of an abstract data type. Object state is now accessible only through these methods. Fully object-oriented systems provide the capability for modeling both the structure as well as behavior of objects. We will discuss later in the paper, access control mechanisms that are specific to each category.

The current literature in access control and integrity mechanisms for object-oriented databases do not elaborate in any detail issues such as separation of duties, authorizations based on multiple approvals, temporal semantics, to name a few. We identify some promising approaches to address these issues.

The rest of this paper is organized as follows. Section 2 discusses the many issues, approaches, and mechanisms of discretionary access control that have been reported in the literature. Section 3 highlights some research directions, and section 4 concludes the paper.

2 DISCRETIONARY ACCESS CONTROL: ISSUES AND APPROACHES

In this section we give a brief introduction to the object-oriented paradigm and basic concepts in access control, followed by a discussion of access control issues and mechanisms for structurally and behaviorally object-oriented databases.

2.1 Overview of Basic Concepts

In the object-oriented paradigm, the *object* is a central abstraction that models a real world entity. Every object encapsulates some state and is further uniquely identified by an object-identifier. The state of an object is made of the values of its attributes (that describe the real world entity modeled). In behaviorally object-oriented databases the object state is accessible only through the operations (methods) supported by its interface(s). Every operation (method) is associated with a method body that contains some piece of executable code that models the behavior of the corresponding real world entity. Every object belongs to a type that is determined by its *class*, and is thus considered to be an instance of the class. A class is thus akin to an abstract data type definition. Classes can be organized into class hierarchies enabling the sharing of structure and behavior through the mechanism of *inheritance*. A class may inherit from higher classes, but in addition may also contain locally defined structure and behavior. A class lower in the hierarchy is thus considered to be more specialized than the higher superclasses.

When an object references a second object and is related to the latter by an IS-PART-OF relationship, we model the notion that the second object is a part (component) of the first. A collection of related objects in this manner can be treated logically as a single unit called a *composite object*. In the model for composite objects discussed in [9] an individual component may be exclusive or shared. If a component is declared to be exclusive then it can be a component only in one composite object, at any given time. If it is shared, it may be a component of several composite objects.

In addition to composite objects, some systems also support the notion of a *versioned object*. A versioned object consists of a hierarchy of objects called a version hierarchy. Objects in the version hierarchy are derived from one another, with the root object (called the generic object/instance) storing the history of change in the hierarchy.

Historically, the access control problem has been couched within the framework of subjects, objects, and rights (access types). Within this subject-object paradigm of access control, an *object* refers to any entity that holds data (such as files, records, directories). When we discuss access control in object-oriented databases, we must map this general notion of objects to the narrower meaning of objects in the object-oriented sense.

All access control problems eventually seek an answer to a fundamental question typically posed as follows: Is subject s allowed access of type a on object o? As given in [14], it is useful to consider the notion of an *authorization* as a 3-tuple (s, o, a), where $s \in S$, the set of subjects, $o \in O$, the set of objects, and $a \in A$, the set of access/authorization types. An example of an authorization would be $(John, Mydirectory, Read)$. The answer to any access control request can now be obtained by utilizing a function f that determines if the corresponding authorization (s, o, a) is true or false. In [14], the authors advance the notions of implicit, positive, negative, strong, and weak authorizations. A brief look at these concepts is useful for later discussion.

Rather than storing the value of the function f explicitly for all possible triplets (s, o, a), the idea of implicit authorization allows us to deduce some of these values from ones that are stored (in the authorization base). This may be useful for example, if we want authorization to a class to imply authorization to all instances of the class. A positive authorization gives permission for access ($f(s, o, a) = true$), while a negative authorization models a prohibition ($f(s, o, \neg a) = true$). Finally, a strong authorization (including implied ones) cannot be overridden, while weak ones can.

A well known access control principle is to organize subjects into access control groups, based on their roles in an organization [16]. This makes it easier to grant and revoke authorizations to entire groups of subjects/users at a time. The existing proposals on discretionary access control in object-oriented databases, have taken advantage of this. In [14] a role lattice is used to define such groups, and implicit authorizations propagate from the top to the bottom of the lattice.

The approach in [14] also utilizes an authorization object lattice. Thus if we want authorization on a class to imply authorizations on all instances of the class, we would define authorization objects *class* and *setof-instances* and form a lattice with the latter being lower in the lattice. Implicit authorizations are now applied on the lattice (see figure 1). As per the convention in [14], we show in *italics* the nodes from which implicit authorizations may flow to a set of authorization objects.

We end this overview section by giving a useful categorization of both structural and behavioral access control issues in terms of where access is mediated. We distinguish three cases:

- **Subject to Object:** Here we are concerned with how a subject (or principal) establishes an initial authorized point of contact with an object.[2]

- **Inter-object:** Inter-object access control is concerned with issues such as the visibility and and propagation of authorizations across object boundaries, as a consequence of an initial subject to object authorization.

- **Intra-object:** Here we deal with access control within the internal structure and behavior of an individual object. These issues are thus irrelevant to other objects in the system.

The above categorization allows us to see which dimension of the overall access control problem is tackled by individual approaches, and where these approaches are collectively lacking.

[2]For convenience we use the terms "subject" and "principal" synonymously. In a strict sense, a human user may have several principals, with each principal associated with one or more subjects in the system.

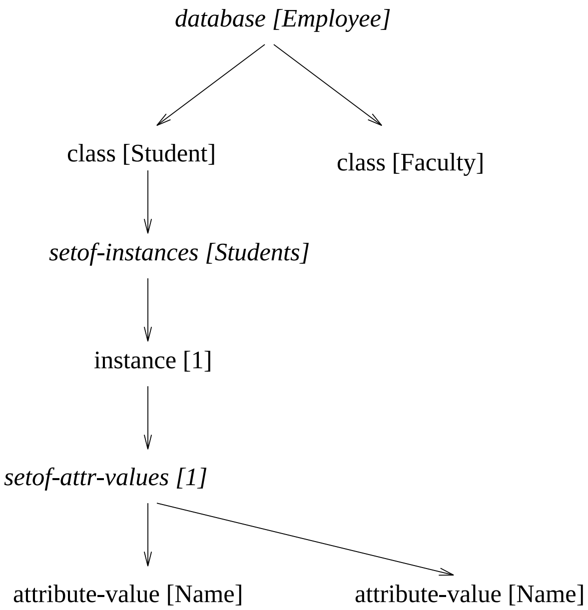

Figure 1: An authorization object lattice for classes

2.2 Structure-based access control

In structural approaches to access control, the access/operation types we deal with are typically read, write, delete, and read-definition. Thus, an access control request poses the basic question: Is subject A allowed to read/write/delete object O? Let us see the details on how this question is answered.

2.2.1 Subject to object access control

Existing approaches in the literature for subject to object structure-based access control are rather straightforward [4, 14]. The basic idea is to group subjects into access control groups and to grant authorizations in terms of access types such as read, write, and delete. These access types are usually ordered such that the authorization for one right may include others. Thus an authorization for a delete may imply authorization for a write, which in turn may imply authorization for a read. In [14] this is accomplished by utilizing implicit authorizations along an access/authorization type lattice.

2.2.2 Inter-object and intra-object access control

We now discuss inter-object and intra-object access control issues. In our discussion, some of the issues are difficult to categorize cleanly as inter-object or intra-object, or both. For example, the inheritance of attributes is an inter-object issue since it involves at least two objects, but at the same time is also an intra-object issue for the object that is inheriting the attributes.

Variable granularity for access units

In structurally object-oriented database systems, the access control mechanisms would have to be flexible enough to support varying granularity of access units. For example, it may be desirable in some applications to have fine-grained access control at the level of the individual attributes of an object. But it may also be desirable to grant access/authorization to larger substructures (such as entire objects or composite objects) as a single unit.

We highlight briefly two contrasting approaches, one in the DAMOKLES database [4] and the other for the authorization model based on the ORION system [14]. We defer discussion on providing varying access granularity in composite objects to a later section. In DAMOKLES, every object (in the data modeling and object-oriented sense) is further broken down into smaller access units called protection objects (p-objects). These p-objects include the descriptive part D consisting of the object's attributes, the structural part S consisting of the components/composite objects, and version part V consisting of the object's versions. To treat all the attributes as a single unit, an authorization is granted on the D part.

The ORION approach utilizes the idea of implicit authorizations along an authorization object lattice. Thus to grant authorization on all attributes of an instance, we grant authorization on the authorization object type *Setof-Attr-Values* which leads to an implied authorization on authorization object *Attribute-Value*.

Class hierarchy and structure inheritance

Support for class hierarchies and inheritance in the object-oriented paradigm have an impact on how we approach access control issues. In particular, the following questions need to be addressed.

- What effect does allowing implicit authorizations in the class hierarchy have on the reusability of classes? on query processing?
- What should be the semantics for the inheritance of structure (attributes) among classes when subjects have differing authorizations on these classes?

Consider the alternate ways of handling implicit authorizations between a class and the instances of its subclass [14]. Our first option would be for a creator of a class to be given implicit authorizations on all instances of a subclasses derived (specialized) from the class. Queries rooted at the class and spanning lower subclasses can now be evaluated successfully as the required authorization can be obtained. However, this approach makes the classes too interdependent making their potential for reuse very low. The second option would be to prohibit implicit authorizations from classes to instances of derived subclasses. This would encourage the reusability of classes, but this benefit comes at the cost of query failures.

In [21] Spooner has raised some of the issues that arise when subjects have differing authorizations on classes, and inheritance is allowed. To restate an example in [21], consider a class B that is a subclass of another class A. If a subject is authorized to access B but not A, should the subject be allowed to see the inherited attributes from A when he accesses B? The approach taken in [7] would allow access to all the inherited attributes in B. The advantages and disadvantages of this approach need further analysis.

Access control in composite objects

Supporting composite objects requires us to address the following questions (among others).

- What is the implication of several components of the same object being owned by different subjects/users?
- How do we connect rights (authorizations) through composite object hierarchies?
- How do changes in the composite object structure (hierarchy) affect existing and future connections of authorizations?
- What are the semantics to handle conflicting authorizations at points in the composite object hierarchy?

- How do we handle transitive authorizations in composite object hierarchies?

In environments such as those supporting CAD/CAM, it is typical for designers to create and work on the design of individual components. The objects representing these components will thus be owned by different subjects. However when cooperative activity such as the exchange of partial designs or the assembling of entire composite objects are involved, a subject may have to obtain authorization from the individual owners of the composite objects.

In DAMOKLES, the approach to connecting authorizations along composite object hierarchies involves the use of *complex authorizations/rights*. A complex authorization differs from a simple one as follows. When applied to a root object in a composite object hierarchy, an authorization extends to all current as well as future composite objects connected to this root, so long as they have the same owner as the root.

The approach used in [14] to connect authorizations on composite objects is based on an authorization object lattice defined for composite objects. By making use of implicit authorizations along this lattice we get more flexibility than the hard-wired approach of DAMOKLES. However, the propagation of positive and negative authorizations along the composite object lattice may lead to conflicts. This happens for example if a previously granted authorization on a component object conflicts with a new authorization (implicit or explicit) that is received. As mentioned in [14] conflicts from negative authorizations also arise on objects that are components of more than one composite object. The access control mechanism must reject conflicting authorizations based on some consistent semantics.

Access control on versions

To provide access control on versions, the proposal in [14] once again utilizes an authorization object lattice with authorization objects such as *setof-generic-instances*, *setof-versions*, among others. The notion of implicit authorizations are again used on this lattice. In DAMOKLES [4], an authorization on the V part of an object obtains authorizations on all the versions of the object.

These approaches need to be refined to provide more selectivity on the versions belonging to a version hierarchy. For example, we may want to specify that a subject be authorized for the first/last three versions.

Meaningful interconnection controls among component objects

All our discussions above on access control in composite objects have assumed that component objects are linked in some meaningful way. Existing approaches place the burden on users/subjects for establishing meaningful interconnections and visibilities across component object boundaries. This might be a reasonable expectation in some environments. After all, we would expect a designer in a CAD environment to be knowledgeable enough not to mix and match the components of say, cars and trucks. However, when discrimination between components is not easy, we would like the access control mechanisms to help. For example we could have an access control list that governs how objects are interconnected. The access control list would place restrictions on the IS-PART-OF relationships that can be formed between component objects.

2.3 Behavioral and semantic based access control

2.3.1 Subject to object access control

As mentioned before, subject to object access control is concerned with the authorization of the initial point of contact with an object by a subject. In a behaviorally object-oriented database, this would involve authorization to invoke an initial method in a chain/tree of method invocations. Thus if a subject invoked an initial method m_1 which in turn invoked m_2, and m_2 in turn invoked m_3, we are concerned with how the subject gets authorization for m_1. Authorization for the other methods

m_2 and m_3 fall into the category of inter-object and intra-object access control and will be discussed subsequently.

We describe three approaches for subject to object access control that have been reported in the literature. In the first [15], associated with every object is an access control list (ACL) and an object owner. The owner of an object controls through the ACL the other principals that may invoke the operations (methods) defined for the object.

In the second approach [23], access groups based on user roles are defined with the help of a user role definition hierarchy (URDH). A node in the URDH represents an access group. Based on the access control requirements, the publicly accessible methods are assigned to nodes in the URDH. A subject/user belonging to a particular node in URDH will be allowed to invoke only those methods assigned to that node.

A third approach described in [5] uses the notion of interface objects. Every database object is associated with a collection of interface objects. An interface object supports only a subset of the total methods in a database object. Subjects are allowed to interact with the database objects only by invoking methods defined in their corresponding interface objects. In summary, an effective subject to object access control mechanism is built by defining a collection of interface objects for every database object, and by restricting subjects to one or more interfaces.

2.3.2 Inter-object access control

What are the implications of supporting behavior based access control, across object boundaries? In particular, it is important to recognize that objects are autonomous entities taking part in a distributed computation. Two important questions come to the forefront.

- How do we control visibility and interaction between objects, in terms of behavior?
- What are the semantics for propagating authorizations along method invocation chains and trees?

We consider answers to these questions in turn.

Inter-object method invocation and visibility

If we wish to control the visibility of a method m, then we should restrict the number of client methods than can invoke m. An approach suggested in [2] is to associate a set $< invokers >$ with the definition of every method (such as m). This set contains the names of methods and classes to which m is made visible to. In the case of a class, m is visible to all the methods in the class. Although this appears to be a good first step, several avenues need further investigation. Figure 2 illustrates a method m_0 which is visible to a class c_1 that is part of the invoker set of m_0. The locally defined method m_1 in c_1 can thus invoke m_0. Now consider the class c_2 which is a subclass of c_1 and has a locally defined method m_2 and by virtue of its position in the class hierarchy inherits m_1. Should an invocation of method m_1 locally from class c_1 be treated differently from an invocation of m_1 by m_2 from the subclass c_2, since in either case m_1 eventually invokes m_0? Should access control prevent method m_2 from invoking m_1 as long as m_1 can invoke m_0 but m_2 cannot? If it does not, we may as well make class c_2 part of the invoker set of m_0.

Authorization propagation through method invocation chains/trees

Here we are interested in coming up with consistent semantics as well as flexible mechanisms for propagating authorizations through method invocation chains/trees. The use of implicit, positive, negative, strong, and weak authorizations need to be studied. Once again conflicts from negative and positive rights may arise.

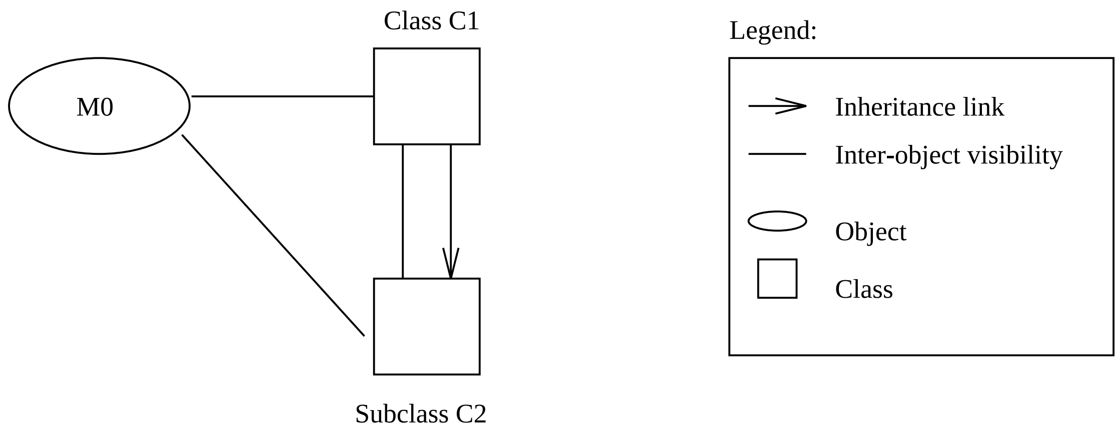

Figure 2: Visibility across methods and classes

2.3.3 Intra-object access control

The need for access control resurfaces even within the boundary of an object. It must be recognized that in object-oriented systems, access control and integrity mechanisms are closely linked. This is because methods modify the states of objects and we often enforce access control on method invocations. Integrity after all, is concerned with the improper modification of data.

Method to attribute visibility

For some applications, it may be desirable to allow only certain methods in the object (or class) to access a local attribute. For example, in a military application an attribute 'Target-coordinate' may be updated only by a method 'Approve-coordinate' which ensures that the new coordinates are not erroneous. An obvious way to achieve this would be for every attribute to maintain a list of methods that are allowed access (to the attribute).

Method to method visibility

Within an object boundary we may want to restrict the visibility of local methods to each other. Again an obvious way to accomplish this would be for every method to maintain some list. More complicated data structures are worth investigating, especially if the notion of implicit authorizations can be applied.

3 RESEARCH DIRECTIONS

In this section, we identify some issues that are beyond the capabilities in current proposals for discretionary access control in object-oriented databases. Addressing these issues would lead to access control models and mechanisms that accommodate the diverse security policies and controls of information management in organizations.

3.1 Transitive rights

The implications of the transitive propagation (taking and granting) of rights in composite-object hierarchies and method invocation chains warrants further investigation. Some of discussion of this for method invocation chains can be found in [15] (we do not discuss this work due to space constraints). Given a certain set of explicit authorizations, we would like to know if an access control request from a user will succeed. Also, when a change is made to certain authorizations, what is the overall effect on users?

3.2 Separation of duties and multiple approvals

The operational procedures in many organizations are designed to prevent fraud. Separation of duties and multiple approvals are well known principles to achieve this. For an activity to be authorized, it may need multiple approvals by separate individuals. Recently, Sandhu in [17, 18] has proposed *transactions control expressions* as an approach to implement these in computerized systems. It is based on a database activity model that utilizes the notions of transient and persistent objects. Transient objects include vouchers, purchase orders, sales slips, to name a few. These objects are transient in nature in the sense that they issue a finite set of operations and then leave the system (in a paper world this happens when a form is archived). These operations eventually affect persistent objects such as inventory databases, and bank accounts. The fundamental idea is to enforce controls primarily on the transient objects, and for transactions to be executed on persistent objects only as a side effect of executing transactions on transient objects.

As an example, consider a check processing application where a clerk has to prepare a check and assign an account, followed by three (separate) supervisors who have to approve the check and account, and finally the check to be issued by a different clerk. This can be represented by the following transaction control expressions:

prepare • clerk;
3: approve • supervisor;
issue • clerk;

The colon is a voting constraint specifying 3 votes from 3 different supervisors. Each expression consists of a *transaction* and a *role*. Separation of duties is achieved by requiring the users who execute different transactions in the transaction control expression be all distinct.

We are currently investigating adapting transaction control expressions for transient objects modeled as objects. We would also like to model transaction control expressions as typed classes and objects. In this way we will be able to apply specialized classes of these expressions to specialized classes of transient objects.

In concluding this discussion on separation of duties, we note that the authors in [19, 20] have alluded to the Clark-Wilson integrity model [3] and hence the need to support separation of duties. The proposal in [19] calls for an "AUTHORIZATIONS" object in the system to manage access control and separation of duties. Details on how these and other ideas can be implemented need further investigation.

3.3 Intra-object method control expressions

The scope of transaction control expressions cross object boundaries in that the transactions are public to all objects. It may be desirable in some applications, that private methods in an object (these methods are only accessible within an object boundary) be invoked in a certain sequence, and in addition for separation of duties to be enforced for intra-object accesses. We will be investigating the use of intra-object method control expressions for this purpose.

3.4 Content based authorization

The approaches surveyed in this paper generally do not address content dependent authorization issues in a clean way. This needs more investigation, especially in regard to behavioral approaches. It is not clear if an authorization should be defined in terms of the ability to invoke a certain method on an object. How do we access the object contents for content-based authorization if the method which can access the required attribute(s) cannot be invoked?

3.5 Authorization and temporal semantics

Consider the following access control/authorization requirements:

1. Let $(s_1, o_1, write)$ be true **as long as** $(s_2, o_1, read)$ is true;
2. Let $(s_1, o_1, read)$ be true **whenever** $(s_2, o_1, read)$ is false;
3. Let $(s_1, o_1, read)$ be true if only if subject s_2 has not written to object o_1 **so far**;
4. Grant authorization to subject s_1 for the last three versions created after June 12, 1993, of the versioned object o_1, and have not been updated **since** subject s_2 was authorized to write object o_1.

The models of discretionary access control that have been reported in the literature cannot accommodate the above constraints and requirements. The above requirements call for models that unify implicit authorizations, content-based access control, and the semantics of time.

3.6 Transaction based authorization

In all the work we have surveyed and discussed, the access control problem seeks an answer to the question: Is subject s allowed access type a on object o? An authorization was thus seen as a 3-tuple (s, o, a). This view of access control (and authorization) is heavily influenced by the subject-object paradigm of access control in general computer systems. We believe it is time to reexamine this view of access control in the context of databases. Why not specify authorizations in terms of transactions and objects. After all, in a strict sense it is transactions (and not subjects) that access and modify the database objects. A similar view is expressed by Clark and Wilson in [3] with transformation procedures being transactions (see rule E2), although their work needs to be adapted to object-oriented databases. We would of course expect the decision to authorize a transaction to depend on among others things, the identity and rights of the user who invokes the transaction.

Another promising research direction is the notion of an *authorization transaction* [22]. Such a transaction is one that is created for every regular database transaction, but is primarily concerned with the acquiring and management of all authorizations and access control information required to successfully commit the database transaction. In particular, this will give us the flexibility to incorporate failure semantics in the management of authorizations. Thus if a particular authorizations fails, we may be able to specify alternate authorizations to be requested.

As an illustration, consider a sales order processing system, where the processing of a sales request involves two transient objects, a purchase order and a sales order. In such an environment, we envision a nested model of authorization transactions. Every transient object is managed by an individual *authorization subtransaction* that executes the local transaction control expressions for the transient object. These subtransactions enforce separation of duties and other access control requirements on the transient object. A root authorization transaction manages these subtransactions, and further enforces separation of duties across transient objects.

4 SUMMARY AND CONCLUSIONS

In this paper we have provided a framework that breaks down discretionary access control issues into three categories: subject to user, inter-object, and intra-object. We identified some of the issues and current proposals in these categories for both structurally and behaviorally object-oriented database systems. While reasonable progress has been made, more work still needs to be done. We have identified some of the areas that warrant further research including authorizations based on separation of duties, multiple approvals, object contents, and temporal semantics. We have also argued for the advancement of transaction based authorization models. Such models would constitute a departure from the traditional subject-object paradigm of access control, and rely on transactions as a central abstraction for specifying access control in databases.

References

[1] F. Bancilhon, W. Kim, and H. F. Korth. A model of CAD transactions. *Proc. of the VLDB conference*, 1985, Stockholm, pp. 25-33.

[2] E. Bertino. Data hiding and security in object-oriented databases *Proc. of the IEEE Data Engineering Conference*, pp. 338–347.

[3] D.D. Clark and D.R. Wilson. A comparison of commercial and military security policies. *Proc. of the IEEE Symposium on Security and Privacy*, 1987.

[4] K.R. Dittrich, M. Hartig, and H. Pfefferle. Discretionary access control in structurally object-oriented database systems. *Database Security II, Status and Prospects*, C.E Landwehr (Editor), Elsevier Science Publishers B.V. (North-Holland)

[5] D.B. Faatz and D.L. Spooner. Discretionary access control in object-oriented engineering database systems. Database Security IV, Status and Prospects, S. Jajodia and C.E Landwehr (Editors), Elsevier Science Publishers B.V. (North-Holland)

[6] D. Fisherman. IRIS: An object-oriented database management system. *ACM Transactions on Office Information Systems*, 5(1):pp. 48–69, January 1987.

[7] E. Gudes, H. Song, and E.B. Fernandez. Evaluation of negative, predicate, and instance-based authorization in object-oriented databases. *Database Security IV, Status and Prospects*, S. Jajodia and C.E Landwehr (Editors), Elsevier Science Publishers B.V. (North-Holland)

[8] W. Kim, R. Lorie, D. McNabb, and W. Plouffe. A transaction mechanism for engineering design databases. *Proc. of the VLDB conference*, 1984, Singapore, pp. 355-362.

[9] W. Kim, E. Bertino, and J.F. Garza. Composite Objects Revisited. *Proc. of the ACM-SIGMOD Intl. Conference on the Management of Data*, Portland, Oregon, 1989.

[10] W. Kim et al. Features of the ORION object-oriented database system. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley Publ. Co., Inc., Reading, MA, 1989.

[11] R. Lorie and W. Plouffe. Complex objects and their use in design transactions. In *Proc. Databases for Engineering Applications*, Database Week, 1983 (ACM), May 1983, pp.115-121.

[12] D. Maier. Development of an object-oriented DBMS. In *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, ACM, New York, 1986, pp. 472-482.

[13] D. Maier. Why isn't there an object-oriented data model? In *Proc. of the 11th IFIP World computer conference*, San Francisco, CA, August-September, 1989, pp. 793-798.

[14] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM Trans. on Databse Systems*, Vol 16, No. 1, March 1991.

[15] J. Richardson, P. Schwarz, and L. Cabrera. CACL: Efficient fine-grained protection for objects. In *Proc. Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, ACM, New York, 1992, pp. 263–275.

[16] R.S. Sandhu. The NTree: A two dimension partial order for protection groups. *ACM Tran. on Computer Systems*, Vol. 6, No. 2, May 1988, pp. 197-222.

[17] R.S. Sandhu. Transaction control expressions for separation of duties. *Proc. of the Fourth Computer Security Applications Conference*, pp. 282–286, 1988.

[18] R.S. Sandhu. Separation of duties in computerized information systems. *Database Security IV, Status and Prospects*, S. Jajodia and C.E Landwehr (Editors), Elsevier Science Publishers B.V. (North-Holland)

[19] C.A. Schiller. Potential benefits from implementing the Clark-Wilson integrity model using an object-oriented approach. In *Proc. of the 15th National Computer Security Conference*, Baltimore, MD.

[20] J.M. Slack and E. A. Unger. Protected groups: An approach to integrity and secrecy in an object-oriented database. In *Proc. of the 15th National Computer Security Conference*, Baltimore, MD.

[21] D.L. Spooner The impact of inheritance on security in object-orinted databse systems. *Database Security II, Status and Prospects*, C.E Landwehr (Editor), Elsevier Science Publishers B.V. (North-Holland)

[22] R.K. Thomas and R.S. Sandhu. Task-based authorization: A paradigm for flexible and tailorable access control in distributed applications. Submitted for Publication.

[23] T.C. Ting, S.A. Demurjian, and M.Y. Hu. Requirements, capabilities, and functionalities of user-role based security for an object-oriented design model. *Database Security V, Status and Prospects*, C.E Landwehr and S. Jajodia (Editors), Elsevier Science Publishers B.V. (North-Holland)