A SECURITY MODEL FOR OBJECT-ORIENTED DATABASES

Eduardo B. Fernandez, Ehud Gudes [+], and Haiyan Song

Department of Electrical and Computer Engineering
Florida Atlantic University,
Boca Raton, Florida 33431

Abstract

The integration of object-oriented programming concepts with databases is one of the most significant advances in the evolution of database systems and several recent projects are developing object-oriented databases. Among the many issues brought along by this combination, one that is becoming important is the protection of information.

We develop here an authorization model for object-oriented databases. This model consists of a set of policies, a structure for authorization rules, and an algorithm to evaluate access requests against the authorization rules. For concreteness we use a specific database system to illustrate our model, but its concepts are applicable to a range of object-oriented databases.

1. Introduction

The integration of object-oriented programming concepts with databases is one of the most significant advances in the evolution of database systems and several recent projects are developing object-oriented databases. Among the many issues brought along by this combination, one that is becoming important is the protection of information.

Most of the current models for authorization in database systems were developed for relational databases [Fern81, Grif76]. Object-oriented databases have a much richer structure and those models are not adequate. The addition of semantic relationships makes the authorization problem even more complex. There is a need for new models. Until now very few authorization models for object-oriented databases have been proposed [Rabi87, Rabi88]. A related study [Tost88] considers a Smalltalk system without a database system.

We develop here an authorization model for object-oriented databases. This model consists of a set of policies, a structure for authorization rules, and an algorithm to evaluate access requests against the authorization rules. For concreteness we use a specific database system, which is now under development at the University of Florida. This system, intended for CAD/CAM applications, incorporates knowledge rules with a database of objects combined through an Object-Oriented Semantic Association Model (OSAM*) ([SuSY85],[SuSY88]). The database is composed of objects that include a collection of facts and a collection of relevant rules. An object binds knowledge rules to database facts.

All the knowledge manipulation operations can be used to express the rules. Some of these rules could be integrity or security rules, i.e., they could be the basis for a mechanism to enforce integrity or security.

Section 2 considers security policies and the structure of the authorization rules for such a system, while Section 3 discusses access requests validation. Section 4 develops an algorithm for access validation in the context of the OSAM* model, Section 5 shows some possible extensions, while the last section describes conclusions and directions for future work.

2. Security policies and authorization rules

A coherent set of policies is needed as a guideline for the design and use of a database system. The choice of policies for security is important because it can influence the flexibility, usability, and performance of the system [Fern81].

A fundamental choice is having an *open* or a *closed* system. In an open system everything is accessible unless forbidden, while in a closed system we have the inverse situation. Security requires closed systems while flexibility indicates open systems. In general, when security is an important objective, we should use a closed system. Another issue is the use of positive or negative authorization rules, For example, the system described in [Rabi87] uses positive and negative authorizations: a subject may be denied access to an object either because it has no authorization for it or because it has a negative authorization on it. The use of predicates for negative authorization present evaluation problems as well as possible contradictions in the security structure which may be hard to detect. However, in a system with a rich data model negative authorizations may be needed to express subtle access constraints. Negative authorization constraints are also required by the Orange book for security classes B3 and A1 [DODC85]

Another dimension is *ownership* versus *administration*. In the first case users own and administer their data; in the second case the information belongs to the enterprise, users are given access to it to perform their functions and special users (administrators) control the structure and the use of the information. Since a Data Base Management System is used to support an enterprise, administration is a more logical choice for this case. This view is also supported by recent work on enterprise policies [Moff88]. Additionally one may want to allow some (or all) user to define private databases.

Another relevant policy is the choice between *discretionary* and *multilevel* security. The National Computer Security Center recommends the use of multilevel systems for general computational environments [DODC85]. For the environment considered here a discretionary policy seems adequate.

[+] On leave from Ben Gurion University, Israel.

Multilevel security for object-oriented databases is under development at SRI and Honeywell.

Figure 1 illustrates a portion of a university database using the OSAM* model. (A few other concepts are introduced in the example of Section 3). Class Person (P) has *attributes* SSN (Social Security Number), Name (and maybe others). Classes Student (S) and Teacher (T) are subclasses of Person. The generic properties of Student and Teacher define Person through a *generalization association* (G in Figure 1). Attribute "Year"(year of graduation) is defined for Student and attribute "Course" for Teacher. Foreign Student (FS) is a subclass of Student. Attributes defined for subclasses reflect the fact that some features or properties only apply to specific subclasses, e.g., Visa is only meaningful for Foreign Students. Person, Student, Teacher, and Foreign Student are *object classes* (similar to Smalltalk *classes*). Their attributes correspond to Smalltalk *instance variables*. In addition to the generalization association, Figure 1 also shows an *aggregation association*, indicated by A, which defines a set of attributes for some class.

Class inheritance properties make Figure 1 to define the effective database of Figure 2, where it can be seen that all attributes of a class are inherited by its subclasses (the dotted attributes are the inherited attributes). It is clear now that access to some attribute of a class implies also access to the corresponding values in its subclasses. Note that these values are a subset of those of the superclass, i.e., SSN as an attribute of Student represents only the SSNs of Students, while the values of SSN as an attribute of Person represent SSNs of Students as well as of Teachers.

These considerations can be summarized in the following policies:

P_1 (inheritance policy) -- a user that has access to a class is allowed to have similar type of access in the corresponding subclasses to the attributes inherited from that class.

P_2 -- access to a complete class implies access to the attributes defined in that class as well as to attributes inherited from a higher class (but only to the class-relevant values of these attributes).

P_3 -- an attribute defined for a subclass is not accessible by accessing any of its superclasses.

Additional policies are necessary to consider predicates and multiple inheritance. A discussion of possibilities for discretionary and mandatory systems is given in [Spoo88].

In general, an authorization rule is a tuple (U, A, O, p, f), which defines that *subject* or user U has authorization of *type* A (access type) to those occurrences of *object* class O for which *predicate* p is true (note that the word object here is not used in the sense of object-oriented databases but it represents any named entity). User U can grant the access right (O,A) if the *copy flag* f is true. This model has been used to describe most of the authorization systems for relational databases. We use here a more specific version of these rules defined as below.

An authorization rule is a triple (U,A,AO) where U is a user or user group, A is an access type or set of access types, and AO is the set of attributes of the object to be accessed, i.e.,A O = {Oi.a_1, Oi.a_2,...}. A rule can either refer to AO as a whole or to its individual components. Attribute a_i must be defined for object Oi, or inherited by it.

For example, consider the graph of Figure 2. Assume the following authorization rules are defined:

R1: **(SA, R, S.SSN)** - - **The Student Advisor can read SSN** of students.

R2: (FSA, R, (FS.SSN, FS.visa)) - - The Foreign Student Advisor can read SSN and visa of foreign students

A Student Advisor (SA) could have access to SSNs of all students (P_1), but no access to their visas (P_3), a Foreign Student Advisor (FSA) could have access to visas but only to SSNs of Foreign Students (P_2).

We can also separate *user rights* defined by *user authorization rules* as described above from *administrative rights*, the ability to control the database access actions. Administrative rights are defined by *administrative rules* described by tuples (U, A, O, f). Examples of administrative access types are the rights to create and delete administrative groupings of data, to define user authorization rules, to revoke delegated rights, etc. This separation proved to be useful in a decentralized model [Wood79] and has been further elaborated in [Moff88].

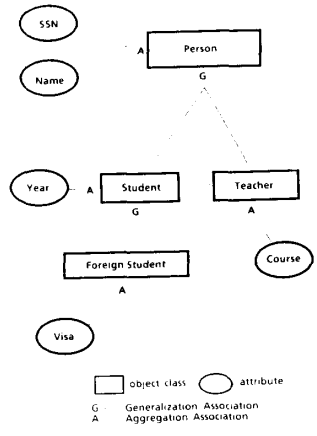

Figure 1. A university database

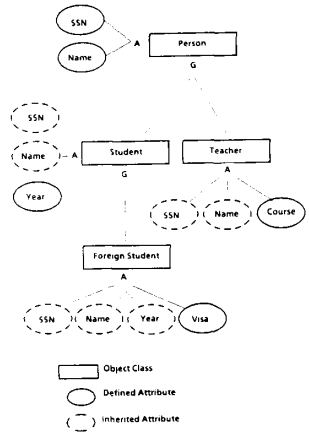

Figure 2. Effective structure of the database of Figure 1.

3. Validation of access requests

Access validation occurs by extracting a *data request* from a user query or from an executing program. This request has a structure (U',A', O'), where U' is the subject (user, process) making the request, O' is the requested object, and A' is the requested access type. This request is validated against the authorization rules to decide if the request should be granted totally or partially.

A *security context* is a set of object classes grouped together for security purposes. A security context may be equivalent to a conventional view or other partitions of the database schema. A security context defines a partially ordered set of object classes (in terms of the associations) which delimits the access for user queries, i.e., a data request is validated using the rules in a specific context. In Figure 3 we show a more global picture of the university database. Security Context SC1 is defined to include Classes Person, Student, Teacher, and Foreign Student, as well as their corresponding associations. Validation of a user's request associated with this security context will only consider classes and associations within SC1 (the portion in the dotted circle). SC1 is used as a boundary when constructing the Query Security Graph (defined later) for the user's query. Authorization rules are usually associated with a particular security context.

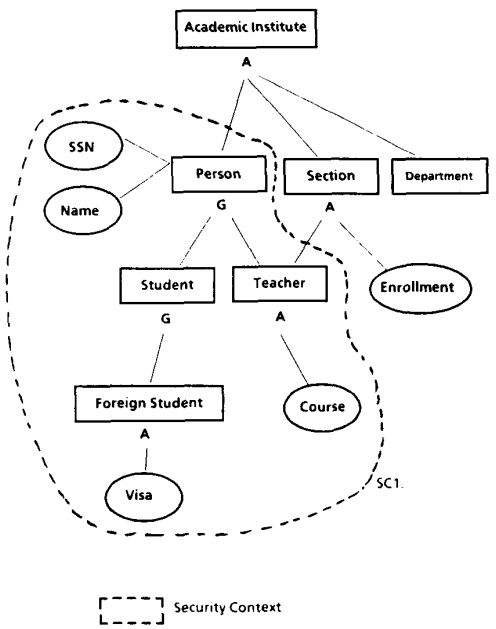

Figure 3. A security context in the university database.

The placement of authorization rules affects considerably the way of validating access requests. Authorization rules can be placed at special classes (e.g., a context root), at the class to which they refer (i.e., the class defined by the object part of the rule), or propagated throughout the hierarchy as discussed in [Fern75].

We consider the following placement rule.

Placement Rule 1: an access rule $(U,A,\{O_i.a_1, O_i.a_2,...\})$ can be placed only at node O_i.

Consider now rules R1 and R2 of Section 2. According to Placement rule 1, R1 must be placed at class Student. Similarly, R2 can only be placed at class Foreign Student.

A *query-graph* is the subgraph of the security context defined by the nodes that the query intends to access and their corresponding associations.

For the example above we define the following two queries, each of which is issued by SA and FSA.

Q1: read SSN for all students
Q2: read SSN and visa for all foreign students.

The corresponding query-graphs are shown in Figure 4.

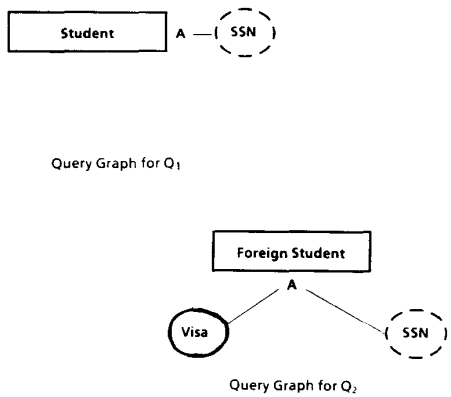

Figure 4. Query-graphs for example.

According to the policies of Section 2 we expect the following behavior as a result of the evaluation of the indicated requests*:

(SA,Q1) = (SA, Read, S.SSN) - - all SSNs can be read (since we do not deal with exceptions or negative authorizations, we do not exclude, for example, foreign students or other subgroups that may not be accessible to a student advisor....) (Policy P_1)

(SA,Q2) = (SA, Read, {FS.Visa, FS.SSN}) - -only SSNs of foreign students are to be read and not their visas. (Policy P_3)

(FSA,Q1) = (FSA, Read, S.SSN) - - only foreign student SSNs are to be read (Policy P_2)

(FSA,Q2) = (FSA, Read, {FS.Visa, FS.SSN}) - - both foreign student SSNs and visas are to be read. (Policy P_2)

We now define the concept of the *Query Security Graph*. For each node in the query graph we add all of its descendants (recursively) and all of its ancestors (recursively) until we reach the boundaries of the security context. The result is the query security graph. For example the query security graphs for queries Q1 and Q2 issued within SC1 are shown in Figure 5 (in this case this is the same graph for both queries).

*We denote a request as (U,Q), where U is a subject and Q a query. Their combination is equivalent to the set of components of the request model defined in Section 2. For clarity we show also the expanded request.

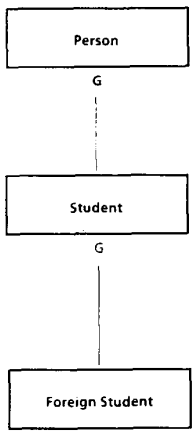

Figure 5. Query Security graphs for the example.

4. Access request evaluation algorithm

This algorithm assumes Placement Rule 1 and that the security graph is a tree (a more general algorithm, that can handle arbitrary class graphs, has been developed [Fern89].) For each node in the query graph the algorithm checks for authorization in the corresponding node in the query security graph. If it finds a rule authorizing all the requested access, it returns. Otherwise it looks for more general rules in the ancestor of the current node. If still not fully authorized, it looks for more restrictive rules in its descendant subtrees using, for example, a depth-first search procedure. The algorithm is based on the policies defined earlier.

For the algorithm we use the following additional definitions:

AT - - set of requested attributes
AT_yes - - the set of attributes already authorized at a given moment
QG is the query graph, SG is the query security graph
AUTH_UP and AUTH_DOWN are procedures to traverse the tree

The algorithm is composed of three major procedures: AUTH, AUTH_UP and AUTH_DOWN.

AUTH may traverse the entire query security graph for each node of the query graph (the examples below assume a single query node..). If the current query node contains access rules which completely authorize access, then the algorithm stops and sets the set AT to empty, and the set AT_yes to the set AT. If only partial authorization is found (ie., only some of the attributes in AT are authorized), this is indicated by substracting the authorized attributes from the set AT, and adding them to the set AT_yes. In this case, the security tree has to be traversed up and down looking for other authorizing rules. This is done by procedures AUTH_UP and AUTH_DOWN respectively.

AUTH_UP looks for rules authorizing attributes in the remaining set AT in all ancestors of the current node until the root of the SG. On the way up, it may not be possible to find access rules for all attributes because some of them are not known at higher level nodes, and according to placement rule 1 they cannot be authorized there. Therefore such attributes are temporarily eliminated from the set AT. If at any point the set AT becomes empty, the traversal up stops although this may not indicate full authorization because of the eliminated attributes.

If AUTH_UP did not get the full authorization for the set AT, then AUTH_DOWN must be invoked. AUTH_DOWN traverses the subtree rooted in the current node in a depth-first manner. AUTH_DOWN must consider restrictive authorization. This means giving authorization to a subclass even if access to the full class was requested (e.g., giving access to only FS in case access to S was required...) In order to maximize authorization, all children of the current node must be searched. However, once restrictive authorization was given to all attributes in the set AT, there is no need to search further down because no more authorization can be found.

After both procedures finish, either full authorization is given (set AT is empty), or partial authorization is given by the set AT_yes. The set AT_yes is then used by the query evaluation algorithm to restrict access to authorized attributes only.

```
AUTH(U',A',QG,SG)
for each node Ni in QG do
    /*AT is the set of attributes to be accessed at Ni, determined from QG, and
    O is the object class of Ni*/
    AT_yes := φ
    if there is a rule (U,A,AO) such that (U⊇U', A⊇A', AO⊇AT) then
     /*access authorized for all attributes*/
                                                    do
                                                    AT_yes = AT,
                                                    AT = φ; exit
                                                    end
    else if there is a rule (U,A,AT1) such that (U⊇U', A⊇A') and where AT2 = (AT1 ∩ AT)
    and AT2 ≠ φ then
    /*access authorized for some attributes*/
                                                    do
                                                    AT_yes = AT2
                                                    AT = AT - AT2
                                                    end
    /* now check ancestors */
    AUTH_UP(U',A',O',AT,AT_yes)
    if AT = φ then exit;
    /*now check descendants*/
    O: = left_son(O) in SG
    if O ≠ null then AUTH_DOWN(U',A',O,AT,AT_yes)
end do

end AUTH
```

```
AUTH_UP(U',A',O,AT,AT_yes)   /*go upwards in SG*/
    O- = father(O) in SG
    if O- = null then return
    /*if there is any attribute in AT which is not known at this node remove it from AT
    since there cannot be any rule referencing it at this node or above*/
    ATR = AT - attributes not known at O-
        /*ATR is used not to destroy the original AT*/
    if there is a rule (U,A,AO) such that U⊇U' and A⊇A', and AO⊇ATR then
                                do
                                AT_yes- = AT_yes∪ATR;
                                AT- = AT - ATR; return;
                                end
    else if there is a rule (U,A,AT1) such that U⊇U' and A⊇A', where AT2 = (AT1∩ATR)
    and AT2 ≠ φ and AT2 not included in AT_yes then
                                do
                                AT_yes- = AT_yes ∪ AT2
                                AT- = AT-AT2
                                ATR = ATR - AT2
                                end
        if ATR = φ then return; /*no need to continue going up*/
    else AUTH_UP(U',A',O-,AT,AT_yes)
end AUTH_UP

AUTH_DOWN(U',A',O,AT,AT_yes)   /*go downwards in SG*/
    if there is a rule (U,A,AT1) where U⊇U' and A⊇A' and AT2 = (AT1∩AT) and AT2 ≠ φ
    and AT2 not included in AT_yes
    then do
        AT_yes- = AT_yes ∪ O AT2
        /*in this case O is a more restrictive object and that must be noted in AT_yes*/
        if ATTR(AT_yes) = ATTR(AT) then NO_DOWN- = TRUE
    /*NO_DOWN is a flag and ATTR(O) give the set of attribute names of O*/
    /*no need to go more down but we still may have to look to the right for additional
    restricted objects*/
    O- = left_son (O)
    if O- ≠ null and not NO_DOWN then AUTH_DOWN(U',A',O-,AT,AT_yes)
    O- = right_brother(O)
    while O- ≠ null do
                AUTH_DOWN(U,A,O-,AT,AT_yes)
                O- = right_brother (O-)
                end
    /*all right brothers must be checked because they may add some specific subtypes*/
    return
end AUTH_DOWN
```

Now let us see how the algorithm works on the four cases above.

Case 1: (SA,Q1) = (SA, Read, S.SSN)
The algorithm finds rule R1 and therefore gives SA access to all SSNs.

Case 2: (SA,Q2) = (SA, Read, {FS.SSN, FS.Visa})
The algorithm first looks at node FS and cannot find any rules. Then it looks at node S. Now it updates AT by removing attribute "visa" since it is not known at this node and therefore no rule can reference it. Now it finds the rule (SA,R,S.SSN) and since obviously S includes FS it allows access to all SSNs for foreign students but not to their visas.

Case 3: (FSA,Q1) = (FSA, Read, S.SSN)
The algorithm first looks at S and cannot find any rules. Then it looks at P and cannot find any rules. Now it goes downwards and finds R2. It updates AT_yes with FS.SSN and therefore allows access to SSNs of foreign students only.

Case 4: (FSA,Q2) = (FSA, Read, {FS.SSN, FS.Visa})
The algorithm looks at FS, finds rule R2 and allows access to both SSNs and visas.

5. Extensions

We may want to use a more general placement rule:

Placement Rule 2: an access rule (U,A,{Oi.a1, Oi.a2...}) can be placed at any node AO such that:

(1) AO is Oi or one of its ancestors
(2) ai is known for object AO (i.e., defined or inherited from above).

If we use the second placement rule instead of the first, there are very few changes to the algorithm. The basic change is that when we update AT_yes we always have to update it with RO.AT and not with AO.AT where RO is the object in the rule and not the object we are currently working on. This is because RO may be more restrictive than AO.

The second placement rule may be useful if we want to concentrate our authorization rules in higher level nodes, and also most of our queries try to access higher level nodes. However, the distribution of queries versus access rules is not known in general.

Currently the algorithm repeats for each node of the query graph. So if the query graph contains two nodes and their corresponding security graphs overlap, we search portions of the same graph twice, which is redundant, the algorithm needs to be modified for this case.

Another alternative is to propagate the rules along the hierarchy as proposed in [Fern75]. In that case the algorithm becomes much more efficient at the expense of the storage space required for all the propagated rules.

6. Conclusions and Further Work

We have presented a model and a corresponding algorithm to validate authorization in object-oriented databases. Further work is needed for:

- Making the algorithm more efficient and general. In particular we are considering how to handle multiple query nodes using Placement Rule 2 [Fern89]
- Handle predicates. The rule with the complete format, as described in Section 2, includes predicates

These have been ignored at this stage

- Determine access rules for the other types of associations. OSAM* has several other types. These should be simpler than generalization but have to be studied systematically.
- Study administrative rights. These are the rights needed to manipulate the schema and control authorization. We have started considering this problem [Song89].
- Consider the use of ordered access types. As shown in [Fern75] this allows a simplification in the administration and evaluation of security as well as a reduction in storage requirements for these rules. One can also define partial orders for user classes [Fern75, Rabi87, Sand88].

Acknowledgments

This work has been supported by a grant from the University of Florida. Our students Jie Wu, Maria Petrie and Francis Yu provided valuable discussions. The reviewers' comment have had a significant impact on the style of this paper. Mrs. Joan Buttery typed innumerable versions of this manuscript.

References

[Alas88] A. M. Alashqur, S. Y. W. Su, H. Lam, M. S. Guo, and E. Barkmeyer, "OQL -- An object-oriented query language", Report, Database Systs. Res. and Dev. Center, U. of Florida, Gainesville, FL.

[Bane87] J. Banerjee, W. Kim, et al., "Semantics and implementation of schema evolution in object-oriented databases", Proc. 1987 SIGMODIntl. Conf. ACM, New York, 1987, 311-322.

[DODC85] Dept. of Defense Computer Security Center, *Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1985.

[Fern75] E. B. Fernandez, R. C. Summers, and T. Lang, "Definition and evaluation of access rules in data management systems". Proc. 1st Int. Conf. on Very Large Databases, Boston, 1975, 268-285.

[Fern81] E. B.Fernandez, R. C. Summers, C. Wood, "*Database security and integrity*, Addison-Wesley, Reading, Massachusetts, System Programming Series, February 1981.

[Fern89] E. B. Fernandez, E. Gudes, and H. Song, paper in preparation.

[Grif76]P. P. Griffiths and B. W. Wade, "An authorization mechanism for a relational database system", ACM TODS, 1, 3(September 1976), 242-253.

[LamH88] H. Lam, S. Y. W. Su, A. M. Alashqur, and M. S. Guo, "OSAM*: A departure from tuple-record oriented databases", Report, Database Systs. Res. and Dev. Center, U. of Florida, Gainesville, FL.

[Moff88] J. D. Moffett and M. S. Sloman, "The source of authority for commercial access control", Computer, 21, 2(February 1988), 59-69.

[Rabi87] F. Rabitti, D. Woelk, and W. Kim, "A model of authorization for object-oriented and semantic databases", MCC Tech. Rept., ACA-ST-327-87, October 1987.

[Rabi88] F. Rabitti, E. Bertino, W. Kim, and D. Woelk, "A model of authorization for next-generation database systems", MCC Tech. Rept. ACA-ST-395-88, November 1988.

[Sand88] R. S. Sandhu, "The NTree: A two dimension partial order for protection groups", ACM Trans. on Comp. Systs., 6, 2(May 1988), 196-220.

[Song89] H. Song and E. B. Fernandez, "Administrative authorization in object-oriented semantic databases", submitted for publication.

[Spoo88] "D. L. Spooner, "The impact of inheritance on security in object-oriented database systems", Report, Computer Science Department, Rensselaer Polytechnic Institute, November 1988.

[SuSY85] S. Y. W. Su, and L. Raschid, "Incorporating knowledge rules in a semantic data model: An approach to integrated knowledge management", Proc. of A.I. Applics. Conf., December 1985.

[SuSY86] S. Y. W. Su, "Modeling integrated manufacturing data with SAM*", Computer, January 1986, 34-49.

[SuSY88] S. Y. W. Su, V. Krishnamurthy, and H. Lam, "An object-oriented semantic association model (OSAM*)", in A.I. in industrial engineering and manufacturing: theoretical issues and applications, S. Kumara, R. Kashyap, and A. L. Soyster (eds.), AIIE, 1988.

[Tost88] R. S. Tosten, "Data security in an object-oriented environment such as Smalltalk-80", Proc. of the 1988 Int. Conf. on Computer Languages, 234-241.

[Wood79] C. Wood, and E. B. Fernandez, "Authorization in a decentralized database system", Proceedings of the 5th International Conference on Very Large Databases, Rio de Janeiro, 1979, 352-359.