6 SECURITY POLICIES IN REPLICATED AND AUTONOMOUS DATABASES

Ehud Gudes and Martin S. Olivier

Abstract: Autonomous object databases are becoming important in the Internet world of today and involve integration of several local databases. Such databases support local access for transactions and queries and local control over authorization of classes and objects. At the same time, these database objects are often replicated in various sites and are available for access by global queries and transactions. Such global access, which may involve a global query optimizer, is required to handle conflicts between the local authorizations of replicated objects, but give consistent results regardless of site dependent optimizations.

The paper uses previous models for object-based authorization, and extends them with policies to handle conflicts between local and global authorizations. It also discusses object migration and security administration. The problem of providing autonomy in a consistent way is discussed extensively.

6.1 INTRODUCTION

Autonomy is an important concept in today's fragmented but connected world. Organizations that support databases on the Web, often require autonomy in local access of their local data and in controlling access to it; at the same time they want to provide access to their data objects (or to copies of their objects) to a community of external users who may access them through a global system or a distributed query interface. In this environment conflicts may occur when different sites or security administrators place different access restrictions on such replicated data objects. The policies and algorithms to handle such conflicts in a consistent way, is the topic of this paper.

In principle, most of the ideas presented in this paper, hold in both relational and object-oriented database environments. However, since object databases seem to be more dynamic in nature and more local in character, we feel that the object database model is a more appropriate context to investigate this question. In this paper we will rely on some of our previous work on authorization in object-oriented (OO) databases such as [1]. In that paper the question of query modification in OO databases in the presence of authorization rules was investigated, and, in particular, the relationship between inheritance and authorization was discussed. An access evaluation algorithm was presented, and an administration model and policies were discussed. In particular, the inheritance policies of Negative vs. Positive, and Implicit vs. Explicit were handled. These ideas were generalized for Methods in [2]. The main idea we need from [1] is the policy to evaluate the access rights on an object, giving several authorization rules on objects (classes) above or below its inheritance hierarchy. In most of the paper we discuss hierarchies along one dimension — the inheritance sub-class/super-class hierarchy. Other hierarchies are discussed briefly at the end of the paper.

When an object database is distributed, it is often the case that its schema is also distributed and therefore its authorization information is distributed as well. Furthermore, it is common that different sites may have their own security rules and their local security administrator (SA) to define them. We know of three models that address the issue of security policies in autonomous databases. Argos [4] assumes a global policy which must be consistent with the local DBMS policies and is enforced as such. (The Argos policy is as follows: if global denies then deny, else if global permits then select one local DBMS; if permitted by the LDBMS fine, else try to grant it; if grant successful fine, else deny.) DOK [11] accepts the local policies and tries to integrate them into a global one. DOK, however, does not consider all the different cases as they are described in this paper. SPO [8] assumes that the owning site has the final say in how its data is accessed — the local policy is therefore paramount, with a relatively small federal policy adhered to by all sites. The current work differs from the others because it assumes a shared global (or federal) schema (even though the schema may only exist in partial forms at the various sites and be integrated by a query optimizer as and when required) and, because of local autonomy, conflicting access rules may be specified (or implied) for the same class at different sites. Note specifically that we assume that there may be multiple administrators where each is issuing her own authorization rules on the local copy of the database. As an example, consider a number of libraries with on-line reference material in their databases. These libraries have established a federated database to increase the available information to their members. Each library maintains an autonomous site. When a document is to be retrieved from this federated database, the optimizer may, in general, use criteria such as proximity, link availability and link speed to select the optimal sources from which to retrieve requested information. Access restrictions may, however, influence the selection of sources. The licence agreement that a particular library has with the supplier of a particular document, may restrict use of that document to members of the specific library. If a global user can get access to the document at another library the access should be permitted to the document at that library — access is only restricted at this library. Similarly, a library may find that the requirements of a particular user is excessively high and prohibit that user from accessing the information at its site; again accesses at other sites are, in principle, still permitted.

It is also possible to envisage another scenario, where library members pay a subscription fee to access specific information. In this case, when a member has not subscribed to some document, access to that document should be prevented at all sites. The paper will address these alternative cases systematically.

Also note that it is, in principle, possible to replicate information from one library at other libraries. Similarly, information may be moved from one site to another site. This may be done for efficiency or various other reasons.

Suppose now that a global query enters the system and the optimizer must decide where to get the data from. We want the retrieved results to be consistent *regardless* of the optimizer decisions! This means that we have to integrate the various authorization states of the various local databases before the optimization. This integration is not trivial and raises several policy issues such as: 1) Conflicts between implicit vs. explicit, and negative vs. positive rules in the different sites; and 2) how the decisions will be influenced by whether we have equal authority administrators vs. a 'master administrator' and 'local administrators' One possible solution is to provide several types of authorization rules, such as authorization rules only allowing or denying access to local copies of an object compared to authorization rules allowing or denying access to all copies of an object.

The existence of various authorization rules require consistent policies to handle conflicts between them. These policies are the main subject of this paper, and they are discussed in general in the next section, and in detail in a later section. Once the policy decisions are made, the algorithms to integrate two (or more) sets of authorization rules are straightforward, and they are discussed briefly in the section on algorithms. The problem of copying and migrating objects in this environment is subsequently discussed. Finally we briefly discuss a few other issues, including other hierarchies (granularities), security administration and information flow.

6.2 POLICY ISSUES

6.2.1 The model

First we present the model and its assumptions. The model we use is depicted in figure 6.1. We have an object-oriented database distributed over several sites. Each site has its own schema of classes, attributes and authorization rules. Each site has its own security administrator (SA). Sites maintain local

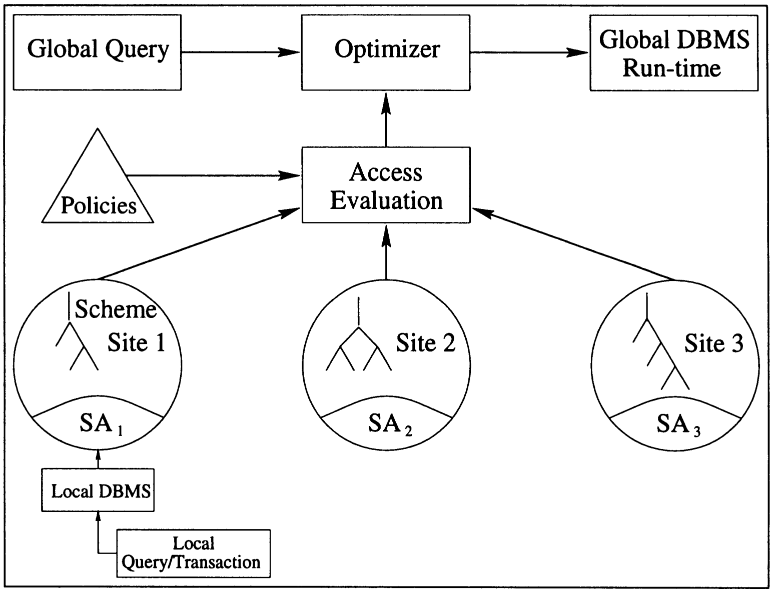

Figure 6.1 The Model for Global Queries and Autonomous Authorization Schemas

autonomy in that they expect all local queries and transactions to obey the local authorization rules. Schema parts and objects may be replicated in several sites. Global queries or transactions first enter a global optimizer which accumulates all authorization information and physical information and decides on the actual sites from where data will be fetched. Evaluation of authorization in a single site is done by the algorithm presented in [1] and its result is the authorization tree (AT) — the set of attributes and classes to which access is allowed.

Next we state our assumptions on the database and the authorization rules. We assume an inheritance-based object-oriented database, with the policies specified in [1]. In that paper we mainly discussed the class-generalization hierarchy, and assumed mostly that all rules are defined on classes and sub-classes. Similar rules may be used for the granularity hierarchy (i.e. class/attribute or class/object). To simplify matters they will be discussed separately later. The same will hold for rules which define predicates, i.e granules defined by predicates (see [6]). Also, in most of the paper we assume that authorization rules are defined on classes and attributes and not on individual objects (they are discussed later), but we often will use the concepts of classes and objects interchangeably.

Assigning access rights to a class has multiple possible interpretations [7]. In this paper we assume that a class is merely labelled to control access to instances of the class. In particular do we assume that access rights may be added or revoked at any level of the class hierarchy and are inherited from a level where they are defined down to — but not including — the level where they are modified. (Note that this interpretation presents another valid alternative to those discussed in [7] because axiom 4 of [7] does not apply here.)

We assume that in each site (or set of sites) there is a hierarchy of object-classes with its own security administrator who can specify authorization rules. The schemas in two different sites may not be the same, although we assume for simplicity that one schema extends the other (up or down), but if two classes appear in both schemas, then the path between them also appears in both schemas.

As stated above, we assume complete autonomy on behalf of the administrators of the autonomous objects (actually, autonomous object hierarchy), The security administrators (SAs) act independently and specify various authorization rules on the objects. The problem is the combination of these rules, since we assume *sharing* of some (large) parts of the schema.

6.2.2 The policies

Now we state several of the principles we want this autonomous system to obey:

P1) When a local security administrator makes a decision about her local data, such as denying local access to person P, she should not be concerned that person P can get that access from some other site. On the other-hand, the run-time should be consistent: if the optimizer decides to access the data from her site, it should not get that access! Therefore, we should always *guide* the optimizer where to get the data from (i.e pass the positive access site information).[1]

P2) When a security administrator has the right to define authorization rules which may apply to *more* than one site, we expect the system to follow through and apply them correctly.

P3) One *never* gets more access by accessing only a local site than by issuing a global query! This means that by issuing a global query we can only get more data permitted — not less than accessing the local site only (e.g by a local query or application). This will considerably influence the policy decisions below. We call this last policy the principle of *maximum access.*[2]

The application of the policies above depends on the type of *organization* one has. We distinguish between two different types of organizations:

[1]As a corollary of P1, if a global access rule restricts local access, it may be inconvenient for the local system to consult other sites for local access. Therefore, for *performance* and *reliability* reasons such a restrictive rule should be propagated to all local sites. See also the section on *Administration issues.*

[2]One positive result of this principle is that when a site fails, one never gets more access than without that failure.

1. **EQUAL (EQ)** — all sites are equal, each SA has the same power.
2. **Master-Slave (MS)** — The master's SA has more power than the slaves' SAs. We do not precisely define the exact 'division' of power; some possibilities are discussed below.

All access rules may be

1. Intended globally; or
2. A distinction may be made between global and local rules:
   (a) **Local** — rules (positive or negative) which apply to the local site only, called below PL or NL (positive or negative *local*).
   (b) **Global** — Rules which apply in all relevant sites, called below PG or NG (positive or negative *global*).

Usually, all four combinations of organizations and rules types are possible, although some restrictions may apply in some cases. Next we detail how the access evaluation is done in each of the above cases, while enforcing the general policies outlined above.

6.3 ACCESS RULES — DETAILED POLICIES

Suppose that some subject s wants to access some object of class X. Let $A(X)$ be the set of access values that have been specified for s in the concerned mode m (such as reading, writing, etc). Denote each such access rule by a_i^d, where i indicates the site where the rule has been specified and d indicates the distance in the lattice (or class hierarchy) from X to the (higher) class where the rule has been specified. If the rule has been explicitly specified for X, then $d = 0$, etc.

Each a is P or N, indicating a positive rule (allowing access) or a negative rule (denying access), respectively. If local policies are supported, a may be PG or PL — for a global or local positive access specification to X — or NG or NL — for a negative global or local access specification to X, respectively.

It is assumed that every access rule is propagated from the class where it is specified, to all classes lower in the lattice (with d incremented for every level that the rule is propagated down). At any given class a variety of rules therefore exist that need to be combined to determine the effective rule to be used. The following two principles are usually used when combining such rules (same as in [1]):

1. The shorter the distance that a rule has been propagated, the more specific it is considered to be; rules propagated over a shorter distance therefore take precedence over rules that have been propagated over a longer distance; and
2. If rules propagated over an equal distance conflict, access is denied.

Let $\lfloor a_i^d, a_j^e \rfloor$ be the access value (a_i^d or a_j^e) propagated the shorter distance:

$$\lfloor a_i^d, a_j^e \rfloor \quad = \quad \text{if } d < e \text{ then } a_i^d$$

```
if $d > e$ then $a_j^e$
if $d = e$ then
  if $i \leq j$ then $a_i^d$
  else $a_j^e$
```

When $d = e$ either of the two values may in fact be used — they have been propagated an equal distance. For formal manipulation, we find it useful to select a unique value in this case — hence the last three lines of the definition. Note that, where $d = e$ one authorization may in principle be negative while another may be positive; however this situation will not occur where we use this operation below.

To express these two principles formally, the function *min* is defined to determine the effective access rule when considering two explicit or propagated access rules:

```
$min(a_i^d, a_j^e)$ = if $d \neq e$ then $\lfloor a_i^d, a_j^e \rfloor$
                      if $d = e$ then
                        if $a_i^d = N$ then $N_i^d$
                        else if $a_j^e = N$ then $N_j^e$
                        else $P_i^d$
```

The conditional expression *if* $a_i^d = N$ (without superscripts or subscripts following N) means if a_i^d is equal to *any* N (negative) value. Whenever superscripts or subscripts are omitted in such a conditional expression, it should be interpreted in this way.

If local policies are supported, the shortest distance principle does not necessarily apply: If a site denies subject s local access to some subtree for which global access has been granted higher in the lattice, the local denial should not override the global positive authorization — as long as there is some site that contains the subtree where a negative rule has not been specified. The same occurs when two local rules are combined, if none is more specific than the other, then the positive rule wins, unless both rules are in the same site. To determine the effective access rule, where the two constituent rules are both local rules:

```
$local(a_i^d, a_j^e)$ = if $(i = j) \vee (a_i^d = NL \wedge a_j^e = NL)$
                          $\vee (a_i^d = PL \wedge a_j^e = PL)$ then $\lfloor a_i^d, a_j^e \rfloor$
                        else if $a_i^d = PL$ then $a_i^d$
                        else $a_j^e$
```

The combination of a local and a global policy is problematic in one sense: The global policy will override the local policy, but if all local policies deny access, global access has effectively been denied. Consider the following, where we assume that the first argument is global and the second one local:

```
$globloc(a_i^d, a_j^e)$ = if $a_i^d = NG$ then $a_i^d$
```

$$\begin{array}{l}\text{else if } i = j \wedge a_j^e = NL \text{ then } PG_\infty^d \\ \text{else } a_i^d\end{array}$$

The logic behind this definition is that a global rule always takes precedence over local rules. The only 'exception' occurs if the site that granted a positive global authorization denies local access to the data: the positive global authorization still exists, but no site where it may be accessed by user s may remain. For that reason, in such a case the combination results in a positive rule where the site is specified as ∞, meaning that no positive site is known (at this stage).

Let $+_n(a_i^d, a_j^e)$ combine two access rules a_i^d and a_j^e. The definition of $+_n$ is considered below for the various cases. Consider access rule combination for *read* access in the following four cases.

1. EQUAL, no local policies If the two rules are not of the same level, the more specific rule takes precedence, otherwise the negative rule takes precedence. Therefore, let $+_1(a_i^d, a_j^e) = min(a_i^d, a_j^e)$. The logic behind this choice to combine rules has been discussed when *min* was defined. That is, when all rules are global, the simple 'min' policy which is commonly used in a centralized database is used.

2. EQUAL, with local policies If both rules are global or at the same site then this case should be treated similar to the previous case. Otherwise, if there is a global negative rule, it takes precedence. If no such negative rule exists and a positive rule does exist, it takes precedence. These are the consequences of precedence of global rules over local rules and the principle of maximum access.

$$\begin{array}{rcl} +_2(a_i^d, a_j^e) & = & \text{if } i = j \vee (a_i^d = \mathit{global} \wedge a_j^e = \mathit{global}) \text{ then } min(a_i^d, a_j^e) \\ & & \text{else if } a_i^d = \mathit{local} \wedge a_j^e = \mathit{local} \text{ then } \mathit{local}(a_i^d, a_j^e) \\ & & \text{else if } a_i^d = \mathit{global} \text{ then } \mathit{globloc}(a_i^d, a_j^e) \\ & & \text{else } \mathit{globloc}(a_j^e, a_i^d) \end{array}$$

3. Master-slave, without local policies This case should essentially be handled similar to the case for equals without local policies.

$$+_3(a_i^d, a_j^e) = +_1(a_i^d, a_j^e)$$

The above case means that there is a distinction between a master and a slave, but all rules are in a sense global. A common 'division' of power will be that only Master SAs can define authorization rules. This should not be seen as self contradicting, since even if we restrict the security administrators to be on the Master site only, there is still the case of two Master sites which is handled by the above rule.

4. Master-slave, with local policies Again the Master-slave has an impact on the administration of security. For example, we may want to restrict that global rules can only be issued by master administrators, while slave SAs can only issue local rules. (See also the discussion of administration in below). However, we may want to restrict that only one site can issue global rules.

This case is therefore handled similar to the case for equals with local policies, but Master-slave rules are treated similar to Global/local rules:

$$\begin{aligned} +_4(a_i^d, a_j^e) \quad = \quad & \text{if } i \neq j \wedge a_i^d = \mathit{global} \wedge a_j^e = \mathit{global} \text{ then } \mathit{error} \\ & \text{else } a_i^d +_2 a_j^e \end{aligned}$$

Note that if both a master and a slave issue local rules, the Master has no advantage over the slave when combining these rules.

Theorem 1 *$+_n$ is commutative and associative for each n.*

Proof A detailed proof is outside the scope of this paper. Intuitively, one can argue that in each of the above operators a single selection is done from a pair (X,Y), where the selection is dependent only on the values of X and Y and not on their order. Thus commutativity is trivial. Associativity is assured by the transitivity of the 'selection' process. If X was selected over Y, and Y was selected over Z, then X will be selected over Z, whether it is first combined with Y, or whether Y is first combined with Z. □

Since $+_n$ is commutative and associative, it is possible to define operators to combine an arbitrary number of rules. Let $\sum_n$ do this combination for each of the four cases. Where n is not significant, it will be omitted.

Because of the operator commutativity and associativity, the order in which the various access rules will be combined within the access evaluation algorithm, is immaterial. Likewise, if rules are at different levels of the hierarchy, the order of their combination is also immaterial. This is specified formally in the next theorem.

Denote the effective access rule that applies at a class X for subject s by $e(X)$, then

$$e(X) = \sum_{a_i^d \in A(X)} a_i^d$$

Denote the fact that Y is immediately above X in the lattice by $Y \succ X$. Let $D(X)$ be the set of all the access rules that have been directly specified for X. The significance of the next theorem is that the effective access rights to some class X may be computed from its immediate ancestors and its direct access specifications.

Theorem 2

$$e(X) = \sum_{Y \succ X} e(Y) + \sum_{a_i^d \in D(X)} a_i^d$$

Proof Let $\succ^*$ be the transitive closure of $\succ$. (The transitive closure includes the case where the operands are equal; i.e. $\forall X, X \succ^* X$.) Then, from the definition of $A(X)$:

$$A(X) = \sum_{a_i^d \in D(Y), Y \succ^* X} a_i^d$$

If X is the root node in the lattice, the theorem follows trivially.

To use induction, consider some node X which is not the root in the lattice. Assume that the theorem holds for all nodes above X in the lattice. Then

$$\sum_{Y \succ X} e(Y) + \sum_{a_i^d \in D(X)} a_i^d = \sum_{a_i^d \in D(Z), Z \succ^* Y, Y \succ X} a_i^d + \sum_{a_i^d \in D(X)} a_i^d = \sum_{a_i^d \in D(Y), Y \succ^* X} a_i^d$$

which proves the theorem. □

The correctness of the algorithms given below, depends on these theorems.

6.4 ALGORITHMS

The algorithm to merge two authorization trees according to the above policies is quite straightforward; therefore, only a general outline is given here.

First, we assume that in both trees the rules were propagated to all nodes of the trees. Theorem 2 allows us to do this without considering all rules for every node, but only the nodes immediately superior to the node under consideration. It is therefore possible to propagate rules from the top of the lattice to the bottom efficiently.

Once this propagation is done separately for each AT, then the merge algorithm scans the two trees in parallel. There may be two cases:

1. The trees have exactly the same structure — for each node combine the rules according to the policies (see theorem 1).
2. One tree has parts which are not in the other. According to our assumptions these can be of three types: Disconnected parts, higher parts, lower parts.
   (a) For disconnected parts just union the authorization rules.
   (b) For higher parts union the rules and propagate their results down.
   (c) For lower parts, propagate from the shorter tree to the lower parts.

An important result that the algorithm must record and return to the optimizer, is the identification of the site of the 'remaining' rule. This means for example, that if one site denies local access to an object and another site allows local access, then according to our policy, access will be allowed. Now one can argue that the optimizer should be trusted to select the site using physical optimization decisions unrelated to security. On the other hand one may be 'paranoid' and argue that, for reasons of autonomy, the denying site does not like to yield that denied access, and would like to leave the 'responsibility' for a positive access to the other site. In this case the security related site information is important, and should be transferred by the algorithm to the optimizer. Also remember that the local access may have been denied for load or other reasons as described in the introduction.

6.5 COPYING AND MOVING OBJECTS

The existence of local autonomy systems as discussed above raises interesting issues with regards to copying or moving objects. Again, we want to maintain the autonomy of local users to copy local objects, while keeping the security of the system consistent. Note, that when we talk about copying objects, we usually mean copy classes and attributes; the special case of object instances is discussed below. Note also that copying an object (or attribute) may require copying of its class (and even superclasses). We are not specifying here the semantics of copy or move operations, but worry mostly about which access rights should be copied along with these classes.

6.5.1 Copying objects

Copying objects within the same site is handled by local policies and is of no interest here. When copying objects from one site to a second site, it is obviously assumed that the copier has read access to the object and create access in the other site. The main issue is whether or not the authorization rules in the first site are copied with the object. Below we discuss this issue for the four cases we described above.

1. EQUAL, no local policies In the EQUAL one option environment case, the copier may be one of two classes of users:

1. The SA herself, in which case she can grant any access rights she likes and the copied rights are of no concern.
2. The user himself. Here, we believe the access rules governing this object should be copied (plus write and delete rights for the user).

Remember that the principle of maximum access restricts queries to the local site from accessing more information than a global query would. This means that global restrictions (Negative rules) should always be copied along when the object is copied.

Another option is not to copy any rights except for the copier's own rights. This may not achieve the performance results desired for other users, but will be simpler to implement.

Note that copying should not always be permitted. We do not address policy issues in this regard in the current paper.

2. EQUAL, with local policies Global rules should be copied as in the previous case. Whether local rules should be copied may depend on the circumstances. If a local negative rule is in force at the source site for performance reasons, load at the destination site may determine its desirability there. If, on the other hand, a local rule expresses the local site's opinion on whether a given subject should be allowed to access the information, the rule is better copied — otherwise copying information may affect the effective access rules for the object. While a change in effective access rules may be acceptable in the former case, it will almost always be unacceptable in the latter case.

Since all SAs have the same power, a local SA may change or delete a global rule without any problems.

3,4. Master-Slave cases These cases are similar to the respective EQUAL cases. The only difference is that a Slave SA cannot change or delete global rules.

6.5.2 Moving objects

Because of the inheritance structure, moving an object may cause many problems, since it also removes the access rules associated with the object which will impact lower objects access. There are several options:

1. Before moving the object, propagate the access rules below. However, since the distance that a rule has been propagated influences the effective access rule, this downward propagation may have to consider the existence of other rules. Alternatively, explicit rules may be given a 'distance' attribute so that it is possible to have an explicit rule as if it has been propagated some distance already. We do not consider this aspect further in the current paper.

2. Allow only SAs to move an object (actually only the SA from the source site, provided she has a create right on the target site). Then the SA may manually specify access rules for any objects below the moved object.

3. Allow copying of "root" objects only, and in that case move the entire schema and access rights associated with that root class.

The rest may be treated as in copy.

However, again for the Master-Slave situation a possible problem exists. Moving an object from a Master causes a problem if it is required that a Master should have copies of all objects that it has been designated master for.

6.6 OTHER ISSUES

6.6.1 Other Granularities

So far we have looked at the class hierarchy granularity. The policies for other granularities seem to be similar. The main exception may be the Class/Instance relationship (see also [5]). Generally, it should behave like a typical is-a relationship; for example, if in a single site there is a rule which denies access to a class, and another rule which grants access to an instance of a class, the 'shorter' second rule obviously wins. Even if the rules exist on separate sites the principle of *maximum access* determines the effective access rules. So most of the policies above apply. One case that may be different is that of a Master-Slave for instance-based protection. A class-based protection is often defined for an entire schema; this may not be effective when instances may also be protected individually: Since an instance is a physical object, e.g. a user's bank account, there may be a site where that physical object usually resides, i.e. a 'home' site. Now these 'home' sites may be different for different instances! Therefore, different instances will have different 'masters'. It seems that it makes sense to define only a single home site per instance, and in that case the policies for home/no-home can be similar to the Master-Slave policies. The problem, however, is that generally such policies cannot be applied at compile time since, at compile time, we do not know what specific object will be required (if a query selects some instance of some identified class).

A similar problem exists for predicate-based access rules, and it was also pointed out in [6] that such checks are needed. How do we expect to integrate such checks? Currently, run-time checks are usually used by Mandatory systems or by Information-flow systems [9]. In SQL-based systems, instance-based rules are usually translated into views, and there will be as many views as instance-based rules. This, seems to be too much overhead in object-based systems, where instance-based rules may be more common.

What we suggest to do in this case is as follows: First, a note on the existence of such a conflict (between class and object) should be known to the optimizer and the optimizer should gather all these types of rules in a packet called the *class-instance packet*. The optimizer then can decide on the site from which to retrieve the instances without regard to this packet, but this packet must be sent by the run-time query evaluation to the chosen site, and that site must check the class-instance packet and apply the required policies, before transferring the results back.

6.6.2 Administration issues

The administration of security in object-oriented databases was discussed in [1]. Several issues were discussed there, including the impact of schema changes on authorization rules, the delegation and revocation of security contexts, and the relationship between inheritance and rule maintenance (addition and deletion of rules). Basically, all the policies from [1] can be used also in the autonomous objects case. One may, though, restrict the operation of some SAs in some cases. For example, as mentioned in earlier, in a Master-Slave environment slave SAs should not be allowed to define (or remove) global rules.

A new problem arises when a security administrator (SA) defines a new authorization rule or deletes an authorization rule which affects replicated objects (classes) in other sites. Our basic assumption that at different sites separate SAs may add and delete authorization rules still holds, since these rules will be merged by the access-evaluation algorithm as was discussed earlier. The only problem is with global negative rules. Whenever a new negative authorization rule is defined, except for the Local case, it should be propagated to all sites with copies of this object. This is essential, since otherwise we could violate the policy that a local access can never have more rights than a global one!

So the procedure for addition of rules is the following: If it is not a global negative rule (or a negative Master rule) then add the rule to the local schema, otherwise propagate the rule to all relevant schemas[3]. Similar problems exist when an authorization rule is removed. A local SA should not be allowed to remove a global negative negative rule. Such a rule should be removed from all relevant schemas.

A related issue is which SA can do what? It may be the case that we have a hierarchy of SAs, where some can only define local rules on local schemas, others can define or remove global rules. A similar hierarchy can exist between master and local SAs. We think that such hierarchy can be handled by a simple role-based model (see [17]).

6.6.3 Information Flow

Information flow in object-oriented databases was discussed in [9], using a run-time approach, and in [3] using a compile-time approach. Both approaches can be extended to include the policies above. Basically, one has to construct the RACL and WACL lists using the policies above and update them when rules are added or deleted. Similarly, one has to construct the UAT and CUAT data structures of [3].

One problem that may arise is the following: Since local access is always more restrictive than global access, it may be that by local analysis a transaction may not cause information flow, while by global analysis it will. This creates a problem in local transactions since it restricts their freedom considerably, and, in particular, restricts the flexibility of local users developing local transactions. In such an environment it may make sense to propagate all rules to all sites. Only then is local flow analysis guaranteed to be correct. Further research is needed to see if there is a way around this undesirable propagation.

6.7 SUMMARY

In this paper a model for autonomous objects security was presented. The main problem in such a model is that each site may have its own security administrator who maintains its own authorization rules. A local site requires that local transactions behave consistently with local rules. The question that arises is what happens to global transactions and queries which may access copies of the local objects in various sites, and how conflicting authorization rules are handled. The paper provided a set of policies to handle the various cases, all are based on a simple principle, the principle of *maximum access*: by globally accessing the database you always get at least all the information you can get from a local access. Other issues such as object migration, rule administration and information flow were also discussed.

In conclusion, autonomy is an important concept in today's distributed but connected database world. The issue of supporting this autonomy in a consistent and least restrictive way has been discussed in length.

[3]Since propagation may take time, a two-phase commit protocol may be used to ensure consistency of the security specifications.

References

[1] EB Fernandez, E Gudes, H Song, "A Model for Evaluation and Administration of Security in Object-Oriented Databases," *IEEE Trans. on Knowledge and Data Engineering*, **6**, 2, April 1994, 275–292

[2] N Gal-Oz, E Gudes and EB Fernandez, "A Model of Methods Access Authorization in Object-Oriented Databases," *Proc. of the 19th VLDB Conference*, Dublin, Ireland, 1993

[3] M Gendler-Fishman and E Gudes, "Compile-time flow analysis of transactions and methods in object-oriented databases," in TY Lin, S Qian and R Sandhu (eds), *Database Security XI, Status and prospects*, Chapman & Hall, 1997, 95–109

[4] D Jonscher and KR Dittrich, "Argos — A Configurable Access Control System for Interoperable Environments," in DL Spooner, SA Demurjian and JE Dobson (eds), *Database Security IX: Status and Prospects*, Chapman & Hall, 1996, 43–60

[5] W Kim, *Introduction to Object-Oriented Databases*, MIT Press, 1990

[6] M Larrondo-Petrie, E Gudes, H Song, EB Fernandez, "Security Policies in object-oriented databases," in DL Spooner and CE Landwehr (eds), *Database Security IV: Status and Prospectus*, Elsevier Science Publishers, 1990, 257–268

[7] MS Olivier and SH von Solms, "A Taxonomy for Secure Object-oriented Databases", *ACM Transactions on Database Systems*, **19**, 1 (1994) 3–46

[8] MS Olivier, "Self-protecting Objects in a Secure Federated Database", in DL Spooner, SA Demurjian and JE Dobson (eds), *Database Security IX: Status and Prospects*, Chapman & Hall, 1996, 27–42

[9] P Samarati, E Bertino, A Ciampichetti and S Jajodia, "Information Flow Control in Object-Oriented Systems," *IEEE Trans. on Knowledge and Data Engineering*, **9**, 4, August 1997, 524–538

[10] R Sandhu, E Coyne, H Feinstein and C Youman, "Role-Based Access Control Models," *IEEE Computer*, **29**, 2, February 1996

[11] Z Tari and G Fernandez, "Security enforcement in the DOK federated database system," in P Samarati and R Sandhu (eds), *Database Security X, Status and prospects*, Chapman & Hall, 1997, 3–42