Resolving Conflicts in Authorization Delegations

Chun Ruan and Vijay Varadharajan

School of Computing and Information Technology
University of W.Sydney
Kingswood, NSW 2747, Australia
{chun,vijay}@cit.uws.edu.au

Abstract. In this paper, we first discuss some drawbacks of the existing conflict authorization resolution methods when access rights are delegated, and then propose a flexible authorization model to deal with the conflict resolution problem with delegation. In our model, conflicts are classified into comparable and incomparable ones. With comparable conflicts, the conflicts come from the grantors that have grant connectivity relationship with each other, and the predecessor's authorizations will always take precedence over the successor's. In this way, the access rights can be delegated but the delegation can still be controlled. With incomparable conflicts, the conflicts come from the grantors that do not have grant connectivity relationship with each other. Multiple resolution policies are provided so that users can select the specific one that best suits their requirements. In addition, the overridden authorizations are still preserved in the system and they can be reactivated when other related authorizations are revoked or the policy for resolving conflicts is changed. We give a formal description of our model and describe in detail the algorithms to implement the model. Our model is represented using labelled digraphs, which provides a formal basis for proving the semantic correctness of our model.

1 Introduction

In an access model with both positive and negative authorizations, conflicting situations can arise. If a subject (user) is granted both positive and negative authorizations on the same object, then we say that these two authorizations *conflict* with each other with respect to this subject. For instance, when a subject s is granted both "read" and "not read" rights on a file F from different subjects (grantors), then these two authorizations are in conflict with each other. Solving authorization conflicts in security policy specification is an important design issue in an access control model. Several previous research work have looked at this issue of conflict resolution policy, though in practice the realisation of such schemes has lagged behind the need.

Currently the proposed conflict resolution policies can be summarised as follows(see references [2,3,5,6,7,8]):

Negative (Positive)-takes-precedence: If a conflict occurs on some subject, the negative (positive) authorizations will take precedence over positive (negative) ones.

Strong-and-Weak: Authorizations are classified into two types, strong and weak. The strong authorizations will always override the weak ones when conflict occurs. Conflicts between strong authorizations are not permitted. When conflict occurs between weak authorizations, the negative ones will take precedence.

More specific-take-precedence: The authorization granted to a subject will take precedence over the authorizations granted to a group to which the subject belongs when conflict occurs.

Time-take-precedence: The new authorization will take precedence over the old one.

In a flexible access control model, it is necessary to have delegation of access rights between subjects especially in a large distributed system. Furthermore, there could be multiple administrators in such a distributed authorization model. Most of the conflict resolution policies are limited when delegation requirements are taken into consideration. For instance, they suffer from the following common problem: when a subject s_1 delegates some privilege to another subject s_2, s_1 can lose control of the delegated privilege with respect to further delegations. This situation can lead to unexpected situations; for instance, s_2 may then give back to s_1 a negative authorization for the same access privilege. For example, in a company, suppose the chairman creates a file and then delegates its "read" right to each member of the executive committee. Let us assume that each member of the executive committee further delegates this "read" right to his (her) subordinate managers so that they can grant the "read" right to the members in their project teams. In this circumstance, this file's "read" right has multiple administrators, i.e. the chairman, executive committee members and managers. If the policies mentioned above are used to resolve the conflicts, it may not be possible to prevent the following situation: a manager can grant a "not read" right for the file to the member of the executive committee (his/her) grantor) or even to the chairman (his/her) grantor's grantor); furthermore, this negative authorization can dominate the previous positive one the member already has. As a result, the member of the executive committee or the chairman can be denied to read the file. This is certainly not reasonable in practice. We claim that the problem comes from delegation without any control. This can lead to users not exercising the delegation of access rights since this means that they can lose control of the object and therefore risk sacrificing their privileges. We believe that delegation of rights should be supported in a large-scale decentralized access control system for flexibility, but at the same time, the delegation must be controlled. A promising approach to exercising this control is to adopt an appropriate conflict resolution policy.

In this paper, we propose a conflict resolution policy to achieve this controlled delegation. In our policy, if s_1 delegates or transtively delegates to s_2 some privilege, then for this privilege, s_1's granted privilege will have higher priority than s_2's granted privilege whenever they conflict over some other subject. Moreover s_2 is not allowed to grant s_1 any further authorizations. In other words, the priority of the subject decreases as the privilege delegation moves from one subject to another, and the subject with lower priority cannot grant authorizations to the subject with higher priority. Assuming all the rights on an object are first delegated from the owner of this object, the owner will always have the highest priority for this object and his/her authorizations can never be overridden. Thus users do not need to worry about losing control of the objects by using delegation of rights. The priority information comes from the grant connectivity relationship, which is dynamic and is usually different from object to object. In this way we can support controlled delegation of access rights, and take advantage of both distributed and centralised administrations of rights. As for the above example, since the priorities of subjects for "read" right on the file decreases from the chairman to committee members to managers, the unexpected situation discussed above will not occur. Furthermore, if some member of a team gets both "read" right on the file from his manager and "not read" right from the chairman, the chairman's granted privilege will dominate the other.

The remainder of this paper is organized as follows. In section 2, we propose a formal model of authorization conflict resolution. In section 3, we present the relevant algorithms that implement our model. In section 4, we briefly consider authorization state transformations based on our model. Finally, in section 5 we summarize the major contributions of this paper and outline some future work.

2 The Authorization Conflict Resolution Model

In this section, we outline the basic idea and provide a formal description of our authorization conflict resolution model.

2.1 The Basic Idea

In our authorization model, we allow both positive and negative authorizations, and permit access rights to be delegated from one subject to another. So, for any access right on any object, it may have multiple administrators that can grant authorizations. Different to the previously proposed conflict resolution policies, we classify conflict authorizations into two categories namely *comparable* and *incomparable* conflicts. Consider the situation where a subject s_3 is granted two conflicting authorizations with respect to an access right r on an object o from subjects s_1 and s_2 respectively. We say that these two conflicting authorizations are comparable if s_2's administrative privilege for r on o is granted (or transitively granted) by s_1, or vice versa. In the first case we assign a higher priority to s_1's grant than s_2's grant to solve the conflict occurring over the subject s_3. On the other hand, if there is no grant connectivity relationship between s_1 and s_2, then this conflicting authorization is said to be incomparable. In our model, we support multiple policies to solve incomparable conflicts to meet different users' requirements. For example, we may use the positive authorization to override the negative authorization or vice versa. We require that all the rights of an object be originally delegated from the owner of the object, so that the owner's authorization will take precedence over any other conflicting authorizations.

In addition, although some authorizations may be overridden by other authorizations, they are not eliminated from the authorization state. We preserve all the authorizations; they are either revoked explicitly or by recursive revocation. So conflicting authorizations can be present simultaneously in our model. The main advantages of this approach include the ability of re-activating the overridden authorizations after the other related authorizations are revoked, and the ability of changing the policy of incomparable conflicts resolution.

2.2 Notation and Definitions

Let S be a finite set of subjects (users), O be a finite set of objects (files, relations), R be a finite set of access rights (e.g. read, write, select, etc.), and T be a finite set of grant types. Then we have the following definition for authorization.

Definition 1. *(***Authorization***) An* authorization *is a 5-ary tuple* (s, o, t, r, g)*, where* $s \in S$*,*$o \in O$*,* $t \in T$*,* $r \in R$*,* $g \in S$*.*

Intuitively, an authorization (s, o, t, r, g) states that a grantor g has granted subject s the access right r on object o with grant type t. In this paper, we will consider three grant types: $T = \{*, +, -\}$, where

$*$: delegatable, which means the subject has been granted the access right r on o as well as the privilege for administration of r on o.
$+$: positive, which means the subject has been granted the access right r on o.
$-$: negative, which means the subject has been denied the access right r on o.

For example, $(user_1, file_1, +/-, read, user_2)$ states that $user_1$ is granted / denied to "read" $file_1$ by $user_2$, and $(user_1, file_1, *, read, user_2)$ states that $user_1$ is granted by $user_2$ not only the privilege to "read" $file_1$, but also the privilege to grant authorizations with respect to the "read" right on $file_1$ to other subjects.

Note that $*$ means $+$ together with administrative privilege on an access. The administrative privilege is related to a specific access right on an object. That is a subject may have the administrative privilege for "read" but not for "write".

Definition 2. *(***Authorization State***) An* authorization state *is the set of all authorizations at a given time.*

In this paper, we will usually use $\mathcal{A}$ and a (possibly with subscripts) to denote an authorization set and a single authorization respectively, and use $a.s$, $a.o$, $a.t$, $a.r$, $a.g$ to denote the corresponding components of subject, object, type, right and grantor of a respectively.

In order to formalize our approach, we use a *labelled digraph* to represent an authorization state as follows. For every object o, G_o is used to represent all the authorizations with respect to the object o. Let $G_o = (V, E, t, l)$ be a labelled digraph, where V is a finite set of vertices representing the subjects that hold some authorizations on o, E is a finite set of arcs such that if there exists an authorization (s, o, t, r, g) in authorization state $\mathcal{A}$, then (g, s) is in E, t is a type function from E to T, which maps every arc in E to a specific type in T. We will use different types of arcs, denoted as $t(e)$, to represent different grant types, as shown in Figure 1.

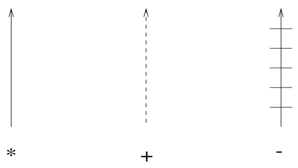

Fig. 1. Different Arc Types for Different Grant Types.

Suppose E_*, E_+, E_- denote the sets of $*$ arcs, $+$ arcs, and $-$ arcs respectively. Then $E = E_* \cup E_+ \cup E_-$. l is a label function from E to the power set of R, which maps every arc (g, s) of type t in E to a set of rights on o

that g grants to s and the grant type is t. For instance, if $t((g, s)) = *$, then $l((g, s)) = \{r \mid \exists(s, o, *, r, g) \in \mathcal{A}\}$. In G_o, every arc e is labelled with $l(e)$. In the rest of this paper, we will sometimes omit t and l and simply write $G = (V, E)$, whenever there is no confusion in the context. Following this, an authorization state $\mathcal{A}$ can be represented by a digraph G, which is a set of G_o for all objects o in the system. That is, $G = \{G_o \mid o \in O\}$. Figure 2 is an example of G_o.

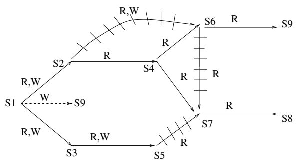

Fig. 2. G_o: an example of graph representation of authorizations on an object o, where $S1, ..., S9$ are subjects, R and W are access rights.

In addition, we will use $G_{o,r}$ to denote all the authorizations with respect to a specific access right r on o. That is, $G_{o,r}$ is a subgraph of $G_o = (V, E, t, l)$ that contains all arcs with the label containing r and the corresponding vertices. More formally, $G_{o,r} = (V', E', t')$, where $E' = \{(s_1, s_2) \mid (s_1, s_2) \in E$ and $r \in l((s_1, s_2))\}$, $V' = \{v \mid \exists v'(v, v') \in E'$ or $(v', v) \in E'\}$, and for any $e' \in E'$, $t'(e') = t(e')$. Note that there is no need for arc labels in $G_{o,r}$ anymore. For example, with reference to the G_o denoted in Figure 2, $G_{o,R}$ and $G_{o,W}$ can be illustrated in Figure 3 and Figure 4 respectively.

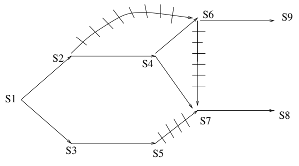

Fig. 3. $G_{o,R}$ of Figure 2.

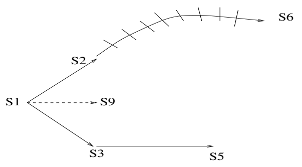

Fig. 4. $G_{o,W}$ of Figure 2.

In the rest of this paper, we will use the following graph terminologies. For an arc (a, b) in a digraph, a is called the *initial vertex* of (a, b), and b is called the *terminal vertex* of (a, b). *In-arc* of a vertex v is the arc with v as its terminal vertex and *in-degree* of a vertex v is the number of its *in-arcs*. *Out-arc* of a vertex v is the arc with v as its initial vertex and *out-degree* of a vertex v, is the number of its out-arcs. A *path* of *length* n from a to b is a sequence of one or more arcs (a, x_1) $(x_1, x_2), ..., (x_{n-1}, b)$, denoted by $a, x_1, \cdots, x_{n-1}, b$, and a is called the *predecessor* of b, while b is called the *successor* of a.

Now we define a binary relation on the set of subjects.

Definition 3. *(***Grant Connectivity Relation** $<_{o,r}$ **on Subjects***) Given an authorization state $\mathcal{A}$, for any subjects s_1,$s_2 \in S$, object $o \in O$, and access right $r \in R$, we say that s_1 is* grant-connected to s_2 *with respect to r and o in $\mathcal{A}$, denoted by $s_1 <_{o,r} s_2$, if there exists an authorization (s_2, o, t, r, s_1) for some t in $\mathcal{A}$, or there exists some subject s_3 satisfying $s_1 <_{o,r} s_3$, and $s_3 <_{o,r} s_2$.*

$s_1 <_{o,r} s_2$ means there exists a sequence of subjects $s_1, x_1, x_2, \cdots, x_n, s_2$ such that (x_1, o, t_0, r, s_1), (x_2, o, t_1, r, x_1), $\cdots$, (s_2, o, t_n, r, x_n) are all in the authorization state. In terms of our graph notation, $s_1 <_{o,r} s_2$ if and only if there exists a path from s_1 to s_2 in $G_{o,r}$, or in other words, s_1 is the predecessor of s_2 and s_2 is the successor of s_1 in $G_{o,r}$. The grant connectivity relation provides us with an important priority information about the subjects, which will be used later to solve the conflict problem. When the object and right are clear in the context, we sometimes simply write $s_1 < s_2$. For example, in the digraph $G_{o,R}$ of Figure 3, we have:

$S_1 < S_2 < S_4 < S_6 < S_7 < S_8$,
$S_1 < S_2 < S_4 < S_6 < S_9$, and
$S_1 < S_3 < S_5 < S_7 < S_8$.

2.3 Formal Description of the Model

We say that an authorization state $\mathcal{A}$ is **delegation correct**, if for any subject s, object o and right r, s can grant r on o to other subjects if and only if s has been granted r on o with delegation type $*$, that is, $\exists g, (s, o, *, r, g) \in \mathcal{A}$. In our graph representation, this means that in $G_{o,r}$, only the vertices pointed to by at least one $*$ arc can have out-arcs, while the vertices pointed to only by $+$ or $-$ arcs must be terminal ones, that is, their out-degrees must be zero. We assume that for every object o, only the owner of o, denoted by s_o, has been implicitly granted all the rights on o with delegation type by the system when the object is created. So if a state is delegation correct, then there will be a path from s_o to any other vertex in $G_{o,r}$.

We say that an authorization a_1 **contradicts** an authorization a_2 if $a_1.s = a_2.s$, $a_1.o = a_2.o, a_1.g = a_2.g, a_1.r = a_2.r$, but $a_1.t \neq a_2.t$. The contradictory authorizations state that a grantor gives the same subject two different types of authorizations over the same object with the same access right. For example, authorizations $(s_2, F_1, *, R, s_1), (s_2, F_1, +, R, s_1)$ and $(s_2, F_1, -, R, s_1)$ contradict each other. Figure 5 gives the corresponding graph representation. An authorization state $\mathcal{A}$ is **not contradictory** if for any a and a' in $\mathcal{A}$, a does not contradict a'. In our graph representation, this means that in any $G_{o,r}$ there is only one arc from each vertex to another.

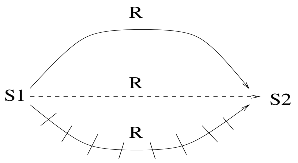

Fig. 5. Contradictory grants on object F_1.

Definition 4. (**Consistent Authorization State**) *An authorization state is* consistent *if it satisfies the following three conditions: 1. It is delegation correct, 2. It is not contradictory, and 3. For any object o and access right r, $<_{o,r}$ is a strict partial order.*

Recall that a strict partial order is transitive and anti-symmetric. In our graph notation, requiring relation $<_{o,r}$ to be a strict partial order means that the corresponding $G_{o,r}$ is acyclic.

In fact, by considering the properties of delegation correctness and not contradictory together, we have the following theorem:

Theorem 1. *Let $\mathcal{A}$ be a consistent state, then for any object o and access right r, $G_{o,r}$ in $\mathcal{A}$ is a simple rooted acyclic digraph, with the owner of the object as the root.*

Remember that in a simple graph there are no multiple arcs between each pair of vertices, and in the rooted acyclic graph, from the root one can reach any vertex in the graph. This theorem is easy to prove using the definition of consistent authorization state; so we omit the proof here. Figure 6 shows three examples of inconsistent authorization state, where $G1_{o,r}$ is not delegation correct because of the arc (s_2, s_3); $G2_{o,r}$ is contradictory because there are two arcs from s_5 to s_6; and $G3_{o,r}$ is cyclic because of the cycle s_8, s_9, s_8.

By requiring that $G_{o,r}$ acyclic, we have the following: if a subject s receives an authorization directly or indirectly from another subject s' on some object o and access right r, then s cannot grant s' any further authorization on o and r later on. In this way, we can solve the problem that exists in most conflict resolution methods discussed in section 1.

For a consistent authorization state $\mathcal{A}$ and a single authorization a, if $\mathcal{A} \cup \{a\}$ is still consistent, then we call a is *consistent* with $\mathcal{A}$. In our model, we require that the authorization state should always be consistent.

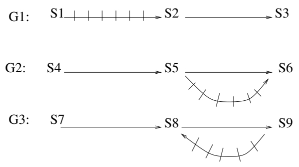

Fig. 6. Examples of Inconsistent Authorization State.

Definition 5. (**Conflicting Authorizations**) *For any two authorizations a_1 and a_2 in $\mathcal{A}$, a_1* conflicts *with a_2 if $a_1.s = a_2.s$, $a_1.o = a_2.o$, $a_1.r = a_2.r$, $a_1.t \neq a_2.t$ and $a_1.g \neq a_2.g$.*

From the definition, two authorizations are in conflict if they have the same subject, object and access right, but have different grant types and grantors. In our graph $G_{o,r}$, this means that the conflicting arcs have the same terminal vertex but different initial vertices and arc types. Since there are three grant types in our model, three kinds of conflicts may arise, as illustrated in Figure 7.

Note that type $*$ and $+$ are considered conflicting in the sense that $*$ holds the administrative privilege on an access right while $+$ does not. Conflicts are additionally classified into comparable conflicts and incomparable conflicts as follows.

Definition 6. (**Comparable Conflicts**) *Suppose a_1 and a_2 are any two conflicting authorizations on object o and access right r. Then a_1 and a_2 are* comparable *if $a_1.g <_{o,r} a_2.g$ or $a_2.g <_{o,r} a_1.g$. Otherwise they are* incomparable.

In other words, two conflicting authorizations are comparable if their grantors are grant-connected to each other. In our graph $G_{o,r}$, two conflicting arcs are comparable if there is a path between their initial vertices. For example, in Figure 3, (s_2, s_6) and (s_4, s_6), (s_4, s_7) and (s_6, s_7) are two pairs of comparable conflicts, while (s_4, s_7) and (s_5, s_7) are pairs of incomparable conflicts.

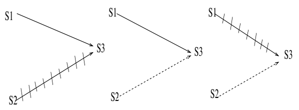

Fig. 7. Three Kinds of Conflicting Grants.

In fact, the grantors are comparable in comparable conflicts. In the grant relation path, we have higher priorities for the predecessors than the successors. So, when authorizations conflict with each other, the predecessor's grant will take precedence over the successor's. This idea can be formalized by the following overriding rule.

Definition 7. (**Overriding Rule**) *For any two authorizations a_1 and a_2 in $\mathcal{A}$, a_1* overrides *a_2 if $a_1.s = a_2.s$, $a_1.o = a_2.o$, $a_1.r = a_2.r$, and $a_1.g <_{o,r} a_2.g$. An authorization is* inactive *if there exists some authorization that overrides it. Otherwise it is* active. *We use $Act(\mathcal{A})$ to denote the set of all active authorizations in an authorization state $\mathcal{A}$.*

The overriding rule tells us that if two authorizations are about the same subjects, objects and rights, and their grantors are grant-connected to each other, then the authorization from the predecessor will override the one from the successor. Note that this definition does not require the grant types of the two authorizations to be different. Hence the predecessor's authorization will override the successor's even though they are not in conflict. This is reasonable since this means that the two authorizations are identical except for the grantor.

Correspondingly in the graph $G_{o,r}$ for some o and r, if two arcs point to the same vertex, and there is a path between their initial vertices, then the arc from the predecessor will override the arc from the successor. Let G be the graph corresponding to an authorization state $\mathcal{A}$; then active graph of G, denoted by $Act(G)$, is the subgraph of G that contains only active arcs. It is easy to show that for any $G_{o,r}$, $Act(G_{o,r})$ is still a rooted acyclic graph, since by using the overriding rule, the in-degrees of some vertices may be reduced but not to zero. But $Act(G_{o,r})$ may become inconsistent. For example, in Figure 3, (s_2, s_6) overrides (s_4, s_6) because s_2 is s_4's predecessor. For the same reason (s_4, s_7) overrides (s_6, s_7). Figure 8 gives the active graph of Figure 3. Note that it is inconsistent because the arc (s_6, s_9) is not delegation correct.

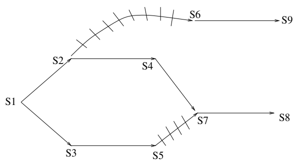

Fig. 8. Active Graph of Figure 3.

Definition 8. (**Effective State**) *If an authorization state $\mathcal{A}$ is consistent, then the maximal consistent subset of $Act(\mathcal{A})$ forms the* effective state *of $\mathcal{A}$, denoted by $Eff(\mathcal{A})$.*

Let G be the graph corresponding to an authorization state $\mathcal{A}$, then the effective graph of G, denoted by $Eff(G)$, corresponds to $Eff(\mathcal{A})$. $Eff(G)$ is in fact a set of $Eff(G_{o,r})$ for all objects o and access rights r of the system. Note that in the effective state, we have already eliminated all the comparable conflicts, that is, the conflicts in which their grantors are grant-connected to each other. Hence only the incomparable conflicts exist. Figure 9 gives effective graph of Figure 3.

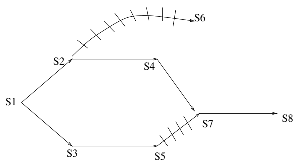

Fig. 9. Effective Graph of Figure 3.

Theorem 2. *A consistent authorization state $\mathcal{A}$ has a unique effective state $Eff(\mathcal{A})$.*

Proof. Obviously $Act(\mathcal{A})$ is unique. So we only need to prove that the maximal consistent subset of $Act(\mathcal{A})$ is unique.

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two maximal consistent subsets of $Act(\mathcal{A})$, and $a = (s, o, t, r, g)$ be any authorization in $\mathcal{A}_1$. Then there should be a corresponding arc (g, s) in $G'_{o,r}$ of $\mathcal{A}_1$. We need to prove (g, s) is also in $G''_{o,r}$ of $\mathcal{A}_2$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are both consistent, $G'_{o,r}$ and $G''_{o,r}$ are both rooted acyclic graph with root s_o. Let $maxlen(g)$ denote the length of the largest paths from the root s_o to g in $G'_{o,r}$. We will prove by induction of the $maxlen(g)$.

When $maxlen(g) = 0$, g is the root of $G'_{o,r}$, and is the owner of object o, and hence the result is certainly true. Suppose that the result is true when $maxlen(g) \leq k$. Consider the case when $maxlen(g) = k+1$. Suppose (g, s) is not in $G''_{o,r}$, then since $\mathcal{A}_2$ is a maximal consistent subset of $Act(\mathcal{A})$, (g, s) must be not consistent with $G''_{o,r}$. But (g, s) can not make $G''_{o,r}$ contradictory (i.e. there is more than one arc from g to s) or cyclic, since $G''_{o,r} \cup (g, s)$ is still a subgraph of $G_{o,r}$ of $\mathcal{A}$ and this will lead to $\mathcal{A}$ to be inconsistent. So (g, s) must make $G''_{o,r}$ to be not delegation correct. This means that the in-arcs of g does not have $*$ type in $G''_{o,r}$. According to the inductive hypothesis, all the in-arcs of g in $G'_{o,r}$ will be in $G''_{o,r}$ too and hence this will lead to $G'_{o,r}$ being not delegation correct. This is a contradiction. So (g, s) is also in $G''_{o,r}$. This concludes that $\mathcal{A}_1 \subseteq \mathcal{A}_2$. For the same reason $\mathcal{A}_2 \subseteq \mathcal{A}_1$. Thus $\mathcal{A}_1 = \mathcal{A}_2$.

Now let us consider the incomparable conflicts. We call an authorization state $\mathcal{A}$ is *conflict-free* if for any $a_1 \in \mathcal{A}$ and $a_2 \in \mathcal{A}$, a_1 is not in conflict with a_2.

Definition 9. *(***Stable State***) If an authorization state $\mathcal{A}$ is consistent, then the maximal consistent and conflict-free subset of $Eff(\mathcal{A})$ forms a* stable state *of $\mathcal{A}$, denoted as $stable(\mathcal{A})$.*

Note that an authorization state may have more than one stable state. In theory, one stable state presents one resolution to incomparable conflicts. Let G be the graph corresponding to an authorization state $\mathcal{A}$, then the stable graph of G, denoted by $stable(G)$, corresponds to $stable(\mathcal{A})$. $stable(G)$ is in fact a set of $stable(G_{o,r})$ for all objects o and access rights r in the system. Figure 10 and Figure 11 are two stable graphs of Figure 3.

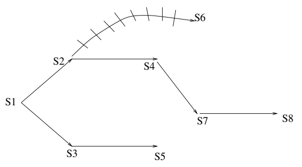

Fig. 10. One Stable Graph of Figure 3.

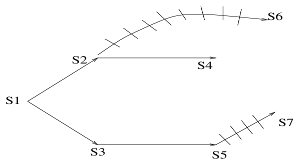

Fig. 11. Another Stable Graph of Figure 3.

For incomparable conflicts, we cannot resolve them using their grantor's priorities, since their priorities are not comparable. In our model we can support different strategies for resolving incomparable conflicts by evaluating different stable states. For example, we can support the following three strategies according to the grant types of authorizations:

(1) Pessimistic: the priority sequence is $- > + > *$;
(2) Optimistic: the priority sequence is $* > + > -$;
(3) Any : the priority sequence is $* = + = -$.

Hence a user can select the appropriate strategy that best suits the needs of his/her application. Even in one application, the strategy can vary from object to object. For example, for some objects that are very confidential, one can select the pessimistic strategy; for other objects that are not that sensitive, one can select the optimistic strategy. One can tell the system which strategy to apply to an object when the object is created, and one can change the strategy later when the sensitivity of the object is changed.

Another possible strategy for resolving incomparable conflicts is to grant an additional authorization to the subject over whom the conflicts occur by a common predecessor of the grantors of these conflicting authorizations, in particular, by the owner of the object. In this way we can change the incomparable conflicts to comparable conflicts and then can resolve them. In fact, the common predecessor here works like a judge in the sense that his/her decision has higher priority and hence can solve the dispute.

Now we can define our access control policy. We use 3-ary tuple (s, o, r) to denote an *access request* to the system, where $s \in S$, $o \in O, r \in R$. It states that a subject s requests to exercise access right r on object o. Then we have following access control policy.

Definition 10. (Access Control Policy) *Let $\mathcal{A}$ be an authorization state, (s, o, r) be an access request, P be a policy to resolve the incomparable conflict authorizations on o, and stable$(\mathcal{A}, P)$ be a stable state of $\mathcal{A}$ when applying P to o. We say that (s, o, r) is* permitted *if there exists some grantor g such that*

$(s, o, *, r, g)$ or $(s, o, +, r, g)$ is in stable$(\mathcal{A}, P)$; (s, o, r) is denied *if there is some grantor g such that $(s, o, -, r, g)$ is in stable$(\mathcal{A}, P)$; otherwise, (s, o, r) is* undecided.

It is worth mentioning that in our model, the system is the implicit grantor of any object's owner for all access rights on this object with delegatable grant type. In practice, the answer *undecided* may be treated as denial too. We prefer to distinguish them here to make the semantics more clear.

3 Algorithms

According to our access control policy, to determine an access request (s, o, r), we need to compute *stable*$(G_{o,r})$, which then need to compute $Eff(G_{o,r})$, and then compute $G_{o,r}$ from G_o. $G_{o,r}$ can be easily obtained by selecting all the arcs with r in their label and corresponding vertices from G_o. For $Eff(G_{o,r})$ and *stable*$(G_{o,r})$, we give the detailed algorithms in this section. We will also give the theorems about correctness and computational complexity of the algorithms, but will omit their proofs because of space limit.

Algorithm 3.1 is used to evaluate the effective graph of a $G_{o,r}$ for some object o and right r. The output is a graph G', and G'' is a temporary working graph used to construct a topological sorting of $<_{o,r}$.

Algorithm 3.1: *Evaluate_Eff_Graph*$(G_{o,r}, s_o)$
Input: $G_{o,r} = (V, E, t)$ for some object o and access right r, with root s_o
and arc type function t
Output: $Eff(G_{o,r}) = G' = (V', E', t')$

```
begin
1       $E' = \{(s_o, x) | (s_o, x) \in E\}$;
2       $V' = \{s_o\} \cup \{x | (s_o, x) \in E'\}$;
3       for all $e' \in E'$ do $t'(e') = t(e')$;
4       $E'' = E - E'$;
5       $V'' = V - \{s_o\}$; (* copy the root and out-arcs of root from $G_{o,r}$ to $G'$ and then copy
                    the remaining part of $G_{o,r}$ to $G''$*)
6       for each $v \in V''$ with 0 in-degree do begin
7           if the in-arcs of $v$ in $E'$ include $*$ type then begin
8               $P = \{x | (x, v) \in E\}$;
9               for each $p \in P$ do $P = P \cup \{x | (x, p) \in E\}$;
                (* compute all predecessors of $v$ in $G_{o,r}$*)
10              for each arc $(v, x)$ that goes out from $v$ in $E$ do begin
11                  if for each $p \in P$, $(p, x) \notin E$ then begin
12                      $E' = E' \cup \{(v, x)\}$ ;
13                      $V' = V' \cup \{x\}$;
14                      $t'(e') = t(e')$;
15                  end
```

```
16            end
17        end
18        $E'' = E'' - \{(v,x)|(v,x) \in E''\}$;
19        $V'' = V'' - \{v\}$;
20    end
end
```

Theorem 3. *Algorithm Evaluate_Eff_Graph is correct, and $Eff(G_{o,r})$ can be computed by Evaluate_Eff_Graph$(G_{o,r}, s)$ in $\mathcal{O}(N^3)$ time, where N is the number of vertices in $G_{o,r}$.*

Algorithm 3.2 evaluates a stable graph according to the policy for incomparable conflicts. Its input includes an effective graph G of $G_{o,r}$ for some object o and right r, the root s_o, and the policy P to be used. Its output is G', the stable graph of $G_{o,r}$ corresponding to P. G'' is a temporary working graph used to construct a topological sorting of $<_{o,r}$.

Algorithm 3.2: *Evaluate_Stable_Graph*(G, P, s_o)

```
Input: $G = (V, E, t)$ – $G$ is a effective graph of $G_{o,r}$ for some object $o$ and access right $r$,
          with root $s_o$ and arc type function $t$,
       $P$ – the policy of solving incomparable conflicts over $o$
Output: $stable(G_{o,r}, P) = G' = (V', E', t')$
begin
1     $E' = \{(s,x)|(s,x) \in E\}$;
2     $V' = \{s\} \cup \{x|(s,x) \in E'\}$;
3     for all $e' \in E'$ do $t'(e') = t(e')$;
4     $E'' = E - E'$;
5     $V'' = V - \{s\}$; (* copy the root and out-arcs of root in $G$ to $G'$ and
                         then copy the remaining part in $G$ to $G''$ *)
6     for each $v \in V''$ with 0 in-degree do begin
7         if the in-degree of $v$ is greater than 1 in $G'$
8         then select any in-arc that has the highest priority according
               to policy $P$ for incomparable conflicts and delete other in-arcs from $G'$
9         if $v$'s in-arc in $E'$ is type $*$ then begin
10            $E' = E' \cup \{(v,x) \mid (v,x) \in E\}$ ;
11            $V' = V' \cup \{x \mid (v,x) \in E'\}$;
12            $t'((v,x)) = t((v,x))$ for all $(v,x) \in E'$;
13        end
14        $E'' = E'' - \{(v,x)|(v,x) \in E''\}$;
15        $V'' = V'' - \{v\}$;
16    end
end
```

Theorem 4. *Algorithm Evaluate_Stable_Graph is correct, and stable($G_{o,r}$) can be computed by Evaluate_Stable_Graph(G, P, s) in $\mathcal{O}(N^2)$ time, where N is the number of vertices in G.*

4 Authorization State Transformation

In a dynamic environment, an authorization state is not static since users may need to add, update, or revoke certain authorizations. In this section, we consider how our proposed authorization state can be changed. Since an update can be implemented by revoking and adding, here we only consider the addition and revocation of authorizations in our model. Note that adding and revoking authorizations will update both the effective and stable state.

In the case of addition, an authorization $a = (s, o, t, r, g)$ can be added to the authorization state $\mathcal{A}$ if and only if it is consistent with the current state $\mathcal{A}$. This means that in $\mathcal{A}$, g must get the delegatable right for r on o, a can not contradict with any other authorization in $\mathcal{A}$ and $<_{o,r}$ must still be a partial order after the addition.

For revocation, on the other hand, we adopt a cascading revocation approach to implement this operation. An authorization $a = (s, o, t, r, g)$ can be revoked from the system if the requester is g, and the authorization state must remain consistent after the revocation. Users are also allowed to change the policy of resolving incomparable conflicts for an object. This would lead to an update of the stable state.

For example, for the $G_{o,R}$ shown in Figure 3, $(s_3, o, R, -, s_7)$ or $(s_8, o, R, -, s_7)$ cannot be added to $G_{o,R}$ because adding $(s_3, o, R, -, s_7)$ will result in a cyclic graph while adding $(s_8, o, R, -, s_7)$ generates a graph which represents a contradictory authorization state. But adding $(s_5, o, R, +, s_1)$ is allowed. On the other hand, consider the situation where s_2 requests to revoke $(s_4, o, R, *, s_2)$. This will lead to the deletion of arc (s_2, s_4) and cascading deletion of arcs (s_4, s_6), (s_6, s_9), (s_6, s_7) (s_4, s_7), and (s_7, s_8) from the graph. The resulting $G_{o,R}$ after the addition and revocation is shown in Figure 12.

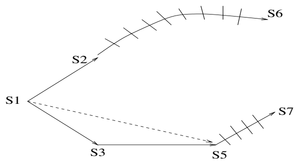

Fig. 12. $G_{o,R}$ after update.

5 Conclusions and Future Work

In this paper, we have proposed a conflict resolution model to resolve conflicts that can occur when access rights are delegated. A major feature of our approach is that we classified conflicts into comparable and incomparable ones and this classification is useful not only in the control of access right delegation but also for supporting multiple policies to resolve conflicts. Our model also provides a flexible framework to preserve conflict authorizations so that it is possible to re-activate some early overridden access rights if proper authorizations are revoked or the policy of conflict resolution is changed.

With respect to the incomparable conflict resolution, our current model provides four different policies. In fact, this can be further extended. For example, under some situations, we may expect to have a logical mechanism to deal with incomparable conflicts. In this case, we can re-formalize the notion of stable state by associating proper logical relationships among those incomparable conflicts. The other issue we have not discussed in this paper is concerned with inheritance in conflict resolution. If we consider the inheritance relationship between subjects, objects, and access rights respectively, then the conflict resolution can become complex because many conflicts may be *implicit* and solving these implicit conflicts will require some reasoning procedure to deal with inheritance in the access control model. These issues will be investigated in our future work.

References

1. E. Bertino, F.Buccafurri, E.Ferrari, P.Rullo, A logical framework for reasoning on data access control policies. *proceedings of the 12th IEEE Computer Society Foundations Workshop*, IEEE Computer Society Press, Los Alamitos, 1999, pp.175-189.
2. E. Bertino, S. Jajodia, P. Samarati, Supporting multiple access control policies in database systems. *Proc.of the IEEE Symposium on Research in Security and Privacy*, Oakland(CA), 1996.
3. R. Fagin, On an authorization mechanism. *ACM Transaction on Database Systems*, Vol. 3, 1978, pp 310-319.
4. M.Harrison, W.Ruzzo and J.Ullman, Protection in operating systems. *Communications of ACM 19(8)*,pp 461-471, 1976.
5. N.Gal-Oz, E. Gudes, and E.B. Fernandez, A model of methods access authorization in object-oriented databases. *Proceedings of International Conference on Very Large Data Bases*, pp 52-61,
1993.
6. T.F. Lunt et al, *Secure Distributed Data Views*, Vol. 1-4, SRI International, 1989.
7. F. Rabitti, E. Bertino, W. Kim, and D. Woelk, A model of authorization for next generation database systems. *ACM Transaction on Database Systems*, Vol 16, pp 88-131, 1991.
8. M. Satyanarayanan, Integrating security in a large distributed system. *ACM-TOCS*, vol.7, no.3, pp 247-280, Aug. 1989.
9. T. Woo and S. Lam, Designing a distributed authorization service. *Proceedings of IEEE INFOCOM'98*, 1998.