A Unified Conflict Resolution Algorithm

Conference Paper · September 2007

DOI: 10.1007/978-3-540-75248-6_1 · Source: DBLP

Amir H. Chinaei
University of Puerto Rico at Mayagüez
9 PUBLICATIONS **49** CITATIONS
SEE PROFILE

Hamid R. Chinaei
Carleton University
29 PUBLICATIONS **147** CITATIONS
SEE PROFILE

Frank W. Tompa
University of Waterloo
137 PUBLICATIONS **2,545** CITATIONS
SEE PROFILE

A Unified Conflict Resolution Algorithm

Amir H. Chinaei, Hamid R. Chinaei, and Frank Wm. Tompa

David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON N2L 3G1, Canada
Tel.: +1 (519) 888-4567 extensions {36612, 33400, 34675}
{ahchinaei,hrchinaei,fwtompa}@uwaterloo.ca

Abstract. While some authorization models support either positive or negative authorizations, hybrid frameworks take advantage of both authorizations. Resolving authorization conflicts is quite a challenge due to the existence of sophisticated inheritance hierarchies and the diversity of ways to combine resolution policies. Some researchers have addressed conflict resolution for tree-structured hierarchies, and others have applied a simple conflict resolution policy. The challenge is to combine several policies and to support sophisticated structures in one single framework. This paper proposes a unified framework together with a single parametric algorithm that supports all the legitimate combinations simultaneously, based on four conflict resolution policies. We validate our approach by testing the algorithm against both real data and synthetic examples to provide extensive experimental results.

Keywords: Access Control Models, Hybrid Authorizations, Conflict Resolution.

1 Introduction

Because individual users can assume several personae (e.g., payroll clerk, member of the social committee, and occupant on the fourth floor), they may be simultaneously authorized for some activity and denied authorization for that same activity when viewed in another role. There exist several conflict resolution policies, such as "denial takes precedence" and "the most specific authorization takes precedence," in the literature of access control models. However, combining the policies provides a variety of different comprehensive conflict resolution strategies, which have not been well addressed. Designers of access control models typically choose a specific approach to conflict resolution and incorporate a hardwired conflict resolution method within their models. Consequently, if an enterprise subsequently decides to choose an alternative conflict resolution strategy, the whole system has to be replaced. Maintaining separate software for multiple strategies is expensive for software providers. We propose one single parameterized algorithm by which security administrators can invoke a chosen strategy, among many, without needing to reinstall the whole system.

1.1 Motivating Example

Figure 1 illustrates a subject inheritance hierarchy including nine subjects. The arrows represent group membership (e.g., subjects S_4 and S_5 are members of subject/group S_3) and the sign labels represent explicit authorizations (+ indicates positive authorization and – represents denial). For simplicity of exposition, we assume that access to an object is either granted or denied (rather than separately controlling reading, writing, and other operators), and we illustrate only authorizations for a single object. The figure shows that subjects S_2 and S_4 are explicitly authorized to access the object, whereas subject S_5 is explicitly denied from accessing it.

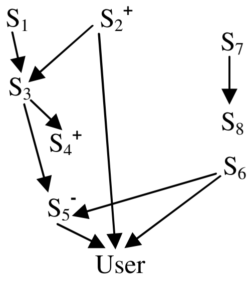

Fig. 1. An example of the subject hierarchy

Given the data in Figure 1, assume we are interested in knowing whether or not subject *User* is authorized to access the object. One may interpret the data to mean that the object is accessible to subject *User* since *User* is a descendant of S_2 and thereby inherits S_2's authorizations. However, another may argue that the object should not be accessible to *User* since he is a member of S_5 which is denied access. In fact, there is a conflict in the system. Conflict resolution policies are needed to answer such questions.

Adopting one simple policy, such as “the most specific authorization takes precedence,” is not sufficient in practice. For instance, such a policy is insufficient where the subject hierarchy is more complex than tree-based structures and therefore, a subject may have more than one “most specific” authorization. For example, in Figure 1, neither S_2 nor S_5 is more specific to *User*, with respect to the other, since both of them are at the minimum distance of 1 from *User*. Furthermore, there are situations in which the highest authority (not the most specific one) should be the final arbiter. For instance, assume a student is authorized by the university athletic office to referee hockey games on campus (which requires more than 20 hours per week for several weeks); however, he is required by the department not to accept heavy non-departmental tasks (in order to comply with his full-time registration status). In such a case, the university administration may resolve the conflict by deciding to let him to referee for a limited time. To visualize such a case, assume there is an edge from S_1 to S_2 in Figure 1 and S_1 is labeled positively. Representing the student by *User*, the referees group by S_2, the members of the department by S_5, and the university members by S_1, it is apparent that for this enterprise the most global authorization should take precedence in resolving the conflict.

Some have proposed the "negative takes precedence" policy, but this too is not universally acceptable. For instance, conflicts often are resolved by the "majority takes precedence" rule in voting systems. Additionally, the open policy recommends a default positive authorization for subjects which are not explicitly permitted to access a particular object [6, 10]. Therefore, there are applications in which "positive takes precedence."

Even from these simple examples we see that it is required to combine various conflict resolution policies to obtain a comprehensive conflict resolution framework that supports several strategies simultaneously. Moreover, each policy may encompass several variants, and consequently many strategy instances are possible. What are all the legitimate strategy instances? Is there a unified algorithm to support all instances parametrically?

Chinaei and Zhang addressed the first question by providing a conflict resolution framework in which 32 legitimate instances are supported [2]. In that same paper, they proposed the *Dominance()* algorithm for one of the compound conflict resolution strategies and provided some guidelines to extend the algorithm to other strategies. However, designing a single parameterized conflict resolution algorithm to support all combined strategies remained open.

In this paper, we first extend the framework to support 16 additional strategy instances. Thereafter, we present a unified parametric algorithm that supports the comprehensive conflict resolution framework for hybrid authorization models in which the subject hierarchy is a directed acyclic graph.

1.2 Outline

Section 2 reviews four major conflict resolution policies and consequent combined strategies as presented in Chinaei and Zhang's work [2], and extends the framework to support 48 strategy instances. Section 3 describes our unified parametric algorithm and justifies how the algorithm supports all the instances. Section 4 presents our experimental results. Section 5 reviews the literature. Finally, Section 6 summarizes our contributions and addresses several future directions.

2 Conflict Resolution Models

Before proceeding, we assume the reader is familiar with the following terminology. Access control data can be viewed conceptually as being represented by an access control matrix, where the rows represent subjects, the columns represent objects, and authorizations are stored at the intersections [10]. However, not every subject-object pair has explicit rights assigned. Instead, access control for subject/object pairs with no explicit rights must be derived by other means, e.g. through inheritance via the subject hierarchy.

We therefore distinguish between an *effective access control matrix*, which represents all explicit and derived authorizations, and an *explicit access control matrix*, which represents explicit authorizations only. Given an explicit matrix, conflict resolution strategies are used to fill in all derived authorizations to determine the effective matrix. It is important to note that the explicit matrix is typically very sparse in practice and that the effective matrix is by definition completely filled; therefore practical systems will store the explicit matrix and compute access control authorizations as needed by executing a conflict resolution algorithm on an appropriately extracted subset of that matrix.

Chinaei and Zhang outlined four main policies included in their conflict resolution framework: *preferred authorization*, *locality* (or *globalization*), *majority*, and *default authorization* [2]. These policies have been articulated by other researchers and appear in various real world applications, but they are typically discussed independently and not in combination. In Section 2.1, we restate each policy briefly and independently of other policies, as well as providing some examples of their applicability. Then, in Section 2.2 we explain how legitimate combinations of these policies lead us to define several consequent strategy instances.

2.1 Conflict Resolution Policies

Our model assumes that the subjects for whom authorizations are to be determined are structured as a directed acyclic graph. Individuals are represented as sink nodes; a group of individuals is represented by a node with outgoing edges to each member of the group; a group of groups is represented by a node with outgoing edges to each subgroup member of that group. In general, a group can have zero or more subgroups and zero or more individual nodes. We do not restrict the subject hierarchy to form a tree.

All authorizations of a group may be applicable to a member of that group. As illustrated in Section 1, propagating explicit authorizations to derive effective authorizations of the group members may cause conflicts. Access control must determine for each subject/object/operation whether the subject is to be allowed to execute the operation on the object or denied such execution. Conflict resolution is required when propagating authorizations results in no decision for a particular subject/object/operation triple or when both positive and negative authorizations can be derived for that triple. Here, we outline popular conflict resolution policies that we include in our model.

Preference Policy. Preferred authorization (with one of two modes: either positive or negative) is determined by the system installer at configuration time. This policy determines which authorization wins when both positive and negative authorizations (or neither negative nor positive authorization) can be derived for a particular subject. Negative authorization is preferred (known as closed policy) in more restricted systems such as military; positive authorization may be preferred in more open applications such as public information systems.

Locality Policy. The common mode of this distance-based policy states that the most specific authorization takes precedence. It applies to distributed organizations whose local branches may recognize an exception to a general rule. For instance, a department in a university may admit an outstanding applicant although the general admission requirement is not completely met. Thus, for a given subject, when both positive and negative authorizations can be derived from different ancestors, the one that is closer to the subject wins. Note that the distance between two nodes (subjects) in a directed acyclic graph is measured by computing the shortest directed path.

The locality policy is not deterministic since no authorization wins when the distances are equal.

As an alternative for the locality policy, some enterprises might choose “globality”, where the most general authorization takes precedence. One application of this policy is in distributed organizations whose headquarters makes the final decision on a pre-approved task by a local office. For instance, a supreme court may override the appealed decision. For a given subject, when both positive and negative authorizations can be derived from different ancestors, the one that is farther from the subject wins. Similar to the usual locality policy, globality is not deterministic since no authorization may win.

Majority Policy. This policy states that the conflict can be resolved based on votes, and the authorization that has the majority wins. The application of this policy is in situations where several parties have different opinions for giving or not giving the authorization to a particular member and the decision is made by votes. For instance, GATT’s current members vote to determine if a new application can get into the group. By applying this policy, the dominant authorization takes precedence. This policy is also non-deterministic since it can result in a tie.

Default Policy. This policy is applied only to root nodes for which no authorization has been defined. Closed systems, such as in the military, require negative authorization by default; however, open systems, such as public information applications, initially allow any subject to enjoy a positive authorization. This policy is deterministic and has two modes (default positive or default negative), but applies to root nodes only. Note that for non-root nodes only the preference policy is deterministic.

2.2 Combined Strategies

Figure 2 illustrates five conflict resolution strategies presented in Chinaei and Zhang’s work [2], based on combining the popular conflict resolution policies summarized in Section 2.1. They are given the mnemonics DLP, DLMP, DP, DMLP, and DMP, in which D, L, M, and P indicate Default, Locality, Majority, and Preference policies, respectively. Two properties are guaranteed: first, none of the policies are redundant, and second, there is no conflict after applying the last step. Note that in this framework the Default and Preference policies are always the first and the last applicable policies, respectively, and the other two policies, Locality and Majority, are optional.

Chinaei and Zhang state that because the Default, Locality, and Preference policies can take two modes each, there are 32 different strategies instances in total that can be derived from Figure 2 [2]. (Paths ending with *a*, *b*, and *d* generate eight instances each, and paths ending with *c* and *e* generate four instances each.)

We recognize that there are also applications in which the default policy is not appropriate. For instance, in determining whether S_3 is authorized access according to Figure 1, we may wish to give priority to the explicit authorization on S_2 regardless of whether the value assigned to S_1 is positive or negative. This can be accomplished in general by omitting the default policy, using locality or majority as desired to arbitrate a value for S_3 and using the preferred authorization to assign a value to S_1 only afterwards.

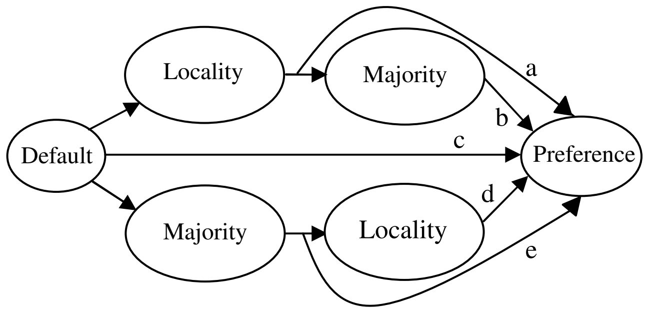

Fig 2. Combined conflict resolution strategies

Therefore, to achieve a comprehensive framework, we augment the combined strategies illustrated in Figure 2 with making the default policy optional as well. This results in five more combined strategies namely LP, LMP, P, MLP, and MP. Hence, the framework supports 16 more instances; strategies LP, LMP, and MLP generate four instances each, and strategies P and MP generate two instances each.

Note that no other combined strategy can be meaningfully composed from these basic conflict resolution policies. For example, the preference policy cannot be optional and must be considered last, since it is the only policy that is well-defined on every node.

3 Implementation

This section describes an algorithm that propagates explicit authorizations through the subject hierarchy, and resolves the possible conflicts based on any of the 48 strategy instances illustrated in Section 2. In particular, Section 3.1 describes details of our conflict resolution algorithm (called *Resolve()*). After that, Section 3.2 describes the propagation of explicit authorizations.

To determine whether a given object, o_j, is effectively accessible to a given subject, S_i, with respect to a given right, r_k, the idea is to apply the following four-step procedure:

Step 1: Consider the maximal sub-graph (called *H*) of the subject hierarchy in which S_i is the sole sink and all other nodes are its ancestors.
Step 2: Assign a letter “d” to all root subjects in *H* that are unlabeled with respect to object o_j and right r_k.

Figure 3 illustrates the result of Steps 1 and 2 for subject *User*, object *obj*, and right *read*, illustrated in Figure 1 as the motivating example.
Step 3: Propagate all authorization labels down every path to subject *User* and store the distance of each propagated authorization from its source node to its destination node (*User*). For instance, the distance of label - (on S_5) to node *User* is 1; also, there are two distances for label “d” (on S_6) to node *User*: one (with value 1) directly from S_6 to *User*, and one (with value 2) via S_5.

Table 1 illustrates the result of authorization propagation for subject *User*, object *obj*, and right *read* as represented by Figure 3.

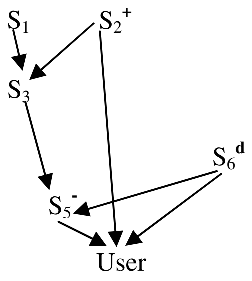

Fig. 3. Sub-graph of subject *User*

Table 1. All *read* authorizations of *User* on *obj*

subject	**object**	**right**	**dis**	**mode**
User	obj	read	1	-
User	obj	read	1	d
User	obj	read	2	d
User	obj	read	1	+
User	obj	read	3	+
User	obj	read	3	d

Table 2. Resolved authorization for each combined strategy

strategy	mode	strategy	mode	strategy	mode	strategy	mode
D$^+$L**M**P$^+$	+	D$^+$L**P**$^+$	+	LM**P**$^+$	+	D$^+$**M**LP$^+$	+
D$^+$L**M**P$^-$	+	D$^+$L**P**$^-$	-	LM**P**$^-$	-	D$^+$**M**LP$^-$	+
D$^-$L**M**P$^+$	-	D$^-$L**P**$^+$	+	**G**MP$^+$	+	D$^-$**M**LP$^+$	-
D$^-$L**M**P$^-$	-	D$^-$L**P**$^-$	-	**G**MP$^-$	+	D$^-$**M**LP$^-$	-
D$^+$**G**MP$^+$	+	D$^+$**G**P$^+$	+	**M**P$^+$	+	D$^+$**M**GP$^+$	+
D$^+$**G**MP$^-$	+	D$^+$**G**P$^-$	+	**M**P$^-$	+	D$^+$**M**GP$^-$	+
D$^-$GM**P**$^+$	+	D$^-$G**P**$^+$	+	L**P**$^+$	+	D$^-$**M**GP$^+$	-
D$^-$GM**P**$^-$	-	D$^-$G**P**$^-$	-	L**P**$^-$	-	D$^-$**M**GP$^-$	-
D$^+$**M**P$^+$	+	D$^+$**P**$^+$	+	**G**P$^+$	+	**M**LP$^+$	+
D$^+$**M**P$^-$	+	D$^+$**P**$^-$	-	**G**P$^-$	+	**M**LP$^-$	+
D$^-$**M**P$^+$	-	D$^-$**P**$^+$	+	**P**$^+$	+	**M**GP$^+$	+
D$^-$**M**P$^-$	-	D$^-$**P**$^-$	-	**P**$^-$	-	**M**GP$^-$	+

Step 4: Apply a particular conflict resolution strategy to resolve any conflicts and derive a final *effective* authorization for the triple <S_i, o_j, r_k>.

Table 2 illustrates the result of applying each of the 48 strategy instances (explained in Section 2) to Table 1. For example, D$^+$LMP$^+$ is the strategy instance in which first the default policy is applied and every root subject which is null is initialized to +; then, if there is a conflict, the Locality policy ("the most specific authorization takes precedence") is applied; then if there is still a conflict, the Majority policy is applied; and finally, if the conflict is not resolved, the preference policy in which the positive (+) authorization takes precedence is applied. Let's see what the result of this strategy instance is on Table 1: by applying the default policy D$^+$, all mode d's are replaced by +; by applying the locality policy, the conflict is not resolved since there are conflicting modes + and - from the shortest distance 1; however, by applying the majority rule, mode + wins over mode - since there are more + entries than - entries. Note that the preference policy is not applicable to this case since the conflict is resolved before this rule is triggered; however in other hierarchies the conflict need not yet have been resolved.

For each strategy instance in Table 2, we use a bold font to show which policy has determined the effective authorization when applied to our example. For example, in the last strategy instance, MGP$^-$, by applying the first policy (Majority), the positive authorization wins since there are two +'s (rows 4 and 5) as opposed to only one – (row 1) in Table 1. Therefore, the localization and preference policies of the MGP$^-$ instance are not applicable to this case.

3.1 Algorithm *Resolve()*

This **section** defines our conflict resolution algorithm. Figure 4 illustrates Algorithm *Resolve()* which computes the derived authorization mode of a given subject with respect to a given object and right. The algorithm parameters are *s*, *o*, *r*, *dRule*, *lRule*, *mRule*, and *pRule;* and the result is either + or -. Parameters *s*, *o*, and *r* designate a particular subject, object, and right, respectively, on which the caller is interested to know whether or not the object is accessible to the subject with respect to the specified right. Parameters *dRule*, *lRule*, *mRule*, and *pRule* determine the conflict resolution strategy based on which the final right of the subject on the object must be derived. In particular, parameter *dRule* represents the default policy and takes either of the three values "+", "-", or "0", which respectively states that the unlabelled root ancestors of the subject are to be initialized to positive authorization, negative authorization, or remain null (no default policy). Parameter *lRule* represents the locality policy; its value is either *min*(), *max*(), or *identity*(), which represent "the most specific authorization takes precedence, " "the most general authorization takes precedence, " or "no locality policy" modes, respectively. Parameter *mRule* takes three values *before*, *after*, or *skip*, which determines whether the majority policy is applied before the locality policy, after it, or not at all, respectively. Finally, parameter *pRule* represents the preference policy and determines whether positive or negative authorization is preferred in the case of remaining conflicts. (We assume that the subject hierarchy (SDAG) and the explicit access control matrix (EACM) are globally defined in the algorithm.)

In Line 1, relation *allRights* is created by calling Function *Propagate().* The details of this function are explained in the next section, but the effect is to apply the first three steps of the procedure described in the introduction to Section 3. An instance of relation *allRights* is illustrated in Table 1, where all the rights of subject *User* on object *obj* with respect to authorization *read* are depicted.

In Line 2, we check whether the caller is interested in applying the default authorization policy (*dRule* = "+" or "-") or not (*dRule* = "0"). In the latter case, only those rows of relation *allRights* are considered in which *mode* <> "d". In the former case (Line 3), those rows of relation *allRights* in which *mode*="d" are updated with the value of *dRule* ("+" if positive authorization is to be the default policy, "-" otherwise).

In Line 4, if the majority policy should be applied before the locality policy, we count the number of positive authorizations and negative authorizations that exist in relation *allRights*; however, as stated in Line 5, if the majority policy should be applied after the locality policy, we first apply the locality on relation *allRights*, and then count the number of positive and negative authorizations. In either of these cases (Line 6), the algorithm returns the authorization which is in majority.

```
Algorithm Resolve(s, o, r, dRule, lRule, mRule, pRule)
¤ To check whether subject s has a positive or a negative authorization for right r on object o
¤ Default rule dRule ∈ {"+", "-", "0"}
¤ Locality rule lRule ∈ {max(), min(), identity()}
¤ Majority rule mRule ∈ {"before", "after", "skip"}
¤ Preference rule pRule ∈ {"+", "-"}
¤ SDAG (subject hierarchy) and EACM (explicit access control matrix) are globally defined.

Output: either "+" or "-"

1. allRights ← Propagate (s, o, r, SDAG, EACM);
2. if dRule = "0" allRights ← $\sigma_{mode<>"d"}\ allRights$
3. else update allRights set mode=dRule where mode="d";
4. if mRule = "before" {$c_1 \leftarrow \Pi_{count()}\left(\sigma_{mode="+"}\ allRights\right)$; $c_2 \leftarrow \Pi_{count()}\left(\sigma_{mode="-"}\ allRights\right)$}
5. if mRule = "after" {$c_1 \leftarrow \Pi_{count()}\left(\sigma_{mode="+",\ dis=lRule(dis)}\ allRights\right)$; $c_2 \leftarrow \Pi_{count()}\left(\sigma_{mode="-",\ dis=lRule(dis)}\ allRights\right)$}
6. if mRule <> "skip" { if $c_1 > c_2$ return "+"; if $c_2 > c_1$ return "-"; }
7. Auth ← $\Pi_{mode}\left(\sigma_{dis=lRule(dis)}\ allRights\right)$;
8. if count(Auth) = 1 return Auth;
9. return pRule;
```

Fig. 4. Algorithm *Resolve ()*

If neither positive nor negative labels is in the majority or the majority policy is not designated at all, we apply the locality policy to relation *allRights* to select its relevant rows (Line 7); if *lRule = min()*, only rows in which the value of column *dis* is equal to the minimum distance (the most specific authorizations) are selected; similarly, if *lRule = max()*, only rows in which the value of column *dis* is equal to the maximum distance (the most general authorizations) are selected; however, if *lRule=identity()*, all rows are selected (this is equivalent to no locality policy being designated).

Next, the values of column *mode* of corresponding rows are projected to form a set called *Auth*, which may be empty or contain positive and negative authorizations. If only one type of authorization is present (Line 8), it is returned; otherwise, the preferred authorization (*pRule*) is returned and the algorithm ends.

Table 3 illustrates the result of Algorithm *Resolve()* applied to our motivating example for several illustrative strategies. In particular, we trace the algorithm for eight strategy instances (selected from Table 2) namely D^+LMP^+, D^-GMP^-, D^-MP^-, D^-LP^+, D^+GP^-, P^-, GMP^-, and MGP^-. Table 3 shows values of c_1, c_2, *Auth*, and the effective mode derived by the algorithm, as well as its corresponding return line number. In the table, *n/a* means that the algorithm does not use the corresponding variable for the conflict resolution.

For instance, if one chooses the strategy instance $D^{-}GMP^{-}$, all default values of relation *allRights* are replaced with "-" (Line 3). Since the global mode of the locality policy is in place and there are one positive and one negative authorization from distance 3 in Table 1, both c_1 and c_2's values are assigned the value 1 (Lines 5). Then, since neither positive nor negative is in majority, the algorithm continues to Line 7, and *Auth* is assigned the value {+,-}. Finally, since there is a conflict in *Auth*, Line 9 of the algorithm returns the value of preference policy, which is "-" (indicated by P^{-} in the strategy instance), as the final derived decision with respect to triple <*User*, *obj*, *read*>.

As another example, if one chooses strategy instance MGP^{-}, in Line 2, only those rows of relation *allRights* in which the mode is not "d" are selected. Then, since the globalization policy is in place and there is one explicit positive authorization from distance 3 in relation *allRights*, the value of c_1 is set to 1 and the value of c_2 is set to 0 in Lines 4. Finally, the algorithm returns "+" from Line 6 as the final derived decision with respect to triple <*User*, *obj*, *read*>.

Table 3. Trace of *Resolve()*

Strategy	c_1	c_2	*Auth*	*mode*	Line
$D^{+}L\mathbf{M}P^{+}$	2	1	*n/a*	+	6
$D^{-}GM\mathbf{P}^{-}$	1	1	+,-	-	9
$D^{-}\mathbf{M}P^{-}$	2	4	*n/a*	-	6
$D^{-}L\mathbf{P}^{+}$	*n/a*	*n/a*	-,+	+	9
$D^{+}\mathbf{G}P^{-}$	*n/a*	*n/a*	+	+	8
$\mathbf{G}MP^{-}$	1	0	*n/a*	+	6
$\mathbf{P}^{-}$	*n/a*	*n/a*	-,+	-	9
$\mathbf{M}GP^{-}$	1	0	*n/a*	+	6

3.2 Function *Propagate*()

In this section, we explain the details of Function *Propagate()*, which returns all corresponding authorizations of a given subject, object, and authorization, shown as <*s*, *o*, *r*> when called from Line 1 in Algorithm *Resolve()*. The idea is to extract the part of the subject hierarchy in which *s* is the only sink. Then, using top-down breadth-first propagation, all authorizations from root subjects are propagated towards *s*. If a root subject has no authorization assigned in the explicit matrix, a letter "d" is assigned to it to represent the default authorization. Moreover, the distance of each authorization to *s* is computed, so that it can be exploited by Algorithm *Resolve()* if the locality policy is applied. Note that authorizations are propagated from *all paths* starting from the source node and ending at the destination. For instance, in Figure 3, authorization + from subject S_2 is propagated to subject *User* along two paths: $S_2 \rightarrow User$, and $S_2 \rightarrow S_3 \rightarrow S_5 \rightarrow User$.

Figure 5 illustrates Function *Propagate()*, which inputs parameters *s*, *o*, and *r* (representing the subject, object, and authorization on which the conflict should be resolved), as well as *SDAG* and *EACM*, which represent the subject hierarchy and the explicit access control matrix, respectively.

```
Function Propagate (s, o, r, SDAG, EACM);
¤ To obtain all authorizations, with respect to triple <s, o, r> by propagating explicit authorizations in EACM through subject hierarchy SDAG
¤ EACM has attributes <subject, object, right, mode>
¤ SDAG has attributes <subject, child>
¤ $ancestors(s) = \{s\} \cup \{x \mid \exists y\ \langle y,s\rangle \in SDAG \wedge x \in ancestors(y)\}$
Output: table allRights
1. $SDAG' \leftarrow \sigma_{subject \in ancestors(s),\ child \in ancestors(s)} SDAG$;
2. $i = 0$;
3. $P \leftarrow \Pi_{subject, object, permission, i, mode} (SDAG' \bowtie \sigma_{permission=r,\ object=o} EACM)$;
4. $Roots \leftarrow \Pi_{subject} SDAG' - \Pi_{child} SDAG' - \Pi_{subject} P$;
5. $P \leftarrow P \cup Roots \times \{\langle o, r, i, \text{"d"}\rangle\}$;
6. $P' \leftarrow \sigma_{subject \neq s} P$;
   repeat
7.     $i = i + 1$;
8.     $P' \leftarrow \Pi_{child, object, permission, i, mode} P' \bowtie SDAG'$;
9.     $P \leftarrow P \cup P'$;
10.    $P' \leftarrow \sigma_{subject \neq s} P'$;
11.    until $P' = \varnothing$;
12. return $\sigma_{subject=s} P$;
```

Fig. 5. Function *Propagate()*

In Line 1, we extract from *SDAG* the maximal connected sub-hierarchy *SDAG'*, in which *s* is the sole sink. In our motivating example, this line extracts the hierarchy depicted in Figure 3 from the hierarchy depicted in Figure 1. We set the initial depth of the iteration to 0.

In Line 3, we create a relation *P*, which consists of five columns namely, *subject*, *object*, *right*, *dis*, and *mode*. Values for columns *subject*, *object*, *right*, and *mode* are taken from the corresponding ones in relation *EACM* (as explained in Section 1). Column *dis* represents the distance of the explicit authorization from the subject, and takes its value from the iteration number *i*. Thus, the *dis* value for explicit authorizations is 0, for an authorization inherited from a parent it is 1, and for an authorization inherited from a grandparent it is 2. The first two lines of Table 4 show the result of Line 3 on our motivating example. Before completing the Function *Propagate()*, relation *P* will record all relevant authorizations propagated from all subjects to all other nodes.

In Line 4, we store all unlabelled root subjects of *SDAG'* into a relation called *Roots*. For instance, *Roots* contains $\{S_1, S_6\}$ if applied to our motivating example. In

Line 5, for each root subject with no explicit authorization with respect to *o* and *r*, we insert an additional row into relation *P* to assign it the default authorization. This results in the third and fourth entries in Table 4. In Line 6, we select as *P′* all identified authorizations other than those on the sink node *s*.

In Lines 7 to 11, we iteratively propagate all of the newly identified authorizations to all of the children of the corresponding nodes, stopping when no more nodes exist in *P′*. This involves increasing the distance by 1 (Line 7), copying the authorizations from each node to its children (with the increased distance) (Line 8), inserting the new authorizations into P (Line 9), and re-determining which authorizations still need to be propagated further (Line 10). The remainder of Table 4 shows the result of these lines on our motivating example. Finally, Line 12 selects and returns authorizations that correspond to subject *s*, which are shown in Table 1 for our example.

Table 4. All *read* authorizations on *obj*

subject	object	right	dis	mode
S_2	obj	read	0	+
S_5	obj	read	0	-
S_1	obj	read	0	d
S_6	obj	read	0	d
User	obj	read	1	+
S_3	obj	read	1	+
User	obj	read	1	-
S_3	obj	read	1	d
User	obj	read	1	d
S_5	obj	read	1	d
S_5	obj	read	2	+
S_5	obj	read	2	d
User	obj	read	2	d
User	obj	read	3	+
User	obj	read	3	d

3.3 Computational Analysis

The performance of Algorithm *Resolve()* depends on the structure of the subject hierarchy, on the placement of the explicit authorizations in the explicit access control matrix, and on the choice of subject, object, and right. We will examine the performance in practice in the next section, but here we summarize its asymptotic behavior in the worst case.

Consider first the structure of the subject hierarchy as represented by *SDAG*. Let ***r*** be the number of roots of the graph and let ***n*** be the total number of subjects in the hierarchy. We assume that at most one authorization is explicitly given for every subject-object-right triple; duplicates are meaningless and contradicting authorizations can be assumed to be disallowed. Thus, when selected subjects from *SDAG* are matched with explicit authorizations for a given object and right (Line 3 of Function *Propagate()*), at most one explicit authorization is joined to each subject in the subject hierarchy. Let ***p*** be the number of subjects assigned explicit authorizations for the given object-right pair. Finally, let $\boldsymbol{d}$ be the sum of the path lengths for *all* paths leading from a root or an explicitly authorized subject to the given subject of interest s.

Algorithm *Resolve()* first calls Function *Propagate()*. Lines 1 through 6 take time $O(\boldsymbol{n})$ to select a subset of the subjects, attach the explicit authorizations, and set the defaults in the remaining roots. Each authorization (default or explicit) is then pushed down each path to the node representing the given subject. The loop from Lines 7 though 11 of Function *Propagate()* thus require $O(\boldsymbol{d})$ time in total. Finally relation P contains all these propagated authorizations, but only those associated with the given subject s are returned; this returned relation includes exactly one tuple for each explicit authorization and at most one tuple for each root. In summary, Function *Propagate()* takes time $O(\boldsymbol{n}+\boldsymbol{d})$ and returns a structure of size $O(\boldsymbol{p}+\boldsymbol{r})$.

The remainder of Algorithm *Resolve()* repeatedly examines subsets of the relation *allRights*, and thus each line requires time at most $O(\boldsymbol{p}+\boldsymbol{r})$. Thus the total time for executing Algorithm *Resolve()* is $O(\boldsymbol{p}+\boldsymbol{r}+\boldsymbol{n}+\boldsymbol{d})$. Clearly $\boldsymbol{p}+\boldsymbol{r}$ is $O(\boldsymbol{n})$, so the complexity can be stated more simply as $O(\boldsymbol{n}+\boldsymbol{d})$. Unfortunately, since the number of paths in a directed acyclic graph can grow exponentially in the number of nodes in the graph, $\boldsymbol{d}$ is $O(\boldsymbol{n}2^{\boldsymbol{n}})$ in the worst case. We shall see that in practice, however, the algorithm is typically much better behaved, as subject hierarchies seldom contain the repeated diamond patterns that cause the number of paths to explode.

4 Experiments

We tested our algorithm first on synthetic data. We constructed several random complete directed acyclic graphs. In particular, KDAG(n) includes n nodes, one of which is a root and one of which is a sink, and $\binom{n}{2}$ edges (an edge between every pair of nodes), directed in such a way as to prevent cycles. Thus such graphs contain many more paths than would be expected in typical applications, and constitute good stress tests for the algorithm.

We executed our algorithm on random KDAGs of three different sizes. For each graph, we assigned explicit authorizations to subjects at random, choosing subjects proportionally to the number of members. In particular, 0.5% to 10.0% of the graph's *edges* were selected at random and their source nodes were assigned explicit authorizations. We then measured the CPU time for computing the result of the Function *Propagate()* (that being the dominant part of the algorithm) for each of the resulting SDAG-EACM pairs, and averaged over 20 random repetitions with the same parameters. Our experiments show that for small authorization rates (which often occur in practical cases), the running time is linearly proportional to the authorization rates (see Figure 6).

We also evaluated our algorithm on the subject hierarchy extracted from an installation of Livelink, Open Text's enterprise content management system[1]. In Livelink, groups can be arbitrarily structured and nested to arbitrary depth. In the environment we tested, the subject hierarchy has over 8000 nodes and 22,000 edges. There are 1582 sinks (individual users), each of which represents a real-world sample for our experiments. The depths of the induced sub-graphs range from 1 to 11.

[1] http://www.opentext.com/

We measured the time of our algorithm for each of the sinks in the Livelink subject hierarchy, using an authorization rate of 0.7% of the edges as above. The results are presented in Figure 7(a), plotting the CPU time as a function of ***d***, the sum of the lengths of all paths from explicit and default authorizations to the selected sink.

Figure 7(a) also compares the execution time of the *Resolve()* algorithm to that of the *Dominance()* algorithm, presented in Chinaei and Zhang's work [2]. The latter algorithm was designed to evaluate the $D^{-}LP^{-}$ strategy as efficiently as possible under the assumption that there are relatively few explicit authorizations (i.e., that the authorization rate is low). Thus the comparison sheds some light on the overhead imposed by adopting a unified conflict resolution algorithm. It is important to note that the *Dominance()* algorithm is dependent on the placement of negative authorizations whereas the *Resolve()* algorithm is not. To account for this, we calculated the average of three trials for each data point for the *Dominance()* algorithm: one where 1% of the explicit authorizations are negative, one where half of them are negative, and one where all explicit authorizations are negative.

The *Dominance()* algorithm is occasionally very fast due to visiting an early negative authorization in the hierarchy, but it is not as efficient as *Resolve()* for objects that have few negative authorizations. Figure 7(a) shows that the run time for the *Dominance()* algorithm can fall anywhere below the time for the *Resolve()* algorithm, and occasionally it can be higher. On average, over all graph sizes and shapes in these experiments, *Resolve()* required 1260 ms to compute whether or not a sink subject was authorized to access an object, whereas the *Dominance()* average is 920 ms. Thus the flexibility to compute the value for any strategy comes at a cost of 27% overhead.

Finally, Figure 7(b) restates the behavior of Algorithm *Resolve()* against the number of nodes in the sub-graph rather than the total length of all paths in the sub-graph. The results show that graphs with very many subjects do not necessarily require much more time to resolve than do small graphs. From this data we conclude that it is unlikely that the asymptotic worst case performance will be problematic in practice.

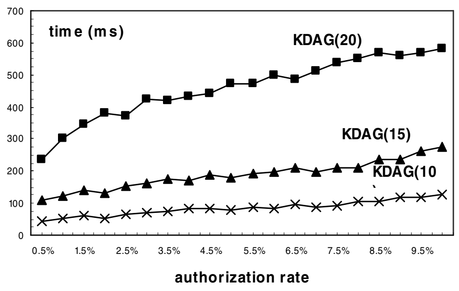

Fig. 6. Function *Propagate()* on synthetic data

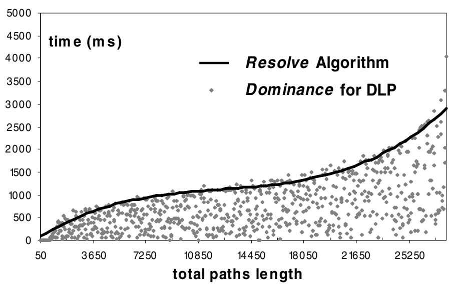

Fig. 7(a). Algorithms *Resolve()* and *Dominance()* on Livelink data

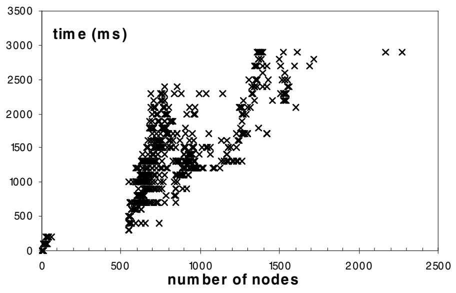

Fig. 7(b). Total paths lengths vs. number of nodes

5 Related Work

Bertino et al. propose an authorization mechanism for relational models in which conflicts are mainly resolved based on "the negative authorization takes precedence" policy [1]. They also introduce the concept of weak and strong authorizations, which is equivalent to using our combined strategy instance $D^{-}LP^{-}$.

Jajodia et al. use Datalog programs to model access controls of hybrid authorizations with a wide range of conflict prevention/resolving policies [7]. Their modeling stores the raw authorizations and computes the effective authorizations for a <*subject, object*> pair in time linear to the size of the Datalog program (rules and ground facts). However, their ground facts include the transitive closure of the subject hierarchy (which cannot be computed in linear time) plus all the raw authorizations. The potentially large number of ground facts implies that even a linear time solution may not be efficient in practice. To answer access control queries efficiently, they suggest materializing the entire effective access control. The accessibility check for a given <*subject, object*> pair is thus equivalent to checking the materialized effective access control table (constant time). However, considering the formidable size of the effective access controls, which is the product of the number of objects and the number of subjects, this approach is not practical for very large systems. Moreover, the materialized effective access controls are not self-maintainable with respect to updating the explicit authorizations, and even a slight update to the explicit authorizations could trigger a drastic modification to the effective ones, making the maintenance task very expensive.

Some existing solutions for computing effective authorizations assume that the explicit authorizations are propagated on tree-structured data [4, 12, 14, 15]. This trivializes conflict resolution since there is only one path between any ancestor and a leaf. Moreover, the number of ancestors for a leaf is bounded by the depth of the tree, which is usually a small value in real world data [11]. Unfortunately, real world subject hierarchies are mostly DAG-structured rather than trees: the UNIX file system allows a user to be member of several groups at the same time, and in role-based access control systems, a user can be assigned several roles and each role can be assigned to multiple parent roles [5]. When explicit authorizations are propagated on a

DAG subject hierarchy, a sink subject potentially has all subjects as its ancestors, and each ancestor may have several paths reaching to that sink. Therefore, none of the approaches for tree-structured data are appropriate in this setting.

Cuppens et al. propose a conflict resolution model for documents containing sensitive information [3]. They address the problem of downgrading the classification of these documents when their contents become obsolete. Their approach is to impose a strict order of preference between rules and does not include any hierarchy among subjects.

Koch et al. provide a systematic graph-based conflict detection and resolution algorithm based on two properties namely, *rule reduction* and *rule expansion* [9]. Using these properties, they transform a conflicting graph into a conflict-free one. However, their approach is applicable only to the rules that are related to one another, whereas our approach addresses independent policies.

Finally, our approach is also different from the combining algorithms in XACML [12], in which the resolution model relies on the data hierarchy rather than the subject hierarchy.

6 Conclusions

In this work, we have extended Chinaei and Zhang's conflict resolution framework for hybrid authorizations, by investigating legitimate combinations of a variety of popular conflict resolution policies. Our framework includes a variety of resolution models to support both closed and open systems, as well as both restricted and relaxed applications. Most importantly, we have enhanced this framework with a unified parameterized algorithm in which a wide range of combined strategy instances are simultaneously supported. This framework inspires access control system providers to separate the conflict resolution component from the system itself. With such an approach, system buyers no longer need to replace the whole system when they decide to change the access control policy. Instead, security administrators can trigger a chosen strategy, among many, without needing to reinstall the whole system. Moreover, access control system providers can sell the same system to all users, regardless of their specific access control needs.

We have experimented with different sets of data layouts. Our pilot experiments show that our algorithm incurs little overhead and performs well when the explicit authorization rate is below 10%, which is appropriate for most practical applications.

We propose several directions to continue this work. First, since there are many nodes in the subject hierarchy that are ancestors of several sinks, it would significantly improve the performance of the algorithm if the derived authorizations of such nodes stored in a cache for later uses. Second, in this work we exploit the subject hierarchy only. It is important to support mixed hierarchy of subjects and objects. Third, our framework can be augmented to support three different propagation modes: when propagating an authorization along a path and meeting another authorization on that path, either the first, the second, or both authorizations could be propagated further. Fourth, we suggest to enhance the framework by adding other access control constraints such as separation of duties and conflict of interests [8, 13]. Lastly, it is interesting to optimize the *Resolve()* algorithm for special purposes of covering a smaller subset of combined strategies for applications in which performance is more important than complete flexibility.

Acknowledgments

We gratefully acknowledge the Natural Sciences and Engineering Research Council of Canada, Communications and Information Technology Ontario, Open Text Corporation, the Mathematics of Information Technology and Complex Systems, and the University of Waterloo for their financial support.

References

1. Bertino, E., Jajodia, S., Samarati, P.: A flexible authorization for relational data management systems. ACM Transactions on Information Systems 17(2), 101–140 (1999)
2. Chinaei, A.H., Zhang, H.: Hybrid authorizations and conflict resolution. In: Jonker, W., Petković, M. (eds.) SDM 2006. LNCS, vol. 4165, pp. 131–145. Springer, Heidelberg (2006)
3. Cuppens, F., Cholvy, L., Saurel, C., Carrere, J.: Merging Security Policies: Analysis of a Practical Example. In: Proceedings of the 11th Computer Security Foundations Workshop, pp. 123–136 (1998)
4. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P.: A fine-grained access control system for XML documents. ACM Transaction on Information and System Security 5(2), 169–202 (2002)
5. Ferraiolo, D.F., Kuhn, D.R.: Role Based Access Control. In: Proceeding of the 15th NIST-NCST National Computer Security Conference, pp. 554–563 (1992)
6. Harrison, M.A., Ruzzo, W.L., Ullman, J.D.: Protection in Operating Systems. Communications of ACM 19(8), 461–471 (1976)
7. Jajodia, S., Samarati, P., Sapino, M.L., Subrahmanian, V.S.: Flexible Support for Multiple Access Control Policies. ACM Transactions on Database Systems 26(2), 214–260 (2001)
8. Joshi, J., Bertino, E., Sahfiq, B., Ghafoor, A.: Dependencies and Separation of Duty Constraints in GTRBAC. In: Proceeding of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT ’03), pp. 51–64. ACM Press, New York (2003)
9. Koch, M., Mancini, L.V., Parisi-Presicce, F.: Conflict Detection and Resolution in Access Control Specifications. In: Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures, pp. 223–237 (2002)
10. Lampson, B.W.: Protection. In: Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems, pp. 437–443 (March 1971)
11. Mignet, L., Barbosa, D., Veltri, P.: The XML Web: A First Study. In: Proceedings of the International World Wide Web Conference, pp. 500–510 (2003)
12. Moses, T.: eXtensible Access Control Markup Language Version 2.0, Technical Report, OASIS (February 2005)
13. Nyanchama, M., Osborn, S.L.: The Role Graph Model and Conflict of Interest. ACM Transaction on Information Systems Security 2(1), 3–33 (1999)
14. Yu, T., Srivastava, D., Lakshmanan, L.V.S., Jagadish, H.V.: Compressed Accessibility Map: Efficient Access Control for XML. In: Bressan, S., Chaudhri, A.B., Lee, M.L., Yu, J.X., Lacroix, Z. (eds.) CAiSE 2002 and VLDB 2002. LNCS, vol. 2590, pp. 478–489. Springer, Heidelberg (2003)
15. Zhang, H., Zhang, N., Salem, K., Zhuo, D.: Compact Access Control Labeling for Efficient Secure XML Query Evaluation. In: Proceedings of the 2nd International Workshop on XML Schema and Data Management (2005)