

**University of Western Sydney**

**School of Computing and Information Technology**



# Delegatable Authorization Program and Its Application

Chun Ruan

chun@cit.uws.edu.au

MAY 2003

TECHNICAL REPORT NO. CIT/20/2003

# Delegatable Authorization Program and Its Application

*Chun Ruan and Vijay Varadharajan and Yan Zhang*  
School of Computing and Information Technology  
University of Western Sydney  
Penrith South DC, NSW 1797 Australia  
E-mail: {chun,vijay,yan}@cit.uws.edu.au

## Abstract

Data protection is a significant issue in any secure information systems. In this paper, we present a decentralized authorization delegation model in which users can be delegated, granted or forbidden some access rights. This security model is formulated as an extended logic program, and the detailed considerations of how to evaluate the semantics of the program is given. In particular, the conflicting problem is addressed and a resolution method based on the underlying delegation relations and hierarchical structures of subjects, objects and access rights is presented. Finally, as an application, we show how this framework can support different electronic consent models within the context of health care.

**Key words:** information security, authorization, access control, logic programming

## 1 Introduction

Data protection is a significant issue in any secure information systems. Proper access control models are needed to protect the information privacy and security. Discretionary access control model has long been a widely accepted and used model in the real world. One of key issues for discretionary access control is related to the authorization administration policy, which refers to the function of granting and revoking authorizations. Two major administration policies are centralized and decentralized administration. With centralized administration, only one central authorization unit may have the privilege to grant and revoke authorizations. It usually reflects the situation in enterprises and authorization remains easy to survey. But it is rather inflexible, since usually no individual can know what controls are appropriate for every object when the amount of objects is very large. With decentralized administration, on the other hand, multiple subjects may have the privilege to grant and revoke authorizations, and the administration privilege can usually be delegated between subjects. Decentralized authorization usually follows the ownership paradigm; i.e. every creator of an object possesses all possible access rights to access them, and can grant and delegate authorizations on this object to other subjects. It is rather flexible and apt to the particular requirements of individual subjects. Most commercial DBMSs adopt such decentralized authorization. Nevertheless, the authorizations become

more difficult to control since multiple subjects can grant and revoke authorizations, and the problem of cascading and cyclic authorization may arise. Further more, when both positive and negative authorizations are allowed in a decentralized authorization model, conflict problem becomes crucial since multiple administrators greatly increase the chance of conflict and cyclic authorizations may lead to unexpected situations. Currently most existing database systems do not support authorization delegations and negations at the same time.

On the other hand, supporting inheritance of authorizations can often effectively simplify the specification and evaluation of authorizations, especially in some application domains where inheritance is an important feature, such as object-oriented databases. For example, a member of a group usually can inherit all the access rights granted to the group. If someone is granted to write a directory, it is often implied that he/she should be able to read the directory and all files in that directory. When authorization inheritance is under consideration, the problem of conflict becomes more complex since a lot of implicit conflicts among different types of authorizations may arise.

This paper presents a framework which supports authorization delegations, authorization negations and authorization inheritance. A conflict resolution method based on the underlying delegation relation is proposed which gives higher priorities to the predecessors to achieve the controlled delegation. For conflicts whose grantors are not delegation-connected, the more specific-take-precedence principle is used to support exception. To take advantage of strong expressive and reasoning power of logic programming, we will develop our framework based on extended logic programs [4], which supports both negation as failure and classical negation. The extended logic programs, which is formalised based on nonmonotonic reasoning semantics, has strong expressive power in the sense that it can deal directly with incomplete information in reasoning. Since the incomplete information is a common issue in the security world, many access control policies are easier to specify in extended logic programs. For example, if we want to express negation by default, like  $s$  is denied to read the file  $F$  if  $s$  is not granted to read it, the negation as failure (weak negation) is often the most direct way to express this intention. On the other hand, in many situations, classical negation (strong negation) is useful to explicitly specify that something is forbidden.

In our framework, authorization rules are specified in a delegatable authorization program (DAP) which is an extended logic program associated with different types of partial orderings on the domain, and these orderings specify various inheritance relationships among subjects, objects and access rights in the domain. The semantics of a DAP is defined based on the stable model concept and the conflict resolution is achieved in the process of model generation of the DAP. As an application, we show how this framework can support different electronic consent models within the context of health care.

## 2 Syntax of DAP

Our language  $\mathcal{L}$  is a many-sorted first order language, with four disjoint *sorts* S, O, A, and T for subject, object, access right and authorization type respectively. Variables are denoted by strings starting with underscore, and constants by ordinary strings. Three

partial orders  $<_S, <_O, <_A$  are defined on sorts  $S, O, A$  respectively, which represent the inheritance hierarchical structures of subjects, objects and access rights. We use  $\#$  in  $S$  to denote the security administrator, and it is not comparable to any subjects in  $S$  w.r.t.  $<_S$ . In the constant set of authorization types  $T = \{-, +, *\}$ ,  $-$  means *negative*,  $+$  means *positive*, and  $*$  means *delegatable*. A negative authorization specifies the access that must be forbidden, while a positive authorization specifies the access that must be granted. A delegatable authorization specifies the access that must be delegated as well as granted. That is,  $*$  means  $+$  plus administrative privilege on the access.

The predicate set  $P$  consists of a set of ordinary predicates defined by users, and one built-in predicate symbol for delegatable authorization, *grant*. *grant* is a 5-term predicate symbol with type  $S \times O \times T \times A \times S$ . The first argument is the *grantee*, the second argument is the *object*, the third argument is the *authorization type*, the fourth argument is the *access right*, and the fifth argument is the *grantor* of this authorization. Intuitively,  $grant(s, o, t, a, g)$  means  $s$  is granted by  $g$  the access right  $a$  on object  $o$  with authorization type  $t$ . *grant* is called *authorization predicate*.

A *term* is either a variable or a constant. Note that we prohibit function symbols in our language. An *atom* is a construct of the form  $p(t_1, \dots, t_n)$ , where  $p$  is a predicate of arity  $n$  in  $P$  and  $t_1, \dots, t_n$  are terms. A *literal* is either an atom  $p$  or the negation of the atom  $\neg p$ , where the negation sign  $\neg$  represents classical negation. Correspondingly, a literal that has an authorization predicate is called an *authorization literal*. Two literals are *complementary* if they are of the form  $p$  and  $\neg p$ , for some atom  $p$ . A *rule*  $r$  is a statement of the form:

$$b_0 \leftarrow b_1, \dots, b_k, \text{not } b_{k+1}, \dots, \text{not } b_m, m \geq 0$$

where  $b_0, b_1, \dots, b_m$  are literals, and *not* is the negation as failure symbol. The  $b_0$  is the *head* of  $r$ , while the conjunction of  $b_1, \dots, b_k, \text{not } b_{k+1}, \dots, \text{not } b_m$  is the *body* of  $r$ . Obviously, the body of  $r$  could be empty. We sometimes use  $Head_r$  and  $Body_r$  to denote the head and body of  $r$  respectively. Correspondingly, when  $b_0$  is an authorization literal, the rule is called *authorization rule*. A *Delegatable Authorization Program*, DAP, consists of a finite set of rules. A term, an atom, a literal, a rule or program is *ground* if no variable appears in it.

**Example 1** Let  $S = \{\#, s_1, s_2; s_1 <_S s_2\}$ ,  $O = \{o_1, o_2; o_1 <_O o_2\}$ ,  $A = \{write, read; write <_A read\}$ , then the following is an example DAP II:

$$\begin{aligned} r_1 &: dba(s_1) \leftarrow \\ &(/* s_1 \text{ is a dba */}) \\ r_2 &: \neg secret(o_1) \leftarrow \\ &(/* o_1 \text{ is not secret */}) \\ r_3 &: secret(o_2) \leftarrow \\ &(/* o_2 \text{ is secret */}) \\ r_4 &: grant(s_1, o_2, *, write, \#) \leftarrow \\ &(/* The administrator grant } s_1 \text{ to write and grant on } o_2 */) \\ r_5 &: grant(s_2, o_2, -, write, s_1) \leftarrow \\ &(/8 s_1 \text{ grant } s_2 \text{ "not write" on } o_2 */) \\ r_6 &: grant(s_2, o_2, -, write, \#) \leftarrow secret(o_2), \text{not } dba(s_2) \end{aligned}$$

(/\* If  $o_2$  is secret,  $s_2$  is not a dba, then  $s_2$  can not write on  $o_2$ . The grantor is the administrator.\*/) ■

### 3 Considerations of the semantics

To develop a formal semantics of a DAP, we should consider three aspects that will affect the process of evaluating a DAP: delegation correctness, authorization propagation along the hierarchies of subjects, objects and access rights, and conflict resolution.

#### Delegation correctness

We say that an authorization set is *delegation correct* if it satisfies the following two conditions: (a) subject  $s$  can grant other subjects an access right  $a$  over object  $o$  if and only if  $s$  is the security administrator ‡ or  $s$  has been granted  $a$  over  $o$  with a delegatable type \*; (b) if subject  $s$  receives a delegatable authorization directly or indirectly from another subject  $s'$  on some object  $o$  and access right  $a$ , then  $s$  cannot grant  $s'$  any further authorization on the same  $o$  and  $a$  later on.

Intuitively, the Condition (a) comes from the system assumption and the feature of the delegatable authorization type, while condition (b) says that, for a given object and an access right, a subject is forbidden from giving authorizations to its grantors, the grantors' grantors, and so on. This restriction makes delegation more reasonable and can solve the problem that exists in most conflict resolution methods when authorization delegation is considered. This issue has been explored in detail in our previous paper [6]. To illustrate the problem, it is sufficient to consider a simple example: Suppose  $s$  is delegated by  $s'$  the right to grant to others for *read* right of a file  $F$ , and then, unexpectedly,  $s$  may deny  $s'$  to read  $F$  by granting  $s'$  *not to read*  $F$  later on. Hence, we will require the authorization set derived by a DAP be delegation correct.

#### Authorization propogations

Rule based authorization specification allows implicit authorizations to be derive from the authorization set, and hence this can greatly reduce the size of explicit authorization set. In our model, we support the implicit authorizations by permitting rules inheritance. From the object-oriented feature, rules defined for a given object are automatically inherited by its sub-objects. In particular, our model supports three dimensional inheritance along hierarchies of subjects, objects and access rights. For instance, if a group is authorized to write a directory  $D$ , then by inheritance, we can derive the following implicit authorizations: all members of the group can write  $D$ , the group can write all files and subdirectories under  $D$ , the group can also read  $D$  and therefore can read all files and subdirectories under  $D$ , and all members of the group can read  $D$  as well and therefore can read all files and subdirectories under  $D$ , and so on.

### Conflict resolution

Since both positive and negative authorizations are acceptable in our framework, conflicts among authorizations may arise. And since we also allow implicit authorizations, implicit conflicts may occur, which makes the problem more complicated. The basic idea of our method of solving conflicts is outlined as follows.

- Solving conflicts using delegation relation. If subject  $s$  delegate subject  $s'$  directly or indirectly an authorization on object  $o$  and access right  $a$ , then, when a conflict w.r.t  $o$  and  $a$  occurs, the authorization from  $s$  (i.e.  $s$  is the grantor) will always override the one from  $s'$ .
- Solving conflicts from grantee inheritance. If the grantors of two conflicting authorization are identical, we consider grantees of the authorizations next. We use the more *specific-take-precedence* principle. If  $s <_S s'$ , then the authorization with  $s'$  as grantee will override the inherited one with  $s$  as grantee.
- Solving conflicts from object inheritance. If both the grantors and grantees are identical, we then consider the object inheritance relation. According to the inheritance hierarchy of objects, If  $o <_O o'$ , then the authorization on  $o'$  will override the inherited one on  $o$ .
- Solving conflicts from access right inheritance. When all grantors, grantees and objects are identical, we finally consider the access right inheritance relation. According to the inheritance hierarchy of access rights, if  $a <_A a'$ , then the authorization on  $a'$  will override the inherited one on  $a$ .
- Conflicts that are unsolvable. If all above policies fail to solve the conflict between the two authorizations, we treat this conflict as *unsolvable* in our framework.

**Example 2** Suppose  $s_1 < s_2, o_1 < o_2$ . Consider two authorizations  $l_1 : grant(s_1, o_1, *, read, \#)$  and  $l_2 : grant(s_2, o_1, -, read, s_1)$ . From inheritance of subjects, we can get  $l_3 : grant(s_2, o_1, *, read, \#)$ .  $l_3$  and  $l_2$  are conflicting and  $l_3$  will override  $l_2$  because grantor  $s_1$  in  $l_2$  get its *grant* right from grantor  $\#$  in  $l_3$ , although  $l_2$  is more specific than  $l_3$ .

Consider another two authorizations  $l_1 : grant(s_1, o_2, +, read, \#)$  and  $l_2 : grant(s_2, o_1, -, read, \#)$ . From inheritance of subjects and objects, we can get two conflicting authorizations  $l_3 : grant(s_2, o_2, +, read, \#)$  and  $l_4 : grant(s_2, o_2, -, read, \#)$ . Since their grantors are identical, we consider grantee next. Since  $l_3$  and  $l_4$  come from  $l_1$  and  $l_2$  which have  $s_1$  and  $s_2$  as their grantees respectively, and  $s_1 < s_2$ ,  $l_4$  will override  $l_3$ .

Now we consider the third case. Let  $l_1 : grant(s_1, o_1, +, read, \#)$  and  $l_2 : grant(s_1, o_2, -, read, \#)$  be two authorizations. From inheritance of objects, we can get authorization  $l_3$  from  $l_1$ ,  $l_3 : grant(s_1, o_2, +, read, \#)$ , which conflicts with  $l_2$ . Since  $l_3$  comes from  $l_1$  that has the same grantor and grantee with  $l_2$ , we consider objects next. Since  $l_1$  has  $o_1$  as object,  $l_2$  has  $o_2$  as object and therefore more specific,  $l_2$  will override  $l_3$ .

■

### Stable model semantics

There are two leading semantics for extended logic programs: well-founded semantics and stable model semantics [4]. In our approach, we will develop a stable model based semantics for our DAPs because stable model semantics provides a more flexible manner to deal with contradicted and incomplete information, and hence seems more suitable for our purpose of handling authorization conflicts.

## 4 e-Consent Application

In this section we show how our model can apply to the access control within the context of e-Consent in health care system.

Information privacy, which is the interest individuals have in exercising control over information about themselves. Information privacy is seriously challenged by health care practices, particularly in a coordinated care setting. Health's information technology needs to support confidential consumer and service provider interactions. This will require new uses and disclosure of patient data. Moreover, it will increasingly rely on electronic transmission of patient data. Consideration therefore needs to be given to the need for electronic forms of patient consent, referred to as e-consent.

Information security concerns that arise in the context of e-consent relate to: personal health data, consents and denials given by patients, the capacity to create a consent or consent delegation. Various forms of patient consent should be supported.

### Consent delegation

A patient can generally give a health care professional the capacity to further disclosure his health data, because of the need of cooperated care, but denies consent as follows: to the disclosure of particular information; to the disclosure to a particular party or category of parties; to the disclosure for a particular purpose. For example, a patient John delegate consent to his doctor (D) in a hospital, but denies disclosure to his family GP (FGP). In our model, this can be expressed as:

$$\text{grant}(D, \text{alldata}, *, \text{access}, \text{John}) \leftarrow$$
$$\text{grant}(FGP, \text{alldata}, -, \text{access}, \text{John}) \leftarrow$$

According to our conflict resolution method, the authorizations from John will have higher priority than the authorizations from his Doctor, since the doctor gets the capacity to grant from John. Therefore, no matter John's doctor grants his family GP to access his health data or not in the future, his family GP can not access because of his negative authorization.

### General consent

This is a 'blanket consent' given by a patient for any health care professional to access any of their health information for any purpose relating to the consumer's care. Thus

in a future episode care, a health worker does not need to ask for consent when looking at details of past episodes of care. For example a patient John give consent to any care professional for accessing any of his health data for care purpose. In our model, defining CP less than all the other specific classes of care professionals , such as GP, nurse or consultant in the subject hierarchy, then the consent requirement can be expressed by the following:

$$\text{grant}(CP, \text{alldata}, +, \text{access}, \text{John}) \leftarrow$$

This authorization will be automatically propagated to all the subordinate care professionals according to our rule inheritance semantics.

#### General consent with specific denials

In this case, a patient provides a general consent, but denies consent as follows: to the disclosure of particular information; to the disclosure to a particular party or category of parties; to the disclosure for a particular purpose. For example, a patient John provides a general consent to a health care professional (CP) for accessing his health data, but precludes the disclosure to his family GP (FGP). In our model, defining  $CP < FGP$ , then the consent requirement can be expressed as:

$$\text{grant}(CP, \text{alldata}, +, \text{access}, \text{John}) \leftarrow$$

$$\text{grant}(FGP, \text{alldata}, -, \text{access}, \text{John}) \leftarrow$$

The first rule will be inherited by FGP, but according to our conflict resolution policy, the second rule will override the inherited one, since the second one is more specific.

#### General denial with specific consents

In this case, a patient denies all access to their health data, except for circumstances tht are the subject of specific consent: to the disclosure of particular information; to the disclosure to a particular party or category of parties; to the disclosure for a particular purpose. For example, a patient John provides a general denial to a health care professional (CP) for accessing his health data, but precludes to his family GP (FGP). In our model, defining  $P < FGP$ , then the consent can be expressed as:

$$\text{grant}(CP, \text{alldata}, -, \text{access}, \text{John}) \leftarrow$$

$$\text{grant}(FGP, \text{alldata}, +, \text{access}, \text{John}) \leftarrow$$

The first rule will be inherited by FGP, but according to our conflict resolution policy,

the second rule will override the inherited one, since the second one is more specific.

### General denial

In this case, a patient provides a 'blanket denial', and denies consent for information to be used in future circumstances. Here, in each new episode of care, a new consent would need to be obtained. For example, a patient John generally denies the access to his treatments for STD. In our model, this can be expressed as:

$$\text{grant}(CP, STD, -, \text{access}, \text{john}) \leftarrow$$

## 5 Future work

There are several interesting issues that worth further exploring. First, we are considering to extend our current model by combining dynamic inheritance relations among subjects, objects and access rights in authorization specification and evaluation. Second, we will develop a dynamic prioritized conflict resolution method which can support different policies in terms of various properties of subjects, objects and access rights. Finally, we will also implement a prototype of our framework based on logic programming technique.

## References

- [1] M.Abadi, M.Burrows, B.Lampson, G.Plotkin, A calculus for access control in distributed systems. *ACM Trans. on programming languages and systems*, 15(4):706-734, 1993.
- [2] 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.
- [3] J.Crampton, G.Loizou, G.O'Shea A logic of access control. *The Computer Journal*, vol.44, pp.54-66, 2001.
- [4] M.Gelfond and V.Lifschitz, Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:pp365-385, 1991.
- [5] S. Jajodia, P. Samarati, and V.S. Subrahmanian, A logical language for expressing authorizations. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, pp 31-42, 1997.
- [6] C. Ruan and V. Varadharajan, Resolving conflicts in authorization delegations. *Proceedings of the 7th Australasian Conference on Information Security and Privacy*, pp 271-185, 2002.
- [7] T. Woo and S. Lam, Authorization in distributed systems: a formal approach. *Proceedings of IEEE on Research in Security and Privacy*, pp33-50,1992.