World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:2, No:1, 2008

A Formal Implementation of Database Security

Yun Bai

Abstract**—** This paper is to investigate the impplementation of security mechanism in object oriented database system. Formal methods plays an essential role in computer security due to its powerful expressiveness and concise syntax and semantics. In this paper, both issues of specification and implementation in database security environment will be considered; and the database security is achieved through the development of an efficient implementation of the specification without compromising its originality and expressiveness.

Keywords**—** database security, authorization policy, logic based specification

Open Science Index, Mathematical and Computational Sciences Vol:2, No:1, 2008 publications.waset.org/7076/pdf

I. Introduction

Authorization specifications have long been an important issue in computer system security. Specifically, in a database system, the function of the authorization is to control access to the system. It only allows the authorized users performing authorized operations on the data resource of the system.

A variety of logic authorization specification approaches have been proposed. Woo and Lam's [7] proposed a formal approach using default logic to represent and evaluate authorizations. However, they did not consider the constraints of the access control of the system. Hence it is not clear how to judge a policy base is legitimate or not with respect to the system restriction. This approach is not suitable for object oriented database security specification since it's hard to capture the hierarchical structure and the constraints of the database. In [2], a general framework on a logic formalism was proposed to model discretionary, mandatory access control and role-based access control models. The syntax and the semantics of the framework were given and also some example applications were presented. This work was mainly used for the analysis and the comparison of some existing access control models and their decidability. Whether this general framework can be used to model access control in object oriented database scenario is not clear. Bettini *et al* [3] formalized a rule-based security policy framework which included provisions and obligations, and investigated a reasoning mechanism within this framework in a general database scenario. This work investigated authorization policy with a logic framework from management point of view. The specification of authorization policy itself was not fully investigated and the implementation issue was not discussed.

In summary, most of the current works emphasize enchancing the expressive power of the authorization policies, their evaluation, propagations, reasoning and delegations, while the implementation efficiency for such powerful formalizations are not seriously considered.

This paper is to address high level authorization specifications and its efficient implementation in object oriented database scenario. The rest of the paper is organized as follows. Section 2 specifies object oriented database system; section 3 incorporates authorization rules into database and section 4 maps the specification into a logic programming language and implements the language. Section 5 concludes the paper.

Yun Bai is with School of Computing and Mathematics, University of Western Sydney, NSW 1797, Australia, E-mail: ybai@scm.uws.edu.au

II. Object Oriented Database Specification

The basic building blocks of an object oriented database are **objects** nested in the hierarchical structure; the **relationships** among these objects to indicate how they are related; and the **restrictions** on the data objects to satisfy certain user and system requirements. We propose a logic language $\mathcal{L}^o$ which has a simple yet flexible syntax, it can be suitably used to formalize various features of object oriented database system. In the language, two sorts of objects will be specified: a finite set of object *constants* which represent object instances of the database and a finite set of object *variables* which represent a general framework of objects of the database. Generally, a set of derived data objects is generated by applying some functions to certain existing data objects of the database. To achieve this, we define a finite set of function symbols as *methods* which apply to object(s) to generate new object(s) from the database.

We propose an *object proposition* to represent object constant, object variable and object derived from various methods; an *isa proposition* to capture the hierarchical structure and membership feature of object oriented database; and *constraint proposition* to represent different restrictions on different objects.

For instance, the hierarchical structure and membership feature of an object oriented database can be represented by *isa proposition* as: O **isa member of** C or O **isa subclass of** C. When the details of a data proposition is not interested in the context, we usually use the notation ϕ to denote it. If we use ϕ to denote the general data proposition, a *constraint proposition* can be expressed as: ϕ **if** $\phi_1, \cdots, \phi_k$. it states that if $\phi_1, \cdots, \phi_k$ are true, then ϕ is true. That is, ϕ is a derived data object from data objects $\phi_1, \cdots, \phi_k$.

With the three propositions, an object oriented database can be formally defined as a finite set of **object propositions** specifying different kinds of data objects, a finite set of **isa propositions** expressing how these objects are related within the database and a finite set of **constraint propositions** restricting the objects to satisfy various user and system requirements.

III. Incorporate Authorization Rules into Database Specification

This section is to extend the database specification to include authorization rules in order to control access to the database

Open Science Index, Mathematical and Computational Sciences Vol:2, No:1, 2008 publications.waset.org/7076/pdf

system. A specific authorization rule is to define which subject holding what kind of access right for which object. To properly define our authorization rules, we need to define a finite set of *subjects*, a finite set of *access rights* and a finite set of *objects*. In our specification, a data object can act as either a subject or an object in terms of access control. All three sets consist *constants* to represent specific instances and *variables* to represent general items. An authorization fact that a subject S has access right R for object O is represented using a ground formula $holds(S, R, O)$. A ground formula is a formula free of variables. We use lower case letters for variables and capital letters for constants.

We define an *access fact* is an atomic formula $holds(s, r, o)$ (or $holds(s, r,\ o|f)$, where $o|f$ indicates a method associated with object o.) or its negation. An *access fact expression* is defined as follows: (i) each access fact is an access fact expression; (ii) if ψ is an access fact expression and ϕ is an isa or object proposition, then $\psi \wedge \phi$ is an access fact expression; (iii) if ψ and ϕ are access fact expressions, then $\psi \wedge \phi$ is an access fact expression.

Based on the above definition, we define an *access proposition* as: ψ **implies** ϕ **with absence** γ. This proposition says that ϕ is true if ψ is true under the condition that γ is not presented.

A special form of the above access proposition occurs when γ is empty. In this case, we can rewrite it as: ψ **provokes** ϕ. This is viewed as a *causal* or *conditional* relation between ψ and ϕ. Furthermore, when ψ is also empty, we rewrite it as: **always** ϕ, which specifies a condition that ϕ should always be true.

Now we can define an *authorization rule base*. It is defined as a finite set of **data propositions** representing various objects and subjects of the database; a finite set of **access propositions** specifying access permits of the data objects; and a finite set of **constraints** being complied by the data objects regulated by users or systems.

For example, some authorization rules for accessing database can be specified as:

$$\begin{aligned} holds(s, r, c) \wedge o \ \textbf{isa subclass of}\ c \\ \textbf{implies}\ holds(s, r, o) \\ \textbf{with absence}\ \neg holds(s, r, o), \end{aligned} \tag{1}$$

and

$$\begin{aligned} holds(s, r, c) \wedge o \ \textbf{isa member of}\ c \\ \textbf{implies}\ holds(s, r, o) \\ \textbf{with absence}\ \neg holds(s, r, o), \end{aligned} \tag{2}$$

The above two propositions express the hierarchy and membership authorization inheritance properties.

The proposed high level specification language is expressive enough to represent authorizations in object oriented database environment. Within this specification, constraints, causal and inherited authorizations as well as general default authorizations can be properly justified.

IV. Logic Program and Implementation

We propose to use logic programming approach to implement this high level specification language $\mathcal{L}^o$. The major reason is that logic programming has a powerful declarative semantics which can be used to express default reasoning in authorization rules. On the other hand, the proof procedure associated with logic programs is also well developed and can be employed in the implementation of the language.

The basic idea is that for an arbitrary authorization rule base, each proposition in this rule base is translated into a logic program rule; then we need to prove that there exists a one-to-one correspondence between the model of the rule base and the answer set of the translated logic program. By restricting every formula ϕ (ψ or γ) occurring in the propositions to be a literal L, our propositions can be translated into logic program rules (3) - (5) respectively as follows:

$$Holds(L', s) \leftarrow Holds(L_1, s), notHolds(L_2, s), \tag{3}$$

$$Holds(L', s) \leftarrow Holds(L, s), \tag{4}$$

$$Holds(L, s) \leftarrow \tag{5}$$

(3) is a special rule which includes *negation as failure* **not** to represent default information. (3) states that if L_1 holds in state s, and L_2 does not present in state s, then L holds in state s. (4) and (5) are just direct translations of other normal propositions.

To ensure that there is no semantic loose of this logic program translation, it must show that there is a one-to-one correspondence between the rule base semantics and the logic program semantics. To achieve this result, *answer set semantics* will be used to characterize the translated logic program.

The implementation of the high level specification language consists of the following stages:

(1) Implementing a parser for syntax checking of the high level specification language $\mathcal{L}^o$. The user specifies an authorization rule base by using this high level language. The function of parser is to check if there is any syntactic error in this rule base.

(2) Implementing an automatic translation procedure from the rule base specification to a logic program. The translation method has been described. However, to implement this translation procedure, it is necessary to add one more set of rules that are used to derive persistent facts. In the rule base, the persistence of rule facts are represented by model intersections, i.e. any fact that is in every model of the rule base is viewed as persistent. But in logic program, as a function symbol *Result* is been introduced, the fact which is true in the current state is represented by the rule:

ψ **implies** ϕ **with absence** γ

To carry those persistent fact over the state, a special default rule as the following is needed:

$$Holds(L, Result(T, s)) \leftarrow Holds(L, s), not\neg Holds(L, Result(T, s)),$$

which states that if L holds in state s, and no evidence

to show that L does not hold in state $Result(T, s)$, then L is derived *by default* to be persistent in state $Result(T, s)$.

(3) Implementing a computation mechanism of computing answer set of the translated logic program. As mentioned before, after proving the equivalence between the semantics of rule base and the answer set of the logic program, the answer set of the translated logic program to characterize the original authorization rule base can be used. To implement the answer set computation, proper optimalized strategies may be used to simplify the procedure.

In summary, the complete implementation of the proposed high level language can be described as the following figure:

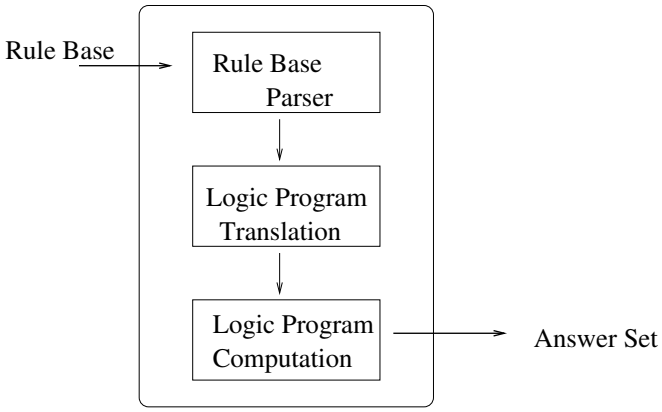


Fig. 1. High level database security language.

V. Conclusion

In this paper, we proposed a logic specification and implementation approach for object oriented database security. The aims of this paper are to propose a formal specification for object oriented database and the authorization rules for accessing it, then to investigate the implementation of the specification for secure database system. So far, the specification part has been completed. We are currently investigating the mapping of the authorization rules to logic programs and the implementation of logic programs.

References

[1] E. Bertino, F. Buccafurri, E. Ferrari and P. Rullo, "A Logic-based Approach for Enforcing Access Control". *Computer Security*, vol.8, No.2-2, pp109–140, 2000.

[2] E. Bertino, B. Catania, E. Ferrari and P. Perlasca, "A Logical Framework for Reasoning about Access Control Models". *ACM Transactions on Information and System Security*, Vol.6, No.1, pp71–127, 2003.

[3] C. Bettini, S. Jajodia, X. S. Wang and D. Wijesekera, "Provisions and Obligations in Policy Management and Security Applications". *Proceedings of the Very Large Database Conference*, pp502–513, 2002.

[4] S. Jajodia, P. Samarati, M.L. Sapino and V.S. Subrahmanian, "Flexible Support for Multiple Access Control Policies". *ACM Transactions on Database Systems*, Vol.29, No.2, pp214–260, 2001.

[5] N. Li, B. Grosof and J. Feigenbaum, "Delegation Logic: A Logic-based Approach to Distributed Authorization". *ACM Transactions on Information and System Security*, Vol.6, No.1, pp128–171, 2003.

[6] L. Wang, D. Wijesekera and S. Jajodia, "A logic-based framework for attribute based access control," *Proceedings of the ACM Workshop on Formal Methods in Security Engineering*, pp45–55, 2004.

[7] T.Y.C. Woo and S.S. Lam, "Authorization in Distributed systems: A Formal Approach". *Proceedings of IEEE Symposium on Research in Security and Privacy*, pp33-50, 1992.

Open Science Index, Mathematical and Computational Sciences Vol:2, No:1, 2008 publications.waset.org/7076/pdf