CHAPTER 33

Authorization Model in Object-Oriented Systems

Keiji Izaki, Katsuya Tanaka, and Makoto Takizawa
Dept. of Computers and Systems Engineering Tokyo Denki University
{izaki, katsu, taki}@takilab.k.dendai.ac.jp

Abstract In object-oriented systems, data and methods of a class are inherited by lower-level classes according to the is-a hierarchy. It is difficult to specify access rules for every class and object, because the system is composed of various types of classes, and objects which are dynamically created and dropped. If access rules on some class could be reused for other classes, the access rules are easily specified. This paper discusses how to inherit access rules in hierarchical structure of classes and objects.

Keywords: Access Control, Inheritance, Object-oriented systems

Introduction

Various kinds of distributed applications (Dittrich *et al.* 1989) are required to be realized in secure information systems. Various kinds of access control models are discussed so far, e.g. basic model (Lampson *et al.* 1971) and lattice-based model (Bell *et al.* 1975, Denning *et al.* 1982). An access rule $\langle s, o, op \rangle$ means that a subject s is allowed to manipulate an object o by an operation op. An access rule which a subject granted can be granted the to another subject in the discretionary model like relational database systems (Oracle *et al.* 1999). In the role-based model (Sandhu *et al.* 1996), a *role* is modeled to be a collection of access rights. A subject is granted a role.

Distributed systems are now being developed according to object-oriented frameworks like CORBA (Object *et al.* 1997). The papers (Dittrich *et al.* 1989, Samarati *et al.* 1997) discuss a *message filter* in an object-oriented system to prevent illegal information flow. The paper (Spooner *et al.* 1989) points out some problems to occur in the inheritance hierarchy, but does not discuss to make the system secure.

The paper (Yasuda *et al.* 1989) discusses the *purpose-oriented* access control model in an object-based system.

The object-oriented system is composed of various kinds of classes and objects which are dynamically created and destroyed. It is cumbersome to specify access rules for all classes and objects. If access rules for a class are inherited by subclasses, access rules are easily specified for classes. We discuss how to inherit access rules on classes and objects structured the *is-a* relation in a discretionary way.

In section 2, we briefly review the object-oriented model. In section 3, we discuss how to inherit access rules in the object-oriented model.

1. OBJECT-ORIENTED MODEL

The object-oriented system is composed of multiple classes and objects. A class c is composed of a set π_c of *attributes* $A_{c1}, ..., A_{cm_c}$ $(m_c \geq 1)$ and a set μ_c of *methods* $op_{c1}, ..., op_{cl_c}$ $(l_c \geq 1)$. An object o is created from the class c by allocating memory area for storing values of the attributes. Let μ_o be a set of methods of o. The methods are inherited from c, i.e. $\mu_o = \mu_c$. The object o is allowed to be manipulated only through methods in μ_o. *o is* referred to as *instance of* the class c.

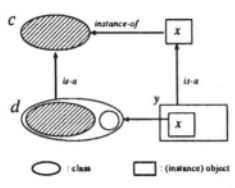

Figure 1 Classes and objects.

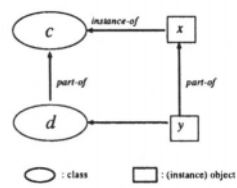

Figure 2 Part-of relation.

Classes and objects are hierarchically structured with *is-a* and *part-of* relations. A new class d is derived from an existing class c, where d inherits attributes and methods from c. Here, d is in an *is-a* relation with c, i.e. d is a *subclass* of c. Additional methods and attributes can be defined for the subclasses. Furthermore, attributes and methods inherited from c can be overridden for d. Figure 1 shows an *is-a* relation between a pair of classes c and d, i.e. d is a subclass of c. d inherits attributes and methods from c. In addition, the object y is in an is-*a* relation with x, i.e. the values of x are inherited by y. $\sigma_x \subseteq \sigma_y$ and $\mu_x \subseteq \mu_y$. Let $y.c$ denote values of y inherited from x. The object y in fact does not have the value of x, in order to reduce the storage space. A class c can be composed of other classes $c_1, ..., c_n$. Here, each class c_i is a *part* or *component* class of c. Let d be a component class of a class c [Figure 2], Let x and y be objects of the classes c and d, respectively.

y is also a component object of x. However, there is neither *is-a* nor *part-of* relation between objects in the traditional systems.

A *manipulation* method manipulates values of attributes in the object. Another one is a *schema* method, by which classes and objects are created and destroyed, e.g. *create object.*

2. INHERITANCE OF ACCESS RULES

2.1. INSTANCE-OF RELATION

First, suppose an object x is created from a class c. An owner of the class c grants a subject an access right to create an object from c. Then, the subject creates an object x from c and is an owner of x. Suppose a set α_c of access rules are specified for c. The object x inherits the access rules α_c from c in addition to the attributes and methods. Here, a set α_x of access rules for the object x is $\{\langle s, x, op\rangle \mid \langle s, c, op\rangle \in \alpha_c$ and op is manipulation method$\}$.

Every subject s granted an access right $\langle c, op\rangle$ is granted $\langle x, op\rangle$ for every objet x of the class c. Only access rules on manipulation methods are inherited by the object x. The owner of x can define additional access rules and can revoke the access rules inherited from c.

There are *class* and *object* access rules. A *class* access rule $\langle s, c, op\rangle$ is specified for a class c. Here, every object x of c inherits the rule $\langle s, x, op\rangle$. If $\langle c, op\rangle$ is revoked from s, $\langle x, op\rangle$ is automatically revoked from s. If a new class rule $\langle s, c, op\rangle$ is specified, $\langle s, x, op\rangle$ is also inherited by every object x of c. The class access rules are allowed to be changed only by the owner of the class c. On the other hand, the object access rules of a class c are inherited by the objects created from c, but can be changed by the owner of the object. In fact, the object access rules of c are copied to the object x while the class rules are maintained in the class c [Figure 4(1)].

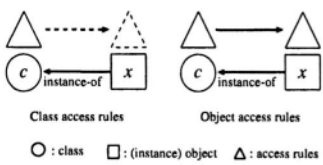

Figure 3 Access rules.

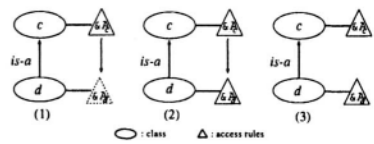

Figure 4 Inheritance of access rules.

2.2. IS-A RELATION OF CLASSES

Suppose a class d is a subclass of a class c as shown in Figure 1. The access rules of c are inherited by d. Let α_c and α_d be sets of access

rules of c and d, respectively. There are following ways on how to inherit access rules α_c from c to d [Figure 4]; (1) the access rules α_c are inherited by d, (2) the access rules α_c are copied to d, and (3) no access rule α_c is inherited by d. In the first case, the access rules inherited by the subclass d depend on c. Values of attributes of c are manipulated through a manipulation method *op*. Here, *op* is also performed on the attribute $d.c$ in the class d. If the access rules in α_c are changed in c, the access rules in d are also changed. If a new rule $\langle s, c, op \rangle$ is specified for c, $\langle s, d, op \rangle$ is automatically *authorized*. The access rules are in fact maintained only in the class c and are not in the subclass d. c is referred to as *home* class of the access rule. Next, let us consider how to inherit access rights on a schema method *op* of the class c. For example, suppose an access rule $\langle s, c, op \rangle$ is specified for the class c. Here, $\langle s, c, op \rangle$ is inherited by d. If d is derived from c, the subject s can create an object from d.

In the second case, the access rules α_c of the class c are copied in the subclass d [Figure 4 (2)]. The access rules of d are independent of c. For example, even if a new access rule is authorized for c, the access rule is not authorized for d. In the last case, the access rules of c are not inherited by d. The access rules of d are defined independently of c.

Suppose an access rule $\langle s, c, op \rangle$ is specified for a class c. There are *mandatory* and *optional* access rules. Suppose a class d is derived from c. If an access rule $\langle s, c, op \rangle$ is *mandatory*, d is required to inherit an access rule $\langle s, d, op \rangle$ from c. The rule $\langle s, d, op \rangle$ cannot be changed for d. Each time the class d and its objects are manipulated, the access rules of the class c are checked. Next, let us consider an optional rule $\langle s, c, op \rangle$. Here, every subclass d of the class c can decide whether or not d inherits $\langle s, c, op \rangle$. The rule $\langle s, d, op \rangle$ can be one of the types *inherit* and *copy* in the subclass d. If $\langle s, d, op \rangle$ is *inherit* type, $\langle s, d, op \rangle$ cannot be changed. $\langle s, d, op \rangle$ is not maintained in d as discussed for *mandatory* inheritance for c. If $\langle s, d, op \rangle$ is a *copy* type, $\langle s, d, op \rangle$ is independent of $\langle s, c, op \rangle$. Every mandatory rule $\langle s, c, op \rangle$ cannot be specified as *copy* in d. The mandatory access rule is automatically an *inherit* type in d.

In Figure 5, an access rule α is mandatory while β and γ are normal for the class c. The classes d and e inherit α from c. c is the home class of α. β is a *copy* type and γ is an *inherit* type for d. If α and γ are changed for c, α for d and e and γ for d are also changed. However, even if β is changed for c, β of d is not changed. If β of d is changed, β of e is changed. The home class of β of e is d.

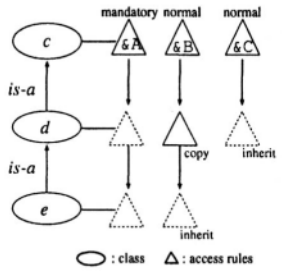

Figure 5 Types of Inheritance.

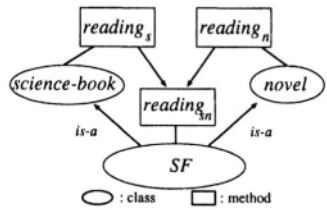

Figure 6 Multiple inheritance.

2.3. MULTI-INHERITANCE

Let us consider *novel* and *science-book* classes each of which supports a manipulation method *reading*. An *SF* (*science fiction*) class is derived from *novel* and *science-book*. *SF* inherits *reading* from *novel* and *science-book*. Suppose a subject s is granted an access right $\langle novel, reading\rangle$ but not $\langle science\text{-}book, reading\rangle$. Question is whether or not s can *read SF*. The subject s cannot read *science-book* while it can read *novel*. Thus, the access rights from *science-book* and *novel* conflict. Here, let $\sim\langle s, c, op\rangle$ show that an access rule $\langle s, c, op\rangle$ is not authorized for a class c. $\sim\langle s, c, op\rangle$ is *negative* rule of a *positive* one $\langle s, c, op\rangle$. If a subclass c inherits a pair of access rules $\langle s, c, op\rangle$ and $\sim\langle s, c, op\rangle$ from classes, the inheritance is referred to as *conflict*.

If $\langle s, c, op\rangle$ is not specified for a class c, a negative rule $\sim\langle s, c, op\rangle$ is assumed to be authorized. There are two types of inheritance of negative rules as discussed for positive rules. We have to specify which negative rules are mandatory. In addition, negative rules can be explicitly specified for each class. If $\sim\langle s, c, op\rangle$ is mandatory in a class c, every subclass d is required to inherit $\sim\langle s, c, op\rangle$ from c. Here, if d inherits both $\langle s, d, op\rangle$ and $\sim\langle s, d, op\rangle$ through the mandatory type of classes, the inheritances conflict in d. Suppose a subclass d is derived for classes c_1 and c_2. d inherits an access rule α_1 for c_1 and α_2 for c_2. Suppose c_1 and c_2 conflict. If α_1 is mandatory in c_1 and α_2 is optional in c_2, d inherits α_1. If α_1 and α_2 are optional in c_1 and c_2, the subclass d can inherit either α_1 or α_2, but not both. If α_1 and α_2 are mandatory in c and c_2, d cannot be defined from c_1 and c_2.

3. CONCLUDING REMARKS

This paper discussed a discretionary access control model in the object-oriented system. The object-oriented system supports inheritance of

properties. We made clear how to inherit the access rules in the *instance-of* and *is-a* relations. By using the inheritance of the access rules, it is easy to grant and revoke access rules in systems which are composed of various kinds of classes and objects.

References

Bell, D. E. and LaPadula, L. J., "Secure Computer Systems: Mathematical Fousdations and Model," *Mitre Corp. Report*, No.M74-244, 1975.

Dittrich K R, Haertig M, Pfefferle H., "Discretionary Access Control in Structurally Object-Oriented Database Systems," *Database Security* 2, ppl05-121, 1989.

Grosling, J. and McGilton, H., "The Java Language Environment," *Sun Microsystems*, Inc., 1996.

Lampson, B. W., "Protection," *Proc. of the 5th Princeton Symp. on Information Sciences and Systems*, 1971, pp.437-443.

Thuraisingham, M. B., "Mandatory Security in Object-Oriented Database Systems," *ACM Sigplan Note*, Vol. 24, No. 10, 1989 pp.203-210.

Object Management Group Inc., "The Common Object Request Broker : Architecture and Specification," Rev. 2.1, 1997.

Oracle Corporation,"Oracle8*i* Concepts", Vol. 1, Release 8.1.5, 1999.

Samarati, P., Bertino, E., Ciampichetti, A., and Jajodia, S., "Information Flow Control in Object-Oriented Systems," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 9, No. 4, 1997, pp. 254–238.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E., "Role-Based Access Control Models," *IEEE Computer*, Vol. 29, No. 2, 1996, pp. 38–47.

Spooner, D., "The Impact of Inheritance on Security in Object-Oriented Database System," *Database Security* 2, 1989, pp. 141–150

Stroustrup, B., "The C++ Programming Language (2nd ed.)," *Addison-Wesley*, 1991.

Yasuda, M., Higaki, H., and Takizawa, M., "A Purpose-Oriented Access Control Model for Information Flow Management," *Proc of 14th IFIP Int'l Information Security Conf. (IFIP'98)*, 1998, pp. 230–239.