2F-5

Authorization Model in Object-Oriented Systems [*]

Keiji Izaki, Katsuya Tanaka, and Makoto Takizawa [†]
Tokyo Denki University [‡]
Email : {izaki, katsu, taki}@takilab.k.dendai.ac.jp

1 Introduction

Various kinds of applications like electronic commerce are required to be realized in secure information systems. The system is *secure* only if authorized subjects are allowed to manipulate objects in authorized ways. Various kinds of access control models are discussed so far, e.g. basic model [1] and lattice-based model. An access rule is specified in a form $\langle s, o, op \rangle$ which means that a subject s is allowed to manipulate an object o by an operation op. Here, s is granted an *access right* $\langle o, op \rangle$. In the mandatory model, the access rules are defined only by an authorizer. On the other hand, a subject granted an access right can grant the access right to another subject in the discretionary model like relational database systems. In addition, objects mean simple files with simple methods *read* and *write*.

The distributed systems are now being developed according to the object-oriented frameworks like CORBA. An object is an encapsulation of data and methods for manipulating the data. There are two types of objects, i.e. *classes* and *instances*. An instance is created from a class. The objects are structured in *is-a* and *part-of* relations.

The object-based system supports only the encapsulation of data and methods and the message-passing means for invoking the methods. In object-oriented programming languages C++ and Java, variables and methods in a class are defined to be *public* ones which can be used by the outside of the class or *private* ones which can be used only in the class. However, the access rules cannot be specified for each subject.

The object-oriented system is composed of various kinds of classes and instances like systems including various database systems. It is cumbersome to authorize access rules for the objects. For example, if access rules for a class are inherited to instances of the class, access rules are easily authorized for the instances. We discuss how to authorize and inherit access rules on classes and instances structured in *instance-of*, *is-a*, and *part-of* relations.

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

2 Authorization

2.1 Class

First, the owner s of the system grants a subject s_c an access right $\langle m, \textbf{create class} \rangle$ on a metaclass m of the system. Then, s_c can define a class c with attributes A_1, ..., A_{m_c} and methods op_1, ..., op_{l_c} by using the **create class** method as follows:

create class c {
A_1 T_1, ..., A_{m_c} T_{m_c}; op_1, ..., op_{l_c} };

Here, A_i is an attribute of a type T_i ($i = 1, ..., m_c$). The type is a primitive class like *integer* or another class. Let π_c and μ_c be a set $\{A_1, ..., A_{m_c}\}$ of the attributes and a set $\{op_1, ..., op_{l_c}\}$ of the methods of the class c, respectively. s_c is now an owner of the class c.

A subject cannot manipulate the class c without obtaining an access right on c. The owner s_c can grant an access right $\langle c, op_i \rangle$ to a subject s and revoke the access right by the following **grant** and **revoke** methods :

grant op_i **on** c **to** s;

revoke op_i **on** c **from** s [**with cascading**];

The subject s can grant the access right $\langle c, op_i \rangle$ to other subjects. If the **cascading** option is specified, the access right $\langle c, op_i \rangle$ is revoked from not only the subject s but also every subject granted by s.

The access rules are specified in a form $\langle s, c, op_i \rangle$ where s is a subject, c is a class, and op_i is a method. $\langle s, c, op_i \rangle$ is referred to as *class access rule*. Here, let α_c be a set of access rules authorized for the class c.

2.2 Object

The owner s_c of a class c grants a subject s_o an access right $\langle c, \textbf{create object} \rangle$ as follows :

grant create object **on** c **to** s_o;

Then, the subject s_o can create an object x of the class c as follows:

create object x **from** c;

The methods of the class c are inherited by the object x. The values of x can be manipulated only through the methods op_1, ..., op_{lc} of c. Let σ_x be a set of values of the object x, i.e. *state* of x. The owner s_o grants an access right $\langle x, op_i \rangle$ to a subject s by the following **grant** method.

grant op_i **on** x **to** s;

Access rules on objects are referred to as *object access rule*. The access rules authorized for the class c are also inherited by an object x of c. Let α_x be a set of access rules for x. The class access rules in α_c are inherited by the object x as follows:

- For each $\langle s, c, op_i \rangle \in \alpha_c$, $\langle s, x, op_i \rangle \in \alpha_x$.

If a subject s is granted a class access right $\langle c, op_i \rangle$, s is also allowed to manipulate every object x of the class c through op_i. Here, x inherits an object rule $\langle s, x, op_i \rangle$ from c. The owner s_x of the object x can grant $\langle x, op_i \rangle$ to a subject s and revoke $\langle x, op_i \rangle$ by

[*] オブジェクト指向システムにおける権限付与モデル
[†] 井崎 慶之, 田中 勝也, 滝沢 誠
[‡] 東京電機大学

following **grant** and **revoke** methods:

grant op_i **on** x **to** s;

revoke op_i **on** x **from** s [**with cascading**];

If $\langle c, op_i \rangle$ is revoked from s, an access right $\langle x, op_i \rangle$ for every object x of the class c is also revoked.

2.3 Subclass

A subject s_d can create a subclass d from a class c if s_d is granted an access right $\langle c$, **create class from**$\rangle$. The subclass d is defined by using a following **create class from** method:

create class d **from** c {
$B_1\ U_1, ..., B_{k_d}\ U_{k_d};\ op_{d_1}, ..., op_{d_{l_d}}$}

The attributes A_1, ..., A_{m_c} and methods op_1, ..., op_{l_c} of the class c are inherited by the subclass d. In addition, the attributes B_1, ..., B_{k_d} and methods op_{d_1}, ..., $op_{d_{l_d}}$ are defined for the class d. Here, U_i denotes a type, i.e. a primitive class or a class of an attribute B_i (i=1, ..., k_d). The access rights, attributes, and methods of the class c are inherited by the class d. If a new access right $\langle op_i, c \rangle$ is granted to a subject s, an access right $\langle op_i, d \rangle$ is also granted to s. A manipulation method inherited from c can be applied on only the attributes of d inherited from c. A definition method inherited from c can be one of d.

The owner s_c of the class c can newly grant access rights of the class d to other subjects by using the **grant** method.

3 Inheritance of Access Rules

3.1 Instance-of relation

First, suppose an object x is created from a class c. In the *instance-of* relation, a set of α_c of the access rules of the class c are inherited by the object x as presented in the preceding section. The owner of the object x can define additional access rules and can revoke the access rules inherited from the class c. If a class access right $\langle c, op_i \rangle$ is revoked from a subject s, the object access right $\langle x, op_i \rangle$ is also revoked from s. If a new class access right $\langle c, op_i \rangle$ is defined, the object access right $\langle x, op_i \rangle$ is also authorized for every object x of the class c.

3.2 Is-a relation of classes

Let d be a subclass of a class c. The access rules of c are inherited by the subclass d. Let α_c show a set of access rules of the class c. There are following approaches to inheriting access rules of the class c to the subclass d :

1. The access rules of α_c are inherited by d.
2. The access rules of α_c are copied to d.
3. No access rule of the class c is inherited by d.

In the first approach, the access rules inherited by the subclass d depend on the class c. If the access rules in α_c are changed, the access rules in α_d are also changed. In the second approach, the access rules of α_c on c are copied to the subclass d. After defining the class d from c, the access rules of the class d are independent of the class c. In the last approach, the access rules of c are not inherited by d. The access rules of d are defined independently of c. The subclass d is defined as follows:

create class d **from** c { ... }

{ **with normal** | **copy** | **independent** };

Here, **normal**, **copy**, and **independent** show the cases 1, 2, and 3, respectively.

3.3 Multiple inheritance

A class c can be derived from multiple classes c_1, ..., c_n ($n > 1$). The attributes and methods of the classes c_1, ..., c_n are inherited by the class c. The access rules α_{c_i} of the class c_i are also inherited by the class c. Suppose a pair of classes c_i and c_j have a same attribute A and support a same method op which manipulates the attribute A by the polymorhism. Suppose an access right $\langle c_i, op \rangle$ is granted to a subject s but $\langle c_j, op \rangle$ is not granted to s. That is, s is allowed to use op for c_i but not for c_j. The class c inherits an access rule $\langle s, c.A, op \rangle$ from c_i and not the access rule from c_j. That is, the access rules inherited from c_i and c_j conflict. It is problem to decide if s can manipulate $c.A$ by op.

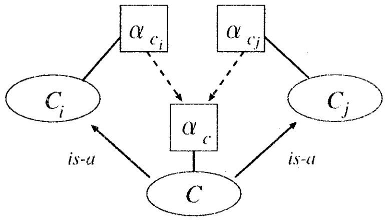

Figure 1: Multiple inheritance.

Suppose that each class c_i supports a method op (i=1, ..., n). Let $s_i(op)$ be a set of subjects who are granted an access right $\langle c_i, op \rangle$ for a method op. A new class c is derived from the classes c_1, ..., c_n. There are two approaches to inheriting access rules to c:

1. $s(op) = s_1(op) \cap ... \cap s_n(op)$.
2. $s(op) = s_1(op) \cup ... \cup s_n(op)$.

In the first case, only subject who is granted an access right for every class is granted an access right $\langle c, op \rangle$. This is the most closed way. In the second case, a subject s is allowed to manipulate the class c by op if s is granted an access right for at least one class. This is the most open way. Thus, the access rights inherited from multiple classes may conflict. The owner of c decides which access rules to be inherited to resolve the conflict.

4 Concluding Remarks

This paper discussed a discretionary access control model in the object-oriented system. The object-oriented system supports data encapsulation, class and instance, is-a and part-of relation, inheritance, and nested invocation of methods We made clear how to inherit the access rules in the *instance-of*, *is-a*, and *part-of* 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

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

[2] Izaki, K., Tanaka, K., and Takizawa, M., "Access Control Model in Object-Oriented Systems," *Proc. of the ICPADS-00 Workshops*, 2000, pp.69-74.