マルチメディア通信と分散処理 97−15
コンピュータセキュリティ 8−15
(2000. 3. 22)

Access Control Model in Object-Oriented Systems

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

The authors discuss a discretional access control model to realize secure object-oriented systems. An object is manipulated only through methods supported by the object. There are two types of objects, i.e. classes and instances. The objects are hierarchically structured in generalization (is-a) and aggregation (part-of) relations. In addition, methods are invoked in a nested manner. We discuss how the access rights are inherited to classes and instance objects in the hierarchical structure.

オブジェクト指向システムにおけるアクセス制御モデル

井崎 慶之　田中 勝也　滝沢 誠
東京電機大学理工学部情報システム工学科

本論文では、安全なオブジェクトシステムを実現するためのアクセス制御モデルについて論じる。オブジェクトは、オブジェクトがサポートするメソッドを通して操作される。オブジェクトには、クラスとインスタンスの二種があり、オブジェクトは *is-a* と *part-of* により、階層的に構成される。また、メソッドは入れ子な形式により呼び出される。本論文では、オブジェクト指向システム上の新しいアクセス制御モデルについて述べる。

1 Introduction

Various kinds of applications like electronic commerce [4] are required to be realized in secure information systems. The system is referred to as *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 [8] and lattice-based model [2,5]. In the access control model, an access rule is specified in a form $\langle s, o, op \rangle$ which shows 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 [4, 11, 14]. However, the access control model implies the *confinement* problem where illegal information flow occurs, i.e. data in an object may be obtained via other objects by unauthorized subjects of the object. In order to resolve the confinement problem, the access rules are specified in the lattice based model so that the illegal information flow never occurs based on the partially ordered *can-flow* relation among the security classes. In the access control models, objects indicate simple files which are manipulated by simple methods *read* and *write*. In the role-based model [12], a *role* is modeled to be a collection of access rights, which show some job function in the enterprise. A subject is granted a role on behalf of granting each access right to the subject.

The systems are now being developed according to the object-oriented or object-based frameworks like CORBA [9]. In the object-oriented system, 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. In the *is-a* relation, a method and data structure of a class object are *inherited* by a lower level class named *subclass* or *derived* class. If an object o is composed of other objects o_1, ..., o_n, each o_i is *a part of*, i.e. *component* object of o. A subject issues a request to an object o to invoke a method op. On receipt of the request op, the method op is performed on the object o. Then, the response of the request is sent back to the subject. Here, op may invoke another method op_i supported by an object o_i. Thus, the methods are invoked in a *nested* manner in the object-oriented system.

The object-based system supports only the encapsulation of data and methods and message-passing means for invoking the methods. Here, the methods

are invoked in the nested manner. Takizawa *et al.* [15] discuss the *purpose-oriented* access control model in the object-based system. Here, a *purpose* concept shows why a subject s can manipulate an object o through a method op in terms of the nested invocation in the object-based system. They also discuss how illegal information flow occurs in the nested invocation by analyzing the *input-state-output* relation of the methods.

In the object-oriented system, it is critical to discuss how to inherit access rights for a class to instances, subclasses, and component classes. In this paper, we take the discretionary approach. We discuss how the access rights on classes and instances are inherited by classes and objects in *instance-of*, *is-a*, and *part-of* relations.

In section 2, we present the object-oriented model. In section 3, we present how to authorize access rules. In section 4, we discuss how to inherit access rules in the object-oriented model.

2 Object-Oriented Model

The object-oriented system is composed of multiple objects which are structured as *instance-of*, *is-a* and *part-of* relations. An object is an encapsulation of data and methods for manipulating the data.

2.1 Nested invocation

An object is allowed to be manipulated only though the methods supported by the object. The methods are performed on the object on the basis of the remote procedure call (RPC). A subject like an application program first issues a request to an object o to manipulate o by a method op. On receipt of the request, the method op is performed on the object o. After op is performed, the response is sent back to the subject. Here, while op is being performed on the object o, op may further invoke other methods by the remote procedure call. Thus, the invocations of methods are nested.

In the object-oriented system, a method invoked by a subject may invoke other methods. Suppose that an object o_1 supports a method op_1 and a subject s invokes op_1 by issuing a request of op_2 to the object o_1. If op_1 is realized by using a method op_2 of an object o_2, then, op_1 invokes op_2. op_2 may furthermore invoke a method op_3 in an object o_3, op_3 may invoke op_4, ... Thus, methods are invoked in a nested manner.

Relational database systems like Sybase [14] and Oracle [11] support *trigger* mechanisms on table objects. If a table R is manipulated by a SQL command, a trigger for the SQL command on the table R is automatically performed and other tables are manipulated by SQL commands issued in the trigger. Suppose that there are a pair of table objects *Emp*(*eno*, *ename*, *dno*) and *Dept*(*dno*, *dname*) and a trigger on *Dept* is defined to manipulate *Emp* to satisfy the referential integrity [4]. Suppose that a user subject s issues a SQL command **delete** on the table *Dept*. Here, s is granted an access right $\langle$*Dept*, **delete**$\rangle$ to manipulate *Dept* by **delete**. Then, the trigger on *Dept* is performed to delete records in *Emp* whose *dno* values are the same as the records deleted in *Dept*. In Oracle, the table is allowed to be manipulated by the trigger only if the owner of *Emp* is the same as *Dept* or the owner of *Dept* is granted an access right for manipulating *Emp*. In Sybase, in addition to the mechanisms supported by Oracle, the trigger on *Dept* can be performed to manipulate *Emp* if the user subject s is granted an access right $\langle$*Emp*, **delete**$\rangle$.

In object-oriented programming languages C++ [13] and Java [7], each variable and method in a class is defined to be a *public* one which can be used by the outside of the class or a *private* one which can be used only in the class. However, the access rules cannot be specified for each subject.

Suppose a subject s first invokes a method op_1 of an object o_1 and then op_1 invokes a method op_2 of an object o_2. If s is granted an access right $\langle o_1, op_1 \rangle$, s is allowed to invoke op_1 on o_1. Here, problem is whether or not the method op_1 is allowed to invoke op_2 on o_2. There are two approaches to controlling the access to the objects in the nested invocation; *subject invocation* and *object invocation*. Here, suppose s first invokes op_1 on o_1, and then op_1 invokes op_2 on o_2. In the subject invocation model [Figure 1(1)], op_2 is considered to be invoked by the subject s on behalf of op_1. Here, op_1 is allowed to invoke op_2 only if s is granted an access right $\langle o_2, op_2 \rangle$ in addition to the access right $\langle o_1, op_1 \rangle$. On the other hand, op_1 directly invokes op_2 in the object invocation [Figure 1(2)]. Here, op_1 is allowed to invoke op_2 only if the object o_1 is granted an access right $\langle o_2, op_2 \rangle$ even if the subject s is not granted the access right. The web system [1] adopts the subject invocation.

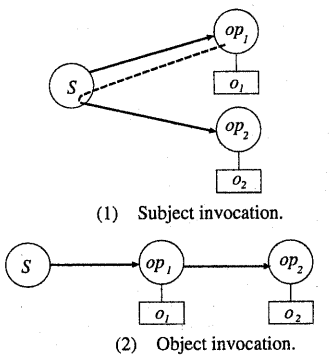

Figure 1: Nested invocation.

In the object invocation, the method op_1 of the object o_1 is allowed to invoke the method op_2 on the object o_2 only if the access rule $\langle o_1, o_2, op_2 \rangle$ is authorized. Here, op_i can invoke op_2 even if the subject s is not granted an access right $\langle o_2, op_2 \rangle$.

2.2 Classes and instances

There are two types of objects: *classes* and *instances*. A class c is composed of *attributes* and *methods*. An instance object o is created from the class c by allocating memory area for storing values of

the attributes. Here, let π_c be a set of attributes A_{c1}, ..., A_{cm_c} ($m_c \geq 1$) and μ_c be a set of methods op_{c1}, ..., op_{cl_c} ($l_c \geq 1$) of the class c. The instance object o can be manipulated only through a method in M_c. Here, o is an *instance of* the class c. Let σ_o be a set of values v_{o1}, ..., v_{om_c}, where each v_{oi} is a value of the attribute A_{ci} of the class c ($i = 1, ..., m_c$). Let μ_o be a set of methods which can be performed on the instance object o, i.e. $\mu_o = \mu_c$. In most object-oriented programming languages like C++ [13] and Java [7], a term "object" means an instance. From here, the term *object* is used to denote an *instance* in this paper, according to the convention.

The classes and objects are hierarchically structured with two kinds of relations: *is-a* and *part-of* relations, in addition to the *instance-of* relation. A new class c_2 is derived from an existing class c_1 by inheriting attributes and methods of c_1 and defining additional attributes and methods. Here, c_2 is in an *is-a* relation with c_1. c_2 is a *subclass* of c_1. Here, c_1 shows a more general concept than c_2. For example, a class *workstation* is a subclass of a class *computer*. In the traditional object-oriented systems, the *is-a* relation is defined only for classes. However, there can be the *is-a* relation between objects in pure object-oriented systems like Smalltalk [6].

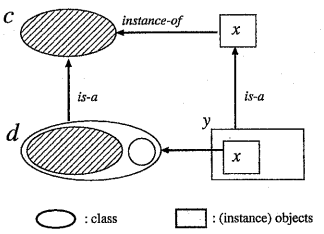

Figure 2: Classes and objects.

Figure 2 shows the *is-a* relation between a pair of classes c and d and between a pair of objects x and y. d is a subclass of the class c. x and y are objects created from the classes c and d, respectively. In addition, y is in an *is-a* relation with x, i.e. the values of x are inherited to y. $\sigma_x \subseteq \sigma_y$ and $\mu_x \subseteq \mu_y$. The object y in fact does not have the value σ_x of x in the real implementation in order to reduce the storage and reuse the data and methods of x in y. If the values of x in the object y are manipulated, the values of the object x are really manipulated. Let $y.c$ denote the values and methods of y inherited from x.

A class c can be composed of other classes c_1, ..., c_n. Here, each class c_i is a *part of* the class c. c_i is a component class of c. For example, the class *computer* is composed of component classes *cpu*, *memory*, *bus*, *io*, and *disk*. Suppose that a class d is a component of a class c [Figure 3]. Let x and y be objects of the classes c and d, respectively. The object y is also a part of the object x.

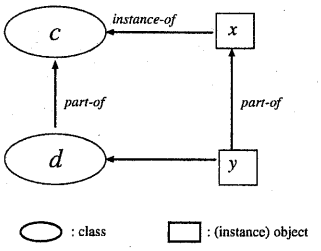

Figure 3: Part-of relation.

3 Authorization in Object-Oriented Systems

We discuss a discretionary access control model for object-oriented systems.

3.1 Class

First, the owner s of the system grants a subject s_c an access right $\langle m, \textbf{create class} \rangle$ on a meta class m of the system. Then, the subject 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 an atomic class like *integer* and *character* or another class. Let π_c be a set $\{A_1, ..., A_{m_c}\}$ of the attributes of the class c. Let μ_c be a set $\{op_1, ..., op_{l_c}\}$ of the methods supported by the class c. The subject 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 of the class c can grant an access right $\langle c, op_i \rangle$ on the class c to a subject s by the following **grant** method :

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

The access rule $\langle s, c, op_i \rangle$ on the class c is referred to as *class access rule*. The owner s_c of the class c can revoke the access right $\langle c, op_i \rangle$ from the subject s:

revoke op_i **on** c **from** s;

Suppose the subject s grants its access right $\langle c, op_i \rangle$ to another subject s'. In order to revoke the access right from s', the following **revoke** method is used:

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 the subjects granted by s.

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

Suppose that there are classes c_1, ..., c_m in the system. Let op_{ij} denote a method of a class c_i. A *role* R

is defined to be a collection of access rights $\{\langle c_i, op_{it}\rangle$, $\langle c_j, op_{ju}\rangle$, ... $\}$. A role R is defined as follows:

create role R **as** op_{it} **on** c_i, ..., op_{iu} **on** c_j;

Since the role R is defined on the classes c_u, ..., c_j, R is referred to as *class role*. The role R is granted to and revoked from a subject s as follows:

grant role R **to** s;

revoke role R **from** s;

The subject s is allowed to use the method op_{it} of the class c_i in the role R.

3.2 Object

If a subject s_o would like to create an object from a class c, s_o is required to obtain an access right for creating the object. The owner subject s_c grants s_o an access right $\langle c_o$, **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 by the following method **create object**. Here, s_o is an owner of x.

create object x **from** c;

The methods of the class c are inherited to the object x. The values of the object x can be manipulated only through the methods op_1, ..., op_{l_c} supported by the class c. Let σ_x be a set of values of the object x. σ_x is referred to as *state* of the object 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;

Then, the subject s can manipulate a state σ_x of the object x by using the method op_i. The access rule on the object x is referred to as *object access rule*.

The access rights authorized for the class c are also inherited to an object x of c. Let α_x be a set of access rules for the object x. On creating an object x from the class c, the access rules in α_c are inherited to the object x as follows:

- For each class access rule $\langle s, c, op_i\rangle$ in α_c, an object access rule $\langle s, x, op_i\rangle$ is authorized for the object x, i.e. $\langle s, x, op_i\rangle \in \alpha_x$.

If a subject s is granted an access right $\langle c, op_i\rangle$ for a class c, s is also allowed to manipulate every object x of c through op_i. Here, a rule $\langle s, x, op_i\rangle$ is *inherited* from c. In addition to the access rules inherited from c, the owner s_x of the object x can grant an access right $\langle x, op_i\rangle$ to other subjects by using the following **grant** method as presented before. In addition, the owner s_o can revoke an access rule $\langle s, o, op_i\rangle$ inherited from c by the following **revoke** method :

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

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

By revoking an access rule $\langle s, x, op_i\rangle$ inherited from the class $\langle s, c, op_i\rangle$, we can restrict the access rights of the class c to be inherited to the object x. If an access right $\langle c, op_i\rangle$ is revoked from the subject s, an access right for every object x of the class c is also revoked.

Suppose that a class role R is granted to a subject s. For each access right $\langle c, op\rangle$ in R, the subject s is automatically granted an access right $\langle x, op\rangle$ for every object x of the class c.

3.3 Subclass

A subject s_d is allowed to create a subclass d from a class c if s_d is granted an access right $\langle c$, **create class from**$\rangle$ by the owner s_c of the class c. A subclass d of the class c 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}}

The attributes A_1, ..., A_{m_c} and methods op_1, ..., op_{m_c} of the class c are inherited to the class d. In addition, the attributes B_1, ..., B_{k_d} and the methods op_{d_1}, ..., op_{dl_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 subject s_d is an owner of the subclass d. The access rights, attributes, and methods of the class c are inherited to the class d. An access right $\langle op_i, c\rangle$ of c is granted to d. The method op_i of the class d can be applied on only the attributes inherited from the class c.

- $\alpha_c \subseteq \alpha_d$, $\pi_c \subseteq \pi_d$, and $\mu_c \subseteq \mu_d$.

The class access rule $\langle s, c, op\rangle$ of the class d inherited from the class c can be revoked by the owner s_d of d as follows:

revoke op **on** d **from** c;

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.

3.4 Object of subclass

We present how to create an object from the class d. One way is to newly create an object from the class d as discussed before. The other way is to create an object y from an object x where y is inherited from the object x of the class c, which is a superclass of d.

An object y of the class d is defined so as to inherit the values and methods of x as follows:

create object y **from** d **for** x;

The object y inherits all the methods op_1, ..., op_{dm_d} of the class d. The state, i.e. values of the object x is inherited to y. The access rights are inherited from the object x. $\mu_x \subseteq \mu_y$, $\sigma_x \subseteq \sigma_y$, and $\alpha_x \subseteq \alpha_y$.

Figure 2 shows the relation among the classes c and d and the objects x and y. In current object-oriented systems like C++, Java, and database systems, there is no *is-a* relation between the objects. In the object-oriented system, there is an *is-a* relation among the objects like "y *is an* x" as shown in Figure 2.

3.5 Component class

There are two ways to create an object x from a class c as discussed in this section. One way is to just create an object x from the class c as shown in

Figure 4(1). The object x created is independent of the objects created from the component classes.

A class c is defined on other classes d_1, ..., d_m as follows:

create class c { B_1 d_1, ..., B_m d_m; ..., } ...(1)

Here, the classes d_1, ..., d_m are components of the class c. An object x is created from the class c. Here, attributes B_i shows an object of the class d_i which is also created on creation of x.

An object x shown in Figure 4(1) is created from the class c as follows:

create object x **from** c;

The other way to create an *object* x is to include an existing object y_i of a component class c_i (i=1, ..., m).

create class c { $*B_1$ d_1, ..., $*B_i$ d_i, ..., $*B_m$ d_m; ..., } ...(2)

Here, $*B_i$ shows that an attribute B_i of an object x, created from the class c, i.e. $x.B_i$ refers to an existing object created from a class d_i as shown in Figure 4(2).

An object x is created from the class c by the following method.

create object x **from** c **with**
y_1 **for** B_1 **from** d_1, ..., y_m **for** B_m **from** d_m;

Here, each attribute $x.B_i$ refers to an existing object y_i of the class d_i as shown in Figure 4(2).

There is further case the object y_i of the class d_i is copied to the object x as shown in Figure 4(3). Here, y_i and $x.B_i$ are required to be consistent. The object x is created as follows:

create class c { ..., B_i d_i, ...};

create object x **with** y_i **for** B_i **from** d_i;

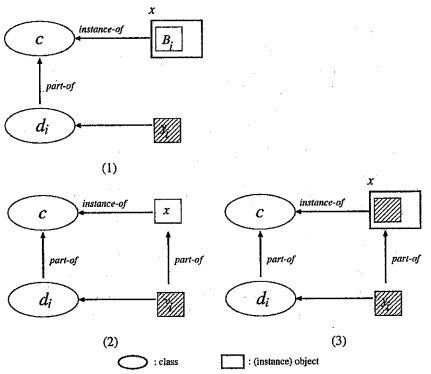

Figure 4: Objects with part-of relation.

4 Inheritance of Access Rights

4.1 Instance-of relation

First, suppose an object x is created from a class c. In the *instance-of* relation, the access rules α_c of the class c are inherited to the object x as presented in the preceding section. The owner of the object x can define new 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.

4.2 Is-a relation of classes

Let a class d be a subclass of a class c as shown in Figure 2. The access rules of the class c are inherited to the subclass d. Let α_c show a set of access rules of the class c. There are three approaches to inheriting the access rules from the class c to the subclass d:

1. The access rules of α_c are inherited to d.
2. The access rules of α_c are inherited to d by copying α_c for d, but the access rules inherited to d are independent of α_c.
3. No access rule of the class c is inherited to d.

In the first approach, the access rules inherited to the class d depend on the class c. If the access rules in R_c are changed, the access rules in R_d are changed. If a new access rule is authorized for the class c, the access rule is also authorized as an access rule of the class d. If an access rule on the class c is revoked, the access rule on d is also revoked.

In the second approach, the access rules of R_c on the class 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. For example, even if a new access rule is authorized for the class c, the access rule is not authorized for the subclass d.

In the last approach, the access rules of the class c are not inherited to the subclass d. The access rules for the class d are newly defined independently of the class c.

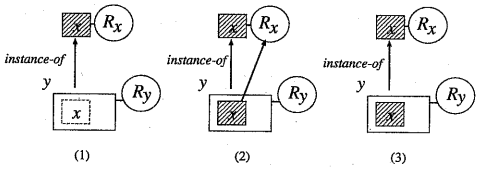

Figure 5: Objects with *instance-of* relation.

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 to the class c. As explained here, the access rights α_{c_i} of the class c_i are also inherited to the class c. Suppose an access right $\langle c_i, op \rangle$ is granted to a subject s and an access right $\langle c_j, op \rangle$ is not granted to s. Suppose c_i and c_j have the same attribute A. The subject s can manipulate A for c_i through op but cannot for c_j. Thus, the access rights inherited from multiple classes may conflict. The owner of c decides which access rules to take in order to resolve the confliction.

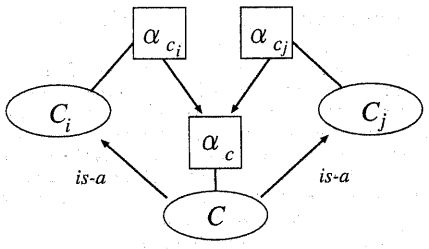

Figure 6: Multiple inheritance.

4.3 Is-a relation of objects

Next, suppose that an object y of a class d is created from an object x of a class c as shown in Figure 2. There are two ways to inherit the values and methods of the object x to the object y. In one way, the values and methods of x are not copies to y, i.e. y does not have the values and methods of x [Figure 6(1)]. Smalltalk and other artificial intelligence systems like frame systems adopt the way. Suppose a subject s manipulates the object $y.c$ by using a method *op*. The subject s is allowed to perform *op* on the object x only if an access right $\langle x, op \rangle$ is granted to s. The owner s_y of the object y cannot grant any access right on $y.c$ to other subjects. Only the owner s_x of the object x can grant access rights on x. In the other way, the values and methods are copies to the object y. In the first way, the access rules α_x are used to manipulate the values and methods of the object x in the object y [Figure 6(2)]. In the second way, there are the same approaches as discussed in the class-class inheritance [Figure 6(3)].

4.4 Part-of relation

Suppose a class c is composed of component classes d_1, ..., d_m. There are the following cases on how the access rules on the class c are related with each component class d_i as discussed in the *is-a* relation.

1. The access rules of α_{d_i} are inherited to c.
2. The access rules of α_{d_i} are copied to c.
3. No access rule of the class d_i is inherited to c.

5 Concluding Remarks

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

References

[1] Aviel D., Daniel G., and Marcus J., "Web Security," *Wiley Computer Publishing*, 1997.

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

[3] Castano, S., Fugini, M., Matella, G., and Samarati, P., "Database Security," *Addison-Wesley*, 1995.

[4] Date C. J., "An Introduction to Database Systems," *Addison-Wesley*, 1990.

[5] Denning, D. E. and Denning, P. J., Cryptography and Data Security, *Addison Wesley*, 1982.

[6] Goldberg, A., "Smalltalk-80 The Interactive Programming Environment," *Addison-Wesley*, 1984.

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

[8] Lampson, B. W., "Protection," *Proc. of the 5th Princeton Symp. on Information Sciences and Systems*, 1971, pp.437-443 (also in *ACM Operating Systems Review*, Vol.8, No.1, 1974, pp.18-24).

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

[10] Oracle Corporation,"Oracle8*i* Concepts, Volume 1, Release 8.1.5," 1999.

[11] Oracle Corporation, "Oracle Server Administrator's Cuide Release 8.0," 1997.

[12] 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.

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

[14] Sybase Inc., "Sybase Adaptive Server Enterprise Security Administraton," 1997.

[15] Tachikawa, T., Yasuda, M., and Takizawa, M., "A Purpose-oriented Access Control Model in Object-based Systems," *Trans. of IPSJ*, Vol.38, No.11, 1997, pp.2362-2369.

[16] Tachikawa, T., Yasuda, M., Higaki, H., and Takizawa, M., "Purpose-Oriented Access Control Model in Object-Based Systems," *Proc. of the 2nd Australasian Conf. on Information Security and Privacy (ACISP'97)*, 1997, pp. 38–49.

[17] 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.