Role-Based Purpose-Oriented Access Control in Object-Oriented Systems

Masashi Yasuda, Tsunetake Ishida, Hiroaki Higaki, and Makoto Takizawa
Tokyo Denki University
Email : {masa, tsune, hig, taki}@takilab.k.dendai.ac.jp

Various kinds of distributed applications have been developed by using object-oriented technologies. Object-oriented technologies like CORBA are widely used to realize the interoperability of the applications. Object-oriented systems are composed of multiple objects which cooperate to achieve some objectives by passing messages. In addition to realizing the interoperability, it is essential to make the system secure. The purpose-oriented access control nearly discusses a purpose why a subject manipulates an object in a method. In this paper, we discuss the purpose-oriented access control in the object-oriented system.

オブジェクト指向システムにおける役割に基づいた目的指向アクセス制御

安田 昌史　石田 常竹　桧垣 博章　滝沢 誠
東京電機大学理工学部経営工学科
E-mail {masa, tsune, hig, taki}@takilab.k.dendai.ac.jp

さまざまな種類の分散型アプリケーションは、CORBA に代表されるオブジェクト指向技術を用いて実現されている。オブジェクト指向システムは複数のオブジェクトから構成され、それらオブジェクトの協調動作により実現される。目的指向アクセス制御は、あるサブジェクトがあるオブジェクトに対してある操作演算で操作するときの目的を考慮する。本論文では、オブジェクト指向システムにおける目的指向アクセス制御について議論する。

1 Introduction

By using object-oriented technologies, various kinds of systems like database systems [2] and languages like C++ and JAVA [9] have been developed. Object-oriented systems are composed of multiple objects which cooperate to achieve some objectives by exchanging request and response messages. An object is an encapsulation of data and methods for manipulating the data. The objects are structured in *part-of* and *is-a* relations in the object-oriented systems while the objects are not structured in the object-based system.

The Common Object Request Broker Architecture (CORBA) [12] is now getting a standard framework for realizing the interoperability among various kinds of distributed applications. In addition to realizing the interoperability, the applications are required to be secure. That is, not only objects have to be protected from illegally manipulated but also have to be prevented illegal information flow [4,13,6] among objects.

In the basic access control model [10], an access rule is specified in a form $\langle s, o, t\rangle$ which means that a subject s can manipulate an object o in an access type t. A pair $\langle o, t\rangle$ is an *access right* granted to s. Only the access request which satisfies the access rules is accepted to be computed. However, the access control model implies the *confinement* problem [11], i.e. illegal information flow may occur among subjects and objects. In order to make every information flow legal in the system, the *lattice based* access control model [1,4,13] is proposed. The legal information flow is given by classifying objects and subjects and defining the *can-flow* relation [4] between classes of objects and subjects. In the *mandatory* model, the access rules are specified by an authorizer so that only the legal information flow occurs. For example, if a subject s reads an object o, information in o flows to s. Hence, s can read o only if a *can-flow* relation from o to s is specified. In the *discretionary* model [3,5,6], the access rules are defined in a distributed manner while the mandatory access rules are specified only by the authorizer in a centralized manner. For example, the access rules can be granted to other subjects in the relational database systems like Sybase [15]. In the *role-based* model [7, 14, 17], a *role* is defined to be a collection of access rights, i.e. pairs of access types and objects which show a job function in the enterprise. The access rule is specified by granting subjects the roles while each subject is granted an access right in the access control model. The rule-based model is now being used in kinds of applications.

The traditional access control models discuss what object can be manipulated by what subject in what access type. The authors [16, 18] newly propose a *purpose-oriented* model which takes into account a *purpose* concept why each subject manipulates objects in the object-based system. In the object-based system, methods are invoked in a nested manner. The purpose is modeled to be a method which invokes another method in the object-based system. It is critical to discuss how to specify access rules in the nested invocation of methods. One way is that a method op_1 of an object o_1 can invoke a method op_2 of an object o_2 if a subject which invokes op_1 is granted an access right $\langle o_2, op_2 \rangle$. Sybase [15] adopts the *ownership chain* mechanism where op_1 can invoke op_2 if the owner of o_2 is the same as o_1 even if s is not granted an access right $\langle o_2, op_2 \rangle$. It is not easy, possibly impossible to specify access rules for huge number of objects and subjects. Another way is that op_1 can invoke op_2 only if o_1 is granted an access right $\langle o_2, op_2 \rangle$. We take this *object pairwise approach*.

In addition, we discuss how to incorporate the role concepts into the purpose-oriented model in an object-oriented system where methods are invoked in the nested manner. Then, we discuss information flow to occur among the roles through the nested invocations.

In section 2, we present the model in the object-oriented systems. In section 3, we discuss access rules. In section 4, we discuss information flow.

2 System Model

2.1 Object-oriented system

Object-oriented systems are composed of objects. Objects are encapsulations of data and methods for manipulating the data. Each object is uniquely identified by an object identifier. There are two kinds of objects; *classes* and *instances*. A class is defined to be a set of *attributes* and methods. An instance is a tuple of values, each of which is a value of an attribute in the class, with the methods of the class. A term "object" means an instance in most object-oriented systems like JAVA.

A method of an object is invoked by sending a request message to the object. The method specified by the message is performed on the object. On completion of the computation of the method, the object sends the response back to the sender object of the message. The method may further invoke methods in other objects. Thus, the invocations of the methods are *nested*.

A class can be derived as a specialization of one or more classes. Here, suppose a class c_2 is derived from a class c_1. c_2 is called a *subclass* of c_1. In turn, c_1 is a *supperclass* of c_2. The attributes and methods of c_1 are inherited by c_2. *Inheritance* provides means for building new classes from the existing classes. The relation between a pair of a superclass and subclass is referred to as *is-a* relation. A subclass may *override* the definition of attributes and methods inherited from the supperclass.

2.2 Roles

Each subject plays some *role* in an organization, like a designer and clerk. A role represents a job function that describes the authority and responsibility in the organization. In the role-based access control model [7, 14, 17], a *role* is modeled to be a set of *access rights*. An access right is given a pair of a method op and an object o which supports op, i.e. $\langle o, op \rangle$. That is, a role means what method can be performed on what object. In the role-based model, a subject s is granted a role r while s is granted access rights in the access control model. Here, a subject s is referred to as *bound* with the role r. s is referred to *as belong* to r. This means that s can perform a method op on an object o if $\langle o, op \rangle \in r$. For example, let us consider two roles *Professor* and *Student* in a university. In the university, professors give examinations to students and mark the examination papers written by the students. There is an object *Paper* showing an examination paper and another object *Record* includes the marks which the students obtained at the examinations. A role *Professor* is { ⟨ *Paper*, *make* ⟩, ⟨ *Paper*, *mark* ⟩, ⟨ *Record*, *record* ⟩, ⟨ *Record*, *publish* ⟩, ⟨ *Record*, *look* ⟩ } and *Student* is { ⟨ *Paper*, *write* ⟩, ⟨ *Record*, *look* ⟩ }. In the role-based model, a person who plays a role of *Professor* in the university is granted the role *Professor*. A student is granted the role *Student*. Thus, it is easier to grant subjects access rights than the access control model.

Some roles are *hierarchically* structured to represent organization's logical authority and responsibility. If a role r_i includes all of access rights of another role r_j, r_i is *higher than* r_j ($r_j \preceq r_i$). The relation "$\preceq$" is transitive. Here, if neither $r_j \preceq r_i$ nor $r_i \preceq r_j$, r_i and r_j are uncomparable. Here, let us consider an *Assistant* who can mark the examination paper and look at the record, that is, *Assistant* = { ⟨ *Paper*, *mark* ⟩, ⟨ *Record*, *look* ⟩ }. Here, *Assistant* $\preceq$ *Professor* since *Professor* $\supset$ *Assistant*, i.e. *Professor* takes a higher position than *Assistant*. However, professors cannot write the examination paper although they can make questions for the examination and can mark the papers. Therefore, *Student* is uncomparable with *Professor* and *Assistant*.

In a role-based model, each subject s can manipulate an object o by a method op of o only if s is granted a role including an access right $\langle o, op \rangle$. If a subject s would like to exercise the authority of a role r to which s belongs, the subject s first establishes a *session* to the role r. Then, s can play a role of r, i.e. s can manipulate o by op.

[Access condition] A subject s can manipulate an object o by invoking a method op of o if

1. the owner of o assigns an access right $\langle o, op \rangle$ to a role r,
2. s belongs to a role r, and
3. s is establishing a session to r. □

For example, a subject s can perform *mark* on an object *examination paper* while a session between s and a role *Professor* or *Assistant* is established in Figure 1. The authority of a role r can be exercised only while a subject s establishes a session to r.

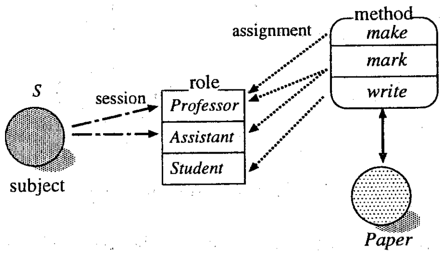

Figure 1: Role-based access.

We assume every object satisfies the following properties :

1. o can be manipulated only through methods supported by o, and
2. no methods malfunction.

3 Purpose-Oriented Access Control

The purpose-oriented model [16, 18] newly introduces a *purpose* concept to the access control model. A *purpose* shows why each subject s manipulates an object o by invoking a method op of o. In the object-based system, methods are invoked in the nested manner. Suppose that a subject s invokes a method op_1 of an object o_1 and then op_1 invokes a method op_2 of an object o_2. In the purpose-oriented model, the method op_1 invoking a method op_2 of an object o_2 shows the purpose why the object o_1 manipulates the object o_2, while the access control model specifies whether or not o_1 can manipulate o_2 by issuing op_2. For example, let us consider that a person s withdraws money from a bank object b. In the access control model, the person s can withdraw money from b if an access rule $\langle s, b, withdraw \rangle$ is authorized independently of purposes for what s spends the money. On the other hand, s can get money from the bank b for purpose of *house-keeping* but not for *drinking*. An access rule $\langle s : \textit{house-keeping}, b : withdraw \rangle$ is specified where a method *house-keeping* of s shows the purpose. Finally, the method op_1 of the object o_1 can invoke op_2 of o_2 only if the access rule $\langle o_1 : op_1, o_2 : op_2 \rangle$ is authorized.

Here, suppose that a subject s invokes a method op_1 on an object o_1 and then op_1 invokes a method op_2 on another object o_2. Here, suppose s is granted an access right $\langle o_1, op_1 \rangle$. In one way, only if s is granted an access right $\langle o_2, op_2 \rangle$, op_1 can invoke op_2. However, it is cumbersome for each object to specify which subject can manipulate the object. In Sybase [15], the ownership chain method is adopted. Here, if the object o_2 has the same owner as the object o_1 and s is granted an access right $\langle o_1, op_1 \rangle$, op_1 can invoke op_2 even if s is not granted an access right $\langle o_2, op_2 \rangle$. Otherwise, op_1 is allowed to invoke op_2 only if s is granted an access right $\langle o_2, op_2 \rangle$. Suppose the response of op_2 carries some data derived from the object o_2. On receipt of the response, the object o_2 may store the data carried by the response in itself, e.g. the data is stored in the file of o_2 while o_2 continues to compute op_1 by using the response. This means, information in o_2 flows to o_1 through the invocation. The data may be brought to other objects by further invocation. By using the ownership chain method, illegal information flow may occur. In this paper, we assume that the system is composed of multiple autonomous objects, that is, objects have different owners. Furthermore, it is difficult, maybe impossible for each autonomous object to grant access rights to subject persons. In this paper, we take an object pairwise approach where access rules are specified for a pair of autonomous objects o_i and o_j.

Here, suppose that a method op_1 of an object o_1 invokes a method op_2 of an object o_2. There are types of invocations, i.e. synchronous, asynchronous, and one-way invocations. In the synchronous invocation, the method op_1 blocks until receiving the response of op_2. This is a well-known remote procedure call (RPC). In the asynchronous invocation, op_1 does not block and continues the computation after invoking op_2. However op_1 eventually receives the response from op_2. This is similar to *fork* mechanism in Unix. In the one-way invocation, op_1 *neither* blocks after invoking op_2 nor receives the response from op_2. op_2 is computed independently of op_1. In the invocation of op_2 by op_1, the object o_1 plays a role of *subject* and o_2 plays a role of *object* in the access control tradition. In the nested invocation, the subject-object relation is relative.

A role is specified in a collection of access rights in the role-based model [7, 14, 17]. We extend the purpose-oriented access control model to incorporate the role-based model. In the object-based system, objects are related in the invocation relation. In this paper, we consider an object based system where objects are hierarchically structured. For example, let us consider a travel agent object A. A supports methods BookTravel, Payment. The travel agent object A is realized by Hotel objects, Air line objects, Train objects, and RentaCat objects. For example, BookTravel invokes Book methods of hotel object H and airline object L. Here, the travel agent object is at a higher level than the other objects.

An object o_1 is *higher* than o_2 ($o_1 \succ o_2$) iff a method of o_1 invokes a method of $o_2 \ldots o_n$, $o_1 \succ o_3 \succ o_2$ for some object o_3. Here, $A \succ H$ in the example of the travel agent. The objects are hierarchically structured in the system iff $o \succ o$ does not hold for every object o. A pair of object o_1 and o_2 are at the same level ($o_1 \equiv o_2$) iff neither $o_1 \succ o_2$ nor $o_2 \succ o_1$. Objects at the level 0 are objects which are not invoked by other objects. Objects at the level i are objects which are invoked by object at the level $i-1$. In this paper, we assume that each object belongs to one level. That is, each object at a level i invokes only methods of objects at the level $i+1$.

We consider roles on the objects hierarchically structured. A role of a level i is a collection of access rights on the objects at the level i. Let R^i be a role of a level i which is $\{ \langle o^i, op \rangle \mid o^i$ is at the level i and op is a method of $op \}$.

Suppose that a method op_1 of an object o_1 invokes op_2 of o_2. Here, o_1 is at a level i and o_2 at level $i+1$. We also suppose that o_1 is invoked in a role R_1.

Each method op_i of an object o_i is granted a role $r_i = \{\langle o_{i1}, op_{i1}\rangle, \ldots, \langle o_{ih_i}, op_{ih_i}\rangle\}$. This means, the method op_i can invoke a method op_{ij} of an object o_{ij} (for $j = 1, \ldots, h_i$). In turn, op_{ij} may be granted a role $r_{ij} = \{\langle o_{ij1}, op_{ij1}\rangle, \ldots, \langle o_{ijh_{ij}}, op_{ijh_{ij}}\rangle\}$. op_{ij} can invoke a method op_{ijk} of o_{ijk} if op_{ij} is granted the role r_{ij}. An access rule has to show in what role the method op_i of the object o_i is bound to the role r_i.

[Purpose-oriented role-based access (POR) rule] $\langle r : o_i : op_i,\ r_i\rangle$ means that a method op_i of an object o_i is invoked in a role r and op_i can invoke methods specified in a role r_i. □

[Example 1] Suppose that there are two roles *entertainment* and *house-keeping* including access right $\langle p$, *drinking*$\rangle$ and $\langle p$, *shopping*$\rangle$, respectively. A person p plays the roles in a community and manipulates the bank object b by authority of its role. If the method *drinking* of p is invoked in the role *entertainment*, p is allowed to withdraw money from the bank b. However, p is not allowed to do so if *drinking* of p is invoked in the role *house-keeping*. Thus, the access rule is specified in a form $\langle$*entertainment* : p : *drinking*, b : *withdraw*$\rangle$ where the method *drinking* shows the *purpose* of p. □

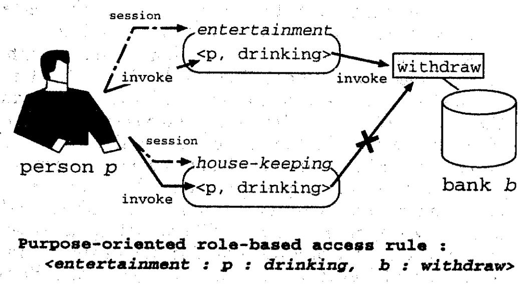

Figure 2: Purpose-oriented role-based access.

The object-oriented system is composed of classes and objects, i.e. instances of the classes. There are two kinds of access rights, *class* and *instance* access rights. A class access right is in a form $\langle c, op\rangle$ where c is a class and op is a method of the class c. On the other hand, an instance access right is in a form $\langle o, op\rangle$ where o is an object and op is the method of o.

There are two kinds of roles, i.e. class roles and instance roles. A class role r is defined in terms of methods and classes, i.e. $r = \{\langle c, op\rangle$ where c is a class and op is a method of $c\}$. On the other hand, an instance role r' is defined in terms of methods and objects, i.e. $r' = \{\langle o, op\rangle$ where o is an object and op is a method of $o\}$. r' is instantiated from the class role r. In the instance role r', o is an object which is instantiated from a class c.

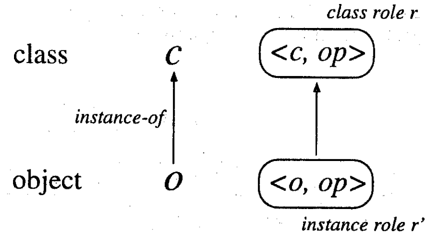

Figure 3: class role and instance role.

For example, in Figure 4, a class role *member* is defined as *member* $= \{\langle$*computer*, *use*$\rangle\}$. A class role *member* is bound to a class *student*, i.e. $\langle$*student*, *member*$\rangle$. This means that the class *student* is authorized to access to the class *computer* by the method *use* and authority of the class role *member*. On the other hand, an object p is instantiated from a class *student* as an instance of *student*. PC_1 and PC_2 are also instantiated from a class *computer*. p would manipulate PC_1 in the system. An instance role *member* is instantiated from a class role *member* to control the access between p and PC_1. An instance role *member* is associated to the p. Even if PC_2 is an instance of *computer*, $\langle PC_2$, *use*$\rangle$ does not exist in the instance role *member* where p should not manipulate PC_2.

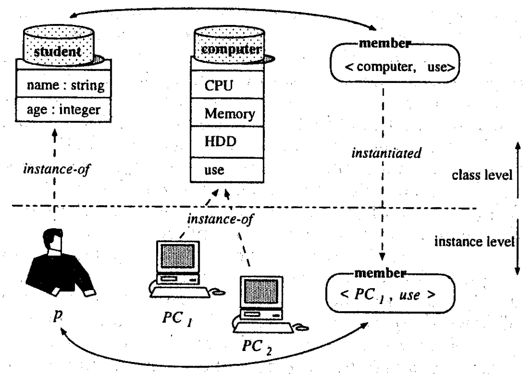

Figure 4: Instantiation of class and role.

Furthermore, there is an *is-a* relation in object-oriented systems. The *is-a* relation is defined among classes. We extend the role concept to conform to the *is-a* relation. Suppose that there are two classes c_1 and c_2. The class c_2 is defined as a specialization of the class c_1, i.e. c_2 *is a* c_1. The access right $\langle c_2, op\rangle$ is automatically included in the role r where r is given as $\{\langle c_1, op\rangle\}$. This means that the access right of specialized class is given to the role when the role has an access right of its supperclass.

4 Information Flow Control

In the role-based access control model presented in the previous section, it is assured that subjects manipulate objects based on roles to which the subjects belong. However, illegal information flow among objects may occur. Because legal and illegal information flow among the objects are not discussed. For example, in Figure 5, suppose that a subject s_i invokes *write* on an object o_j after invoking *read* on o_i by the authority of a role r_i. This means that s_i may write data obtained from o_i to o_j. s_j can read data in o_i even if read access right is not authorize to a role r_j. This is the confinement problem pointed out in the basic access control model. In addition, a subject can have multiple roles in the role-based model even if they can play only one role at the same time. In Figure ??, suppose that a person A belongs to two roles *chief* and *clerk*. A person A obtains some information from *book* as a *clerk* and then stores the data derived from the information into book as a *chief*.

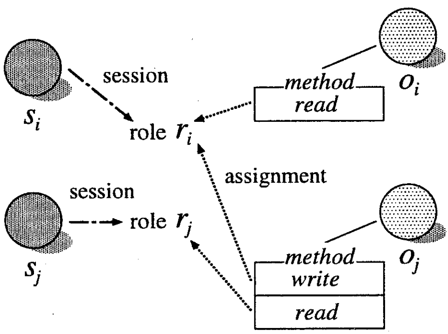

Figure 5: Illegal information flow.

We classify methods of objects with respect to the following points:

1. whether or not a value v_i of attribute a_i from an object o_i is output.
2. whether or not a value of a_i in o_i with input parameter is changed.

The methods are classified into four types in 1) m_R, 2) m_W, 3) m_{RW}, and 4) m_N. m_R means that the method outputs a value but does not change o_i. m_W means that the method does not output but changes o_i. The method m_{RW} outputs a value and changes o_i. The method m_N neither outputs a value nor changes o_i. For example, a method *count-up* is classified to be m_N because *count-up* changes the state of the object but does not need input parameter. *count-up* does not bring information into an object.

[Example 2] Let us consider a simple example about information flow between a pair of objects o_i and o_j in shown Figure 6. A subject s is now in a session with a role r_i. Here, s can invoke methods classified into m_R on o_i and m_{RW} on o_j by the authority of r_i, respectively. If s obtains information from o_i through m_R, s can invoke m_{RW} on o_j after the invocation of m_R on o_i. Because a set of roles on o_i which is authorized to execute methods classified into m_R is a subset of roles on o_j which is authorized to perform methods classified into m_R. □

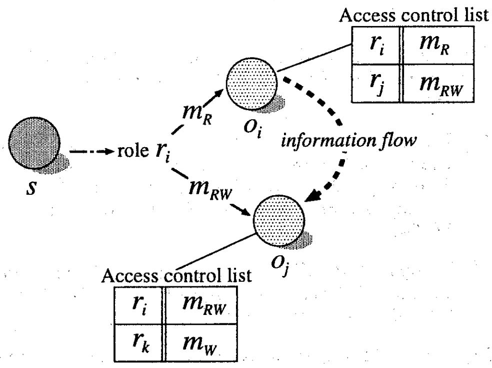

Figure 6: Information flow control.

5 Concluding Remarks

This paper has presented an access control model for distributed object-oriented systems with role concepts. Roles are higher level representation of access control models. We have defined a role to mean what method can be performed on which object. Furthermore, we have discussed how to control information flow to occur through roles.

References

[1] Bell, D. E. and LaPadula, L. J., "Secure Computer Systems: Mathematical Foundations and Model," *Mitre Corp. Report,* No. M74–244, *Bedford, Mass.,* 1975.

[2] Bertino, E. and Martino, L., "Object-Oriented Database Management Systems : Concepts and Issues," *IEEE Computer,* Vol. 24, No. 4, 1991, pp. 33–47.

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

[4] Denning, D. E., "A Lattice Model of Secure Information Flow," *Communications of the ACM,* Vol. 19, No. 5, 1976, pp. 236–243.

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

[6] Ferrai, E., Samarati, P., Bertino, E., and Jajodia, S., "Providing Flexibility in Information Flow Control for Object-Oriented Systems," *Proc. of 1997 IEEE Symp. on Security and Privacy,* 1997, pp. 130–140.

[7] Ferraiolo, D. and Kuhn, R., "Role-Based Access Controls," *Proc. of 15th NIST-NCSC Nat'l Computer Security Conf.,* 1992, pp. 554–563.

[8] Harrison, M. A., Ruzzo, W. L., and Ullman, J. D., "Protection in Operating Systems," *Communication of the ACM,* Vol. 19, No. 8, 1976, pp. 461–471.

[9] Gosling, J. and McGilton, H., "The Java Language Environment," Sun Microsystems, Inc, 1996.

[10] Lampson, B. W., "Protection," *Proc. of 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.)

[11] Lampson, B. W., "A Note on the Confinement Problem," *Communication of the ACM,* Vol. 16, No. 10, 1973, pp. 613–615.

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

[13] Sandhu, R. S., "Lattice-Based Access Control Models," *IEEE Computer,* Vol. 26, No. 11, 1993, pp. 9–19.

[14] 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.

[15] Sybase, Inc., "Sybase Adaptive Server Enterprise Security Administration," 1997.

[16] 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.

[17] Tari, Z. and Chan, S. W., "A Role-Based Access Control for Intranet Security," *IEEE Internet Computing,* Vol. 1, No. 5, 1997, pp. 24–34.

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