

## A Transaction-Based Purpose-Oriented Access Control Model for Information Flow Management

Tsunetake Ishida, Masashi Yasuda, Hiroaki Higaki,  
and Makoto Takizawa

Tokyo Denki University

E-mail {tsune, masa, hig, taki}@takilab.k.dendai.ac.jp

*Distributed applications are realized by cooperation of multiple objects. Each object is manipulated through a method supported by the object and then the method may further invoke methods of other objects. Purpose-oriented access rules indicate what methods in objects can invoke methods of other objects. Information flow among the objects occurs if the requests and responses of the methods carry data. The information flow depends on an invocation sequence of methods. We discuss how the invocation sequence of methods influences the information flow.*

### 情報流制御のためのトランザクションに基づいた 目的指向アクセス制御モデル

石田 常竹 安田 昌史 桧垣 博章 滝沢 誠

東京電機大学理工学部経営工学科

E-mail {tsune, masa, hig, taki}@takilab.k.dendai.ac.jp

分散型アプリケーションは、複数のオブジェクトの協調動作により実現される。各オブジェクトは、オブジェクトの提供するメソッドによってのみ操作される。また、メソッドは、他のオブジェクトのメソッドを呼び出すことがある。目的指向アクセス制御では、あるオブジェクトのメソッドが他のオブジェクトのメソッドを呼び出すことができるかどうかを制御される。あるオブジェクトのメソッドがオブジェクトのデータを操作するのであれば、オブジェクト間で情報流が生じる。オブジェクト間の情報流は、メソッドの呼び出しの順序に依存する。本論文では、メソッドの呼び出しの順序がどのように情報流に影響するかを議論する。

### 1 Introduction

It is significant to consider what subject  $s$  can manipulate what object  $o$  by what operation  $t$  in the access control model [1]. An access rule is specified in a form  $\langle s, o, t \rangle$ . A system is *secure* if and only if (*iff*) every object is manipulated according to the access rules. A subject  $s$  is granted an access right  $\langle o, t \rangle$  by an authorizer. However, the access control model cannot resolve the *confinement* problem [5] where information illegally flows among subjects and objects. The *lattice-*

*based* model [1,3] aims at protecting against the illegal information flow. One security class is given to each subject and object in the system. A *flow* relation among the security classes is defined to denote that information in one class  $s_1$  can *flow into*  $s_2$ . In the *mandatory* model [1,7], an authorizer specifies an access rule  $\langle s, o, t \rangle$  so that the flow relation between a subject  $s$  and an object  $o$  holds. In the *discretionary* model, the access right of one subject can be granted to other subjects. In the *role-based* model [8,11], a *role*  $R$  shows a

job function in the application, which is given to be a set of access rights, i.e.  $R = \{(o, t)\}$ . The access rule is defined to bind a subject  $s$  to a role  $R$ , i.e.  $\langle s, R \rangle$ .

Distributed applications are modeled in an object-based model like CORBA [6]. Each object is an encapsulation of data structure and methods. A method of the object is invoked on receipt of a request message. The method may invoke methods on other objects, i.e. invocation is *nested*. Takizawa *et, al* [10, 12] propose a purpose-oriented access control model in the object-based system. For example, a person  $s$  may withdraw money from a bank  $o$  in order to do house-keeping, but  $s$  cannot get money from  $o$  to go drinking. In the access control model,  $s$  can get money from  $o$  for any purpose only if an access rule  $\langle s, o, t \rangle$  is authorized. Thus, it is essential to discuss a *purpose* of  $s$  to manipulate  $o$ . In the *purpose-oriented* model [10], an access rule shows a *purpose* for which each subject  $s$  manipulates an object  $o$  by a method  $t$  of  $o$ . In the object-based system, the purpose is modeled to be a method  $u$  of  $s$  which invokes  $t$  of  $o$ . That is, a purpose-oriented access rule is specified in a form  $\langle s : u, o : t \rangle$ , where  $u$  shows the purpose. Even if each purpose-oriented rule between a pair of objects satisfies the information flow relation, some data in one object may illegally flow to another object through the nested invocation. Yasuda, Higaki, and Takizawa [10, 12] discuss what purpose-oriented access rules imply the legal information flow in the nested invocations. The relational database system Sybase [9] adopts the ownership chain mechanism. Suppose a subject, i.e. user  $s$  issues some method  $op_1$  to an object  $o_1$  and  $op_1$  invokes another method  $op_2$  on an object  $o_2$ . That is,  $op_2$  is *triggered* by  $op_1$ . If the owner of  $o_2$  is the same as  $o_1$ ,  $op_2$  can be invoked by  $op_1$  even if  $s$  is not granted the access right on  $o_2$ . This mechanism cannot be adopted to a collection of autonomous objects. In this paper, access rules are specified for a pair of autonomous objects.

In section 2, we review access control models. In section 3, we discuss the purpose concept which

shows an invocation sequence of methods in the object-based systems.

## 2 Access Control Models

### 2.1 Lattice-based model

A unit of resource in the system is referred to as *entity*. There are two roles which an entity plays, *subject* and *object* in an application. A subject  $s$  manipulates an object  $o$  by issuing a type  $t$  of method. The object  $o$  computes the method  $t$  and then sends back the response to the subject  $s$ . The authorizer specifies an access rule  $\langle s, o, t \rangle$  showing that a subject  $s$  can manipulate an object  $o$  in a method  $t$ . Here,  $s$  is referred to as *granted an access right*  $\langle o, t \rangle$ . The system is *secure* iff every subject  $s$  manipulates every object  $o$  only through an authorized method type  $t$  as specified in the access rule  $\langle s, o, t \rangle$ . In the discretionary model, the access rules are specified in a distributed manner. A subject can grant other subjects some access rights granted to the subject. The relational database systems [9] adopt the discretionary model. However, the access control model implies the confinement problem [5]. Here, even if each subject is assumed to manipulate objects so as to satisfy the access rules, illegal information flow between subjects and objects may occur.

The lattice-based model [1, 3] is proposed to keep the information flow legal. Each entity  $e_i$  is given one *security class*  $\lambda(e_i)$ . A security class  $s_1$  can flow to  $s_2$  ( $s_1 \rightarrow s_2$ ) iff information in an entity of  $s_1$  can flow into an entity of  $s_2$ .  $s_1$  and  $s_2$  are *equivalent* ( $s_1 \equiv s_2$ ) iff  $s_1 \rightarrow s_2$  and  $s_2 \rightarrow s_1$ . For every pair of security classes  $s_1$  and  $s_2$  in  $S$ ,  $s_1 \prec s_2$  iff  $s_1 \rightarrow s_2$  but  $s_2 \not\rightarrow s_1$ . Here,  $s_2$  *dominates*  $s_1$  ( $s_1 \preceq s_2$ ) iff  $s_1 \prec s_2$  or  $s_1 \equiv s_2$ .  $s_1 \preceq s_2$  means that  $s_2$  is more sensitive than  $s_1$ .  $\cup$  and  $\cap$  are the least upper bound (*lub*) and the greatest lower bound (*glb*), respectively. In the mandatory model [1, 7], access rules are defined so as to satisfy " $\preceq$ ". If  $s$  reads  $o$ , the information in  $o$  is derived by  $s$ , i.e. information in  $o$  flows to  $s$ . Hence,  $\lambda(s) \succeq \lambda(o)$  is required to hold.  $s$  writes  $o$  only if  $\lambda(s) \preceq \lambda(o)$  holds. Lastly,  $s$  modifies  $o$  only if  $\lambda(s) \equiv \lambda(o)$ .

## 2.2 Role-based model

An information system is considered to be a collection of roles which users play in the application. In the enterprise, a role shows a job function and each individual is assigned a job. In the role-based model [8, 11], each role  $R$  is modeled to be a collection of access rights  $\{(o_1, t_1), \dots, (o_m, t_m)\}$ . That is, job function is composed of the methods  $t_1, \dots, t_m$ . Each individual  $s$  plays a role  $R$  in the enterprise. This is modeled to bind a subject  $s$  to a role  $R$ . In the access control model, each subject  $s$  is assigned each access right  $(o_i, t_i)$ . Hence, an access rule is specified in a form  $(s, R)$  where  $s$  shows a subject and  $R$  indicates a role. The inclusion hierarchy among the roles is also discussed.

## 2.3 Object-based model

In the object-based system, each object  $o_i$  supports more abstract level of data structure and methods than read and write on a simple object like file. In addition,  $o_i$  is *encapsulated* so that  $o_i$  can be manipulated only through the methods supported by  $o_i$ .

First, methods are assumed to be unnested. An object  $s$  sends a request message  $q$  of  $op_i$  to an object  $o_i$ . On receipt of  $q$ ,  $o_i$  computes  $op_i$  and sends the response  $r$  back to  $s$ .  $q$  and  $r$  carry the input and output of  $op_i$ , respectively. If  $op_i$  changes the state of  $o_i$  by using the input, the data in  $s$  may flow into  $o_i$  if  $q$  carries some data in  $s$ . If  $op_i$  derives data from  $o_i$  and then returns the data to  $s$ , the data in  $o_i$  may flow out to  $s$  if  $r$  carries the data derived from  $o_i$  by  $op_i$ . Thus, input and output of  $op_i$  have to be discussed to clarify the information flow relation between  $s$  and  $o_i$ . Each method  $op_i$  of  $o_i$  is characterized in terms of input ( $I_i$ ), output ( $O_i$ ), and state transition of  $o_i$ . The input  $I_i$  exists if some data flows from  $s$  to  $o_i$ , e.g. the request of  $op_i$  includes the data. The output  $O_i$  exists if some data in  $o_i$  flows out to  $s$ . In this paper, the communication among objects is assumed to be *secure*. Only data stored in  $o_i$  can flow out from  $o_i$  to  $s$  and the data in  $s$  can flow to  $o_i$  in the computation of the method of  $o_i$ .

Each method  $op_i$  is classified into one *flow* type  $\tau(op_i)$  [Figure 1]: *non-flow* (NF), *flow-in* (FI),

*flow-out* (FO), and *flow-in/out* (FIO). An NF method  $op_i$  implies no information flow from or to  $o_i$ . In addition,  $op_i$  does not change  $o_i$ . Even if the input data  $I_i$  exists, no information in  $s$  flows to  $o_i$  unless  $op_i$  changes  $o_i$ . Similarly, no data in  $o_i$  flows out to  $s$  unless the output data  $O_i$  is derived from  $o_i$ . An FI method  $op_i$  changes  $o_i$  by using  $I_i$  where data in  $s$  may flow into  $o_i$ . *write* is FI. In addition,  $o_i$  is updated without  $I_i$ . For example, a *count-up* method has no input but changes the *counter*. An FO method  $op_i$  does not change  $o_i$ . Since the output  $O_i$  of  $op_i$  carries data in  $o_i$  to  $s$ , data in  $o_i$  may flow to  $s$ . *read* is FO. An FIO method  $op_i$  changes  $o_i$  by using  $I_i$  and sends  $O_i$  including data in  $o_i$  back to  $s$ . Not only data in  $s$  may flow into  $o_i$  but also data in  $o_i$  may flow out to  $s$ . In *modify*,  $s$  first reads  $O_i$  in  $o_i$  and writes to  $o_i$ . FIO may not carry  $I_i$  like FI. The mandatory access rule is extended as follows [10].

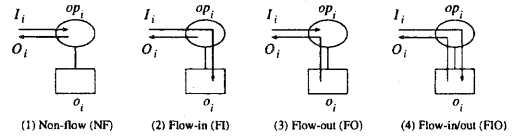


Figure 1: Information flow.

**[Extended access rules]** The subject  $s$  can manipulate  $o_i$  by  $op_i$  of  $o_i$  according to the following rules.

- (1)  $\tau(op_i) \in \{NF, FI\}$  only if  $\lambda(s) \preceq \lambda(o_i)$ .
- (2)  $\tau(op_i) \in \{NF, FO\}$  only if  $\lambda(s) \succeq \lambda(o_i)$ .
- (3)  $\tau(op_i) \in \{NF, FI, FO, FIO\}$  only if  $\lambda(s) \equiv \lambda(o_i)$ .□

The types of methods and the security class  $\lambda(o_i)$  of  $o_i$  are specified when  $o_i$  is defined based on the semantics of  $o_i$ . Each time  $s$  invokes  $op_i$ ,  $op_i$  is accepted to be computed on  $o_i$  if  $\tau(op_i)$  and  $\lambda(o_i)$  satisfy the access rules.

**[Example 1]** Suppose WWW [2] server object  $w$  is manipulated by two hosts  $h_1$  and  $h_2$ . Here,  $w$  supports *GET* and *POST* methods. *GET* is an FO type method because the output data is derived from  $w$ . *POST* is FI because  $w$  is updated by using the input data. If  $\lambda(h_1) \preceq \lambda(w)$  and

$\lambda(h_2) \succeq \lambda(w)$ ,  $h_1$  can *POST* but cannot *GET* data in  $w$ , and  $h_2$  can *GET* but cannot *POST* data in  $w$ .  $w$  can also support abstract methods like Common Gateway Interface (CGI).  $\square$

### 3 Purpose-oriented Model

#### 3.1 Purpose concept

We assume a pair of methods  $op_1$  and  $op_2$  supported by an object  $o$  can exchange data only through the state of  $o_i$ . If data  $d$  flowing from an object  $o_i$  to another object  $o_j$  is neither derived from  $o_i$  nor stored in  $o_j$ , it is meaningless to consider the information flow from  $o_i$  to  $o_j$ . If data derived from  $o_i$  is stored in  $o_j$ , the data may flow out to other objects.

Let us consider a person object  $p$  and a bank object  $b$ . In the access control model,  $p$  can derive money from  $b$  for any purpose if an access rule  $\langle p, b, withdraw \rangle$  is specified. Suppose a person  $p$  can withdraw money from a bank object  $b$  for the house-keeping. However,  $p$  cannot get money from  $b$  to go drinking. Thus, it is critical to consider a *purpose* for which a subject  $s$  manipulates an object  $o_i$  by a method  $t_i$  [10, 12].

**[Purpose-oriented (PO) rule]** An access rule  $\langle o_i : op_i, o_{ij} : op_{ij} \rangle$  means that an object  $o_i$  can manipulate another object  $o_{ij}$  through a method  $op_{ij}$  invoked by  $op_i$  of  $o_i$ .  $\square$

In the PO rule,  $op_i$  shows a *purpose* for which  $o_i$  manipulates  $o_{ij}$  by  $op_{ij}$ . Here,  $o_i$  and  $o_{ij}$  are named *parent* and *child* objects of the access rule, respectively.

**[Example 2]** If a person object  $p$  can withdraw money from a bank account  $b$  [Figure 2], an access rule  $\langle p : house\text{-}keeping, b : withdraw \rangle$  is specified. The method *house-keeping* of the object  $p$  is allowed to invoke the method *withdraw* of  $b$ . Here, *house-keeping* shows the purpose for withdrawing money from  $b$ . The method *drinking* cannot invoke *withdraw* since  $\langle p : drinking, b : withdraw \rangle$  is not specified.  $\square$

#### 3.2 Purpose in role

A role is specified in a collection of access rights in the role-based model. We would like to extend the purpose-oriented access control to the role-based model. In the object-based system, methods are invoked in a nested manner. In the rela-

tional database system like Sybase [9], the ownership chain method is adopted. Here, suppose that a subject  $s$  invokes a method  $op_1$  on an object  $o_1$  and then  $op_1$  invokes another method  $op_2$  on an object  $o_2$ . If  $o_2$  has the same owner as  $o_1$  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  $\langle o_2, op_2 \rangle$ . 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 right to subject in other objects. If the owners of the objects to be invoked are different, a subject  $s$  is checked if  $s$  is granted an access right to objects. In this paper, we take an approach where access rules are specified for a pair of autonomous objects  $o_i$  and  $o_j$ .

Each method  $op_i$  of an object  $o_i$  is granted a role  $R_i = \{ \langle op_{i1}, o_{i1} \rangle, \dots, \langle op_{ih_i}, o_{ih_i} \rangle \}$ . This means,  $op_i$  can invoke a method  $op_{ij}$  of an object  $o_{ij}$ . In turn,  $op_{ij}$  may be granted a role  $R_{ij} = \{ \langle op_{ij1}, o_{ij1} \rangle, \dots, \langle op_{ijh_{ij}}, o_{ijh_{ij}} \rangle \}$ .  $op_{ij}$  can invoke a method  $op_{ijk}$  of  $o_{ijk}$  if  $op_{ij}$  is granted the role  $R_{ij}$ . In Figure 2, it is significant to discuss what role the person  $p$  plays in a community when considering a purpose to manipulate the bank object  $b$ . For example, if  $p$  plays a role of *entertainment* in the community,  $p$  can get money from  $b$  for the purpose drinking. An access rule has to specify 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]**  $\langle R : o_i : op_i, R_i \rangle$  where  $R_i = \{ \langle op_{i1}, o_{i1} \rangle, \dots, \langle op_{ih_i}, o_{ih_i} \rangle \}$ .  $\square$

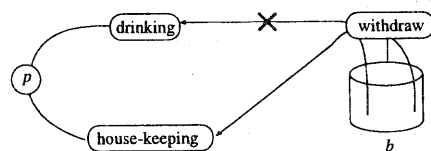


Figure 2: Purpose-oriented access control.

### 3.3 Invocation sequence

We assume that methods  $op_1$  and  $op_2$  are computed on objects  $o_1$  and  $o_2$ , respectively [Figure 3]. We also assume information flow constraints  $\lambda(p_1) \preceq \lambda(o_1)$  and  $\lambda(p_1) \succeq \lambda(o_2)$  hold on for  $p_1$  and  $\lambda(p_2) \succeq \lambda(o_1)$  for  $p_2$ . This means a person  $p_1$  can compute the methods  $op_1$  of  $o_1$  and  $op_2$  of  $o_2$ . On the other hand, a person  $p_2$  can compute  $op_1$  of  $o_1$ . If flow types of  $op_1$  and  $op_2$  are FIO and FI, respectively, an illegal information flow may occur because  $p_2$  can obtain indirectly the data of  $o_2$  although  $p_2$  is not allowed [Figure 3]. This is the confinement problem [5] which the access control model cannot resolve as presented before.

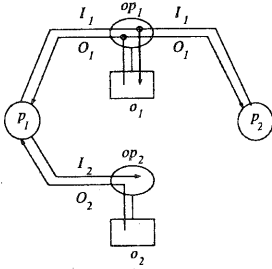


Figure 3: Illegal information flow

Suppose that a pair of objects  $o_1$  and  $o_2$  support only *read* and *write* methods. These methods are not nested. Roles  $R_1 = \{\langle read, o_1 \rangle, \langle write, o_1 \rangle, \langle write, o_2 \rangle, \langle read, o_2 \rangle\}$  and  $R_2 = \{\langle read, o_2 \rangle\}$  are assumed to be hold. Here,  $R_2 \subseteq R_1$ .  $p_1$  is granted the role  $R_1$  and  $p_2$  is granted  $R_2$  [Figure 4]. That is,  $p_1$  can manipulate  $o_1$  and  $o_2$  but  $p_2$  cannot write  $o_1$  while reading  $o_2$ . As discussed in the confinement problem,  $p_2$  may illegally obtain data in  $o_1$  if  $p_1$  and  $p_2$  are granted  $R_1$  and  $R_2$ , respectively. First, suppose that  $p_1$  writes data in  $o_2$  after reading data from  $o_1$ . Here,  $p_1$  can indirectly obtain the data stored in  $o_1$ . Next, suppose that  $p_1$  reads data from  $o_1$  after writing data to  $o_2$ . Here, no data in  $o_2$  flows to  $o_1$ . Therefore, there occurs no illegal information flow from  $o_1$  to  $o_2$  if  $p_1$  writes  $o_2$  before reading  $o_1$ . This example shows that whether or not il-

legal information flow occurs depends on how  $p_1$  issues methods to  $o_1$  and  $o_2$ .

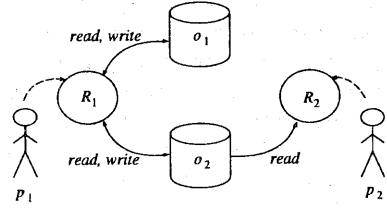


Figure 4: Information flow

We assume that an object  $o_i$  has an access list  $AL(o_i) = \{\langle s, t \rangle \mid \langle s, o_i, t \rangle \text{ is authorized}\}$ .  $AL(o_i)$  indicates what subject can manipulate  $o_i$  by what method.

A subject  $s$  manipulates a directed graph  $H(s)$  to check whether or not illegal information flow to occur.  $H(s)$  is composed of two kinds of nodes, subject node and object node. Initially,  $H(s)$  includes a subject node showing  $s$ . If the subject  $s$  would manipulate an object  $o_i$  by issuing a method  $t_i$ ,  $s$  manipulates the graph  $H(s)$  as follows:

#### [Graph constructions]

1. An object node  $o_i$  is created in  $H(s)$ .
  - a. If  $t$  is *read*, a directed edge from  $o_i$  to  $s$  ( $o_i \rightarrow s$ ) is created in  $H(s)$ .
  - b. If  $t$  is *write*, a directed edge from  $s$  to  $o_i$  ( $s \rightarrow o_i$ ) is created in  $H(s)$ .
2. For each element  $\langle s_j, t_j \rangle$  in  $AL(o_i)$ , a subject node  $s_j$  is created in  $H(s)$  unless  $H(s)$  includes  $s_j$ .
  - a. If  $t_j$  is *read*, a directed edge  $o_i \rightarrow s_j$  is created.
  - b. If  $t_j$  is *write*, a directed edge  $s_j \rightarrow o_i$  is created.  $\square$

For every nodes  $a$ ,  $b$ , and  $c$ , if  $a \rightarrow b$  and  $b \rightarrow c$  in  $H(s)$ , an edge  $a \rightarrow c$  is created in  $H(s)$ . If the following condition holds,  $s$  is not allowed to manipulate  $o_i$  by  $t_i$ .

**[Access condition]** For every subject  $s_j$ ,  $o_j \rightarrow s_j$  in  $H(s)$  but  $\langle s_j, o_j, read \rangle$  is not authorized.  $\square$

#### 4 Concluding Remarks

The information system is composed of multiple autonomous objects. Each object supports more abstract level of methods than *read* and *write*. The *purpose-oriented* access control model discusses why an object manipulates other objects in the object-based systems while the traditional access control model discusses if each subject can use an object by a method. In addition, the methods of the objects are invoked in a nested manner. The access rules have to satisfy the information flow relation among objects. The occurrence of illegal information flow depends on in which order methods are invoked. In this paper, a *purpose* of the object to manipulate objects is modeled to be a invocation sequence of methods.

#### References

- [1] Bell, D. E. and LaPadula, L. J., "Secure Computer Systems: Mathematical Foundations and Model," *Mitre Corp. Report No. M74-244*, 1975.
- [2] Berners-Lee, T., Fielding, R., and Frystyk, H., "Hypertext Transfer Protocol - HTTP/1.0," *RFC-1945*, 1996.
- [3] Denning, D. E., "A Lattice Model of Secure Information Flow," *Comm. of the ACM*, Vol. 19, No. 5, 1976, pp. 236-243.
- [4] Denning, D. E. and Denning, P. J., *Cryptography and Data Security*, Addison-Wesley, 1982.
- [5] Lampson, B. W., "A Note on the Confinement Problem," *Comm. of the ACM*, Vol. 16, No. 10, 1973, pp. 613-615.
- [6] Object Management Group Inc., "The Common Object Request Broker : Architecture and Specification," Rev. 2.1, 1997.
- [7] Sandhu, R. S., "Lattice-Based Access Control Models," *IEEE Computer*, Vol. 26, No. 11, 1993, pp. 9-19.
- [8] 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.
- [9] Sybase, Inc., "Sybase Adaptive Server Enterprise Security Administration," 1997.
- [10] 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.
- [11] 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.
- [12] Yasuda, M., Higaki, H., and Takizawa, M., "A Purpose-Oriented Access Control Model for Information Flow Management," to appear in *Proceeding of 14th Int'l Information Security Conf. (IFIP/SEC'98)*, 1998.