Vol. 43 No. 2 IPSJ Journal Feb. 2002

Regular Paper

Information Flow Control among Objects in Role-based Access Control Model

Vlad Ingar Wietrzyk,† Keiji Izaki,†† Katsuya Tanaka†† and Makoto Takizawa††

Various kinds of applications have to be secure in an object-based model. The secure system is required to not only protect objects from illegally manipulated but also prevent illegal information flow among objects. In this paper, we discuss how to resolve illegal information flow among objects in a role-based model. We define safe roles where no illegal information flow occurs. In addition, we discuss how to safely perform transactions with unsafe roles. We discuss an algorithm to check if illegal information flow occurs each time a method is performed.

1. Introduction

Various kinds of object-based systems like object-oriented database systems, JAVA [7] and CORBA [14] are widely used for applications. Object-based systems are composed of multiple objects cooperating to achieve some objectives by passing messages. An object is an encapsulation of data and methods for manipulating the data. Methods are invoked on objects in a nested manner. The object-based system are required to not only protect objects from illegally manipulated but also prevent illegal information flow among objects in the system.

In the access control model [12], an access rule $\langle s, o, t\rangle$ means that a subject s is allowed to manipulate an object o in an access type t. Only access requests which satisfy the access rules are accepted to be performed. However, the *confinement* problem [13] is implied, i.e. illegal information flow occurs among subjects and objects. In the *mandatory lattice-based* model [1),3),17)], objects and subjects are classified into security classes. Legal information flow is defined in terms of the *can-flow* relation [3] between classes. Access rules are specified 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, the subject s can read the object o only if a *can-flow* relation from o to s is specified. In the role-based model [6),18),20)], a *role* is defined to be a collection of access rights, i.e. pairs of access types and objects, to denote a job function in the enterprise. Subjects are granted roles which show their jobs. In an object-based system, the methods are invoked on objects in a nested manner. The purpose-oriented model [19),21)] discusses which methods can invoke another method in the object-based system. In the paper Ref. 16), a *message filter* is used to block read and write requests if illegal information flow occurs. The authors [10] discuss what information flow to *possibly* occur among objects if subjects issue methods by the authority of the roles in case every method invocation is not nested. Methods are invoked in the nested manner in the object-based systems. Let us consider a database application in a multi-tier client-server model by using Java servlet [11]. First, an application program, i.e. a method A is invoked by a client. The program A manipulates data in a data server and then invokes an application program B in another application server. Here, a method A invokes another method B in a nested manner. Data derived by A from the data server may be included in the parameters of B and be brought to B. This is an example of information flow to occur in the nested invocation.

In this paper, we consider a role-based access control in an object-based system where methods are invoked in a nested manner. We newly discuss illegal information flow to occur among objects by transactions in the role-based access control. Objects support more abstract methods than read and write ones. First, we classify the methods supported by objects from the information flow point of view. We define a *safe* role where no illegal information flow occurs by performing any transaction with the role.

† School of Computing and Information Technology, University of Western Sydney
†† Department of Computers and Systems Engineering, Tokyo Denki University

In addition, we discuss an algorithm to check for each method issued by a transaction if illegal information flow occurs by performing the method. By using the algorithm, some methods issued by a transaction can be performed even if the transaction is in a session with an unsafe role. Data flowing from an object o_1 to o_2 can belong to o_2 some time after the data flows. We discuss how to manage timed information flow.

In Section 2, we classify methods from information flow point of view. In Section 3, we discuss information flow to occur in a nested invocation. In Section 4, we discuss how to resolve illegal information flow.

2. Object-based Systems

An object-based system is composed of objects which are encapsulation of data and methods. A transaction invokes a method by sending a request message to an object. The method is performed on the object and then the response is sent back to the transaction. During the computation of the method, other methods might be invoked. Thus, methods are invoked in a nested manner.

Each subject plays a *role* in an organization. In the role-based model [6),18),20)], a *role* is modeled to be a set of *access rights*. An access right $\langle o,\ t\rangle$ means that t can be performed on the object o. A subject s is granted a role which shows its job function in an enterprise. This means that the subject s can perform a method t on an object o if $\langle o,\ t\rangle \in r$. If a subject s is in a *session* with r, s can issue methods in r. Each subject can be in a session with at most one role.

Each method t on an object o is characterized by the following parameters:

1. *Input type* $= I$ if the method t has input data in the parameter, else N.
2. *Manipulation type* $= M$ if the object o is changed by t, else N.
3. *Derivation type* $= D$ if data is derived from o by t, else N.
4. *Output type* $= O$ if data is returned to the invoker of t, else N.

Each method t of an object o is characterized by a *method type* $mtype(t) = \alpha_1\alpha_2\alpha_3\alpha_4$, where input $\alpha_1 \in \{I,\ N\}$, manipulation $\alpha_2 \in \{M,\ N\}$, derivation $\alpha_3 \in \{D,\ N\}$, and output $\alpha_4 \in \{O,\ N\}$. For example, a method class "$IMNN$" shows a method which carries data in the parameters to an object and changes the state of the object. Here, N is omitted in the method type. For example, "IM" shows $IMNN$. Especially, "N" shows a type $NNNN$. Let MC be a set $\{IMDO$, IDO, IMO, IO, IMD, ID, IM, I, MDO, DO, MO, O, MD, D, M, $N\}$ of sixteen possible method types. A *counter* object c supports methods *display*(*dsp*), *increment*(*inc*), and *decrement*(*dec*). $mtype(dsp) = DO$ and $mtype(inc) = mtype(dec) = IMD$. Here, DO means D and O. A notation "$\beta_1, \ldots, \beta_k \in mtype(t)$" ($k \leq 4$) shows $mtype(t) = \alpha_1\alpha_2\alpha_3\alpha_4$ and $\beta_i \in \{\alpha_1,\ \alpha_2,\ \alpha_3,\ \alpha_4\}$ ($i \leq k$). For example, $I \in mtype(inc)$ and $ID \in mtype(dec)$. In the object-based systems, objects are created and dropped. $IM \in mtype(created)$ and $N \in mtype(drop)$. The method type $mtype(t)$ is specified for each method t by the owner of the object.

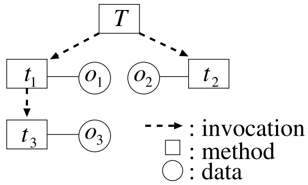
We assume that each subject does not have any persistent storage. That is, the subject does not keep in record data obtained from objects. The subject issues one or more than one method to objects. A sequence of methods issued by the subject is referred to as a *transaction*, which is a unit of work. Each *transaction* T can be in a session with only one role r. A transaction has a temporary memory. Data which the transaction derives from objects may be stored in the temporary memory. On completion of the transaction, the memory is released. Any transaction does not share data with the other transactions. In this paper, objects show persistent objects.

Suppose T with a role r invokes a method t_1 on an object o_1 since $\langle o_1, t_1\rangle \in r$. Suppose t_1 invokes another method t_2 on an object o_2. Here, we assume $\langle o_2, t_2\rangle \in r$. That is, $\langle o, t\rangle \in r$ for every method t invoked on an object o in T.

3. Nested Invocation

3.1 Invocation Tree

Suppose a transaction T invokes a method t_1 on an object o_1 and a method t_2 on an object o_2. Then, t_1 invokes a method t_3 on an object o_3. The invocations of methods are represented in a tree form named *invocation tree* as shown in **Fig. 1**. Each node $\langle o, t\rangle$ shows a method t invoked on an object o in the transaction T. A dotted directed edge from a parent to a child shows that the parent invokes the child. A notation "$\langle o_1, t_1\rangle \vdash_T \langle o_2, t_2\rangle$" means that a method t_1 on an object o_1 invokes t_2 on



method	mtype
t_1	O
t_2	IM
t_3	DO

Fig. 1 Invocation tree.

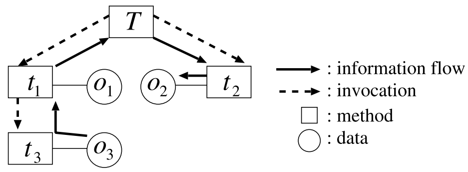


Fig. 2 Information flow.

o_2 in the transaction T. A node $\langle \_, T\rangle$ shows a root of invocation tree of T. Here, *mtype*(T) is N according to the assumption.

If a method serially invokes multiple methods, the left-to-right order of nodes shows an invocation sequence of methods, i.e., tree is ordered. Suppose $\langle o_1, t_1\rangle \vdash_T \langle o_2, t_2\rangle$ and $\langle o_1, t_1\rangle \vdash_T \langle o_3, t_3\rangle$ in an invocation tree of a transaction T. If t_1 invokes t_2 before t_3, $\langle o_2, t_2\rangle$ *precedes* $\langle o_3, t_3\rangle$ ($\langle o_2, t_2\rangle \prec_T \langle o_3, t_3\rangle$). In addition, $\langle o_4, t_4\rangle \prec_T \langle o_3, t_3\rangle$ if $\langle o_2, t_2\rangle \vdash_T \langle o_4, t_4\rangle$. $\langle o_2, t_2\rangle \prec_T \langle o_4, t_4\rangle$ if $\langle o_3, t_3\rangle \vdash_T \langle o_4, t_4\rangle$. The relation "$\prec_T$" is transitive. T invokes t_1 before t_2 as shown in Fig. 1. Here, $\langle o_1, t_1\rangle \prec_T \langle o_2, t_2\rangle$ and $\langle o_3, t_3\rangle \prec_T \langle o_2, t_2\rangle$.

3.2 Information Flow

Suppose *mtype*(t_3) = DO, *mtype*(t_2) = IM, and *mtype*(t_1) = O in Fig. 1. In a transaction T, data is derived from an object o_3 through the method t_3. The data is forwarded to t_1 as the response of t_3. The data is brought to t_2 as the input parameter, and is stored into o_2 through t_2. Thus, the information in o_3 is brought to o_2. A straight arc indicates the information flow in **Fig. 2**. This example shows that information flow among objects may occur in a nested invocation.

[Definition] Suppose a pair of methods t_1 and t_2 on objects o_1 and o_2, respectively, are invoked in a transaction T.

1. Information *passes down* from $\langle o_1, t_1\rangle$ to $\langle o_2, t_2\rangle$ in T ($\langle o_1, t_1\rangle \stackrel{T}{\rightharpoondown} \langle o_2, t_2\rangle$) iff t_1 invokes t_2 ($\langle o_1, t_1\rangle \vdash_T \langle o_2, t_2\rangle$) and $I \in$ *mtype*(t_2), or $\langle o_1, t_1\rangle \stackrel{T}{\rightharpoondown} \langle o_3, t_3\rangle \stackrel{T}{\rightharpoondown} \langle o_2, t_2\rangle$ for some $\langle o_3, t_3\rangle$ in T.
2. Information *passes up* from $\langle o_1, t_1\rangle$ to $\langle o_2, t_2\rangle$ in T ($\langle o_1, t_1\rangle \stackrel{T}{\rightharpoonup} \langle o_2, t_2\rangle$) iff $\langle o_2, t_2\rangle \vdash_T \langle o_1, t_1\rangle$ and $O \in$ *mtype*(t_2), or $\langle o_1, t_1\rangle \stackrel{T}{\rightharpoonup} \langle o_3, t_3\rangle \stackrel{T}{\rightharpoonup} \langle o_2, t_2\rangle$ for some $\langle o_3, t_3\rangle$ in T. □

[Definition] Information *passes* from $\langle o_1, t_1\rangle$ to $\langle o_2, t_2\rangle$ in an ordered transaction T ($\langle o_1, t_1\rangle \xrightarrow[O]{T} \langle o_2, t_2\rangle$) iff $\langle o_1, t_1\rangle \stackrel{T}{\rightharpoondown} \langle o_2, t_2\rangle$ or $\langle o_1, t_1\rangle \stackrel{T}{\rightharpoonup} \langle o_2, t_2\rangle$, $\langle o_1, t_1\rangle \stackrel{T}{\rightharpoondown} \langle o_3, t_3\rangle \stackrel{T}{\rightharpoonup} \langle o_2, t_2\rangle$ and $\langle o_1, t_1\rangle \prec_T \langle o_2, t_2\rangle$, or $\langle o_1, t_1\rangle \xrightarrow[O]{T} \langle o_3, t_3\rangle \xrightarrow[O]{T} \langle o_2, t_2\rangle$ for some $\langle o_3, t_3\rangle$ in T. □

[Definition] Information *passes* from $\langle o_1, t_1\rangle$ to $\langle o_2, t_2\rangle$ in an unordered transaction T ($\langle o_1, t_1\rangle \xrightarrow[U]{T} \langle o_2, t_2\rangle$) iff $\langle o_1, t_1\rangle \stackrel{T}{\rightharpoondown} \langle o_2, t_2\rangle$ or $\langle o_1, t_1\rangle \stackrel{T}{\rightharpoonup} \langle o_2, t_2\rangle$, or $\langle o_1, t_1\rangle \xrightarrow[U]{T} \langle o_3, t_3\rangle \xrightarrow[U]{T} \langle o_2, t_2\rangle$ for some $\langle o_3, t_3\rangle$ in T. □

Suppose t_1 is invoked before t_2, i.e. $\langle o_1, t_1\rangle \prec_T \langle o_2, t_2\rangle$ in Fig. 2. $\langle o_3, t_3\rangle \stackrel{T}{\rightharpoonup} \langle o_1, t_1\rangle \stackrel{T}{\rightharpoonup} \langle \_, T\rangle \stackrel{T}{\rightharpoondown} \langle o_2, t_2\rangle$. $\langle o_1, t_1\rangle \not\xrightarrow[O]{T} \langle o_2, t_2\rangle$ if $\langle o_2, t_2\rangle \prec_T \langle o_1, t_1\rangle$. However, $\langle o_1, t_1\rangle \xrightarrow[U]{T} \langle o_2, t_2\rangle$. A relation "$\stackrel{T}{\rightarrow}$" shows "$\xrightarrow[O]{T}$" or "$\xrightarrow[U]{T}$". A notation "$o_1 \stackrel{T}{\rightarrow} o_2$" shows "$\langle o_1, t_1\rangle \stackrel{T}{\rightarrow} \langle o_2, t_2\rangle$" for some methods t_1 and t_2. Here, $T \stackrel{T}{\rightarrow} o$ and $o \stackrel{T}{\rightarrow} T$ indicate $\langle \_, T\rangle \stackrel{T}{\rightarrow} \langle o, t\rangle$ and $\langle o, t\rangle \stackrel{T}{\rightarrow} \langle \_, T\rangle$, respectively. According to the definitions, $o_1 \xrightarrow[U]{T} o_2$ if $o_1 \xrightarrow[O]{T} o_2$.

[Definition] $\langle o_1, t_1\rangle$ *flows into* $\langle o_2, t_2\rangle$ in a transaction T ($\langle o_1, t_1\rangle \stackrel{T}{\Rightarrow} \langle o_2, t_2\rangle$) iff $\langle o_1, t_1\rangle \stackrel{T}{\rightarrow} \langle o_2, t_2\rangle$, $D \in$ *mtype*(t_1), and $M \in$ *mtype*(t_2). □

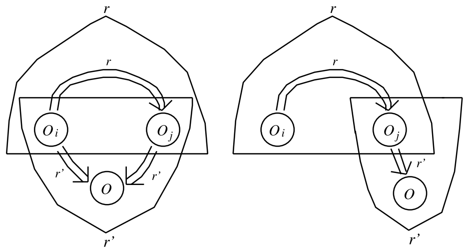
In Fig. 2, $\langle o_3, t_3\rangle \stackrel{T}{\Rightarrow} \langle o_2, t_2\rangle$ where $\langle o_3, t_3\rangle$ is a *source* and $\langle o_2, t_2\rangle$ is a *sink*. Here, data in o_3 flows into o_2. "$\langle o_1, t_1\rangle \stackrel{T}{\Rightarrow} \langle o_2, t_2\rangle$" can be abbreviated as $o_1 \stackrel{T}{\Rightarrow} o_2$. $T \stackrel{T}{\Rightarrow} o$ if $T \stackrel{T}{\rightarrow} o$ and o is a sink. $o \stackrel{T}{\Rightarrow} T$ if $o \stackrel{T}{\rightarrow} T$ and o is a source. $o_1 \stackrel{r}{\Rightarrow} o_2$ for a role r iff $o_1 \stackrel{T}{\Rightarrow} o_2$ for some transaction T with r.

[Definition] Information in o_i *flows into* o_j ($o_i \Rightarrow o_j$) iff $o_i \stackrel{r}{\Rightarrow} o_j$ for some role r or $o_i \Rightarrow o_k \Rightarrow o_j$ for some object o_k. □

$o_i \Rightarrow o_j$ is *primitive* for a role r if $o_i \stackrel{r}{\Rightarrow} o_j$. $o_i \Rightarrow o_j$ is *transitive* for a role r iff $o_i \stackrel{r}{\Rightarrow} o_j$ is not primitive for r, i.e. $o_i \Rightarrow o_k \stackrel{r}{\Rightarrow} o_j$ but $o_i \stackrel{r}{\not\Rightarrow} o_j$ for some o_k. If $o_i \Rightarrow o_j$ is transitive for r, a transaction T with r may get data in o_i through o_j even if T is not allowed to get data from o_i.

[Definition] "$o_i \Rightarrow o_j$" is *illegal* iff $o_i \Rightarrow o_j$ is transitive for some role r. □

[Definition] A role r *threatens* another role r_1



(1) Safe (2) Unsafe

Fig. 3 Safeness.

iff for some objects o_i, o_j, and o, $o_i \overset{r_1}{\Rightarrow} o_j \overset{r}{\Rightarrow} o$ and $o_i \Rightarrow o$ is transitive for r. $\square$

Suppose information in o_i might flow into an object o_j ($o_i \overset{r_1}{\Rightarrow} o_j$) by performing a transaction T_1 with a role r_1. Even if a transaction T_2 is not granted a role to derive data from o_i, T_2 can get data in o_i from o_j if T_2 is granted a role r to derive data from o_j. Thus, if there is another role r threatening a role r_1, illegal information flow might occur if some transaction with r is performed.

[Definition] "$o_i \overset{r}{\Rightarrow} o_j$" is *safe* for a role r iff r is not threatened by any role. $\square$

Figure 3 shows a system including a pair of roles r and r' where $o_i \overset{r}{\Rightarrow} o_j$. For another role r', $o_i \overset{r'}{\Rightarrow} o$ and $o_j \overset{r'}{\Rightarrow} o$ in Fig. 3 (1). Since r' does not threaten r, $o_i \overset{r}{\Rightarrow} o_j$ is safe. In Fig. 3 (2), $o_j \overset{r'}{\Rightarrow} o$ but $o_i \overset{r'}{\not\Rightarrow} o$. However, T is not allowed to derive data from o_i. Hence, r' threatens r and $o_i \overset{r}{\Rightarrow} o_j$ is not safe. $o_i \Rightarrow o$ is illegal. This is a *confinement* problem on roles. It is noted that o may show a transaction. For example, the transaction T manipulates o_j through a *DO* method t. Here, $o_i \overset{r'}{\Rightarrow} T$.

[Definition] A role r is *safe* iff r neither threatens any role nor is threatened by any role. $\square$

A transaction is *safe* iff the transaction is in a session with a *safe* role. An *unsafe* transaction is in a session with an *unsafe* role.

[Theorem] If every transaction is safe, no illegal information flow occurs. $\square$

That is, no illegal information flow occurs if every role is safe. The paper [10] discusses an algorithm to check whether or not illegal information flow possibly occurs if the method is performed.

3.3 Invocation Models

Suppose a transaction T is in a session with a role r. It is not easy to make clear what transactions exist for each role and how each transaction invokes methods. Hence, we first discuss a basic (B) model where there is one transaction T_r which is in a session with a role r and invokes all the methods in r, i.e. $\langle \_, T_r \rangle \vdash_{T_r} \langle o, t \rangle$ for every $\langle o, t \rangle$ in the role r. An invocation tree of T_r is an unordered, two-level tree. Here, $\langle \_, T_r \rangle \overset{r}{\rightarrow} \langle o, t \rangle$ if $\langle o, t \rangle \in r$ and $I \in mtype(t)$ according to the definition of $\rightarrow$. $\langle o, t \rangle \overset{r}{\rightarrow} \langle \_, T \rangle$ if $\langle o, t \rangle \in r$ and $O \in mtype(t)$. $\overset{r}{\rightarrow}$ is transitive. $\langle o, t \rangle \overset{r}{\Rightarrow} \langle \_, T \rangle$ iff $\langle o, t \rangle \overset{r}{\rightarrow} \langle \_, T_r \rangle$ and $D \in mtype(t)$. $\langle \_, T_r \rangle \overset{r}{\Rightarrow} \langle o, t \rangle$ iff $\langle \_, T_r \rangle \overset{r}{\rightarrow} \langle o, t \rangle$ and $M \in mtype(t)$. $\langle o_1, t_1 \rangle \overset{r}{\Rightarrow} \langle o_2, t_2 \rangle$, iff $\langle o_1, t_1 \rangle \overset{r}{\rightarrow} \langle \_, T_r \rangle$ and $\langle \_, T_r \rangle \overset{r}{\rightarrow} \langle o_2, t_2 \rangle$ and $D \in mtype(t_1)$ and $M \in mtype(t_2)$. Here, $r \overset{r}{\Rightarrow} o$ and $o \overset{r}{\Rightarrow} r$ show "$\langle \_, T_r \rangle \overset{r}{\Rightarrow} \langle o, t \rangle$" and "$\langle o, t \rangle \overset{r}{\Rightarrow} \langle \_, T_r \rangle$" for some method t, respectively. "$\underset{B}{\overset{r}{\Rightarrow}}$" shows "$\overset{r}{\Rightarrow}$" in the B model.

Next, suppose a collection of transactions are *a priori* defined. $Tr(r)$ is a set of transactions which are in sessions with r. Let $N(T)$ be a set $\{\langle o, t \rangle \mid t$ is invoked on o in a transaction $T\}$ and $Al(r)$ be $\{\langle o, t \rangle \mid \langle o, t \rangle \in N(T)$ for every transaction T in $Tr(r)\}$ ($\subseteq r$). Suppose two transactions T_1 and T_2 are in sessions with a role r. T_1 invokes a method t_1 on an object o_1. T_2 invokes a method t_2 on an object o_2 and then t_2 invokes a method t_3 on an object o_3 and t_4 on o_4. Here, $Tr(r) = \{T_1, T_2\}$. $N(T_1) = \{\langle o_1, t_1 \rangle\}$, and $N(T_2) = \{\langle o_2, t_2 \rangle, \langle o_3, t_3 \rangle, \langle o_4, t_4 \rangle\}$. $Al(r) = N(T_1) \cup N(T_2)$. There are two cases: invocation sequence of methods is *a priori* fixed or not, i.e. invocation tree of each transaction is ordered (O) or unordered (U). In the basic (B) model, T_r invokes t_1 and t_2. Since $o_1 \overset{r}{\Rightarrow} T_r \overset{r}{\Rightarrow} o_2 \overset{r}{\Rightarrow} o_3$, i.e. information in o_1 possibly flows to o_3. In the unordered (U) and ordered (O) models, there is no information flow between o_1 and o_3, because o_1 and o_3 are manipulated by T_1 and T_2, respectively. If the transactions are not ordered, $o_4 \overset{r}{\Rightarrow} o_3$ as shown in **Fig. 4**. On the other hand, if the transactions are ordered, o_4 is manipulated before o_3. Hence, $o_4 \overset{r}{\not\Rightarrow} o_3$. $o_i \underset{U}{\overset{r}{\Rightarrow}} o_j$ if $o_i \underset{O}{\overset{r}{\Rightarrow}} o_j$. $o_i \underset{B}{\overset{r}{\Rightarrow}} o_j$ if $o_i \underset{U}{\overset{r}{\Rightarrow}} o_j$.

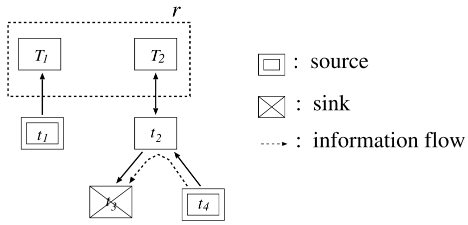


Fig. 4 Invocation trees.

If transaction are not well defined in applications, the basic (B) model is taken. If it is a priori well defined what methods are invoked in each transaction but the invocation order of methods is not a priori known, the unordered (U) model is taken. Furthermore, if it is well defined what order method are invoked in each transaction, the O model is taken. Thus, it depends on applications which model is taken.

4. Resolution of Illegal Information Flow

4.1 Flow Graph

Every safe transaction is allowed to be performed because no illegal information flow occurs. As discussed in Fig. 4, $o_1 \overset{r}{\Rightarrow} o_3$ does not hold in the U and O models even if $o_1 \overset{r}{\Rightarrow} o_3$ in the B model. $o_1 \overset{r}{\Rightarrow} o_3$ in the U model but $o_1 \overset{r}{\Rightarrow} o_3$ does not hold in the O model. This means it depends on an invocation sequence of methods whether or not illegal information flow occurs. The paper Ref. 10) discusses how to decide if a role is safe and an algorithm for each method issued by an unsafe transaction to check whether or not illegal information flow *possibly* occurs if the method is performed. However, it is not easy, possibly impossible to decide whether or not each role is safe if roles include large number of objects and roles are dynamically created and dropped. In this paper, we discuss an algorithm to check whether or not illegal information flow *necessarily* occurs if each method issued by every transaction is performed. A system maintains a following directed *flow graph* G.

[Flow graph]

1. Each node in G shows an object in the system. Here, each transaction is also an object. If an object is created, a node for the object is added in G. Initially, G includes no edge.
2. A directed edge $o_1 \rightarrow_\tau o_2$ is created if $o_1 \overset{T}{\Rightarrow} o_2$ by performing a transaction T of a role r at time τ. If $o_1 \rightarrow_{\tau_1} o_2$ already exists in G, $o_1 \rightarrow_{\tau_1} o_2$ is changed to $o_1 \rightarrow_\tau o_2$ if $\tau_1 < \tau$.
3. For each object o_3 such that $o_3 \rightarrow_{\tau_2} o_1 \rightarrow_\tau o_2$ in G,
   - 3.1 $o_3 \rightarrow_\tau o_2$ is created if there is no edge from o_3 to o_2 in G and $\tau_2 < \tau$. go to Step 2.
   - 3.2 $o_3 \rightarrow_{\tau_2} o_2$ if $o_3 \rightarrow_{\tau_3} o_2$ is already in G and $\tau_2 > \tau_3$.

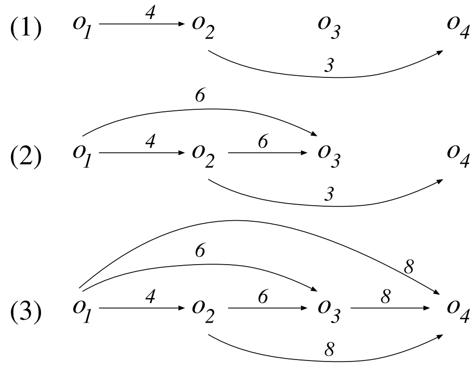


Fig. 5 Flow graph G.

□

Figure 5 shows a flow graph G including four objects o_1, o_2, o_3, and o_4. First, suppose $o_1 \rightarrow_4 o_2$ and $o_2 \rightarrow_3 o_4$ hold in G. Then, information flow $o_2 \overset{r_1}{\Rightarrow} o_3$ occurs by performing a transaction at time 6. Here, a directed edge $o_2 \rightarrow_6 o_3$ is created in G. Since $o_1 \rightarrow_4 o_2 \rightarrow_6 o_3$, information flowing to o_2 from o_1 at time 4 might flow to o_3 by the transaction. Hence, $o_1 \rightarrow_6 o_3$ since $4 < 6$ (Fig. 5 (2)). Then, $o_3 \overset{r_2}{\Rightarrow} o_4$ at time 8. $o_3 \rightarrow_8 o_4$. Since $o_1 \rightarrow_4 o_2 \rightarrow_6 o_3 \rightarrow_8 o_4$, an edge $o_1 \rightarrow_8 o_4$ is also created and another edge $o_2 \rightarrow_8 o_4$ is tried to be created. However, "$o_2 \rightarrow_3 o_4$" in G. Since $3 < 8$, the time 3 of the edge "$o_2 \rightarrow_3 o_4$" is replaced with 8 (Fig. 5 (3)). In Fig. 5 (3), information in the objects o_1, o_2, and o_3 flow into o_4. Let $In(o)$ be a set $\{o_i \mid o_i \rightarrow_\tau o \text{ in } G\}$ of objects whose information has flown into an object o. Let $Out(o)$ be a set $\{ o_i \mid o \rightarrow_\tau o_i \text{ in } G \}$ of objects whose information are flown from object o. For example, $In(o_4) = \{o_1, o_2, o_3\}$ in Fig. 5.

Suppose a method t is issued to an object o in a transaction T with a role r. Each time a method t is invoked on an object o in the transaction T, a pair $\langle o, t\rangle$ is logged in an invocation tree form into a log L_T. $\langle o, t\rangle$ shows a node of ordered invocation tree of T. A flow graph G is maintained according to the algorithm presented in preceding section. If the following condition is satisfied, the method t cannot be invoked in the object o by the transaction.

[Condition for a method *t*] (Fig. 6)

1. for every "$o_2 \rightarrow_\tau o_1$" in a flow graph G if $IM \in$ *mtype*(t) and "$o_1 \overset{T}{\Rightarrow} o$" is obtained from L_T.
2. for every "$o_2 \rightarrow_\tau o$" in G if $DO \in$ *mtype*(t).

□

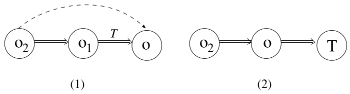


Fig. 6 Condition.

In the condition 1, data in some object o_2 might have been brought into an object o_1 ($o_2 \overset{T}{\Rightarrow} o_1$) in a transaction T before the transaction T manipulates an object o. Hence, we have to check if information in an object o_2 could flow into another object o_1. Here, if the role r includes an access right to derive data from the object o_2, a method t is allowed to be performed on the object o. Otherwise, the method t is not allowed to be performed since illegal information flow occurs. The second condition shows that a transaction T with a role r issues a method t to derive data from the object o. Here, some data in another object o_2 might have been brought to an object o by another transaction before the transaction T starts. Hence, the method t is allowed to be performed on an object o only if the transaction T is allowed to derive data from every object o_2 in the input set $In(o)$ ($o_2 \Rightarrow o$). If the method t could be performed according to the condition, the method t is logged in the log L_T of the transaction T if $DO \in mtype(t)$.

4.2 Timed Information Flow

Suppose some data in an object o_i illegally flows to another object o_j by performing a transaction T with a role r at time τ ($o_i \rightarrow_\tau o_j$ in G). Security level of data is changing time by time. After it takes some time δ, the data brought from o_i is considered to belong to o_j. δ is decided by security administrator. An edge "$o_i \rightarrow_\tau o_j$" is *aged* if $\tau + \delta < \sigma$ where σ shows the current time. Every aged edge is removed from the graph G for σ. In Fig. 5, suppose $\delta = 10$. If σ gets 15, an edge timed 4 is aged now and removed. **Figure 7** shows the flow graph G obtained here. Suppose some transaction T with a role r_1 issues a request t_3 on an object o_3 which $DO \in mtype(t_3)$ in Fig. 5 (3) but data in o_1 is not allowed to be derived. In Fig. 5 (3), T is rejected according to the conditions. However, the DO method t_3 can be performed in Fig. 7 because of no illegal information flow from o_1 to T.

Suppose an object o_3 is dropped in a flow graph G of Fig. 5 (3). Since "$o_3 \rightarrow_4 o_4$" exists in G, some data in o_3 might have been copied in o_4. Hence, only transaction which is granted to manipulate o_3 is allowed to manipulate o_4 even after o_3 is dropped.

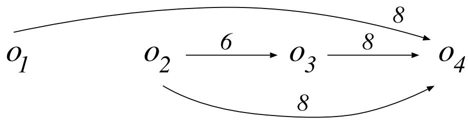


Fig. 7 Flow graph.

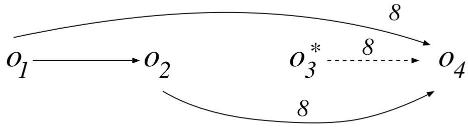


Fig. 8 Flow graph.

[Drop of an object] An object o is dropped.

1. A node o is marked.
2. Every incoming edge in $In(o)$ is removed from G.
3. Every outgoing edge in $Out(o)$ is marked.

□

Figure 8 shows a flow graph G obtained by dropping the object o_3 through the algorithm from Fig. 5 (3). The node o_3 is marked $*$. A dotted edge from o_3 to o_4 shows a marked edge. All incoming edges to o_3, i.e. "$o_1 \rightarrow_6 o_3$" and "$o_2 \rightarrow_6 o_3$" are removed from G. Here, suppose some transaction T issues a DO method t_4 on o_4. t_4 is rejected if T is not allowed to derived data from o_3 even if o_3 is dropped already, because there is still data of o_3 in o_4. Each marked edge is removed after it takes δ time units. If a marked node o does not have any outgoing edge, i.e. $Out(o) = \phi$, o is removed from G.

[Remove of aged edge]

1. For any edge "$o_i \rightarrow_\tau o_j$" in G, the edge is removed if $\tau + \delta \leq \sigma$.
2. Every marked node o_i is removed if $Out(o_i) = \phi$.

□

5. Concluding Remarks

This paper discussed an access control model for the object-based system with role concepts. We discussed how to control information flow in a system where methods are invoked in a nested manner. We first defined a safe role where no illegal information flow possibly occurs in types of invocation models; basic (B), unordered (U), and ordered (O) models. We presented the algorithm to check if each method could be performed, i.e. no illegal information flow occurs after the method is performed. By using the algorithm, some methods issued by an unsafe transaction can be performed depending on in what order a transaction performs the methods. We also discussed a case that security level is *time-variant*. Information flowing to another object can be considered to belong to the object after some time.

Another idea to protect illegal information flow is to partially order access rights in roles. The ordering relation show safe invocation sequences of methods. We would like to discuss this point in another paper.

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) Castano, S., Fugini, M., Matella, G. and Samarati, P.: *Database Security*, Addison-Wesley (1995).
3) Denning, D.E.: A Lattice Model of Secure Information Flow, *Comm. ACM*, Vol.19, No.5, pp.236–243 (1976).
4) Fausto, R., Elisa, B., Won, K. and Darrell, W.: A Model of Authorization for Next-Generation Database Systems, *ACM Trans.Database Syst.*, Vol.16, No.1, pp.88–131 (1991).
5) Ferrai, E., Samarati, P., Bertino, E. and Jajodia, S.: Providing Flexibility in Information Flow Control for Object-Oriented Systems, *Proc. 1997 IEEE Symp. on Security and Privacy*, pp.130–140 (1997).
6) Ferraiolo, D. and Kuhn, R.: Role-Based Access Controls, *Proc. 15th NIST-NCSC Nat'l Computer Security Conf.*, pp.554–563 (1992).
7) Gosling, J. and McGilton, H.: *The Java Language Environment*, Sun Microsystems, Inc. (1996).
8) Harrison, M.A., Ruzzo, W.L. and Ullman, J.D.: Protection in Operating Systems, *Comm. ACM*, Vol.19, No.8, pp.461–471 (1976).
9) Izaki, K., Tanaka, K. and Takizawa, M.: Authorization Model in Object-Oriented Systems, *Proc. IFIP Database Security* (2000).
10) Izaki, K., Tanaka, K. and Takizawa, M.: Information Flow Control in Role-Based Model for Distributed Objects, *Proc. IEEE Int'l Conf. on Parallel and Distributed Systems* (2001).
11) Jason, H., William, C. and Ferguson, P.: *Java Servlet Programming*, O'Reilly and Associates (1998).
12) Lampson, B.W.: Protection, *Proc. 5th Princeton Symp. on Information Sciences and Systems*, pp.437–443 (1971). (also in *ACM Operating Systems Review*, Vol.8, No.1, pp.18–24 (1974))
13) Lampson, B.W.: A Note on the Confinement Problem, *Comm. ACM*, Vol.16, No.10, pp.613–615 (1973).
14) Object Management Group Inc.: *The Common Object Request Broker: Architecture and Specification*, Rev. 2.1 (1997).
15) Oracle Corporation: *Oracle8i Concepts*, Vol.1, Release 8.1.5 (1999).
16) 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, pp.524–538 (1997).
17) Sandhu, R.S.: Lattice-Based Access Control Models, *IEEE Computer*, Vol.26, No.11, pp.9–19 (1993).
18) Sandhu, R.S., Coyne, E.J., Feinstein, H.L. and Youman, C.E.: Role-Based Access Control Models, *IEEE Computer*, Vol.29, No.2, pp.38–47 (1996).
19) Tachikawa, T., Yasuda, M. and Takizawa, M.: A Purpose-oriented Access Control Model in Object-based Systems, *Trans. IPSJ*, Vol.38, No.11, pp.2362–2369 (1997).
20) Tari, Z. and Chan, S.W.: A Role-Based Access Control for Intranet Security, *IEEE Internet Computing*, Vol.1, No.5, pp.24–34 (1997).
21) Yasuda, M., Higaki, H. and Takizawa, M.: A Purpose-Oriented Access Control Model for Information Flow Management, *Proc. 14th IFIP Int'l Information Security Conf.* (*SEC '98*), pp.230–239 (1998).

(Received June 7, 2001)
(Accepted December 18, 2001)

Vlad Ingar Wietrzyk obtained his M.Sc. degree from Prague University. EU and his Dip. in Computer Science from UTS, Sydney. Since 1999 he has been at the University of Western Sydney. Since 1997 until 1998 he had been a visiting researcher of Stuttgart and Mannheim Universities. He has publications in national and international conferences and workshops. In 1999 he was a visiting researcher at the Institute of Software Engineering, Montreal University. He has served on the program committees of international conferences like ICPADS, IDEAS, CIT, COMAD, ENTER. He has deliverd industrial seminars on computing to companies like VERSANT and ALCATEL. While at the Analytical Service Corporation, Sydney, 1987∼1995 he designed and implemented in software, a hierarchical clustering method which was the first to support the analysis of data based on groups and data exploration. His current research interests are: object distributed databases, various aspects of information systems design methodologies (including distributed systems), transaction processing in distributed systems, concurrency control, distributed and federated database systems, and distributed workflow technology supporting electronic commerce. He is a member of IEEE and AIEA.

Keiji Izaki was born in 1978. He received his B.E. degree in computers and systems engineering from Tokyo Denki Univ., Japan in 2000. He is now a graduate student of the master course in the Dept. of Computers and Systems Engineering, Tokyo Denki Univ. His research interests include distributed database systems and security. He is a student member of IPSJ.

Katsuya Tanaka was born in 1971. He received his B.E. and M.E. degrees in Computers and Systems Engineering from Tokyo Denki University, Japan in 1995 and 1997, respectively. From 1997 to 1999, he worked for NTT Data Corporation. Currently, he is an assistant in the Department of Computers and Systems Engineering, Tokyo Denki University. He received the D.E. degree from Dept. of Computers and Systems Engineering, Tokyo Denki University, Japan, in 2000. His research interests include distributed systems, transaction management, recovery protocols, and computer network protocols. He is a member of IEEE CS and IPSJ.

Makoto Takizawa was born in 1950. He received his B.E. and M.E. degrees in Applied Physics from Tohoku Univ., Japan, in 1973 and 1975, respectively. He received his D.E. in Computer Science from Tohoku Univ. in 1983. From 1975 to 1986, he worked for Japan Information Processing Developing Center (JIPDEC) supported by the MITI. He is currently a Professor of the Dept. of Computers and Systems Engineering, Tokyo Denki Univ. since 1986. From 1989 to 1990, he was a visiting professor of the GMD-IPSI, Germany. He is also a regular visiting professor of Keele Univ., England since 1990. He was a program co-chair of IEEE ICDCS-18, 1998 and serves on the program committees of many international conferences. He chaired SIGDPS of IPSJ from 1997 to 1999. He is IPSJ fellow. His research interests include communication protocols, group communication, distributed database systems, transaction management, and security. He is a member of IEEE, ACM, and IPSJ.