マルチメディア通信と分散処理　103－16
（２００１．６．８）

Information Flow Control in Role-Based Access Control Model

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

Various kinds of applications are designed and implemented in an object-based model. Object-based systems are required to be secure in order to realize the applications. 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 to occur among object in a role-based model. We define safe roles where no illegal information flow occurs. In addition, we discuss how to safely perform transactions belonging to unsafe roles. We discuss an algorithm to check if illegal information flow occurs.

役割に基づくアクセス制御におけるオブジェクト間の情報流制御

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

様々な応用がオブジェクトベースモデルで設計、実装されるとともに、応用の安全性を保つ必要がある。安全なシステムとは、不正な操作だけでなく、不正な情報流をも防ぐシステムである。本論文では、オブジェクト間の不正な情報流を役割に基づくアクセス制御モデルを用い制御する方法について定義する。また、安全な役割について定義し、安全でない役割に属するトランザクションを安全に実行する方法について定義する。

1 Introduction

Various kinds of object-based systems like object-oriented database systems, JAVA, and Common Object Request Broker Architecture (CORBA) [8] widely used to design and implement applications. Object-based systems are composed of multiple objects which cooperate to achieve some objectives by passing messages. An object is an encapsulation of data and methods for manipulating the data. Methods are invoked through the message passing mechanism. In addition, methods invoked may invoke other methods in a nested manner. In addition to realizing the interoperability of applications, secure systems are required to not only protect objects from illegally manipulated but also prevent illegal information flow among objects in the system.

In the basic access control model, an access rule is specified in a form $\langle s, o, t \rangle$ which means that a subject s is allowed to manipulate an object o in an access type t. Only access requests which satisfy the authorized access rules are accepted to be performed. However, the access control model implies the *confinement* problem [7], i.e. illegal information flow occurs among subjects and objects. In the *mandatory lattice-based* access control model [1, 3, 10], objects and subjects are classified into security classes. Legal information flow is defined in terms of the *can-flow* relation [3] between classes. In the mandatory model, 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, 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 [4, 11, 14], a *role* is defined to be a collection of access rights, i.e. pairs of access types and objects, which shows a job function in the enterprise. Subjects are granted roles which show jobs assigned to the subjects. An object-based system is a restricted version of an object-oriented system where inheritance hierarchy is not supported. The purpose-oriented model [13, 15]

discusses which methods may invoke another method in the object-based system. Since methods are invoked in the nested manner in the object-based systems, we have to discuss information flow to occur in nested invocations of methods. We define a set of $safe$ roles where no possible illegal information flow occurs in presence of nested invocations. That is, no illegal information flow occurs as long as every transaction is in a session with a safe role. In addition, we discuss an algorithm to check for a transaction which is in a session with an unsafe rule if illegal information flow possibly occurs each time a method is issued to an object. 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.

In section 2, methods supported by objects are classified from information flow point of view. In section 3, we discuss information flow to occur in a nested invocation. In section 4, we discuss information flow to occur in performing transactions with roles. In section 5, we discuss how to resolve illegal information flow among objects for unsafe roles.

2 Object-based Systems

2.1 Role-based access control model

An object-based system is composed of classes and objects. A class is composed of attributes and methods. Objects are instances of the class, which are created by giving values to the attributes of the class. The objects are only manipulated through the methods of the class. A transaction invokes a method of an object by sending a request message to the object. On receipt of the message, the method specified in the message is performed on the object. On completion of the method, the response is sent back to the sender of the message. During the computation of the method, other methods might be invoked. Thus, methods are invoked in a nested manner.

In access control models, there are *subject* and *object*. A *subject* shows a user or an application program. The subject manipulates objects by invoking their methods. On the other hand, an *object* is a passive entity. The relation of subjects and objects are relative in the object-based system.

Each subject plays a *role* in an organization. A role represents a job function in the organization. In the role-based model [4, 11, 14], a *role* is modeled in a set of *access rights*. An access right is specified as a pair $\langle o, t\rangle$ of an object o and a method t meaning that t can be performed on the object o. Let R be a set of roles in the system. In the role-based model, a subject s is granted a role which shows its job function. This means that the subject s can perform a method t on an object o if $\langle o, t\rangle \in r$. A subject s establishes a *session* with r. Then, s can issues methods in r. Each subject can be in a session with at most one role.

2.2 Classification of methods

Each method t on an object o is characterized by the following parameters [Figure 1]:

1. *Input type* (I): If the method t has input data in the parameter, the input type of t is I, else N.
2. *Manipulation type* (M): If the state of the object o is changed by t, the manipulation type of t is M, else N.
3. *Derivation type* (D): If data is derived from the state of the object o by t, the derivation type of t is D, else N.
4. *Output type* (O): If data is returned to the invoker of t, the output type of t is O, else N.

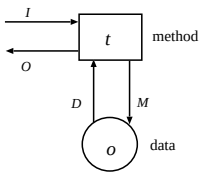

Figure 1: Information flow on an object.

Each method t of an object o is characterized by a *method type* $mtype(t)$ in terms of the input α_1, manipulation α_2, derivation α_3, and output α_4 types, i.e. $mtype(t) = \alpha_1\alpha_2\alpha_3\alpha_4$ where $\alpha_1 \in \{I, N\}$, $\alpha_2 \in \{M, N\}$, $\alpha_3 \in \{D, N\}$, and $\alpha_4 \in \{O, N\}$. 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, e.g. the data is stored in the object. Here, if some type α_i in the specification of the method type is N, N is omitted in the method type. For example, "IM" shows $IMNN$. Especially, "N" shows a type $NNNN$. There are sixteen method types from information flow point of view as shown in Figure 2. Let MC be a set $\{IMDO, IDO, IMO, IO, IMD, ID, IM, I, MDO, DO, MO, O, MD, D, M, N\}$ of possible method types.

Suppose a subject s is in a session with a role r. The subject s manipulates objects through methods according to the access rights in the roles. We assume that each subject does not have any persistent storage. That is, the subject does not keep in record data obtained from the object by manipulating the object. The subject issues one or more than one method to objects to do some work. A sequence of methods issued by the subject is referred to as a *transaction*, which is a unit of work. Each *transaction* can be in a session

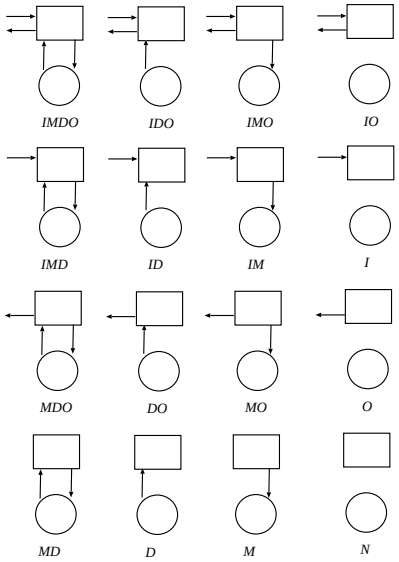

Figure 2: Method types.

with one role. A transaction has a temporary memory. Data which the transaction derives from objects may be stored in the memory of the transaction. On completion of the transaction, the transaction memory is released. Any transaction initiated for a subject does not share data with the other transactions.

In the object-based system, methods are invoked in the nested manner. Suppose a transaction T is in a session with a role r and T invokes a method t_1 on an object o_1. If $\langle o_1, t_1 \rangle \in r$, T is allowed to manipulate o_1 through t_1. Here, suppose t_1 invokes another method t_2 on an object o_2. If $\langle o_2, t_2 \rangle \in r$, there is no problem t_2 manipulates o_2 through t_2. Let us consider case $\langle o_2, t_2 \rangle \notin r$. In one approach, t_1 is not allowed to invoke t_2. That is, every method invoked in the transaction T is considered to be invoked by T itself. The method t_2 can be invoked in T only if an access right $\langle o_2, t_2 \rangle$ is in the role r which is in a session with T. This approach is named *impersonation*. In the other approach, if the owner of the object o_1 is granted an access right $\langle o_2, t_2 \rangle$, t_1 is allowed to invoke t_2 even if T is not granted $\langle o_2, t_2 \rangle$. This approach is taken in the relational database system. In the role-based access control model, t_1 is allowed to invoke t_2 if the owner of o_1 is in a session with some role r' including the access right $\langle o_2, t_2 \rangle$. This one is named *ownership chain* approach.

3 Nested Invocation

3.1 Invocation tree

A transaction invokes methods and then some of the methods invoke other methods. For example, a transaction T invokes a method t_1 on an object o_1 and another 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 in the transaction T are represented in a tree form as shown in Figure 3. The tree is named *invocation tree* $Tree(T)$ of T. In Figure 3, each node shows a method t invoked on an object o, i.e. $\langle o, t \rangle$, in the transaction T. A dotted directed edge from a parent to a child shows that the parent invokes the child. $\langle o_1, t_1 \rangle \stackrel{T}{\Vdash} \langle o_2, t_2 \rangle$ means that a method t_1 on an object o_1 invokes t_2 on 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 children, the left-to-right order of children shows an invocation sequence of methods. i.e. tree is ordered. Suppose $\langle o_1, t_1 \rangle \stackrel{T}{\Vdash} \langle o_2, t_2 \rangle$ and $\langle o_1, t_1 \rangle \stackrel{T}{\Vdash} \langle o_3, t_3 \rangle$ is 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 T. In addition, $\langle o_4, t_4 \rangle \prec_T \langle o_3, t_3 \rangle$ if $\langle o_2, t_2 \rangle \stackrel{T}{\Vdash} \langle o_4, t_4 \rangle$. $\langle o_2, t_2 \rangle \prec_T \langle o_4, t_4 \rangle$ if $\langle o_3, t_3 \rangle \stackrel{T}{\Vdash} \langle o_4, t_4 \rangle$. $\prec_T$ is transitive. For example, T invokes t_1 before t_2 as shown in Figure 3. $\langle o_1, t_1 \rangle \prec_T \langle o_2, t_2 \rangle$ and $\langle o_3, t_3 \rangle \prec_T \langle o_2, t_2 \rangle$.

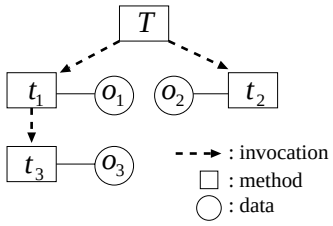

Figure 3: Invocation tree.

3.2 Information flow

In Figure 3, suppose $mtype(t_3) = DO$, $mtype(t_2) = IM$, and $mtype(t_1) = O$. 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 the method t_2 as the input parameter. The data is stored into the object o_2 through t_2. Thus, the information in o_3 is carried to o_2. A dotted arc shows an invocation and straight arc indicates information flow in Figure 4. Here, data derived from o_3 may flow into o_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 ($\langle o_1, t_1 \rangle \stackrel{T}{\rightarrow} \langle o_2, t_2 \rangle$) iff t_1 invokes t_2 ($\langle o_1, t_1 \rangle \stackrel{T}{\Vdash}$

$\langle o_2, t_2 \rangle$) and $I \in mtype(t_2)$, or $\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_3, t_3 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$ 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 ($\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$) iff $\langle o_2, t_2 \rangle \overset{T}{\vdash} \langle o_1, t_1 \rangle$ and $O \in mtype(t_2)$, or $\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_3, t_3 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$ 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 a transaction T ($\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$) iff $\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$, $\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$, $\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_3, t_3 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$ and $\langle o_1, t_1 \rangle \prec_T \langle o_2, t_2 \rangle$, or $\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_3, t_3 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$ for some $\langle o_3, t_3 \rangle$ in T. □

Suppose the invocation tree of T shown in Figure 4 is ordered, i.e. $\langle o_1, t_1 \rangle \prec_T \langle o_2, t_2 \rangle$ and $\langle o_3, t_3 \rangle \prec_T \langle o_2, t_2 \rangle$. Here, $\langle o_3, t_3 \rangle \xrightarrow{T} \langle o_1, t_1 \rangle \xrightarrow{T} \langle \_, T \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$.

[Definition] $\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$ iff $\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$, $\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$, or $\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_3, t_3 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$ for some $\langle o_3, t_3 \rangle$ in T. □

A relation "$o_1 \xrightarrow{T} o_2$" shows "$\langle o_1, t_1 \rangle \xrightarrow{T} \langle o_2, t_2 \rangle$" for some methods t_1 and t_2. The relation "$o_1 \xrightarrow{T} o_2$" means that some data in an object o_1 flows to another object o_2 in a transaction T.

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

In Figure 4, $\langle o_3, t_3 \rangle \overset{T}{\Rightarrow} \langle o_2, t_2 \rangle$ where $\langle o_3, t_3 \rangle$ is a *source* and $\langle o_2, t_2 \rangle$ is a *sink*. Here, data in the object o_3 flows into the object o_2. "$\langle o_1, t_1 \rangle \overset{T}{\Rightarrow} \langle o_2, t_2 \rangle$" can be written as "$o_1 \overset{T}{\Rightarrow} o_2$".

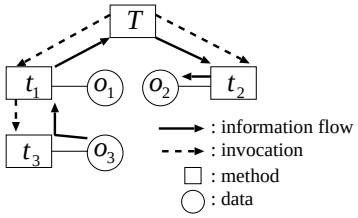

Figure 4: Information flow.

Let $Tr(r)$ be a set of transactions each of which is in a session with a role r. $o_1 \overset{r_9}{\Rightarrow} o_2$ shows $o_1 \overset{T}{\Rightarrow} o_2$ for some transaction T in $Tr(r)$.

[Definition] An role r_1 *threatens* another role r_2 iff for every pair of objects o_i and o_j,

- $o_i \overset{r_9}{\Rightarrow} o_j$, and
- $o_i \overset{r_4}{\not\Rightarrow} o_j$ and $o_j \overset{r_4}{\Rightarrow} o$ for some object o. □

By performing a transactions in a session with r_2, information in o_i might flow to another object o_j. If another transaction is not granted an access right to derive data from o_i while granted an access right to derive data from o_j, the transaction can get data in o_i even if the transaction is not allowed to do it. A role of the transaction is threatens r_2. If there is another role threatens a role r, illegal information flow might occur. We define a safe information flow.

[Definition] An information flow relation "$o_i \overset{r}{\Rightarrow} o_j$" is $safe$ for a role r iff every other role neither threatens r nor is threatened by r □.

[Definition] $o_i \Rightarrow o_j$ is *illegal* iff $o_i \Rightarrow o_k \overset{r}{\Rightarrow} o_j$ but $o_i \overset{r}{\not\Rightarrow} o_j$ for some object o_k and some role r. □

[Definition] A role r is $safe$ iff $o_i \overset{r}{\Rightarrow} o_j$ is safe for every pair of access rights $\langle o_i, t_i \rangle$ and $\langle o_j, t_j \rangle$ in the role r. □

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. □

The paper [6] discusses an algorithm for checking if each method issued by an unsafe transaction might imply illegal information flow.

4 Information Flow for Roles

4.1 Confinement problem on roles

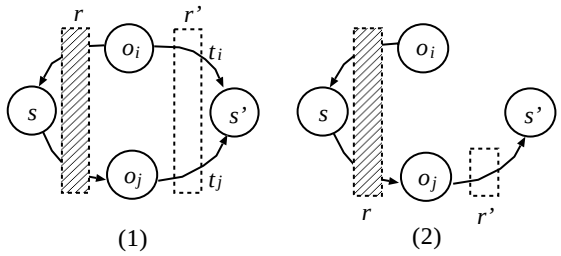

Figure 5: Roles.

Suppose there is a role r including access rights $\langle o_i, u_i \rangle$ and $\langle o_j, u_j \rangle$ for a pair of objects o_i and o_j. If $DO \in mtype(u_i)$ and $IM \in mtype(u_j)$, a subject in a session with r can derive data from o_i through u_i and then can write the data into o_j through u_j. Here, information in o_i might be brought into o_j by performing methods in r, i.e. $o_i \overset{r}{\Rightarrow} o_j$.

Suppose there is only a role r' in addition to the role r in a system. First, suppose the role r' includes $\langle o_i, t_i \rangle$ and $\langle o_j, t_j \rangle$ where $DO \in mtype(t_i)$ and $DO \in mtype(t_j)$. Some subject s' in a session with r' can get data from the objects o_i and o_j through DO methods t_i and t_j, respectively, as shown in Figure 5 (1). Even if the subject s brings data from o_i to o_j, the subject s' can obtain the data from o_j because s' is allowed to obtain data from o_i. Secondly, suppose $r = \{ \langle o_j, t_j \rangle \}$ and $DO \in mtype(t_j)$ In Figure 5 (2), a role r' includes only $\langle o_j, t_j \rangle$, not $\langle o_i, t_i \rangle$ since s' cannot access to o_j.

There is no illegal information flow from o_i. Here, another subject s' can get information which s has derived from the object o_i by manipulating the object o_j although the subject s' is not allowed to manipulate o_j. This is also a kind of the *confinement* problem [7], i.e. illegal information flow might occur.

4.2 Models

Suppose a transaction T is in a session with a role r. Here, we assume an access right $\langle o, t\rangle$ is in r if a method t is invoked on an object o by T. It is not easy to make clear what transaction there are for each role and how each transaction invokes methods. First, we discuss a basic (b) simple model where a transaction in a session with a role r may invoke all the methods in r. A virtual object *transaction* Tr is assumed to invoke a method t on an object o ($\langle \_, Tr\rangle \vdash \langle o, t\rangle$) for every $\langle o, t\rangle$ in the role r. An invocation tree of Tr is an an ordered, two-level tree. For example, $\langle \_, Tr\rangle \xrightarrow{r} \langle o, t\rangle$ if $\langle o, t\rangle \in r$ and $I \in mtype(t)$. $\langle o, t\rangle \xrightarrow{r} \langle \_, T\rangle$ if $\langle o, t\rangle \in r$ and $o \in mtype(t)$. $\xrightarrow{r}$is transitive. $\langle o, t\rangle \overset{r}{\Rightarrow} \langle \_, T\rangle$ iff $\langle o, t\rangle \xrightarrow{r} \langle \_, Tr\rangle$ and $D \in mtype(t)$. $\langle \_, Tr\rangle \overset{r}{\Rightarrow} \langle o, t\rangle$ iff $\langle \_, Tr\rangle \xrightarrow{r} \langle o, t\rangle$ and $M \in mtype(t)$. $\langle o_1, t_1\rangle \overset{r}{\Rightarrow} \langle o_2, t_2\rangle$ iff $\langle o_1, t_1\rangle \xrightarrow{r} \langle \_, Tr\rangle$ and $\langle \_, Tr\rangle \xrightarrow{r} \langle o_2, t_2\rangle$. Here, $o_1 \overset{r}{\Rightarrow} o_2$ shows "$\langle o_1, t_1\rangle \overset{r}{\Rightarrow} \langle o_2, t_2\rangle$" for some methods t_1 and t_2. $\overset{r}{\Rightarrow}$ is referred to as *inter-role information flow* in r. $Tr \overset{r}{\Rightarrow} o$ and $o \overset{r}{\Rightarrow} Tr$ show "$\langle \_, Tr\rangle \overset{r}{\Rightarrow} \langle o, t\rangle$" and "$\langle o, t\rangle \overset{r}{\Rightarrow} \langle \_, Tr\rangle$" for some method t, respectively.

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 of method t invoked on o in a transaction T. Let $Al(r)$ be $\{\langle o, t\rangle \mid \langle o, t\rangle \in N(T)$ for every transaction T in $Tr(r)\}$. That is, $Al(r)$ gives a collection of methods invoked in transactions which are to be in a session with r.

Suppose there are two transactions T_1 and T_2 which 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 $N(T_2) = \{\langle o_2, t_2\rangle, \langle o_3, t_3\rangle, \langle o_4, t_4\rangle\}$. 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).

A forest of invocation trees, each of which has one of the transactions as the root, can be constructed as shown in Figure 6. In the basic model $Tr(r)$ invokes t_1 and t_2. Here, there is no information flow between o_1 and o_3, because o_1 is manipulated by T_1 and o_3 is manipulated by T_2. If transaction are not ordered, $o_4 \overset{r}{\Rightarrow} o_3$ as shows in Figure 6. On the other hand, if transactions are ordered, o_4 is manipulated before o_3. Hence, $o_4 \overset{T_2}{\not\Rightarrow} o_3$ Thus, possible illegal information flow can be reduced.

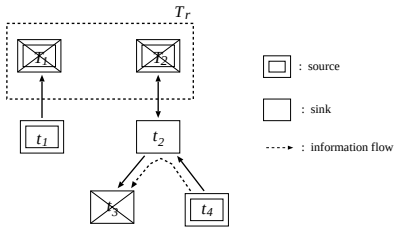

Figure 6: Invocation trees.

For each model of basic (b), unordered (u), and ordered (o) models, information flow relations are defined for a role r as follows:

$o_i \overset{r}{\underset{b}{\Rightarrow}} o_j$ iff $o_i \overset{r}{\Rightarrow} o_j$ in the b model.
$o_i \overset{r}{\underset{u}{\Rightarrow}} o_j$ iff $o_i \overset{r}{\Rightarrow} o_j$ in the u model.
$o_i \overset{r}{\underset{o}{\Rightarrow}} o_j$ iff $o_i \overset{r}{\Rightarrow} o_j$ in the o model.

Flow relations $\overset{}{\underset{b}{\Rightarrow}}$, $\overset{}{\underset{u}{\Rightarrow}}$ and $\overset{}{\underset{o}{\Rightarrow}}$are defined in a similar way.
[Theorem] $o_i \overset{r}{\underset{o}{\Rightarrow}} o_j$ if $o_i \overset{r}{\underset{u}{\Rightarrow}} o_j$. $o_i \overset{r}{\underset{u}{\Rightarrow}} o_j$ if $o_i \overset{r}{\underset{b}{\Rightarrow}} o_j$. □

5 Resolution of Illegal Information Flow

Every safe transaction is allowed to be performed because no illegal information flow occurs. We discuss how to perform even unsafe transactions Suppose a transaction T is in a session with a role r. A method t on an object o is invoked in T. There are two cases:

1. Another method t_1 is invoked on an object o_1 before t in T and $o_1 \overset{r}{\Rightarrow} o$. Here, $IM \in mtype(t)$.
2. Data in o is derived through t, i.e. $DO \in mtype(t)$.

Suppose a system maintains a following directed *flow graph* G is constructed.
[Flow graph]

1. Each node shows an object in the system.
2. A directed edge $o_1 \xrightarrow{\tau} o_2$ is created if $o_i \overset{r}{\Rightarrow} o_j$ at time τ by a transaction T.
3. $o_1 \xrightarrow{\tau} o_2$ is created if $o_1 \xrightarrow{\tau} o_3 \xrightarrow{\tau'} o_2$, $\tau < \tau'$, and no directed edge from o_1 to o_2. □

$In(o) = \{o_1 \mid o_1 \xrightarrow{\tau} o$ for some τ in $G\}$. If the following condition is satisfied, the method t is allowed to be invoked in the object o.

1. For every object o_2 in $In(o_1)$, $\langle o_2, t_2\rangle \in r$ and $DO \in mtype(t_2)$ for case 1 [Figure 7 (1)].
2. For every object o_2 in $In(o)$, $\langle o_2, t_2\rangle \in r$ and $DO \in mtype(t_2)$ for case 2 [Figure 7 (2)].

In the condition 1, data in some object o_2 might have been brought into o_1 before T manipulates o. If t is invoked in T, data in o_1 is carried to o, that is,

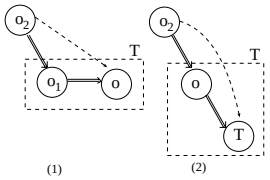

Figure 7: Conditions.

data in o_2 might have been brought to o. Here, if the role r has an access right to derive data from o_2, t is allowed to be performed. Otherwise, t is not allowed.

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 ($o_i \xrightarrow{\tau} o_j$ in G). Security level of the data is changing time by time. After it takes some time, the data flowing from o_i to o_j is really to be manipulated according to roles other then r. In the flow graph G, edges are removed as follows:

1. Each edge $o_i \xrightarrow{\tau} o_j$ is removed from the flow graph G if $\tau + \delta < \sigma$. Here, σ shows the current time.

6 Concluding Remarks

This paper presented an access control model for the object-based system with role concepts. We discussed how to control information flow in a system where method are invoked in a nested manner. We first defined a set of safe roles 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. 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 which order a transaction performs the methods even if the methods are not allowed to be performed due to the unsafeness of the roles. We also discussed a case that security level is *time-variable*

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," *Communications of the ACM,* Vol. 19, No. 5, 1976, pp. 236–243.

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

[5] Izaki, K., Tanaka, K., and Takizawa, M., "Authorization Model in Object-Oriented Systems," *Proc. of IFIP Database Security,* 2000.

[6] Izaki, K., Tanaka, K., and Takizawa, M., "Information Flow Control in Role-Based Model for Distributed Objects," to appear in *Proc. of IEEE Int'l Conf. on Parallel and Distributed Systems* (*ICPADS-2001*), 2001.

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

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

[9] 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, 1997, pp. 524–538.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.* (*SEC'98*), 1998, pp. 230–239.