Role-based Informartion Flow Control in Object-based Systems *

1H−02 Keiji Izaki, Katsuya Tanaka, and Makoto Takizawa †
Tokyo Denki University ‡
Email : {izaki, katsu, taki}@takilab.k.dendai.ac.jp

1 Introduction

Various kinds of object-based systems like CORBA are widely used to realize distributed applications by using JAVA. Object-based systems are composed of multiple objects which cooperate to achieve some objectives. An object is an encapsulation of data and methods for manipulating the data. Methods are invoked through the message passing mechanism. Here, methods are invoked in a nested manner.

Kinds of access control models are developed and widely used to protect resources in systems from illegally manipulated. 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 through a method t. Here, a pair $\langle o, t\rangle$ is an *access right* granted to the subject s. Only access requests which satisfy the authorized access rules are accepted to be performed. However, the *confinement* problem [1] is implied, i.e. illegal information flow occurs among subjects and objects. In the role-based model, a *role* is defined to be a collection of access rights to denote a job function in an enterprise. A subject s is granted a role r which shows its job function in the organization. This means that the subject s can perform a method t on an object o if an access right $\langle o, t\rangle$ is included in the role r.

In this paper, we discuss information flow to occur in case methods are invoked in a nested manner. In addition, access control is role-based. We define a *safe* role where no illegal information flow occurs by performing any transaction with the role. In addition, we discuss an algorithm to interpretively check for each method issued by a transaction if illegal information flow occurs by performing the method.

In section 2, we present a system model and 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 present a flow graph to show information flow. In section 5, we discuss an interactive algorithm to check if a method could be performed to prevent illegal information flow.

2 Classification of Methods

Objects support various abstract types of methods which are procedures for manipulating the states. Each method t on an object o is characterized by following parameters:

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

3 Nested Invocation

3.1 Invocation tree

Methods are invoked in a nested manner. For example, 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. We assume at most one method is invoked at a time in every transaction and method. The invocations of methods in the transaction T are represented in a tree form named *invocation tree*. Each node $\langle o, t\rangle$ shows a method t invoked on an object o in the transaction T. 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 o_2 in the transaction T. A node $\langle -, T\rangle$ shows a root of invocation tree of T.

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 T invokes t_1 before t_2. Here, a node $\langle o_1, t_1\rangle$ precedes another node $\langle o_2, t_2\rangle$ in T ($\langle o_1, t_1\rangle \prec_T \langle o_2, t_2\rangle$). Since t_2 is invoked after t_3, $\langle o_3, t_3\rangle \prec_T \langle o_2, t_2\rangle$. $\prec_T$ is transitive.

4 Flow Graph

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 which shows information flow.

[Flow graph]

1. Each node shows an object in the system. Here, each transaction is also an object. Initially, the flow graph G includes no edge. If an object is created, a node for the object is added in G.
2. A labeled directed edge $o_1 \xrightarrow{\omega} o_2$ from a node o_1 to a node o_2 is created if information flow from an object o_1 to another object o_2 occurs ($o_1 \overset{r}{\Rightarrow} o_2$) by performing a transaction T of a role r at time ω. If a directed edge "$o_1 \xrightarrow{\omega_1} o_2$" already exists in the flow graph G and $\omega_1 < \omega$, the label ω_1 of the edge $o_1 \xrightarrow{\omega_1} o_2$ is changed to ω ($o_1 \xrightarrow{\omega} o_2$).
3. For each object o_3 such that $o_3 \xrightarrow{\omega_1} o_1$ in G,

*オブジェクトベースシステムにおける役割に基づく情報流制御
†井崎 慶之, 田中 勝也, 滝沢 誠
‡東京電機大学

3.1 a directed edge "$o_3 \overset{\omega}{\rightarrow} o_2$" is created if there is no edge from o_3 to o_2 in G and $\omega_1 < \omega$. Step 2 is performed,

3.2 if a directed edge $o_3 \overset{\omega_2}{\rightarrow} o_2$ already exists in G and $\omega_2 > \omega$, the edge "$o_3 \overset{\omega_2}{\rightarrow} o_2$" is changed with "$o_3 \overset{\omega}{\rightarrow} o_2$". □

Figure 1 shows a flow graph G including four objects o_1, o_2, o_3, and o_4. First, suppose $o_1 \overset{4}{\rightarrow} o_2$ and $o_2 \overset{3}{\rightarrow} o_4$ in G. Then, an information flow relation "$o_2 \overset{r_1}{\Rightarrow} o_3$" holds by performing a transaction with a role r_1 at time 6. Here, a directed edge $o_2 \overset{6}{\rightarrow} o_3$ is created in G. Since $o_1 \overset{4}{\rightarrow} o_2 \overset{6}{\rightarrow} o_3$, information flowing from the object o_1 to the object o_2 at time 4 might flow to o_3 by the transaction. Hence, $o_1 \overset{6}{\rightarrow} o_3$ since $4 < 6$ [Figure 1 (2)]. Then, $o_3 \overset{r_2}{\Rightarrow} o_4$ at time 8. $o_3 \overset{8}{\rightarrow} o_4$. Since $o_1 \overset{4}{\rightarrow} o_2 \overset{6}{\rightarrow} o_3 \overset{8}{\rightarrow} o_4$, an edge $o_1 \overset{8}{\rightarrow} o_4$ is also created and another edge $o_2 \overset{8}{\rightarrow} o_4$ is tried to be created. However, $o_2 \overset{3}{\rightarrow} o_4$ exists already in G. Since $3 < 8$, the edge $o_2 \overset{3}{\rightarrow} o_4$ is replaced with $o_2 \overset{8}{\rightarrow} o_4$ [Figure 1 (3)]. In Figure 1 (3), information in the objects o_1, o_2, and o_3 flow into o_4. Let $In(o)$ be a input set $\{o_1 \mid o_1 \overset{\omega}{\rightarrow} o$ in $G\}$ of objects whose information has flown into an object o. For example, $In(o_4) = \{o_1, o_2, o_3\}$ in Figure 1(3). In the Figure 1(1), $o_2 \overset{2}{\Rightarrow} o_3$ occurs at time 2. Here, sine $4 > 2$, just an edge $o_2 \overset{2}{\rightarrow} o_3$ is added.

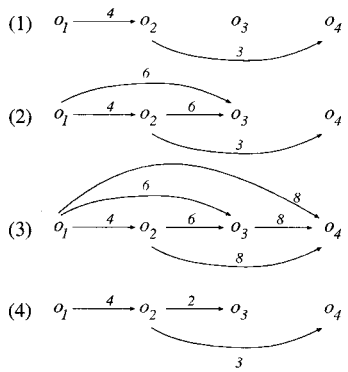

Figure 1: Flow graph G.

5 Checking Algorithm

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*] [Figure 2]

1. for every "$o_2 \overset{\omega}{\rightarrow} o_1$" 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 "$o_2 \overset{\omega}{\rightarrow} o$" in G if $DO \in mtype(t)$. □

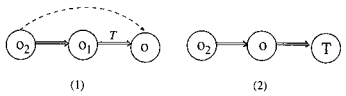

Figure 2: Condition.

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 $DO \in mtype(t)$.

6 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 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.

References

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