Information Flow Control in Object-based Systems *

3Aa－4

Masashi Yasuda, Takayuki Tachikawa, and Makoto Takizawa †

Tokyo Denki University ‡

Email {masa, tachi, taki}@takilab.k.dendai.ac.jp

1 Introduction

Distributed applications are modeled in an object-based model like CORBA [1]. Here, the system is a collection of objects. Each object is an encapsulation of more abstract data structure and operations than *read* and *write*. The objects are manipulated only through operations supported by themselves. The access rules are defined based on the operation types. It is essential to discuss the *purpose* of s to access o by t. The *purpose-oriented* model [2] is proposed where an access rule shows for what each subject s manipulates an object o by an operation t of o so as to keep the information flow legal. The *purpose* of s to access o by t is modeled to be what operation u of s invokes t to manipulate o. That is, the purpose-oriented access rule is specified in a form $\langle s : u,\ o : t\rangle$. In the object-based system, on receipt of a request op from an object o_1, an object o_2 computes op and then sends back the response of op to o_1. Here, if the request and the response carry data, the data in o_1 and o_2 is exchanged among o_1 and o_2. Furthermore, the operations are nested in the object-based system. 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 of operations. In this paper, we discuss what the information flow is legal in the nested invocations in the purpose-oriented model of the object-based system.

2 Purpose-Oriented Model

First, we define secure objects.

[Definition] An object o_i is *secure* iff

(1) o_i can be only accessed through the operations supported by o_i,

(2) no operation of o_i malfunctions, and

(3) a pair of operations op_1 and op_2 can exchange data only through the state of o_i. □

If data d flowing from an object o_i to another 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. We assume that every object is secure.

In the access control model, an access rule $\langle s, o_i, op_i\rangle$ means that a subject s manipulates an object o_i through an operation op_i. In order to make the system secure, it is important to consider a *purpose* for which s manipulates o_i by t_i in addition to discussing whether s can manipulate o_i by t_i. Suppose o_i manipulates o_{ij} by invoking an operation op_{ij} of o_{ij}. Here, the *purpose* of o_i for manipulating o_{ij} is modeled to show which *operation* in o_i invokes op_{ij} of o_{ij}. Hence, the access rule is written in a form $\langle o_i : op_i,\ o_{ij} : op_{ij}\rangle$ in the purpose-oriented model. op_i shows the *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.

[Purpose-oriented (PO) rule] The access rule $\langle o_i : op_i,\ o_{ij} : op_{ij}\rangle$ means that o_i can manipulate o_{ij} through an operation op_{ij} invoked by op_i of o_i. □

3 Information Flow

We discuss what purpose-oriented rules are allowed to be specified from the information flow point of view.

3.1 Computation model

Each object o computes an operation op on receipt of a request op. o creates a thread of op named an *instance* of op. op may invoke operations $op_1,\ldots,op_l$ where each op_i is computed on an object o_i. There are *synchronous* and *asynchronous* ways for op to invoke op_i. In the synchronous invocation, op waits for the completion of op_i. In the asynchronous one, op does not wait for the completion of op_i, i.e. op_i is computed independently of op. Furthermore, there are *serial* and *parallel* invocations. In the serial invocation, op serially invokes $op_1,\ldots,op_l$, i.e. op invokes op_i after the completion of op_{i-1}. Hence, the information carried by the response of op_{i-1} may flow to op_i. On the other hand, op invokes $op_1, \ldots, op_l$ in parallel. Each op_i is computed on o_i independently of another op_j. This means that the information carried by the response of op_i does not flow to op_j while flowing to op.

The invocations of $op_1,\ldots,op_l$ by op are represented in an ordered *invocation tree*. In the invocation tree, each branch ($op \rightarrow op_i$) shows that op invokes op_i. In addition, $op_1,\ldots,op_l$ are partially ordered. If op_i is invoked before op_j, op_i precedes op_j ($op_i \rightarrow op_j$). For example, suppose a user serially invokes two operations op_1 and op_2. op_1 invokes op_{12} and op_{13} in parallel after op_{11}. "$\rightarrow$" shows the computation order of the operations. We assume that no operation instance appears multiple times in the tree.

In the object-based system, the operations are invoked in the nested manner. Suppose an object o invokes an operation op_i in o_i. op_i further invokes operations $op_{i1},\ldots,op_{il_i}$ where each op_{ij} is in o_{ij}. op_i in o_i communicates with o and o_{ij} while exchanging data with o.

3.2 Invocation graph

An *invocation graph* is introduced to show the information flow relation among operations. Each node indicates an operation. There are *request* (Q) and *response* (S) edges. If an operation op_i of an object o_i invokes op_j of o_j, there is a Q edge from op_i to op_j denoted by a straight arrow line, i.e. a connection between β_3 of op_i and α_1 of op_j. There are the following points to be discussed on the Q edge ;

(1) whether or not op_i sends data in o_i to op_j, and

(2) whether or not op_j changes the state of o_j.

op_i sends a request message op_j without data to o_j and op_j does not change o_j. There is no information

*分散オブジェクトシステムにおける情報流制御

†安田 昌史, 立川 敬行, 滝沢 誠

‡東京電機大学

flow from o_i to o_j. The second (2) is QON. op_i sends a request op_j with data to o_j but op_j does not change o_j. Although some data is derived from o_i, the data does not flow to o_j. The third (3) is QNI. op_j changes o_j while op_i does not send data to o_j. Some data flows into o_j but the data does not flow out from o_i. The last (4) is named QOI. Here, op_i sends data to o_j and op_j changes o_j. Some data in o_i flows to o_j.

Next, let us consider the response (S) edges which show information flow carried by the responses from o_j to o_i. The S edges are indicated by dotted arrow line. There are the following points to be discussed on the S edges ;

(1) whether or not op_j sends data in o_j to op_i, and
(2) whether or not op_i changes the state of o_i.

The first type (1) is referred to as SNN, where no information flow from o_j to o_i. The second (2) is SNO, where op_j sends o_i the response with data derived from o_j, but op_j does not change o_i. The third (3) is SIN. op_i changes o_i but op_j sends the response without data to o_i. The fourth (4) is SIO. Here, op_j sends back the response with data derived from o_j to o_i and op_i changes o_i. That is, data in o_j flows to o_i.

3.3 Flow graph

The nested invocation is represented in an *invocation* tree as presented in the previous subsection. Here, suppose that an operation op_i invokes op_j in an invocation tree T. There are a Q edge Q_{ij} from the parent op_i to the child op_j and an S edge S_{ij} from op_j to op_i. Thus, each branch between op_i and op_j represents a couple of Q_{ij} and S_{ij} edges between op_i and op_j. Here, let $root$ (T) denote a root of the tree T. In order to analyze the information flow among the operations, a *flow graph* F is obtained from the invocation tree T by the following procedure.

[Construction of flow graph]

(1) Each node in F indicates an operation of T.
(2) For each node op_d connected to the parent by QNI or QOI edge in T, a path P from $root$ (T) to op_d is obtained. For each node op_s in P, there is a directed edge $op_s \rightarrow op_d$ in F if there is a QON or QOI edge from op_s to a child node in P [Figure 1 (1)].
(3) For each node op_p in T, $op_{c_1} \rightarrow op_{c_2}$ if op_{c_1} and op_{c_2} are descendents of op_p in T, which are included in different subtrees of op_p, op_{c_1} has an SNO or SIO edge with the parent of op_{c_1}, and op_{c_2} has a QNI or QOI edge with the parent of op_{c_2} and op_{c_1} precedes op_{c_2} in T [Figure 1 (2)].
(4) $op_1 \rightarrow op_3$ if $op_1 \rightarrow op_2 \rightarrow op_3$ [Figure 1 (3)].

Let us consider a leaf node op_l in the invocation tree T. A leaf node does not invoke other operations. If op_l is invoked with some data and sends back the response, op_l may forward the input data carried by the request to the parent of op_l. Therefore, we have to consider the following additional rules for each leaf node op_l.

(5) For each node op_l connected to the parent by an SNO or SIO edge in T, a path P from $root$ (T) to op_l is obtained. For each node op_d in P, there is a directed edge $op_l \rightarrow op_d$ in F if there is an SIN or SIO edge from a child node to op_d [Figure 1 (4)].
(6) For each leaf node op_l, a path P from $root$ (T) to op_l is obtained. For every node op_s in P, $op_s \dashrightarrow op_l$ if op_s is connected with the child in a QON or QOI edge. For each node op_d in P, there is a directed edge $op_l \dashrightarrow op_d$ in F if op_d is connected to the child in an SIN or SIO edge. For each node op_s in P, there is a directed edge $op_s \rightarrow op_d$ if (1) $op_s \dashrightarrow op_l$ or $op_s \rightarrow op_l$ and (2) $op_l \dashrightarrow op_d$ [Figure 1 (5)].
(7) For each node op_i which is connected to the parent in SNO or SIO edge, a path P from root (T) to op_i is obtained. If op_j in P is connected to the child in QNI or QOI and SIO or SIN edge, $op_i \rightarrow op_j$ [Figure 1 (6)]. □

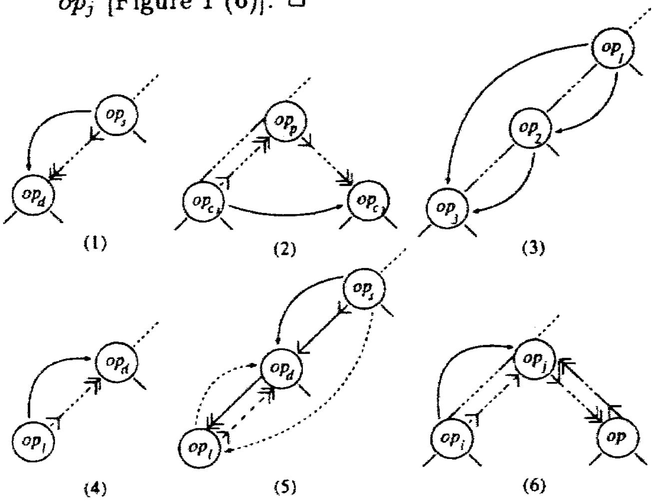

Figure 1: Directed edges.

3.4 Access rules

The flow graph shows the possible information flow to occur if the operations are invoked according to the purpose-oriented rules. Each purpose-oriented access rule $\langle o_i : op_i,\ o_j : op_j \rangle$ is allowed to be specified if the rule satisfies the information flow relation among the objects. The directed edge $\rightarrow$ between op_i and op_j is *legal* in F if the following rule is satisfied.

Even if an access rule $\langle o_j : op_j,\ o_k : op_k \rangle$ is specified, op_i cannot invoke op_j if op_j and op_k are not legally related to the information flow relation. Here, $\langle o_i : op_i,\ o_j : op_j \rangle$ is allowed to be specified if all the directed edges incident to and from op_i and op_j are legal.

4 Concluding Remarks

In the distributed systems, objects support more abstract operations than *read* and *write*. In the *purpose-oriented* access control model [2], it is discussed why an object manipulates other objects while the mandatory model discusses if each subject can access an object by an operation. In addition, the operations of the objects are nested. The access rules have to satisfy the information flow relation among objects. In this paper, we have discussed how to validate the purpose-oriented access rules.

References

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

[2] Tachikawa, T., Yasuda, M., and Takizawa, M., "A Purpose-oriented Access Control Model in Object-based Systems," *Trans. of IPSJ*, Vol. 38 No. 11, pp. 2362–2369, 1997.