Purpose-Oriented Access Control Model in Object-Based Systems

Takayuki Tachikawa, Takayuki Tachikawa, Hiroaki Higaki, and Makoto Takizawa
Dept. of Computers and Systems Engineering
Tokyo Denki University
{tachi, hiro, hig, taki}@takilab.k.dendai.ac.jp

In distributed applications, a group of multiple objects are cooperated to achieve some objectives by sending messages through communication networks. An object is modeled as a pair of data structure and a collection of operations. In the cooperation among multiple objects, it is important to prevent from the illegal information flow among the objects. In this paper, we would like to discuss an object-based access control model to deal with the information flow in the presence of more abstract, application-oriented operations than *read* and *write* operations. We furthermore would like to propose a new purpose-oriented access control model which takes into account the purpose for which subjects access objects by operations of the objects.

目的指向アクセス制御方式

立川 敬行　村井 洋成　桧垣 博章　滝沢 誠
東京電機大学理工学部経営工学科

情報システムは、複数の処理実体が通信網によって相互接続された分散型の形態となっている。各実体は、実体内のデータを操作するための操作演算を提供する。複数の実体のグループ通信が協調動作することにより、分散処理が行なわれる。安全な分散処理を行うためには、実体間での情報流を制御する必要がある。本論文では実体の提供する抽象演算を対象とした、強制アクセス制御方式を論じる。さらに、ある実体を操作する実体が、どのような「目的」で操作を行うかを制御する目的指向アクセス制御方式を論じる。

1 Introduction

Current information systems are composed of clients and servers interconnected by communication networks. The application programs in the clients access the resources in the servers. Units of resources in the system are named *entities*. Entities which support other entities with some services like the database servers are named *objects*. On the other hand, entities which manipulate the objects like application programs in the clients named *subjects*. In the information system, the objects have to be assured to be securely accessed by the subjects. In this paper, we assume that the communication network is secure, e.g. by using the encryption technologies [11]. It is important to consider which subject can access which object by which operation in the access control model. An *access rule* is given in a tuple $\langle s, o, t\rangle$ which means that a subject s can manipulate an object o through an operation type t [6,10]. The system is referred to as *secure* if and only if (*iff*) every object is only accessed according to the access rules specified by the authorizer in the system. However, the illegal information flow between subjects and objects cannot be protected by the access control model if the subject accesses transitively information in the objects or the access rights are granted discretionarily.

The information flow model [6] aims at protecting from the illegal information flows among the entities. One security class is given to each entity in the system. A partially ordered *information flow* relation $\rightarrow$ among the security classes is defined to denote that information in one class s_1 can *flow into* another class s_2 (written as $s_1 \rightarrow s_2$). A set of the entities partially ordered by the information flow relation $\rightarrow$ is represented in a *lattice* [7]. In the *mandatory* access control model [9], the access rule $\langle s, o, t\rangle$ for each subject s and object o is specified so that the information flow relation between s and o is not violated. For example, if s reads o, the information in o flows into s. Hence, s can read o if the security class of o *can flow* to the security class of s. In the mandatory access control, only *read* and *write* operations are considered as the access types of the objects.

The information system is composed of multiple entities. The entities mean units of resources in the system. Databases, programs, and users are examples of the entities. Each entity is composed of abstract data structure and abstract operations for manipulating the data structure. The entities are manipulated only through operations supported by themselves. In addition, the operation computed in an entity may invoke operations on other entities, i.e. *nested* operations. In this paper, we would like to discuss non-discretionary access models showing how the information is flowing from operation to operation in the entities. First, we would like to propose an *object-based* access control model to prevent from the illegal information flow among the entities. The operations are classified into four types from the information flow point of view. The object-based access rules are defined based on the operation types. Next point is that the basic and mandatory access control models do not discuss *for what* the subjects manipulate the objects by the operations. For example, s may read data in o in order to just pass the data to another object o'. Here, we do not need to consider the information flow from o to s. Thus, it is essential to discuss the *purpose* of s to access o by t in this paper. In this paper, we newly discuss a *purpose-oriented* access control model where the access rules specify 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 of s invokes t to manipulate o.

In section 2, we overview the access control models. In section 3, the object-based access control is discussed. In section 4, we discuss the purpose-oriented access control model.

2 Access Control Models

2.1 Basic model

In the basic access control model [6], the system is composed of multiple entities. There are two roles of which the entities play, i.e. *subjects* and *objects*. The access to the entities is modeled so that a subject s manipulates an object o by an operation t. Let S, O, and T be sets of subjects, objects, and types of operations in the system, respectively. An access rule is specified in a tuple $\langle s, o, t\rangle \in S \times O \times T$, which means that $s \in S$ can manipulate $o \in O$ by $t \in T$. The system is *secure iff* every object is only accessed so that the access rules are satisfied.

For example, suppose that the authorizer specifies the access rules $\langle A, F, read\rangle$, $\langle A, G, write\rangle$, and $\langle B, G, read\rangle$ meaning that a subject A can read records in a file F and write records in G, but B cannot read records in F while being able to read records in G. If A reads some record d from F and writes d to G, B can read d from G without accessing F. That is, the information in F illegally flows to B. Thus, the illegal information flow between subjects and objects cannot be protected by the basic access control model.

2.2 Lattice-based model

The lattice-based model [6, 9] is proposed to keep the information flow legal in the system. Here, one *security class* is given to each entity. Here, let E be a set of entities and S be a set of security classes in the system. For each e_i in E, let $\lambda(e_i)$ denote a security class of e_i, i.e. $\lambda : E \rightarrow S$.

The legal information flow among the entities is denoted by the *can-flow* relation "$\rightarrow$" [6].

[Definition] For every pair of security classes s_1 and s_2 in S, s_1 *can flow* to s_2 ($s_1 \rightarrow s_2$) *iff* the information in an entity of a class 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 security class s_i in S, $s_i \equiv s_i$, i.e. $\rightarrow$ is reflexive. In addition, $\rightarrow$ is transitive. For example, suppose that there are three classes, *top secret*, *secret*, and *unclassified* in S. Suppose that one file entity A is in a class *secret* and another file B is in a class *unclassified*. There is an information flow relation among the classes in S, *unclassified* $\rightarrow$ *secret* $\rightarrow$ *top secret*. Here, the information in B can be stored in A although the information in A cannot be stored in B.

By using "$\rightarrow$", the *dominant* relation "$\preceq$" among the security classes is defined as follows.

[Definition] 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. For example, *unclassified* $\preceq$ *secret* $\preceq$ *top secret*. If $s_1 \preceq s_2$ or $s_2 \preceq s_1$, s_1 and s_2 are *comparable*. If neither $s_1 \preceq s_2$ nor $s_2 \succeq s_1$, s_1 and s_2 are not comparable ($s_1 \parallel s_2$). No information flow is allowed among s_1 and s_2 if $s_1 \parallel s_2$. Here, suppose that there are two entities e_1 and e_2 in E. If $\lambda(e_1) \preceq \lambda(e_2)$, the information in e_1 can flow into e_2 but the information in e_2 cannot flow into e_1. A partially ordered set S with $\preceq$ is given in a lattice $\langle S, \preceq, \cup, \cap\rangle$ [7] where $\cup$ and $\cap$ are the greatest lower bound (*glb*) and the least upper bound (*lub*), respectively. For two entities e_1 and e_2 in E, the information in an entity e can flow to both e_1 and e_2 if $\lambda(e) \preceq \lambda(e_1) \cap \lambda(e_2)$. e can get the information in e_1 and e_2 if $\lambda(e_1) \cup \lambda(e_2) \preceq \lambda(e)$.

In the mandatory access control model [1, 9], the access rule is defined based on the dominant relation $\preceq$. Here, there are two roles of entities, i.e. subjects and objects. We have to decide whether or not a subject s can manipulate an object o by an operation t. $\lambda(s)$ and $\lambda(o)$, are the security classes of s and o, respectively. There are three types, $T = \{read, write, modify\}$.

We would like to consider if a subject s could manipulate an object o by an operation t in T from the information flow relation between s and o. If s reads o, the information in o is derived and taken by s, i.e. the information in o flows to s. Hence, it is required that $\lambda(s) \succeq \lambda(o)$ be held for s to read o. Next, if s writes o, the data of s is stored in o, i.e. the information in s flows to o. Hence, $\lambda(s) \preceq \lambda(o)$. Lastly, the modification of o means that s reads *data* from o, changes the data, and writes the changed data into o. Hence, $\lambda(s) \preceq \lambda(o)$ and $\lambda(s) \succeq \lambda(o)$, i.e. $\lambda(s) \equiv \lambda(o)$. In the mandatory access control model, the following access rules are defined so that $\preceq$ is not violated.

[Mandatory access rules] Suppose that a subject s would manipulate an object o. The access rules are specified as follows :

(1) s can read o only if $\lambda(s) \succeq \lambda(o)$.
(2) s can write o only if $\lambda(s) \preceq \lambda(o)$.
(3) s can modify o only if $\lambda(s) \equiv \lambda(o)$.□

For example, *secret* or *top secret* persons can read data in an *unclassified* file, but *unclassified* persons cannot write a *secret* file. The *secret* file can be modified only by *secret* persons. Neither *top* *secret* nor *unclassified* persons can modify the *secret* file.

3 Object-Based Access Control Model

In the mandatory access control model, only *read* and *write* operations are considered for simple objects like files and relational databases. In this paper, we consider a system which is composed of multiple objects, where each object supports data structure and more abstract operations for manipulating the data structure than read and write. Each object o_i is encapsulated so that o_i can be accessed only through the operations supported by o_i. For example, the *schedule* object supports a *booking* operation which is internally realized by *read* and *write* accessing the internal data structure like tables. The users can manipulate the *schedule* object only by using *booking*, but not by *read* and *write*. In this paper, we would like to newly discuss an *object-based access control* model for the objects supporting more abstract, application-oriented complex operations.

First, suppose that a *subject* entity s manipulates an *object* entity o_i. If an operation op_i of o_i does not invoke any other operation, op_i is *unnested*. First, we assume that all operations in the system are unnested. In the traditional client-server systems like the database systems, the application programs in the clients manipulate the objects in the servers through the unnested operations like SQL [11]. The subject s sends a request message q of op_i to o_i. On receipt of the request message q, o_i computes op_i. Then, o_i sends the response r back to s. q is the input of op_i and r is the output of op_i. In addition, op_i may change the state of o_i by using the input. Here, the information in s may flow into o_i if the request q carries some data in s. op_i may derive the data from o_i as the output. Here, the information in o_i may flow out to s if r carries the data derived by op_i from o_i. Thus, it is important to think about the input and output of op_i to make clear the information flow relation among s and o_i. Each operation op_i of o_i is characterized by the following points:

(1) input data (I_i) to op_i,
(2) output data (O_i) from op_i, and
(3) state transition of o_i by op_i.

The input data I_i exists if some data flows from s to o_i. The output data O_i exists if some data in O_i flows out to s. In this paper, we assume that the communication among the subjects and objects are secure. In addition, each object o_i is encapsulated so that o_i can be accessed only through the operations supported by o_i. This means that only the information stored in o_i can flow out from o_i and the information in subjects can flow to o_i. Hence, it is important to discuss whether or not the operations change the state of o_i.

We would like to classify the operations of the objects into the following types from the information flow point of view [Figure 1]:

(1) non-flow (NF),
(2) flow-in (FI),
(3) flow-out (FO), and
(4) flow-in/out (FIO).

Here, let type(op_i) denote a type of op_i of o_i, i.e. type(op_i) $\in$ {NF, FI, FO, FIO}. In the NF operation op_i, there is no information flow from and to o_i. In addition, op_i does not change the state of o_i. That is, there is no information flow between o_i and s. Even if I_i exists, no information in s flows to o_i unless op_i changes the state of o_i. Similarly, no information in o_i flows out to s unless O_i is derived from o_i. If op_i is realized by read or write, op_i reads data from o_i or changes o_i. In addition, some data may be carried to op_i from s. However, unless the data is brought to or from o_i, there is no information flow from s to o.

The FI operation op_i changes the state of o_i by using the input data I_i. o_i is updated by using I_i which includes information in s. Hence, the information in s may flow into o_i. *write* is an example of the FI operation.

The FO operation op_i does not change the state of o_i. Since the output data O_i includes the information in o_i, O_i carries the information in o_i back to s. Here, the information in o_i may flow to s. *read* is an example of the FO operation.

Finally, the FIO operation op_i changes the state of o_i by using the input data I_i. In addition, op_i sends the output data O_i including the information in o_i to s. The information in s may flow into o_i and then the information in o_i may flow out to s. *modify* is an example of the FIO operation.

We introduce the following object-based access rules so that the information flow between s and o_i is not violated. Here, suppose that s manipulates o_i by an operation op_i supported by o_i.

[Object-based access rules]

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

The types of the operations and the security class of the object are specified when the objects are defined based on the semantics of the objects. Each time o_i is accessed in op_i by s, the access rules are tested to be satisfied by using type(op_i) and $\lambda(o_i)$. If op_i satisfies the access rule, op_i is allowed to be computed in o_i.

[Example] We would like to consider an example of a *bank* object b accessed by a person s. The *bank* object b supports operations *deposit*, *withdraw*, *check*, and *transfer*. *deposit* and *withdraw* are FI operations because the state of b is changed by using the input data. If $\lambda(s) \preceq \lambda(b)$, s can *deposit* to and *withdraw* from b. *check* is an FO operation because the output data is derived from b. If $\lambda(s) \succeq \lambda(b)$, s can *check* b. *transfer* is an FIO type. If $\lambda(s) \equiv \lambda(b)$, s can *transfer* in b.

4 Purpose-Oriented Access Control Model

4.1 Information flow

In the access control models like the mandatory access control model and object-based access control model, it is not discussed for what a subject s

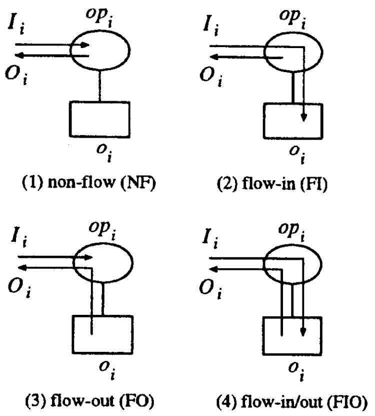


Figure 1: Information flow

accesses an object o_i by a type t_i of operation. In the access control model, a subject s accesses an object o_i through an operation t_i if an access rule $\langle s, o_i, t_i \rangle$ is authorized in the system. The subject s may just forward another object the information obtained from the object o_i. In the object-based access control model, s may just pass o_i the data obtained from another object o_j by the FI operation op_j. We assume that each entity e_i is so secure that e_i can be only accessed by the operations supported by e_i and the operation are secure. The data in e_i cannot be accessed by the lows-level operations like read and write. Thus, even if the data d in one entity e_i flows to another entity e_j, it is meaningless to consider the information flow from e_i to e_j if d is neither derived from e_i nor stored in e_j. If some data from e_i is stored in e_j, the data may flow out to another entities which access e_i.

Suppose that an entity e_i manipulates e_{ij} by issuing an operation op_{ij} to e_{ij}. It is important to discuss *for what* e_i manipulates e_{ij} by using op_{ij} of e_{ij} in addition to discussing if e_i can manipulate e_{ij} by op_{ij}. In this paper, we would like to discuss a new kind of the mandatory access control model named a ***purpose-oriented*** access control model by taking into account for what e_i manipulates e_{ij} by op_{ij}.

The entity e_i is modeled to manipulate e_{ij} by an operation op_{ij} while computing an operation op_i. That is, op_i in e_i invokes op_{ij} in e_{ij}. This is a nested operation. Thus, the purpose of e_i for manipulating e_{ij} is modeled to show by which *operation* in e_i invokes op_{ij} to manipulate e_{ij}. Hence, the access rule is written in a form $\langle e_i : op_i, e_{ij} : op_{ij} \rangle$ in the purpose-oriented model while the access rule $\langle e_i, e_{ij}, op_{ij} \rangle$ is specified in the access control models. Here, the access rule $\langle e_i : op_i, e_{ij} : op_{ij} \rangle$ means that e_i can manipulate e_{ij} by op_{ij} if e_i is computing op_i. For example, suppose that there are two entities ***person*** and ***bank***. Suppose that an access rule $\langle$***person*** : *house-keep*, ***bank*** : *withdraw*$\rangle$ is specified for two entities ***person*** and ***bank***. ***person*** can withdraw money from ***bank*** only for the housekeeping. However, ***person*** can withdraw no money from ***bank*** if ***person*** would go shopping. If the access rule $\langle$*person*, ***bank***, *withdraw*$\rangle$ is specified in the mandatory access control model, ***person*** can *withdraw* money from ***bank*** independently of why ***person*** accesses ***bank***. Thus, in the access rule $\langle e_i : op_i, e_{ij} : op_{ij} \rangle$, op_i shows the ***purpose*** why s accesses to o_{ij} by op_{ij}. Here, e_i and e_{ij} are named *parent* and *child* entities of the access rule, respectively.

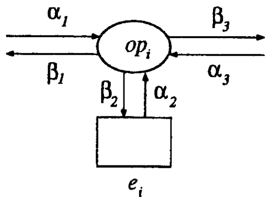


Figure 2: Input and output in op_i

Here, suppose that an entity e invokes an operation op_i in an entity e_i. op_i further invokes operations $op_{i1}, \ldots, op_{il_i}$ where each op_{ij} is computed in an entity e_{ij} $(j = 1, \ldots, l_i)$. An operation op_i in the entity e_i communicates with entities e and e_{ij} while transferring data to/from the state of e. Hence, op_{ij} is modeled to be a collection of three kinds of inputs α_1, α_2, and α_3, and outputs β_1, β_2, and β_3 as shown in Figure 2. Here, α_1 means the input data I_i from the entity e to op_i. For example, the request of op_i carries the input data I_i as α_1. β_1 means the output data O_i to s. The response of op_i is an example of β_1. β_2 shows that the state of e_i is changed by using data carried by β_2. For example, the data denoted by β_2 is stored in e_i. α_2 means that the information stored in e_i is derived to op_i. The output β_3 means that some data is output to e_{ij}. For example, the request of another operation op_{ij} with the input data I_{ij} is sent to e_{ij}. α_3 shows that some output data O_{ij} is carried to e_i from e_{ij}. For example, the response with the output data is sent from the operation op_{ij}.

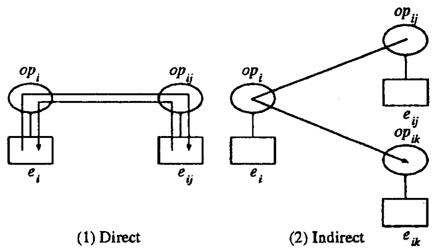


Figure 3: Information flow

We have to consider the following kinds of information flow :

(1) information flow between e_i and e_{ij} [Figure 3(1)] and

(2) information flow between e_{ij} and e_{ik} [Figure 3(2)].

(1) is named a *direct* or *parent-child* information flow. The information flows between the parent e_i and the child e_{ij}. (2) is named an *indirect* information flow. Figure 3 shows that the information

in e_{ij} flows to other e_{ik} via e_i but there is neither d direct flow between e_i and e_{ij} nor between e_i and e_{ik}. For example, op_{ij} derives the data from e_{ij}. op_i receives the data from op_{ij} and just passes the data to op_{ik}. op_{ik} stores the data in e_{ij}. In addition, suppose that op_{ij} invokes op_i and that op_i invokes op_{ik} where op_i just passes the data from op_{ij} to op_{ik}. This is also a case of the indirect information flow.

4.2 Direct information flow

The information in the parent e_i may flow to the child e_{ij} and vice versa. For example, if op_{ij} outputs the data derived from e_{ij} to e_i and op_i changes the state of e_i by using the data derived from op_{ij}, the information in e_{ij} flows to e_i. First, we would like to discuss the direct information flow. There are kinds of operations, i.e. NF, FI, FO, and FIO as presented in the object-based access control model. However, the parent-child relations as shown in Figure 2 are not taken into account since the operations are unnested. We have to consider the relation between the output β_1 of op_i and the input α_1 of op_{ij} and between the output β_3 of op_{ij} and input α_3 of op_i.

Each operation op_i in o_i with a child op_{ij} in o_{ij} is characterized to be in a form π_i/ω_{ij}. π_i means the type of op_i with respect to the input α_1 and output β_1 with the child e_{ij} as discussed in the object-based access control. As shown in Figure 1, $\pi_i \in$ {NF, FI, FO, FIO}. For example, if π_i is FO, op_i sends the data in e_i to the child e_{ij} through β_3. ω_{ij} is defined for the input α_3 and the output β_3 of the child entity e_{ij} with e_i similarly to π_i, i.e. $\omega_{ij} \in$ {NF, FI, FO, FIO}. Here, π_i and ω_{ij} are named the *parent* type of op_i and the *child* type of op_{ij}, respectively. In the direct information flow, op_i in e_i manipulates e_{ij} by op_{ij}, i.e. op_i invokes op_{ij}. For example, in the type FO/FI, the output derived from e_i by op_i flows to e_{ij} through op_{ij}.

There are kinds of relations between op_i and op_{ij} as shown in Figure 4. Let π_i and ω_{ij} be the output type of op_i and the input type of op_{ij}, respectively, where op_i invokes op_{ij}. The access control rules for a pair of the parent e_i and the child e_{ij} are specified as follows.

[Direct access rules] [Figure 4]

(1) $\pi_i/\omega_{ij} \in$ {FO/FI, FO/FIO, FIO/FI} only if $\lambda(e_i) \preceq \lambda(e_{ij})$.

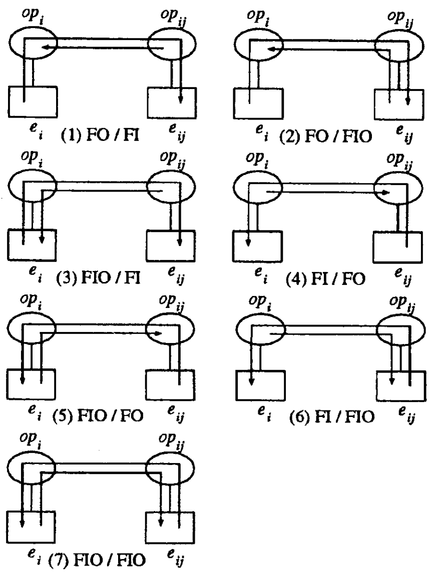
(2) $\pi_i/\omega_{ij} \in$ {FI/FO, FIO/FO, FIO/FO} only if $\lambda(e_i) \succeq \lambda(e_{ij})$.

(3) $\pi_i/\omega_{ij} \in$ {FIO/FIO} only if $\lambda(e_i) \equiv \lambda(e_{ij})$.□

For example, if the type π_i/ω_{ij} of op_i and op_{ij} is the FO/FI type as shown in Figure 4(1), the information in the parent e_i flows to the child e_{ij}. Hence, $\lambda(e_i) \preceq \lambda(e_{ij})$. In the FI/FO type [Figure 4(4)], the information in e_{ij} flows to e_i. Hence, $\lambda(e_i) \succeq \lambda(e_{ij})$. In the FIO/FIO type [Figure 4(7)], the information in e_i flows to e_{ij} while the information in e_{ij} flows to e_i. Hence, $\lambda(e_i) \equiv \lambda(e_{ij})$.

4.3 Indirect information flow

Next, we discuss the indirect information flow. Suppose that there are three entities o_i, o_{ij}, and



op_i
op_{ij}
e_i (1) FO / FI e_{ij}
e_i (2) FO / FIO e_{ij}
e_i (3) FIO / FI e_{ij}
e_i (4) FI / FO e_{ij}
e_i (5) FIO / FO e_{ij}
e_i (6) FI / FIO e_{ij}
e_i (7) FIO / FIO e_{ij}

Figure 4: Direct information flow

o_{ik} supporting operations op_i, op_{ij}, and op_{ik}, respectively. One case is that op_i invokes op_{ij} and op_{ik}. This shows an inter-child information flow. There is another case that op_{ij} invokes op_j and then op_i invokes op_{ik}. There is a transitive information flow. We discuss how the information is indirectly flowing among the entities e_{ij} and e_{ik} via the entity e_i.

We discuss the indirect information flow by using the input-output types π_i/ω_{ij} and π_i/ω_{ik}. Suppose that π_i/ω_{ij} is FO/FIO and π_i/ω_{ik} is NF/FI. The data in e_{ij} and e_{ik} are brought into e_i. Hence, the information may flow from e_{ij} to e_{ik} via e_i if the data from e_{ik} is stored in e_i and the data in e_i is read by e_{ij}. This case can be discussed in the combination of two direct relations π_i/ω_{ij} and π_i/ω_{ik}, i.e. $\lambda(e_{ij}) \preceq \lambda(e_{ik})$ since $\lambda(e_{ij}) \preceq \lambda(e_i)$ and $\lambda(e_i) \preceq \lambda(e_{ik})$. Therefore, we have to discuss a case that the information flowing from e_{ij} or e_{ik} to e_i is not stored in e_i. Figure 5 shows the possible information flows among the child operations op_{ij} and op_{ik} through the parent op_i. For example, in Figure 5(a), π_i/ω_{ij} is NF/FIO and π_i/ω_{ik} is FO/FO. No information in e_{ij} and e_{ik} flows into e_i while op_i sends the data in e_i to op_{ik}. Here, the information in e_{ik} flows from e_{ik} to e_{ij} via e_i. Hence, $\lambda(e_{ik}) \preceq \lambda(e_{ij})$ has to be held while any dominant relation with e_i is allowed. In Figure 5(c), π_i/ω_{ij} is NF/FIO and π_i/ω_{ik} is NF/FO. Here, the information in e_{ij} flows into e_{ik} and vice versa. Hence, $\lambda(e_{ij}) \equiv \lambda(e_{ik})$.

When considering the indirect information flow between e_{ij} and e_{ik}, it is critical to discuss which operation op_{ij} or op_{ik} precedes the other in e_i. For example, in Figure 5(a), unless op_i invokes op_{ij} after op_{ik}, there is no information flow from op_{ik} to op_{ij}. However, it is difficult, even impossible to know the precedence among op_{ij} and op_{ik}

because the data flow relation in op_i has to be analyzed. Hence, in this paper, we assume that every operation invoked in op_i is interrelated, i.e. there is assumed to be some information flow between op_{ij} and op_{ik} in op_i.

Now, we discuss the access rules for the indirect information flow. The access rules on op_{ij} of e_{ij} and op_{ih} of e_{ih} for op_i of e_i are specified as follows.

[Indirect access rules]

(1) $\pi_i/\omega_{ij} \in$ {NF/FIO, FI/FIO, FO/FIO, FIO/FIO} and $\pi_i/\omega_{ik} \in$ {NF/FIO, FI/FIO, FO/FIO, FIO/FIO} only if $\lambda(e_{ij}) \equiv \lambda(e_{ik})$.

(2) $\pi_i/\omega_{ij} \in$ {NF/FI, NF/FIO, FI/FI, FI/FIO} and $\pi_i/\omega_{ik} \in$ {NF/FO, NF/FIO, FO/FO, FO/FIO} only if $\lambda(e_{ij}) \succeq \lambda(e_{ik})$.

(3) $\pi_i/\omega_{ij} \in$ { NF/FO, NF/FIO, FO/FO, FO/FIO } and $\pi_i/\omega_{ik} \in$ {NF/FI, NF/FIO, FI/FI, FI/FIO} only if $\lambda(e_{ij}) \preceq \lambda(e_{ik})$. □

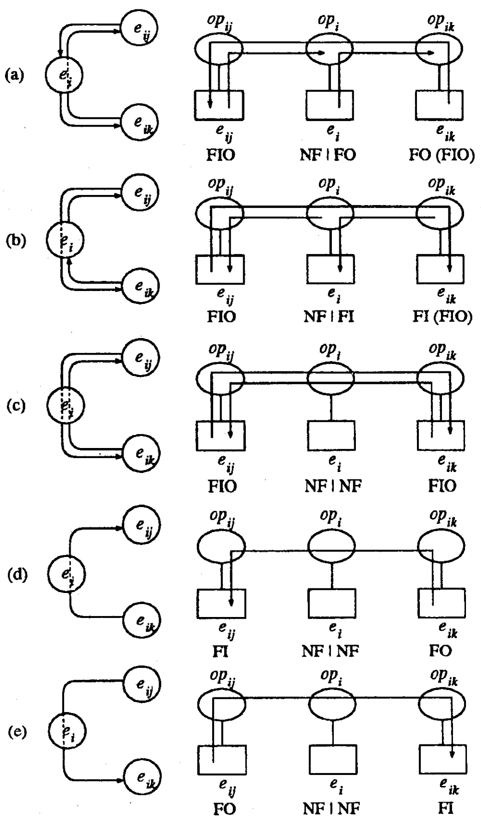


Figure 5: Indirect information flow

For example, if π_i/ω_{ij} = NF/FI and π_i/ω_{ik} = NF/FO, the information in e_{ij} flows to e_{ik}. Hence, $\lambda(e_{ij}) \preceq \lambda(e_{ik})$ has to be held while there is a dominant relation neither between e_{ij} and e_i nor between e_{ik} and e_i.

Suppose that op_i in the entity e_i invokes op_{ij} and op_{ij} in the entity e_{ij} further invokes op_{ijk} in an entity e_{ijk}. If op_{ij} and op_{ijk} are in NF/FO or NF/FIO, the information derived from e_{ijk} does not flow into e_{ij}. If e_i and e_{ij} are in FI/NF or FIO/NF, the information passed from e_{ijk} may flow into e_i. Here, $\lambda(e_i) \succeq \lambda(e_{ijk})$ has to be held while there is neither information flow relation between $\lambda(e_i)$ and $\lambda(e_{ij})$ nor between $\lambda(e_{ij})$ and $\lambda(e_{ijk})$. The inter-child access rules can be adopted by replacing e_i with e_{ij}, e_{ij} with e_{ijk}, and e_{ij} with e_i.

5 Concluding Remarks

Each entity supports more abstract operations than *read* and *write* and where the operations are nested. In this paper, we have discussed how to keep the information flow among the entities legal. In this paper, we have discussed the ***object-based access control*** model for the system composed of subject and object entities where operations are not nested. In addition, we have proposed newly a ***purpose-oriented*** access control model which takes into account for what an entity manipulates other entities. In this paper, the purpose is represented in an operation of the subject entity which invokes the operations of other entities.

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] Bertino, E., Samarati, P., and Jajodia, S., "High Assurance Discretionary Access Control in Object Bases," *Proc. of the 1st ACM Conf. on Computers and Communication Security*, 1993, pp.140–150.

[3] Biba, K. J., "Integrity Considerations for Secure Computer Systems," *Mitre Corp. Report* TR-3153, *Bedford, Mass.*, 1977.

[4] Castano, S., Fugini, M., Matella, G., and Samarati, P., "Database Security," *Addison-Wesley*, 1995.

[5] Date, C. J., "An Introduction to Database Systems," Vol.1, *Addison-Wesley*, 1992.

[6] Denning, D. E. and Denning P. J., "Cryptography and Data Security," *Addison-Wesley Publishing Company*, 1982, pp.191–327.

[7] Maclane, S. and Birkoff, G., "Algebra," *Collier Macmillan*, 1979,

[8] Merkl, D., Tjoa, A. M., and Vieweg, S., "BRANT - An Approach for Knowledge Based Document Classification Retrieval Domain," *Proc. of DEXA '92*, 1992, pp.254–259.

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

[10] Sandhu, R. S., "Role-Based Access Control Models," *IEEE Computer*, Vol.29, No.2, 1995, pp.38–47.

[11] Schneier, B., "Applied Cryptography," *John Wiley & Sons*, 1996,

[12] Takizawa, M. and Mita, H., "Secure Group Communication Protocol for Distributed Systems," *Proc. of IEEE COMPSAC'93*, 1993, pp.159–165.