Journal of Information Processing Vol. 17 318–327 (Dec. 2009)

Regular Paper

Matrix-Based Algorithm for Integrating Inheritance Relations of Access Rights for Policy Generation

Kazuhiro Kono,[†1] Yoshimichi Ito,[†1]
Akihito Aoyama,[†2] Hiroaki Kamoda[†1,†2]
and Noboru Babaguchi[†1]

This paper presents a matrix-based algorithm for integrating inheritance relations of access rights for generating integrated access control policies which unify management of various access control systems. Inheritance relations of access rights are found in subject, resource, and action categories. Our algorithm first integrates inheritance relations in each category, and next, integrates inheritance relations of all categories. It is shown that these operations can be carried out by basic matrix operations. This enables us to implement the integration algorithm very easily.

1. Introduction

With the increase and distribution of information, data security and privacy are critical issues. System administrators pay a lot of attention in order to prevent information leakage caused by fraudulent actions and mis-operations, and introduce access control systems to establish access control policies for various applications and file systems.

Since each access control system has its own access control policy, administrators have to configure policies individually in all systems. Therefore, when the personal information protection law and the security guideline of organizations are changed, the administrators have to pay enormous attention for updating every policy for each system. Thus, the policy generation framework for unifying management of various access control systems is strongly required.

Regarding the policy generation framework for unifying management of various access control systems, several methods have been proposed [4),6),9),12),13),17)]. Among them, Refs. 4), 6), and 9) consider the policy generation framework by integrating *inheritance relations* of access rights. An inheritance relation is a partial order defining a seniority relation between the attributes and behaviors of objects, whereby the senior attributes and behaviors of objects acquire the permissions of their juniors, and thus, the access rights of their juniors are inherited to the seniors. Such inheritance relations are found in subject, resource, and action categories. Inheritance relations are introduced in various access control systems, e.g., Microsoft Windows® Rights Management Services (RMS) [14)] for OFFICE documents, Adobe® LiveCycle™ Rights Management ES [1)] for PDF documents, and *PolicyComputing*™ [18)].

Concerning the integration of the inheritance relations of different systems, Gong, et al. discuss complexity, composability, and conditions for integration in Ref. 9). In Ref. 6), Dawson, et al. consider the conditions for which the integrated inheritance relations do not include conflicts of inheritance relations and redundant inheritance relations. Although these two works clarify the requirements for integrating the inheritance relations, they do not offer a specific method for generating integrated inheritance relations. In contrast with these two works, Bonatti, et al. propose an algorithm for generating the integrated inheritance relations in Ref. 4). In their work, they introduce a graph theoretic approach and derive a logical programming based algorithm for integrating inheritance relations. However, their algorithm seems rather complicated and is not easily implementable. Another drawback of the above three works is that they only deal with the inheritance relations in a single category.

The purpose of this paper is to provide an easily implementable algorithm for integrating the inheritance relations of access control systems. In contrast with previous works, the algorithm can deal with the inheritance relations in subject, resource, and action categories. Furthermore, since the algorithm is based on basic matrix operations, it is easily implementable.

The procedure for generating integrated policies of access control systems is as follows: first, integrate inheritance relations in a single category; second, integrate inheritance relations of all categories; third, establish fundamental policies by administrators, and generate related policies automatically by referring to the

†1 Osaka University
†2 NTT DATA CORPORATION

integrated inheritance relations. The first and second steps are for generating integrated inheritance relations, and these operations can be accomplished by matrix operations. The derivation of the algorithm is based on graph theoretic approach, where a useful property of adjacency matrices is exploited. It is also shown that the elimination of conflicts and removal of redundant inheritance relations are also carried out by matrix operations.

This paper is organized as follows: In Section 2, we introduce inheritance relations of access rights for subject, resource, and action categories. Section 3 presents an overview of our proposed method, and indicates the direction for integrating inheritance relations using graphs. In Section 4, we provide a method for generating an integrated inheritance relation using adjacency matrices. Section 5 summarizes the results of this paper with some remarks.

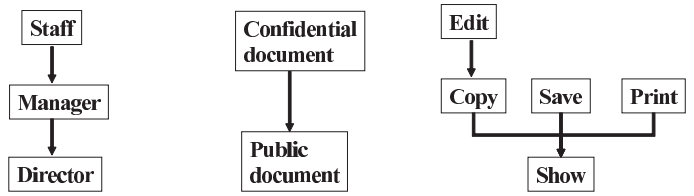
2. Inheritance Relations

In policy specification languages, e.g., eXtensible Access Control Markup Language (XACML) [15),16)], every access control rule is essentially specified by three elements, i.e., *subject*, *resource*, and *action*. Subjects are the entities that can perform actions in systems (e.g., users, computers). Resources are the entities that contain or receive information for which access is requested (e.g., files, devices). Actions are the types of access that is being requested (e.g., edit, copy, read, write).

In many practical situations, we can find inheritance relations of access rights in each category of the above three elements. Examples of inheritance relations in subject, resource, and action categories are shown in **Fig. 1** (a), Fig. 1 (b) and Fig. 1 (c), respectively.

In Fig. 1 (a), the arrow directed from "Staff" to "Manager" represents that if staff can perform an action on a resource, managers can perform the same action on the resource. Similarly, the arrow from "Confidential document" to "Public document" in Fig. 1 (b) indicates that when a subject can perform an action on confidential documents, the subject can perform the same action on public documents. In the same way, the arrow from "Edit" to "Copy" in Fig. 1 (c) implies that if a subject can edit a resource, the subject can also copy the resource.

In general, an inheritance relation is described by the following statement: if a subject s_i can perform an action a_m on a resource r_k, a subject s_j can perform an action a_n on a resource r_l. We express this statement as $(s_i, r_k, a_m) \rightarrow (s_j, r_l, a_n)$. As shown in Fig. 1, we focus on the inheritance relations described in a single category. Therefore, we only deal with the inheritance relations expressed by the following forms:



(a) Subject category (b) Resource category (c) Action category

Fig. 1 Examples of inheritance relations.

inheritance relations in subject category:

$$(s_i, r_k, a_m) \rightarrow (s_j, r_k, a_m)\ (\forall r_k \in \mathrm{RES}, \forall a_m \in \mathrm{ACT})$$

inheritance relations in resource category:

$$(s_i, r_k, a_m) \rightarrow (s_i, r_l, a_m)\ (\forall s_i \in \mathrm{SUB}, \forall a_m \in \mathrm{ACT})$$

inheritance relations in action category:

$$(s_i, r_k, a_m) \rightarrow (s_i, r_k, a_n)\ (\forall s_i \in \mathrm{SUB}, \forall r_k \in \mathrm{RES})$$

where RES, ACT, and SUB represent the sets of resources, actions, and subjects, respectively.

In the first expression, the access right for subject s_i is inherited to subject s_j regardless of resources and actions. In this case, we simply express the inheritance relation as $s_i \rightarrow s_j$. In such a case, we say that the inheritance relation is *independent* from resource and action categories. The independence of inheritance relation in resource category and that in action category are similarly defined, and we use notations $r_k \rightarrow r_l$ and $a_m \rightarrow a_n$ for the above cases. Throughout this paper, we assume that all inheritance relations in each category are independent from other categories.

3. Overview of the Integration of Inheritance Relations

As shown in Fig. 1, inheritance relations are expressed by directed graphs. First, we introduce some notions in graph theory [3),5)] to express inheritance re-

lations.

A directed graph $G(V, E)$ consists of a set of vertices denoted by V, and a set of edges denoted by E. An edge $e = (v, u)$ in E is an ordered pair of two different vertices v and u in V, where v and u are called *initial* and *terminal* vertices of the edge e, respectively. An edge is called a loop if its initial vertex and terminal vertex are the same. We only deal with a directed graph that has no loops throughout the paper. A path in a directed graph is defined as a finite sequence $e_1 e_2 \cdots e_n$ of multiple edges where the terminal vertex of each edge coincides with the initial vertex of the succeeding edge, and we refer to the number of edges as the length of the path. A path $e_1 e_2 \cdots e_n$ is called a circuit if the initial vertex of e_1 coincides with the terminal vertex of e_n.

3.1 Integration of Inheritance Relations in a Single Category

The integration of inheritance relations is accomplished as follows: first, integrate inheritance relations for each category; second, integrate inheritance relations for three categories. As stated in Section 2, we assume that all inheritance relations in each category are independent from other categories. Therefore, in this subsection, we only deal with inheritance relations such as $a_m \rightarrow a_n$, and do not consider general inheritance relations such as $(s_i, r_k, a_m) \rightarrow (s_j, r_l, a_n)$.

3.1.1 Basic Operation for Integration

Here, we provide a method for integrating inheritance relations in action category by using graphs. The integration in subject and resource categories is carried out in a similar manner.

Let $G_k(\mathrm{ACT}_k, \mathrm{IR}_k^{\mathrm{ACT}})$ be the graph which represents the inheritance relation in action category of a system k ($k \in \{1, \cdots, N\}$), where ACT_k is a set of vertices representing the actions of system k, and $\mathrm{IR}_k^{\mathrm{ACT}}$ is a set of edges representing the inheritance relations between the actions of system k. The basic operation for integrating the inheritance relations is the operation taking the union of these inheritance relations:

$$\mathrm{ACT} = \mathrm{ACT}_1 \cup \mathrm{ACT}_2 \cup \cdots \cup \mathrm{ACT}_N, \quad (1)$$

$$\mathrm{IR}^{\mathrm{ACT}} = \mathrm{IR}_1^{\mathrm{ACT}} \cup \mathrm{IR}_2^{\mathrm{ACT}} \cup \cdots \cup \mathrm{IR}_N^{\mathrm{ACT}}, \quad (2)$$

where ACT is the set of actions of the integrated system, and $\mathrm{IR}^{\mathrm{ACT}}$ is the set of inheritance relations between the actions of the integrated system. By this, we can obtain the graph which represents the integrated inheritance relation of the integrated system as $G(\mathrm{ACT}, \mathrm{IR}^{\mathrm{ACT}})$.

However, the above simple operations may cause problems due to the existence of circuits and redundant edges [6]. In the following subsections, we explain the problems, and provide the ways to remove these problems.

3.1.2 Circuits

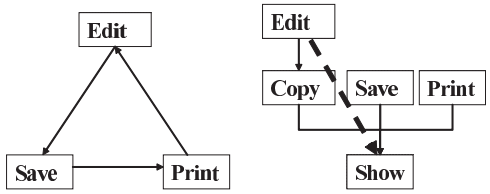
An example of a circuit in action category is shown in **Fig. 2** (a). In this example, the actions "Edit", "Save", and "Print" are cyclically dependent. In this case, if the "edit" action is permitted to a subject on a resource, the "save" and "print" actions are also permitted to the subject on the resource.

The problem due to the circuits caused by the integration of the inheritance relations of all access control systems is that the integration yields conflicts between the integrated inheritance relations and the inheritance relations of some access control systems. When circuits are generated, there are two different ways to cope with this problem:

- If administrators think that the conflicts will cause serious problems, stop the integration.
- If administrators think that the problems caused by the conflicts are not serious, continue the integration.

In the latter case, we unify cyclically dependent actions as a single set of actions, and eliminate the circuits. The methods for the detection and the elimination of the circuits will be presented in Section 4.2.2, where it is shown that they can be accomplished by some basic matrix operations.

Remark 1 In general, to determine whether the above conflicts are serious or not is a difficult task because it depends on the situation. One of the way for



(a) A circuit (b) A redundant edge

Fig. 2 Examples of a circuit and a redundant edge.

this determination is to refer to the security guidelines of the organization. If the integrated inheritance relations contradict their security guideline, we must stop the integration. Otherwise, we unify cyclically dependent inheritance relations.

3.1.3 Redundancy

First, we give the definition of a redundant edge.

Definition 1 Let $V = \{v_1, v_2, \cdots, v_p\}$ denote a set of vertices in a directed graph without circuits. Let e_{ij} be an edge directed from v_i to v_j. The edge e_{ij} is redundant if there exists another path directed from v_i to v_j.

An example of a redundant edge is shown in Fig. 2 (b), where a redundant edge is denoted by a dashed arrow.

The inheritance relations corresponding to the redundant edges are not needed for integrated inheritance relations since such relationships can be obtained by tracing the corresponding paths. Therefore, we need to detect and remove all redundant edges. The procedures for the detection and the removal of the redundant edges using basic matrix operations are given in Section 4.2.3.

Summarizing this subsection, the integration of inheritance relations in a single category can be accomplished by the following steps:

1. Take a union of inheritance relations for all systems.
2. Detect and eliminate circuits by unifying cyclically dependent inheritance relations.
3. Detect and remove redundant edges.

3.2 Integration of Inheritance Relations for Different Categories

In this subsection, we present a basic idea for integrating inheritance relations of different categories through a simple example. A general method using adjacency matrices will be presented in Section 4.3.

Here, we consider the integration of the inheritance relations of subject, resource, and action categories given by graphs $G_{\rm S}(V_{\rm S}, E_{\rm S})$, $G_{\rm R}(V_{\rm R}, E_{\rm R})$, and $G_{\rm A}(V_{\rm A}, E_{\rm A})$, respectively, where

$$V_{\rm S} = \{s_1, s_2\},\ V_{\rm R} = \{r_1, r_2\},\ V_{\rm A} = \{a_1, a_2\},$$
$$E_{\rm S} = \{s_1 \to s_2\},\ E_{\rm R} = \{r_1 \to r_2\},\ E_{\rm A} = \{a_1 \to a_2\}.$$

First, we integrate the inheritance relations of resource and action categories. By the inheritance relations $r_1 \to r_2$ and $a_1 \to a_2$, and from the assumption that inheritance relations are independent from other categories, the integrated inheritance relations are obtained as follows:

$$\begin{aligned}&(s_i, r_1, a_1) \to (s_i, r_1, a_2), \quad (s_i, r_2, a_1) \to (s_i, r_2, a_2),\\ &(s_i, r_1, a_1) \to (s_i, r_2, a_1), \quad (s_i, r_1, a_2) \to (s_i, r_2, a_2) \qquad (i = 1, 2).\end{aligned} \tag{3}$$

For simplicity, we rewrite the above relations as $r_1a_1 \to r_1a_2$, $r_2a_1 \to r_2a_2$, $r_1a_1 \to r_2a_1$, $r_1a_2 \to r_2a_2$. From this, it is observed that these inheritance relations can be regarded as the edges of the graph defined on the vertex set $\{r_1a_1, r_1a_2, r_2a_1, r_2a_2\}$.

Next, we integrate the inheritance relation $s_1 \to s_2$ in subject category, and the inheritance relations given by Eq. (3). Since Eq. (3) holds for $i = 1, 2$, we obtain the following eight inheritance relations:

$$\begin{aligned}&s_1r_1a_1 \to s_1r_1a_2,\ s_1r_2a_1 \to s_1r_2a_2,\ s_1r_1a_1 \to s_1r_2a_1,\ s_1r_1a_2 \to s_1r_2a_2,\\ &s_2r_1a_1 \to s_2r_1a_2,\ s_2r_2a_1 \to s_2r_2a_2,\ s_2r_1a_1 \to s_2r_2a_1,\ s_2r_1a_2 \to s_2r_2a_2,\end{aligned} \tag{4}$$

where $s_ir_ka_m$ is an abbreviation of (s_i, r_k, a_m). In addition, since inheritance relation in subject category is independent from action and resource categories, the access right for s_1 is inherited to s_2 whatever resources and actions may be. Therefore, we obtain four inheritance relations as

$$s_1r_1a_1 \to s_2r_1a_1,\ s_1r_1a_2 \to s_2r_1a_2,\ s_1r_2a_1 \to s_2r_2a_1,\ s_1r_2a_2 \to s_2r_2a_2. \tag{5}$$

From this, it is observed that the graph representing the integrated inheritance relations is composed of the set of the edges given by Eqs. (4) and (5), and the set of the vertices $\{s_1r_1a_1, s_1r_1a_2, s_1r_2a_1, s_1r_2a_2, s_2r_1a_1, s_2r_1a_2, s_2r_2a_1, s_2r_2a_2\}$.

As shown above, the integration of inheritance relations of different categories is too complicated to implement even for the above simple case. In Section 4.3, we present a general procedure for integrating inheritance relations of different categories, and show that the procedure can be easily implemented by using adjacency matrices.

4. Integration of Inheritance Relations Using Adjacency Matrices

In this section, we present a matrix based algorithm for integrating inheritance relations in a single category, as well as a method for integrating inheritance relations of all categories. The expressions of graphs via *adjacency matrices* play crucial roles.

4.1 Adjacency Matrices and Their Properties

Here, we give the definition of adjacency matrices, as well as the definitions of summation and multiplication of matrices. We also introduce a useful property of adjacency matrices.

Definition 2 Let G be a directed graph with a vertex set $V = \{v_1, v_2, \cdots, v_n\}$. The adjacency matrix R of the graph G is defined as $R = [r_{ij}]$ $(i = 1, \cdots, n;\ j = 1, \cdots n)$ where each element r_{ij} is given by

$$r_{ij} = \begin{cases} 1 & \text{(if there is an edge from } v_i \text{ to } v_j\text{)} \\ 0 & \text{(otherwise)} \end{cases}. \tag{6}$$

The summation and multiplication of adjacency matrices are the same as the usual matrix operations except for the elementwise summation.

Definition 3 Sum of elements a and b is defined as the logical sum, that is,

$$a + b = \begin{cases} 0 & (a = b = 0) \\ 1 & \text{(otherwise)} \end{cases}. \tag{7}$$

The following lemma concerns the condition for the existence of a path connecting two vertices, and is exploited for detecting circuits and redundant edges.

Lemma 1 Let R be an adjacency matrix of a directed graph G whose vertex set is given by $V = \{v_1, v_2, \cdots, v_n\}$. Then, there exists a path of length p directed from v_i to v_j if and only if the (i, j) element of R^p is equal to 1 [3),5)].

Remark 2 As mentioned in Section 3, we only deal with a directed graph without loop. Therefore, the diagonal elements of all adjacency matrices are equal to zero.

4.2 Integration in a Single Category Using Adjacency Matrices

In this section, we give a matrix based method for integrating inheritance relations in a single category. Although we derive the method for action category, the method can be applied for other categories in a similar way.

Let the number of systems be N, and inheritance relations in action category of a system k be represented by the graph $G_k(\mathrm{ACT}_k, \mathrm{IR}_k^{\mathrm{ACT}})$ $(k = 1, \cdots, N)$. In integrating these inheritance relations, the size of the corresponding adjacency matrices of all systems must be the same, since matrix operations include matrix summation. Therefore, we first rewrite graphs by replacing the vertex set ACT_k by ACT given by Eq. (1).

In the following, we give a matrix based algorithm for the integration according to the procedure at the end of Section 3.

4.2.1 The First Operation: Taking a Union

The first operation is to take a union of inheritance relations. This is accomplished by adding adjacency matrix corresponding to the inheritance relations of each system. Let A_k be an adjacency matrix expressing the inheritance relations in action category of system k. Then, the adjacency matrix A corresponding to the integrated inheritance relation in action category is given by

$$A = A_1 + A_2 + \cdots + A_N. \tag{8}$$

4.2.2 The Second Operation: Detection and Elimination of Circuits

We can detect and eliminate all circuits by repeated use of the following procedure until all circuits in the graph are eliminated.

1. remove all vertices composing a circuit and introduce a new vertex instead.
2. connect the edges associated with removed vertices to the new vertex.

For this procedure, the following theorems are useful:

Theorem 1 Let G be a directed graph with n vertices and R be an adjacency matrix of G. Then, G does not include circuits if and only if $R^n = 0$.

Theorem 2 Let G be a directed graph and R be its adjacency matrix. Then, there exists a circuit of length l including v_i if and only if (i, i) element of R^l is equal to 1.

These theorems are easily proved by Lemma 1, so they are omitted.

Based on the above two theorems, the procedure is carried out by the following matrix operations:

Step. 1 Let $l = 2$.

Step. 2 Compute A^l.

Step. 3 If some diagonal elements of A^l are equal to 1, at least one circuit exists. In this case, go to the following steps. Otherwise, go to Step. 4.

- Step. i Choose a circuit and select a vertex v_i consisting the circuit.
- Step. ii Replace the i-th column by the logical sum of the columns corresponding to the vertices of the circuit for A. Similar operation is applied for rows.
- Step. iii Replace the (i, i) element of A by 0.

Step. iv Remove all columns and rows corresponding to the vertices of the circuit except for the i-th column and i-th row.

Step. v Redefine the new generated matrix as A and return to Step. 1.

Step. 4 If $A^l = 0$, the graph does not include any circuits, and the algorithm is finished. Otherwise, add 1 to l and return to Step. 2.

The implication of Step. 3 in this algorithm is as follows: In Step. i, a selected vertex is regarded as a new vertex. In Step. ii, add every edge of each vertex in a circuit to the selected vertex in the circuit. In Step. iii, remove all loops because we have assumed that every graph has no loop. in Step. iv, remove all the vertices in the circuit except for the selected vertex.

Remark 3 One may think that the graph obtained by the above algorithm depends on the choice of a vertex in Step. i, and the graph is not determined uniquely. However, as seen from Step. ii to Step. iv, the graph is uniquely determined whatever a chosen vertex of a circuit may be, since the newly generated vertex has every edge of each eliminated vertex.

4.2.3 The Third Operation: Detection and Removal of Redundant Edges

The following theorem is used for detecting redundant edges, which is easily derived from Lemma 1.

Theorem 3 Let G be a directed graph without circuits and R be an adjacency matrix of G. Then, the edge directed from v_i to v_j is redundant if and only if the (i, j) element of R^l is equal to 1 for some $l \geq 2$.

Based on Theorem 3, a procedure for detecting and removing redundant edges via matrix operations is obtained as follows:

Step. 1 Let $l = 2$.

Step. 2 Compute A^l.

Step. 3 If (i, j) elements of A and A^l are equal to 1, replace the (i, j) element of A by 0, add 1 to l, and return Step. 2.

Step. 4 If $A^l = 0$, the graph does not include any redundant edges, and the algorithm is finished. Otherwise, add 1 to l and return to Step. 2.

Here, we note that Theorem 1 ensures that the algorithms for detection and removal of circuits and redundant edges finish within finite steps.

Remark 4 The graph obtained by the above algorithm is uniquely determined since the above algorithm is free from the problem due to the choice of a vertex, which appears in the elimination of a circuit.

4.3 Integration of Inheritance Relations for Different Categories Using Adjacency Matrices

Before presenting a method for integrating inheritance relations of three categories, we present a method for integrating inheritance relations of two different categories using adjacency matrices. Let $G(\mathrm{RES}, \mathrm{IR}^{\mathrm{RES}})$ and $G(\mathrm{ACT}, \mathrm{IR}^{\mathrm{ACT}})$ be the graphs associated with the inheritance relations in subject category and action category, respectively, where $\mathrm{RES} = \{r_1, \cdots, r_{n_R}\}$ and $\mathrm{ACT} = \{a_1, \cdots, a_{n_A}\}$, and corresponding adjacency matrices be R and A, respectively.

Now, we generate the adjacency matrix X_{RA} associated with the integrated inheritance relations of resource and action categories. As seen from the observation in Section 4.2, the graph representing the integrated inheritance relation of resource and action categories is defined on the vertex set given by

$$\mathrm{RES} \times \mathrm{ACT} = \{r_1a_1, \cdots, r_1a_{n_\mathrm{A}}, \cdots, r_{n_\mathrm{R}}a_1, \cdots, r_{n_\mathrm{R}}a_{n_\mathrm{A}}\}.$$

Therefore, the order of the adjacency matrix associated with the integrated inheritance relation of resource and action categories is $n_\mathrm{R} \times n_\mathrm{A}$.

First, note that the inheritance relations in resource category hold whatever action may be. Therefore, we can obtain the following adjacency matrix X_{RA_1}, which corresponds to the dashed line of **Fig. 3**:

$$X_{\mathrm{RA}_1} = R \otimes I_{n_\mathrm{A}} = \begin{bmatrix} r_{11}I_{n_\mathrm{A}} & \cdots & r_{1n_\mathrm{R}}I_{n_A} \\ \vdots & \ddots & \vdots \\ r_{n_\mathrm{R}1}I_{n_\mathrm{A}} & \cdots & r_{n_\mathrm{R}n_\mathrm{R}}I_{n_\mathrm{A}} \end{bmatrix}, \tag{9}$$

where r_{ij} is the (i, j) element of the matrix R, I_{n_A} is the identity matrix of order n_A, and $\otimes$ denotes Kronecker product defined as follows: For a matrix $X = [x_{ij}]$ of the size k-by-l and a matrix Y, the Kronecker product of X and Y is defined as

$$X \otimes Y := \begin{bmatrix} x_{11}Y & \cdots & x_{1l}Y \\ \vdots & & \vdots \\ x_{k1}Y & \cdots & x_{kl}Y \end{bmatrix}. \tag{10}$$

Next, since inheritance relations in resource category hold whatever action may be, we can obtain the following adjacency matrix X_{RA_2}, which corresponds to

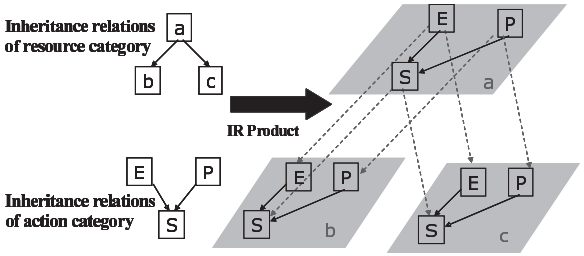


Fig. 3 An example of IR product between resource and action categories.

the solid line of Fig. 3:

$$X_{\mathrm{RA}_2} = I_{n_\mathrm{R}} \otimes A = \begin{bmatrix} A & & 0 \\ & \ddots & \\ 0 & & A \end{bmatrix}. \tag{11}$$

By Eqs. (9) and (11), we can obtain the adjacency matrix X_{RA} of the integrated inheritance relation of resource and action categories as follows:

$$X_{\mathrm{RA}} = X_{\mathrm{RA}_1} + X_{\mathrm{RA}_2} = R \otimes I_{n_\mathrm{A}} + I_{n_\mathrm{R}} \otimes A. \tag{12}$$

Example 1 Consider the integration of inheritance relations of resource and action categories shown in Fig. 3. The adjacency matrix R associated with the inheritance relation of resource category defined on $\{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$, and the adjacency matrix A for the action category defined on $\{\mathrm{E}, \mathrm{P}, \mathrm{S}\}$ are given, respectively by

$$R = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}. \tag{13}$$

In this case, the integrated inheritance relations are defined on the vertex set

$$\mathrm{RES} \times \mathrm{ACT} = \{\mathrm{aE}, \mathrm{aP}, \mathrm{aS}, \mathrm{bE}, \mathrm{bP}, \mathrm{bS}, \mathrm{cE}, \mathrm{cP}, \mathrm{cS}\}, \tag{14}$$

and corresponding X_{RA_1}, X_{RA_2}, and X_{RA} are obtained as follows:

$$X_{\mathrm{RA}_1} = \begin{bmatrix} 0 & I_3 & I_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad X_{\mathrm{RA}_2} = \begin{bmatrix} A & 0 & 0 \\ 0 & A & 0 \\ 0 & 0 & A \end{bmatrix}, \quad X_{\mathrm{RA}} = \begin{bmatrix} A & I_3 & I_3 \\ 0 & A & 0 \\ 0 & 0 & A \end{bmatrix}. \tag{15}$$

The matrix operation in Eq. (12) plays a central role for integrating inheritance relations of different categories. We refer to the matrix operation as *IR product*.

Definition 4 Let X and Y be the square matrices of order n and m, respectively. The IR product of X and Y, denoted by $X \otimes_{\mathrm{IR}} Y$, is defined as follows:

$$X \otimes_{\mathrm{IR}} Y := X \otimes I_m + I_n \otimes Y. \tag{16}$$

IR product is also applied for integrating inheritance relations of three categories. Let S be the adjacency matrix of the corresponding inheritance relations of subject category. Then, in a similar way to the above discussions, it is shown that the adjacency matrix X_{SRA} of the integrated inheritance relations of three categories are given by

$$X_{\mathrm{SRA}} = S \otimes_{\mathrm{IR}} X_{\mathrm{RA}} = S \otimes_{\mathrm{IR}} R \otimes_{\mathrm{IR}} A. \tag{17}$$

Using IR product, we can easily integrate all inheritance relations of three categories.

Remark 5 In matrix theory, the matrix operation in Eq. (16) is called Kronecker summation. In Eq. (17), one may think that the right-hand side of the equation should be $S \otimes_{\mathrm{IR}} (R \otimes_{\mathrm{IR}} A)$. However, Eq. (17) is correct since Kronecker product is associative, and thus, IR product is also associative, that is, $(S \otimes_{\mathrm{IR}} R) \otimes_{\mathrm{IR}} A = S \otimes_{\mathrm{IR}} (R \otimes_{\mathrm{IR}} A)$.

Remark 6 The graph obtained by *IR Product* is uniquely determined. By this, together with Remark 3 and 4, it is concluded that the graph obtained by our algorithm is uniquely determined.

Here, we have to check whether the integrated inheritance relation includes circuits or redundant edges. Concerning these points, the following theorems hold:

Theorem 4 Let G_a and G_b be graphs with adjacency matrices A and B, respectively. If G_a and G_b do not include circuits, the graph associated with the adjacency matrix $A \otimes_{\mathrm{IR}} B$ does not include circuits.

Theorem 5 Let G_a and G_b be graphs with adjacency matrices A and B, respectively. If G_a and G_b do not include circuits and redundant edges, the graph associated with the adjacency matrix $A \otimes_{\mathrm{IR}} B$ does not include redundant edges.

The proofs of Theorem 4 and Theorem 5 are given in Appendix A and B, respectively. Combining the above theorems, the following corollary is obtained:

Corollary 1 Let G_a and G_b be graphs with adjacency matrices A and B, respectively. If both G_a and G_b include neither circuits nor redundant edges, the graph associated with the adjacency matrix $A \otimes_{\mathrm{IR}} B$ include neither circuits nor redundant edges.

By this corollary, it is assured that the inheritance relation obtained by IR product does not have circuits and redundant edges provided that inheritance relations of all categories do not include circuits and redundant edges.

5. Conclusion

In this paper, under the assumptions that inheritance relations in a single category are independent from other categories, we have presented a matrix-based algorithm for integrating inheritance relations of access rights for generating integrated access control policies. By applying our algorithm, we can generate integrated inheritance relations automatically, and the results can be applied to centralized access control management systems, such as Microsoft Windows RMS, Adobe LiveCycle Rights Management ES, and PolicyComputing, where administrators have to prepare integrated inheritance relations manually in the current framework.

One of the most significant problem is that our algorithm will stop if an administrator thinks that the conflicts caused by the integration yield serious problems. Therefore, we have to improve the algorithm so as to preclude the inheritance relations which cause conflicts, and to continue the integration process without the inheritance relations.

Our proposed method will be especially useful for unifying management of access control systems which possess various inheritance relations satisfying independency assumption. However, we cannot apply the algorithm when we want to set up complex access control policies which do not satisfy the independency assumption.

In such a situation, Role-Based Access Control (RBAC) model [7),8)] will be useful. Using RBAC model, administrators can establish policies in detail. However, RBAC model can deal with inheritance relations in subject category only, and they have to address resources and actions individually. Therefore, it is desirable to use our proposed method together with RBAC model in a mutually complementary manner. In this case, we have to check whether the policies established by RBAC model and the policies generated by our proposed method do not conflict by using policy conflict detection tools [2),10),11),17)].

Acknowledgments This work was supported in part by a Grant-in-Aid for scientific research from the Japan Society for the Promotion of Science.

References

1) Adobe Systems: Adobe LiveCycle Rights Management ES, Adobe Systems (online), http://www.adobe.com/products/server/policy/ (accessed 2008-12-24).
2) Agrawal, D., Giles, J., Lee, K. and Lobo, J.: Policy Ratification, *Proc. 6th IEEE International Workshop on Policies for Distributed Systems and Networks*, pp.223–232 (2005).
3) Berge, C.: *The Theory of Graphs and its Applications*, John Wiley & Sons, Inc. (1962).
4) Bonatti, P., Sapino, M.L. and Subrahmanian, V.S.: Merging Heterogeneous Security Orderings, *Proc. European Symposium on Research in Computer Security*, pp.183–197 (1996).
5) Christofides, N.: *Graph Theory: An Algorithmic Approach*, Academic Press (1975).
6) Dawson, S., Qian, S. and Samarati, P.: Providing Security and Interoperation of Heterogeneous System, *Distributed and Parallel Databases*, Vol.8, No.1, pp.119–145 (2000).
7) Ferraiolo, D. and Kuhn, D.: Role Based Access Control, *Proc. 5th National Computer Security Conference*, pp.554–563 (1992).
8) Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R. and Chandramouli, R.: Proposed NIST Standard for Role-Based Access Control, *ACM Trans. Information and System Security*, Vol.4, No.3, pp.224–274 (2001).
9) Gong, L. and Qian, X.: Computational Issues in Secure Interoperation, *IEEE Trans. Software Engineering*, Vol.22, No.1, pp.43–52 (1996).
10) Kamoda, H., Kono, K., Ito, Y. and Babaguchi, N.: Access Control Policy Inconsistency Check Using Model Checker (in Japanese), *The Special Interest Group Notes of IPSJ*, pp.441–446 (2006).
11) Kamoda, H., Yamaoka, M., Matsuda, S., Broda, K. and Sloman, M.: Policy Conflict Analysis Using Free Variable Tableaux for Access Control in Web Services Environments, *Proc. 13th ACM Conference on Computer and Communications Security*, pp.134–143 (2006).
12) Koch, M., Mancini, L. and Parisi-Presicce, F.: On the Specification and Evolution of Access Control Policies, *Proc. 6th ACM Symposium on Access Control Models and Technologies*, pp.121–130 (2001).
13) Mazzoleni, P., Bertino, E. and Crispo, B.: XACML Policy Integration Algorithms,

Proc. 11th ACM Symposium on Access Control Models and Technologies, pp.219–227 (2006).
14) Microsoft: Windows Server 2003 Rights Management Services, Microsoft (online), http://www.microsoft.com/windowsserver2003/technologies/rightsmgmt/default.mspx (accessed 2008-12-24).
15) OASIS: *Core and Hierarchical Role Based Access Control* (*RBAC*) *Profile of XACML Version 2.0*, OASIS Standard (2005).
16) OASIS: *eXtensible Access Control Markup Language* (*XACML*) *Version 2.0*, OASIS Standard (2005).
17) Shafiq, B., Joshi, J.B.D., Bertino, E. and Ghafoor, A.: Secure Inteoperation in a Multidomain Environment Employing RBAC Policies, *IEEE Trans. Knowledge and Data Engineering*, Vol.17, No.11, pp.1557–1577 (2005).
18) Sugano, M., Tanaka, S., Sakata, Y., Oguma, K. and Shiratori, N.: Application and Implementation of Policy Control Method "PolicyComputing" in Computer Networks (Special Issue on Multimedia Network System) (in Japanese), *Trans. Information Processing Society of Japan*, Vol.42, No.2, pp.126–137 (2001).

Appendix

A.1 Proof of Theorem 4

Let A and B be square matrices of order n and order m, respectively. Since the graphs associated with A and B do not include circuits, we obtain the following equations by Theorem 1.

$$A^l = 0 \ \ (\forall l \geq n), \quad B^k = 0 \ \ (\forall k \geq m). \tag{18}$$

In order to prove Theorem 4, it suffices to show that $(A \otimes_{\mathrm{IR}} B)^{mn} = 0$. To this end, we introduce the following lemma, which is derived by simple calculations.

Lemma 2

$$(I_m \otimes A)^k = I_m \otimes A^k, \tag{19}$$
$$(B \otimes I_n)^k = B^k \otimes I_n, \tag{20}$$
$$(A \otimes I_m) \cdot (I_n \otimes B) = (I_n \otimes B) \cdot (A \otimes I_m), \tag{21}$$
$$(A \otimes_{\mathrm{IR}} B)^k = \sum_{l=0}^{k} (A \otimes I_m)^l \cdot (I_n \otimes B)^{k-l}. \tag{22}$$

By Lemma 2, the following equations are obtained.

$$\bar{A}^l = 0 \ \ (\forall l \geq n), \quad \bar{B}^k = 0 \ \ (\forall k \geq m), \tag{23}$$
$$(A \otimes_{\mathrm{IR}} B)^{mn} = \sum_{l=0}^{mn} \bar{A}^l \bar{B}^{mn-l}, \tag{24}$$

where $\bar{A} = A \otimes I_m$ and $\bar{B} = I_n \otimes B$. Here, note that $mn - n + 1 \geq m$ since $(m-1)(n-1) \geq 0$. Hence, from Eq. (18), we obtain:

$$(A \otimes_{\mathrm{IR}} B)^{mn} = \sum_{l=0}^{n-1} \bar{A}^l \bar{B}^{mn-l} = \sum_{k=mn-n+1}^{mn} \bar{A}^{mn-k} \bar{B}^k = 0. \tag{25}$$

A.2 Proof of Theorem 5

Let $A = [a_{ij}]$ and $B = [b_{ij}]$ be square matrices of order n and order m, respectively. In this proof, $a_{ij}^{(k)}$ and $b_{ij}^{(k)}$ denote the (i,j)-element of A^k and B^k, respectively.

Since the graphs associated with A and B do not include circuits and redundant edges, the following equations are obtained from Theorem 1 and Theorem 2:

$$a_{ii}^{(k)} = 0 \ \ (1 \leq i \leq n, \ k \geq 1), \tag{26}$$
$$b_{ii}^{(k)} = 0 \ \ (1 \leq i \leq m, \ k \geq 1). \tag{27}$$

By Theorem 3, we obtain:

$$\text{if } \ a_{ij} = 1 \ (i \neq j) \ \text{ then } \ a_{ij}^{(k)} = 0 \ \ (k \geq 2), \tag{28}$$
$$\text{if } \ b_{ij} = 1 \ (i \neq j) \ \text{ then } \ b_{ij}^{(k)} = 0 \ \ (k \geq 2). \tag{29}$$

By Lemma 2, $(A \otimes_{\mathrm{IR}} B)^k$ $(k \geq 2)$ is calculated as follows:

$$\begin{aligned}(A \otimes_{\mathrm{IR}} B)^k &= \sum_{l=0}^{k} (A \otimes I_m)^l (I_n \otimes B)^{k-l} = \sum_{l=0}^{k} (A^l \otimes I_m)(I_n \otimes B^{k-l}) \\ &= \sum_{l=0}^{k} \begin{bmatrix} a_{11}^{(l)} I_m & \cdots & a_{1n}^{(l)} I_m \\ \vdots & \ddots & \vdots \\ a_{n1}^{(l)} I_m & \cdots & a_{nn}^{(l)} I_m \end{bmatrix} \times \begin{bmatrix} B^{k-l} & & 0 \\ & \ddots & \\ 0 & & B^{k-l} \end{bmatrix} \\ &= \sum_{l=0}^{k} \begin{bmatrix} a_{11}^{(l)} B^{k-l} & \cdots & a_{1n}^{(l)} B^{k-l} \\ \vdots & \ddots & \vdots \\ a_{n1}^{(l)} B^{k-l} & \cdots & a_{nn}^{(l)} B^{k-l} \end{bmatrix}.\end{aligned}$$

On the other hand, $(A \otimes_{\mathrm{IR}} B)$ is calculated as

$$(A \otimes_{\mathrm{IR}} B) = (A \otimes I_m) + (I_n \otimes B)$$

$$= \begin{bmatrix} B & a_{12}I_m & \cdots & a_{1n}I_m \\ a_{21}I_m & B & & \vdots \\ \vdots & & \ddots & a_{n-1n}I_m \\ a_{n1}I_m & \cdots & a_{nn-1}I_m & B \end{bmatrix}.$$

Therefore, in order to prove Theorem 5, it suffices to show the following propositions:

(i) if $b_{ij} = 1$ $(i \neq j)$, then the (i, j) element of $\sum_{l=0}^{k} a_{pp}^{(l)} B^{k-l}$ is equal to zero for $p = 1, \cdots n$ and $k \geq 2$.

(ii) if $a_{ij} = 1$ $(i \neq j)$, then the diagonal elements of $\sum_{l=0}^{k} a_{ij}^{(l)} B^{k-l}$ are equal to zero for $k \geq 2$.

Concerning proposition (i), we obtain $\sum_{l=0}^{k} a_{pp}^{(l)} B^{k-l} = B^k$ from Eq. (26), since for every p, $a_{pp}^{(l)} = 1$ if and only if $l = 0$. From Eq. (29), when $b_{ij} = 1 (i \neq j)$, the (i, j) element of B^k is equal to zero for $k \geq 2$. Therefore, we can conclude that proposition (i) is true.

Concerning proposition (ii), we obtain $\sum_{l=0}^{k} a_{ij}^{(l)} B^{k-l} = B^{k-1}$ from Eq. (28), since for $i \neq j$, $a_{ij}^{(l)} = 1$ if and only if $l = 1$. From Eq. (27), the diagonal elements of B^{k-1} are equal to zero for $k \geq 2$. Therefore, we can conclude that proposition (ii) is also true.

This completes the proof.

(Received December 24, 2008)
(Accepted September 11, 2009)
(Released December 9, 2009)

Kazuhiro Kono received the B.E. and M.E. degrees in communication engineering from Osaka University, Japan, in 2005 and 2007, respectively. He is currently pursuing the Ph.D. degree in engineering at Osaka University. He is a student member of IPSJ, IEICE, IEEE, and ACM.

Yoshimichi Ito received the B.E. and M.E. degrees in electrical engineering from Kyoto University, Japan, in 1990 and 1992, respectively. He is currently an assistant professor in the Department of Information and Communications Technology, Osaka University. His research interests include system control theory, digital signal processing, and network security. He is a member of IEICE, IEEJ, ISCIE, and SICE.

Akihito Aoyama received his M.S. degree from Tokyo University of Science in 2001. He has been working in NTT DATA Corp. since 2001. His research interests include network and security.

Hiroaki Kamoda received his M.S. degree from Chiba University in 2000. He has been working in NTT DATA Corp. since 2000. From 2003 to 2004 he was a visiting researcher at FOKUS, Germany, and from 2004 to 2005 he was a visiting researcher at Imperial College London, U.K. He has been engaged in research in the areas of network security and software engineering. He is a member of IPSJ and SPM.

Noboru Babaguchi received the B.E., M.E., and Ph.D. degrees in communication engineering from Osaka University, Japan, in 1979, 1981, and 1984, respectively. He is currently a professor in the Department of Information and Communications Technology, Osaka University. From 1996 to 1997, he was a visiting scholar at the University of California, San Diego. His research interests include multimedia computing and artificial intelligence. He is a fellow of IEICE, and a member of IPSJ, ITE, IEEE, ACM, and JSAI.