PAPER Special Section on Foundations of Computer Science

Extending a Role Graph for Role-Based Access Control

Yoshiharu ASAKURA[†,††a)] ***and*** **Yukikazu NAKAMOTO**[††]**,** ***Members***

SUMMARY Role-based access control (RBAC) is widely used as an access control mechanism in various computer systems. Since an organization's lines of authority influence the authorized privileges of jobs, roles also form a hierarchical structure. A role graph is a model that represents role hierarchies and is suitable for the runtime phase of RBAC deployment. Since a role graph cannot take various forms for given roles and cannot handle abstraction of roles well, however, it is not suitable for the design phase of RBAC deployment. Hence, an extended role graph, which can take a more flexible form than that of a role graph, is proposed. The extended role graph improves diversity and clarifies abstraction of roles, making it suitable for the design phase. An equivalent transformation algorithm (ETA), for transforming an extended role graph into an equivalent role graph, is also proposed. Using the ETA, system administrators can deploy efficiently RBAC by using an extended role graph in the design phase and a standard role graph in the runtime phase.

key words: *RBAC, role graph, transformation algorithm, equivalence*

1. Introduction

In many computer systems, such as devices or databases, users perform work by accessing the objects, such as files or tables, provided by these systems. A subject (e.g., a user or a process) performs multiple operations (e.g., read and write) on the objects in these systems. Because these objects can contain user information and confidential data, they must be protected against malicious subjects. Access control, which limits the capability of subjects to perform operations on objects, is useful for protecting such data. As an example, discretionary access control is widely used in operating systems such as Linux [*] or UNIX. In the case of file access in Linux, a subject is a process, an object is a file, and the operations are read, write, and execute. An owner of a file sets file permissions (e.g., 644 or 755) for the owner, a group, and other users.

A pair consisting of an object and a set of operations is called a privilege. If a privilege is granted to a subject, that subject can perform any operation in the operation set on the object. To implement proper access control in a system, it is important to grant appropriate privileges to subjects and to manage the associations between subjects and privilege sets. For system administrators, however, this is costly work.

Role-based access control (RBAC) [1] can be used to decrease the cost of granting privileges to subjects. Several models are used in RBAC, and some of them form hierarchical structures. A role graph [2], [3] is one such hierarchical structure. When administrators deploy RBAC in a computer system, there are two phases: a design phase, and a runtime phase. Because the form of a role graph is not redundant for a given role set, by definition, a role graph is suitable for the runtime phase. In a role graph, however, there is no diversity of role hierarchies, because of the role graph's uniqueness. Since administrators cannot design role hierarchies as they desire, a role graph is not suitable for the design phase.

In this paper, to enable administrators to design diverse role hierarchies, we propose an *extended role graph* by extending the definition of a role graph. We also clear up the conditions that an extended role graph and a standard role graph are equivalent and propose an equivalent transformation algorithm (ETA) that transforms an extended role graph into an equivalent role graph. Our contribution in this paper can be summarized in the following two points:

1. Improving diversity of role hierarchies:
   We improve the diversity of role hierarchies by defining an extended role graph, which can take a more flexible form than that of a role graph.
2. Simplifying deployment of RBAC:
   We suppose that administrators use an extended role graph in the design phase and a role graph in the runtime phase, in order to best use the advantages of both graphs. Administrators can design role hierarchies by using an extended role graph and then transform it into an equivalent role graph for the runtime phase. The ETA enables this transformation.

The rest of the paper is organized as follows. We review the background and related works in Sect. 2 and describe the role graph in Sect. 3. Then, we define the extended role graph and the ETA in Sects. 4 and 5, respectively. We discuss the effectiveness of the extended role graph and the ETA in Sect. 6. Finally, we give conclusions and remarks and mention future works in Sect. 7.

Manuscript received March 28, 2008.
Manuscript revised July 2, 2008.

[†]The author is with System Platforms Research Laboratories, NEC Corporation, Kawasaki-shi, 211–8666 Japan.
[††]The authors are with the Graduate School of Applied Informatics, University of Hyogo, Kobe-shi, 650–0044 Japan.
a) E-mail: asakura@cd.jp.nec.com

DOI: 10.1587/transinf.E92.D.211

[*]All trademarks and registered trademarks referenced herein are the properties of their respective owners.

2. Motivation

2.1 Background of Our Work

In RBAC [1], a role is a unit that associates users [†] with privileges and corresponds to a job in an organization. Privileges are associated with a role and users are assigned to a role as shown in Fig. 1. A user assigned to a role can use privileges associated with the role. Administrators have only to assign roles to a user to grant privileges to the user. For example, a user assigned to the Programmer role in Table 1 can use the *use_compiler* privilege. Since an organization's lines of authority influence the authorized privileges of jobs, roles also form a hierarchical structure. $RBAC_1$ [1] and Hierarchical RBAC [4] define role hierarchies.

A role graph [2], [3] is a model of role hierarchies. It is a directed acyclic graph showing hierarchies based on the privilege inclusion relationship [††] among roles. A senior role inherits privileges associated with junior roles. For example, Table 1 lists four roles, ProjectMember, Programmer, NoviceTester, and ExpertTester, and the associated privileges of each role. Since the associated privilege set of ExpertTester includes the associated privilege sets of Programmer and NoviceTester, these roles form a hierarchy in a role graph, as shown in Fig. 2 [†††].

2.2 Limitations of a Role Graph

Deployment of RBAC consists of a design phase and a runtime phase. In the design phase, administrators define role hierarchies and associate privileges with roles. In the runtime phase, they assign roles to users according to the users' job functions and implement access control on a computer system. The form of a role graph is unique for a given role set, because it is a transitive reduction and has no redundant edges and no redundant privileges [2]. Therefore, a role graph is suitable for the runtime phase. Because there is no diversity of role hierarchies for a given role set and abstraction of roles is not well defined in a role graph, however, it is not suitable for the design phase. We give two examples here to illustrate these problems.

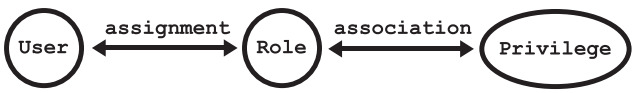

Fig. 1 Assignment and association in RBAC.

Table 1 Examples of privileges associated with roles.

Role	Associated privileges
ProjectMember	read_file, write_file
Programmer	read_file, write_file, use_compiler
NoviceTester	read_file, write_file, use_profiler
ExpertTester	read_file, write_file, use_compiler, use_profiler

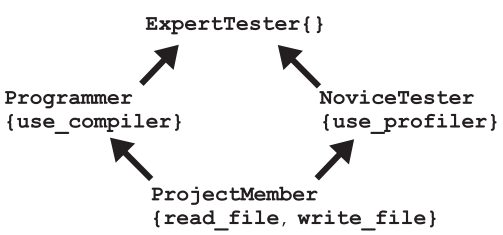

Fig. 2 Allowed role hierarchy.

Example 1: Suppose hierarchies are formed by the four roles, ProjectMember, Programmer, NoviceTester, and ExpertTester, listed in Table 1. From the definition of a role graph, the role hierarchy shown in Fig. 2 is the only one allowed in a role graph. The role hierarchies shown in Fig. 3 are not allowed in a role graph, even if administrators wanted to design such role hierarchies. In other words, a role graph has no flexibility for a given role set, and it is difficult for administrators to design role hierarchies as they desire.

Example 2: To clarify the common features of specific roles, a virtual role [5] can be used in a role graph, just as an abstract class clarifies the common features of specific classes in object-oriented design. Because no user can be assigned to a virtual role in user-role assignment, virtual roles intrinsically become useless in the runtime phase. To reduce consumption of system resources, administrators can delete virtual roles in the runtime phase.

To verify whether a role graph changes or not after deleting virtual roles, we need equivalence definitions. Suppose that a role graph includes a virtual role, Pro-

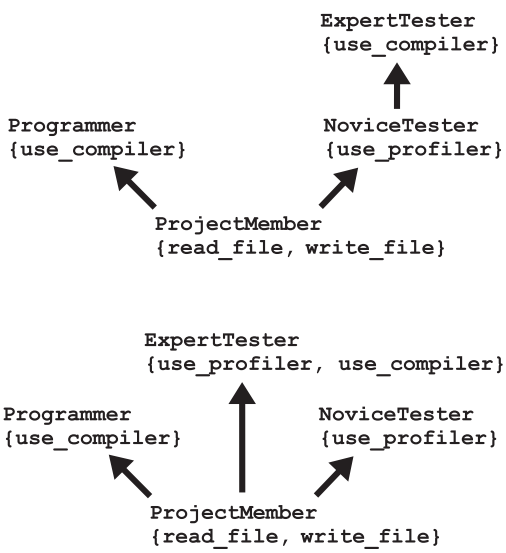

Fig. 3 Disallowed role hierarchies.

[†]A user is generally used as a subject in RBAC.

[††]A stricter definition is given in [2] and discussed here in Sect. 3.

[†††]In a role graph, we do not show privileges inherited from junior roles.

grammer. The role graph should maintain equivalence before and after administrators remove the Programmer role. It is not clear, however, how administrators can remove the Programmer role while preserving equivalence.

In this paper, we propose an extended role graph that can take various forms and clarifies a virtual role. We clear up the conditions that an extended role graph and a standard role graph are equivalent. We also propose an equivalent transformation algorithm (ETA) that transforms an extended role graph into an equivalent role graph, which includes no virtual roles. The ETA enables administrators to deploy RBAC efficiently by using an extended role graph in the design phase and a standard role graph in the runtime phase.

2.3 Related Work

RBAC [1] and the role graph [2], [3] are the bases of our work. As described above, the form of a role graph is unique for a given role set. In contrast, our extended role graph can flexibly take various forms for a given role set. Therefore, administrators can design desired role hierarchies more easily with an extended role graph than with a role graph.

In a role graph, a virtual role [5] is one such that no user can be assigned to this role in user-role assignment and is not well defined. Here, an extended role graph clarifies a virtual role. We also define equivalent transformation manipulations (ETMs) for transforming an extended role graph while preserving equivalence. ETMs include the manipulation for deleting virtual roles.

RBAC can be deployed not only in individual systems but also in distributed systems. Enterprise Security Management Systems (ESMS) are examples of such distributed systems. RBAC deployed in an ESMS is called Enterprise RBAC (ERBAC) [6]–[8]. Enterprise Roles in ERBAC also form role hierarchies over distributed systems, and a role graph can be used in such role hierarchies [9]. An extended role graph can also be used.

3. Role Graph

Nyanchama and Osborn proposed the role graph in [2], [3]. A role graph is a directed acyclic graph in which the nodes represent roles and the edges represent the is-junior relationships. The is-junior relationship (denoted $\rightarrow$) is defined between two roles. For any two roles r_1 and r_2, $r_1 \rightarrow r_2$ if and only if $r_1.rpset \subset r_2.rpset$ [†] holds. Here, $r.rpset$ means a set of privileges of r.

r_1 is said to be junior to r_2 and to be a junior role of r_2. Let $Juniors(r)$ be the role set of every junior role of a role r. In contrast, r_2 is said to be senior to r_1 and to be a senior role of r_1. Let $Seniors(r)$ be the role set of every senior role of a role r. A role graph is represented by a pair of (RS, ES), where RS represents a role set and ES represents a set of the is-junior relationship (in other words, ES is a set of edges). Privileges are divided into two types in the is-junior relationship [2]. The first type is a *direct privilege*. The direct privileges of a role r are those not assigned to any junior roles of r. Let $Direct(r)$ be the set of direct privileges of r. The second type is an *effective privilege*. The effective privileges of a role r are the union of $Direct(r)$ and effective privileges of every junior role of r. Here, $r.rpset$ represents the set of effective privileges of r.

Nyanchama and Osborn showed that a role graph has the following properties [3], hereafter referred to as the Role Graph Properties:

1. There is a single MaxRole.
2. There is a single MinRole.
3. The Role Graph is acyclic.
4. There is a path from $MinRole$ to every role r_i.
5. There is a path from every role r_i to $MaxRole$.
6. For any two roles r_i and r_j, if $r_i.rpset \subset r_j.rpset$, then there must be a path from r_i to r_j.

Here, the MaxRole has all of the privileges of every role in the graph, while the MinRole role has only those privileges common to every role in the graph. The privilege set of MinRole might thus be $\emptyset$. Nyanchama and Osborn showed another property of a role graph, namely, well-formedness [2]. A well-formed role graph has no redundant edges and no redundant privileges. Since it is a directed acyclic graph and a transitive reduction, the form of a role graph is unique for a given role set [10]. Algorithms for manipulating a role graph were also proposed in [3]. These algorithms were for role addition, role deletion, role partition, privilege addition, privilege deletion, edge insertion, and edge deletion. These algorithms preserve the Role Graph Properties and the well-formedness of a role graph.

4. Extended Role Graph

In this section, we discuss the extended role graph and the equivalence of extended role graphs. An extended role graph eases the Role Graph Properties and well-formedness of a role graph. Therefore, the form of an extended role graph is not unique for a given role set. In other words, equivalent extended role graphs can take various forms. Administrators can thus flexibly manipulate an extended role graph as they desire. In addition, an extended role graph clarifies abstraction of roles. We can consider a standard role graph to be a type of extended role graph satisfying stricter properties and conditions. The algorithms in [3] can also be applied to manipulate an extended role graph (except for preserving well-formedness).

4.1 Definitions

We introduce several definitions to facilitate our discussion of the extended role graph. We redefine a direct privilege and clarify a virtual role in order to define the extended role

[†]Roles with the same privileges are regarded as one role [3].

graph and describe the equivalence of extended role graphs. First, we redefine a direct privilege to allow the existence of redundant privileges.

Definition 1 (Direct privilege (redefined)): Direct privileges are associated with a role r. Unlike in the original definition, a direct privilege p of a role r can also be inherited from the junior roles of r. If p is also inherited, it is a redundant privilege for r.

Next, we clarify a virtual role in our extended role graph.

Definition 2 (Virtual role): A role r is said to be a virtual role if the number of users assignable to r is 0. The direct privilege set of a virtual role can be $\emptyset$.

In other words, a virtual role is the same as a role except that the number of assignable users is 0 and its direct privilege set can be $\emptyset$. A virtual role is only used for defining the common privileges for its senior roles, making it useful in the design phase of RBAC. It is not useful, however, in the runtime phase. Therefore, virtual roles should be removed from an extended role graph before the runtime phase. The concept of a role set should be extended in order to include virtual roles. Therefore, we define an extended role set as follows.

Definition 3 (Extended role set): A role set ERS is an extended role set if it includes at least zero virtual role. If ERS includes no virtual roles, it is simply a role set.

Now, we can define the extended role graph.

Definition 4 (Extended role graph): A role graph ERG is an extended role graph if it satisfies any of the following conditions:

C1: The role set of the role graph is an extended role set.
C2: The role graph does not satisfy Property 6 of the Role Graph Properties.
C3: The role graph is not well-formed.

Let $ERG = (ERS, ES)$ denote an extended role graph, where ERS represents an extended role set and ES represents a set of the is-junior relationships [†].

4.2 Equivalence Definitions

We next introduce several equivalence definitions. First, we clarify the condition whereby two roles are equivalent.

Definition 5 (Role equivalence): For any two roles r_i and r_j, r_i is equivalent to r_j (denoted $r_i = r_j$) if and only if $r_i.rpset = r_j.rpset$.

Roles are considered equivalent if their privilege sets are equivalent, even if the names are different. Next, we define the inclusion of role sets.

Definition 6 (Inclusion of role sets): For any two role sets RS_i and RS_j, RS_i is included in RS_j (denoted $RS_i \subseteq RS_j$) if and only if for any role $r_i \in RS_i$, there exists $r_j \in RS_j$ such that $r_i = r_j$.

Now, we define the equivalence of extended role sets and the equivalence of extended role graphs.

Definition 7 (Equivalence of extended role sets): For any two extended role sets ERS_1 and ERS_2, ERS_1 is equivalent to ERS_2 (denoted $ERS_1 = ERS_2$) if and only if $(ERS_1 - VRS_1) \subseteq (ERS_2 - VRS_2) \wedge (ERS_1 - VRS_1) \supseteq (ERS_2 - VRS_2)$. Here, VRS_1 and VRS_2 are the virtual role sets included in ERS_1 and ERS_2, respectively.

Definition 8 (Equivalence of extended role graphs): For any two extended role graphs $ERG_1 = (ERS_1, ES_1)$ and $ERG_2 = (ERS_2, ES_2)$, ERG_1 is equivalent to ERG_2 (denoted $ERG_1 = ERG_2$) if and only if $ERS_1 = ERS_2$.

Since a role graph can be regarded as a type of extended role graph, we can replace an extended role graph with a role graph in Defs. 7 and 8.

4.3 Extended Role Graph Diversity

As discussed in Sect. 3, the form of a role graph is unique for a given role set. In contrast, the form of an extended role graph is not unique for a given extended role set. We show this property in the following theorem.

Theorem 1 (Extended role graph diversity): There exist two extended role graphs $ERG_1 = (ERS_1, ES_1)$ and $ERG_2 = (ERS_2, ES_2)$ such that $ERS_1 = ERS_2 \wedge ES_1 \neq ES_2$ holds if these two extended role graphs satisfy any condition in Def. 4.

Proof : For any two extended role graphs satisfying any condition in Def. 4, we show examples of two extended role graphs $ERG_1 = (ERS_1, ES_1)$ and $ERG_2 = (ERS_2, ES_2)$ such that $ERS_1 = ERS_2 \wedge ES_1 \neq ES_2$ holds.

In the case of satisfying C1: Consider two extended role graphs $ERG_1 = (ERS_1, ES_1)$, such that $ERS_1 = \{MinRole, MaxRole, r_1, vr_1\}$, and $ERG_2 = (ERS_2, ES_2)$, such that $ERS_2 = \{MinRole, MaxRole, r_1, vr_1,$

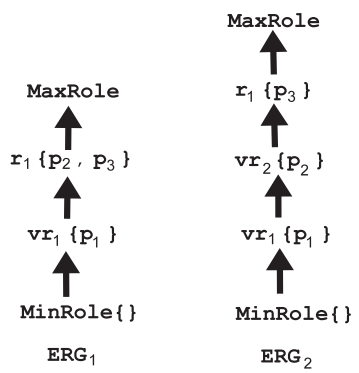

Fig. 4 Example of satisfying C1.

[†]Since the set of direct privileges of a virtual role can be $\emptyset$, an extended role set can include several roles with equivalent effective privileges. Therefore, for any virtual role r_i in an extended role graph, $r_i \rightarrow r_j$ means $r_i.rpset \subseteq r_j.rpset$.

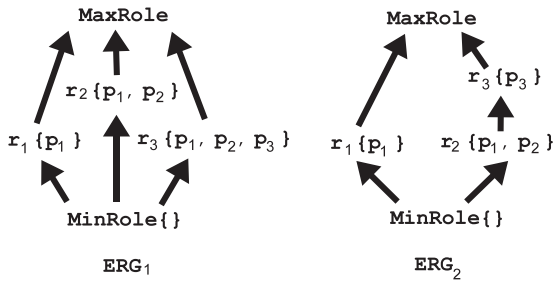

Fig. 5 Example of satisfying C2.

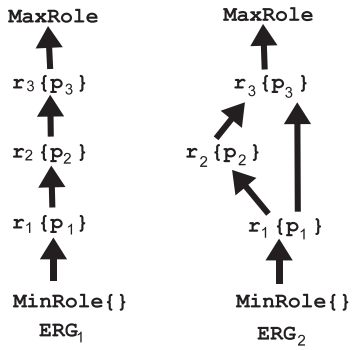

Fig. 6 Example of satisfying C3.

$vr_2\}$, as given in Fig. 4. Since $(ERS_1 - \{vr_1\}) \subseteq (ERS_2 - \{vr_1, vr_2\}) \wedge (ERS_1 - \{vr_1\}) \supseteq (ERS_2 - \{vr_1, vr_2\})$ holds, $ERS_1 = ERS_2$ holds from Def. 7. Moreover, $ES_1 \neq ES_2$ holds, as shown in Fig. 4. Therefore, there exist two extended role graphs such that $ERS_1 = ERS_2 \wedge ES_1 \neq ES_2$ holds.

In the case of satisfying C2: Consider two extended role graphs $ERG_1 = (ERS_1, ES_1)$ and $ERG_2 = (ERS_2, ES_2)$, as given in Fig. 5. $ERS_1 = ERS_2$ holds, since each role has identical privileges. Moreover, $ES_1 \neq ES_2$ holds, as shown in Fig. 5. Therefore, there exist two extended role graphs such that $ERS_1 = ERS_2 \wedge ES_1 \neq ES_2$ holds.

In the case of satisfying C3: Consider two extended role graphs $ERG_1 = (ERS_1, ES_1)$ and $ERG_2 = (ERS_2, ES_2)$, as given in Fig. 6. Then there exist two extended role graphs such that $ERS_1 = ERS_2 \wedge ES_1 \neq ES_2$ holds, in the same way discussed for the case of satisfying C2.

From the above discussion, $ERS_1 = ERS_2 \wedge ES_1 \neq ES_2$ holds if the two extended role graphs satisfy any condition in Def. 4. □

5. Equivalent Transformation Algorithm

The extended role graph defined in Sect. 4 improves diversity and clarifies abstraction of roles. Therefore, an extended role graph facilitates the design phase of RBAC. Since an extended role graph can have redundant edges and privileges, however, a standard role graph is more suitable than an extended role graph for the runtime phase. In fact, an extended role graph is not suitable for the runtime phase. One solution is for administrators to use an extended role graph in the design phase and transform it, after designing role hierarchies, into a standard role graph. This solution requires the role graph to be equivalent to the extended role graph.

In this section, we discuss the ETA for transforming an extended role graph into an equivalent role graph. First, we define equivalent transformation manipulations for the ETA. Next, we give the ETA itself. Finally, we show the complexity of the ETA. The ETA enables administrators to best use the advantages of both an extended role graph and a role graph, thus improving deployment of RBAC.

5.1 Equivalent Transformation Manipulations

Transformation manipulations include such operations as adding/deleting a role and an edge to/from a role graph and adding/deleting a privilege to/from a role's direct privilege set. For an extended role graph ERG_2 created by applying some transformation manipulations to an extended role graph ERG_1, these manipulations are equivalent transformation manipulations (ETMs) if $ERG_1 = ERG_2$ holds. We define ETMs in Theorem 2.

Theorem 2 (ETMs): The following five transformation manipulations are ETMs.

Privilege distribution (PD): This manipulation distributes a direct privilege to immediate senior roles. Given a virtual role vr, if there exists a direct privilege $cp \in Direct(vr)$, then we can delete cp from $Direct(vr)$ and add cp to every direct privilege set of the immediate senior roles of vr.

Virtual role deletion (VRD): This manipulation deletes a virtual role from an extended role graph. Given a virtual role vr, if $Direct(vr) = \emptyset$, then we can delete vr from the extended role graph. We can also add edges from every immediate junior role of vr to every immediate senior role of vr to maintain the paths between these roles.

Edge addition (EA): This manipulation adds an edge to an extended role graph. Given two roles r_i and r_j, we can add an edge (r_i, r_j) if $r_i.rpset \subseteq r_j.rpset$ holds.

Redundant edge deletion (RED): This manipulation deletes a redundant edge from an extended role graph. Given two roles r_i and r_j, we can delete an edge (r_i, r_j) if there exists a path such that $r_i \rightarrow r_1 \rightarrow \cdots \rightarrow r_k \rightarrow r_j$, $k \geq 1$.

Redundant privilege deletion (RPD): This manipulation deletes a redundant privilege from a role. Given a direct privilege rp of a role r, if there exists $jr \in IJuniors(r)$ such that $rp \in jr.rpset$ holds, then we can delete rp from $Direct(r)$. Here, $IJuniors(r)$ is the immediate junior role set of r.

Proof : Suppose that we transform an extended role graph $ERG_1 = (ERS_1, ES_1)$ into an extended role graph $ERG_2 =$

(ERS_2, ES_2), and that VRS_1 and VRS_2 are virtual role sets included in ERS_1 and ERS_2, respectively.

PD: Suppose that we distribute a direct privilege cp of a virtual role vr_1 in VRS_1 to the immediate senior roles, transforming vr_1 to vr_2 in VRS_2. For any immediate senior role sr_1 of vr_1, we add cp to $Direct(sr_1)$, so that sr_1 becomes sr_2 in ERS_2. Then, $cp \in sr_1.rpset$ holds, since sr_1 inherits cp from the immediate junior role vr_1, and $cp \in sr_2.rpset$ also holds. Thus, $sr_1 = sr_2$ holds. Since $(ERS_1 - VRS_1) \subseteq (ERS_2 - VRS_2) \wedge (ERS_1 - VRS_1) \supseteq (ERS_2 - VRS_2)$ holds, $ERS_1 = ERS_2$ holds from Def. 7. Hence, from Def. 8, $ERG_1 = ERG_2$ holds.

VRD: Suppose that we delete a virtual role vr, which has no direct privilege, from ERS_1. That is, $VRS_1 - \{vr\} = VRS_2$ and $ERS_1 - \{vr\} = ERS_2$ hold. $ERS_1 = ERS_2$ holds from Def. 7, and hence, from Def. 8, $ERG_1 = ERG_2$ holds.

EA: Suppose that we add an edge (r_i, r_j) to ES_1. Since $r_i.rpset \subseteq r_j.rpset$ holds, $r_j.rpset$ does not change after adding the edge. $ERS_1 = ERS_2$ holds from Def. 7, and hence, from Def. 8, $ERG_1 = ERG_2$ holds.

RED: Suppose that we delete a redundant edge (r_i, r_j) from ES_1. Since there exists a path such that $r_i \rightarrow r_1 \rightarrow \cdots \rightarrow r_k \rightarrow r_j$, $k \geq 1$, $r_j.rpset$ does not change after deleting the redundant edge. $ERS_1 = ERS_2$ holds from Def. 7, and hence, from Def. 8, $ERG_1 = ERG_2$ holds.

RPD: Suppose that we delete a redundant privilege rp from a role r_1 in ERS_1, so that r_1 becomes r_2 in ERS_2. Since there exists $jr \in IJuniors(r_1)$ $(= IJuniors(r_2))$ such that $rp \in jr.rpset$ holds, $r_1.rpset = r_2.rpset$ holds. Since $r_1 = r_2$ holds from Def. 5, $ERS_1 = ERS_2$ also holds. Hence, from Def. 8, $ERG_1 = ERG_2$ holds.

□

5.2 Equivalent Transformation Algorithm

The ETA is composed of the steps given in Theorem 3, which guarantees that any extended role graph can be transformed into an equivalent role graph.

Theorem 3 (Transformation): Any extended role graph can be transformed into an equivalent role graph.

Proof : The following steps form the transformation from an extended role graph ERG into an equivalent role graph RG. We can find every (virtual) role in ERG with a depth-first search starting at $MinRole$.

Step 1: For every virtual role vr, distribute $Direct(vr)$ to every immediate senior role of vr by applying PD, obtaining ERG_1.

Step 2: Delete every virtual role vr by applying VRD, obtaining ERG_2.

Step 3: For any two roles r_1 and r_2, other than $MinRole$ and $MaxRole$, to which EA is applicable, add the edge (r_1, r_2) by applying EA, obtaining ERG_3.

Step 4: For any edge e to which RED is applicable, delete e by applying RED, obtaining ERG_4.

Step 5: For any role r, delete redundant privileges by applying RPD, obtaining ERG_5.

After Steps 1 and 2, ERG_2 does not satisfy C1 of Def. 4, because every virtual role has been deleted. After Step 3, ERG_3 does not satisfy C2 of Def. 4, because ERG_3 satisfies Property 6 of the Role Graph Properties. After Steps 4 and 5, ERG_5 does not satisfy C3 of Def. 4, because all redundant privileges and all redundant edges have been deleted. Therefore, ERG_5 does not satisfy any condition of Def. 4. Moreover, the (extended) role set (except for virtual roles) remains unchanged, since the transform manipulations used in the above steps are ETMs. Hence, $ERG = ERG_5 = RG$ holds, and therefore, any extended role graph can be transformed into an equivalent role graph by applying the above steps. □

We show an example of transformation from an extended role graph into an equivalent role graph in Fig. 7. The extended role graph (a) includes seven roles: $MinRole$, $MaxRole$, R_1, VR_2, R_3, R_4, and R_5. VR_2 is a virtual role. We transform the extended role graph (a) according to the steps given in Theorem 3. In Step 1, we distribute the privilege p_2 to R_4 and R_5, transforming (a) into (b). In Step 2, we delete the virtual role VR_2. Because of the virtual role deletion, we add the edges $(MinRole, R_4)$ and $(MinRole, R_5)$, transforming (b) into (c). In Step 3, we add the edge (R_3, R_4), transforming (c) into (d). In Step 4, we delete the redundant edges $(MinRole, R_4)$ and $(MinRole, R_5)$, transforming (d) into (e). Finally, in Step 5, we delete the redundant privilege p_3 of R_4, transforming (e) into (f), a role graph that is equivalent to (a).

5.3 Complexity

In this section, we show the complexity of the ETA. Here, we assume that an extended role graph is structured as follows.

- A role set is structured as a list.
- The incoming edge set and the outgoing edge set of a role r are represented as two adjacency lists. One adjacency list represents the reversed incoming edge set, namely, the edge set from r to every immediate junior role of r. The other adjacency list represents the outgoing edge set, namely, the edge set from r to every immediate senior role of r.
- The direct privilege set and the effective privilege set of a role r are represented as two lists sorted by some total order relation among privileges [†]. Let ω be the computational cost of comparison between two privileges.

First, we show the complexity of each ETM. Let n, e, and p be the number of roles, the number of edges, and the number of a type of privilege, respectively, in a role graph.

[†]For example, comparison between the character strings of privilege names is a candidate for a total order relation among privileges.

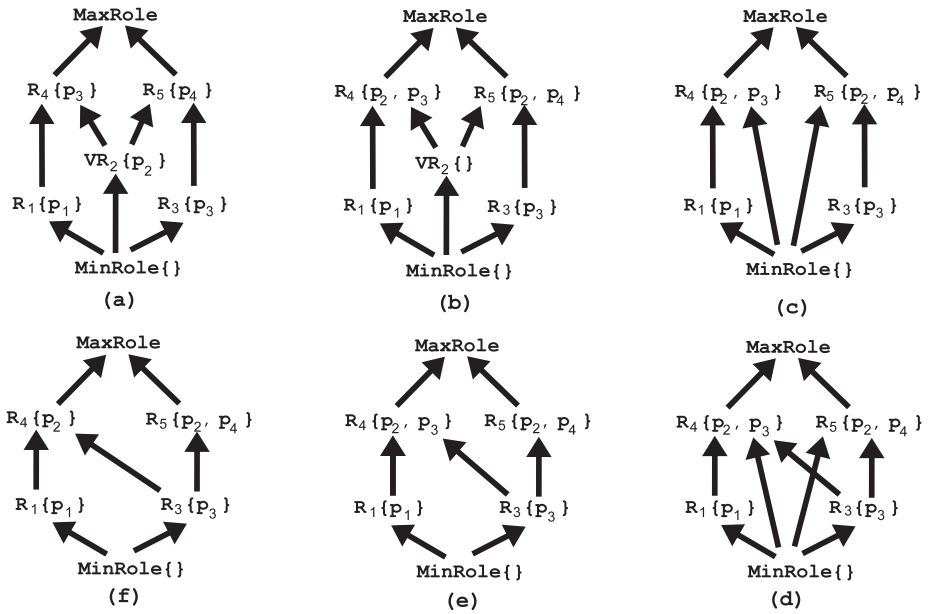

Fig. 7 Example of transformation.

PD: For any virtual role vr and any direct privilege cp, we take $O(\omega p)$ to decide whether cp is included in $Direct(vr)$ and to delete cp from $Direct(vr)$. We take $O(e)$ to visit every immediate senior role of vr. We take $O(\omega p)$ to add cp to the direct privilege set of an immediate senior role of vr. Therefore, PD takes $O(\omega p) + O(e)O(\omega p) = O(e\omega p)$.

VRD: For any virtual role vr, we take $O(1)$ to decide whether $Direct(vr) = \emptyset$ and $O(n + e)$ to delete vr. We take $O(e^3)$ to add edges from every immediate junior role of vr to every immediate senior role of vr. Therefore, VRD takes $O(n + e^3)$.

EA: For any two roles r_i and r_j, we take $O(\omega p)$ to decide whether $r_i.rpset \subseteq r_j.rpset$ holds. We take $O(e)$ to add an edge (r_i, r_j). Therefore, EA takes $O(e + \omega p)$.

RED: For any role r_i and r_j, we can decide whether there exists a path such that $r_i \to r_1 \to \cdots \to r_k \to r_j$, $k \geq 1$ by applying a depth-first search starting at r_i to the extended role graph obtained by removing an edge (r_i, r_j). Therefore, we take $O(n+e)$ to decide whether this path exists and $O(e)$ to delete the edge (r_i, r_j). Therefore, RED takes $O(n + e)$.

RPD: For any direct privilege rp of any role r, we take $O(n\omega p)$ to decide whether there exists $jr \in IJuniors(r)$ such that $rp \in jr.rpset$ holds. Therefore, RPD takes $O(n\omega p)$.

Next, we give the complexity of the ETA in the following theorem.

Theorem 4 (Complexity): If the number of roles, the number of edges, and the number of a type of privilege are n, e, and p, respectively, in an extended role graph ERG, then the complexity of the ETA for ERG is $O(ne(e^2 + \omega p) + n^2(e + \omega p^2))$.

Proof : We show the complexity of each step in Theorem 3.

Step 1: For each role r in ERG, if r is a virtual role, then we apply PD to r. Since PD takes $O(e\omega p)$, Step 1 takes $O(n)O(e\omega p) = O(ne\omega p)$.

Step 2: For each role r in ERG_1, if r is a virtual role, then we apply VRD to r. Since VRD takes $O(n + e^3)$, Step 2 takes $O(n)O(n + e^3) = O(n(n + e^3))$.

Step 3: For any two roles r_1 and r_2, we apply EA to r_1 and r_2. Since EA takes $O(e + \omega p)$, Step 3 takes $O(n^2)O(e + \omega p) = O(n^2(e + \omega p))$.

Step 4: In a depth-first search, we visit each role once and walk each edge once. For any edge e, we apply RED to e. Since visiting each role takes $O(n)$, walking every edge takes $O(e)$, and RED takes $O(n + e)$, Step 4 takes $O(n) + O(e)O(n + e) = O(e(n + e))$.

Step 5: For each role r, we apply RPD to every direct privilege of r. Since RPD takes $O(n\omega p)$, Step 5 takes $O(n)O(p)O(n\omega p) = O(n^2\omega p^2)$.

Therefore, the complexity of the ETA is $O(ne\omega p) + O(n(n + e^3)) + O(n^2(e + \omega p)) + O(e(n + e)) + O(n^2\omega p^2) = O(ne(e^2 + \omega p) + n^2(e + \omega p^2))$. □

6. Effectiveness

Here, we discuss the effectiveness of the extended role graph and the ETA.

6.1 Effectiveness of the Extended Role Graph

We examine the effectiveness of improving diversity and clarifying abstraction of roles by considering Examples 1 and 2 given in Sect. 2.

In Example 1, since Programmer.rpset ⊂ ExpertTester.rpset and NoviceTester.rpset ⊂ ExpertTester.rpset holds, ExpertTester should be a senior role of Programmer and NoviceTester in a role graph, as shown in Fig. 2. Since an extended role graph eases the Role Graph Properties, however, ExpertTester does not have to be a senior role of Programmer or NoviceTester in an extended role graph, as shown in Fig. 3. An extended role graph takes a flexible form in this manner. This gives an example of improving the diversity of role hierarchies.

By clarifying a virtual role, we clear up the conditions that an extended role graph and a role graph are equivalent. VRD preserves equivalence before and after deleting virtual roles, enabling administrators to remove virtual roles systematically. In Example 2, administrators can remove a virtual role Programmer by applying VRD to the extended role graph. An extended role graph clarifies abstraction of roles in this manner.

6.2 Effectiveness of the ETA

An extended role graph is suitable for the design phase of RBAC, as described in Sect. 5. The extended role graph, however, can have redundant edges and privileges and include virtual roles. Redundant edges and privileges are not useful in the runtime phase. Since users cannot be assigned to virtual roles, they also are not useful in the runtime phase. In contrast, a role graph is suitable for the runtime phase because it has no redundancy for a given role set. The ETA mediates between an extended role graph and a role graph. Since the complexity of the ETA is of polynomial order, the ETA can be implemented as a tool and automated. Therefore, administrators can easily use the ETA to build suitable role hierarchies in both the design phase and the runtime phase. The ETA thus improves the efficiency of RBAC deployment in this manner.

7. Conclusions and Remarks

In this paper, we have proposed an extended role graph that improves diversity and clarifies abstraction of roles for RBAC. We have also proposed an equivalent transformation algorithm (ETA), which transforms an extended role graph into an equivalent role graph. The extended role graph is suitable for the design phase of RBAC, while the role graph is suitable for the runtime phase. By using the ETA, system administrators can efficiently deploy RBAC, because they can best use the advantages of both graphs. Since the complexity of the ETA is of polynomial order, the ETA can be implemented as a tool for easy use by administrators.

On some points, the relationship between a role graph and an extended role graph is similar to that between a nondeterministic finite automaton (NFA) and a deterministic finite automaton (DFA). Since we have proposed the ETA in this paper, for any extended role graph for a given role set, there exists a role graph that includes the equivalent role set. This relationship is similar to the relationship, which for any NFA, there exists a DFA that accepts the same language. However, we have not yet described the reverse manipulations of the ETMs. Therefore, the converse relationship is not currently similar.

Administrators can use the algorithms given in [3] to manipulate an extended role graph. When they transform an extended role graph $ERG_1 = (ERS_1, ES_1)$ into another extended role graph $ERG_2 = (ERS_2, ES_2)$, there will be circumstances in which they would like to transform it so that $ERS_1 \subseteq ERS_2$ holds. The algorithms in [3], however, do not satisfy this need.

In future works, we will explore algorithms for manipulation that do satisfy $ERS_1 \subseteq ERS_2$, which could be applied in the above situation. Moreover, we will define the reverse manipulations of the ETMs. Then, we can show that the converse relationship is also similar.

References

[1] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman, "Role-based access control models," Computer, vol.29, no.2, pp.38–47, Feb. 1996.
[2] M. Nyanchama and S.L. Osborn, "Access rights administration in role-based security systems," Proc. IFIP WG11.3 Working Conference on Database Security VII, pp.37–56, Amsterdam, The Netherlands, Aug. 1994.
[3] M. Nyanchama and S.L. Osborn, "The role graph model and conflict of interest," ACM Trans. Information and System Security, vol.2, no.1, pp.3–33, Feb. 1999.
[4] D.F. Ferraiolo, R.S. Sandhu, S.I. Gavrila, D.R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," ACM Trans. Information and System Security, vol.4, no.3, pp.224–274, Aug. 2001.
[5] M. Petkovic and W. Jonker, eds., Security, Privacy, and Trust in Modern Data Management, Springer, 2007.
[6] A. Kern, M. Kuhlmann, A. Schaad, and J. Moffett, "Observations on the role life-cycle in the context of enterprise security management," Proc. 7th ACM Symposium on Access Control Models and Technologies, pp.43–51, California, USA, June 2002.
[7] A. Kern, "Advanced features for enterprise-wide role-based access control," Proc. 18th Annual Computer Security Applications Conference, pp.333–342, Nevada, USA, Dec. 2002.
[8] A. Kern, A. Schaad, and J. Moffett, "An administration concept for the enterprise role-based access control model," Proc. 8th ACM Symposium on Access Control Models and Technologies, pp.3–11, Como, Italy, June 2003.
[9] D.F. Ferraiolo, G.J. Ahn, R. Chandramouli, and S.I. Gavrila, "The role control center: Features and case studies," Proc. 8th ACM Symposium on Access Control Models and Technologies, pp.12–20, Como, Italy, June 2003.
[10] A.V. Aho, M.R. Garey, and J.D. Ullman, "The transitive reduction of a directed graph," SIAM J. Comput., vol.1, no.2, pp.131–137, June 1972.

Yoshiharu Asakura received an M.E. degree in 1998 from Osaka University; since then, he has worked for NEC Corporation. As of 2007, he is also a student at the Graduate School of Applied Informatics, University of Hyogo. His research interests include embedded systems, distributed systems, and information security technologies.

Yukikazu Nakamoto received M.E. and Ph.D. degrees from Osaka University in 1982 and 2000, respectively. From 1982 to 2004, he worked for NEC Corporation. In 2004, he joined the University of Hyogo and is currently a Professor of Applied Informatics. From 1990 to 1991, he was a Visiting Researcher at Cornell University. From 2003 to 2004, he was a Visiting Professor at The University of Electro-Communications. Since 2006, he has been a Designated Professor of the Center for Embedded Computing Systems, in the Graduate School of Information Science, Nagoya University. His research interests include real-time systems, distributed systems, mobile systems, and software development environments. He is a member of IPSJ, IEEE Computer Society, ACM, and USENIX.