Designing an Agent-Based RBAC System for Dynamic Security Policy

Wataru Yamazaki , Hironori Hiraishi, and Fumio Mizoguchi
Information Media Center
Tokyo University of Science
{yamazaki, hiraishi, mizo}@imc.tus.ac.jp

Abstract

Most practical applications have dynamic attributes, but conventional access control mechanisms have not addressed the problem sufficiently. In this paper, we discuss how to realize an access control system that enables us to manage dynamic security policies. Our proposed method is based on Role-Based Access Control (RBAC), and the agent decides access rights dynamically for the abstract role, which is defined by the role administrator statically using context-enabled rules and an inference engine. By defining rules using declarative representation (logic programming style), bidirectional queries can be realized for USER-ROLE-PERMISSION relationships. In this paper, we will demonstrate the usefulness of our proposed system by presenting our project management application and its access control system.

1. Introduction

Access control is important in all computer programs. Access control both defends against illegal access by malicious attackers and prevents honest users from making inappropriate access and possibly causing administrative errors. However, managing access control appropriately is generally a difficult problem for two reasons. First computer applications are getting more complex, and the numbers of control objects and users of these applications are becoming larger than ever. Second, many applications have more or less dynamic attributes, but defining all possible situations of these applications is difficult.

A modern approach to the first problem is applying Role-Based Access Control (RBAC) [1], in which the target of access control is a role, i.e., a set of access rights based on the user's usage pattern. In RBAC, users have associated roles, and roles have associated access rights. RBAC enables simpler management than conventional access control systems such as MAC and DAC in adding or eliminating users and adding or deleting access rights.

However, the second problem has not been fully discussed. The dynamic attributes here consist of things like access based on application states, access based on time, access based on place, exclusive access control, and delegation of access rights. When we think of recent large-scale applications and distributed applications, we have to consider access control and its management system. Many existing applications must also deal with dynamic access control. In such situations, the problem in management of static roles is that the role administrator has to manage all possible roles statically during application development because conventional RBAC systems have not been implemented. Most role administrator operations are similar in all applications.

In this paper, we implement an extended RBAC system that can manage the second problem. The basic idea is that we define abstract roles, and the actual role (access right set) is decided dynamically by using rules and context information, such as user place and time. Our purpose in proposing this method is to simplify the management of dynamic role assignment in a uniform way. The basic idea is that we define abstract roles, and the actual role (access right set) is decided dynamically by using rules and context information, such as user place, time, and so on. Our model consists of an *Abstract Role Set* (ARS), a *Context Rule* (CR), and an *Agent*, which derives actual access rights by using CR and *Context Information* (CI). CI is generated dynamically by an agent, and the access right is derived by the agent's inference engine. ARS, CR, and CI are defined in first-order logic programming (prolog) style, so the bidirectional queries for these relationships are easily accommodated.

In this paper, we discuss the efficiency of our proposed model by applying the model to the access control of our project management application.

Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'04)
1524-4547/04 $ 20.00 IEEE
IEEE COMPUTER SOCIETY

The remainder of this paper is structured as follows. In Section 2, we briefly introduce the basic RBAC system. In Section 3, we discuss the limitations of conventional RBAC systems by summarizing our project management system and its requirements for access control. In section 4, we present the basic idea of our model by using the access control mechanism of our project management system as an example. In section 5, we define our agent-based RBAC management extension and its usage pattern. Section 7 describes related work and presents the conclusion.

2. Role Based Access Control

By categorizing the minimum privilege for assigning a duty to a role, RBAC enable us to both defend against illegal accesses and harmful operations, and reduce management errors because RBAC is simple but supports a variety of security policies. The relationships of user, role, and access rights in the original RBAC (RBAC0) are defined as follows [1].

- *type user* of individual users
-*type role* of role identifiers
-*RM(r:role)* $\rightarrow$ 2^{usrs}, the role-to-member mapping that gives the set of users authorized for role r.
- *type permission* = $2^{(operation \times object)}$
- *RP(r:role)* $\rightarrow$ $2^{permissions}$, the role-to-access rights mapping that gives the set of access rights authorized for role r.

RBAC has two main properties: hierarchy and constraint. Role hierarchy is realized when all access rights of one role are contained in another role. The notation A $\succ$ B means role A contains role B, and in that case role A can also be treated as role B.

$$(\forall i, j : role)(\forall u : user) i \prec j \wedge u \in RM[i]$$
$$\Rightarrow u \in RM[j]$$

Here, RM[r] is a set of users who can perform given role r.

RBAC0 provides two static constraints on user-role and role-permission authorization: *Static Separation of Duty* (SSD) and *Role Cardinality*.

SSD represents the constraint of the role for which the user is authorized and is mutually exclusive with other roles for which the user has already obtained membership.

Ea: role $\times$ role, *Exclusive authorization set* that yields the pairs of roles that are mutually exclusive with each other for role membership.

$$(\forall u : user)(forall i, j : role) u \in RM[i] \wedge u \in RM[j]$$
$$\Rightarrow (i, j) \notin Ea$$

In other words, if a user is a member of both “i” and “j”, then these two roles cannot be SSD.
Another constraint supported by RBAC0 is role cardinality. Cardinality defines the maximum number of the users authorized for the role.

3. Our Project Management System and Its Access Control

In this section, we discuss the limitations of the original RBAC system by presenting some functions and some dynamic situations of the access control system for our target application, the project management system.

3.1 Project Management System

The purpose of our project management system is to efficiently manage a project and the system control resources, such as human resources and time. The system schedules project tasks, reports the results of the task, confirms and follows up on the schedule, evaluates task results, and so on. Using this system that we designed and implemented, managers and members can see the total flow of the project and its tasks, so they can share a common view of the project. Almost all operations of this system are subjected to access control. However, here we show some simple examples of system operations for ease of explanation.

1. The owner of the project task divides the task into subprojects, allocates adequate resources to the subproject, and reports the schedule to executants and members.
2. The executant of the project executes the allocated task or subtask and reports finishing or progress of the task.
3. Each project member checks the progress or finishing report from other members.
4. When the task is finished, the owner of the task evaluates the task and task executants. When the task is delayed, the owner reschedules the task.

For the four operations above, we need at least three roles for controlling access.

1. **Task Manager:** This role is for operations 1 and 4 above. This role is for the manager or owner of the task, and the role corresponds to the set of access

IEEE COMPUTER SOCIETY

rights of generation, resource allocation, deleting, and evaluation of the task.

2. **Task Executant:** This role is for operation 2 above, and consists of access rights like writing reports and reading reports.

3. **Task member:** This role is for operation 3 above, and the member can read task information such as time schedule, reports and evaluations.

Here, access rights of the upper role are contained in the lower role. The upper roles can thus be defined using inheritance in RBAC. In that case, the relationships of roles are:

$$menber \prec executant \prec manager$$

3.2 Dynamic Aspects of Our System

Our project management system cannot use the original RBAC for the operations mentioned in the previous section, primarily because there are multiple tasks in one project. For example, a manager in one project is not a manger in another project. In contrast, an executant in a project may be a manager in another project. For this reason, we cannot obtain actual access rights for the above role directly using the original RBAC system. The simplest solution is to define all possible roles for each task and project, but this is not rational because many roles have identical structures but different project names and access target. Besides, the conventional RBAC system cannot perform dynamic role allocation such as by role time constraint or delegation. For example, a user may share one role depending on the login time. Furthermore, we may want access control based on login place and priority access operations. We need an extended RBAC that would enable us to manage these dynamic attributes efficiently.

4 RBAC System for Dynamic Security Policy

In a computer system, one general approach for solving the problem of dynamic behavior is to first represent the target in abstract form and then decide details dynamically during runtime. For example, object-oriented programming languages provide interfaces and abstract classes for dynamic operations. When designing and implementing libraries, the library programmers define interfaces and abstract objects and then use these abstract methods in their programs. Library users then implement concrete instructions and decide the details of implementation when they implement there own applications. Program variables can be considered the simplest abstraction. For example, in logic programming, we can operate on a list using abstract logical variables such as [top| Tail] for separating the top of the list.

We apply this approach for our RBAC extension for dynamic access control. Namely, abstract role sets are defined statically, we use rules for decide the details of the relation of access rights. Concrete access rights are allocated dynamically using the abstract role and rules and context information at the runtime. For example, abstract role sets used in our project management system are illustrated in Table 1.

Table 1 Abstract Role Set

Project Name	Name
Target	T1, …
ROLE: Manager	T1.makeSchedule(); T1.deleteSchedule(); ….
ROLE: Executant	T1.setResult(); …
ROLE: Member	T1.readSchdule(); …

When the variable Name, which represents the name or project, is decided, target object Target and each access right of each role are automatically assigned. In Table 1, access rights of the target object are defined using object-oriented programming notation, such as T1.makeSchedule(), and T1.setResult(). Table 1 indicates that when the project name is decided by variable Name, the Manager role can perform the operation in operation two above, the executant can perform serResult() of the target object, and the member can perform readSchedule() of the target object. In Table 1, we use only one target T1, but our model allows using multiple targets in one abstract role set.

An example rule used for deciding the project and the access rights is defined in Table 2.

Table 2 Example of role-mapping rules.

Name	Project_task1
Target	/home/xml/task1.xml/, …
Manager	userA
Executant	userB, userC,
Member	userA, userB, userC, userD,…

When the rule in Table 2 is applied, userA can act as role Manger or Member for the project named project_name. For example, when a user has the role of Manager in this rule, he has the access right to

IEEE COMPUTER SOCIETY

makeSchedule() or delete Schedule() of the target related to "/home/xml/task1.xml." . The rule in Table 2 is the simplest one, and all roles can be determined directly by this rule. Generally our model allows more complex relations. For example, if the role administrator wants the executant role to be valid (active) only from 10:00 to 17:00, he may define a rule as in Table 3.

Table 3 Another example of role-mapping rules

Name	Project_task1
Target	/home/xml/task1.xml/
Manager	userA
Executant	Executant_Candidates & (1000<time<1700)
Member	userA, userB, userC, userD,…
Executant _ Candidates	userB, userC

The difference between Tables 2 and 3 is that the executant role definition in the abstract role set in Table 3 is not a list of users, but the rule is defined by using another "Executant_candidates" and a time constraint (1000<time<1700). The Executant_Canditate (line 6 in Table 3) declares that the corresponding users are userB and userC. By defining rules as in Tables 2 and 3, our system finds the appropriate relationships between user and permission that suit the condition for mapping to the abstract role set.

5. Agent-Based Management System

This section illustrates the agent-based method of realizing a system that can derive access rights from abstract role set, rules, and context information that we mentioned in the previous section. Some of the rules we introduced in the previous section are automatically generated at run time, and some other rules are defined statically and explicitly by the role administrator.

5.1 Components of Our Model

Figure 1 depicts an overview of the system, which is divided into three major parts, agent, agent input, and agent output. The agent input consists of an Abstract role set, a Context, and a Context Role. The agent derives access rights by using these inputs and an inference engine. The details of the components of our model are explained below.

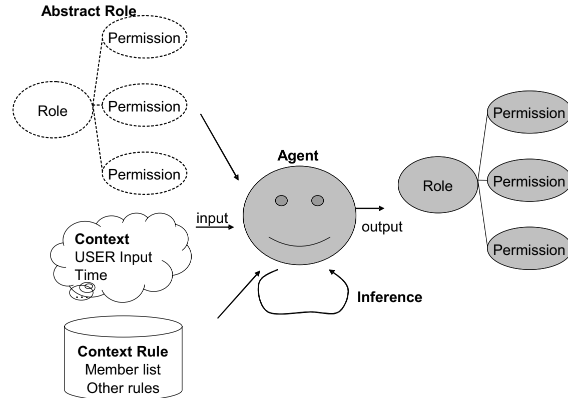

Figure 1 Agent Based Dynamic Role Management

- **Abstract Role Set (ARS)**
  Abstract Role Set defines the abstract permission mapping using variables such as presented in the previous section, and permission mapping is determined by binding these abstract variables. These variables are bound by agent inference, which was executed using context and context rules we will show below. This Abstract role set is a static relation that was defined in the role management phase by the role administrator.
- **Context Information (CI)**
  Context Information consists of states observed by the agents we mention below. For example, current time, user location, input data from user interface (UI), and dynamically generated rules such as delegation of roles, are examples of Context Information. Context Information thus consists of facts and rules that change dynamically based on the context of the application and user environment.
- **Context Rule**
  Context Rules are static rules and part of the agent input. Agents generate all rules with the above CI.
- **Agent**
  An *Agent* decides access by inference. Agent inputs are the Abstract Role Set, Context Information, and Context Role; the output is an access right. User queries are also examples of Context Information, and agents can answer the question by inference.
- **Access rights (Concrete Role Set)**
  Access rights are a set of access rights determined by the agent. Role is a set of access rights, so access rights can be seen as a concrete role set.

IEEE COMPUTER SOCIETY

5.2 I/O of Agent

We use a logic programming system to realize a system consisting of the components we introduced in the previous section. ARS, Context, and CR are facts and rules, so they can be defined in first-order logic programming (prolog) style. Besides, agent is designed with a prolog engine as the inference system. Here, Context Information is represented as the clauses generated by agents dynamically based on the context that the agent observed. In this section, we define agents and the relationship of an agent's input and output by presenting an example of our project management application that we mentioned in section 4.

5.2.1 Abstract Role Set (ARS)

An *Abstract Role Set* is defined as a set of clauses. The role administrator defines the name of an abstract role set, role names to which the user is mapped, the object which the role targets, and access rights as target operations by using prolog programming syntax. The prolog rule that corresponded to Table 1 is as follows.

```
role(member, U) :- role(executant, U).         (Rule 1)
role(executant, U) :- role(manager, U).         (Rule 2)
makeSchedule(U, T) :-
          role(manager, U), target(T) .         (Rule 3)
deleteSchedule(U, T) :-
          role(manager, U) target(T).           (Rule 4)
serResult(U ,T) :-
          role(executant, U), target(T).        (Rule 5)
readSchedule(U,T) :-
          role(member,  U), target(T).          (Rule 6)
```

The ARS definition programs must contain all abstract types of the final role and all the types of access rights. In the above example, the rule has to contain and define access rights for each manager role, executant role, and member role. In this rule program, Rule 1 and Rule 2 are rules for role hierarchy. Rule 1 argues that if a user acts as an executant, the user can also act as a member. Rule 2 says that when a user acts as manager, then the user also has access rights of the executant role. Predicates role(R,U) and target(T) are not defined in this program. These predicates are defined in Context Information and Context Rules, which we mention below.

5.2.2 Context Information

Context Information is a set of clauses that an agent generates dynamically based on the context. In Table 3, current time (time) and current user information are defined as clauses as shown below. Context Information is used like built-in predicates in the Abstract Rule Set and Context Rule provided by the role administrator.

```
user(userA).                          (Rule 7)
target("/home/xml/task1").            (Rule 8)
selected(manager).                    (Rule 9)
```

Here, the clause user(X). (Rule 7) corresponds to the fact that a user is logging onto the system. The clause target(X). (Rule 8) corresponds to the fact that the selected (or to be selected) project is X. Rule 9 says that the selected (or to be selected) role is now manager. Other examples of Context that do not appear in Table 3 contain context constraints such as user place information (log-in machine), exclusive control, or priority control.

5.2.3 Context Rule

A *Context Rule* defines the rule for generating a role based on the context observed by an agent. Using the context rule, an agent can decide access rights. Some of the rules are defined as Context Information, and others are defined as context rules. For example, the remaining rules in Table 3 are defined as Context Rules as follows.

```
rolemember(member,userD).                   (Rule 10)
executant_candidate(userB).                 (Rule 11)
executant_cadidate(userC).                  (Rule 12)
rolemember(executant, U) :-
    executant_candidate(U) ,sys_time(X),
    X>1000, X<1700.                         (Rule 13)
rolemember(manager, userA).                 (Rule 14)
role(R,U) :-
   user(U), selected(R), rolemember(R,U).   (Rule 15)
```

Here, Rule 10 declares that userD can act as a role "member," and Rule 11 declares that UserB is a candidate for an "executant" role. The actual executant role is assigned when a user is the candidate, and the log-in time has to be from 1000 to 1700. In our project management example, one Abstract Role set and Context Rules for each project and Context Information are used for dynamically determining the access right. As an implementing issue, SSD and role cardinality are also represented as Context Rules.

5.2.4 Agent

The primary function of an agent is to generate appropriate access rights by executing the above three rules. Generating the Context, which is one of the three rules, is also an agent responsibility. In the above

IEEE COMPUTER SOCIETY

example, when there are three rules and agentA selects the target object taskA and has already obtained a manager role, then the agent dynamically adds Rule 7, Rule 8, and Rule 9 as Context Information. In this situation, the query for the agent is as follows when the system wants to know if this user has the access right to makeSchedule of target file "/usr/xml/task1."

?- makeSchedule(userA,"/usr/xml/task1"). (Query 1)

For this query, the agent infers that user A has access to the operation by using a prolog engine. Using logic programming style rules enables us to make bidirectional queries. This is another advantage of this programming style. For example, when the system wants to know the list of users who may execute setResult, the dynamically generated Context Information and queries are as shown below. In this case, the agent replies with the names of corresponding users UserB, UserC, and UserA.

```
user(_).
target(/home/xml/task1).
selected(_).
?-setResult(X,/home/xml/task).
X = userB;
X = userC;
X = userA
yes.
```

6. Related Work and Conclusion

Our proposed Abstract Role Set is structured similarly to Template proposed by Giuri et al. [2]. In our model, we assume that the Abstract Role Set is used with Context Information generated by an agent and the Context Role defined by the role administrator; we also emphasize using user context. Our approach can thus deal with dynamic information. Besides, our model enables bidirectional queries and is executed in a unified way because we apply a first-order logic programming style for defining the rules and context information.

Zhang [3] uses prolog-style rules for defining role delegation, and we use rules to dynamically select access rights by using context information. They do not emphasize context and abstract roles for a dynamic environment, so their objects or usage rules differ from our approach.

Convington et al. extended the original RBAC to apply concepts or roles not only to subjects but also to objects and environments, and proposed a method to define security policies more finely[4]. In this model, role administrators have to define many roles, and role management tends to be complex. In our model, the role administrator defines only necessary roles when designing the system.

In this paper, we propose a model and method for realizing an access control system for applications with many dynamic attributes. The dynamic access rights are determined by using rules, context information, and agents for inference by these rules. The dynamic access rights are determined by using rules, context information, and agents for inference by these rules. We are now developing a web-based role server that supports our model and method, as well as it's API for role manager, to enable our proposed method to be used in many applications. We also plan to evaluate the relationships between the rate of dynamic rules, the total number of dynamic rules, and scalability. We now plan to evaluate the relationships between the rate of dynamic rules, the total number of dynamic rules, and scalability.

7. References

[1] David F.Ferraiolo, John F.Barkly, and D.Richard Kuhn, "A role based access control model and reference implementation within a corporate internet" in ACM Transactions on Information Systems Security, volume 1, February 1999.

[2] Luigi Giuri and Pietro Iglio. Role templates for content-based access control. In Proceedings of the Second ACM Workshop on Role Based Access Control, pp.153-159, 1997.

[3] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu, "A Rule-Based Framework for Role-Based Delegation and Revocation", ACM Transactions on Information and System Security, Vol.6, No.3, August 2003, pp.404-441.

[4] Michael Convington, etc. al., "Securing Context-Aware Applications Using Environment Roles", SACMAT'01 2001, pp.10-20.

IEEE COMPUTER SOCIETY