ELSEVIER

Information and Software Technology 44 (2002) 649–657

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Reconstructing a formal security model

Gail-Joon Ahn[a,*], Seung-Phil Hong[b], Michael E. Shin[c]

[a]*Department of Software and Information Systems, University of North Carolina at Charlotte, Charlotte, NC 28223, USA*
[b]*Research and Development Center, LG CNS Co., Korea*
[c]*Department of Information and Software Engineering, George Mason University, Fairfax, VA 22030, USA*

Received 14 August 2001; revised 6 May 2002; accepted 22 May 2002

Abstract

Role-based access control (RBAC) is a flexible approach to access control, which has generated great interest in the security community. The principal motivation behind RBAC is to simplify the complexity of administrative tasks. Several formal models of RBAC have been introduced. However, there are a few works specifying RBAC in a way which system developers or software engineers can easily understand and adopt to develop role-based systems. And there still exists a demand to have a practical representation of well-known access control models for system developers who work on secure system development. In this paper we represent a well-known RBAC model with software engineering tools such as Unified Modeling Language (UML) and Object Constraints Language (OCL) to reduce a gap between security models and system developments. The UML is a general-purpose visual modeling language in which we can specify, visualize, and document the components of a software system. And OCL is part of the UML and has been used for object-oriented analysis and design as a de facto constraints specification language in software engineering arena. Our representation is based on a standard model for RBAC proposed by the National Institute of Standards and Technology. We specify this RBAC model with UML including three views: static view, functional view, and dynamic view. We also describe how OCL can specify RBAC *constraints* that is one of important aspects to constrain what components in RBAC are allowed to do. In addition, we briefly discuss future directions of this work. © 2002 Elsevier Science B.V All rights reserved.

Keywords: Access control; Role-based; Formal model; UML

1. Introduction

In role-based access control (RBAC) permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the roles' permissions. RBAC has emerged as a widely accepted alternative to classical discretionary and mandatory access controls [10]. RBAC is also a flexible approach that has generated great interest in the security community [2]. Several models of RBAC have been published and several commercial implementations are currently available. RBAC regulates the access of users to information and system resources on the basis of activities that users need to execute in the system. It requires the identification of roles in the system. A role can be defined as a set of permissions associated with a particular working activity. Then, instead of specifying all the accesses each individual user is allowed, access authorizations on objects are specified for roles. Since roles in an organization are relatively persistent with respect to user turnover and task re-assignment, RBAC provides a powerful mechanism for reducing the complexity, cost, and potential for error in assigning permissions to users within the organization. Because roles within an organization typically have overlapping permissions RBAC models include features to establish role hierarchies, where a given role can include all of the permissions of another role. Another fundamental aspect of RBAC is authorization constraints (also simply called constraints).

Although the usefulness of RBAC has been recognized and several frameworks for the development of role-based systems have been introduced [3–5], these prior works were sometimes hard for system developers to understand because some are too abstract and formal, and others are often ad-hoc solutions focused on specific application-oriented frameworks. These frameworks were not sufficient to provide a sound blueprint to system developers. There still exists a demand in developing a

* Corresponding author. Tel.: +1-704-687-3783; fax: +1-704-687-4893.
E-mail addresses: gahn@uncc.edu (G. J. Ahn), philhong@lgens.com (S.-P. Hong), eshin@gmu.edu (M.E. Shin).

0950-5849/02/$ - see front matter © 2002 Elsevier Science B.V All rights reserved.
PII: S0950-5849(02)00092-7

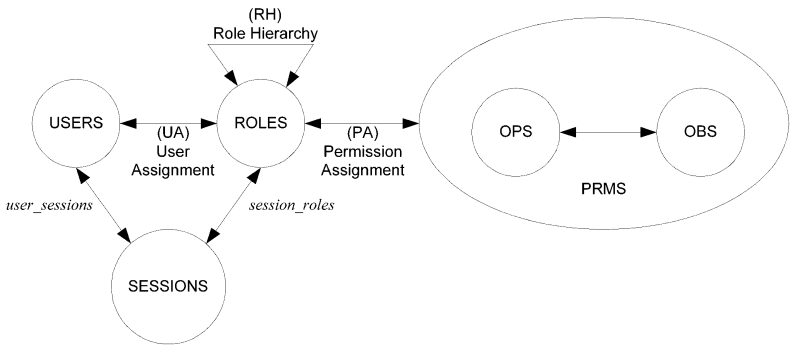

Fig. 1. RBAC model.

practical methodology to represent formal RBAC models using widely accepted software engineering tools. This issue is rarely addressed in the research literature while domain-specific proof-of-concept implementations have been practiced and discussed at considerable length.

In this paper our focus is to reduce a gap between security models and system developments. To do so, we adopt the NIST proposed RBAC standard [6,24] since it includes most of well-recognized RBAC work [24]. In this paper we represent this RBAC model with a general-purpose visual modeling language UML. We choose Unified Modeling Language (UML) because it has been a standard language in the modeling community.

The constraints in RBAC have been considered as one of the most important components that enforce the principal motivations of RBAC model [7]. In the past few years, researchers and vendors have proposed enhancements of specification of RBAC constraints. Ref. [24] dealt with more fine-granularity constraints such as operation-based and object-based. Recently Ahn and Sandhu [8] proposed a formal language called Role-based constraints specification language 2000 (RCL2000) and identified useful role-based authorization constraints such as *prohibition*and *obligation* constraints. The users of RCL2000 are security researchers and security policy designers who have to understand organizational objectives and articulate major policy decisions to support theses objectives. RCL2000 also provides *n*-ary expressions and more flexibility, sharing a great deal of common semantics about expressing access control constraints [9]. Next, we may face the following question: *How can we realize these useful constraints at the system design stage*? The idea of our approach is to inject constraints specifications into an UML-representation of RBAC. Using Object Constraint Language (OCL) that has been used to express constraints in analysis and design, we demonstrate that OCL can help us specify previously identified constraints at the system design step. The constraints include separation of duty constraints, prerequisite constraints, and cardinality constraints. This approach is moderately convenient for system developers to specify and understand constraints of RBAC model.

The rest of this paper is organized as follows. In Section 2, we briefly describe related technologies including RBAC model, UML and OCL. Section 3 demonstrates how these languages can specify the formal model. In Section 4, we briefly discuss future works. Section 5 concludes this paper.

2. RBAC model

2.1. Role-based access control model

RBAC has recently received considerable attention as a promising alternative to traditional discretionary (DAC) and mandatory access controls (MAC) (see, for example, [1, 10–12]). As MAC is used in the classical defense arena, the policy of access is based on the classification of objects such as top-secret level. The main idea of DAC is that the owner of an object has discretionary authority over who else can access that object. But RBAC policy is based on the roles of the subjects and can specify security policy in a way that maps to an organization's structure.

A standard for RBAC model was proposed by Ferraiolo et al. [24]. They defined three models: Core RBAC, Hierarchical RBAC and Constrained RBAC. And their Constrained RBAC adds separation of duty relations such as Static Separation of Duty relations and Dynamic Separation of Duty relations. Fig. 1 illustrates the Hierarchical RBAC model. Motivation and discussion about various design decisions made in developing this family of models is given in Ref. [24]. Fig. 1 shows (regular) roles and permissions that regulate access to data and resources. Intuitively, a user is a human being or an autonomous agent, a role is a job function or a job title within the organization with some associated semantics regarding the authority and responsibility conferred on a member of the role, and a permission is an approval of a particular operation one or more objects in the system. Roles are organized in a partial order $\geq$, so that if $x \geq y$ then role x inherits the permissions of role y. Members of x are also implicitly members of y. In such cases, we say x is senior to y. Each session relates one user to possibly many roles. The idea is that a user establishes a session and activates some subset of roles that he or she is a member of (directly or indirectly by means of the role hierarchy). The

RBAC model has the following components and these components are formalized from the above discussions.

- *USERS* is a set of users,
- *ROLES* is a set of roles,
- *OPS*is a set of operations,
- *OBS*is a set of objects,
- $PRMS = 2^{(OPS \times OBS)}$, is a set of permissions,
- $UA \subseteq USERS \times ROLES$, is a many-to-many user to role assignment relation,
- $PA \subseteq PRMS \times ROLES$, is a many-to-many permission to role assignment relation,
- $RH \subseteq ROLES \times ROLES$, is partially ordered role hierarchies (written as $\geq$ in infix notation),
- *SESSIONS*, is a set of sessions,
- *assigned_users* $(r : ROLES) \rightarrow 2^{USERS}$, the mapping of role r onto a set of users,
- *assigned_permissions* $(r : ROLES) \rightarrow 2^{PRMS}$, the mapping of role r onto a set of permissions,
- *user_sessions* $(u : USERS) \rightarrow 2^{SESSIONS}$, the mapping of user u onto a set of sessions,
- *session_roles* $(s : SESSIONS) \rightarrow 2^{ROLES}$, the mapping of session s onto a set of roles,
- *avail_session_perms* $(s : SESSIONS) \rightarrow 2^{PRMS}$, the permissions available to a user in a session,
- *authorized_user* $(r : ROLES) \rightarrow 2^{USERS}$, the mapping of role r onto a set of users in the presence of a role hierarchy, and
- *authorized_permissions* $(u : ROLES) \rightarrow 2^{PRMS}$, the mapping of role r onto a set of permissions in the presence of a role hierarchy.

A user can be a member of many roles and a role can have many users. Similarly, a role can have many permissions and the same permissions can be assigned to many roles. Each session relates one user to possibly many roles. Intuitively, a user establishes a session during which the user activates some subset of roles that he or she is a member of. The permissions available to the users are the union of permissions from all roles activates in that session. The nature of permissions is not specified in the NIST RBAC model. Permissions can be fine-grained (e.g. at the level of individual objects) or coarse-grained (e.g. at the level of entire sub-systems). They can be defined in terms of primitive operations such as `read` and `write` in an OS and `create` and `delete` in DBMS, or abstract operations, such as `credit` and `debit`. An operation is an executable representation of a program. Permissions can also be customized. Each session is associated with a single user. This association remains constant for the life of a session. A user may have multiple sessions open at the same time, each in a different window on the workstation screen for instance. Each session may have a different combination of active roles. The concept of a session equates to the traditional notation of a subject in access control. A subject is a unit of access control, and a user may have multiple subjects (or sessions) with different permissions active at the same time. There is a collection of constraints that allow or forbid values of various components of the RBAC model.

2.2. Constraints

Constraints are an important aspect of access control and are a powerful mechanism for laying out a higher-level organizational policy [7]. Consequently, the specification of constraints needs to be considered. This issue has become crucial in the RBAC. There are several works such as Refs. [13,14] that deal with constraints in the context of RBAC. These previous works, however, are preliminary and tentative, and need substantial further development. Most prior work has focused on separation of duty constraints [15]. Chen and Sandhu [13] suggested how constraints could be specified. Giuri and Iglio [14] defined a new model to provide the capability of defining constraints on roles. In their model, a role is defined as a named set of constrained protection domains (NSCPD) that is activatable only if the corresponding constraint is satisfied. Their description focused on the activation of roles. But we should also consider that constraints be applied to other components in RBAC. Jaeger and Tidswell [9] proposed a constraint specification framework in which constraints are represented with graphical language expressions. Jajodia et al. [16] introduced a language based authorization constraints specification. While these language-based approaches have been presented, these tend to lack the notion of role. Recently, Ahn and Sandhu [8] introduced a formal language, called RCL2000 and identified the major classes of constraints in RBAC such as *prohibition constraints* and *obligation constraints*, including *cardinality constraints*. Also Ferraiolo et al. [24] clearly identified and formalized *separation of duty constraints* as major constraints in NIST RBAC model. In this paper, we try to accommodate these well-known constraints using UML and OCL. Our UML representation could be extended to include other static and dynamic constraints but it remains for future work.

2.2.1. Prohibition constraints

In organizations, we need to prevent a user from doing (or being) something that he is not allowed to do (or be) based on organizational policy. Prohibition constraints are constraints that forbid the RBAC component from doing (or being) something that is not allowed to do (or be). A common example of prohibition constraints is separation of duty (SOD). SOD is a fundamental technique for preventing fraud and errors, known and practiced long before the existence of computers. We can consider the following statement as an example of this type of constraint: if a user is assigned to purchasing manager, he cannot be assigned to accounts payable manager. This statement requires that the same individual cannot be assigned to both roles that are declared mutually exclusive.

2.2.2. Obligation constraints

We also need to force a user to do (or be) something that he is allowed to do (or be) based on organizational policy. We derived another class of constraints from this motivation. Obligation constraints are constraints that force the RBAC component to do (or be) something. The motivation of this class is from a constraint that requires a user to have certain combinations of roles in user-role assignment. We name this type of constraints as *prerequisite constraints*. There is another constraint that requires that certain roles should be simultaneously active in the same session in the simulation of lattice-based access control in RBAC [17].

2.2.3. Cardinality constraints

Another constraint is a numerical limitation for the number of users, roles, and sessions. For example, only one person can fill the role of department chair; similarly, the number of roles (sessions) an individual user can belong to (activate) could be constrained.

3. Related technologies

3.1. Unified Modeling Language

The UML is a general-purpose visual modeling language in which we can specify, visualize, and document the components of software systems. It captures decisions and understanding about systems that must be constructed [18]. The UML has become a standard modeling language in the field of software engineering [19].

The UML consists of functional, static, and dynamic models. In a functional model, the functional requirements of systems are specified using use case diagrams. A use case defines the services that a system provides to users. A static model provides a structural view of information in a system. Classes are defined in terms of their attributes and relationships. The relationships include association, generalization/specialization, and aggregation of classes. A dynamic model shows a behavioral view of a system. It can be described with collaboration diagrams, sequence diagrams, and statechart diagrams. A collaboration diagram and sequence diagram are developed to capture how objects collaborate with each other to execute a use case. State dependent views of objects are defined in statechart diagrams.

In this paper, we take class diagrams, use case diagrams, and object collaboration diagrams for a static view, a functional view, and a dynamic view of the RBAC model, respectively. In the rest of this paper, we use UML notations introduced in Refs. [19–21].

3.2. Object Constraint Language

The OCL [21,22] is an expression language that describes constraints on object-oriented models. A constraint is a restriction on one or more values of an object-oriented model. OCL is an industrial standard for object-oriented analysis and design. Each OCL expression is written in the context of an instance of a specific type. In an OCL expression, the reserved word **self** is used to refer to the contextual instance. The type of the context instance of an OCL expression is written with the **context** keyword, followed by the name of the type. The label **inv**: declares the constraint to be an invariant constraint. For example, suppose that employees work for a company and they are involved in projects. These relationships can be modeled using the class model of the UML. If the context is Company, then *self* refers to an instance of Company. The following shows an example of OCL constraint expression describing a company that has more than 200 employees:

context Company **inv**:
self.employee → size > 200

The self.employee is a set of employees that is selected by navigating all the classes such as Company and Employee classes through an association. The '.' stands for a navigation. A property of a set is accessed by using an arrow '→' followed by the name of the property. A property of the set of employees is expressed using a keyword 'size' in this example.

The following shows another example describing that an employee can join a project A only if the employee is already involved in a project B:

context Employee **inv**:
self.project → includes('A') implies self.project → includes('B')

The self.project → includes('A') means that the project A is an element of projects in which an employee is involved. The **implies** statement is true if self.project → includes('A') is false, or if self.project → includes('A') and self.project → includes('B') are true.

An OCL expression delivers a subset of a collection. That is, the OCL has special constructs to specify a selection from a specific collection. For example, the following OCL expression specifies that the collection of employees whose age is over 50 is not empty:

context Company **inv**:
self.employee → select(age > 50) → notEmpty

The '**select**' takes an employee from self.employee and evaluates an expression (age > 50) for the employee. If this evaluation result is true, then the employee is in the result set.

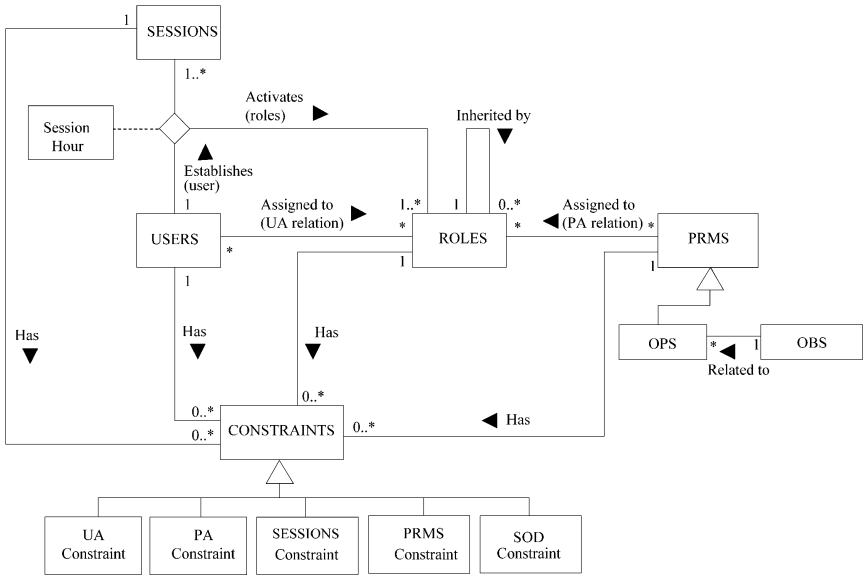

Fig. 2. Class diagram (static view).

4. Practical transition towards an application

4.1. UML-representation of RBAC

Major components in RBAC are users, roles, permissions, sessions, and constraints. In order to represent RBAC model using UML, we analyze each component with a notion of object class. In Sections 4.1.1–4.1.3, our analysis is specified by three different views such as static, functional, and dynamic views.

4.1.1. Static view

The conceptual static model for RBAC is depicted in Fig. 2. It contains classes, relationships between classes, and cardinalities in relationships. The basic entities are user, role, permission, constraint, and session classes. The permission class is represented as a relationship between operations and objects. The constraints in the RBAC model can have various forms, which are dependent on application systems. In order to simplify the analysis model, the constraint in our static model has only five constraints such as user constraint, permission constraint, session constraint, permission constraint, and separation of duty constraint. Also, the static model has a special class called session hour. This class is used when a user establishes a session to activate her/his roles. This notion is useful to express session-based constraint. For example, an organization may require that a user can establish her/his session only during the certain amount of time. In order to enforce this kind of constraints, we need to keep track of session hours for each session.

In the static model, UA relation and PA relation are represented as 'Assigned to' relation with a many-to-many cardinality. User-session relation is viewed as a user can establish one or more sessions to activate at least one or more roles per each session with the constant session lifetime. The role inheritance relation is shown as well.

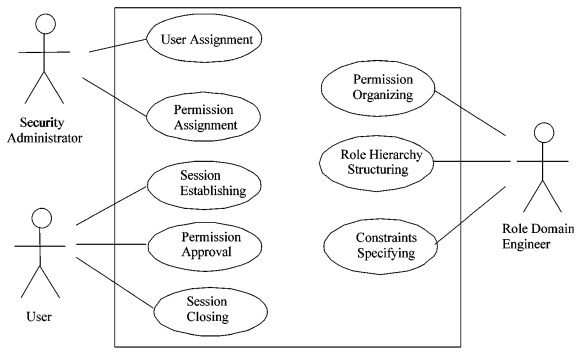

Fig. 3. Use case model.

4.1.2. Functional view

Next, we make more concrete functional requirements to represent functions in RBAC systems. The functional view is depicted in Fig. 3 using the use case model that has three actors such as a security administrator, a user, and a role domain engineer. The role domain engineer organizes a set of permissions, constructs role hierarchies, and specifies constraints. The security administrator who administrates a role-based system may assign users to roles and assign permissions to roles. The user who would be real persons or external systems may establish sessions, make a request to perform an operation on an object, and close sessions.

The following shows the brief specification of the Session Establishing use case:

Use case: Session Establishing use case
Actors: User
Precondition: System idle
Description: A user presents information for establishing a session.

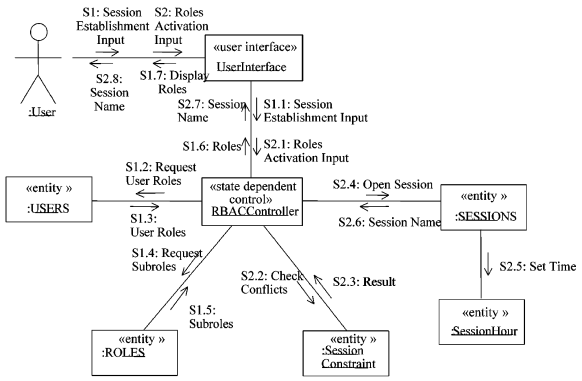

Fig. 4. Collaboration diagram (session establishment).

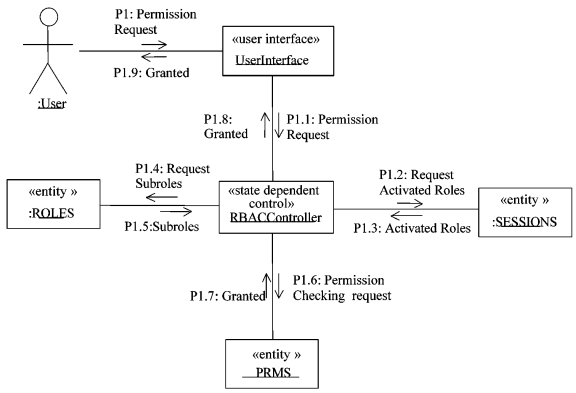

Fig. 5. Collaboration diagram (permission checking).

System displays the roles that a user can activate. The user selects roles to activate. System activates a session with the roles that the user selected.

After a user establishes her/his session with selected roles, the user may need to access the system resources requiring authorization procedures that should be based on her/his role information. In other words, the permissions that are associated with her/his roles should be approved by the system. The following shows the brief specification of such a use case, called Permission Checking use case:

Use case: Permission Checking use case
Actors: User
Precondition: A session was activated by a user.
Description: A user makes a request to perform an operation on an object.
System notifies a user whether or not the permission is approved.

In this paper, the limited functions are inferred from the RBAC model. We may also consider other functions in the RBAC system that may be articulated. For example, we may require additional functions for monitoring sessions initiated by a security administrator or inquiring a session's status initiated by a user.

4.1.3. Dynamic view

In the dynamic view, the use cases are refined to show the interactions among the objects in each use case. The collaboration diagram for the session establishment is depicted in Fig. 4, where a user initiates the use case through a user interface, and RBACController coordinates interactions between the objects in the use case. The collaboration diagram for permission checking is shown in Fig. 5, where it requires a precondition that a session needs to be activated before the execution of permission checking use case. The complete descriptions of each diagram are described in Table 1.

4.2. Constraints specification

The conceptual static model (see Fig. 2) contains classes, their attributes, and their relationships. The basic entities are user, role, permission, constraint, and session classes. Each class has an attribute, that is, a name, which can be an identification of instance of the class. In the class model, the UA and PA relations indicate that users can be assigned to roles and permissions can be assigned to roles, respectively. Next, we need to express constraints that regulate the construction and the activities of each class from this UML representation. Our expression includes separation of duty constraints, prerequisite constraints, and cardinality constraints.

4.2.1. Separation of duty constraints

Separation of duty is a well-known principle for preventing fraud by identifying conflicting roles—such as Purchase Manager and Accounts Payable Manager—and ensuring that the same individual can belong to at most one conflicting role. We may apply this conflicting notion to other components such as user and permission in RBAC. The concept of conflicting permissions defines conflict in terms of permissions rather than roles. Thus the permission to issue purchase orders and the permission to issue payments are conflicting, irrespective of the roles to which they are assigned. Conflict defined in terms of roles allows conflicting permissions to be assigned to the same role by error (or malice). Conflict defined in terms of permissions eliminates this possibility. In the real world, we may also have a notion of conflicting users based on organizational policy. The following examples show how we can specify this type of constraints using OCL.

Example 1. *Conflicting roles cannot be assigned to the same user*. Consider two mutually exclusive roles such as accounts payable manager and purchasing manager. Mutual exclusion specifies that one individual cannot have both roles. This constraint on UA can be specified using the OCL expression as follows:

Table 1
Use cases of session establishment and permission checking

Session establishment	Permission Checking
[S1] A user inputs information for establishing a session	[P1] A user inputs information for session approval that includes a session name and a permission that s/he requests
[S1.1–1.7] User Interface sends session establishment input to RBAC Controller. RBAC Controller requests user roles from User entity. User entity replies user roles to RBAC Controller. If user roles from User entity are not available, RBAC Controller notifies the user through User Interface. RBAC Controller requests the user's subroles from Role entity. Role entity infers the user's subroles and replies the subroles to RBAC Controller. RBAC Controller sends the user's roles to User Interface. User interface displays roles that the user can select for activating the session	[P1.1] User interface sends the permission approval input to RBAC Controller
[S2] The user selects roles that s/he wants to activate in a session	[P1.2] RBAC Controller requests activated roles from Session entity
[S2.1–2.8] User interface sends input for the session activation to RBAC Controller. RBAC Controller requests Session Constraint entity to check for conflicts among the roles that the user selected. Session Constraint entity checks conflicts with the roles and replies the result to RBAC Controller. If the roles that user selected have conflicts, RBAC Controller notifies the user through User Interface. RBAC Controller requests opening a session from Session entity. Session entity requests setting time from Session entity. Session entity replies a session name to RBAC Controller. RBAC Controller sends the session name to User Interface. User Interface displays the session name	[P1.3] Session entity replies the user's activated roles to RBAC Controller. If a session name from the user is not available, RBAC Controller notifies the user through User Interface
	[P1.4] RBAC Controller requests the subroles of the user's activated roles from Role entity
	[P1.5] Role entity infers the subroles and replies them to RBAC Controller
	[P1.6] RBAC Controller requests a permission approval from Permission entity, sending the user's activated roles, their subroles, and the permission that the user want to get the approval
	[P1.7] Permission entity checks permissions for the roles and sends an approval to RBAC Controller. If the permission is not approved, RBAC Controller notifies the user through User Interface
	[P1.8] RBAC Controller sends the approval to User Interface
	[P1.9] User Interface displays the approval to the user

context User **inv**:
let M: Set = {{accounts payable manager, purchasing manager},…} **in**
M → select(m|self.role → intersection(m) → size > 1) → isEmpty

This constraint expression selects all mutually exclusive sets, checks all roles assigned to each user, and determines whether the user has more than one mutually exclusive role. Hence, this expression ensures that a user can have at most one of mutually exclusive roles.

Example 2. *Conflicting permissions cannot be assigned to the same role*. This example says that a user can have, at most, one conflicting permission acquired through roles assigned to the user. This constraint is a stronger formulation than Example 1, which prevents mistakes in role-permission assignment. Suppose we have two conflicting permissions such as 'prepare check' and 'issue check'. The OCL expression is as follows:

context Role **inv**:
let M: Set = {{prepare check, issue check},…} **in**
M → select(m|self.permission → intersection(m) → size > 1) → isEmpty

Example 3. *Conflicting users cannot be assigned to the same role*. Conflicting users should also be considered. For example, for the process of preparing and approving purchase orders in the purchasing manager role, it might be company policy that members of the same family should not prepare the purchase order, and also be a user who approves that order. The following expression ensures that two conflicting users, user α and user β, cannot be assigned to the same role.

context Role **inv**:
let M: Set = {{user α, user β},…} **in**
M → select(m|m → intersection(self.user → select(self.name = 'purchase manager')) → size > 1) → isEmpty

Example 4. *Conflicting roles cannot be activated in the same session*. This example is a simple dynamic separation of duty constraint. Suppose that a user has the supervisor roles and inherits permissions from both accounts payable manager role and purchasing manager role. It may be acceptable for the user not to activate these two conflicting roles at the same time. The following is OCL expression about this constraint.

context User **inv**:
let M: Set = {{accounts payable manager, purchasing manager},…} **in**
M → select(m|m → intersection(self.session.role) → size > 1) → isEmpty

4.2.2. Prerequisite constraints

This constraint is based on the concept of prerequisite

roles introduced in Ref. [1]. For example, a user can be assigned to the engineer role only if the user already is assigned to the employee role. It ensures that only users who are already assigned to the employee role can be assigned to the engineer role. We call this kind of constraint as prerequisite-role constraints. The following examples demonstrate that OCL can also specify prerequisite constraints.

Example 5. *A user can be assigned to role r1 only if the user is already a member of role r2*. Mostly, the prerequisite role is junior to the new role being assumed. Consider only those users who are already members of the project_team role can be assigned to the tester role within that project. This constraint can be specified as follows:

context User **inv**:
self.role → includes('tester') implies self.role → includes('project_team')

Example 6. *A permission p can be assigned to a role only if the role already possesses permission q*. This constraint is the dual form of Example 5. For instance, in many systems permission to read a file requires permission to read the directory in which the file is located. Assigning the former permission without the latter would be incomplete. This constraint on PA can be specified using the OCL expression as follows:

context Permission **inv**:
self.role → includes('read file') implies self.role → includes('read directory')

4.2.3. Cardinality constraints

Another constraint type is a numerical limitation for classes in a role-based system. This numerical limitation may vary depending upon the organizational policy. We show that OCL can specify these constraints without any extension of language.

Example 7. *Numerical limitation N that exists for the number of users authorized for a role cannot be exceeded*. This example limits the number of users to be assigned to a role. Suppose there is only one person in the role of chairman of a department. The chairman role should be assigned to only one user. The OCL expression for this constraint on UA can be as follows:

context Role **inv**:
self.user → select(u|self.name = 'chairman') → size = 1

Example 8. *Numerical limitation N that exists for the number of sessions a user can have active at the same time*. This example limits the number of sessions to be activated by a user. For example, a user is allowed to activate only two sessions at the same time. This constraint can be specified using OCL as follows.

context User **inv**:
self.session → size ≤ 2

5. Concluding remarks

We have demonstrated how a well-known model can be represented using software engineering tools. Rather than simply enumerating each component in RBAC model, we showed UML-based analysis model using class diagrams, use case diagrams, and object collaboration diagrams. Also, we have shown how authorization constraints for role-based systems can be specified using an industrial standard constraints specification language, OCL. We have specified separation of duty constraints, prerequisite constraints and cardinality constraints, OCL. As a result, we can represent a formal security model so that system developers may understand components and requirements on secure role-based systems development. This issue has rarely been addressed in the literature. We believe that our framework can be a basis in developing role-based systems built upon formal RBAC models.

There is room for much additional work with our approach. Based on this work, we would investigate how each component and functional specifications in NIST RBAC model can be fully represented with UML. Also, the validation of OCL specifications and time-based constraints will be studied. A case study for validating our specifications using an OCL parser [23] is currently under investigation.

References

[1] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, Role-based access control models, IEEE Computer 29 (2) (1996) 38–47.
[2] J. Joshi, A. Ghafoor, W. Aref, E. Spafford, Digital government security infrastructure design challenges, IEEE Computer 34 (2) (2001) 66–72.
[3] P. Epstein, R. Sandhu, Towards a UML based approach to role engineering, Proceedings of the Fourth ACM Workshop on Role-based Access Control, Fairfax, VA, 1999, October 28–29, pp. 135–142.
[4] C. Youman, E. Coyne, R. Sandhu (Eds.), Proceedings of the Second ACM Workshop on Role-based Access Control, Fairfax, VA, 1997, November 6–7.
[5] C. Youman, E. Coyne, R. Sandhu (Eds.), Proceedings of the Third ACM Workshop on Role-based Access Control, Fairfax, VA, 1998, October 22–23.
[6] R. Sandhu, D. Derraiolo, R. Kuhn, The NIST model for role-based access control: towards a unified standard, Proceedings of Fifth ACM Workshop on Role-based Access Control, Berlin, Germany, 2000, July.
[7] T. Jaeger, On the increasing importance of constraints, Proceedings of

the Fourth ACM Workshop on Role-based Access Control, Fairfax, VA, 1999, October 28–29, pp. 135–142.

[8] G.-J. Ahn, R. Sandhu, Role-based authorization constraints specification, ACM Transactions on Information and Systems Security 3 (4) (2000).

[9] T. Jaeger, J. Tidswell, Practical safety in flexible access control models, ACM Transactions on Information and Systems Security 4 (3) (2002) in press.

[10] D. Ferraiolo, R. Kuhn, Role-based access controls, Proceedings of the 15th NIST–NCSC National Computer Security Conference, Baltimore, MD, 1992, October 13–16, pp. 554–563.

[11] M.Y. Hu, S.A. Demurjian, T.C. Ting, User-role based security in the ADAM object-oriented design and analyses environment, in: J. Biskup, M. Morgernstern, C. Landwehr (Eds.), Database Security VIII: Status and Prospects, North-Holland, Amsterdam, 1995.

[12] I. Mohammed, D.M. Dilts, Design for dynamic user-role-based security, Computers and Security 13 (8) (1994) 661–671.

[13] F. Chen, R. Sandhu, Constraints for role based access control, Proceedings of the First ACM Workshop on Role-Based Access Control, Gaithersburg, MD, 1995, November 30–December 1, pp. 39–46.

[14] L. Giuri, P. Iglio, A formal model for role-based access control with constraints, Proceedings of Ninth IEEE Computer Security Foundations Workshop, Kenmare, Ireland, 1996, June, pp. 136–145.

[15] R. Simon, M. Zurko, Separation of duty in role-based environments, Proceedings of 10th IEEE Computer Security Foundations Workshop, Rockport, MA, 1997, June, pp. 183–194.

[16] S. Jajodia, P. Samarati, V. Subrahmanian, A logical language for expressing authorizations, Proceedings of the IEEE Symposium on Security and Information Privacy, Oakland, CA, 1997, May, pp. 31–42.

[17] R.S. Sandhu, Role hierarchies and constraints for lattice-based access controls, in: E. Bertino (Ed.), Proceedings of the Fourth European Symposium on Research in Computer Security, Rome, Italy, Springer, Berlin, September 25–29, 1996, Published as Lecture Notes in Computer Science, Computer Security (ESORICS96).

[18] M. Shin, G.-J. Ahn, UML-based representation of role-based access control, Proceedings of Fifth IEEE International Workshop on Enterprise Security, WETICE 2000, NIST, MD, 2000, June 14–16.

[19] J. Rumbaugh, G. Booch, I. Jacobson, The Unified Modeling Language Reference Manual, Addison-Wesley, Reading, MA, 1999.

[20] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, Reading, MA, 1999.

[21] OMG Web site, Unified modeling language notation guide, Version 1.3, September 2000.

[22] J. Warmer, A. Kleppe, The Object Constraint Language: Precise Modeling with UML, Addison-Wesley, Reading, MA, 1999.

[23] IBM Web site, OCL Parser, Version 0.3, 1999.

[24] D. Ferraiolo, R. Sandhu, S. Gavrila, R. Kuhn, R. Chandramouli, Proposed NIST standard for role-based access control, ACM Transactions on Information and Systems Security 4 (3) (2001).