Role-based Authorization Constraints Specification Using Object Constraint Language

Gail-Joon Ahn
Department of Computer Science
University of North Carolina at Charlotte
gahn@uncc.edu

Michael. E. Shin
Department of Information and Software Engineering
George Mason University
eshin@gmu.edu

Abstract

The purpose of access control is to limit the actions on a computer system that a legitimate user can perform. The role-based access control (RBAC) has generated great interest in the security community as a flexible approach in access control. One of important aspects in RBAC is constraints that constrain what components in RBAC are allowed to do. Although researchers have identified useful constraints using formal specification languages such as RCL2000, there still exists a demand to have constraints specification languages for system developers who are working on secure systems development. In this paper we discuss another approach to specify constraints using a de facto constraints specification language in software engineering arena. We use a declarative language, Object Constraints Language (OCL) that is part of the Unified Modeling Language (UML) and has been used in object-oriented analysis and design. We describe how to specify previously identified role-based authorization constraints and future direction of this work is also addressed.

1. Introduction

The role-based access control (RBAC) is a flexible approach that has generated great interest in the security community [1]. RBAC has emerged as a widely accepted alternative to classical discretionary and mandatory access controls [2]. Several models of RBAC have been published and several commercial implementations are 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 actions and responsibilities associate 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 importance of constraints in RBAC has been recognized for a long time, they have not received much attention in the research literature, while role hierarchies have been practiced and discussed at considerable length.

In this paper our focus is on constraints specification, i.e., on how constraints can be expressed. Constraints can be expressed in natural languages, such as English, or in more formal languages. Natural language specification has the advantage of ease of comprehension by human beings, but may be prone to ambiguities. Recently Ahn and Sandhu [3] proposed a formal language called RCL2000 (Role-based constraints specification language 2000) 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 [4].

Next, we may face the following question: *How can we put these useful constraints into system design tasks?* The idea of our approach is to inject constraints specification into a UML-representation of RBAC accomplished by [5]. The constraints in RBAC may be one of the most important components that enforce the principal motivations of RBAC model. Using OCL that has been used to express

constraints in analysis and design as an industrial standard constraints specification language, 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 comparatively convenient for system developers to specify and to understand constraints of RBAC model.

The rest of this paper is organized as follows. In section 2, we briefly describe role-based access control, UML and OCL. Section 3 discusses authorization constraints that are involved in role-based access control. In section 4, we specify previously identified role-based authorization constraints using OCL. Section 5 concludes this paper.

2. Related Technologies

2.1 Role-based Access Control

RBAC has recently received considerable attention as a promising alternative to traditional discretionary (DAC) and mandatory (MAC) access controls (see, for example, [2,5,6,7]). 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 general family of RBAC models called RBAC96 was defined by Sandhu et al [2]. Figure 1 illustrates the most general model in this family. Motivation and discussion about various design decisions made in developing this family of models is given in [2]. Figure 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 mode of access to one or more objects in the system or some privilege to carry out specified actions. 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.

- U is a set of users,
- R is disjoint sets of roles and administrative roles respectively,
- P is disjoint sets of permissions and administrative permissions,
- $UA \subseteq U \times R$, is a many-to-many user to role assignment relation,
- $PA \subseteq P \times R$ is a many-to-many permission to role assignment relation,
- $RH \subseteq R \times R$ is partially ordered role hierarchies (written as $\geq$ in infix notation),
- S is a set of sessions,
- $user : S \rightarrow U$, is a function mapping each session s_i to the single user $user(s_i)$ and is constant for the session's lifetime,
- $roles : S \rightarrow 2^R$ is a function mapping each session s_i to a set of roles $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)\ [(user(s_i), r') \in UA]\}$ (which can change with time) so that session s_i has the permissions $\bigcup_{r \in roles(s_i)} \{p \mid (\exists r^2 \leq r)\ [(p, r^2) \in PA]\}$.

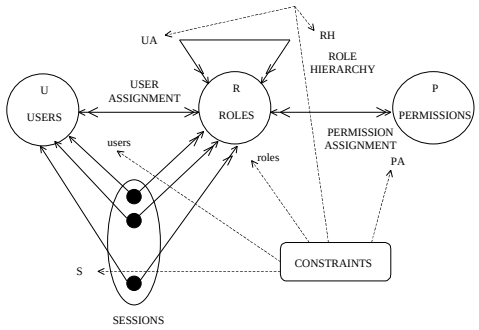
A user can be a member of many roles and a role can have many users. Similarly, a role can have many permissions and the same permissions can be assigned to many roles. Each session relates one user to possibly many roles. Intuitively, a user establishes a session during which the user activates some 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. 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 Unified Modeling Language (UML)

The Unified Modeling Language (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

[8,9,10]. The UML has become a standard modeling language in the field of software engineering.

Figure 1 RBAC Model



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.

2.3 Object Constraint Language

The Object Constraint Language (OCL) [10,11] 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 from Company class to Employee class though 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

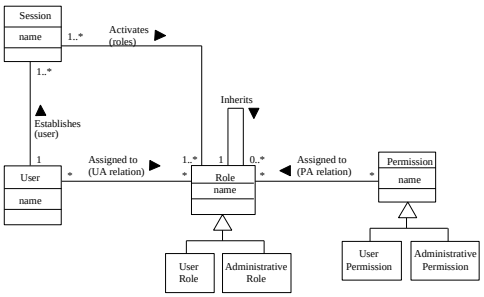
The "**select**" takes an employee from self.employee and evaluates an expression (age > 50) for the employee. If this evaluation result is true, then the employee is in the result set.

3. Role-based Constraints

Constraints are an important aspect of access control and are a powerful mechanism for laying out a higher-level organizational policy. Consequently the specification of constraints needs to be considered. This issue has received surprisingly little attention in the research literature. There is some work such as [12,13] that deal with constraints in the context of role-based access control. This work, however, is preliminary and tentative, and need substantial further development. Most prior work has focused

on separation of duty constraints. Chen and Sandhu [12] suggested how constraints could be specified. Giuri and Iglio [13] 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. Ahn and Sandhu [3] 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*.

Fig.2: Conceptual Class Model for RBAC - entity classes



4. Constraints Specification

The conceptual static model for RBAC is depicted in Figure 2. It contains classes, their attributes, and their relationships [14]. The basic entities are user, role, permission, constraint, and session classes. The role can be specialized to user and administrative roles. The permission can also be specialized to user and administrative permissions. 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.

4.1 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 role-based access control. 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 in terms of UA specifies that one individual cannot have both roles. This constraint on UA can be specified using the OCL expression as follows:

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 enforces above requirements. In other words, 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. In retrospect, this constraint is an obvious property but there is no mention of this property in over a decade of SOD literature. Ahn and Sandhu [3] have recently identified this property. Suppose we have two conflicting permission 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 be also considered. For example, for the process of preparing and approving purchase orders in the purchase 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 Prerequisite constraints

This constraint is based on the concept of prerequisite roles introduced in [2]. 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`)

4.3 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.
Example 7 limits the number of users to be assigned to a role. For example, 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 a t 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

We have shown how authorization constraints for role-based systems can be specified using OCL. The OCL constraints specification can be validated by an OCL parser. Currently the OCL parser [15] does support syntax and type checking. The parsed result can be feedback to OCL constraints specifications. A case study for validating our specifications is currently under investigation.

5. Conclusion

In this paper, we have demonstrated that we can specify role-based authorization constraints using an industrial standard constraints specification language, OCL. We have specified separation of duty constraints, prerequisite constraints and cardinality constraints. As a result, we can utilize constraints identified by a formal language such as RCL2000 when we design and analyze role-based systems. We believe that this work helps system developer understand constraints and requirements on secure systems development. There is room for much additional work with our approach. Validation of OCL specifications and time-based constraints can be studied. A unified way to specify authorization constraints can be investigated so that we can apply our approach to other access control models such as MAC and DAC.

References

[1] James Joshi, Arif Ghafoor, Walid Aref, and Eugene Spafford, "Digital Government security infrastructure design challenges," IEEE Computer, Volume 34, Number 2, pages 66-72, February 2001.

[2] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman, "Role-based access control models," IEEE Computer, Volume 29, Number2, pages 38-47, February 1996.

[3] Gail-Joon Ahn and Ravi Sandhu, "Role-based Authorization Constraints Specification," ACM Transactions on Information and Systems Security, Volume 3, Number 4, November 2000.

[4] Trent Jaeger and Jonathon Tidswell, "Practical Safety in Flexible Access Control Models." ACM Transactions on Information and Systems Security, Volume 4, Number 3, August 2001, to appear.

[5] David Ferraiolo and Richard Kuhn, "Role-based access controls," In Proceedings of 15th NIST-NCSC National Computer Security Conference, pages 554-563, Baltimore, MD, October 13-16 1992.

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

[7] Imtiaz Mohammed and David M. Dilts, "Design for dynamic user-role-based security," Computers & Security, Volume 13, Number 8, pages 661-671, 1994.

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

[9] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language User Guide," Addison Wesley, 1999.

[10] OMG Web site. Unified Modeling Language Notation Guide, Version 1.3, September 2000.

[11] Jos Warmer and Anneke Kleppe, "The Object Constraint Language: Precise Modeling with UML," Addison-Wesley, 1999

[12] Chen, F. and Sandhu, R., "Constraints for role based access control," In Proceedings of the 1st ACM Workshop on Role-Based Access Control, pages 39-46, Gaithersburg, MD, November 30-December 1 1995.

[13] Giuri, L. and Iglio, P., "A formal model for role-based access control with constraints," In Proceedings of 9th IEEE Computer Security Foundations Workshop, pages 136-145, Kenmare, Ireland, June 1996.

[14] Michael E. Shin and Gail-Joon Ahn, "UML-based Representation of Role-Based Access Control," Fifth International Workshop on Enterprise Security (WETICE 2000), Gaithersburg, MD, June, 2000

[15] IBM Web site, "OCL Parser," Version 0.3, 1999