Computer and Information Science

www.ccsenet.org/journal.html

Vol. 2, No. 2

May 2009

Formal Description for an Object-Oriented Role-based Access Control Model

Chungen Xu

Department of Applied Mathematics, Nanjing University of Science & Technology

Jiangsu 210094, China

Tel: 86-25-8431-5877 E-mail: xuchung@mail.njust.edu.cn

Sheng Gong

Library, Nanjing University of Science & Technology

Jiangsu 210094, China

Abstract

Role-based access control(RBAC) is a promising technology for managing and enforcing security in large-scale enterprise-wide system, and we were motivated by the need to manage and enforce the strong access control technology of RBAC in large-scale Web environments. Majority of traditional access control models were passive data-protections, which were not suitable for large and complex multi-user interactive applications. In this paper, we develop a general model to control users' behaviors based on their roles actively, and proposes a framework of well-defined Formal Description for developers to build application-level access control based on users' roles. It ensure that each role is configured with consistent privileges, each actor is authorized to proper roles and then each actor can activate and play his authorized roles without interest conflicts. These formal specifications are consistent and inferable, complete and simplified, abundant and scalable for diversified multi-user applications.

Keywords: Object-Oriented, Formal description, Role, Access control

1. Introduction

Nowadays multi-user applications tend to be large and complex. The functions and structures of large applications are complicated and distributed. And thousands of users perform their diversified duties. All these increase complexity of privilege management, and lead to low control efficiency of users' interaction. Role-Based Access Control(RBAC)(Sandhu, 1996,p.38-47)is a successful technology that will be a central component of emerging enterprise security infrastructures. Moreover, dynamic management and collaborative control are difficult to come into effect in the large-scale Web environment. So the quality of software on the web must get most improvement.

Majority of well-known access control models are passive ones, such as typical subject-object model, that are often implemented by Access Control List (ACL) or access control matrices, and Lattice-based Access Control (LBAC) and many others. These models focus on data-protections at the back-end of applications, and they do not distinguish permission assignment from activation, and further, are not capable of representing or considering any levels of context when processing an operation on an object. Recently, some works address active security models. Such as Task-based Authorization Controls model(Gavrila,1998,p.81-90), and workflow authorization model(Yan, 2000,p.1064-1071) These models consider somewhat context associated with tasks and workflow.

Our focus in this paper is on a general model to control users' behaviors actively based on their roles. Object Technology is used in the model, which is built in Unified Modeling Language (**UML**)(UML Summary Version2.0.,2006). Users' behaviors are abstracted as request services and get results. Thus, user services are protected via a special interface, regardless of complexity of internal implements of these services to simplify management. Role is used to organize behavior specifications of numerous users to reduce the burden of privilege management. Role-Playing is introduced to denote activated role in particular context, and it is modeled as an active class. Every object of Role-Playing runs in particular context, which interact with a user and controls the user's behaviors actively. Users work in collaborative environment. Users use their rights, also perform their obligations. The states, behaviors and lifetime of Role-Playing objects can be monitored or audited to support dynamic management and business

activities control. In this paper, we proposes a framework of well-defined Formal Description for developers to build application-level access control based on users' roles. It ensure that each role is configured with consistent privileges, each actor is authorized to proper roles and then each actor can activate and play his authorized roles without interest conflicts.

2. Model elements and constraint semantics

Figure 1 represents the model. ***Service*** *is an interface, in which user-services are collected, and they are associated with* ***Roles*** *by* ***PA****;* ***Actors*** *are made members of proper roles by* ***SA*** *objects; At runtime actors activate some of their authorized roles with* ***Role-Playing*** *objects created and acquire authorized services.*

As there are complex structural and semantic relations between protected objects in a large application, the model builds a **Service** interface, thereby interface structure, dynamic privilege and privilege implication specifications are specified. Notice that, any interface cannot be instantiated any object, therefor an element of **Service** is a service item rather than an object of it.

Privilege_Authorization (PA) class is an association class between **Role** and **Service**. An instance of it is a tuple (r, s) denoted by pa(r, s), which means role r has permission to a service item s under a context condition specified in its attribute: *context_cond*.

Usually a permission of an operation implies another one, The implication relation is denoted by $s_1 \xrightarrow{cf} s_2$, where *cf* is a function converting the context condition of s1 to that of s2.

A large application may contain numerous and complex associated roles. The model supports role's cardinality constraint, **Role Inheritance** (**RI**), one kind of hierarchy structure, and **Separation of Duties** (**SD**) to signify interest conflict relations.

Status_Authorization (SA) is an association class between **Actor** and **Role**. An instance of it is a tuple (a, r) denoted by sa(a, r), which means the actor a has been authorized to role r.

An instance of ***Separation of Duties (SD)*** *is a tuple(role1, role2) representing conflict of interest between them. There are two subtypes of* ***SD****:* ***Static Separation of Duties (SSD)*** *and* ***Dynamic Separation of Duties (DSD)****.*

At runtime actors activate and play their authorized roles to perform their duties according with privilege specifications. To signify the activated roles, the model introduces **Role-Playing**.

A role-playing is a performance of one role r activated by an actor a in particular context, which is an active object denoted by rp{a, r}. An instance of **Role-Playing**(**RP**) class is a role-playing object running in particular context denoted by *rp{a,r}.context*. A role r is activated, if and only if there exists at least one rp{x, r}.

3. Formal specifications of constraint

As the model uses many associations representing relations of objects, a set of specifications for association is set as global constraints firstly.

3.1 Specifications for a role's privileges to services

F1. A service item in **Service** is either an operation or an interface:

$$\forall s(s \in Service \leftrightarrow s \in Operation \vee s \in Interface)$$

F2. Each operation is declared in particular interface:

$$\forall op \in Operation \rightarrow \exists i \in Interface, op \lhd i$$

Where $op \lhd i$ means that operation op is declared in interface i.

F3. Generalization between interfaces is built in strict partial orders: irreflexive, anti-symmetric and transitive.

F4. A **PA** object is a permission of a role to a service with context condition:

$$\forall r \in Role, s \in Service(\exists pa(r,s) \in PA \leftrightarrow CC(pa(r,s).context\_cond) \rightarrow (r \xrightarrow{cc} s))$$

Where *CC(pa(r, s).context_cond)* means that current context satisfy the condition: *pa(r, s).context_cond*. $r \xrightarrow{cc} s$ means that role r has permission for service *s* under current context *cc*.

F5. If a role has authorization to an operation, then the interface in which the operation declared is authorized:

$$\forall r \in Role, \forall s \in Operation, \forall i \in Interface\ (\exists pa(r,s) \in PA \wedge s \lhd i \rightarrow$$
$$\exists pa'(r,i) \in PA, pa'(r,i).context\_cond = pa(r,s).context\_cond)$$

F6. If a role has authorization to an interface, then the super-interface of it is authorized:

$$\forall r \in Role, \forall s, s' \in Interface\ (\exists pa(r,s) \in PA \wedge \exists g(s',s) \in Generalization \rightarrow$$
$$\exists pa'(r,s') \in PA, pa'(r,s').context\_cond = pa(r,s).context\_cond)$$

F7. If an operation implies another one, and a role has authorization to the operation, then the implied one is authorized with context condition conversion:

$$\forall r \in Role, \forall s_1, s_2 \in Operation((s_1 \xrightarrow{cf} s_2) \wedge \exists pa_1(r, s_1) \in PA \rightarrow$$
$$\exists pa_2(r, s_2) \in PA, pa_2(r, s_2).context\_cond = cf(pa_1(r, s_1).context\_cond))$$

Where $s_1 \xrightarrow{cf} s_2$ denote operation s_1 imply s_2 with context condition conversion by *cf*. $cf(pa_1(r,s_1).context\_cond)$ converts context condition of pa_1 to that of pa_2.

Below inference is derived from F4 and predicate logic:

For role r and service s, if more than one pa(r, s) objects are derived from F5, F6 or F7 or defined directly, then finally only one pa(r, s) exists as a substitute with their context conditions union:

$$\forall r \in Role, \forall s \in Service(\exists pa_1(r, s), pa_2(r, s) \in PA \rightarrow \exists pa(r, s) \in PA,$$
$$pa(r, s).context\_cond = pa_1(r, s).context\_cond \vee pa_2(r, s).context\_cond)$$

3.2 Specifications for associations

F8. Each tuple value in an association class appears at most once:

$$\forall C \in AssoC\ (A, B), \forall t(a, b), t'(a', b') \in C, a = a' \wedge b = b' \rightarrow t = t'$$

where *AssoC(A,B)* is a set of association classes of class A and B.

F9. If a tuple exist in an association class, then its associated objects must exist in the associated classes respectively:

$$\forall C \in AssoC\ (A, B), \forall t(a, b) \in C \rightarrow \exists a \in A \wedge \exists b \in B$$

F10. If class A associate with B, and the minimum of multiplicity of the associate end to A is 1, and if one object b exist in B, then at least one object exists in A linking with b:

$$\forall C \in AssoC(A, B), \min(C->A.multip) = 1 \wedge \forall b \in B \rightarrow \exists a \in A, link(a, b)$$

Where *C->A.multip* is the multiplicity of C association end to A, and *Link(a, b)* means that object a links with b.

F8 and F9 apply to **PA**, **RI, SD, SA** et al. F8 ensures the objects consistency of association classes. F9 and F10 indicate that prerequisite of associated object existence. For example in Figure 1, when a **Role-Playing** object rp{a, r} exist, then a **SA** object sa(a,r) must exist by F10, and then the a exist in **Actor** and the r exist in **Role** by F9. F9 and F10 ensure that the associated object cannot be removed before the associating objects are removed.

3.3 Specifications for roles' relations and authorizations

F11. The **Role Inheritance** (**RI**) relation ri(super-role, sub-role) are strict partial orders: irreflexive, anti-symmetric and transitive.

F12. The **Separation of Duties** (**SD**) relation sd(r1, r2) are irreflexive, symmetric and intransitive.

F13. The **SD** is exclusive with **RI** for any two roles:

$$\forall r_1, r_2 \in Role(\exists sd(r_1, r_2) \in SD \rightarrow \neg \exists ri_1(r_1, r_2) \in RI \wedge \neg \exists ri_2(r_2, r_1) \in RI)$$

F14. If a role inherits another role that is in **SD** with a third role, then the sub-role is in **SD** relation with the third one:

$$\forall r, r_1, r_2 \in Role(\exists ri(r_1, r) \in RI \wedge \exists sd(r_1, r_2) \in SD \rightarrow \exists sd'(r, r_2) \in SD)$$

F15. The **SD** has **Static Separation of Duties** (**SSD**) and **Dynamic Separation of Duties** (**DSD**) as its sub-type:

$$\forall r_1, r_2 \in Role\,(\exists sd\,(r_1, r_2) \in SD \leftrightarrow \exists ssd\,(r_1, r_2) \in SSD \vee \exists dsd\,(r_1, r_2) \in DSD\,)$$

F16. The **SSD** relation is exclusive with **DSD** for any two roles:

$$\forall r_1, r_2 \in Role\,(\exists ssd\,(r_1, r_2) \in SSD \rightarrow \neg \exists\, dsd\,(r_1, r_2) \in DSD\,)$$

F17. If one role inherits another and an actor is authorized for the sub-role, then the actor is also authorized for its super-role:

$$\forall r_1, r_2 \in Role, \forall a \in Actor\,(\exists ri(r_1, r_2) \in RI \wedge \exists sa_1(a, r_2) \in SA \rightarrow \exists sa_2(a, r_1) \in SA)$$

F18. If a role has a privilege to a service, then the role has the privilege override the privileges of its super-roles to the service:

$$\forall r, r' \in Role\,, \forall s \in Service\ (\exists ri\,(r', r) \in RI \wedge \exists pa'\,(r', s) \in PA$$
$$\wedge \exists pa\,(r, s) \in PA \rightarrow CC\,(pa\,(r, s).context\ \_cond\,) \rightarrow (r \xrightarrow{cc} s))$$

F19. If a role has not privileges to a service, then the role inherits the non-overridden privileges of its super-roles. (The formal inheritance algorithm will be discussed elsewhere as its complexity)

F20. The number of authorized actors for any role does not exceed the authorized cardinality of the role:

$$\forall r \in Role(\left|\{sa(a, r) \in SA \mid a \in Actor\}\right| \leq r.authorized\ \_cardinality)$$

F21. An actor cannot be authorized for two roles in **SSD** relation:

$$\forall a \in Actor, \forall r_1, r_2 \in Role(\exists ssd(r_1, r_2) \in SSD \wedge \exists sa(a, r_1) \in SA \rightarrow \neg \exists sa(a, r_2) \in SA)$$

Below inferences can be derived from above specifications:

If one role inherits another, the authorized cardinality of the sub-role cannot exceed that of its super-roles. There is no role inheriting two roles in **SD** relation.

3.4 Specifications for activated roles

F22. The activated number of a role does not exceed its activated cardinality:

$$\forall r \in Role(\left|\{rp\{a, r\} \in Role - Playing \mid a \in Actor\}\right| \leq r.activated\_cardinality)$$

F23. An actor cannot activate two roles in **DSD** relation:

$$\forall a \in Actor, \forall r_1, r_2 \in Role(\exists dsd(r_1, r_2) \in DSD \wedge \exists rp\{a, r_1\} \in Role - Playing \rightarrow \neg \exists rp\{a, r_2\} \in Role - Playing)$$

F24. An actor activates a role no more than once in same context:

$$\forall a \in Actor, \forall r \in Role(\exists rp\{a, r\} \in Role - Playing \rightarrow \neg \exists rp'\{a, r\} \in Role - Playing, rp'\{a, r\}.context = rp\{a, r\}.context)$$

F25. If an actor activate a sub-role in a context, then its super-role is activated in the same context:

$$\forall a \in Actor, \forall r_1, r_2 \in Role(\exists ri(r_1, r_2) \in RI \wedge \exists rp\{a, r_2\} \in Role - Playing \rightarrow \exists rp'\{a, r_1\} \in Role - Playing, rp'\{a, r_1\}.context = rp\{a, r_2\}.context)$$

4. Conclusion

Formal Description specifications for application-level access control are challenging and imperative works. This paper provides a novel framework of formal specifications, in which contain formal, consistent and inferable constraints. They are more complete and simplified than traditional ones, but also they are general and scalable for a wide range of multi-user interactive computing and distributed information–processing systems. The concept of usage control (UCON)(Ferraiolo,2001,224-274.and Zhang, 2008,p.1-36.)[1]is an important access control system after RBAC and introduced as a unified approach to capturing a number of extensions for access control models and systems. In UCON, a control decision is determined by three aspects: authorizations, obligations and conditions. We will give the formal specification of UCON in the future.

References

Dewan,D. and Shen,H. (1998). Controlling access in multiuser interface, *ACM Transactions on Computer-Human Interaction*, Volume 5, No. 1, 34-62.

Enterprise JavaBeans Developers Guide (1999). [Online] *http:/ /java.sun.com /products /ejb/ devguide*. (Aug., 1999)

Ferraiolo, D. Sandhu, R. (2001). Proposed NIST Standard for Role-Based Access Control*, ACM Trans. on Information and System Security (TISSEC)*, 4(3) Aug. 2001, 224-274.

Gavrila,S.I. and Barkley,J.F. (1998). Formal specification for role based access control user/role and role/role relationship management. RBAC '98. *Proceedings of the third ACM workshop on Role-based access control,* Oct. 22-23, 1998, Fairfax, VA, 81-90.

Jonathan D. M. (1998). Control principles and role hierarchies, RBAC '98. *Proceedings of the third ACM workshop on Role-based access control*, Oct. 22-23, 1998, Fairfax, VA, 63-69.

Sandhu ,R.S., Coyne,E.J., Feinstein,H.L., and Youman,C.E. (1996). Role-Based Access Control Models. *IEEE Computer*, Volume 29, Number 2, 38-47.

UML Summary Version2.0. (2006). UML Semantics Version2.0,UML Notation Guide. [Online] Available at: *http://www-01.ibm.com/software/rational/uml/ (Aug.,2006).*

Yan, H., Zhang, H. and Xu, M.W. (2000). Object modeling and implementation of access control based on role, *Chinese Journal of Computers*, v 23, n 10, 1064-1071.

Zhang, X.,Nakae, M., Covington, M. and Sandhu, R. (2008). Toward a Usage-Based Security Framework for Collaborative Computing Systems, *ACM Trans. on Information and System Security. (TISSEC)*, Volume 11, Number 1, 1-36.

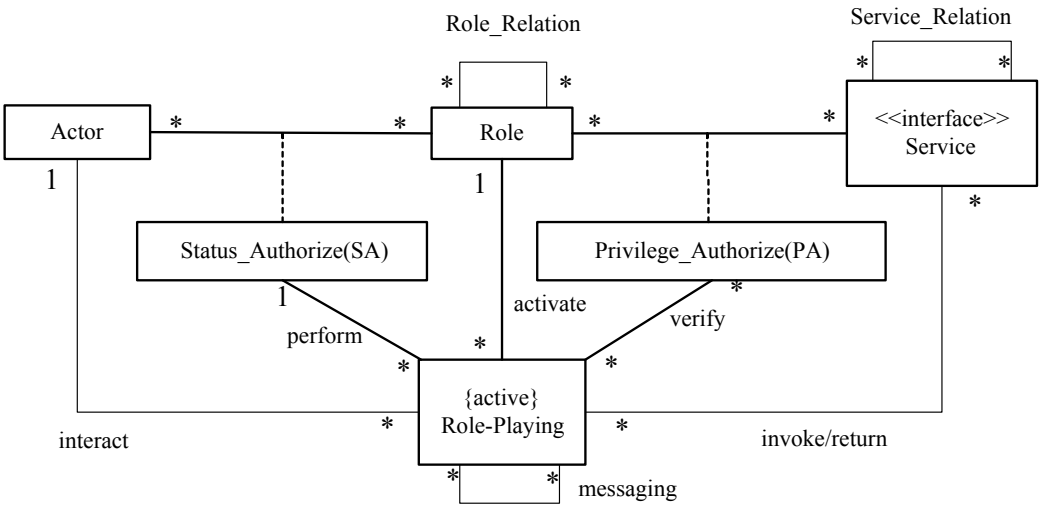

Figure 1. The class diagram of the model