ISSN 1009-3095 *Journal of Zhejiang University SCIENCE* V.3, No.3, P.292-297, July-Aug., 2002
http://www.periodicals.com.cn; http://www.zju.edu.cn/English

http://www.zjupress.com; http://lib.zju.edu.cn/eindex.htm; jzu_s@mail.hz.zj.cn

# Consistency maintenance for constraint in role-based access control model*

HAN Wei-li (韩伟力)†, CHEN Gang (陈刚), YIN Jian-wei (尹建伟), DONG Jin-xiang (董金祥)

(*State Key Laboratory of Computer Aided Design and Computer Graphics, Zhejiang University, Hangzhou 310027, China*)

†E-mail: weili_han@cs.zju.edu.cn

Received June 8, 2001; revision accepted Sept. 20, 2001

**Abstract:** Constraint is an important aspect of role-based access control and is sometimes argued to be the principal motivation for role-based access control (RBAC). But so far few authors have discussed consistency maintenance for constraint in RBAC model. Based on researches of constraints among roles and types of inconsistency among constraints, this paper introduces corresponding formal rules, rule-based reasoning and corresponding methods to detect, avoid and resolve these inconsistencies. Finally, the paper introduces briefly the application of consistency maintenance in ZD-PDM, an enterprise-oriented product data management (PDM) system.

**Key words:** Consistency maintenance, Role-based access control, Product data management, Constraint

**Document code:** A **CLC number:** TP 309.2

## INTRODUCTION

Role-based access control (RBAC) began with multi-user and multi-application on-line systems pioneered in the 1970s and recently have became a very important area of system security (Sandhu et al., 1996; Schaad et al., 2001; Zhang et al. 2001). Constraints can enormously improve system security. But so far few authors have discussed the consistency maintenance for constraint in RBAC (Chen et al., 1995; Lupu et al., 1997a; Ribeiro et al., 2000; Sandhu et al. 1998).

Role-based access control models (RBAC96) (Sandhu et al., 1996; Sandhu et al. 1998) can well simplify the specification and management of authorization policy, namely what actions the controlling subjects are permitted to perform on the controlled objects. But RBAC96 did not deal with the constraint among roles very well, especially in consistency maintenance for constraint among roles. RBAC96 only briefly introduced the constraint among roles in RBAC2, and mentioned in a few words the importance of constraint among roles. Consistency maintenance was neglected in RBAC96, although it was very important.

Some researches analyzed the consistency in security policy and presented a few methods such as logic reason to resolve inconsistency in security policy (Lupu et al., 1997b). The inconsistency they discussed originated from obliged, permitted and forbidden access. They did not come down to the inconsistency originated from constraints among roles. But it led us to explore the feasibility of maintaining consistency by logic reasoning.

Shen et al. (1994) conducted a few researches on consistency maintenance for constraints among roles. They discussed inheritance and conflict resolution rules of subject and right. The concept of subject in their paper was somewhat like the concept of roles in this paper. But their paper only dealt with conflicts originated from positive, negative and take role (take role is similar to the constraint of role hierarchy in RBAC96). It did

* Project (No. [1998] 2077) supported by the State Development Planning Commission of China

not deal with other constraints, which were more important for RBAC, and it had not explored yet how to maintain consistency.

The inconsistency among roles can be detected, avoided and resolved by defining rules and logic reasoning. This paper introduces a series of methods to maintain consistency by defining rules and logic reasoning. These methods include inconsistency detection, inconsistency avoidance and inconsistency resolution. Our research is based on the application of ZD-PDM, an enterprise-oriented product data management (PDM) system. We hope that these methods can simplify the task of system security official (SSO), and be applied to other distributed, multi-user systems.

The organization of this paper is as follows. Section 1 introduces briefly the current researches in RBAC and consistency maintenance. Section 2 formally defines different types of constraint in our application. Section 3 discusses the particular inconsistency in our application. Section 4 discusses the corresponding solution to consistency maintenance. The final section introduces briefly the application of consistency maintenance in ZD-PDM.

## CONSTRAINT AMONG ROLES

Constraint is an important aspect of RBAC. It is also a powerful mechanism in laying out higher-level organizational policy. A common example is that of mutually exclusive roles. Before defining the types of constraint in our application, we are going to predefine some important concepts.

**Definition 0** The definition of the following elements are quoted from RBAC96

$U$, $R$, $P$, and $S$ (users, roles, permissions and sessions respectively),

$PA \subseteq P \times R$, a many-to-many permission to role assignment relation,

$PA \subseteq U \times R$, a many-to-many user to role assignment relation,

Roles: $S \rightarrow 2^R$, a function mapping each session $s_i$ to a set of roles.

Secondly, we define the main types of constraint.

**Definition 1** Mutually exclusive roles (RME)

Mutually exclusive roles can be subcategorized into two types: static mutually exclusive roles and dynamic mutually exclusive roles.

Two roles ($r1$ and $r2$) are static mutually exclusive, if and only if $u \times r1 \Rightarrow 7(u \times r2)$. We denote it by ($r1$, $r2$, $RSME$).

Two roles ($r1$ and $r2$) are dynamic mutually exclusive, if and only if $r1 \in roles(s) \Rightarrow \forall s_i \in S : r2 \notin roles(s_i)$. We denote it by ($r1$, $r2$, $RDME$).

**Definition 2** Role inclusion (RI)

$r1$ includes $r2$, if and only if $(u, r1) \Rightarrow (u, r2)$. We denote it by ($r1$, $r2$, $RI$).

**Definition 3** Prerequisite constraint (RP)

$r1$ is prerequisite to $r2$, if and only if the assignment of $r1$ to a user is a precondition to the assignment of $r2$ to the same user. We denote it by ($r1$, $r2$, $RP$).

**Definition 4** Role hierarchy (RH)

Role hierarchy is a very important constraint among roles. It originated from organizational hierarchy. Defined roles are embodied in an organization that is generally represented in a hierarchical form, so the existence of role hierarchy is widespread. By referring to organizational hierarchy, role hierarchy gives SSO an intuitionistic view in PA, and simplifies the authorization. It is a special instance of RI constraint in RBAC in a sense.

If $r1$ inherits $r2$, which we denote by ($r1$, $r2$, $RH$), then $r1$ inherits all permissions of $r2$, including the permissions inherited by $r2$ from other roles.

The constraints we define above are those between two roles (mutual-constraint). Now we come to self-constraint, which exists only in single role, including cardinality and state constraints.

**Definition 5** Cardinality constraint (RC)

Cardinality constraint describes maximally how many users can be assigned to a role. We denote it by ($r$, $n$, $RC$) ($n$ stands for the maximum number).

**Definition 6** State constraint (RS)

State constraint describes the state of controlled object under which the role r can be activated. We denote it by ($r$, $s$, $RS$).

These constraints imply some other constraints, which can be reasoned by defined rules. Now we will present these rules to get implicit constraints.

**Rule 1** $\forall r1, r2, r3 : (r1, r2, RI) \wedge (r2, r3, RI) \Rightarrow (r1, r3, RI)$

**Rule 2** $\forall r1, r2, r3: (r1, r2, RP) \wedge (r2, r3, RP) \Rightarrow (r1, r3, RP)$

**Rule 3** $\forall r1, r2: (r1, r2, RH) \Rightarrow (r1, r2, RI)$

**Rule 4** $\forall r1, r2, r3: (r1, r2, RME) \wedge (RME \in \{RSME, RDME\}) \wedge (r3, r2, RI) \Rightarrow (r1, r3, RME)$

Some rules can be derived from these rules by logic reasoning based upon the above-mentioned rules, such as rule 34 can be derived from rule 3 and rule 4:

**Rule 3-4** $\forall r1, r2, r3: (r1, r2, RME) \wedge (r3, r2, RH) \Rightarrow (r1, r3, RME)$

**Definition 7** Constraint Assignment (CA) $CA \subseteq R \times R \times MT \cup R \times N \times CC \cup R \times S \times SC$, ($MT = \{RDME, RSME, RI, RP, RH\}, CC = \{RC\}, SC = \{RS\}$).

## INCONSISTENCY AMONG CONSTRAINTS AND CORRESPONDING SOLUTION

### The causes and reasoning of inconsistency among constraints

The following situations are the main causes of inconsistency among constraints.

(a) Circular prerequisite

A closed chain of prerequisite exists, in which each role is prerequisite to other roles. (a) in Fig. 1 shows a simple example of circular prerequisite, and rule 5 is its formal specification.

**Rule 5** $(r1, r2, RP) \wedge (r2, r1, RP) \Rightarrow FALSE$

(b) Circular hierarchy

A closed chain of hierarchy exists, in which each role inherits all permissions of other roles. (b) in Fig. 1 shows a simple example of circular hierarchy, and rule 6 is its formal specification.

**Rule 6** $(r1, r2, RH) \wedge (r2, r1, RH) \Rightarrow FALSE$

(c) Prerequisite and hierarchy conflict

When $r1$ inherits all permissions of $r2$, then r1 can no longer be a prerequisite to $r2$. If $r1$ is a prerequisite to $r2$, like (c) in Fig. 1, the third inconsistency occurs. Rule 7 is its formal specification.

**Rule 7** $(r1, r2, RP) \wedge (r2, r1, RH) \Rightarrow FALSE$

(d) Conflict between mutually exclusive constraints and other mutual-constraints

When $r1$, $r2$ are mutually exclusive (both static mutually exclusive and dynamic mutually exclusive), then the constraint between them cannot be defined as any one of the following constraints (prerequisite constraint, role inclusion, role hierarchy). (d) Fig. 1 provides a simple example of this inconsistency, and rule 8 is its formal specification.

**Rule 8** $(r1, r2, RME) \wedge (r2, r1, RO) \wedge (RME \in \{RSME, RDME\}) \wedge (RO \in \{RP, RI, RH\}) \Rightarrow FALSE$

(e) Inconsistency of cardinality

When we define several times maximally how many users can be assigned to a role, the inconsistency of cardinality will possibly occur. For example, suppose at first we define a maximum of two users can take $r1$, and later on we redefine a maximum of four users can take $r1$ . Then the system may not know maximally how many users can take $r1$, two or four. (e) in Fig. 1 provides an example of this inconsistency, and rule 9 is its formal specification.

**Rule 9** $(r, n1, RC) \wedge (r, n2, RC) \wedge (n1 \neq n2) \Rightarrow FALSE$

(f) Multi-state assignment

This is similar to inconsistency of cardinality. When we define several times the state constraint for the same role, inconsistency of multi-state assignment will possibly occur. (f) in Fig. 1 provides an example of this inconsistency, and rule 10 is its formal specification.

**Rule 10** $(r, s1, RS) \wedge (r, s2, RS) \wedge (s1 \neq s2) \Rightarrow FALSE$

(g) Reasoning about inconsistency

In fact, the above rules are only the primary rules of inconsistency. More inconsistency rules can be derived from rule 1-4; for example, rule 35 can be derived from rule 3 and rule 5. In Fig. 1 (g) (see next page) is an example of derived inconsistency, and rule 35 is its formal specification.

**Rule 3-5** $(r1, r2, RP) \wedge (r2, r3, RP) \wedge (r3, r1, RP) \Rightarrow FALSE$

### Inconsistency detection

To resolve the problem of inconsistency among constraints, we must know which in-

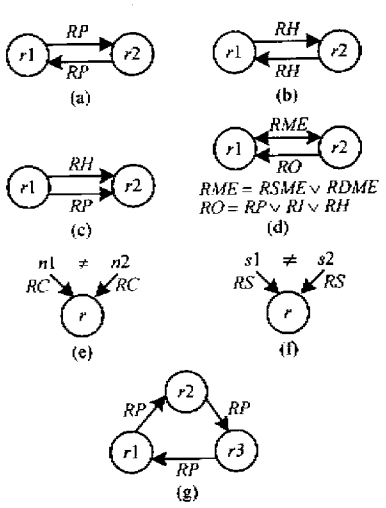


**Fig. 1 The causes and reasoning of inconsistency**

(a) Circular prerequisite;
(b) Circular hierarchy;
(c) Prerequisite and hierarchy conflict;
(d) Conflict between mutually exclusive; constraint and other mutual-constraint;
(e) Inconsistence of cardinality;
(f) Multi-state assignment;
(g) Reason about inconsistency.

consistency exists in the system. So the arithmetic of inconsistency detecting must be given.

The arithmetic of inconsistency detecting is designed in order to detect the above-mentioned inconsistencies by means of logic reasoning by primary inconsistency rules (Rule 5 – 10) and constraint transitive rules (Rule 1 – 4). In Fig. 2, we illustrate the arithmetic in detail by detecting the circular prerequisite.

**Arithmetic 1 DetectCP**
**INPUT**: Two roles ($r1$, $r2$).
**OUTPUT**: TRUE or FALSE. If circular prerequisite exists between $r1$ and $r2$, the arithmetic outputs *FALSE*, else outputs *TRUE*.
**STEP 1**: according to **rule 5** directly, if there exist ($r1$, $r2$, *PR*) and ($r2$, $r1$, *PR*), return *FALSE*; else
**STEP 2**: reasoning by **rule 2** on role-set, and get all direct and indirect prerequisite role sets $RPr1$ of $r1$ and $PRr2$ of $r2$; then
**STEP 3**: if $r1 \in RPr2 \wedge r2 \in RPr1$, return *FALSE*; else,
**STEP 4**: return *TRUE*.

**Fig. 2 Detect circular prerequisite**

It is similar to design DetectCH (detect circular hierarchy), DetectPH (detect prerequisite and hierarchy conflict), DetectMEO (detect conflict between mutually exclusive constraints and other mutual-constraints), DetectCAR (detect inconsistency of cardinality), DetectMS (detect multi-state assignment). We put forward the whole arithmetic frame to detect inconsistency in Fig. 3.

**Arithmetic 2 Detect_InCons**
**INPUT**: Two roles ($r1$, $r2$).
**OUTPUT**: The types of inconsistency. If $r2$ is NULL, and inconsistency exists in $r1$, or $r2$ isn't NULL and inconsistency exists between $r1$ and $r2$, the arithmetic outputs the value of enum type of inconsistency, else outputs *NONC*.
**STEP 0**: *fRet* = *NONC*;
**STEP 1**: if ($r2$ = *NULL* AND ! DetectCAR($r1$)) *fRet* |= *CAR*;
**STEP 2**: if ($r2$ = *NULL* AND ! DetectMS($r1$)) *fRet* |= *MS*;
**STEP 3**: if ($r2$ ! = *NULL* AND ! DetectCP($r1$, $r2$)) *fRet* |= *PC*;
**STEP 4**: if ($r2$ ! = *NULL* AND ! DetectCH($r1$, $r2$)) *fRet* |= *HC*;
**STEP 5**: if ($r2$ ! = *NULL* AND ! DetectPH($r1$, $r2$)) *fRet* |= *PH*;
**STEP 6**: if ($r2$ ! = *NULL* AND ! DetectMEO($r1$, $r2$)) *fRet* |= *MEO*;
**STEP 7**: return *fRet*;

**Fig. 3 Detect inconsistency**

**Inconsistency avoidance**

When we define new constraints among roles, or modify an existing constraint in a system, inconsistency among constraints may occur. We can rollback the definition or modification when the inconsistency happens, or stop defining or modifying when inconsistency occurs. This strategy is called inconsistency avoidance.

The strategy of inconsistency avoidance is, in a few words, to design a system in which the inconsistency can be excluded. It is realized by rolling back the constraint definition or constraint modification when the system detects inconsistencies. It is an important way to keep inconsistency out of a system. Of course, the system itself can provide detailed information on inconsistency to SSO. When inconsistencies occur, SSO can adjust the constraint assignment or security policies to resolve them.

**Inconsistency solution rules**

Inconsistency avoidance cannot be used to resolve the existence of inconsistency among constraints. So finding out a method to resolve the possible existing inconsistency is necessary.

We define corresponding solution rules to resolve the existing inconsistencies.

**Solution rule 1** Constraint defined explicitly is superior to constraint implied in system.

When we detect a special constraint, if it is defined explicitly, we are not going to reason by rule 1 − 4, and return the detected result that indicates the existence of constraint. This rule resolves the inconsistency when one constraint is explicitly defined and the others are implied in the system.

**Solution rule 2** Define the priority of constraint to resolve inconsistency. When the existing inconsistencies are all explicit or implied, we select the constraint that is superior to other constraints.

We define $RME > RP > PH > RI$. For example, like rule 8, when constraints $RSME$ and $RH$ are explicitly defined between $r1$ and $r2$, we consider that $r1$ and $r2$ are $RSME$. Rule 8' is its formal specification.

**Rule 8'** $(r1, r2, RME) \wedge (r1, r2, RO) \wedge (RME \in \{RSME, RDME\}) \wedge (RO \in \{RP, RI, RH\}) \Rightarrow (r1, r2, RME)$

(Note: RME and RO must be all explicit or all implied).

**Solution rule 3** When inconsistency of cardinality occurs, we select a small number. Rule 9' is its formal specification.

**Rule 9'** $(r, n1, RC) \wedge (r, n2, RC) \wedge (n1 > n2) \Rightarrow (r, n2, RC)$

Solution rules 1 − 3 can resolve many inconsistencies encountered in check permission and other arithmetic. They can not only provide a more effective authorization, especially in distributed authorization, but also simplify the task of SSO.

## APPLICATION

We have applied the methods provided in this paper in ZD-PDM. They can deal well with inconsistency of constraint among roles when we control the access of controlled object. They can also simplify the management of SSO. Fig. 4 (see next page) shows the management role, management constraint and position of consistency maintenance in ZD-PDM.

In ZD-PDM, the types of constraints include $RSME$, $RDME$, $RI$, $RP$, $RH$, $RC$, $RS$, and the types of inconsistency among constraints include circular prerequisite, circular hierarchy, prerequisite and hierarchy conflict, conflict between mutually exclusive constraints and other mutual-constraints, inconsistency of cardinality, multi-state assignment. The methods of consistency maintenance, which we introduced above, have been applied in user assignment, constraint assignment and access control. They can detect and avoid most inconsistencies while SSO does user assignment and constraint assignment, and resolve many inconsistencies while the system checks in access control.

## CONCLUSIONS

This paper discusses the defects of current models of constraint among roles. It indicates that current researches did not deal well with constraint among roles; and that they did not provide methods to deal with inconsistency among constraints.

Based on application in PDM, the paper presents rules defining and rule-based reasoning to resolve the inconsistency among constraints in system. It holds that the inconsistency can be avoided by rolling back the constraint assignment when the system detects inconsistency, and provides some solution rules to resolve this problem. These rules have well resolved the inconsistency encountered in ZD-PDM.

The method, which resolves the inconsistency in ZD-PDM, can also be applied to other distributed, multi-user systems. We will do researches on the following problems.

1. Research in distributed constraint assignment, and provide more effective methods to resolve inconsistency,

2. Research in interoperation of the con-

straints in a distributed, heterogeneous system.

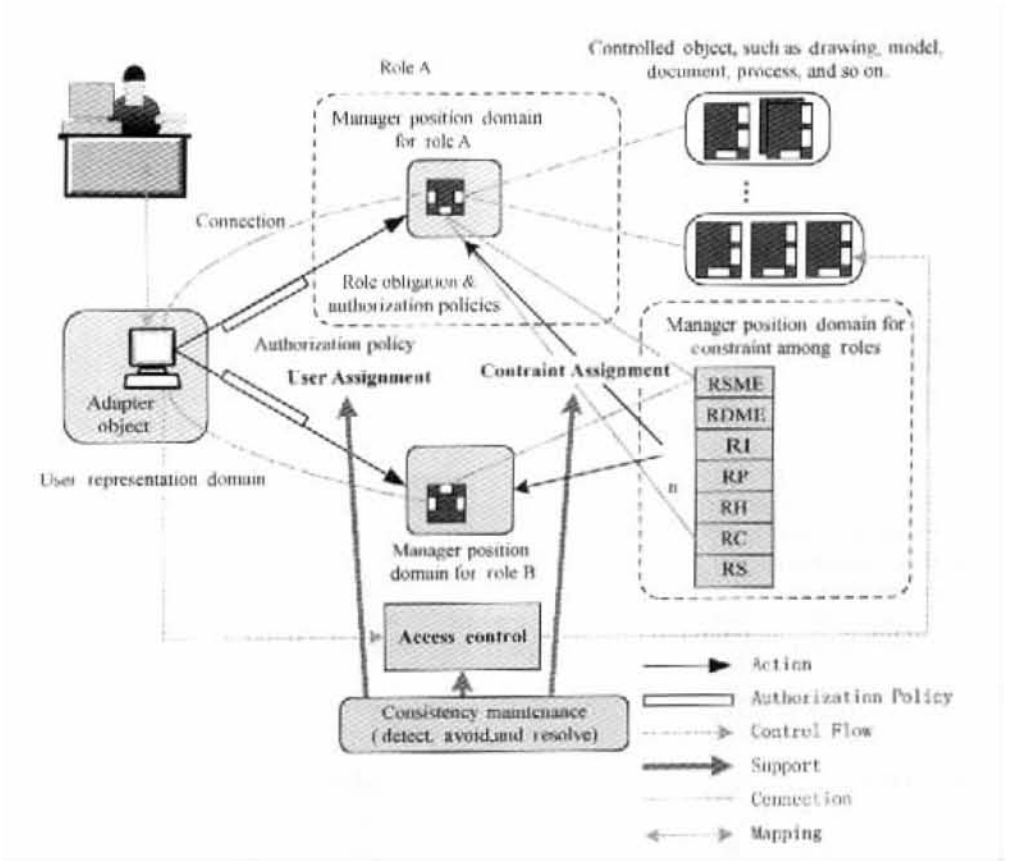


**Fig. 4 Roles management, constraints management and position of consistency maintenance in ZD-PDM**

## References


Chen Fang, Sandhu, R. S., 1995. Constraints for Role-Based Access Control. *In*: 1st ACM Workshop on Role-Based Access Control. ACM, p. 39 – 46.

Laurence Cholvy, Frédéric Cuppens, 1997. Analyzing Consistency of Security Policies. *In*: Proceedings of the 1997 IEEE Symposium on Security and Privacy. IEEE Press, Oakland, CA, USA. p. 103 – 112

Lupu, E., Sloman, M., 1997. Conflict Analysis for Management Policies. IFIP/IEEE International Symposium on Integrated Network Management (IM formerly known as ISINM 97), Chapman & Hall, San Diego.

Lupu, E., Sloman, M., 1997a. A Policy Based Role Object Model. Proceedings of the 1st IEEE Enterprise Distributed Object Computing Workshop (EDOC'97), Gold Coast, Australia, p. 36 – 47.

Lupu, E., Sloman, M., Yialelis, 1997b. Policy Based Roles for Distributed Systems Security. HP-Openview University Associated (HP-OVUA) Plenary Workshop, Madrid.

Ribeiro, C., Zúquete, A., Ferreira, P. et al, 2000. Security Policy Consistency. First Workshop on Rule-based Constraint Reasoning and Programming (CL2000), Imperial College, London, UK.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., 1996. Role-Based Access Control Models. *IEEE Computer*, **29**(2): 38 – 47.

Sandhu, R. S., 1998. Role-Based Access Control. *In*: Advances in Computers, volume 46. Academic Press.

Schaad, A., Moffett, J., Jacob, J., 2001. The Role-Based Access Control Sytem of a European Bank: A Case Study and Discussion. Sixth ACM Symposium on Access control models and technologies. Chantilly, VA USA, p. 3 – 9.

Shen Honghai, Prasun Dewan, 1994. Access Control for Collaborative Environments. In: Turner J. Kraut R eds. Proceedings of the ACM CSCW'92 Conference on Computer Supported Cooperative Work. New York: ACM Press, p. 51 – 58.

Zhang Dianlong, Lukhaub, H., Zorn, W., 2001. A Role-Based Access Control Model and Implementation for Data-Centric Enterprise Application. Proceedings of Third International Conferences, ICISC 2001. Xi'an, China, p. 316 – 327.