CERIAS Tech Report 2002-33

Petri-net Based Modeling for Verification of RBAC Policies

by Basit Shafiq, James B. D. Joshi, Arif Ghafoor

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907

Petri-net Based Modeling for Verification of RBAC Policies

Basit Shafiq, James B. D. Joshi, Arif Ghafoor
Center of Education and Research in Information Assurance and Security (CERIAS)
and
School of Electrical and Computer Engineering,
Purdue University, West Lafayette, IN 47906

Abstract

Role based access control has emerged as a promising new approach to security for advanced applications because of the several benefits it provides. However, most of the research efforts in this area has been focused towards the specification and modeling of RBAC systems. The crucial issue of verification of role based access control policies has not been adequately investigated in the literature. In this paper, we propose a colored Petri-net based policy specification and analysis framework for an RBAC model. The Petri-net model can capture all the cardinality and separation of duty constraints that have been previously identified in the literature. Moreover, the model also allows specification of the precedence and dependency constraints that we introduce in this paper. We use the Petri-net reachability analysis technique for verifying correctness of RBAC policies. A set of consistency rules is used as the basis of detecting undesirable states representing erratic behavior of the system due to the flaws in policy specification. The analysis framework can be used by security administrators to generate correct specification iteratively.

This work was supported by the Center for Education and Research in Information Assurance and Security (CERIAS) *and NSF under grant* IIS-0209111

1 Introduction

Role based access control (RBAC) has emerged as a promising alternative to traditional discretionary and mandatory access control (DAC and MAC) models, which have some inherent limitations [Jos01a, Osb00]. Several beneficial features such as policy neutrality, support for least privilege, efficient access control management, are associated with RBAC models [Jos01a, San96]. The concept of role is associated with the notion of functional roles in an organization, and hence RBAC models provide intuitive support for expressing organizational access control policies [Fer93]. Hence, RBAC models are better suited for handling access control requirements of diverse organizations and emerging, advanced applications such as e-commerce, healthcare-systems, etc. [Jos01a]. Furthermore, use of role hierarchies and grouping of objects into object classes based on responsibility associated with a role makes the management of permissions very easy. RBAC constraints allow expressing user-specific access control policies, and DAC and MAC policies, thus, increasing the applicability of RBAC models. In particular, many separation of duty (SoD) constraint can be easily specified to cater to the access control needs of many commercial applications [Ahn00, Nya99]. By configuring the assignment of the least set of privileges from a role set assigned to a user when he activates the role, inadvertent damage can be minimized in a system. RBAC models have also been found suitable for addressing security issues in the Internet environment, and show promise for newer heterogeneous multi-domain environments that raise serious concerns related to access control across domain boundaries [Bar97, Jos01a, Jos01b].

RBAC has been widely researched, primarily because of its relevance and the benefits it provides as mentioned above, and has been extended in by several researchers [San96, Nya99, Gav98]. One such crucial extension is an RBAC model with temporal constraints, which was proposed in Temporal RBAC model [Ber01] and later generalized into Generalized-TRBAC [Jos01b]. GTRBAC distinguishes among various states of a role - such as *disabled*, *enabled* and *active* states - and extends the notion of RBAC events introduced in TRBAC. An event-based of TRBAC approach is particularly suitable for time-based access control requirements and for dynamic access control models [Jos01b, Jos02].

Although RBAC has today reached a good level of maturity, there are still relevant RBAC-based policy specification and analysis issues that have not been addressed adequately in the literature. Koch *et. al*. [Koc02] present a graph based RBAC model aimed at the analysis of the RBAC policies based on graph transformations. Nyanchama *et. al.* [Nya99] present a graph based RBAC model and focuses primarily on the management of the role-graphs and conflicts among roles using graph algorithms. Jaeger *et. al.* [Jae01] present a graphical approach to capture a generic typed access control model and express RBAC as its special case. These approaches are primarily static in nature and do not adequately take into account various authorization related RBAC events that can be allowed in a system non-deterministically.

In this paper, we combine the event-based approach taken in GTRBAC with the Petri-net based modeling approach to develop a framework for modeling and analysis of non-temporal RBAC policies. The approach is particularly novel because of the intuitive way in which Petri-nets capture both system states and events, thus allowing state-based analysis for policy verification and assisting in deriving an event based execution model of an RBAC system in order to ensure safety. Furthermore, several formal tools and techniques are available for Petri-nets that can be utilized to carry out relevant analysis for correctness verification of specification.

An essential feature of RBAC is that it allows specification of various SoD constraints that are needed in many commercial applications [Ahn00, Fer93]. SoD constraints aim at eliminating any possibility of users committing a fraud in a system by preventing a user from acquiring enough access privileges to commit fraud. Several SoD constraints have been identified in the literature. In this paper, we propose some new SoD, precedence and dependency constraints.

The paper is organized as follows. In section 2, we present relevant background on RBAC models on which we build our Petri-net framework. In section 3, we provide a classification of consistency rules. The colored Petri-net model of RBAC and the policy analysis framework are presented in section 4. In section 5, we discuss related work. Section 5 concludes the paper and provides future directions.

2 Overview of Role Based Access Control

In this section, we provide relevant background on the RBAC and GTRBAC models that we refer to in this paper. The RBAC model as proposed by Sandhu *et. al.* in [San96], currently being used as the basis for the NIST RBAC model, consists of the following four basic components: a set of users `Users`, a set of roles `Roles`, a set of permissions `Permissions`, and a set of sessions `Sessions`. A user is a human being or a process within a system. A role is a collection of permissions associated with a certain job function within an organization. A permission is an access mode that can be exercised on a particular object in the system. A session relates a user to possibly many roles. When a user logs in the system he establishes a session by activating a set of enabled role that he is entitled to activate at that time. If the activation request is satisfied, the user issuing the request obtains all the permissions associated with the role he has requested to activate. On `Roles`, a hierarchy is defined, denoted by $\geq$. If $r_i \geq r_j$, r_i, $r_j \in$ `Roles` then r_i inherits the permissions of r_j. In such a case, r_i is a senior role and r_j a junior role.

The RBAC model does not explicitly model different states of a role and hence do not capture various events that are typical of an RBAC system. Such event based approach was used by Bertino *et. al.* in TRBAC model [Ber01] and later extended by Joshi *et. al*. in GTRBAC [Jos01b], primarily to capture the different transitional actions needed in the context of temporal constraints. The GTRBAC model provides a temporal framework for specifying an extensive set of temporal constraints and uses a language-based framework [Jos01b]. GTRBAC allows various types of temporal constraints such as *temporal constraints on role enabling/disabling*, *temporal constraints on user-role and role-permission assignments/de-assignments*, *role activation-time constraints, etc*. These constraints are useful in capturing the dynamic behavior of systems that employ RBAC.

Fig. 1 highlights four key components of GTRBAC model that include user-role assignment/de-assignment, role-permission assignment/de-assignment, role enabling/disabling, and role activation/deactivation. The latter two events allows one to define fine-grained access constraints based on system events as well as states. Such events, in particular, are useful in describing various precedence and dependency constraints. For instance, a role can be enabled only if some other roles are enabled, defining a precedence relation between them. Traditional RBAC model does not explicitly capture such events and hence is not convenient for expressing such precedence and other dependency constraints.

In this paper, we use an event based approach to model RBAC. This event based RBAC model corresponds to a selected set of temporal constraints of GTRBAC. The motivations for this are two fold:

1. Such an event-based realization of traditional RBAC system allows capturing the dynamic properties of the system that can be used to verify the correctness of an RBAC specification. For example, as we show later, although we start with a consistent initial specification, it is possible that as users are *assigned* and *de-assigned* (*assignment* and *de-assignment* events) to roles and their requests for activating roles granted, we may reach an undesirable state, indicating some potential flaw in the initial specification or weakness in policy specification with respect to the set of consistency properties for the system.
2. Our future goal is to extend the proposed Petri-net modeling framework to model the GTRBAC system and then, to develop techniques for validating and verifying the correctness properties of GTRBAC Policies pertaining to the four components of Fig. 1.

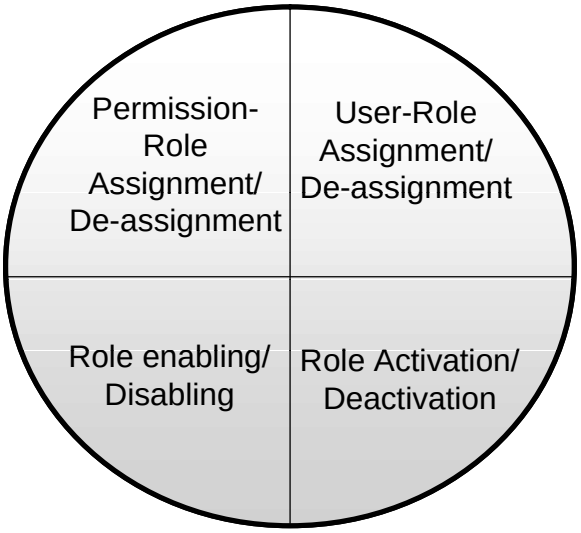

Figure 1 Components corresponding to event set of GTRBAC

3 A verification model for RBAC

Our main objective in this paper is to model RBAC using a Petri net based framework and then use this framework to verify the correctness of the underlying security policies.

3.1 Policy considerations in RBAC

A policy is a set of rules that defines the expected behavior of the system employing that policy. The system is said to be in conformance with the underlying policy if every state of the system can be deduce from the set of rules/axioms comprising the policy. An inconsistent state or erratic system behavior can be attributed to a potential flaw in the policy specification. This flaw may be because of inconsistency in the policy itself or because of incompleteness. An inconsistent policy is the one in which two or more rules from a given set of rules comprising the policy contradict each other. Incompleteness implies that the given set of rules defining the policy is not sufficient to capture all states of the system. In this context, security verification can be stated as the process of proving that the properties or rules specified in a security policy are enforced in the information system.

Gavrila *et. al*. [Gav98] state a set of consistency rules for information systems employing RBAC as an access control mechanism. These rules are defined as consistency rules because the information system is expected to satisfy these rules in all possible states it may take. These consistency rules although specify most of the constraints for the traditional RBAC model [Gav98], do not capture the constraints required in modeling event-based systems. For example precedence and dependency constraints cannot be modeled using the consistency rules specified in [Gav98]. For an RBAC system constraints can be grouped into following classes: 1) cardinality constraints, 2) separation of duties (SoD) constraints, 3) inheritance constraints, and 4) precedence and dependency constraints.

Cardinality, inheritance and SoD constraints are addressed in literature for traditional RBAC [Gav98], however these approaches are primarily static in nature and do not take into account various authorization related RBAC events allowed in a system non-deterministically. These events include user-role assignment enabling/disabling of a role, and activation/deactivation of a role as described in [Jos01b]. A SoD constraint in an event-based environment may prohibit two conflicting roles to be enabled at the same time, or inhibit two conflicting users of some role to activate that role concurrently. Similarly, inheritance and cardinality constraints have new semantics in this (event-based) environment that may not be captured by static approaches. Precedence and dependency conditions are required to model the relative ordering of events. The following two examples describe situations where precedence and dependency constraints are required:

1. A junior employee of an office is allowed to activate the Junior_Employee role in the system *only if* the manager of the office has activated the Manager role. This condition can be modeled by the precedence constraint.
2. A trainee doctor is authorized to activate his/her role only in presence of a senior doctor. In this case the senior doctor cannot deactivate his/her role if there is an active trainee doctor role. This example represents a dependency constraint.

3.1.1 Consistency Rules for RBAC

In the following, we list a set of consistency rules which are major extensions of consistency rules defined by Gavrila in [Gav98]. This extended set of rules allows modeling of various constraints of RBAC with an event-based approach. This set of rules mainly covers the cardinality, inheritance, SoD and precedence and dependency constraints. The correctness of a system state is verified in the context of these consistency rules. The functions and predicates used in defining these consistency rules are listed in Table 1 and 2.

P1. *Cardinality constraints*:

(a) The number of authorized users for any role does not exceed the authorization cardinality of that role. Formally:

$$\forall r \in ROLES, |\, authorized\_userset(r) \,| \leq authorization\_role\_card(r).$$

(b) The number of roles authorized for any user does not exceed the maximum number of roles the user is entitled to acquire (authorization user cardinality).

$$\forall u \in USERS, |\, authorized\_roleset(u) \,| \leq authorization\_user\_card(u).$$

(c) The number of roles activated by any user *u* does not exceed the maximum number of roles the user is entitled to activate at any time(activation user cardinality).

$$\forall u \in USERS, |\, active\_roleset(u) \,| \leq activation\_user\_card(u).$$

Table 1 Functions used in defining consistency properties for RBAC

assigned_roleset(u)	returns the set of active (currently assumed) roles of user *u* in his/her sessions
,active_roles_insessionset(u,s)	returns the set of active (currently assumed) roles of user *u* in session *s*.
active_userset(r)	returns the set of users who have activated a given role *r* in an ongoing session.
active_sessionset(u)	returns the set of sessions activated by user *u*.
authorized_roleset(u)	returns the roles authorized for a given user. We say that a role *r* is authorized for a user *u* if either *r* is assigned to *u*, or *r* is inherited by another role that is assigned to *u*.
authorized_userset(u)	returns the users authorized for a given role
prec_su_assignset(r_y)	defines the assignment time precedence constraint with same user for role r_y. Refer to property P18 of section 3.1.1.
prec_au_assignset(r_y)	defines the assignment time precedence constraint with any user for r_y. Refer to property P19 of section 3.1.1.
prec_enableset(r_y)	defines the enabling time precedence constraint for role r_y. Refer to property P17 of section 3.1.1.
prec_suss_activeset(r_y)	defines the activation time precedence constraint with same user and same session for role r_y. Refer to property P20 of section 3.1.1.
prec_suas_activeset(r_y)	defines the activation time precedence constraint with same user and any session for r_y. Refer to property P21 of section 3.1.1.
prec_auas_assignset(r_y)	defines the activation time precedence constraint with any user and any session for role r_y. Refer to property P22 of section 3.1.1.
dep_su_assignset(r_y)	returns the set of roles dependent on a given role r_y for user to role assignment. Refer to property P24 of section 3.1.1.
dep_au_assignset(r_y)	returns the set of roles dependent on a given role r_y for user to role assignment. Refer to property P25 of section 3.1.1.
dep_enableset(r_y)	returns the set of roles dependent on a given role r_y for role enabling. Refer to property P23 of section 3.1.1.
dep_suss_activeset(r_y)	returns the set of roles dependent on a given role r_y for role activation by same user in same session. Refer to property P26 of section 3.1.1.
dep_suas_activeset(r_y)	returns the set of roles dependent on a given role r_y for role activation by same user in any session. Refer to property P27 of section 3.1.1.
dep_auas_activeset(r_y)	returns the set of roles dependent on a given role r_y for role activation by any user in any session. Refer to property P28 of section 3.1.1.
conflicting_role_assignset(r_y)	returns the set of roles that conflict with a given role r_y for user role assignment.
conflicting_role_enableset(r_y)	returns the set of roles that conflict with a given role r_y for role enabling.
conflicting_role_activeset(r_y)	returns the set of roles that conflict with a given role r_y for role activation.
conflicting_user_assignset(r_y)	returns the set of users that are assignment time conflicting users for a given role r_y.
conflicting_user_activeset(r_y)	returns the set of users that are activation time conflicting users for a given role r_y.
role_authorization_card(r_y)	denotes the authorization cardinality of role r_y, i.e., the maximum number of users authorized for r_y.
role_activation_card(r_y)	denotes the activation cardinality of role r_y, i.e., the total number of times role r_y can be activated concurrently.
user_authorization_card(u)	denotes the authorization cardinality of user *u*, i.e., the maximum number of roles user *u* is entitled to acquire.
user_activation_card(u)	denotes the activation cardinality of user *u*, i.e., the maximum number of roles user *u* is entitled to activate concurrently.

Table 2 Predicates used in CPN modeling of RBAC

role_assigned(r_y,u_z)	evaluates true if $r_y \in$ *assigned_roleset(u_z).*
role_enabled(r_y)	evaluates true role r_y is in enable state; otherwise.
role_active(r_y,u_z)	evaluatess true if $r_y \in$ *active_roleset(u_z).*
role_active_insession(r_y,,u_z,s_k)	evaluates true if $r_y \in$ *active_roles_insessionset(u_z,s_k).*
prec_su_assign(r_y,R)	evaluates true if R∈ *prec_su_assignset(r_y).*
prec_au_assign(r_y,R)	evaluates true if R∈ *prec_au_assignset(r_y).*
prec_enable(r_y,R)	evaluates true if R∈ *prec_enableset(r_y).*
prec_suss_active(r_y,,R)	evaluates true if R∈ *prec_suss_activeset(r_y).*
prec_suas_active(r_y,R)	evaluates true if R∈ *prec_suss_activeset(r_y).*
prec_auas_active(r_y,R)	evaluates true if R∈ *prec_auas_activeset(r_y).*
dep_su_assign(r_y,r)	evaluates true if *r*∈ *dep_su_assignset(r_y).*
dep_au_assign(r_y,r)	evaluates true if *r*∈ *dep_au_assignset(r_y).*
dep_enable(r_y,r)	evaluates true if *r*∈ *dep_enableset(r_y).*
dep_suss_active(r_y,r)	evaluates true if *r*∈ *dep_suss_activeset(r_y).*
dep_suas_active(r_y,r)	evaluates true if *r*∈ *dep_suss_activeset(r_y).*
dep_auas_active(r_y,r)	evaluates true if *r*∈ *dep_auas_activeset(r_y).*
conflicting_role_assign(r_y,r)	evaluates true if *r*∈ *conflicting_role_assignset(r_y).*
conflicting_role_enable(r_y,r)	evaluates true if *r*∈ *conflicting_role_enableset(r_y).*
conflicting_role_active(r_y,r)	evaluates true if *r*∈ *conflicting_role_activeset(r_y*
conflicting_user_assign(r_y, r_z,u,u')	evaluates true if both *u and u'*∈ *conflicting_user_assignset(r_y)* and $r_y \geq r_z$.
conflicting_user_active(r_y,u,u')	evaluates true if both *u and u'*∈ *conflicting_user_activeset(r_y).*
last_session(r_y,s_k)	evaluates true if s_k is the last/only active session associated with role r_y.
su_last_session(u_x,r_y,s_k)	evaluates true if s_k is the last/only active session associated with user u_x and role r_y.

(d) The number of users who have activated a role r in their ongoing sessions does not exceed the role activation cardinality.

$$\forall r \in USERS, | active\_userset(r) | \leq activation\_role\_card(r).$$

(e) The number of sessions activated by any user *u* does not exceed the maximum number of sessions the user is entitled to activate at any time(user session cardinality).

$$\forall u \in USERS, | active\_sessionset(u) | \leq user\_session\_card(u).$$

P2. If a role r_1 inherits role r_2 and both roles are distinct, then r_2 cannot inherit r_1. Formally:

$$\forall r_1, r_2 \in ROLES, (r_1 \geq r_2) \wedge (r_1 \neq r_2) \Rightarrow (r_2 \geq r_1)$$

P3. Any two distinct roles assigned to same user do not inherit (directly or indirectly) one another. Formally:

$$\forall u \in USERS, \forall r_1, r_2 \in ROLES, r_1, r_2 \in assigned\_roleset(u) \Leftrightarrow \neg(r_1 \geq r_2)$$

P4. Any two roles assigned for same user are not in static separation of duties. Formally:

$$\forall u \in USERS, \forall r_1, r_2 \in ROLES, r_1, r_2 \in assigned\_roleset(u) \Rightarrow \neg conflict\_role\_assign(r_1, r_2).$$

P5. There is no role in static separation of duties with itself. Formally:

$$\forall r \in ROLES \Rightarrow \neg conflict\_role\_assign(r, r)$$

P6. The static separation of duties relation is symmetric. Formally:

$$\forall r_1, r_2 \in ROLES, \neg conflict\_role\_assign(r_1, r_2) \Rightarrow \neg conflict\_role\_assign(r_2, r_1)$$

P7. If a role (directly or indirectly) inherits another role and the inherited role is in static separation of duties with a third role, then the inheriting is in static separation of duties with the third role. Formally:

$$\forall r_1, r_2, r_3 \in ROLES, r_1 \geq r_2 \wedge conflict\_role\_assign(r_2, r_3) \Rightarrow conflict\_role\_assign(r_1, r_3)$$

P8. If a role inherits another role, then the assignment time conflicting set of users of the inherited role is a subset of the assignment time conflicting set of users of the inheriting role. Formally:

$$\forall r_1, r_2 \in ROLES, r_1 \geq r_2, conflict\_user\_assignset(r_2) \subseteq conflict\_user\_assignset(r_1).$$

P9(a). Only one user from a set of assignment time conflicting users of role *r* can be assigned role *r*.

$$\forall r \in ROLES, \forall u_1, u_2 \in conflict\_user\_assignset(r), r \notin assigned\_roleset(u_1) \bigcap assigned\_roleset(u_2)$$

P9(b) If two distinct roles r_1 and r_2 with $r_2 \geq r_1$, have some common assignment time conflicting set of users, then only one user from the common set can be assigned any of the two roles r_1 and r_2 and not both. Formally:

$$\forall r_1, r_2 \in ROLES : r_2 \geq r_1,$$
$$\exists u_i \in \{conflict\_user\_assignset(r_1) \bigcap conflict\_user\_assignset(r_2)\} \wedge r_1 \in assigned\_roleset(u_i)$$
$$\Rightarrow \neg(\exists u_j \in \{conflict\_user\_assignset(r_1) \bigcap conflict\_user\_assignset(r_2)\}$$
$$\wedge r_2 \in assigned\_roleset(u_j) \wedge u_i \neq u_j)$$

P10. The active role set of any user is a subset of his/her authorized roles. Formally:

$$\forall u \in USERS, active\_roleset(u) \subseteq authorized\_roleset(u).$$

P11. Any two roles in dynamic separation of duties do not both belong to the active role set of any user. Formally:

$$\forall u \in USERS, \forall r_1, r_2 \in ROLES, r_1, r_2 \in active\_roleset(u) \Rightarrow \neg conflict\_role\_activeset(r_1, r_2)$$

P12. The dynamic separation of duties and static separation of duties are disjoint. Formally:

$$\forall r_1, r_2 \in ROLES, (r_1, r_2) \in conflict\_role\_activeset(r_1, r_2) \Rightarrow \neg conflict\_role\_assignset(r_1, r_2)$$

P13. There is no role in dynamic separation of duties with itself. Formally:

$$\forall r \in ROLES \Rightarrow conflict\_role\_activeset(r, r)$$

P14. The dynamic separation of duties relation is symmetric. Formally:

$$\forall r_1, r_2 \in ROLES, (r_1, r_2) \in conflict\_role\_activeset(r_1, r_2) \Rightarrow conflict\_role\_activeset(r_2, r_1)$$

P15. If a role inherits a junior role, then the activation time conflicting set of users of the inherited role is a subset of the activation time conflicting set of users of the inheriting role. Formally:

$$\forall r_1 , r_2 \in ROLES, r_1 \geq r_2 \; conflict\_user\_activeset(r_2) \subseteq conflict\_user\_activeset(r_1).$$

P16 Role *r* cannot be concurrently activated by users u_1 and u_2, if they are activation time conflicting users for role r. Formally:

$$\forall r \in ROLES, \forall u_1, u_2 \in conflict\_user\_activeset(r) \Rightarrow r \notin active\_roleset(u_1) \bigcap active\_roleset(u_2)$$

The following rules (P17-22) define the precedence constraints.

P17. *Enabling time precedence constraint:* If *prec_enableset(r)*$\neq\phi$, then *role r can be enabled* only if all *r'* are enabled. Where, $r' \in R_i$ for some $R_i \in prec\_enableset(r)$. Note that after getting enabled role *r* can still remain in enable state even though some or all of the roles $r' \in R_i$ get disabled.

P18. *Assignment time precedence constraint with same user*: If *prec_su_assignset(r)*$\neq\phi$, then role *r* can be assigned to user *u* only if all *r'* have been assigned user *u*.. Where, $r' \in R_i$ *for some* $R_i \in prec\_su\_assignset(r)$. Note that the (*u,r'*) de-assignment does not imply that (*u,r*) assignment be cancelled.

P19. *Assignment time precedence constraint with any user*: If *prec_au_assignset(r)*$\neq\phi$, then role *r* can be assigned to any user *u* only if all *r'* have been assigned to one or more users. Where, $r' \in R_i$ *for some* $R_i \in prec\_au\_assignset(r)$**.** Note that this does not imply that all *r'* have to be assigned to same user. Also, the de-assignment of role r' does not imply that (*u,r*) assignment be cancelled.

P20 *Activation time precedence constraint with any user*: If *prec_auas_activeset(r)*$\neq\phi$, then role *r* can be activated by user *u*, if all *r'* have been activated by one or more users. Where, $r' \in R_i$ for some $R_i \in prec\_auas\_activeset(r)$**.** Note that this does not imply that all *r'* have to be activated by same user. Also, after getting activated by user *u*, role *r* can still remain in active state even though some or all of the roles $r' \in R_i$ get deactivated.

P21. *Activation time precedence constraint with same user same session*: If *prec_suss_assignset(r)*$\neq\phi$, then role *r* can be activated by user *u*, if all *r'* have been activated by same user *u* in the same session. Where, $r' \in R_i$ for some $R_i \in prec\_suss\_activeset(r)$**.** Note that after getting activated by user *u* in session *s*, role *r* can still remain in active state even though some or all of the roles $r' \in R_i$ get deactivated by same user *u* in same session *s*.

P22. *Activation time precedence constraint with same user any session*: If *prec_suas_assignset(r)*$\neq\phi$, then role *r* can be activated by user *u*, if all *r'* have been activated by same user *u* any session. Where, $r' \in R_i$ for some $R_i \in prec\_suas\_active(r)$**.** Note that after getting

activated by user *u* in session *s*, role *r* can still remain in active state even though some or all of the roles $r' \in R_i$ get deactivated by same user *u*.

The following rules define the dependency constraint in RBAC

P23. *Enabling time dependency constraint*: A role $r_z \in$ *dep_enableset*(r_y) can be enabled only if role r_y is in enable state. Furthermore, r_y cannot be disabled if r_z is in enable state. Formally:

$$r_z \in dep\_enableset(r_y) \Rightarrow [role\_enabled(r_z) \rightarrow role\_enabled(r_y)].$$

P24. *Assignment time dependency constraint with same user*: A role $r_z \in$ *dep_su_assignset*(r_y) can be assigned to user *u* only if *u* has been assigned role r_y. Furthermore, the (u,r_y) role assignment cannot be canceled unless (u,r_z) assignment has been canceled. Formally:

$$r_z \in dep\_su\_assignset(r_y) \Rightarrow [r_z \in assigned\_roleset(u) \rightarrow r_y \in assigned\_roleset(u)]$$

P25. *Assignment time dependency constraint with any user*: A role $r_z \in$ *dep_su_assignset*(r_y) can be assigned to user *u* only if *u* has been assigned role r_y. Furthermore, the (u,r_y) role assignment cannot be canceled unless (u,r_z) assignment has been canceled. Formally:

$$r_z \in dep\_su\_assignset(r_y) \Rightarrow [r_z \in assigned\_roleset(u) \rightarrow r_y \in assigned\_roleset(u)]$$

P26. *Activation time dependency constraint with same user and same session*: A role $r_z \in$ *dep_suss_activeset*(r_y) can be activated by user *u* in session *s* only if *u* has activated role r_y in the same session s. Furthermore, *u* cannot deactivate role r_y in session *s* if role r_z is active in session *s*.

$$r_z \in dep\_suss\_activeset(r_y) \Rightarrow [r_z \in active\_roles\_insessionset(u,s) \rightarrow r_y \in active\_roles\_insessionset(u,s)]$$

P27. *Activation time dependency constraint with same user and any session*: A role $r_z \in$ *dep_suas_activeset*(r_y) can be activated by user *u* in session *s* only if *u* has activated role r_y in some session. Furthermore, *u* cannot deactivate role r_y if role r_z is in active role set of *u*. Formally:

$$r_z \in dep\_suas\_activeset(r_y) \Rightarrow [r_z \in active\_roleset(u) \rightarrow r_y \in active\_roleset(u)]$$

P28. *Activation time dependency constraint with any user and any session*: A role $r_z \in$ *dep_auas_activeset*(r_y) can be activated by user *u* only if role r_y is in the active role set of some user *u'*. Furthermore, role r_y cannot be deactivated if role r_z is in active role set of any user. Formally:

$$r_z \in dep\_auas\_activeset(r_y) \Rightarrow [r_z \in active\_roleset(u) \rightarrow r_y \in \bigcup_{u' \in USERS} active\_roleset(u')]$$

3.2 Colored Petri-Net model of RBAC

In this section, we present a Colored-Petri-net (CPN) based framework to model RBAC. We first present a brief background on CPNs followed by the detailed description of the RBAC components and its CPN representation.

Petri nets have been widely used for modeling and analysis of systems that are characterized as being concurrent, asynchronous, distributed, parallel and non-deterministic [Jen97]. Various factors that contribute to their success include [Jen97, Mur89]:

- The duality of places/transitions in Petri-nets provides for a balanced treatment of system states and events, allowing better modeling of dynamic event-based systems than that is

allowed by purely state or transition-oriented formalism where only one aspect is explicit. In particular, it makes CPN appropriate for modeling GTRBAC specification as there are state information and event occurrences that need to be captured for analysis purposes.

- CPNs have many desirable practical features for modeling such as graphical nature and the equational representation, thus, integrating the documentation, presentation and analysis features of a desirable model. CPNs provide a intuitive way of expressing *causal dependencies*, *conflicts* and *concurrency*. In other words, "*CPNs are potentially more focused on pragmatism than most other commonly known formal methods*".

CPN Formulation of RBAC: *A CPN* [Jen97] *is a tuple* CP = (Σ, *P, T, A, N, C, G, E, I*), *where*:

a. Σ *is a finite set of non-empty types, called color sets;*
b. P is a finite set of places;
c. T is a finite set of transitions;
d. $A = NA \cup RA \cup IA$ *is a finite set of arcs such that:* $P \cap T = P \cap A = T \cap A = \varnothing$*; where RA is a set of Normal Arcs, RA is a set of Read Arcs and IA is a set of Inhibitor Arcs.*
e. N *is a node function.* $N: A \rightarrow P \times T \cup T \times P$.
f. C: is a color function. $C: P \rightarrow \Sigma$.
g. G is a guard function. It is defined from T into expressions such that: $\forall t \in T$: *[Type(G(t)) = Boolean and Type(Var(G (t)) $\subseteq \Sigma$]*.
h. E is an arc expression function. It is defined from A into expressions such that: $a \in A$: *[Type(E(a)) =* $E(G(a))_{MS}$ *and Type(Var(E(a)) $\subseteq \Sigma$]. Here p(a) is the place of N(a).*
i. I is an initialization function. It is defined from P into closed expression such that: $\forall p \in P$: *[Type(I(p)) =* $C(p(a))_{MS}$*]*.

In the following, we elaborate the above elements of CPN within the context of RBAC.

Color set Σ:

For the RBAC formulation, the elements of the color set Σ with the corresponding data type are listed below.
Color USER = integer, Color ROLE = integer.
Color SESSION = integer.
Color COMMAND = {assign, de-assign, enable, disable, activate, deactivate}
Color UR = product USER * ROLE * ROLE; Color URS = product USER * ROLE * SESSION;
Color CMD = product COMMAND * USER * ROLE * SESSION.

Based on the above set of colors, following tokens are defined for RBAC mode:

- *User token*: <*u*>::color USER
- *Role token*: <*r*>:: color ROLE
- *User-role assignment token*: <*u,r,r'*>::color UR.
- *User-role activation token*: <*u,r,s*>::color URS.
- *Command token*: <*cmd, u, r, s*>:: color CMD.

Places P:

Following CPN places are used to capture the state information for RBAC modeling:

1. *Event token generator* (ETG): This place stores command tokens for user-role assignment and de-assignment, role enabling and disabling, and role activation and deactivation. For any transition to get enabled, there must be a corresponding token in he place ETG. In this sense, this place act as a transition firing controller that helps in analyzing all possible system states against a given command list.

2. *Disabled Roles* (DR): This place can only store role tokens (C(DR) = ROLE). A token $<r_y>$ in this place implies that role r_y is in disable state.

3. *Enabled Roles* (ER). This place can only store role tokens (C(ER) = ROLE). A token $<r_y>$ in ER place implies that role r_y is in enable state.

4. *User Role Assignment/Authorization* (UR). This place contains tokens of color UR (C(UR) = UR). A token $<u,r_y,r_x>$ in this place means that user u is authorized for role r_y. This authorization can be as a result of direct assignment of role r_y to user u ($r_x = r_y$), or because of assignment of role r_x to user u such that r_x inherits r_y ($r_x \geq r_y$ and $r_x \neq r_y$).

5. *User Role Session activation* (URS). This place stores tokens of color URS. . Each $<u,r,s>$ token stored in this place implies that session s is being activated by user u who has assumed role r.

6. *Role Cardinality* (RC): This place contains role tokens only (C(RC) = ROLE). It enforces assignment time role cardinality constraint, i.e., limits the number of users which can be authorized for a given role. If there are n_i number of $<r_y>$ tokens in place RC and no user is assigned role r_y, then role r_y can be assigned to at most n_i number of users.

7. *User Cardinality* (UC): This place contains user tokens only (C(UC) = USER). It enforces assignment time user cardinality constraint, i.e., limits the number of roles for which a given user can be authorized. If there are m_j number of $<u_z>$ tokens in RC and u_z is not authorized for any role, then user u_z can be authorized for at most m_j number of roles.

8. *Role Activation cardinality* (RAC): Place RAC stores token of type ROLE (C(RAC) = ROLE). It enforces activation time role cardinality constraint, i.e., limits the number of concurrent activations of a given role. If there are n_i number of $<r_y>$ tokens present at RC, then at most n_i more copies of role r_y can be activated concurrently.

9. *User Activation cardinality* (UAC): Place UAC stores token of type USER (C(UAC) = USER). This place enforces activation time user cardinality constraint, i.e., limits the number of concurrent activations of roles for a given user. If there are m_j number of $<u_z>$ tokens present at RC, then user u_z can make m_j more activations concurrently. These activations may involve activating same role multiple times or multiple roles for any number of times provided that the total number of such concurrent activation of roles by user u_z do not exceed the user activation cardinality m_j.

Arcs and arc expression:

Arc, arc expressions and guard functions are used to model constraints including cardinality, SoD, inheritance, precedence and dependency constraints as discussed in section 3.1.

In this paper, an arc from place to transition (arrow head points at transition) is referred as an *input arc* and an arc from transition to place (arrow head towards place) is referred as an *output arc*. As stated in the definition of CPN, arc can be of three types: normal arc (*NA*), read arc (*RA*), and inhibitor arc (*IA*). An input arc may be of any of the above types. Input arc defines both pre-conditions and post-conditions of an event modeled by a transition and these conditions are modeled using different types of input arcs (NA, RA, and IA). Satisfaction of all pre-conditions of an event means that such event can take place anytime. In this case, a transition modeling such event is said to be enabled. The following explains how input arc with different types represent different pre-conditions:

- **Input arc of transition *t* is a *NA* arc**: For a transition *t* to be enabled, each input place *p* of *t* connected through the *NA* arc must have at least as many *matching* tokens as defined in the arc expression of the arc. *A token in place p is a matching token if its color matches with the*

color of tokens specified in the arc expression and it satisfies the guard expression associated with the transition and the corresponding arc. When transition *t* is fired, the same number of tokens defined in the arc expression *E* of the *NA* arc is removed from the input place *p*. A *NA* arc is represented by a line with an arrow head towards the output node.

- **Input arc of transition *t* is an *RA* arc**: For transition *t* to be enabled, each input place *p* of *t* connected through the *RA* arc must have at least as many *matching tokens* as defined in the arc expression of the *RA* arc. When a transition is fired, no token is removed from the input place. An *RA* arc is represented by a line with an arrow on both sides.
- **Input arc of transition *t* is an *IA* arc**: For transition *t* to be enabled, each input place *p* of *t* connected through the *IA* arc should not have matching tokens greater than the number of tokens defined in the arc expression of the arc. When a transition is fired, no token is removed from the input place *p*. An *IA* arc is represented by a line with a circle drawn towards the transition.

An output arc is always of type NA. When a transition is fired, a number of matching tokens is deposited in the output place of that transition.

Arc expression E maps each arc, A, into an expression of type $C(P(A))_{MS}$ (MS = multi-set). This means that the variables used in arc expression and the tokens stored in the corresponding place are of same color. An arc expression may take one of the following two forms:

1. $m_1t_1 + m_2t_2 + \ldots\ldots + m_nt_n$; where t_is are the tokens. An output arc expression (from a transition to a place) will always take this form. An input arc expression (from place to transition) of this form associated with normal/read arc implies that input place must have at least m_i number of t_i tokens ($1 \leq i \leq n$) in order to enable its output transition. For input inhibitor arc, all coefficients m_i except for one are zero. The corresponding output transition cannot be enabled if the input place contains m_j or more tokens t_j, where m_j is the nonzero coefficient.
2. $\{t_i\}$. Only input arc expression may take this form. If an input arc expression is of this type then there is always an associated transition guard function with the set $\{t_i\}$ as one of its input parameters. An input guard expression of this type implies that all t_i tokens present in the input place will be evaluated against the corresponding guard function to determine if the output transition can be enabled.

Transitions:

Transitions in this framework represent all four components of Fig. 1 including user-role assignment/de-assignment, role-permission assignment/de-assignment, role enabling/disabling and role activation/deactivation. In this CPN representation, each role r_y has the following six transitions:

1. Assignr_y: assigns user $u \in$ USERS to role r_y. By virtue of this role assignment user u is authorized for all roles inherited by role r_y.
2. De-assignr_y: Cancels all the user role assignment between user u and role r_y. It also nullifies u's authorization for all junior roles that are on u's authorization list by virtue of its assignment to role r_y.
3. Enabler_y: This transition enables role r_y. Upon firing, a token r_y is inserted in place ER from DR, implying that role r_y is enabled and can be activated by a user who is authorized for role r_y.
4. Disabler_y: This transition disables role r_y. Upon firing, r_y is removed from place ER and inserted in place DR, implying that role r_y can not be activated by any user.

5. Activater_y: This transition establishes an active session between user *u* and role r_y.
6. Deactivater_y : This transition deactivates role r_y from the an active session between user *u* and role r_y.

Firing of any of the above transitions changes the state of the system. A transition can fire anytime after its enabling. Enabling of a transition implies that all the constraints associated with the event, the transition is modeling are satisfied. Transitions modeling different events have different enabling and firing rules. For brevity in presentation, we list the enabling/firing rules for assignment of roles only.

Enabling/firing rules of transition assignr_y:

This transition upon firing inserts the set of tokens $\{<u, r_x, r_y>: r_y \geq r_x\}$ in the place UR which implies that the role r_y is assigned to user *u*, and user *u* is authorized for role r_y and all roles r_x junior to role r_y. The transition *assign*r_y and its connecting places are shown in Fig. 2 and the corresponding arc expressions and guard functions are listed in Table 3.

This transition gets enabled if the following constraints are satisfied:

- There is a token <*assign*, u_z, r_y> in place ETG implying that role r_y be assigned to user u_z.
- Assignment time role cardinality constraint specified by the arc expression E3: $r_y+r_{y1}+\ldots r_{yn}$, where, all $r_{yi} < r_y$ and $i \leq n$, is satisfied. Alternatively, tokens $r_y, r_{y1}, \ldots, r_{yn}$ are present in place RC.
- Assignment time user cardinality constraint specified by the arc expression E4: $(n+1)u_z$ is satisfied, where *n* is the number of roles that are junior to r_y in the role hierarchy.
- Assignment time conflicting roles constraint specified by the arc expression (inhibitor) E6: <u_z, r_c, *any r* > and the transition guard function G2: *conflict_role_assign*(r_y r_c) is satisfied. That is Place UR does not contain any token <u_z, r_c, *any r* > for which the above guard function evaluates true.
- Assignment time conflicting users constraint specified by the arc expression (inhibitor) E7:<u_c, r_z, *any r*> and the transition guard function G3: *conflict_user_Assign*(r_y,u_z,u_c) is satisfied. That is Place UR does not contain any token <u_c, r_z , *any r*> for which the above guard function is true.
- Place UR does not contain any token <u_z, r_y, *any r* >. This is specified by the inhibitor arc expression E3 and guard function G1.
- The following two constraints are optional and are only defined for roles which have assignment time precedence constraint(s). Assignment time precedence constraint can be of two types: same user assignment constraint and any user assignment constraint. A given role may have one, both or none of these precedence constraints.

  1. Same user assignment constraint requires that a user u_z can be assigned role r_y, if all roles *r'* are assigned to user u_z. Where $r' \in R_i$ for some $R_i \in$ *prec_su_assignset*(r_y). This constraint is specified by the read arc expression E8 and the transition guard function G4: *prec_su_assign*(r_y,*{r}*). This constraint represents consistency property P18.

  2. Any user assignment constraint requires that a user u_z can be assigned role r_y, if all roles *r''* are assigned to any users. Where $r' \in R_i$ for some $R_i \in$ *prec_au_assignset*(r_y). Note that all r'' may not necessarily be assigned to just one user. This constraint is specified by the read arc A9 and the transition guard function G5: *prec_au_assign*(r_y,*{r}*).

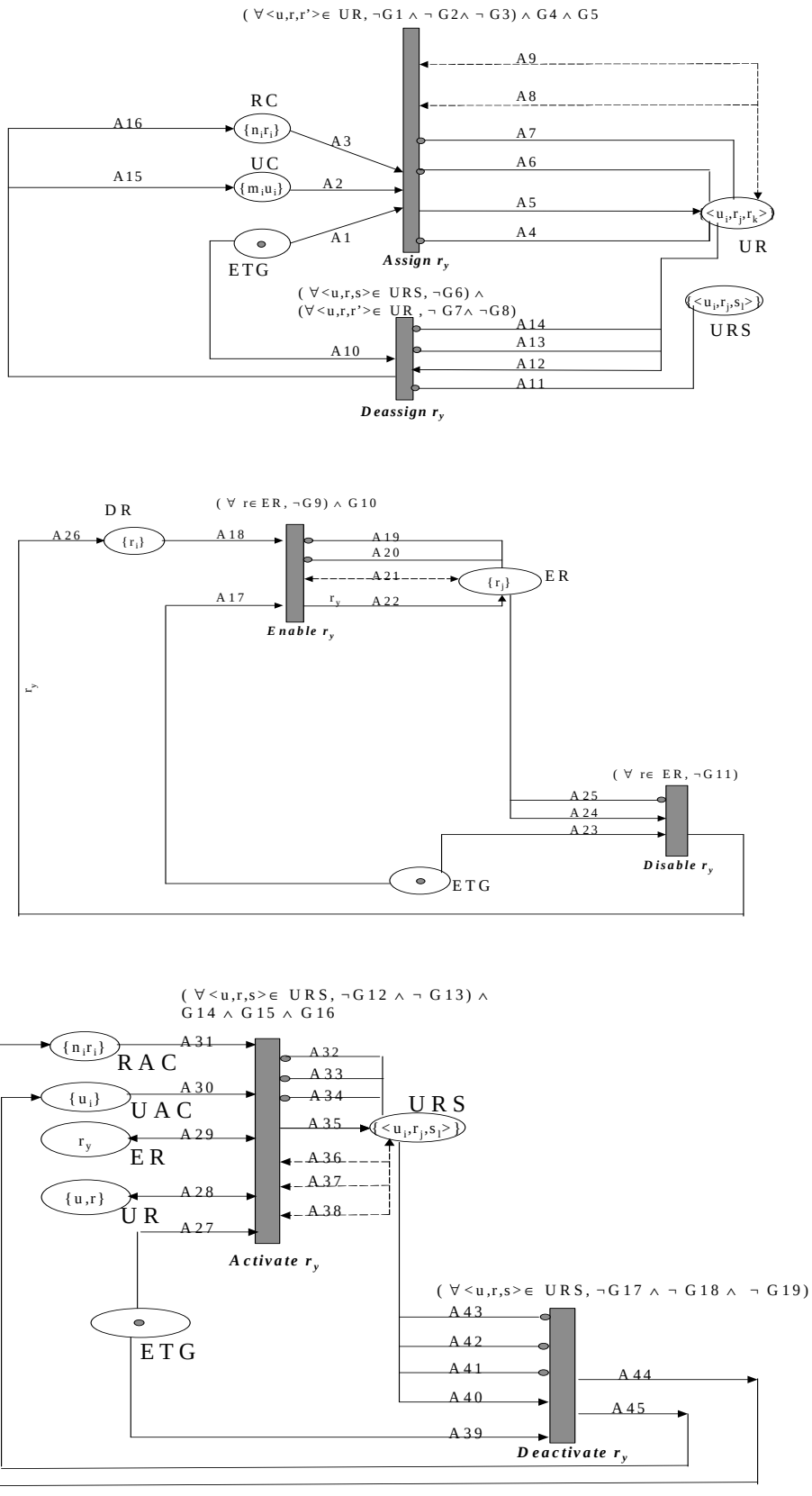

Figure 2. Petri-net construction for (a) User Role Assignment/De-assignment (b) Role Enabling/Disabling (c) Role Activation/Deactivation

Table 3 Arc and guard expressions

Arc Expression E_i for corresponding arc A_i, where			
E1	<assign,u_Z,r_Y>	**E24**	r_Y
E2	(n+1)u_Z	**E25**	r_{de}
E3	r_Y+r_{Y1}+…r_{Yn} (r_{Yi}<r_Y. for all 1≤i≤n)	**E26**	r_Y
E4	<u_Z,r_h, any r>	**E27**	<activate,u_Z,r_Y,s_k>
E5	<u_Z,r_Y, r_Y> + <u_Z, r_{Y1},r_Y>+….+<u_Z, r_{Yn},r_Y>	**E28**	<u_Z,r_Y, any r>
E6	<u_Z r_C, any r> A7: <u_C r_Z, any r>	**E29**	r_Y
E7	<u_C r_Z, any r>	**E30**	u_Z
E8	{<u_Z , r', any r>}	**E31**	r_Y
E9	{<any u , r'', any r>}	**E32**	<u_C,r_Z,any_session>
E10	<de-assign,u_Z,r_Y>	**E33**	<u_Z,r_C,any_session>
E11	<u_Z r_k, any r>	**E34**	<u_Z,r_Y,s_k>
E12	<u_Z,r_Y, r_Y> + <u_Z, r_{Y1},r_Y>+….+<u_Z, r_{Yn},r_Y>	**E35**	<u_Z,r_Y,s_k>
E13	<u_Z,r_i, r_{dsu}> ($r_i \leq r_{dsu}$)	**E36**	{<u_Z,r',s_k>}
E14	<any u, r_j, r_{dau}> ($r_j \leq r_{dau}$)	**E37**	{<u_Z,r'',any_session>}
E15	(n+1)u_Z	**E38**	{<u_Z,r''',any_session>}
E16	r_Y+r_{Y1}+…r_{Yn} (r_{Yi}<r_Y. for all 1≤i≤n)	**E39**	<deactivate,u_Z,r_Y,s_k>
E17	<enable,r_Y>	**E40**	<u_Z,r_Y,s_k>
E18	R_Y	**E41**	<u_Z,r_{d1},s_k>
E19	R_C	**E42**	<u_Z,r_{d2},any_session>
E20	R_Y	**E43**	<any user,r_{d3},any_session>
E21	{r'}	**E44**	r_Y
E22	R_Y	**E45**	u_Z
E23	<disable,r_Y>		
Guard functions associated with transition *Assignry* and *De-ssignry*			
G1	:($r_h \leq r_Y$) ∨($r_Y \leq r_h$)	**G11**	dep_enable(r_Y, r_{de})
G2	conflict_role_assign(r_Y,r_C)	**G12**	conflict_user_activate(r_Y,u_Z,u_C)
G3	conflict_user_assign(r_Y, r_Z, u_Z, u_C)	**G13**	conflict_role_activate(r_Y,r_C)
G4	prec_su_assign(r_Y,{r'})	**G14**	prec_suss_active(r_Y,{r'})
G5	prec_au_assign(r_Y r',})	**G15**	prec_suas_active(r_Y,{r''})
G6	$r_k \leq r_Y$	**G16**	prec_auas_active(r_Y,{r'''})
G7	dep_su_assign(r_{dsu}, r_Y)	**G17**	dep_suss_active(r_Y,r_{d1})
G8	dep_au_assign(r_{dau}, r_Y)	**G18**	dep_suss_active(r_Y,r_{d2})
G9	Conflict_role_enable(ry,rc)	**G19**	dep_suss_active(r_Y,r_{d1})
G10	prec_enable(ry,{r'})		

Referring back to Fig. 1, we now elaborate how the proposed CPN can capture the four components of event-based RBAC. Fig. 2 represents the complete specification of Fig. 1 except the role to permission assignment/de-assignment, which is identical to Fig. 2(a). The guard functions and arc expressions corresponding to Fig. 2 are listed in Table 3. Fig. 2(a) shows a CPN representation of user to role assignment/de-assignment with transition $assignr_y$ and $deassignr_y$ modeling the assignment and de-assignement events for role r_y respectively. The set of places in Fig. 2(a) shows the current state of the system in terms of number of users assigned to role r_y, the number of active sessions associated with role r_y etc. The arcs and guard expressions specify the assignment time cardinality, SoD, precedence and dependency constraints. Similarly, Fig. 2(b) shows the CPN representation of enabling and disabling events for role r_y, and Fig. 2(c) depicts the CPN representation of role activation and deactivation events for role r_y.

Based on the discussions in Section 3.1, we now formalize the notion of a consistent RBAC state in the following definition. This notion of consistency is used to capture the dynamic property of the CPN in Theorem 1.

Definition: The state of an RBAC system is said to be consistent if all the cardinality, inheritance, SoD, precedence and dependency constraints are satisfied in that state.

In the following, we provide a theorem which establishes the validity of models in Fig. 2 for the specification of GTRBAC components of Fig. 1.

Theorem 1: Given a P_{RBAC} (CPN structure for RBAC) structure with an initial consistent state M_0, all states M, reachable from M_0 are consistent.

Proof of this theorem is given in the appendix.

Lemma: Given a bounded initial state and a finite number of command tokens in the place ETG, the P_{RBAC} (CPN structure for RBAC) structure remains bounded.

The proof for this lemma follows from the fact that the following pairs of places - (RC, UR), (UC, UR), (RAC, URS), (UAC, URS) and (ER, DR) form place invariants. Therefore, the total number of tokens in these pairs of places always remains the same as that in the initial state. The ETG initially contains a finite number of command tokens and no transition firing generates a new token in it.

3.3 Reachability analysis for consistency verification of RBAC Policy

In this section, we elaborate the process of verifying the consistency of RBAC policy constraints. The verification is based on the reachability analysis of CPN proposed in the previous section. We use occurrence graph method [Jen97] to enumerate all reachable states of a system employing a given RBAC policy. The above lemma states that our Petri net representation of RBAC system is bounded and so its occurrence graph will have finite number of nodes. However, the exhaustive nature of this method implies that the problem of verifying that a given state is reachable from some initial state takes exponential space and time [Mur89]. Since policy verification can be done offline and is performed before the deployment of actual system, so complexity is not a major issue in using this proposed Petri-net approach.

The following two examples illustrate the use of occurrence graph for security policy verification.

Example 1: Consider three roles r_0, r_1, and r_2 and a single user u_0. Let r_1 be junior to r_0 ($r_0 \geq r_1$ and $r_1 \neq r_0$). Also let r_1 and r_2 be assignment time conflicting roles, i.e., r_1 and r_2 cannot be assigned to the same user implying that roles r_1 and r_2 cannot be activated by the same user concurrently. Fig. 2 shows the sub-graph of the occurrence graph of the RBAC system. In this sub-graph all roles (r_0, r_1, and r_2) are considered to be in enable state and the SoD constraint is only defined between roles r_1, and r_2. Note that in Fig. 3, user u_0 who is assigned role r_0 and r_2 is able to activate roles r_1

and r_2 concurrently. This is a violation of the SoD constraint defined on these two roles. This inconsistency arises because of the fact that in the original specification, roles r_0 and r_2 do not have any SoD constraint while r_1 and r_2 are assignment time conflicting roles. As r_0 is superior to role r_1 and any user assigned to role r_0 is authorized for role r_1, the SoD constraint must also be defined between roles r_0 and r_2. Fig. 4 shows the occurrence sub-graph of the same system with an additional assignment time SoD constraint defined between roles r_0 and r_2. Note that in this figure all the reachable states are consistent with respect to the given policy specifications.

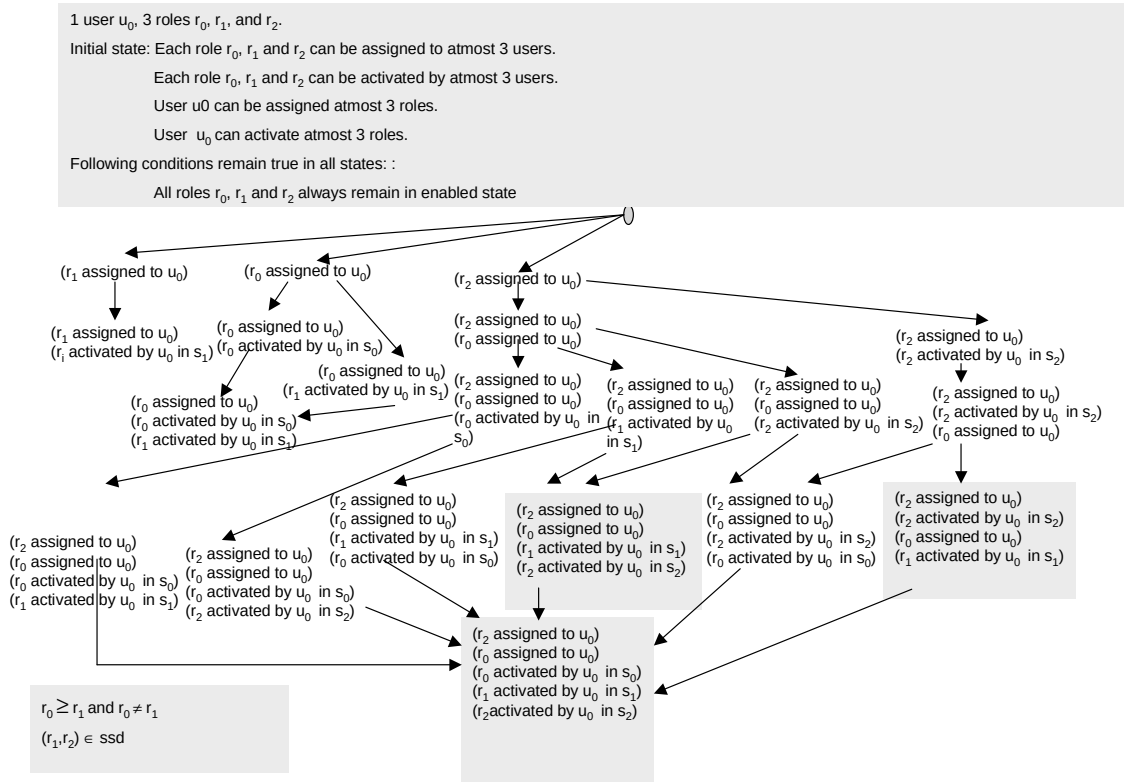

Figure 3. Occurence graph for Example 1 with incomplete specifications

Example 2: Consider four roles r_0, r_1, r_2, and r_3 and a single user u_0. Let r_1 be junior to r_0 ($r_0 \geq r_1$ and $r_1 \neq r_0$). Suppose r_1 has same user activation time dependency on role r_2 which in turn has *same user activation time dependency* on role r_3, i.e., $r_1 \in$ *dep_suas_activeset*(r_2) and $r_2 \in$ *dep_suas_activeset*(r_3). Also, assume that r_1 and r_3 are activation time conflicting roles, i.e., r_1 and r_3 cannot be activated by same user in concurrent sessions. Fig. 5 shows the occurrence graph of the system in which user u_0 is assigned to roles r_0, r_2 and r_3, and all four roles are in enabled state. The occurrence graph depicts that there is no reachable state in which user u_0 can activate role r_1, although u_0 is authorized for role r_1. This implies that a user assigned to role r_0 can never assume its junior role r_1 - a flaw in the security policy.

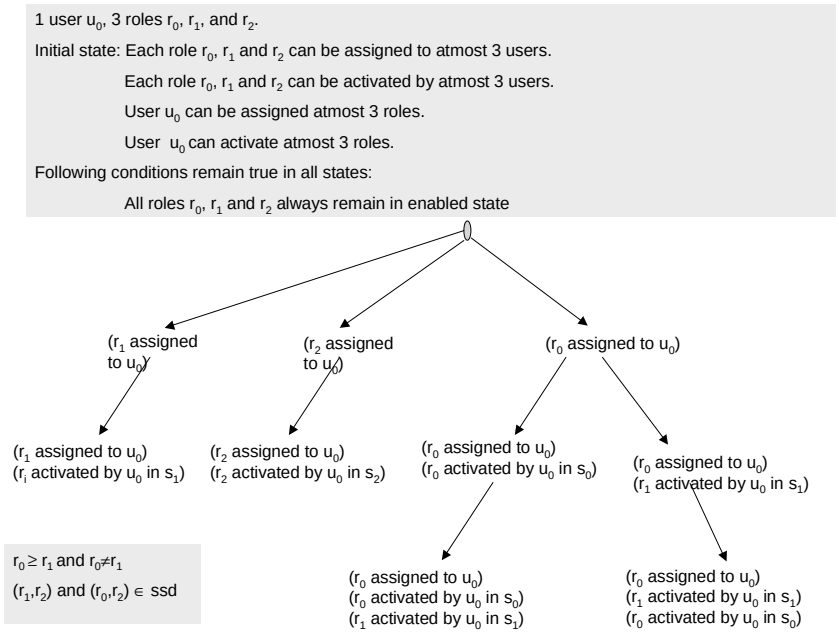

Figure 4. Occurence graph for Example 1 with correct specification

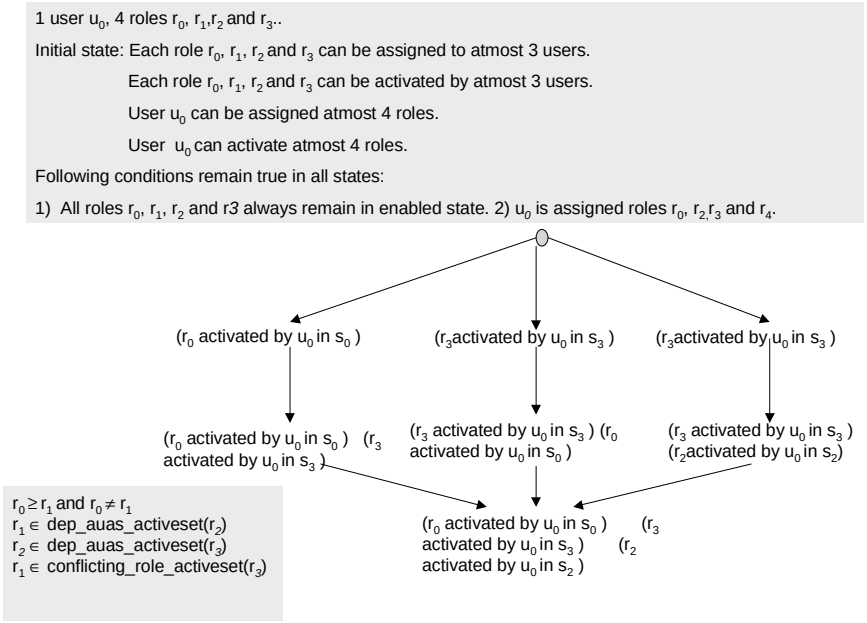

Figure 5. Occurence graph for Example 2

4 Related Work

RBAC models have been proposed and extended by several researchers [Nya99, San96, Fer01], and the efforts in this direction have resulted in the proposal of a standard model – the NIST RBAC model [Fer01]. Need for supporting constraints in an RBAC model has been addressed by many researchers. In particular, the attention has been in supporting *separation of duties* (SoD) constraints [Ahn00, Gav98]. In [Ahn00], Ahn *et. al.* propose *RCL2000* – a role based constraint

specification language. Bertino *et. al.* have proposed a logic based constraint specification language that can be used to specify constraint on roles and users and their assignments to workflow tasks [Ber99]. Although, precedence and dependency constraints have been used in workflow and transaction systems [Ber99], to the best of our knowledge they have not been addressed explicitly for RBAC systems.

Various work address policy analysis and verification issues related to RBAC models. Nyanchama *et. al.* [Nya99] present a graph based RBAC model, where graphs are used to mainly represent hierarchies of users, roles and permissions. It does not address the issue of policy verification. Koch *et. al.* [Koc02] present a graph transformation based formalism for RBAC model and model the SoD constraints identified in the literature. The model provides a graph transformation based specification of static and dynamic consistency conditions of RBAC. Jaeger *et. al.* [Jae97] provide a graphical model or constraint expressions where nodes, similar to places in CPN proposed in this paper, represent sets and edges represent binary relations between those sets. Here, the constraints are expressed using operators on the nodes. All these models, however, do not model events explicitly. The key advantage of our CPN model over these is that it provides a balanced treatment of RBAC states and events.

5 Conclusion

We presented a CPN model of RBAC that incorporates various cardinality, separation of duty, precedence and dependency constraints. The proposed CPN framework is based on the event based approach of TRBAC/GTRBAC model and is suitable for modeling event based aspect of RBAC model. We use the reachability analysis to detect and identify inconsistencies among a given set of RBAC policies.

6 References

[Ahn00] G. Ahn, R. Sandhu, "Role-Based Authorization Constraints Specification", *ACM Transactions on Information and System Security*, Vol. 3, No. 4, November 2000.

[Bar97] J. Barkley, A. Cincotta, D. Ferraiolo, S. Gavrila, and D.R. Kuhn, "Role Based Access Control for the World Wide Web," In *20th National Information System Security Conference*, NIST/NSA, 1997.

[Ber99] E. Bertino, E. Ferrari, and V. Atluri, "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems," *ACM Transactions on Information and System Security*, 2(1):65-104, 1999.

[Ber01] E. Bertino, P. A. Bonatti, E. Ferrari, "TRBAC: A Temporal Role-based Access Control Model," *ACM Transactions on Information and System Security*, 4(3):191-233, August 2001.

[Chr92] Søren Christensen and Niels Damgaard Hansen, "Coloured Petri Nets Extended with Place Capacities, Test Arcs and Inhibitor Arcs," *Technical Report DAIMI PB--398*, Computer Science Department, Aarhus University, DK-8000 Aarhus C, Denmark, May 1992.

[Fer93] D. F. Ferraiolo, D. M. Gilbert, N Lynch, "An examination of Federal and Commercial Access Control Policy Needs," In *Proceedings of NISTNCSC National Computer Security Conference*, Baltimore, MD, September 20-23 1993, pages 107-116.

[Fer01] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, R. Chandramouli, "Proposed NIST Standard for Role-based Access Control,"*ACM Transactions on Information and System Security* (TISSEC) 4(3), August 2001.

[Gav98] S. I. Gavrila , J. F. Barkley, "Formal Specification for Role Based Access Control User/role and Role/role Relationship Management," *Proceedings of the third ACM workshop on Role-based access control*, Fairfax, Virginia, United States, October 22-23, 1998, pages81-90.

[Jae01] T. Jaeger, J. E. Tidswell, "Practical Safety in Flexible Access Control Models," *ACM Transactions on Information System Security*, Vol. 4, No. 2, May 2001.

[Jen97] K. Jensen, "Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use Volume 1", *Springer Verlag*, 1997.

[Jos01a] J. B. D. Joshi, W. G. Aref, A. Ghafoor and E. H. Spafford, "Security Models for Web-based Applications," *Communications of the ACM*, Vol. 44, No. 2, Feb. 2001, pages 38-72.

[Jos01b] J. B. D. Joshi, E. Bertino, U. Latif, A. Ghafoor, "Generalized Temporal Role Based Access Control Model (GTRBAC) (Part I)– Specification and Modeling," Submitted to the *IEEE Transaction on Knowledge and Data Engineering*.

[Jos02] J. B. D. Joshi, E. Bertino, A. Ghafoor, "Temporal Hierarchies and Inheritance Semantics for GTRBAC," *Seventh ACM Symposium on Access Control Models and Technologies*, June 2002, pages 74-83.

[Koc02] M. Koch, L. V. Mancini, F. Parisi-Presicce, "A Graph-based Formalism for RBAC," *ACM Transactions on Information and System Security* (TISSEC) August 2002, Vol. 5 No. 3, pages 332 – 365.

[Mur89] T. Murata, "Petri Nets: Properties, Analysis and Application", *Proceedings of IEEE*, Vol. 77, No. 4, 1989, pages 541-580.

[Nya99] M. Nyanchama and S. Osborn, "The Role Graph Model and Conflict of Interest," *ACM Transactions on Information and System Security*, Vol. 2 No. 1, 1999, pages 3-33.

[Osb00] S. Osborn, R. Sandhu, Q. Munawer, "Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies," *ACM Transactions on Information and System Security* (TISSEC) Vol. 3, No. 2, May 2000, pages 85 - 106

[San96] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, "Role-Based Access Control Models," *IEEE Computer* Vol. 29 No. 2, IEEE Press, 1996, pages 38-47.

7 Appendix

This section provides a detailed proof of theorem 1 stated in section 3.2. The proof is based on the fact that all the user-role assignment/de-assignment, role enabling/disabling, and role activation/deactivation events preserve properties P1-P28. Note that properties P1-P28 are non-interfering in a sense, that satisfying one does not violate another. Also, note that properties P2, P5, P6, P7, P8, P12, P13, and P14 are static properties and are independent of the CPN structure P_{RBAC}, i.e., these properties do not depend on the CPN execution. Thus, these properties will be preserved in all reachable states of P_{RBAC} provided that the initial state is consistent.

One important structural property of the P_{RBAC} (CPN structure for RBAC) is the existence of place invariants. P_{RBAC} has place invariants formed by the following pairs of places:

1. RC and UR.
2. UC and UR.
3. RAC and URS.
4. UAC and URS.

The following lemmas provide a basis for proving that P_{RBAC} structure with consistent initial marking preserves system consistency.

Lemma 1: Given P_{RBAC} with a consistent initial state M_0, all the states M reachable from M_0 satisfy the cardinality constraints (property P1).

Proof:

We will prove this lemma for assignment time cardinality constraints. Activation time cardinality constraint can be proved by similar argument. For the transition *assignr*$_y$ (Fig. 2(a)), the input arcs A2 with expression E2: (n+1)u_z, and A3 with expression E3:r_y + $r_{y1+...+}$ r_{yn} ensure that role r_y is assigned to user u_z only if all role tokens r_y , $r_{y1,....,}$ r_{yn} (where, $r_y \geq r_{yi}$ and $r_y \neq r_{yi}$) are present at place RC and there are at least (n+1) u_z tokens present in UC. Upon firing of *assignr*$_y$, all these tokens are removed from their respective input places. If any of the above token is missing from their input places, the transition cannot be enabled and hence cannot fire.

As a consequence of the above observation and the place invariants RC–UR and UC-UR, properties the assignment time user and role cardinality constraints hold in any state obtained by firing transition $assign_{ry}$.

Lemma 2: Given P_{RBAC} with a consistent initial state M_0, all the states M reachable from M_0 preserve property P3.

The inhibitor arc A4 with arc expression E4:<u_z,r_h,any r> and the Boolean guard function G1($r_h \leq r_y$ or $r_y \leq r_h$) prevent firing of transition *assignr*$_y$ for user u_z, if user u_z is authorized for role r_h and either r_h inherits r_y or r_y inherits role r_h. Consequently, no two roles assigned to same user can inherit (directly or indirectly) one another.

Lemma 3: Given P_{RBAC} with a consistent initial state M_0, all the states M reachable from M_0 preserve property P4 (assignment time role specific SoD).

Proof: Role specific assignment time SoD constraint states that conflicting roles cannot be authorized to same user. Assume that for some user to role assignment this condition does not

hold, i.e., there exist a user u_z who can be assigned two conflicting roles r_x and r_y simultaneously. According to our assumption, tokens <u_z,r_x, any r> and <u_z,r_y, any r> can coexist in place UR. Let M_0 be the state of the system just before the firing of transition of $assignr_w$ that deposit token < u_z,r_y,r_w> where ($r_w \geq r_y$) in place UR and assume that in state M_0 role user u_z is authorized for role r_x (token <u_z,r_x,any r> present in place UR and <u_z,r_y any r> is not). Let r_y is the first role for which the role specific SoD constraint does not hold. In this case, M_0 is a consistent state if it does not break down other consistency rules. From our initial assumption, $assignr_w$ that authorized user u_z for r_y is enabled. This implies that the transition $assignr_w$ can fire. But the inhibitor arc A6 with arc expression E6:<u_z,r_c,*any r*> and the transition guard expression G2:*conflict_role_assign*(r_w,r_c) evaluating true, will prevent the transition to be enabled (a contradiction). Hence, transition $assignr_w$ will not fire in this case. So, the role specific assignment SoD constraint is preserved in any state M reachable from M_0.

Lemma 4: Given P_{RBAC} with a consistent initial state M_0, all the states M reachable from M_0 preserve properties P9(a) and P9(b) (assignment time user specific SoD).

We will first prove that property p9(a) (only one user from a conflicting set of users for role *r* is authorized for *r*) holds in all states reachable from M_0. Suppose users u_z and u_c are assignment time conflicting users for role r_y. Assume that user specific SoD does not hold in some marking M reachable from M_0, i.e., in marking M tokens <u_z, r_y, *any r*> and <u_c,r_y, *any r*> can coexist in place UR. Consider the state of the system M' just before the firing of some transition $assignr_x$ that deposit token <u_c,r_y,r_x> where ($r_x \geq r_y$) in place UR. Assume that role u_z is already authorized for role r_y in state M'. Also, assume that u_c is the first user for which the user-specific SoD does not hold. We can safely assume that the state M' is consistent provided it does not violate other consistency rules and is reachable from M_0. From our initial assumption transition $assignr_x$ is enabled with the system in state M'. This implies that the transition $assignr_x$ can fire. But the inhibitor arc A7 with arc expression E7: <u_c, r_y, *any r*> and the transition guard expression G3:*conflict_user_assign*(r_x,r_x,u_z,u_c), evaluating true, will prevent the transition to be enabled. Hence the transition $assignr_x$ that authorize user u_c for role r_y, will not fire in this case. So, the user specific assignment SoD constraint is preserved in any state M reachable from M_0.

For property P9(b), suppose that role r_y inherits role r_x ($r_y \geq r_x$ and $r_y \neq r_x$). By property P8 conflict_user_assignset(r_x) $\subseteq$ conflict_user_assignset(r_y). Let C= conflict_user_assignset(r_x) $\cap$ conflict_user_assignset(r_y) be a non-empty set. If |C|=1, then property P9(b) holds trivially in any marking reachable from M_0. For the case |C| $\geq$ 2,. Let u1 and u2 $\in$ C and assume that property P9(b) does not hold in some marking M reachable from M_0 implying that tokens <u1,r_x, r_x > and <u2,r_y, r_y > can coexist at place UR in the marking M. Consider the state of the system M' just before the firing of transition of $assignr_y$ that deposit token <u2,r_y,r_y > in place UR and assume that role r_x is already assigned to user u1 in state M' (token <u1,r_x,r_x > present at place UR and <u2,r_y,r_y > is not). We can safely assume that M' is reachable from M_0. From our initial assumption transition $assignr_y$ is enabled with the system in state M'. This implies that the transition $assignr_y$ can fire. But the inhibitor arc A7 with arc expression E7<u_c,r_z,any r> (in this case u_c=u2, r_z=r_x), and the transition guard expression *conflict_user_assign*(r_y,r_x,*u1,u2)*, (*conflict_user_assign*(r_y,r_x,*u1,u2)* is true because u1,u2 $\in$ C $\subseteq$ *conflict_user_assign*(r_y)) , will prevent the transition to be enabled. Hence the transition $assignr_y$ will not fire in this case, which is contrary to our assumption. Similarly it can be shown that if r_y is assigned to u1 then r_x cannot be assigned to u2.

Property P3 maintains that r_x and r_y cannot be assigned to same user. So, from the above argument and property P3, if two role r_x and r_y with $r_y \geq r_x$ have a common set of assignment time

conflicting users, then only one user from the common set can be assigned any one of the two roles r_x and r_y and not both.

Lemma 5: Given P_{RBAC} with a consistent initial state M_0, all the states M reachable from M_0 preserve property P10 ($\forall u \in USERS, active\_roleset(u) \subseteq authorized\_roleset(u)$).

Proof:

The input read arc A28 between the input place UR and transition $activater_y$ ensures that transition $activater_y$ can fire for user u_z only if a token $<u_z,r_y,$any r> is present in the place UR. The token $<u_x,r_y,$any r > in place UR shows that user u_z is either assigned role r_y ($r_y=r$) or is authorized for role r_y by virtue of role r assigned to u_z such that ($r \geq r_y$ and $r \neq r_y$). Hence the active role set of any user is a subset of his/her authorized roles.

Lemma 6: Given P_{RBAC} with a consistent initial state M_0, all the states M reachable from M_0 preserve property P11 (activation time role specific SoD).

Proof:

Can be proved with similar argument as lemma 3 was proved for static case.

Lemma 7: Given P_{RBAC} with a consistent initial state M_0, all the states M reachable from M_0 preserve property P16 (activation time user specific SoD).

Proof:

Can be proved with similar argument as property P9(a) is proved for static case in lemma 4.

Lemma 8: Given a P_{RBAC} structure with initial marking M_0 in which all precedence constraints are satisfied, then the subsequent markings also satisfy all the precedence constraints, i.e., properties P17 – P22 are preserved.

Lets take the enabling time precedence constraint which says that if role some role r_y has enabling time precedence constraint, then role r_y can only be enabled if all roles *r'* are enabled, where**,** $r' \in R_i$ for some $R_i \in$ *prec_enableset(r_y).* For the sake of contradiction, suppose that this does not hold for a marking M which is reachable from M_0 in the P_{RBAC} structure. Without loss of generality, assume that *prec_enableset(r_y)* = $\{R_1, R_2, \ldots, R_n\}$, where $R_i = \{r_{i1}, r_{i2}, \ldots r_{im}\}$. Our assumption implies that in marking M, the place ER has token $<r_y>$, but it does not have all <*r'*> tokens, where, $r' \in R_i$ for some $R_i \in$ *prec_enableset(r_y)***.** Without loss of generality, assume that M is the first marking in which r_y is in enable state. This means that marking M is achieved by firing of transition $enabler_y$ from marking M' which does not have all <r'> tokens in enable state, where, $r' \in R_i$ for some $R_i \in$ *prec_enableset(r_y).* But the read arc A21 and the associated guard function G10: *prec_enable(r_y,{r'})* of the transition $enabler_y$ will prevent $enabler_y$ to fire. This contradicts the assumption that marking M does not preserve the enabling time precedence constraint. Hence, all markings reachable from M_0 in the P_{RBAC} structure preserve the enabling time precedence constraint.

Similarly, it can be proved that all markings reachable from M_0 in the P_{RBAC} structure preserve all precedence constraints defined in section 3.1.1.

Lemma 9: Given a P_{RBAC} structure with initial marking M_0 in which all dependency constraints are satisfied, then the subsequent markings also satisfy all the dependency constraints, i.e., properties P23 – P28 are preserved.

Proof:

We will prove this lemma for enabling time dependency constraint only, the assignment and activation time precedence can be proved in a similar way. Note that enabling time dependency constraint poses two conditions on role r_y and r_{de} with $r_{de} \in dep\_enableset(r_y)$: 1) role r_{de} cannot be enabled if role r_y is not enabled, and 2) role r_y cannot be disabled if role r_{de} is in enable state. The first condition can be satisfied by defining a precedence constraint between role r_{de} and role r_y, i.e., $\{r_y\} \in prec\_enableset(r_{de})$. To satisfy the second condition, the PRBAC structure has an inhibitor arc A23 (E23: $<r_{de}>$between place ER and transition $disabler_y$ and the transition guard function G11: $dep\_enableset(r_y, r_{de})$, which prevents the transition $disabler_y$ to fire if role r_{de} is in enable state. Consequently, role r_y cannot be disabled if role r_{de} is in enable state.

Theorem 1: Given a P_{RBAC} (CPN structure for RBAC) structure with an initial consistent state M_0, all states M, reachable from M_0 are consistent.

Proof:

Since the properties P1-P28 are non-interfering, therefore the proof of this theorem immediate from lemma 1- 9.