Using Timed Colored Petri Nets and CPN-tool to Model and Verify TRBAC Security Policies

Laïd Kahloul
LISSI Laboratory,
Paris Est University,
Paris, France.
Computer Science Department,
Biskra University, Algeria,
laid.kahloul@univ-paris12.fr

Karim Djouani
LISSI Laboratory,
Paris Est University
Paris, France.
F'SATI at TUT,
Pretoria South Africa.
djouani@univ-paris12.fr

Walid Tfaili
LISSI Laboratory,
Paris Est University
Paris, France.
tfaili@univ-paris12.fr

Role Based Access Control (RBAC) is one of the most used models in designing and implementation of security policies in large networking systems. The classical model doesn't consider temporal aspects which are so important in such policies. Temporal RBAC (TRBAC) is proposed to deal with these aspects. Although the elegance of these models, design a security policy remains a challenge. One is obliged to prove the consistency and the correctness of the policy. Using formal verification allows proving that the designed policy is consistent. In this paper¹, we present a formal modelling/analysis approach for TRBAC policies. We use Timed Colored Petri Nets to model the TRBAC policy, and then CPN-tool is used to analyze the generated models. The analysis allows proving many important properties about the TRBAC security policy.

Security Policies, RBAC model, TRBAC, Timed Colored Petri Nets, Formal Modelling, Formal Verification.

1. INTRODUCTION

The complexity of security administration remains an important challenge in large networks management. Role based access control (RBAC) [3], is one of the predominant models for advanced access control because of its ability to reduce the complexity. In [4], a unified model for RBAC was published as the NIST RBAC model and was adopted as an ANSI/INCITS standard. This standardization motivates most information technology vendors to incorporate RBAC into their product lines. The RBAC model defines four basic components: Users, Roles, Permissions, and Sessions. A user is a human or a process within a system. A role is a collection of permissions associated with a certain job function. 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.

The use of RBAC is finding applications in areas ranging from health care to defense, in addition to

the mainstream commerce systems for which it was designed. In [13], RBAC model is used to implement a secure access policy in Electronic health information systems. This system is designed to offer better health care services for patients and to help doctors and other health care workers to treat and diagnose diseases. RBAC access policies ensure high information security and stringent access control for patient's health data. This is necessary to protect the patient's privacy and prevent a harmful or illegal use of data. In [11], RBAC is applied in an office automation system used in several colleges. The authors proved that web applications based on the RBAC model has excellent safety and stability. In [12], authors applied RBAC model to secure Web-based workflow systems. In this work, RBAC is used to control access (without hindering the process). In this last work, RBAC facilitates the access control management.

Although the advantages of RBAC model, it does not explicitly model different states of a role. So, the RBAC does not capture various events that are typical of an RBAC system. To cover this limit, some variants of basic RBAC have been developed. The most important is Temporal RBAC (or TRBAC) [5] to handle temporal contexts. This one is then extended to General Temporal RBAC (or GTRBAC) [6]. TRBAC model [5] allows: (i) temporal constraints on the events of activation and

¹ This work was supported by the MULTIPOL European project (ITEA2). MULTIPOL aims to create a modular, innovative and consistent security suite. The goal is to have strong security features to manage independently administrated domains

deactivation of roles, (ii) periodic role enabling and disabling and (iii) temporal dependencies among such actions. The GTRBAC [6] defines more specific temporal constraints such as temporal constraints on user-role and role-permission assignments/de-assignments, role activation-time constraints, etc. These temporal constraints are necessary in capturing the dynamic behavior of systems that use RBAC.

Ensuring the consistence and the correctness of access control policies is a challenging problem. Along with the development of access control models, it is necessary to perform security analysis. Formal verification techniques have been used to analyze such policies. Petri nets are well suitable to describe discrete processes and to analyze the system concurrency and synchronism. Petri net has a graphical representation and a well-defined rigorous semantics. These characteristics allow Petri nets to be a good formalism to specify and analyze the access control policies. In the literature, we find some works which applied Petri nets to model RBAC policies. In [14-18], Petri nets are used to specify access control policies in workflow systems. In [19], security attributes in mandatory access control are analyzed using reachability of Petri net. In [20], the authors use coverability graph to analyze policies. In [21], a formal security model based on Colored Petri Net is proposed and used to show the analysis and construction methods to information flow security. With respect to [22], the above techniques don't give the consistency verification of policies. Some more important works have used Colored Petri Nets [11, 12, 13, 14, 22] where the coverability tree (or graph) is used to analyze some properties and the consistency of the RBAC policies. We will present a more detailed discussion about these last works in section two.

This paper, will present a formal approach to model and verify TRBAC policies using Timed Colored Petri Nets (CPN) and CPN-tool [23]. In this approach, we define the events that can occur in the system and their preconditions and postconditions. These preconditions and postconditions specify the TRBAC constraints that should be satisfied.

The rest of the paper is organized as follows: section two presents some details about the RBAC and TRBAC model, what constraints to be analyzed formally, and discuss some related works. Section three presents the Timed Colored Petri Nets formalism and its dynamic behaviour. Section four presents the modelling process and the specifications of TRBAC into CPN. Section five presents the verification phase using CPN-tool. Finally, section six concludes this paper.

2. THE USE OF PETRI NETS TO SPECIFY RBAC POLICIES

A policy is a set of rules that define the behavior of a system. The system that uses this policy is expected to satisfy this set of rules in all its states. A state, where one of these rules is not respected, is called an inconsistent state. An inconsistent state is reached if the policy itself contains an inconsistency or because it is incomplete. When RBAC model is used to define a policy, the set of rules are defined through the basic concepts used in the RBAC model: Users, Roles, Permissions and Sessions. The consistency rules in a policy are specified as a set of constraints in the RBAC model. In the basic RBAC, these constraints are classified into three classes: (i) Cardinality constraints, (ii) Separation of duties (SoD) constraints, (iii) Inheritance constraints. In the TRBAC, another class of constraints is considered: Temporal constraints on enabling and disabling roles.

In a formal specification of an RBAC policy, we must specify the RBAC constraints to be satisfied in all states of the system. After the specification is done, the formal verification consists on proving that all reachable states during the execution of the system are consistent with respect to the set of predefined constraints. In [7], the authors defined the ConPN (Conflict Petri Nets) formalism based on CPN (Colored Petri Nets). This formalism is used to find all potential conflicts in an inheritance policy in RBAC: (i) role inheritance conflicts (a role inherits permission that it should not have), (ii) separation of duty (SoD) conflicts (a role accessed by two conflicting users at the same time), (iii) cardinality conflicts (number of users doing a role greater than permitted), and (iv) temporal restrictions conflicts (a user accesses a role in a non permitted time). In [8], authors use CPN without guards to conflicts detection in an interoperation of RBAC policies. More than role inheritance conflict (as in [7]), they studied (i) cardinality conflicts (on roles, on users, and on objects), (i) SoD conflicts (conflicting users on some roles, conflicting roles for some users), (iii) resources sharing conflicts (which can cause deadlock in the interoperation). In [9], the authors use CPN with guards and inhibitor arcs. In this work, the four events (assignment, de-assignment, activation, and de-activation) in an RBAC system are modelled as four transitions in the CPN. They considered the same constraints about: cardinalities, SoD, and inheritance as in [11, 12]. The originality in this last work can be the consideration of two temporal constraints (defined in the GTRBAC): dependency and precedence constraints between the activation, and the assignment of roles. In [10], the authors present a work similar to that presented in [9]. They use the same model CPN to model the SA-RBAC (self authenticated RBAC). In this model, permissions are of two kinds (general and sensitive) and users, who are interested to access sensitive permission, are required to have a self authentication.

Although some of the previous works tried to model the temporal constraints and so the GTRBAC model, we remark that temporal aspects and temporal constraints defined in TRBAC were not well studied (periodic event that enable or disable roles, triggers that can also change the status of roles). The originality of the present work is : (i) The use of Timed Colored Petri Nets, and so CPN-tool [23] to analyze the policy, (ii) the specification of more constraints that are addressed in previous works (as temporal constraints on enabling and disabling roles).

3. TIMED COLOURED PETRI NETS

Informally, A Petri Nets (PN) [2] is a graph composed of two kinds of nodes places and transitions. A set of arcs link places to transitions and transitions to places. The places can be marked with tokens (modeling non typed data). These tokens are called marking of the PN. Transitions can be enabled, and if this is the case they can be fired. Firing a transition update the marking of the places in the net.

Colored Petri Nets (CPN) [1] is an extension of Petri Nets. In CPN, each place has a type (a color). So the tokens can be more complex and typed data. The arcs are labeled by expressions that belong to the types of their incoming places. The transition can have some guards. A guard is a boolean expression. In a guard expression, we can use variables that are used in the input arcs or the output arcs of the transition. In this section, we present the formal definition of a CPN, and we show the dynamic behavior of this formalism.

Timed CPN extends CPN with a set of *stamps* of time. These stamps can be associated to tokens or to transitions. A stamp s associated to a token will make this token ready to be used only after that the time of the system will be more than s . When a stamp s is associated to a transition, all the stamps associated to tokens that are generated, when this transition is fired, are incremented with the stamp s . The following definition that we present in the following paragraphs is inspired from the definition of CPN [1]. Our definition of Timed CPN updated the definition of CPN with the concept of stamps, as implemented in the CPN-tool [23].

3.1 Formal Definition

Firstly, we present some necessary concepts which will be used in the formal definition of the Timed CPN. Let N denote the set of non-negative integers.

Definition 1. A **Time set** T is a set of non negative integers. $T = \{t \in N\}$.

Definition 2. A **timed multi-set** tm , over a non-empty set X , is a function $tm \in [X \rightarrow N^*T]$, for each

$x \in X$, $tm(x) = (O(x), S(x))$, where $O(x) \in [X \rightarrow N]$, is the number of occurrences of x and $S(x) \in [X \rightarrow T]$ is a *stamp* (from a Time type that can be the set of positive integers). $tm(x)$ is represented as a formal sum: $\sum_{x \in X} (O(x) \cdot x @ + S(x))$.

By X_{TMS} we denote the set of all timed multi-sets over X . The non-negative integers $\{O(x) \mid x \in X\}$ are the **coefficients** of the multi-set, and the non-negative integers $\{S(x) \mid x \in X\}$ are the stamps of the multi-set.

For example, if we take the set $\{1, 2\}$. A timed multi-set of this set can be $(2 \cdot 1 @ + 2) + (1 \cdot 2 @ + 4)$. This timed multi-set represents the set that contain two occurrences of 1 (and each occurrence has the stamp=2) and one occurrence of 2 (with a stamp=4). $\{1, 2\}_{TMS}$ is the set of all timed multi-sets of the set $\{1, 2\}$, which is an infinite set.

In the following definition, some keywords are used to facilitate the presentation. We use **Type(E)** to denote the type of the expression E . We use **Var(E)** to extract the set of variables used in the expression E .

Definition 3. A **Timed CP-net** is a tuple $CPN = (\Sigma, P, T, A, C, G, E, I, \tau)$, where:

(i) Σ is a finite set of non-empty **types**, also called colour sets.

(ii) P is a finite set of **places**.

(iii) T is a finite set of **transitions**.

(iv) A is a finite set of **arcs** such that:

$$A \subseteq (P^*T) \cup (T^*P).$$

$$\text{and } P \cap T = P \cap A = T \cap A = \emptyset.$$

(v) C is a **colour** function (it defines the type of each place in the CPN). It is defined from P into Σ .

(vii) G is a **guard** function. It is defined from T into expressions such that:

$$\forall tr \in T: \text{Type}(G(tr)) = \text{Boolean and}$$

$$\text{Type}(\text{Var}(G(tr))) \subseteq \Sigma.$$

(viii) E is an **arc expression** function. It is defined from A into expressions such that:

$$\forall a \in A: \text{Type}(E(a)) = C(p)_{TMS} \text{ and}$$

$$\text{Type}(\text{Var}(E(a))) \subseteq \Sigma. \text{ Where } p \text{ is the place component in } a.$$

(ix) I is an **initialisation** function (or an initial marking of the set of places). It is defined from P into closed expressions such that: $\forall p \in P: \text{Type}(I(p)) = C(p)_{TMS}$.

(x) τ is a temporal function that associated to each transition a stamp, $\tau: T \rightarrow T$. (T is a time set).

3.2 Dynamic behaviour

The dynamic behaviour of the net is obtained when the transitions are fired. A transition can be fired if it is enabled. A transition requires some preconditions to be enabled. These preconditions depend on the marking of its input places, the expressions labelling its input arcs, and its associated guards. Once the transition is fired some post-conditions will be satisfied. Firing the

transition will update the marking of its input and output places. The new marking depends on the expressions labelling the input and output arcs of this transition. To present the preconditions of firing a transition and how the marking is updated, we present firstly some necessary concepts.

We use the notation $\text{Var}(tr)$ to extract the set of variables used in the guards associated to the transition tr , or used in the expressions labelling input or output arcs of tr .

Definition 3. A **binding** of a transition tr is a function b defined on $\text{Var}(tr)$, such that:

- (i) $\forall v \in \text{Var}(tr): b(v) \in \text{Type}(v)$.
- (ii) $G(tr) \langle b \rangle$. The binding satisfies the guard function of tr .

Definition 4. A **timed-binding** of a transition tr is a couple $\langle b, t \rangle$, where b is a binding defined on $\text{Var}(tr)$, t is a time and at the time t , we have :

- (i) $\forall v \in \text{Var}(tr): b(v) \in \text{Type}(v)$.
- (ii) $G(tr) \langle b \rangle$.

Definition 5. if X is the set of tokens in p at time t , the **timed-marking** of the place p at the time t , denoted $M_t(p)$ is defined as the multi-set of tokens x in p with a stamp less than or equal t .

$$M_t(p) = \sum_{x \in X \text{ and } S(x) \leq t} (O(x) \cdot x @ + S(x))$$

The **initial timed-marking** denoted M_0 is the timed-marking of the net at time 0.

Definition 6. A transition tr is **time-enabled** at time t in a marking M iff there is a timed-binding $\langle b, t \rangle$ which satisfies the following property:

$$\forall p \in P: E(p, tr) \langle b \rangle \leq M_t(p).$$

We write that $(tr, \langle b, t \rangle)$ is time-enabled at time t .

Definition 7. Let X be a timed multi-set, E is an expression defined over X , and t is a stamp, the expression $E @ + t$ denotes the expression E in which the stamps of all its operands are incremented with t . for example, if $E = (2 \cdot 1 @ + 2) + (1 \cdot 2 @ + 4)$, then $E @ + 2 = (2 \cdot 1 @ + 4) + (1 \cdot 2 @ + 6)$.

Definition 8. When a transition tr is time-enabled at time t , in a timed-marking M_t , it can **be fired**. Firing tr is an event that can takes a duration Δt . Firing tr changes the marking M_t to another marking $M_{t+\Delta t}$, such that:

$$\forall p \in P,$$

$$M_{t+\Delta t}(p) = (M_t(p) - E(p, tr) \langle b \rangle) + (E(tr, p) \langle b \rangle @ + \tau(tr)).$$

Definition 9. We say that $M_{t+\Delta t}$ is **directly reachable** from M_t . This is written: $M_t [tr] M_{t+\Delta t}$.

4. MODELLING TRBAC POLICY USING TIMED CPN

When modelling a system with Petri nets, we are obliged to define the set of events. These events change the state of the system when they occur. In our case (an access control policy using TRBAC model), we distinguish six major events. These events are responsible of the modification of the status of roles defined in the system. The events are: *Enableness* of a role, *Disableness* of a role, *Assignment* of a role to a user, *Activation* of a role by a user, *Deactivation* of a role, and *De-assignment* of a role. Each of these six events, require some preconditions (some constraints of the TRBAC model) to be satisfied, and once these preconditions are verified, some postconditions will be satisfied (also some other constraints of the TRBAC model). These events are modelled by transitions. The postconditions are modelled by some input places, some expressions of input arcs, and some associated guards. The preconditions are modelled by some output places and some expressions of output arcs. In the following paragraphs, we will show in details the modelling process.

The following subsection 4.1 presents the modelling of the *enableness/disableness* events with Timed CPN and subsection 4.2 presents the modelling of the other events (*Assignment and Activation* of a role by a user) with CPN.

4.1 Modelling of Temporal Constraints

In the TRBAC model, the two events *enable* and *disable* can be **prioritized** and **temporized**. The priority is an integer that is associated to an event, and which is defined with respect to the other events that can occur in the system. The TRBAC version introduces temporal constraints on role *enabling* and *disabling* events. These constraints are presented through two concepts: *periodic events* and *roles triggers*.

- A periodic events is expressed as: $\langle I, P, pr: E \rangle$, where: I is an interval, P : a periodic expression, pr a priority, and E an event (which can *enable* a role or *disable* a role). As an example of a periodic event: $\langle [01/01/2000, \infty], \text{Night_time}, VH: \text{disable_doctor_on_night_duty} \rangle$, which means that the role *doctor_on_night_duty* must be disabled with a very high priority in night time from 01/01/2000 and forever.
- A role trigger is expressed as: $\langle E_1, \dots, E_n, C_1, \dots, C_m \rightarrow pr: E \text{ after } \Delta t \rangle$, where $E_1, \dots, E_n$ are a set of events; $C_1, \dots, C_m$ are a set of statuses; pr is a priority; E : is an event; and Δt is a duration of time. These trigger means that once the set of events $E_1, \dots, E_n$ occur, and if the system contains the set of statuses $C_1, \dots, C_m$, then the event E will occur with a priority pr , after a duration Δt . For example, the trigger: $\langle \text{enable_nurse_on_day_duty} \rightarrow H: \text{disable_nurse_on_training after 2 days} \rangle$ means that the event *disable_nurse_on_training* will occur with a

High priority after 2 days of the occurrence of the event: *enable_nurse_on_day_duty*.

At runtime, roles can be enabled and disabled through requests. In TRBAC, a runtime request expression has the form $\langle \text{pr: E after } \Delta t \rangle$, where pr is a priority, E is an event, and Δt a duration of time. For example the $\langle \text{h: enable_nurse_on_day_duty after 2 hours} \rangle$, is a request to enable the role *nurse_on_day_duty* after two hours. This request has as priority h . In periodic events, role triggers, and runtime request expressions, the value pr (priority) and Δt can be omitted. If this is the case, pr is sited to the low level priority, and Δt to 0. In the Timed CPN model we define the following types:

- **Role_Index:** Elements in this type models roles. A role index is an integer (INT in the specification). In the specification a role R_i is expressed as $R(1)$.
- **PeriodicEvent:** This type is the product of $\text{Interval} * \text{Period} * \text{Prio} * \text{Role_Index} * \text{Time}$. Where Interval is a complex type $\text{INT} * \text{INT}$. For example, $(2,10)$ is an interval. Period, Prio are integer types, Time is the type *timed* (defined in CPN-tool). Tokens in the type **Periodic_Event** are stamped tokens which model periodic events in the system.
- **Event:** This type is the product of $\text{Prio} * \text{Role_Index} * \text{Time}$. Tokens in the type Event are stamped tokens which model incoming request events due to periodic events.

Figure 1: Timed CPN model for enabling/disabling roles.

- **RQEent:** This type is the product of $\text{INT} * \text{Role_Index} * \text{Time}$. Tokens in the type are

stamped tokens which model request events in the system. These requests are due to incoming periodic events or are injected directly in the system.

In the Figure 1, place EPE, DPE will contain periodic (enabling or disabling) events. The expression $((i,j),per,pr,r)$ models a set of periodic events that will enable (case of the arc (EPE, arrival_EPE)) or disable (case of the arc (DPE, arrival_DPE)) the role r , with a priority pr . The occurrences of these events are limited in the interval of time $[i,j]$, and they occur after each period equal to per . When the transition **arrival_EPE** is fired, it puts an event (into the place EE) that will enable some role r after a duration of time per (as the delay associated to the transition is $@+per$). When the transition **arrival_DPE** is fired, it puts an event (into the place DE) that will disable some role r after a duration of time per (as the delay associated to the transition is $@+per$). The two expressions E1 , and E2 , will re-put the periodic event into the places EPE, and DPE if the interval $[i,j]$ is always valid ($i \leq j$). The two expressions are two conditional statements (in ML Meta-language, used in the CPN-tool). As an example:

```
E1 = if (i+per < j) then 1'((i+per,j),per,pr,r);
This means that:
marking(EPE) := marking(EPE)
               + ((i+per,j),per,pr,r)
```

The transition **Arrival_ERQ** models incoming enabling request, the transition **Arrival_DRQ** models incoming disabling request, and the transition **Arrival_CRQ** models incoming of concurrent events (enabling and disabling). The places AEE, ADE are used to disable transitions **Arrival_ERQ**, and **Arrival_DRQ**, and to impose the firing of **Arrival_CRQ** when there are two tokens at the same time in the places EE and DE. In this case, the transition **Arrival_CRQ** decides what kind of request must be putted into the place RQ. The decision is made with respect to the priority of each event (the attributes $pr1$ and $pr2$). The expression E3 is a conditional statement:

```
E3 = if (r1=r2) then
      if (pr1 < pr2) then 1'(0,r1)
      else if (pr1 > pr2) then 1'(1,r1)
      else empty
    else 1'(1,r1)++1'(0,r2)
```

The place RQ will contain the set of requests. A request is a time stamped token. This token is a couple composed of a role r and an integer which is "1" for enabling request and "0" for the disabling request. Finally, transitions **disable_r** and **enable_r**, will enable or disable some role. Of course, to enable a role, this one must be disabled at the current time (place DR), and to disable a role, this one must be enabled at the current time (place ER). The initial marking of the two places AEE, and ADE must be a token 1 (or $1'1$). The marking of the other places depends on the scenario to be studied. In such scenario, we must identify (i) periodic events representing the marking

of places EPE, DPE, (ii) request events representing the marking of the place RQ, and finally (iii) the set of enabled roles and disabled roles, representing the marking of places ER, and DR.

However, the modeling of triggers is not so easy. Triggers can be modeled as a set of transitions. So each trigger $\langle E_1, \dots, E_n, C_1, \dots, C_m \rangle_{pr: E \text{ after } \Delta t}$ is modeled as a transition (time stamped with Δt) that has as inputs places DE, EE (for events: $E_1, \dots, E_n$), DR, ER (for roles status: $C_1, \dots, C_m$), and as output place: RQ (for requests). The Figure 2 presents the part of the TCPN model of the trigger: $\langle (1:\text{enable } R1), (2:\text{enable } R2), (2:\text{disable } R5), R1 \text{ is disabled}, R6 \text{ is enabled} \rangle \rightarrow 3: \text{disable } R4 \text{ after 4 days}$.

Figure 2: Timed CPN model for triggers

4.2 Modelling of Non Temporal constraints

In this section we present the modelling of non temporal constraints. These constraints are defined on the events: *Assignment* of a role to a user, *Activation* of a role by a user, *Deactivation* of a role, and *de-assignment* of a role.

As in the previous section, we start by introducing the types used in the specification, and then we present and we explain the model.

We define three basic types that will be used in this specification:

- **User**: to model users. Each element in this type is a triple composed of three integer elements: the user index, the static cardinality (the maximum number of roles that can be assigned to this user), the dynamic cardinality (the maximum number of roles that this user can activate simultaneously). For example: $(U(1), 2, 3)$ models the user 1, which static cardinality is 2 and which dynamic cardinality is 3.
- **Role**: to model roles. Each element in this type is a triple composed of three integer element: the role index (Role_Index used in the section 4.2), the static cardinality (the maximum number of users for which this role can be assigned), and the dynamic cardinality (the maximum number of users that can activate simultaneously this role). For example: $(R(1), 4, 5)$ models the role indexed 1, which static cardinality is 2 and which dynamic cardinality is 3.

- **Perm** (for permission): to model permission. A permission is modeled as an index (an integer). For example: $P(1)$ models a permission which index is 1.

Using these three basic types, we define the following complex types:

- **UserIndexList**, **RoleIndexList**, **PermIndexList**: three lists type. For example, an element in UserIndexList is a list of indices, and each index is the index of user. These types will be used to define more complex types used in the specification. These complex types are products of the above predefined types. For example, a list of users can be $[U(1), U(2), U(3)]$, which is a list of three users $U(1)$, $U(2)$, and $U(3)$. It is the same case for list of roles and list of permissions. For example: $[R(1), R(2)]$ is a list of two roles, and $[P(1), P(2)]$ is a list of two permissions.

- **AllowedRoles**=UserIndex*RoleIndexList. An element in this type is composed of a user index joined to a list of roles. This specifies the list of roles that a user is allowed to play. For example: in this type, the element $(U(1), [R(1), R(2)])$ specifies that the user $U(1)$ can play the role $R(1)$ and the role $R(2)$.

- **AssignedRoles**=UserIndex*RoleIndexList.

An element in this type is composed of a user index joined to a list of roles. This specifies the list of roles assigned to this use currently. For example: in this type, the element $(U(1), [R(1), R(2)])$ models that the roles $R(1)$ and the role $R(2)$ are assigned currently to the user $U(1)$.

- **UsersAssigned**=RoleIndex*UserIndexList.

An element in this type is composed of a role index joined to a list of users. This specifies the list of users for which this role is assigned currently. For example: in this type, the element $(R(2), [U(1), U(2)])$ models that the role $R(2)$ is assigned simultaneously to the users $U(1)$ and $U(2)$.

- **ActivatedRoles**=UserIndex*RoleIndexList.

An element in this type is composed of a user index joined to a list of roles. This specifies the list of roles activated currently by this user. For example: in this type, the element $(U(1), [R(1), R(2)])$ models that the user $U(1)$ has activated the role $R(1)$ and the role $R(2)$.

- **UsersActivatedRole**=RoleIndex*UserIndexList. An element in this type is composed of a role index joined to a list of users. This specifies the list of users activating currently this role. For example: in this type, the element $(R(1), [U(1), U(2)])$ models that the role $R(1)$ is activated simultaneously by the two users $U(1)$ and $U(2)$.

- **RoleHierarchy**=RoleIndex*RoleIndexList.

An element in this type is composed of a role

index joined to a list of roles. This specifies the inheritance hierarchy relation between some senior role and the list of its junior roles. For example: in this type, the element $(R(1), [R(2), R(3)])$ models that the role $R(1)$ is the senior for the two roles $R(2)$ and $R(3)$.

- **RoleAHierarchy**=RoleIndex*RoleIndexList.

An element in this type is composed of a role index joined to a list of roles. This specifies the Activity hierarchy relation between some senior role and the list of its junior roles.

- **UsersInStaticConflicts**=UserIndex*RoleIndex*UserIndexList. An element in this type is composed of a user index joined to RoleIndex joined to a list of users. This specifies the static conflict relation between some user and a list of users, about some role. For example: in this type, the element $(U(1), R(2), [U(2), U(3)])$ models that the user $U(1)$ is in static conflicts with the two users $U(2)$ and $U(3)$, about $R(2)$. This means that we could not assign $R(2)$ to $U(1)$ if it is assigned currently to the user $U(2)$ or the user $U(3)$.

- **UsersInDynamicConflicts**=UserIndex*RoleIndex*UserIndexList. An element in this type is composed of a user index joined to RoleIndex joined to a list of users. This specifies the dynamic conflict relation between some user and a list of users, about some role. For example: in this type, the element $(U(1), R(2), [U(2), U(3)])$ models that the user $U(1)$ is in dynamic conflicts with the two users $U(2)$ and $U(3)$. This means that $U(1)$ could not activate $R(2)$ which is activated currently by the user $U(2)$ or the user $U(3)$.

- **RolesInStaticConflicts**=RoleIndex*RoleIndexList. An element in this type is composed of a role index joined to a list of roles. This specifies the static conflict relation between some role and a list of roles. For example: in this type, the element $(R(1), [R(2), R(3)])$ models that the role $R(1)$ is in static conflicts with the two roles $R(2)$ and $R(3)$. This means that $R(1)$ could not be assigned to the same user that have $R(2)$ or $R(3)$ as assigned roles currently.

- **RolesInDynamicConflicts**=RoleIndex*RoleIndexList. An element in this type is composed of a role index joined to a list of roles. This specifies the dynamic conflict relation between some role and a list of roles. For example: in this type, the element $(R(1), [R(2), R(3)])$ models that the role $R(1)$ is in dynamic conflicts with the two roles $R(2)$ and $R(3)$. This means that $R(1)$ could not be activated by the same user that have activated $R(2)$ or $R(3)$ currently.

- **UserPerm**=UserIndex*PermIndexList. An element in this type is composed of a user index joined to a list of permission. This specifies the list of permission assigned to a user. For example: in this type, the element

$(U(1), [P(1), P(2)])$ models that the user $U(1)$ has the two permissions $P(1)$ and $P(2)$.

- **RolePerm**=RoleIndex*PermIndexList. An element in this type is composed of a role index joined to a list of permission. This specifies the list of permission assigned with a role. For example: in this type, the element $(R(1), [P(1), P(2)])$ models that the user $R(1)$ defines the two permissions $P(1)$ and $P(2)$.

With this list of complex types, we add the three types (which can be added only to clarify the specification): **StatCardUser**, **DynCardUser**, **StatCardRole**, **DynCardRole**. These three types are defined as integer types. An element in one of these types is an integer which specifies a static or dynamic cardinality for a user or a role. It is clear that we can use directly the **integer** type.

Based on these defined types, we will present in the following paragraphs the precondition and the post-conditions for each event in an RBAC policy. After the presentation of these preconditions and post-conditions, we show the CPN model for these events.

4.2.1 Constraints on the assignment of a role to a user

To assign a role R_i to a user U_i the following conditions must be satisfied:

- There is a user U_i and there is a role R_i : this is modeled by the existence of a token (U_i, SCU, DCU) in the place P_1 , and the existence of a token (R_i, SCR, DCR) in the place P_2 . Where, SCU , SCR , DCU , and DCR are integers that represent the static cardinality and the dynamic cardinality for the user U_i and a role R_i .
- The user U_i is allowed to play the role R_i : this is modeled by the existence of a token (U_i, RLA) in the place P_3 , where RLA is a list of roles and R_i must be in the list RL . We add the guard $[mem\ RL\ R_i]$ to the transition. This guard is true if R_i is a member in RLA .
- The role R_i is enabled: this is modeled by the fact that there is a token (R_i) in the place ER .
- The role R_i is not yet assigned to the user U_i : this is modeled by the fact that there is a token (R_i, RL) in the place P_4 and R_i is not in the list RL . We add the guard $[mem\ RL\ R_i = false]$ to the transition.
- There is no static conflict between the user U_i and the users for which R_i is assigned currently: this is modeled by the fact that there is a token (R_i, UL) in the place $P(5)$, where UL is the list of users for which R_i is assigned, and there is a token $(U_i, R_i, ULSC)$ in the place P_7 , where $ULSC$ is a list of users that have conflicts with U_i , and finally the intersection between $ULSC$ and UL is empty. We add a guard $[intersect\ ULSC\ UL = nil]$ to the transition.
- There is no static conflict between the role R_i and the roles assigned currently to U_i : this is

modeled by the fact that there is a token (U_i, RL) in the place P_4 , where RL is the list of roles assigned to U_i , and there is a token $(R_i, RLSC)$ in the place P_6 , where $RLSC$ is a list of roles that have conflict with R_i , and finally the intersection between $RLSC$ and RL is empty. We add a guard $[intersect\ RLSC\ RL=nil]$ to the transition.

- The number of assigned roles to U_i is less than the static cardinality of the user U_i : this is modeled by the fact that there is a token (R_i, SCU, DCU) in P_1 , and there is a token (R_i, RL) in the place P_4 , where RL is the list of roles assigned to R_i , and finally the number of elements in RL is less than SC . We add the guard $[length\ RL < SCU]$ to the transition.

- The number of users that have R_i in their assigned roles is less than the static cardinality of the role R_i : this is modeled by the fact that there is a token (R_i, SCR, DCR) in P_1 , and there is a token (R_i, UL) in the place P_4 , where UL is the list of users for which R_i is currently assigned, and finally the number of elements in UL is less than SCR . We add the guard $[length\ UL < SCR]$ to the transition.

After the assignment of R_i to the user U_i , we will have:

- The role R_i is assigned to the user U_i : this is modeled by two effects (i) an expression modeling the arc that updates the marking of the place P_4 (where roles assigned to each user are defined). This expression will be $(U_i, ins\ RL\ R_i)$, this means that the role R_i will be added to the list of role assigned currently to U_i . (ii) an expression modeling the arc that updates the marking of the place P_5 (where users assigned to each role are defined). This expression will be $(R_i, ins\ UL\ U_i)$, this means that the user U_i will be added to the list of users assigned currently to R_i .
- The user U_i will be allowed to all the junior roles (with respect to an AHierarchy relation) of R_i : This is modeled by the updating of the place P_3 with the expression $(U_i, union\ RL\ RLAH)$. RL is the old list of roles allowed to U_i , and $RLAH$ is the list of R_i 's juniors roles. The "union $RL\ RLAH$ " is the union of the two lists. The list $RLAH$ is obtained from an input expression $(R_i, RLAH)$ from the place P_8 .

According to the above description, the CPN model of the assignment event will be as shown on the Figure 3. In this Figure, types of places are near the places and their identifiers are inside the circle. The expressions labelling arcs are written near each arc. The guards associated to the transition are written in italic between brackets: $[mem\ RL\ R_i, mem\ RL\ R_i = false, intersect\ ULSC\ UL=nil, intersect\ RLSC\ RL=nil, length\ RL < SCU, length\ UL < SCR]$.

Figure 3: TCPN model for Assignment

4.2.2. Constraints on the activation of a role by a user

The activation process of a role R_i by a user U_i is processed through three steps (modelled by four transitions in the CPN). Firstly, we check that all necessary conditions to activate R_i by U_i are satisfied. If this is the case, the role is added to the activated roles of U_i , and a first transition (**activate**) is fired. Secondly, we check if the role R_i has some junior roles (with respect to the Inheritance Hierarchy relation); if this is the case then all these junior roles must be identified. This identification is done by the two transitions (**inhiriting_roles_1** and **inhiriting_roles_1**). Thirdly, the set of permissions of the role R_i and all its junior roles are assigned to the user U_i (using the transition **taking_permissions**). During these three steps, the U_i and the role R_i must not be used by another transition (for example assign), and once these steps are finished, a fifth transition (**restore**) restores U_i and R_i to their places P_1 , and P_2 .

The subnet that models the activation process shares the places P_1 , P_2 , P_4 with the subnet modelling assignment.

The conditions necessary to fire **activate**:

- There is a user U_i and a role R_i : this is modelled by the existence of a token (U_i, SCU, DCU) in the place P_1 and a token (R_i, SCR, DCR) in the place P_2 .
- The role R_i is assigned to the user U_i : this is modelled by the fact that there is a token (U_i, RL) in the place P_4 , where RL is the assigned roles to U_i , and the R_i is a member in the list RL . We add the guard $[mem\ RL\ R_i]$ to the transition **activate**.
- The role R_i is not yet activated by U_i : this is modelled by the fact that there is a token $(U_i, RLAct)$ in the place P_9 , where $RLAct$ is the list of activated roles by U_i , and the R_i is not a member in the list $RLAct$. We add the guard $[mem\ RLA\ R_i=false]$ to the transition **activate**.

- The role R_i is not in a dynamic conflict with one of the currently activated roles by U_i : this is modeled by the fact that there is a token $(R_i, RLDC)$ in the place P_{10} , where $RLDC$ is the list of roles which are in dynamic conflicts with R_i , and a token $(U_i, RLAct)$ in the place P_9 , where $RLAct$ is the list of activated roles by U_i , and finally the intersection between the two lists $RLAct$ and $RLDC$ is empty. We add the guard $[intersect\ RLAct\ RLDC=nil]$ to the transition **activate**.
- The user U_i is not in a dynamic conflict with one of the users activating the role R_i : this is modelled by the fact that there is a token $(U_i, R_i, ULDC)$ in the place P_{11} , where $ULDC$ is the list of users which are in dynamic conflicts with U_i , and a token $(R_i, ULAAct)$ in the place P_{12} , where ULA is the list of users activating the role R_i , and finally the intersection between the two lists $ULAAct$ and $ULDC$ is empty. We add the guard $[intersect\ ULAAct\ ULDC=nil]$ to the transition **activate**.

- The number of roles activated by U_i is less than the dynamic cardinality of the user U_i : this is modeled by the fact to have a token (U_i, RLAct) in the place P_9 , where RLAct is the list of activated roles by U_i , and the number of

token (R_i , RLIH) from the place P_{13} , where RLIH is the list of junior roles of R_i . This list is then puts into the place P_{16} . (ii) The transition **inheriting_roles_2** will be fired iteratively to extract all roles from the list RLIH. Once **inheriting_roles_2** is fired, it puts a role in the place P_{18} . (iii) The transition **taking_permissions** checks the place P_{18} , and once a role R_j is found in P_{18} , this transition will use a token (R_j , PLR), (where PLR is the list of permissions defined in R_j) from the place P_{19} to update the marking of the place P_{20} with the token (U_i , union ULP RLP). The expression *union ULP RLP* is evaluated to the union of the list RLP and the list ULP (the permissions assigned previously to the user U_i (iv). Once the list RLIH is empty, the transition **restore** will re-put the user U_i and the role R_i into their places P_1 , P_2 , and ER (enabled Roles).

According to the above description, the CPN model of the activation event will be as shown on Figure 4.

5. MODELS VERIFICATION

The verification of the model is realized using the CPN-tool [23]. CPN tool is a free software that can be used to model systems with colored Petri Nets (Timed and hierarchical also). It allows simulations and analysis of different kinds of proprieties. In the case of the CPN model for an RBAC policy, the initial marking of the model represents the initial state of the system. This initial state can be also considered as a scenario example of an RBAC policy. The properties that we must verify depend on the scenario and the system developed. In this section, we present a simple RBAC scenario, and we will show as an analysis process the reachability graph that will be drawn by the CPN. This reachability graph can be used to prove some proprieties and to answer some questions that we will discuss after.

5.1 A scenario example

Let's consider a system in which we have **two users** (i) $U(1)$ with 2 as static cardinality and 1 as dynamic cardinality (ii) $U(2)$ with 1 as static cardinality and 1 as dynamic cardinality; **three roles** $\{R(1), R(2), R(3)\}$. The role $R(1)$ with 2 as static cardinality and 1 as dynamic cardinality, and permissions $\{P(1), P(2)\}$. The role $R(2)$ with 1 as static cardinality and 1 as dynamic cardinality, and permission $\{P(2)\}$. The role $R(3)$ with 1 as static cardinality and 1 as dynamic cardinality and permission $P(4)$. We consider that $U(2)$ could not practice the role $R(2)$, and that $R(3)$ could not be assigned simultaneously to $\{U(1), U(2)\}$. We consider that the Role $R(3)$ is a junior role to $R(1)$. This scenario will represent the initial marking of the CPN model.

5.2 The analysis

The reachability graph is a representation of the state space of the system. If this graph can be computed, so the system has a finite space state. In this case, lot of proprieties can be deduced: reachability of some state, home state, boundedness of places, deadlock or liveness of transitions, ... For example for the above scenario, the CPN tool generates a reachability graph with 755 nodes and 1388 arcs (figure 4 shows some nodes of the reachability graph). All places in all states are bounded. This propriety allows us to verify if some bad marking is reachable. By bad marking, we mean here some state where the constraints are not satisfied. In our case, the scenario specified has no problem.

Figure 5: Some nodes from the reachability graph

5.3 Some important properties of the RBAC policy and their interpretation in the reachability tree

1) Can a user $U(i)$ play all its allowed roles?

This is formulated as the following: from each state where $U(i)$ is allowed to play roles $\{R(1), \dots, R(n)\}$ (which means a state where the marking of P_3 contains $(U(i), [R(1), \dots, R(n)])$), and for each $R(i)$ in $\{R(1), \dots, R(n)\}$, we must find some path in the reachability tree in which there is at least one binding: $\langle \text{assign}, U_i=U(i), R_i=R(i) \rangle$;

2) Can a user $U(i)$ activate all its assigned roles?

This is formulated as the following: From each state where $\{R(1), \dots, R(n)\}$ are assigned to $U(i)$ (which means a state where the marking of P_4 contains $(U(i), [R(1), \dots, R(n)])$), and for each $R(i)$ in $\{R(1), \dots, R(n)\}$, we must find some path in the reachability tree in which there is at least one binding: $\langle \text{activate}, U_i=U(i), R_i=R(i) \rangle$;

3) A user $U(i)$ will never have a role $R(i)$ which could not be assigned to him:

This is formulated as the following: Every role $R(i)$ assigned to a user $U(i)$ in the place P_4 (AssignedRoles), must be an allowed role to the user $U(i)$, in the place P_3 (AllowedRoles).

4) A user will never activate a role which is not assigned to him:

This is formulated as the following: Every role $R(i)$ activated by a user $U(i)$ in the place

$P_9(\text{ActivatedRoles})$, must be an assigned role to this user, in the place $P_4(\text{AssignedRoles})$.

5) Cardinality constraints:

- A user will never have a number of assigned roles more than its SCU (static cardinality for a user):

This is formulated as the following: in the place $P_4(\text{AssignedRoles})$, each user $U(i)$ must have in its list of roles a number of roles less than or equal its SCU, which is defined in the place $P_1(\text{Users})$;

- A user will never activate a number of roles more than its DCU (dynamic cardinality for a user):

This is formulated as the following: In the place $P_9(\text{ActivatedRoles})$, each user $U(i)$ must have in its list of roles a number of roles less than or equal its DCU which is defined in the place $P_1(\text{Users})$;

- A role will never be assigned to a number of users greater than the SCR (static cardinality for a role):

This is formulated as the following: In the place $P_5(\text{AssignedUsers})$, each role $R(i)$ must have in its list of users a number of users less than or equal its SCR which is defined in the place $P_2(\text{Roles})$;

- A role will never be activated by a number of users greater than the DCR (dynamic cardinality for a role):

This is formulated as the following: In the place $P_{12}(\text{UsersActivateRole})$, each role $R(i)$ must have in its list of users a number of users less than or equal its DCR which is defined in the place $P_2(\text{Roles})$;

6) SoD constraints:

- A role must never be assigned to two conflicting users (Static Conflict between users):

This is formulated as the following: In the place $P_5(\text{AssignedUsers})$, for each role $R(i)$, let $LU_1=[U(1),...,U(n)]$ its list of users; for each user $U(i)$ in this list, the list of its conflict users on the Role $R(i)$, $LU_2=[U'(1),...,U'(m)]$ defined in the place $P_7(\text{UsersInStaticConflicts})$, in the token $(U(i),R(i),LU_2)$; then $LU_2 \cap (LU_1) = \emptyset$.

- A role must never be activated by two conflicting users (Dynamic Conflict between users):

This is formulated as the following: In the place $P_{12}(\text{UsersActivateRole})$, for each role $R(i)$, let $LU_1=[U(1),...,U(n)]$ its list of users; for each user $U(i)$ in this list, the list of its conflict users on the Role $R(i)$, $LU_2=[U'(1),...,U'(m)]$ defined in the place $P_{11}(\text{UsersInDynamicConflicts})$, in the token $(U(i),R(i),LU_2)$; then $LU_2 \cap (LU_1) = \emptyset$.

- A user must never have two assigned conflicting roles (Static Conflict between roles):

This is formulated as the following: In the place $P_4(\text{AssignedRoles})$, for each user $U(i)$, let $LR_1=[R(1),...,R(n)]$ be its list of roles; for each role $R(i)$ in this list, let $LR_2=[R'(1),...,R'(m)]$ (defined in the place $P_6(\text{RolesInStaticConflicts})$, in the token $(R(i), LR_2)$) the list of its conflict roles; we must have : then $LR_2 \cap (LR_1) = \emptyset$.

- A user must never activate two conflicting roles (Dynamic Conflict between roles):

This is formulated as the following: In the place $P_9(\text{ActivatedRoles})$, for each user $U(i)$, let $LR_1=[R(1),...,R(n)]$ be its list of roles; for each role $R(i)$ in this list, let $LR_2=[R'(1),...,R'(m)]$ (defined in the place $P_{10}(\text{RolesInDynamicConflicts})$, in the token $(R(i), LR_2)$) be the list of its conflicting roles; then we must have $LR_2 \cap (LR_1) = \emptyset$.

7) A user has only permissions defined in its activated roles and their junior roles (I-Hierarchy relations) even if these last ones are not activated by the user:

This is formulated as the following: Every permission assigned to a user $U(i)$ in the place $P_{20}(\text{UserPerm})$, must be a permission defined in a role $R(j)$ in the place $P_{19}(\text{RolePerm})$, and the role $R(j)$ must be activated by $U(i)$ in $P_9(\text{ActivatedRoles})$.

The CPN-tool offers the possibility to write queries using the ML (meta-language). These queries will use the reachability tree (or graph) to verify these properties.

6. CONCLUSION

The use of Petri Nets to model RBAC (Role Bases Access Control) policies is one of the ambitious axes to ensure the analysis and verification of security policies. The Petri Nets formalism is an event based formalism. So the specification with Petri Nets requires firstly defining events from the RBAC models. Most works consider the enabling, disabling, assignation, and activation of roles as basic events that must be modeled. The analysis of Petri Net model allows to the designer to prove the consistency of its policy. The analysis can be done through the reachability tree or graph. Colored Petri Nets is an extension of Petri Nets with more expressive power. The modeling of RBAC using CPN is more practical than using classical Petri Nets. The CPN has many tools that can be used to verify and to check the models.

The proposed extensions of RBAC (Temporal RBAC, General Temporal RBAC) consider temporal constraints on RBAC events. The use of a temporal formalism like (Timed CPN) is more adequate to model these constraints. In this paper, we have presented an approach to model and to analyze the TRBAC policies. This approach uses Timed CPN formalism to construct models. After the construction of model, we use CPN-tool, to verify the consistency of the model. Through this paper, we have presented how to specify several TRBAC constraints, we have presented a scenario example, and we have discussed about results of its analysis, and what kinds of properties to be analyzed.

The current work can be extended to handle more temporal constraints (those specified in the

GTRBAC). The current specification can be considered as a template where scenario of RBAC written in XML can be injected and verified. This last possibility is important because this work is a part of a project dedicated to specify RBAC policies in XML, composition of policies, the optimization of the composition, and finally the formal verification of policies. By using XML specification language, this work can be integrated with the other parts of the project.

7. REFERENCES

- Jensen, K. (1994) An Introduction to the Theoretical Aspects of Coloured Petri Nets. In J.W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.), *A Decade of Concurrency, Lecture Notes in Computer Science* vol. 803, pp. 230-272. Springer-Verlag, Berlin/Heidelberg,.
- Murata, T. (1984) Petri Nets and Their Applications: An Introduction, Chapter 20. In S. K. Chang (editor), *Management and Office Information Systems*. pp. 351-367. Plenum Press, New York,.
- Ferraiolo, D.F. and Kuhn, D.R. (1992) Role Based Access Control. *15th National Computer Security Conference*, Oct 13-16, 1992, pp. 554-563. Elsevier Advanced Technology Publications, Oxford, UK.
- Sandhu, R. Ferraiolo, D.F. Kuhn, D. R. (2000) The NIST Model for Role Based Access Control: Toward a Unified Standard. *5th ACM Workshop on Role Based Access Control*, Berlin, July 26-27, 2000, pp.47-63. ACM New York, USA.
- Bertino, E. and Bonatti, P.A. (2001) TRBAC, A temporal role based access control model. *ACM Transactions on Information and System Security*, Vol. 4, No. 3, pp. 191-223.
- Joshi, J. B. D. Bertino, E. Latif, U. Ghafoor, A. (2005) A Generalized Temporal Role Based Access Control. *IEEE Transaction on Knowledge and Data Engineering*, Vol. 17, No. 1, pp. 4-23.
- Walvekar, A. Smith, M. Kelkar, M. Gamble, R. (2006) *Using Petri Nets to Detect Access Control Violations in a System of Systems*. Technical Report SEAT-UTULSA-09-12. A shorter version of this paper appeared in the Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA'06), Seattle, August 15 - 16, 2006.
- Huang, H. and Kirchner, H. (2009) *Secure Interoperation in Heterogeneous Systems based on Colored Petri Nets*. INRIA-00396952, version 1 - 19 Jun 2009.
- Shafiq, B. Masood, A. Ghafoor, A. and Joshi, J. B. D. (2005) A Role-Based Access Control Policy Verification Framework for Real-Time Systems. *The 10th IEEE Workshop on Object-oriented Real-time Databases*, Sedona, Arizona, Feb 02nd 2004, pp. 13-20. IEEE Computer Society, Washington, DC, USA.
- Song, M and Pang, Z. (2008) Specification of SA-RBAC Policy Based on Colored Petri Net. *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. Sydney, NSW, Dec 9th-10th, pp. 207-210. IEEE Computer Society, CA, USA.
- Jie, A. (2008) The Realization of RBAC Model in Office Automation System. *The International IEEE Seminar on Future Information Technology and Management Engineering*, Leicestershire (UK), Nov 20th 2008, pp. 360-363. IEEE Computer Society, WDC, USA.
- Wang, X. Bayrak, C. (2009) Injecting a permission-based delegation model to secure web-based workflow system. *IEEE International Conference on Intelligence and Security Informatics*, Texas, USA, pp. 101-106. IEEE Press Piscataway, NJ, USA.
- Steele, R. Min, K. (2009) Role-Based Access to Portable Personal Health Records. *The International Conference on Management and Service Science*, Beijing, China, May 14th-15th, pp. 1-4.
- Knorr, K. (2000) Dynamic access control through Petri net workflows. *16th Annual Computer Security Applications Conference (ACSAC)*, New Orleans, Louisiana, USA, Dec 11th-15th, 2000, pp. 159-167. IEEE Computer Society, WDC, USA.
- Knorr, K. (2001) Multilevel security and information flow in Petri net workflows. *The 9th International Conference on Telecommunication Systems Modelling and Analysis, Special Session on Security Aspects of Telecommunication Systems*, Dallas, Texas USA, March 15th-18th, pp. 9-20. IEEE Computer Society Press, Dallas, Los Alamitos.
- Dong, X. Chen, G. Yin, J. and Dong, J. (2002) Petri net-based context related access control workflow environment. *The 7th International Conference on Computer Supported Cooperative Work in Design*, Rio de Janeiro, Brazil, Sep 25th-27th, pp. 381-384. IEEE Computer Society Press.
- Liang, Z. and Bai, S. (2006) Role based workflow modelling. *The IEEE International Conference on Systems, Man and Cybernetics*, Taipei, Taiwan, Oct 8th-11th, 2006, pp. 4845-4849. IEEE Computer Society Press.
- Jiang, Y. Lin, C. Yin, H. and Tan, Z. (2004) Security analysis of mandatory access control model. *IEEE International Conference on Systems, Man and Cybernetics*, The Hague, Pays-Bas, Oct 10th-13th, pp. 5013-5018. IEEE, Piscataway NJ, ETATS-UNIS.
- Zhang, Z. Hong, F. and Xiao, H. (2006) Verification of strict integrity policy via Petri nets. *The International Conference on System and Network Communication*, Tahiti, Oct. 2006, pp. 23-26. IEEE Computer Society, CA, USA.
- Junszczyzyn, K. (2003) Verifying enterprise's mandatory access control policies with coloured Petri nets. *The IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Linz, Austria, June 9th-11th, 2003, pp. 184-189. IEEE Computer Society, CA, USA.
- Feng, F. Lin, C. Peng, D. and Li, J. (2008) A trust and context based access control model for distributed systems. *The 10th International Conference on High Performance Computing and Communications (HPCC)*, Sep. 2008, pp. 629-634. IEEE Computer Society, CA, USA.
- Feng, F. Li, J. (2009) Verification and Analysis of Access Control Policy with Colored Petri Net. *International Conference on Communication Software and Networks*, Macau, China, feb 27th-28th 2009, pp. 610-614. IEEE Computer Society, CA, USA.
- CPN-tool can be downloaded (free for academics) from: <http://wiki.daimi.au.dk/cpntools/cpntools.wiki>.