PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
UNIVERSITY Mohamed Khider of Biskra

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي و البحث العلمي
جامعة محمد خيضر بسكرة

Faculty of Exact Sciences, Science of Nature and Life.
Computer science department

Order number: .....

Serial number : .....

THESIS

In Candidacy for the Degree of

DOCTOR 3rd CYCLE IN COMPUTER SCIENCE

Option : Artificial Intelligence

Title

Formal Modelling and Verification of Security Policies in Cloud Computing

By

Hasiba BEN ATTIA

Board of Examiners

Chairman: Pr. Okba KAZAR, Professor, Biskra university.
Supervisor: Dr. Laid KAHLOUL, M.C.A, Biskra university.
Co- Supervisor: Dr. Saber BENHARZALLAH, M.C.A, Batna 2 university.
Examinator : Dr. Hammadi BENNOUI, M.C. A, Biskra university.
Examinator : Dr. Khaled REZEG, M.C. A, Biskra university.
Examinator : Pr. Allaoua CHAOUI, Professor, Constantine 2 university.

Biskra, Algeria
24.02.2019

الـمـلـخص

تعالج الأطروحة إشكالية نمذجة السياسات الأمنية في الحوسبة السحابية و التحقق من صحتها. بشكل أدق، المشكل المعالج في البحث المطروح هو نمذجة سياسات التحكم في الوصول والتحقق من صحتها باستخدام شبكات بتري الملونة.

يعتبر نموذج التحكم في الوصول الذي يعتمد على الدور (RBAC) احد أهم السياسات الأمنية المعتمدة في معظم الأنظمة. هذه الأهمية أساسها قدرة هذا النموذج على تسهيل تسير المستخدمين و تقليص تعقيد هذه المهمة. ومع ذلك، RBAC الأساسي لا يعالج بعض الأحداث المهمة. RBAC الزمني (TRBAC) هو احد أهم امتدادات RBAC. لذلك, تتمحور مساهمتنا الأولى في نمذجة نموذج TRBAC باستعمال شبكات بتري ملونة هرمية و زمنية (HTCPN) و التحقق من النماذج الناتجة وتحليلها باستخدام وسيلة CPN-tool. الخاصية الزمنية لـHTCPN تسهل نمذجة القيود الزمنية المعرفة في TRBAC والتحقق منها. أما الجانب الهرمي فيسهل عملية التحكم في النموذج رغم تعقيد عملية وصف سياسة الTRBAC.

من جهة أخرى، يعاني RBAC من بعض السلبيات. رغم أن نموذج التحكم بالوصول القائم على السمات (ABAC) عالج هذه السلبيات، إلا انه يفتقر إلى ايجابيات RBAC نفسها. لهذا كمساهمة ثانية، تقدم الأطروحة نموذجا جديدا للتحكم قي الوصول باسم FRABAC. يتلاءم FRABAC مع طبيعة الحوسبة السحايبة من حيث المرونة، قابلية الامتداد، وكثرة التفاصيل والتفرعات. FRABAC هو دمج وامتداد للنموذجين RBAC و ABAC. لإظهار ايجابيات النموذج المقترح، قمنا بإجراء دراسة تجريبية. في هذه الدراسة تمت مقارنة FRABAC مع النماذج الموجودة بالاعتماد على مقاييس معينة. نتائج الدراسة أظهرت أن النموذج المقترح أكثر ملاءمة وفاعلية من النماذج السابقة في الشبكات واسعة النطاق. إضافة إلى كل ما سبق، و كمساهمة ثالثة نعرض في الأطروحة المقدمة تحقق رسمي و كامل للنموذج المقترح. في المنهجية الرسمية للتحقق تم استخدام HTCPN لوصف سياسة TRBAC و وسيلة tool-CPN لتحليل النتائج.

الكلمات المفتاحية: *التحكم في الوصول، سياسة الأمن، النمذجة الرسمية، التحليل الرسمي، الحوسبة السحابية، RBAC، ABAC، FRABAC.*

Summary

This thesis tackles the problem of formal modelling and verification of security policies in cloud computing. Indeed, our research focuses on the modelling and the verification of access control policies using Coloured Petri Nets (CPNs).

Due to its ability to reduce complexity, Role Based Access Control (RBAC) model was one of the predominant models for access control and the specification of security policies. In its original version, RBAC does not consider several important events, thus, TRBAC (Temporal RBAC) was proposed as an RBAC extension. This thesis provides three basic contributions. In the first contribution, HTCPNs (Hierarchical Timed Colored Petri Nets) formalism is used to model the TRBAC (Temporal Role Based Access Control) policy, and then the CPN-tool is exploited to analyse the obtained models. Indeed, the timed aspect in HTCPN allows us to deal with temporal constraints in TRBAC. The hierarchical aspect of HTCPN makes the TRBAC model "manageable", despite the complexity of the policy.

RBAC as well as TRBAC suffer from several drawbacks in large scale networks as the case of cloud environment. Although Attribute Based Access Control (ABAC) model handles some RBAC drawbacks, ABAC misses RBAC advantages. Hence, as a second contribution, we propose an access control model FRABAC (Fine-Grained Role Attribute Based Access Control) that provides scalability, flexibility, and fine granularity in the cloud environment. FRABAC combines and extends, basically, two models RBAC and ABAC. In order to demonstrate the advantages of the new proposed model, an empirical study is realised. In this study, the new proposed model is compared versus three existing models, using specific metrics. The results demonstrate that the new proposed model is more suitable than existing ones. As a third contribution, we provide a formal specification/verification of FRABAC using HTCPN formalism and CPN-tool.

Key words: *Access Control, Security Policy, RBAC, ABCA, FRABAC, Formal Modelling, Formal Analysing, Petri nets, Cloud Computing.*

Résumé

Cette thèse aborde le problème de la modélisation/vérification formelles des politiques de sécurité dans le cloud computing. En effet, nos recherches portent sur la modélisation et la vérification des politiques de contrôle d'accès à l'aide de réseaux de Petri colorés (RdPCs). En raison de sa capacité à réduire la complexité, le modèle de contrôle d'accès basé sur le rôle (RBAC) était l'un des modèles prédominants pour le contrôle d'accès et la spécification de politiques de sécurité. Dans sa version originale, RBAC ne prend pas en compte plusieurs événements importants; par conséquent, TRBAC (Temporal RBAC) a été proposé comme une extension de RBAC.

Cette thèse fournit trois contributions de base. Dans la première contribution, le formalisme HTCPNs (Réseaux de Petri Colorés Temporisés et Hiérarchiques) est utilisé pour modéliser les politiques TRBAC (Timed RBAC), puis l'outil CPN est exploité pour analyser les modèles obtenus. En effet, l'aspect temporisé dans HTCPN nous permet de gérer les contraintes temporelles dans TRBAC, et l'aspect hiérarchique du formalisme rend le modèle TRBAC "gérable", malgré la complexité de la politique.

Les systèmes RBAC et TRBAC présentent plusieurs inconvénients dans les réseaux à grande échelle, comme dans le cas de could computing. Bien que le modèle de contrôle d'accès basé sur les attributs (ABAC) gère certains inconvénients du RBAC, celui-ci manque les avantages du RBAC. Par consequence et comme une deuxième contribution, nous proposons un nouveau modèle de contrôle d'accès dit FRABAC (Contrôle d'accès basé sur les attributs et les rôles) qui offre une évolutivité, une flexibilité et une granularité fine dans l'environnement cloud. FRABAC combine et étend les deux modèles RBAC et ABAC. Afin de démontrer les avantages du nouveau modèle proposé, une étude empirique est réalisée. Dans cette étude, le nouveau modèle proposé est comparé à trois autres modèles existants, en utilisant des métriques spécifiques. Les résultats démontrent que le nouveau modèle FRABAC est plus approprié que les modèles existants. Enfin et comme une troisième contribution, nous avons réalisé une spécification/vérification formelle de FRABAC à l'aide du formalisme HTCPN et de l'outil CPN-tool.

Mots clés: *Contrôle d'accès, Politique de Sécurité, RBAC, ABAC, FRABAC, Modélisation Formelle, Vérification Formelle, Réseaux de Petri, Cloud Computing.*

Dedication

To my beloved mother, ***FATIMA ZOHRA***
To the greatest man in my life, my father ***MOKHTAR***
To my brothers, ***HAMZA****,* ***YOUNES****,* ***AHMED*** *and* ***MOHAMED***
To my biological sisters, ***WAFA****,* ***HOURIA*** *and* ***OUARDA***
To my soul sisters, ***SIHAM****,* ***SARA****,* ***SAMAH****,* ***KANZA****,* ***YAMINA****,* ***SOUMIA****,* ***IMAN****,****SIHAM****, and* ***AMINA***
To the memory of my paternal grandparents, ***AISHA*** *and* ***AHMED***
To my maternal grandparents, ***SADIA*** *and* ***SAID***
To my aunts and uncles
To my cousins, ***HANAN*** *and her siblings,* ***JOHINA*** *and her siblings,* ***AFAF*** *and her siblings,* ***NESRINE*** *and her siblings*
To all my sweet neighbours in pavilion 10
To all my teachers form the primary school to the university
To my beloved teachers, madam ***Zohra HMIDI*** *and madam* ***Amira MOHAMMEDI***
To all my colleague in LINFI laboratory

Acknowledgements

First and foremost, heartfelt gratitude and praises go to the ***Almighty Allah*** *who guided me through and through.*

I would like to extend my sincere thanks and appreciations to my honorable supervisor ***Dr. Laid KAHLOUL****. He has painstakingly corrected and recorrected draft after draft. Moreover, This work could not have reached fruition without his unflagging assistance and his participation. I would never thank him enough for the huge contribution that made this work what it is now. Once again, I would like to express my deep gratitude to my supervisor because I owe thanks to him.*

Many thanks go to ***Dr. Saber BENHARZALLAH*** *for his advice.*
Most warm gratitude goes to ***Prof. Okba KAZAR****, head of the Intelligent Computer Science Laboratory.*
My sincere appreciation needs to be addressed to the honourable board of examiners, ***Prof. Hammadi BENNOUI****,* ***Dr. Khaled REZEG*** *and* ***Prof. Allaoua CHAOUI****, whose insightful remarks during the viva will certainly enrich this work.*
Last but not least, I wish to thank ***Prof. Med Chaouki Babahenini****, head of the Computer Science department.*

Contents

General Introduction **10**

I Security, Security Policies and Formal Methods (Coloured Petri Nets) **14**
I.1 Introduction . . . 15
I.2 Security Definition . . . 15
I.3 Security Policy . . . 15
I.4 History of access control policies . . . 16
I.5 Access Control policies . . . 18
I.5.1 Mandatory Access Control (MAC) . . . 18
I.5.2 Discretionary Access Control Policies (DAC) . . . 19
I.5.3 Role Based Access Control (RBAC) . . . 20
I.5.4 Attribute-Based Access Control (ABAC) . . . 24
I.6 Formal Analysing and Verification methods . . . 24
I.6.1 Hierarchical Timed Coloured Petri Nets . . . 25
I.7 Conclusion . . . 30

II First Contribution: Using Hierarchical Timed Coloured Petri Nets in the Formal Study of TRBAC Security Policies **31**
II.1 Introduction . . . 32
II.2 Related Work . . . 32
II.3 Modelling and Analysis of TRBAC policy using Hierarchical Timed-CPN 35
II.3.1 The Illustrative example: The Algerian justice information system policy . . . 37
II.3.2 The global TRBAC model . . . 40
II.3.3 Modelling of role-enabling and role-disabling events . . . 41
II.3.4 Modelling of role-“assignment/deassignment” events and role-“activation/deactivation” events . . . 44
II.3.5 Formalisation and verification of properties . . . 50
II.4 Conclusion . . . 57

III Second Contribution: Fine-grained Role-Attribute based Access Control model (FRABAC): A New Hybrid Access Control Model **59**
III.1 Introduction . . . 60
III.2 Related work . . . 60
III.3 A new hybrid access control model . . . 61
III.3.1 Requirements for a suitable access control model . . . 61
III.3.2 Principle of the proposed model . . . 62
III.3.3 Collaborative Cloud Services case in the proposed model . . . 63
III.3.4 The security policy under the proposed model . . . 65

III.4 Evaluation of the new proposed model: empirical comparison with existing models . . . 69
III.4.1 Metrics used in the empirical comparison approach . . . 70
III.4.2 The illustrative example . . . 70
III.4.3 RBAC configuration evaluation . . . 71
III.4.4 ABAC configuration evaluation . . . 74
III.4.5 AERBAC configuration evaluation . . . 76
III.4.6 Evaluation of the new proposed model . . . 77
III.5 Conclusion . . . 79

IV Third Contribution: Using Hierarchical Coloured Petri Nets in the Formal Study of FRABAC Security Policies 80
IV.1 Introduction . . . 81
IV.2 Specification of the request evaluation process . . . 81
IV.2.1 Required types for the HCPN model . . . 82
IV.2.2 Modelling the identification process . . . 83
IV.2.3 Modelling the evaluation process . . . 84
IV.3 Formal verification using CPN-tool of the HCPN models . . . 89
IV.3.1 Formalization and verification of properties on the request identification . . . 89
IV.3.2 Formalization and verification of properties on the request evaluation . . . 90
IV.4 Conclusion . . . 92

General Conclusion 93

List of Figures

I.1 A simple example of a CPN: before firing the transition 29
I.2 A simple example of a CPN: after firing the transition 30

II.1 An abstract view of the multi-domain tree of the Justice Information system in Algeria. 38
II.2 HTCPN abstract model for TRBAC 40
II.3 Hierarchical Timed-CPN model for enabling/disabling roles 42
II.4 Hierarchical Timed CPN model for triggers 44
II.5 HTCPN model for role-assignment event 48
II.6 HTCPN model for role-activation event 50
II.7 marking of P3 52
II.8 marking of P4 52
II.9 marking of P1 53
II.10 marking of P2 53
II.11 marking of P5 53
II.12 Simulation report step 63 54
II.13 Simulation report step 76 55
II.14 Simulation report step 122 55
II.15 Simulation report step 19 55
II.16 Simulation report step 119 55

III.1 Principle of the proposed model 63
III.2 Collaborative cloud services example. 64
III.3 Mechanism of access decision when the user requires access to a specific resource 69
III.4 Mechanism of the access decision for multiple resources sharing the same attributes. 69
III.5 Experimental Results in RBAC 72
III.6 Experimental Results in ABAC 75
III.7 Experimental Results in AERBAC 77
III.8 Experimental Results In The Proposed Model 79

IV.1 Hierarchical CPN model for the identification process 83
IV.2 Hierarchical CPN model for Get precondition step 84
IV.3 Hierarchical CPN model for Take_Pre phase 85
IV.4 Hierarchical CPN model for Sample_Format phase 86
IV.5 Hierarchical CPN model for Attributes_Eval sub-model 87
IV.6 Hierarchical CPN model for Error sub-model 88
IV.7 Hierarchical CPN model for Decision step 89

List of Tables

II.1 TRBAC security analysis related works. 36
II.2 Roles, tasks, and permissions 39
II.3 Mapping from users to roles 40
II.4 An overview table with all important places and their meaning and use. . 41

III.1 The input parameters values for the evaluation of RBAC 73
III.2 AERBAC configuration 76

INTRODUCTION

Context and aims

The notion of security is omnipresent in the daily concerns of individuals. It affects every aspect of life as the safety of individuals, national security, social security, aviation safety, road safety, food security, etc. Security is a state in which risks and conditions that can cause physical, psychological or material harm are controlled in a manner that preserves the health and well-being of individuals and the community. It is an indispensable resource in everyday life that enables the individual and the community to realize their aspirations.

Thereby, security means allowing things the system owner does want, while stopping things which he doesn't want from happening. However, in computer science, and in every information system, the data security (protection) and privacy are considered as one of the critical challenges. Currently, people are getting used to having access to whatever information that they need, at any time, from anywhere and by using any device on a wide range of computing devices. In addition to this, new technologies as cloud computing and internet of things (IoT) has expanded the range of applications. Moreover, with open/distributed systems and the growth of networks, computer security threats are becoming more widespread and increasingly complex. Thus, the system administrator has to use a robust security policy which is used for preventing unauthorized access, use, disclosure, disruption, modification, inspection, recording or destruction of information. Indeed, these facts are the cause which makes the administration of security in large-scale networks remains a major challenge.

Consequently, a variety of access control models have been developed to address different aspects of security problems. Because of its ability to reduce complexity, the Role Based Access Control (RBAC) [1] model is one of the predominant models for specifying security policies and access control. RBAC model has been unified and standardized under the ANSI/INCITS standard in [2]. This standardization has motivated most information technology providers to integrate RBAC into their product lines. This model is implemented in various cloud services like Microsoft Azure and OpenStack and it has been widely adopted by various information systems (such as Windows/Active Directory RBAC, HP-UX, AIX, Oracle).

However, in its basic version, RBAC does not model explicitly the different states of a role. The basic RBAC does not consider various events that are typical of an RBAC system. To overcome these limits, some extensions of RBAC have been proposed. One of the most used extension is TRBAC (Temporal RBAC) [3], proposed to deal with temporal aspects. TRBAC defines necessary temporal constraints which capture the dynamic behaviour of systems that use RBAC, and allow the analysis of these constraints;

First Problematic

Hence, RBAC is the most common security policy for access control. A security policy is a set of rules that define the behaviour of a secure system. The system using such a policy must satisfy this set of rules in all its possible states. A state where one of the rules is not satisfied is considered as an inconsistent state. An inconsistent state may appear either because the policy itself is contradictory (i.e., containing conflicts) or it is incomplete. When the RBAC is used to define the policy, the set of security rules are specified using the basic concepts, i.e., Users, Roles, Permissions and Sessions. On the other hand, the consistency of the policy can be specified in terms of constraints

which are classified into three categories, (i) cardinality constraints, (ii) Separation of duties (SoD) constraints, (iii) and Inheritance constraint. In other extensions of RBAC, new concepts are introduced and therefore other constraints appear. For example, in the case of TRBAC, we consider the time constraints on the activation/deactivation of roles. In a formal specification of the RBAC policy, one must specify all the constraints that the system must satisfy to guarantee its security; Then the formal verification consists in proving that these constraints are satisfied by the behaviour of the system in all its states (No inconsistency). Thus, formal modelling/analysis approach of TRBAC policies is needed to prove its consistency within a system.

The proposed solution to deal with the first problematic

As a formal tool, Petri Nets [4] are well suitable to describe discrete processes and to analyse the system concurrency and synchronisation. A Petri Net has a graphical representation and a rigorous semantics. These characteristics make Petri nets a good formalism to specify and to analyse the access control policies. Coloured Petri Nets (CPNs) [5] represent an extension of Petri Nets; CPNs are more suitable to describe complex and typed data. Timed CPNs (TCPNs) extends CPNs with a set of time stamps. Hierarchical CPNs and Hierarchical Timed CPNs (HTCPNs) are extensions for CPN and TCPN, where the structure of the model can be organized hierarchically.

The use of formal methods allows us to prove that the designed policy is consistent. For that reason, we present a formal modelling/analysis approach of TRBAC policies. This approach uses Hierarchical Timed Coloured Petri Nets (HTCPN) formalism to model the TRBAC policy, and the CPN-tool [6] to analyse the generated models. The timed aspect, in HTCPN, facilitates the consideration of temporal constraints, introduced with TRBAC. The hierarchical aspect of HTCPNs makes the model "manageable", despite the complexity of TRBAC policy specification. In the HTCPNs model, we define the events that can occur in the system and their preconditions and post-conditions. These preconditions and post-conditions specify the TRBAC constraints that should be satisfied. After the specification by using the Hierarchical Timed CPNs formalism, the CPN-tool will be used to analyse the policies and to verify specific properties

Second Problematic: new challenges with Cloud

Although RBAC and TRBAC are predominant models, they do not incorporate environment attributes (contextual information), thus they are not suitable for systems involving frequently changing attributes (as the case of Cloud environments). To resolve these disadvantages, Attribute-Based Access Control (ABAC) [7] [8] was proposed. ABAC introduces the concept of the attribute, so that, an ABAC system is composed of three sets of entities, (i) the set of users, (ii) the set of resources, (iii) and the environment. Each of these three entities has specific attributes. The permissions of users under ABAC depend on their attributes. Indeed, even the ABAC was proposed to facilitate the security management, proposed solutions by ABAC can be as complicated as that of RBAC in some cases and so that ABAC does not benefit from the role's characteristic. In ABAC, role names are still associated with users, but they are no more considered as collections of permissions. Both models RBAC and ABAC have their specific features and they can complement each other. The idea to merge RBAC and ABAC in one model has become an important research topic, in order to combine their advantages. However, the proposed

solutions for merging both models are still insufficient.

The proposed solution to deal with the second problematic

In this thesis, we propose a new hybrid, flexible, scalable and fine-grained model (FRABAC)[9, 10] that combines the advantages of both models: RBAC and ABAC. The new proposed model overcomes the shortcomings of both models RBAC and ABAC (i.e., combinatorial explosion in rules and roles) when the security policy becomes complicated. Besides avoiding the combinatorial explosion, the new proposed model provides a Role Permission Agreement (APA) to handle the inter organizational access decision in collaborative cloud services cases. To illustrate these advantages, an empirical method is applied to compare the new proposed model versus three existing models: RBAC, ABAC, and the hybrid model Attribute Enhanced RBAC (AERBAC) [11]. This empirical comparison is based on four metrics that are inspired from the limitations of RBAC and ABAC.

On the other hand, the consistency and the correctness of the FRABAC policy is analysed through a formal modelling/analysis approach [12]. This formal approach uses Hierarchical Coloured Petri Nets (HCPNs) to model the policy and the CPN-tools to analyse the generated models.

Thesis structure

This thesis organized as follows:

- Chapter 1 introduces basic definitions of security, access control policies, and the types of access control policies. In the same chapter, we present the informal/formal definition and some necessary concepts of the Hierarchical Timed CPNs. Moreover, we discuss the choice of HTCPNs as a formal method to prove access control policies consistency.
- Chapter 2 details the first problematic and our proposed solution.
- Chapter 3 details the second problematic and our proposed solution.
- Chapter 4 presents a formal approach using HCPNs to prove the consistency of the proposed model in the chapter 3.

The thesis is concluded with a general conclusion which summarises the work, discuss its contributions and prospects future works.

Chapter I

Security, Security Policies and Formal Methods (Coloured Petri Nets)

I.1 Introduction

In computer science, and in every information system, the data security (protection) and privacy are considered as one of the critical challenges and has become an important research topic. Access control is a security technique of a comprehensive computer security system that minimizes the information security risks. However, the complexity of access control policies makes the assurance of their consistency and correctness a challenging problem. Moreover, security analysis is performed during the development of an access control policy. The use of formal methods has proved their efficiency to ensure such analysis, in access control policies. As a formal method, Petri Nets are well suitable to prove the consistency and the correctness of access control policies.

Hence, what is a security policy? What are its objectives? What is an access control and how it responses with the security objectives? Why Petri Nets are well suitable to describe and to analyse an access control policies within a system? This chapter answers all of these questions in sections I.2, I.3, I.4, I.5 and I.6 respectively. Finally, the section I.7 gives a brief deduction from what this chapter presents.

I.2 Security Definition

The ITSEC [13] (Information Technology Security Evaluation Criteria) glossary defines security as: "the combination of confidentiality, the prevention of unauthorized disclosure of information, integrity, the prevention of the unauthorized amendment or deletion of information, and availability, the prevention of the unauthorized withholding of information". Hence, Confidentiality, integrity and availability (CIA triad) are considered the three most crucial components of security and privacy. Moreover, CIA triad are the categories of information security risks and they are defined as follows.

- **Confidentiality**: this category means that only authorized users to access information will access them (keep information private).
- **Integrity**: this category means that the information is not made modifiable in an unauthorized manner and is protecting from being impropriety altered by unauthorized users.
- **Availability**: when the information is needed, it should be available to use.

I.3 Security Policy

Every single system has security needs and objectives. Hence, the security policy within a system must correspond with the description of those needs. According to [14] the system security policy specifies the set of laws, rules and practices that regulate how sensitive information and other resources are managed, protected and distributed within a specific system. This last definition means that every single security policy within a system defines two elements which are:

- **Security Objectives**: i.e. CIA properties expected of the system.
- **Security Rules**: rules that are applied to activities which can modify the system security state, in order to ensure that the security objectives are respected.

A secure system requires the establishment of a secure access management process. This last requires the establishment of Identification, Authentication, and Authorization (access control) Policies. Where:

- **Identification**: in the information system, the process that allows the user to identify or introduce himself (enters his identity (username, Card_Number)), it named the Identification process.
- **Authentication**: in the information security system, if the user wants to log in then he must prove that he is whom he says he is. This process is called "Authentication process". Most systems use a static password as the first factor in the authentication process. Different types of authentication methods are used as a second factor (PIN-code; magnetic stripe cards, smart cards, certificates with a digital signature and biometric factors as voice, retina, fingerprints, etc.)
- **Authorization**:when a user proves his authenticity to the system (successful authentication) then the process of authorization occurs based on his identity. Authorization process constrains what users (person or programs executing on behalf of the persons) can do directly or what they are allowed to do (his privileges or permissions). Hence, authorization process controls the user permissions. This access control is based on the following three principals:

  1. Need to know principal – the user will be granted access to resources that are necessary to fulfil his tasks and responsibilities.
  2. Least privilege principal – to protect the objects from an undesired action, the user will be provided with the minimum privileges that are required to perform his job.
  3. Separation of Duty principal – is used to formulate multi-person control policies, requiring that two or more different people be responsible for the completion of a task or a set of related tasks [15]. This guarantees that no user has the ability to perform a fraudulent action and then cover up that action.

An authorization (access control) policy may include identification and authentication policies. The search of this thesis focuses on the access control policies, their history, their types and the mechanism of some access control policies. Hence, the next section introduces the history of access control policies.

I.4 History of access control policies

Computer security began to progress rapidly in the early of 1970s [16] when Lampson [17] proposed an abstract formal model of access control policy which is called Access Control Matrix (ACM). The policy defines who can access what. The matrix abstracts the state relevant to access control. The columns represent entities to which access is to be controlled (each column corresponds to a resource or object that needs to be protected within the system) and rows represent entities that access current objects (each row corresponds to a user or subject or group within the system). The entry for a particular subject-object pair determines the access right (read, write, delete) that the subject is permitted to exercise on the object. According to [18], the Access Control Matrix does not

define the granularity of protection mechanisms. Due to this last, ACM can be used as a model of the static access permissions, in any type of access control system. The contents of the access matrix reflect the current state of the privileges of protection in the system. Hence, the ACM must be changed if there is a new privilege has been granted to a certain subject or in case of revoke privilege that existed before. Therefore, the authors of [19] put a set of rules (commands) to update the access matrix. These commands allow rights to pass from one user to another user and allow users to create or delete an object. The work of [20] proposed a formal ACM model and specified the set of commands. However, the system cannot control the passage of rights. This means that after a chain of rights delegation, the system will not guarantee that a user will not receive unauthorized access. Indeed, the system will not guarantee the least privilege principle. For example, there are three users in a system (Ahmed, Sam and Amir) and one resource (file ah.txt). Ahmed who is the owner of file ah.txt decides to give the permission read-only to Sam. Although Amir cannot access this file, Sam can recopy the content of ah.txt in sam.txt and gives to Amir the privilege to read it. In this case, Amir can read the content of file sam.txt. Consequently, he can read the content of the file ah.txt. According to [20] and as a result of the previous scenario, the safety problem for protection systems is an undecidable problem.

The concept of the matrix is a static concept. Indeed, implementing an access matrix using table or two-dimensional array is not practical and in general, it leads to using a large size of an array. In fact, adding a new object is an adding of entries for every subject in the system. Indeed, adding a new subject is an adding of entries for every object in the system. Moreover, it will be difficult to take advantages of special groupings, such as an object that can access by all subjects. To deal with the issues, two common approaches were used for implementing the access matrix, which are:

- **Access Control List (ACL)**: list for each object. This list consists of access rights (permissions) that are assigned to each subject for that object. Each Object list corresponds with a column of the matrix without taken subjects that have no access rights to the object.
- **Capability lists (CL)**: list for each subject. This list consists of subject privileges over all the objects. Each subject list corresponds with the rows of the matrix without taken the objects which this subject has no rights to access them.

Even although ACM is the simplest abstraction mechanism to describe precisely a protection state within a system, it has several shortcomings and weaknesses as follows:

1. It is difficult to provide Need to know, Least Privilege and Separation of Duty principals without associating access rights with a subject's credentials when performing an operation.
2. Access rights in ACM cannot be related to content (contextual information).
3. It is difficult to determine all the subject's access rights in ACL based systems. In fact, this needs to check the ACL of every object within the system.
4. It is difficult to determine all the allowed users to access a specific object in Capability lists based systems. In fact, this needs to check the CL of every subject within the system.

Several access control policies or strategies are proposed. Indeed, several models are proposed in each policy. Hence, the next section introduces these strategies briefly.

I.5 Access Control policies

To specify how accesses are controlled and how access decisions are determined, different policies have emerged.

I.5.1 Mandatory Access Control (MAC)

Mandatory access control (MAC) is an access control strategy that its access criteria are defined by the system administrator and an individual subject cannot alter those access criteria. The original definition of MAC was introduced in the TCSEC [21] as "a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity". Hence, MAC works by assigning a classification label to every single object within the system. Moreover, the administrator assigns a similar classification to every single subject within the system; this classification is named clearance level. According to the user's clearance level, his access to objects is determined. MAC policies are appropriate for multilevel secure military applications. The following elements present some MAC policies.

I.5.1.1 Multi-level security policy

According to [22], there was in the military a need for multiple levels of security on a single computer. To describe this need [ware] discussed the policies in the Department of Defense (DoD). These last put objects or documents at the corresponding level of sensitivity. A level of sensitivity is determined from different levels of classification (Unclassified< Confidential< Secret< Top Secret), to represent the danger that can be constituted if the file's information is disclosed. The access to an object is granted if: (i) the user clearance (privilege reflects the trust given to him) is on the same level of the object classification; and (ii) this object is required to perform the task of this user (need to know principle). This policy is named a Multi-level security policy. To formalize access control rules of this policy Bell and LaPadula in 1973 proposed a state machine model [23].

I.5.1.2 Bell-LaPadula security model

According to [23], the system consists of:

1. A set S: represents the set of subjects.
2. A set O: represents the set of objects.
3. A set A: represents the set of permissions (reading and writing).
4. A set I: represents the set of classification (access classes).

At a given time, the state of a system is expressed by a set of three components which are:

1. The set b: represents the set of all current rights that a subject has it to access an object. A current access right represented by a triple (s, o, a) to mean that the subject s has the access a on the object o, in a system status.

2. The function $\lambda : S \cup O \rightarrow I$: the function is applied to subjects and objects in order to return their own classifications.

3. The access matrix M: is formed of *i* rows and *j* columns. Each cell M_{ij} contains all the access modes that the subject S_i can apply them on the object O_j.

The model aims to ensure the confidentiality and it is characterized by "no read up, no write down" phrase. It is based on the following three security properties that prevent any leakage of information:

1. The Simple Security Property: it is not permitted to subject at a certain security level to read an object at a higher security level (no read-up).

   $\forall\ s \in S, o \in O : (s, o, read) \in\ b\ \ \lambda\ (s)$ >= $\lambda\ (o)$

   This property protects the confidentiality of those more secure objects.

2. The $*$ "Star or Confinement" Property: it is not permitted to writing in an object at a security level from an untrusted subject at a higher sensitivity level of it (no write-down).

   $\forall\ s \in S, o \in O : (s, o, write) \in\ b\ \lambda\ (o)$ >= $\lambda\ (s)$.

   This property prevents the copies of a high-level object into a low-level object.

3. The Discretionary Security Property – the system specifies the permissions of access by using an access matrix. However, if according to this matrix the subject has a permission to access an object at a security level higher than his clearance level, then this subject cannot exercise this right.

   $\forall\ (S_i, O_i, a) \in b\ a \in M_{ij}$ (with $a \in A$).

The model only addresses confidentiality; neither integrity policy nor availability policy are provided.

I.5.2 Discretionary Access Control Policies (DAC)

In Discretionary Access Control policy, each object within the system has an owner, and each original object owner is the subject that causes its creation. Thus, an object's access permissions are determined by its owner. The original definition of DAC was introduced in the TCSEC [21] as "a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control)". Hence, the policy is based on the identity of the requestor and on access rules to stating what requestors are allowed to do. According to [24], DAC models are characterized often by one or more of the next properties.

1. The Data Owner can assign the object's ownership to another user.

2. The Data Owner can determine the access type of other users (read, write, copy, etc.).

3. After several tries, authorization failures to the same resource generate an alarm and/or restrict the user's access.

4. Unauthorized users should not be able to determine the object's characteristics (file size, file name, directory path, etc.).

As a drawback, the administrator in the DAC model cannot centrally manage the access permissions on files stored within his system. Moreover, the DAC policy has the same disadvantage and drawbacks of ACM.

I.5.3 Role Based Access Control (RBAC)

Role Based Access Control (RBAC) is one of the most used models in designing and implementation of security policies, in large networking systems. The RBAC [1] 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. Hence, in RBAC, permissions are associated with roles assigned to users as a part of an organization. Based on the user's role, his access to resources is decided. Thus, a role can be considered as a collection of users that have the same set of permissions. A role is considered as an active role if there is a subject using it currently. In RBAC, each session associated with a single user and it is a mapping of one user to possibly many roles. Moreover, a user may have multiple sessions open at the same time and each session may have a different combination of active roles. The following definition formalizes the above discussion and resumes the basic RBAC model ($RBAC_0$).

Definition: $RBAC_0$ model has the following components:

- Set of users U, set of roles R, set of permissions P, and set of sessions S.
- Permission-to-role assignment relation: $PA \subseteq P*R$ a many-to-many relation.
- User to role assignment relation: $UA \subseteq U * R$, a many-to-many relation.

Moreover, $RBAC_0$ model has the following function that restrict the mapping of entities :

- Function $assigned - users(r)$ = $R \rightarrow P(U)$ it is the mapping of role r onto a set of users where $u \in U \mid (u, r) \in UA$.
- Function $assigned - permissions(r)$ = $R \rightarrow P(P)$ it is the mapping of role r onto a set of permissions where $p \in P \mid (p, r) \in PA$.
- user: $S \longrightarrow U$, a function mapping each session S_i to the single user U_j (a user can be assigned to many sessions but a session can be only assigned to a user)
- roles: $S \rightarrow R$, a function mapping each session S_i to a set of roles.

$RBAC_0$ has Cardinality constraints and Separation of duty constraints as followings:

- User Static Cardinality constraint: At most, k roles can be assigned to the user U_i.
- Role Static Cardinality constraint: At most, k users can be assigned to the role R_i.

- User Dynamic Cardinality constraint: At most, k roles can be activated by the user U_i.
- Role Static Cardinality constraint: At most, k users can activate the the role R_i.
- Conflicting roles cannot be assigned to the same user.
- Conflicting users cannot be assigned to the same role.
- Conflicting roles cannot be activated in the same session.
- Conflicting users cannot activate the same role.

Thus, RBAC approach has two principal advantages. The first advantage is that users will access only to the resources that they require to achieve their tasks, under the suitable mode. The second advantage is to make easier the system administration (The administrator is not obliged to redefine permissions for each user separately. Thus, all changes are at the role level and are reflected immediately on the permissions of users [25]). As an example, in a system composed of 500 users as students, it will be practical to define a role student, define its permissions, and then to enclose the 500 users in this role. In the basic RBAC, the access decision can be complex [26] and not adequate [27] when the contextual attributes (information of a user, environment attributes, etc) are required to grant the access. In the above example, if the students are divided into 100 specialities then RBAC should define 100 student roles one per speciality. Moreover, the permissions are referring to individual objects, which leads to role-permission explosion problem in situations including a large number of objects. However, RBAC has several extensions which were proposed to deal with some contextual information. This thesis discusses the most common extension of RBAC which is TRBAC [3] that is proposed to handle temporal contexts.

I.5.3.1 Temporal RBAC (TRBAC)

The RBAC classical model doesn't consider temporal aspects which are so important in access control policies. Temporal RBAC (TRBAC) is proposed to deal with these temporal aspects. In the TRBAC, another class of constraints is considered:(i) temporal constraints on the events of roles activation/deactivation, (ii) periodic role enabling/disabling and (iii) temporal dependencies among such actions. Periodic-events and roles-triggers are the temporal constraints that specify the enabling and disabling role events.

• A periodic-event is expressed as a triple: $< I, P, pr : E >$, where: I is an interval, P: a periodic expression, pr a priority, and E an event (which can be: to *enable* a role or to *disable* a role). For example, the periodic event <[01/01/2016,∞], $Night\_time$, VH: $disable\ R1 >$ means that the role "Procurator (R1)" must be disabled with a very high (VH) priority at night time, from the date 01/01/2016 and forever.

In TRBAC policy, when two concurrent events arrive, the event with the highest priority (the greatest number) occurs. For example, if we have the two concurrent events: "*event 1= enable nurse on day*" with a priority equals to 1 and "event 2= disable nurse on day" with a priority equals to 2, then the second event occurs and the first one is ignored. The priority is used to deal with an emergency case, an exceptional case, or to make a decision in conflict cases.

• A role-trigger is expressed as a tuple:
$< E1, ..., En, C1, ..., C_m \rightarrow pr : E\ after\ \triangle t >$, where $E_1, ..., E_n$ are a set of events; $C_1, ..., C_m$ are a set of statuses; pr is a priority; $E :$ is an event; and $\triangle t$ is a duration of time. These triggers means that once the set of events $E_1, ..., E_n$ occur, and if the system contains the set of statuses $C_1, ..., C_m$, then the event E will occur with a priority pr, after a duration $\triangle t$. For example, the trigger: $< enable\ nurse\_on\_day\_duty \rightarrow H : disable$ nurse_on_training after 2 $days >$ 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".

TRBAC was, then, extended to General Temporal RBAC (or GTRBAC) [28]. GTRBAC defines more specific temporal constraints such as temporal constraints on user-role and role-permission assignments/de-assignments, role activation-time constraints, etc.

I.5.3.2 RBAC extensions

Besides TRBAC, OrBAC (Organization Based Access Control) model is another extension of RBAC which includes the temporal aspect. Orbac assigns at the level of each organization *org*: (i) user *u* to role *r*, (ii) action (i.e., operation) *op* to activity *act*, (iii) object *o* to view *v*. Hence, to give user *u* the permission to perform an operation *op* on object *o*, the system must check complexly all permissions one by one until finding permission *(org, r, act, v)* where: *r* is one of roles which are assigned to user *u*, *act* is one of the activities that are assigned to action *op*, and *v* is one of the views that are assigned to object *o*. Indeed, Orbac allows the historical restrictions (for example, role *r* can access only one time to view *v*) and context restrictions. However, the authors of [29] argue that OrBAC is a complex model (it gives more details about permissions, it handles historical and organization aspects) and it uses an incorrect multiplicity relationships (for example, in the *Consider(org, a, act)* relationship that assigns the action *a* to activity *act*, it uses *(0,n)* multiplicity at both side of action and activity. This last means that an activity can have no action and an action may not belong to any activity. The authors of [30] enrich OrBAC with integrity mechanisms and means of differentiation to preserve Critical Infrastructures Integrity (CIIs). By presentation of Integrity-OrBAC (I-OrBAC), the authors define a new proactive, multi-integrity level model that enables quantifying the integrity needs of each CII element, in term of credibility or criticality, to take optimal access control decisions.

Some other RBAC extensions dedicated to specific applications are also proposed. To use RBAC in ubiquitous collaboration systems, the authors of [31] proposed "XGSP-RBAC" (XML based General Session Protocol). In "XGSP-RBAC", authors deal with the protecting secured resources from unauthorized users, which become a complicated task in ubiquitous computing systems. The "H-RBA" (hierarchical RBAC) [32], is proposed to design security policies in SaaS (Software as a Service). In the SaaS approach, software is deployed as a hosted service and accessed over the Internet. Customers don't need to maintain the software code and data on their own servers. Another recent extension is Tie-RBAC [33], dedicated to implementing social networks security policies. In this model, a tie (or a link) is composed of a relation, a sender and a receiver. A tie involves the sender's assignation of the receiver to a role with permissions.

Recently, some works tried to extend RBAC to deal with cloud environments [34], [35], [36], [37], [38]. In [34], the authors propose a Contract-RBAC model extending RBAC model in cloud computing with continuous services for the user to access various

source services provided by different providers. Contract-RBAC model can provide continuous services with more secure flexible management to meet the application requirements including Intra-cross cloud service and Inter-cross cloud service. The authors of [36] aim to decide about the best authorization technique for deployment in Cloud. They present a systematic analysis of the existing authorization solutions in Cloud and evaluate their effectiveness against well-established industrial standards that conform to the unique access control requirements in the domain. On another hand and to construct flexible data access control for cloud storage service, the authors of [35] addressed how to construct an RBAC-compatible secure cloud storage service with a user-friendly and easy-to-manage attribute-based access control (ABAC) mechanism. The ABAC scheme enhances the expressiveness of access policies, decreases the computational overheads, and reduces the size of ciphertexts and private-keys for attribute-based encryption. The authors of [37] present a generalization of RBAC model called Temporal Defeasible Logic (TDL) which allows to specify temporal policies and to handle conflicts to provide a formalism for specifying authorization. Finally, the authors of [38] propose SAT-RBAC (Security and Availability Based Trust Relationship RBAC) model as a Novel Role-based Access Control Model in cloud environments. To assign a role to a user, the SAT-RBAC computes the trust degree of their relationship based on three elements: the host's security situation, the host's network availability and how the server is protected. The value of the degree computed can be in three possible zones (unbelievable, probable believable and believable). The role is authorized for the user if the value is in the third zone, unauthorized if it is in the first zone. If the value is in the probable believable zone then the model uses a Bayesian Probability Distribution (BPD) to decide if the role is authorized or not.

I.5.3.3 Application arias of RBAC

Initially, designed as a model to mainstream commerce systems, RBAC has found applications in several areas: health care [39], work-flow systems [40], education [41], distributed applications security [42], web services and their architecture [43], [44], social networks [45], wireless networks [46], [47], cloud computing [38], etc. In [39], authors used RBAC model 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. In such a system, RBAC access policies ensure high information security and stringent access control for patient's health data. This will protect the patient's privacy and prevent a harmful or illegal use of data. In [40], RBAC model was applied to secure Web-based workflow systems. In this work, RBAC facilitates the access control management (without hindering the process). In [41], RBAC has been applied in an office automation system, used in several colleges. In this last work, the authors proved that web applications based on the RBAC model have excellent safety and stability. Authors, in [42], analyse the level of support for RBAC components and functional specification in Microsoft COM+ (Component Object Model) [48] middle-ware. They concluded that COM+ architecture prevents the support for session management and role activation, as specified in ANSI RBAC. In [49], authentication and authorization concepts, defined in Parametrized Role Based Access Control (PRBAC), are used to design a new secure architecture for the Java RMI (Remote Method Invocation). This new architecture can be used to build secure distributed applications. In [43], authors present a case study where RBAC is used for designing Security Architectures for "web services" and the authors of [44] tackle with security in SOA (Service-Oriented Architecture). In SOA, protection of the client's data,

from unauthorized access by services is a hard task because the services interact and share the client's data dynamically to fulfil the client's request. The authors of [44] use Active Bundle (AB) which contains the client's data and the rules restricting service access to this data (policy). Instead of sending all client's data in the query, one sends just the AB, and when the services interact to fulfil the request, each service interacts with the AB and gets access to the appropriate information based on its type (role) and the restriction of the client policy. In SOA policies, services are considered as users that are assigned to roles in order to access client's data. The authors of [45] apply RBAC to social networks. In facebook-style social network, relationship based access control models are used to control the user's access. These models exploit the relationship between the user and the owner of the resource. This kind of access control suffers from several limits (a user cannot tag users which live in a particular place, or work in a particular company, etc). To avoid these limits, the new access control scheme proposed in [45] exploits the attributes, activities and the common interests of users to group them and to control their access. In [46], the RBAC model was applied in wireless networks to authenticate RFID (Radio frequency identification) readers embedded on mobile devices to read tag embedded on objects. A proposed protocol uses RBAC server which recognizes the reader's identity in role classes and according to the role class of the reader the database defines its security certificate and permissions.

I.5.4 Attribute-Based Access Control (ABAC)

To resolve RBAC's disadvantages, Attribute-Based Access Control (ABAC)[7] [8] was proposed. The ABAC introduces the concept of the attribute, so that an ABAC system is composed of three sets of entities: the set of users, the set of resources and the environment. Each of these three entities has specific attributes. An attribute consists of a pair *(key, value)* (in the example of students, a user will have the attribute $<$*role value, speciality value*$>$ like the couple $<$*student, AI*$>$). The permissions of users under ABAC depend on their attributes. For example, the administrator can define a rule as *(true, role=student and Speciality=AI, read, download , Type=Courses and Speciality=AI)* which means that if the user is a *student* in *AI speciality* and he requests *to read* or *to download AI courses* then *he has the permission*. Even the ABAC was proposed to facilitate the management of security by introducing the concept of the attribute, the proposed solution by ABAC can be as complicated as that of RBAC in some cases [11] and so that does not benefit from the role's characteristic. According to [50], in ABAC the Role names are still associated with users, but they are no more considered as collections of permissions.

I.6 Formal Analysing and Verification methods

Formal methods allow us to prove or disprove the correctness of a system with respect to a certain formal specification or property. Thus, they are naturally based on formal and mathematical roots. The NASA Langley Formal Methods Group [51] given the definition of formal methods as mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems. The phrase "mathematically rigorous" means that the specifications used in formal methods are well-formed statements in a mathematical logic and that the formal verifications are rigorous deductions in that logic (i.e. each step follows from a rule of inference and hence can be checked by a

mechanical process.). Hence, there is a need for a formal specification and verification in order to analyse and verify an access control policy. In a formal specification of an access control policy, one must specify the policy 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. As a formal method, Petri Nets are well suitable to describe discrete processes and to analyse the system concurrency and synchronism. A Petri Net has a graphical representation and a rigorous semantics. These characteristics make Petri nets a good formalism to specify and analyse the access control policies. In the literature, one can find some works which have applied Petri net and its extension Coloured Petri net to model/analyse access control policies. In [52, 53, 54, 55, 56], Petri Nets are used to specify access control policies in work-flow systems. [57] uses coverability graph to analyse policies. In [58], security attributes in mandatory access control are analysed using reach-ability in Petri Nets. In [59], a formal security model based on Coloured Petri Net (CPN) is proposed and used to show the analysis and construction methods to information flow security. To identify the Petri net and its extension hierarchical colored Petri net, the following subsections give an informal and a formal definition of them.

I.6.1 Hierarchical Timed Coloured Petri Nets

Informally, a Petri Nets (PN) [4] 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 (modelling 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.

Coloured Petri Nets (CPN) [5] is an extension of Petri Nets. In CPN, each place has a type (a colour). So the tokens can be more complex and typed data. The arcs are labelled 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.

Timed CPN extends CPN with a set of stamps of time. These stamps can be associated with tokens or to transitions. A stamp s associated with 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 with a transition, all the stamps associated with 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 by the definition of CPN [5]. Our definition of Timed CPN updated the definition of CPN with the concept of stamps, as implemented in the CPN-tool [6].

Hierarchical CPN and Hierarchical TCPN are extensions for CPN and TCPN, where the structure of the model can be organized hierarchically. A hierarchical representation of the specification is a modular representation of the model. In this modular representation, the model is composed of a set of sub-models composed together to build one big model. Therefore, the specification is represented as an abstract principal model that can be refined towards an elaborated model, at any times. Hierarchical Timed CPN uses *hyper transitions* in the abstract model to hold in the sub-models. The refinement of the abstract model passes through the unfolding of these hyper-transitions.

In this section, we present the formal definition of a Hierarchical Timed CPN, and we show the dynamic behaviour of this formalism.

I.6.1.1 Formal Definition

Firstly, we present some necessary concepts (timed sets and timed multi-sets) which will be used in the formal definition of the Hierarchical Timed CPN. Let $\mathbb{N}$ denote the set of non-negative integers. These definitions are extensions for sets and multi-sets concepts, used in CPN [5].

Definition 1. A Time set Γ is a set of non-negative integers. $\Gamma = \{\tau \in \mathbb{N}\}$.

Definition 2. A time multi-set τm, over a non-empty set X, is a function $\tau m \in [X \rightarrow \mathbb{N} \times \Gamma]$, for each $x \in X$, $\tau m(x) = (O(x), S(x))$,where $O(x) \in [X \rightarrow \mathbb{N}]$, is the number of occurrences of x and $S(x) \in [X \rightarrow T]$ is a stamp (from a Time set, in our case a set of positive integers). $\tau m(x)$ is represented as a formal sum: $\Sigma_{x \in X}(O(x)'x@ + S(x))$. In this expression, the symbol @ distinguishes between the O values and the S values.

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

For example, if we take the set $X = \{2, 5\}$. A time multi-set on this set can be $(2'2@+ 2)+ +(1'5@ + 4)$. This time multi-set represents the set that contains two occurrences of 2 stamped with the stamp 2 (each occurrence has the stamp=2) and one occurrence of 5 (with a stamp=4). $\{1, 2\}_{MS}$ is the set of all timed multi-sets of the set $\{1, 2\}$, which is an infinite set.

In the following definitions, some notations 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 $TCPN = (\Sigma, P, T, A, C, G, Exp, I, \tau)$, where:

1. Σ is a finite set of non-empty **types**, also called colour sets,
2. P is a finite set of **places**,
3. T is a finite set of **transitions**,
4. A is a finite set of **arcs** such that: $A \subseteq (P \times T) \cup (T \times P)$ and $P \cap T = P \cap A = T \cap A = \phi$. If $a = (p, t)$ is an arc in A, we say that p is an input place for t, we denote this: $p \in {}^{\circ}t$. If $a = (t, p)$ is an arc in A, we say that p is an output place for t, we denote this: $p \in t^{\circ}$.
5. C is a **colour** function (it defines the type of each place in the TCPN). It is defined from P into Σ.
6. G is a **guard** function. It is defined from T into expressions such that:
   if $tr \in T$ then $Type(G(tr)) =$ Boolean
   and $Type(Var(G(tr)) \subseteq \Sigma$
7. Exp is an arc **expression** function. It is defined from A into expressions such that:
   $a \in A : Type(Exp(a)) = C(p)_{\tau MS}$ and
   $Type(Var(Exp(a))) \in \Sigma$, where p is the place component in a.

8. I is an initialisation function (or an initial marking of the set of places). It is defined from P into closed expressions such that: for each $p \in P : Type(I(p)) = C(p)_{\tau MS}$.

9. τ: is a **time function** that associates with each transition a stamp. $\tau : T \rightarrow \Gamma$, ($\Gamma$ is a time set).

Definition 4. A Hierarchical TCPN is a finite set of nets $H = \{N_1, N_2, N_3, \dots\}$. Each net N in H is a tuple $N = (\Sigma, P, T, A, C, G, Exp, I, \tau, h)$, where:

1. Σ, A, C, Exp, I: are defined as in TCPN,

2. $P = OP \cup IP$, where: OP is a set of ordinary places as defined in TCPN, and IP is a set of interface places. An interface place is a place that is shared between more than one net. An interface place is used in communication between nets in H.

3. $T = OT \cup HT$, where: OT is a set of ordinary transitions as defined in TCPN, and HT: a set of hyper-transitions (can be empty).

4. G: is a **guard** function. It is defined from T into expressions such that:
if $tr \in T$ then $Type(G(tr)) \in$ Boolean
and $Type(Var(G(tr)) \subseteq \Sigma$

5. τ: is a **time function** that associates with each ordinary transition a stamp. $\tau : OT \rightarrow \Gamma$. ($\Gamma$ is a time set).

6. h: is a function that maps each hyper transition to a net. $h : HT \rightarrow H$. We require that if ht is a hyper transition in N, so $h(ht)$ must not be N and must not lead to N indirectly. This means recursion is not allowed in the model. The input-places of ht ($^\circ ht$) and the out-places of ht (ht°) are places in the net $h(ht)$.

I.6.1.2 Dynamic behaviour and semantics of Hierarchical Timed CPN

The dynamic behaviour of the net is obtained when the transitions are fired. A transition can be fired if it is enabled. To be enabled, a transition requires some preconditions. 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 a transition will update the marking of its input and output places. The new marking depends on the labels of the input-arcs 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 function $Var(tr)$ to extract the set of variables used in the guards associated with the transition tr, or used in the expressions labelling input-arcs or output-arcs of tr.

Definition 5 (**binding**). A binding of a transition tr is a function b defined on $Var(tr)$, such that:

(i) For each $v \in Var(tr)$: $b(v) \in Type(v)$,

(ii) The binding of tr satisfies the guard function of tr. Formally, this is written: $G(tr)[b] = true$, or $G(tr)[b]$.

Definition 6 (**timed-binding**). A timed-binding of a transition tr is a couple $< b, t >$, where b is a binding defined on $Var(tr)$, t is a time, and at the time t, we have :
(i) For each $v \in Var(tr) : b(v) \in Type(v)$,
(ii) $G(tr)[b]$.

Definition 7 (**binding element**). A binding element be is a pair (tr, b), such that tr is a transition and b is a binding of tr. The set of all binding elements of a transition tr is denoted by $BE(tr)$. The set BE denotes the set of all binding elements for a CPN.

Definition 8 (**timed binding element**). A timed binding element tbe is a pair $(tr, < b, t >)$, such that tr is a transition and $< b, t >$ is a timed-binding of tr. The set of all timed binding elements of a transition tr is denoted by $tBE(tr)$. The set tBE denotes the set of all timed binding elements for a Timed CPN.

Definition 9 (**timed-marking**). Let L be the set of tokens in the place p at the time t. Then 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 to t: $M_t(p) = \Sigma_{x \in L \text{ and } S(x) \leq t}(O(x)'x@ + S(x))$.
The initial timed-marking, denoted M_0, is the timed-marking of the net at the time 0.

Definition 10 (**time-enabled**). A transition tr is time-enabled at time t in a marking M if and only if there is a timed-binding $[b, t]$ which satisfies the property: for each $p \in^{\circ} tr : Exp(p, tr)[b] \leq M_t(p)$. Where $^{\circ}tr$ is the set of input-places of the transition tr. We write that $(tr, [b, t])$ is time-enabled at the time t.

Definition 11 (**stamped-expression**). Let X be a timed multi-set, exp be an expression defined over X, and t be a stamp, the stamped-expression $exp@ + t$ denotes the expression exp in which the stamps of all its operands are incremented with t. for example, if $exp = (2'1@ + 2)++(1'2@ + 4)$, then $exp@ + 2 = (2'1@ + 4)++(1'2@ + 6)$.

Definition 12 (**firing precondition and reachable marking**). When a transition tr is time-enabled at time t (with a time-binding $[b, t]$), in a timed-marking M_t, it can be fired. Firing tr is an event that can take a duration Δt. Firing tr changes the marking M_t to another marking $M_{t+\triangle t}$, such that:

$$\text{for each } p \in P, M_{t+\Delta t}(p) = (M_t(p)^{\smile} Exp(p, tr)[b]) + (Exp(tr, p)[b]@ + (tr)).$$

We say that $M_{t+\Delta t}$ is directly reachable from M_t, and this is written: $M_t[tr > M_{t+\Delta t}$.

Definition 13 (**firing a hyper transition**). Preconditions to fire a hyper transition ht defined in a net N (in a HTCPN H), are the same as well as for an ordinary transition. When ht is fired, ht is unfolded to the net $h(ht)$. Firing ht changes the marking of N as well as the marking of $h(ht)$.

I.6.1.3 Reachability graph of a Timed CPN

The analysis of Timed CPN models can be done through computing reachability graphs (called also occurrence graphs) [5]. A reachability graph is a graph with a node for each timed reachable marking and an arc for each occurring timed binding element. Let

M_1 and M_2 be two reachable timed markings and tb is a timed binding element enabled in M_1. If the occurring of tb transforms M_1 into M_2, then we denote this by: $M_1[tb > M_2$. The reachability graph is a directed graph considered as a couple (V, A), such that:

- V: a set of reachable timed markings. $V = \{M_1, M_2, \dots\}$.
- A: a set of arcs linking nodes of V. $A \subseteq V \times tBE \times V$.

Given a Timed CPN, one can use the algorithm proposed in [5], to construct the reachability graph. When using CPN-tool, this construction is semi-automatic.

I.6.1.4 An example of a Timed CPN

In order to clarify the previous formal definitions and aspects, related to timed CPN, we consider a simple example. The example illustrates the concepts of marking, timed-marking, timed-binding, timed-enabled and stamped-expression. The example is realised using the CPN-tool. Let us consider a simple CPN (see Figure I.1) composed of three places P1, P2, P3 and one transition T. Initial markings of places are shown next to each place in green colour. Place P1 has the type: *Timed_INT* (i.e., timed integer), hence it contains a timed-marking $1'1@2$. This timed-marking means that place P1 contains an integer token equals which will be ready at a time equals to 2 time-units. P2 has the type *INT* (i.e., integer) and its initial marking is $2'2{+}{+}1'4$, which means that P2 contains three integers (two occurrences of value 2 and a unique occurrence of value 4). $2'2{+}{+}1'4$ is an element in the integer multi-set. Transition T has two input places P1 and P2, one output place P3 and a guard $[x > 2]$. Arcs (P1, T) and (P2, T) are labelled with expressions y, x which belong to the types *Timed_INT* and *INT*, respectively. P3 has the type *INT timed* and its initial timed-marking is empty. Arc (T, P3) is labelled with a stamped-expression $x@{+}2$, which means that firing transition T will add a stamped-token which value is x and which stamp is 2. As shown in Figure I.1, transition T has a green colour, which means that T is time-enabled at a time equals to 2 with timed-binding $[y = 1, x = 2]$. After firing T, the marking of the CPN is updated as shown in Figure I.2. The new marking of P3 is $1'4@4$, which means a token with value 4 and a stamp equals to 4. This new stamp is the sum of the enabling time which was 2 and the stamp of the stamped-expression $x@ + 2$.

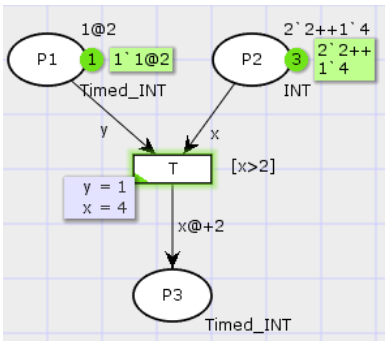


Figure I.1: A simple example of a CPN: before firing the transition

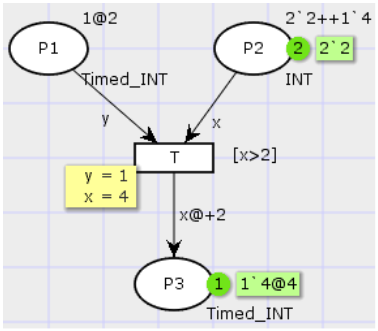


Figure I.2: A simple example of a CPN: after firing the transition

I.7 Conclusion

In large networks management, the complexity of security administration remains an important challenge. Many models were proposed to deal with this complexity. Due to their features, Role Based Access Control and Attribute Access Control are ones of the predominant models for advanced access control. However, they suffer from several problems when dealing with complicated security policies in complicated systems. These previous models have several drawbacks such as: the explosion in the number of roles and rules, problems with context-awareness, problems with the visualisation of policies update, etc. Thus, handling These drawbacks make these models enable to provide scalability, flexibility, and fine granularity in large-scale networks.

On the other hand, the complexity of access control policies makes the assurance of their consistency and correctness a challenging problem. Moreover, security analysis is performed during the development of an access control policy. The use of formal methods has proved their efficiency to ensure such analysis, in access control policies. As a formal method, Petri Nets are well suitable to describe discrete processes and to analyse the system concurrency and synchronism. The analysis of Petri net models allows the designer to prove the consistency of its policy. One can use the reachability graph of Petri net model to analyse and to verify the policy. Coloured Petri Nets (CPNs) represent an extension of Petri nets with more expressive power. The modelling of access control policy using CPNs is more practical than using classical Petri Nets. The next chapter presents a Formal Study of TRBAC Security Policies with the Using of the Hierarchical Timed Coloured Petri Nets.

Chapter II

First Contribution: Using Hierarchical Timed Coloured Petri Nets in the Formal Study of TRBAC Security Policies

II.1 Introduction

Role Based Access Control (RBAC) is one of the most used models in designing and implementation of security policies, in large networking systems. The RBAC classical model doesn't consider temporal aspects which are so important in such policies. Temporal RBAC (TRBAC) is proposed to deal with these temporal aspects. Despite the elegance of these models, designing a security policy remains a challenge. One is obliged to prove the consistency and the correctness of the policy. The use of formal methods allows us to prove that the designed policy is consistent. In this chapter, we present a formal modelling/analysis approach of TRBAC policies. This approach uses Hierarchical Timed Coloured Petri Nets (HTCPN) formalism to model the TRBAC policy, and the CPN-tool to analyse the generated models. The timed aspect, in HTCPN, facilitates the consideration of temporal constraints introduced in TRBAC. The hierarchical aspect of HTCPN makes the model "manageable", despite the complexity of TRBAC policy specification. In the HTCPN model, we define the events that can occur in the system and their preconditions and post-conditions. These preconditions and post-conditions specify the TRBAC constraints that should be satisfied. After the specification by using the Hierarchical Timed CPN formalism, the CPN-tool [6] will be used to ensure an analysis of the policies and to verify some inherent properties. In this chapter, we use the Hierarchical Timed Petri Nets (HTCPNs) in a down-top approach (from sub-models to an abstract model) to model TRBAC. Besides the use of HTCPNs, the chapter presents the analysis phase that focuses on the verification process of several temporal properties. The modelling and verification processes are all showed on a realistic security policy.

The rest of the chapter is organized as follows: section two presents some related works. Section three details the modelling/analysis process of TRBAC into HTCPN and illustrates that on a real policy. Finally, section four concludes this chapter.

II.2 Related Work

A policy is a set of rules that define the behaviour 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, one 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 [60], the authors defined the ConPN (Conflict Petri Nets) formalism based on CPN (Coloured 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) car-

dinality 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 [61], authors use CPN without guards to conflict detection in an inter-operation of RBAC policies. Despite role inheritance conflict (studied in [60]), authors of [61] 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 [62], the authors use CPN with guards and inhibitor arcs. In this last 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 [41, 40]. The originality of their work is the consideration of two temporal constraints (defined in the GTRBAC): dependency and precedence constraints between the activation, and the assignment of roles. In [63], the authors present a work similar to that presented in [62], where they use CPN formalism to model the SA-RBAC (self authenticated RBAC). In SA-RBAC model, permissions are of two kinds (general or sensitive); and to access sensitive permission, users must have a self authentication. In [64], the authors have presented a specification of RBAC using CPN, and they have used the CPN-tool to analyse the specification. In [64], The authors have not treated the temporal constraints. The authors of [65] use CPN to verify some properties in Spatio-Temporal Role-Based Access Control Model. However, in this last work, the CPN models are separated and do not represent the whole system, the authors do not provide the inheritance of permission in assigned of roles, they do not handle the concurrent query of the enabling and disabling event, and finally, the model does not consider periodic events and trigger constraints.

Besides the use of Petri Nets, other formalisms were used in RBAC specifications. In [66], authors present an automatic tool to analyse ARBAC (An Administrative RBAC [67]) policies. An ARBAC policy specifies how each administrator may change the RBAC policy. The analysis deals with six properties of RBAC and ARBAC policies, including (1) **reachability**: e.g., can user u be assigned to role r (called a "goal")? (2) **availability**: e.g., is user u always a member of role r? (3) **role-role containment**: is every member of role r1 also a member of role r2? (4) **weakest precondition**: what are the minimal sets of initial roles that enable a user to get added to roles in the goal? (5) **dead roles**: what roles cannot be assigned to any user? and (6) **information flow**: can information flow from object o1 to object o2?. In their work, authors do not precise the formalism used to specify the policies. The authors of [68], [69], and [70] apply first order logic to formalise RBAC policies.In [68], authors describe logically RBAC models used in web services design. Reachability problems in this logical description are resolved using an automatic tool. The authors use a specific RBAC version:"RBAC4BPEL" [68])dedicated to web services. This work concentrates on the decidability of properties and the time necessary in the verification step. The authors of [69] present a methodology for performing a security analysis of temporal RBAC (TRBAC) using the relations defined in a recently proposed administrative model named AMTRAC (Administrative model for temporal Role-based Access control). In [69], the authors use Alloy (a first order logic based language) to formalise and verify the administrative relations in AMTRAC. A similar work to [69] was proposed in [71]. However, authors do not precise the formalism used to specify the policies and do not present the whole model. The author of [70] uses the first order logic to define the set of roles, permission, users and express the two relations "has-role" and "has-permission". Hence, they illustrate the formalisation on a fictive example and analyse the constraints of the example by using Prover9. However,

the authors have not considered the dynamic SoD constraints, the cardinality constraints and hierarchy constraints. The authors of [72] use a symbolic model checking technique to analyse Administrative TRBAC systems. This last technique uses Bernays-Shonfinkel-Ramsey (BSR) transition system and Satisfiability Modulo Theories (SMT) solvers. BSR is used to represent ATRBAC system as a transition system and SMT solvers are used to computing the set of reachable states. BSR is a fragment of first-order logic formulas and SMT solvers are extensions of Boolean solvers that can determine whether a formula is satisfiable (in decidable fragments of first-order logic).

Besides first order logic, other classes of logics are used in [73], [74], [75] to specify formally RBAC policies. The authors of [73] use modal logic to specify RBAC policies. They translate the RBAC concepts into a set of logical formula and introduce logical rules for determinate whether the query of the user will grant or deny, these rules respect hierarchy constraints. However, the authors do not consider the SoD constraints and static/dynamic cardinality constraints. On another hand, the authors do not distinguish the authorized role from the assigned role. In [74], the authors formally specify and implement a model for RBAC called Smatch (Secure Management of suiTCH) in which authorized users can join, leave, reopen and reuse dynamic sessions. Smatch is specified using situation calculus which deals with contextual access control. They discuss decidability of three properties (decision: to grant or deny of access, prediction: if users will have some privilege in the future, invariants: "SoD, DoD"), using automated induction-based theorem prover. In [75], the authors propose an extension of Descriptive Logic (DL) called Dynamic Descriptive Logic (DDL) to formalize the RBAC in Web services. In this proposed model, services are considered as actions (or operations) which change the states of resources. However, the use of DLL does not deal with all aspects tackled in our current approach. The use of DLL was focused only on the assignment actions. Thus, activation of roles, the relation between roles, and temporal aspects are not treated or specified by DLL. The analysis of RBAC properties is not developed and the authors are limited only to specify the proposed RBAC with the DDL.

Algebra is exploited in [76] to specify RBAC. The authors describe a formal specification, using an algebraic language. They specify users, roles, operations, objects, and sessions. The hierarchical relation between roles is specified as a partial order relation. Based on this language, they specify SOD (separation of duties) constraints. In another work [77], the authors use UML-OCL (UML Object Constraint Language) as a framework to model the policies and the constraints. UML class diagrams are used to model relations between: sessions, users, roles and permissions. The OCL formal language is used to specify some authorisations such as SoD constraints, context constraints depending on sessions, constraints over permissions, and cardinality constraints.

Finally, some works like [78] and [79] exploit model-checking and timed automata to analyse RBAC policies. In [78], the authors propose an approach based on Timed automata (TA) to perform security analysis by analysing both safety and liveness properties in GTRBAC model. Analysis of the RBAC model involves mapping it into a state transition system. To simplify the obtained model, different reduction rules are proposed depending upon the constraints supported by the system. The authors propose a framework in which a GTRBAC set of policies are described as a Timed Automaton and properties as CTL properties, then a model checker is used to verify that the set of policies satisfy or not the required properties. A similar work to[78] was proposed in [80] to analyze TRBAC model. However, authors considered only activation and deactivation events with only one temporal constraint. Although the importance of this last approach, the num-

ber of states in the model which describes the role behaviour grows with the number of permissions (for each permission, three states are added which describe the access to that permission and its denial). In [79], the authors defined and provided the formal definition of a model checking technique that can be used as a management service/tool for the verification of multi-domain cloud policies. This model is based on NIST's (National Institute of Standards and Technology) generic model checking technique and has been enriched with RBAC reasoning.

Although some of the previous works tried to model the temporal constraints and so the GTRBAC model, we have observed that temporal aspects and temporal constraints defined in TRBAC were not well studied (*periodic events that enable or disable roles, triggers that can also change the status of roles*). The originality of the present work lies in: (i) The use of CPN-tool [6] to analyse the policy, (ii) the specification of more constraints that are not all addressed in previous works (as temporal constraints on enabling and disabling roles), and (iii) the use of the Hierarchical Petri nets formalism makes in a down-top modelling approach. The choice of Hierarchical Timed Coloured Petri Nets (HTCPNs) as a formalism to model TRBAC is motivated by several reasons as: (i) The graphical notation of HTCPNs makes the model more readable and understandable; (ii) The hierarchical aspect of HTCPNs makes the modelling process easier and overcomes the complexity of the TRBAC policy. Indeed, the TRBAC model is composed of set of sub-models ("enable/disable" events, "assigned/de-assigned" roles constraints, "activated/deactivated" roles, and "trigger" event); these sub-models are combined with each other to construct the complete model; (iii) and Finally, thanks to CPN-tools which allow the designer to create (edit), simulate, analyse, and validate the TRBAC model.

Table 1 summarizes works that have dealt with the formal analysis of temporal RBAC or those works which have considered a temporal constraint in their analysis. Indeed, most of these works have dealt with Administrative TRBAC. These works specify how members of each administrative role can change the TRBAC policy. We have defined a set of specific comparison criteria (i.e., verification tool, verified properties, security model, specification language, application, and missed or ignored events which are not specified or analysed). This table aims to illustrate the differences between our proposed work and the other related work.

II.3 Modelling and Analysis of TRBAC policy using Hierarchical Timed-CPN

Petri nets are considered as discrete event models. This nature of Petri nets guides the modeller when using Petri nets. So that, the first step in a modelling process requires the investigation of the set of events occurring in a system. When they occur, these events change the state of the system. In our case (an access control policy using TRBAC model), six major events are identified. These events modify the roles statuses defined in the system. The events are: **Enabling** of a role, **Disabling** 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 (constraints in the TRBAC model) to be satisfied. When these events occur, some post-conditions will be satisfied (which are, also, a set of constraints in TRBAC model). In the Petri Net model, these events are modelled by *transitions*. The pre-conditions are modelled by some *input-places*, some expressions on *input-arcs*, and some-associated *guards*. The post-conditions are modelled

Work	Verification tool	Verified Properties	Security model	Specification formalism	Application	Ignored events and Ignored constraints
[72]	ASASP-TIME	Can a user u be assigned to a role r at time t?	ATRBAC	fragment of first-order logic formulas	The example presents as an initial case: an empty TRBAC policy, ATRBAC policy and the goal that will be checked. ATRBAC policy is generated randomly by using a fixed number of roles and rules and periodic-times.	Inheritance of permission. Concurrent query of enable and disable event. Temporal role hierarchy. Trigger constraints.
[65]	CPN Tool	Can a user u get all his authorized permissions? Dynamic and static SOD.	Spatio-Temporal RBAC	CPN	Real world application.	Inheritance of permission. Concurrent query of enable and disable event. Periodic events. Trigger constraints.
[71]	language C	Can a user u be assigned to a role r? Can a permission p be assigned to a role r? Can a role r be the senior of a role r	ATRBAC		The same example of [72] plus a fixed number of cycles.	Inheritance of permission. Concurrent query of enable and disable event. Trigger constraints. Activation/deactivation events.
[69]	Java (7.0.1e17). Alloy Analyzer 4.2	Is there a dead state or a dead role? Can permission p be assigned to user u at time t? Can role r_1 be the senior of role r_2	ATRBAC	First order logic	Sample example composed by 4 users, roles, permissions and periodic-times.	Concurrent query of enable and disable event. Deassign of permission or user events (Alloy does not support removal of facts during analysis). Activation/deactivation role events.
[80]	Uppaal	If role r_1 is active then r_2 is already active? Can all users assigned to role r activate it at time t? A user u will never activate the two roles r_1 and r_2 at the same time?. The system is deadlock free?	ATRBAC	Timed Automata	Sample example of 2 roles an unknown number of users.	All the events and constraints except activation and deactivation of roles with non periodic time.
[62]	Not mentioned	Dynamic separation of duties.	RBAC	CPN	Sample example.	Trigger constraints. Concurrent query of enable and disable events. Periodic and time constraints.
This work	CPN tool	All possible properties to verify a TRBAC policy.	TRBAC	CPN	a Real case study.	

Table II.1: TRBAC security analysis related works.

by some *output-places* and some expressions on *output-arcs*. In the following paragraphs, we will show by details the modelling/analysis process. All the CPN models are accessible in the link [1].

In order to demonstrate the feasibility of the approach, we will propose the formalisation and analysis of a realistic policy: the "Algerian justice information system policy" (to be presented in subsection II.3.1). This policy will be used to build the concrete formal model and to analyse several properties of the TRBAC policy. The modelling of the **role-enabling** events is presented in subsection II.3.3. The modelling of the **role-assignment** and **role-activation** events are presented in Subsection II.3.4. The same ideas are used to model **Deactivation**, and **De-assignment** events. These two last models are not presented in the thesis, but they can be found in the whole model accessible in [1]. The subsection II.3.2 presents the abstract model for a TRBAC model, based on the sub-models realised in II.3.3 and II.3.4. Finally, the subsection II.3.5 presents the formulation and verification of several properties on the proposed models. The verification is illustrated using some concrete examples. In the following subsections, we use often the syntax of CPN-tool [6] specification language. The CPN-tool represents an implementation of the Hierarchical Timed CPNs. This tool allows the graphical representations of HTCPN models and their analysis.

II.3.1 The Illustrative example: The Algerian justice information system policy

We are interested in the access policy through the information system developed and built in the justice domain, in Algeria. The system has hierarchical users, simple time constraints and a conflict between roles. The justice domain is built upon a set of Justice Palaces (JPs) and courthouses (CHs). Each department in the country has one JP and a set of CHs dispatched on the area of the department according to the population number. The principal JP is in Algiers (the capital) and it manages all the justice process. As an example, Biskra department `http://www.courdebiskra.mjustice.dz/` contains one JP and three CHs. The information system which is being built is a multi-domain system, where the principal node is in Algiers. The Figure II.1 shows an abstract representation of these nodes.

[1] `https://drive.google.com/drive/folders/15zYjyRcaZ25L98Mniqqq3VZU4AlC5vrz`

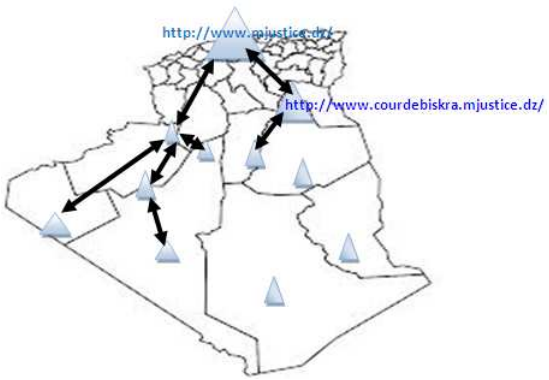


Figure II.1: An abstract view of the multi-domain tree of the Justice Information system in Algeria.

In case of our department (Biskra), the justice structure is composed of: (1) the Local Justice Palace, (2) the principal courthouse (BPCH) located in the downtown, and (3) two other courthouses (located in two suburban areas). This department is a sub-domain of the principal domain (located in Algiers). As an example, we specify only the security policy in the BPCH. The BPCH employs many persons who play different roles and have different permissions on the access and the manipulation of information required in the processed cases in the BPCH. Particularly, we will discuss the case's recourse process. When a recourse is made about some case's decision, the case is transferred to the Palace of justice. In the following, we will be interested in the users, roles, and permissions in the justice palace. The justice palace is a big structure which contains a high number of employees. The employees can be divided into a set of sections and subsections:

1. The procurator section: It is administrated by the procurator. It receives the most recourses. This section contains: two procurator's assistants, and two secretaries;

2. The administration of the Palace: a principal judge which administrate five other judges. Each judge manages a specific room (real estate room, commercial room, social room, civil room, and penal room). In each room, the judge has two consultants (which are also judges). In total, the Palace exploits 5 judges with 10 consultants;

3. The clerk's service: this section manages all the clerks employed in the Palace (more than 100 clerks work in the palace). This service is supervised by the clerk's chief. The service is divided into a set of sub-services. Each one is specialized in some tasks. Some important sub-services are: Editors sub-service, counter service, planning service. We have cited only the sub-services required in the treatment of a task which is: recourse request;

4. The information technology section: it is responsible for the information network and on the software applications. This section employs two engineers.

In this system, we will be interested in the following set of roles: **Secretary**, **Procurator Assistant**,
Procurator, **Planner**, **Consultant**, **Administrator Judge**, **Room Judge**, **Editor**, and **Citizen's Delegate** (which can make recourse on behalf the citizens). Each role has specific permissions on the information system used inside the Palace and connected to a

more large system throughout the entire town. The good definition of the permissions associated with each role and the verification of constraints is a mandatory task in this system. As an example, we detail the roles, tasks, and permissions required in the scenario of processing a recourse send to the office of the procurator. This scenario uses four logical objects: the recourse, the case, the session, and the decision. Each of these objects can be edited, viewed, modified, stamped, saved, sent, archived or deleted. These operations require permission to be done. The table in Table II.2 shows roles, tasks and their required permissions implied in this scenario. The table in Table II.3 presents the mapping between users and roles.

Table II.2: Roles, tasks, and permissions

Role	**tasks and permissions**
Secretary	Receive the recourse (Permission:$P(1)$), save the recourse(Permission:$P(2)$), archive the recourse (Permission:$P(3)$)
Procurator Assistant	Consult the recourse (Permission:$P(4)$), check the recourse(Permission:$P(5)$), request details on the recourse(Permission:$P(6)$), approve the recourse(Permission:$P(7)$)
Procurator	Consult the approved recourse (Permission:$P(8)$), send the recourse to the planing service (Permission:$P(9)$).
Planner	Save the recourse as a case (Permission:$P(10)$), send the case to consultant (Permission:$P(11)$), invoke the concerned person by the case (Permission:$P(12)$), inform the judge about the case (Permission:$P(13)$).
Consultant	Consult the case (Permission:$P(14)$), plan the session for this case (Permission:$P(15)$), send the plan to the planing section (Permission:$P(16)$).
Administrator Judge	Consult the case (Permission:$P(14)$), select a room judge for the case (Permission:$P(17)$).
Room Judge	Consult the case (Permission:$P(14)$), make the decision (Permission:$P(18)$) , consult the session report (Permission:$P(19)$), stamp the session report (Permission:$P(20)$), consult the decision report(Permission:$P(21)$), stamp the decision report (Permission:$P(22)$), archive the decision report (Permission:$P(23)$).
Editor	Edit the session report (Permission:$P(24)$, stamp the session report (Permission:$P(20)$), archive the session report (Permission:$P(25)$), edit the decision report (Permission:$P(26)$), print the decision report (Permission:$P(27)$), archive the decision report (Permission:$P(23)$).
Citizen's delegate	Edit a recourse (Permission:$P(28)$), send the recourse (Permission:$P(29)$), withdraw a recourse before its approving (Permission:$P(30)$), consult the decision report(Permission:$P(21)$).

In this system, there are two "Inheritance relations between roles":

1. The procurator (R1) inherits permissions of Procurator assistant (R2);
2. and the administrator judge (R3) inherits permissions of room judges (R4).

On the other hand, the system defines the following "Static conflicts" between roles:

- The Procurator (R1) and the judge (R3). Hence, no user can have these two roles assigned at the same time.
- The role citizen's delegate (R9) is in conflict with all the roles in the system (R1,..., R8).

Table II.3: Mapping from users to roles

Users	**Allowed roles**	**Role index**	**Enable time**
U1	Procurator	R1	Day time
U2,U3	Procurator assistant	R2	Day time
U4	Administrator Judge	R3	Day time
U5, . . . ,U9	Room Judge	R4	Day time
U10, . . . ,U30	Consultant	R5	Day time
U31,U32	Secretary	R6	Day time
U33, U34, U35	Editor	R7	Day time
U36, U37, U38	Planner	R8	Day time
U(i)∈{ U1, . . . ,U38 }	Citizen's delegate	R9	Day time

II.3.2 The global TRBAC model

Now, using the hierarchical principle in HTCPN, we can construct the global model as an abstract model. Figure II.2 shows this model. In this abstract model, the three events: Assignment, Activation, and enabling/disabling are represented as hyper transitions. The interaction between these events is modelled through the interfaces places. This model gives us an overview of the TRBAC policy, where all the events are modelled.

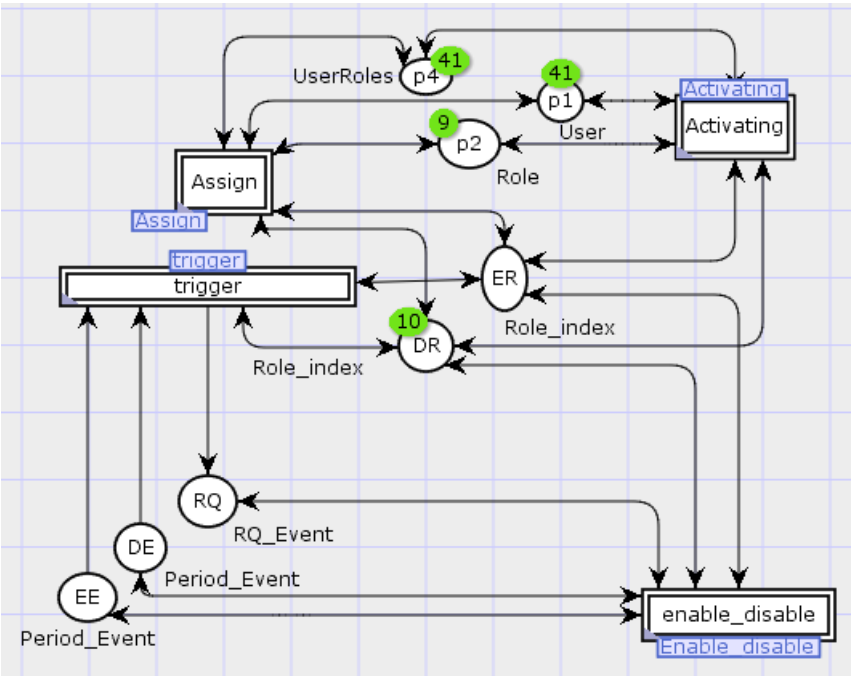


Figure II.2: HTCPN abstract model for TRBAC

In Figure II.2, transitions: trigger, Assign, Activate, and Enable_disable are hyper-transitions. Each transition represents one of the former sub-models. Table II.4 is an overview table presents all important places that are used in II.2, its meaning and its use. On all the following models, we do not depict the guards on transitions, in order to keep the models clear and legible.

Table II.4: An overview table with all important places and their meaning and use.

Place	**Place's meaning**	**Place's use**
DPE	Disabling Periodic Events	The set of periodic events that will disable roles (each event with its priority and its specific disabling time)
EPE	Enabling Periodic Events	The set of periodic events that will enable roles (each event with its priority and its specific enabling time)
EE	Enabling Events	The set of enabling periodic events with their next arrival time in the system
DE	Disabling Events	The set of disabling periodic events with their next arrival time in the system
ER	Enabling Roles	The set of currently enabled roles
DR	Disabling Roles	The set of currently disabled roles
P1	Users	The set of users with their static and dynamic cardinalities
P2	Roles	The set of roles with their static and dynamic cardinalities
P4	Assigned Roles	The set of assigned roles for each user

II.3.3 Modelling of role-enabling and role-disabling events

In the TRBAC model, the two events "role-enabling" and "role-disabling" can be prioritized and temporized. The priority is an integer that is associated with an event, and which is defined according to the other events that can occur in the system. In CPN model, the transition that has the highest priority (i.e., the smallest number) occurs first. For example, if there are two enabled transitions *T1* and *T2*, the priority of *T1* is "P_HIGH" with "$P\_HIGH = 100$", the priority of *T2* is
"P_NORMAL" with "$P\_NORMAL = 1000$", then the transition *T1* will fire before *T2*. We use this transition priority to ensure that if there are concurrent enabling and disabling requests at the same time (i.e., enabling and disabling requests of a role *r* at the same time respectively in places EPE and DPE, see Figure II.3), then the transition Arrival_CRQ (i.e., Arrival of Concurrent Requests) has a high priority more than the two transitions Arrival_EPE (Arrival of Enabling Periodic Event) and Arrival_DPE(Arrival of Disabling Periodic Event). This high priority allows the model to resolve, firstly, the conflicts between the two requests. The delay of role-*enabling* event and role-*disabl-ing* event is specified using temporal constraints. These constraints are: *periodic-events* and *roles-triggers*(they are mentioned in the last chapter.

II.3.3.1 Periodic events modelling

At runtime, roles can be enabled and disabled through requests. In TRBAC, a runtime request expression has the form $< pr : \ E\ after\ \triangle t >$, where pr is a priority, E is an event, and $\triangle t$ a duration of time. For example, the request:
$< h : \ enable$ nurse_on_day_duty after $2\ hours >$ means to enable the role
"nurse_on_day_duty" after two hours. This request has as priority h (high). In periodic-events, role-triggers, and runtime-request expressions, the value pr (priority) and $\triangle t$ can

be omitted. If this is the case, pr is set to the low level priority, and $\triangle t$ to 0.

To specify periodic events, in the Hierarchical Timed CPN model, we define the following types:

• **Role_Index**: This type models the set of roles. A role is specified as an identifier of type integer (INT in the specification). In the CPN-tool syntax specification, a role R_1 is expressed as R(1).

• **Timed_Role_Index**: This type is the product: Role_Index × Time, where: Time is the type timed (defined in CPN-tool. It allows the declaration of a discrete temporal variable). Tokens in the type Timed_Role_Index are stamped tokens.

• **Periodic_Event**: This type is the product: Interval × Prio × Role_Index × Time, where: Interval is a product type INT×INT. For example: (2, 10) denotes the interval of discrete numbers from 2 to 10. Prio is an integer type. Tokens in the type Periodic_Event are stamped tokens which model periodic events in the system.

• **Event**: This type is the product: Prio × Role_Index × Time. Tokens in the type Event are stamped tokens which model incoming request events due to periodic events.

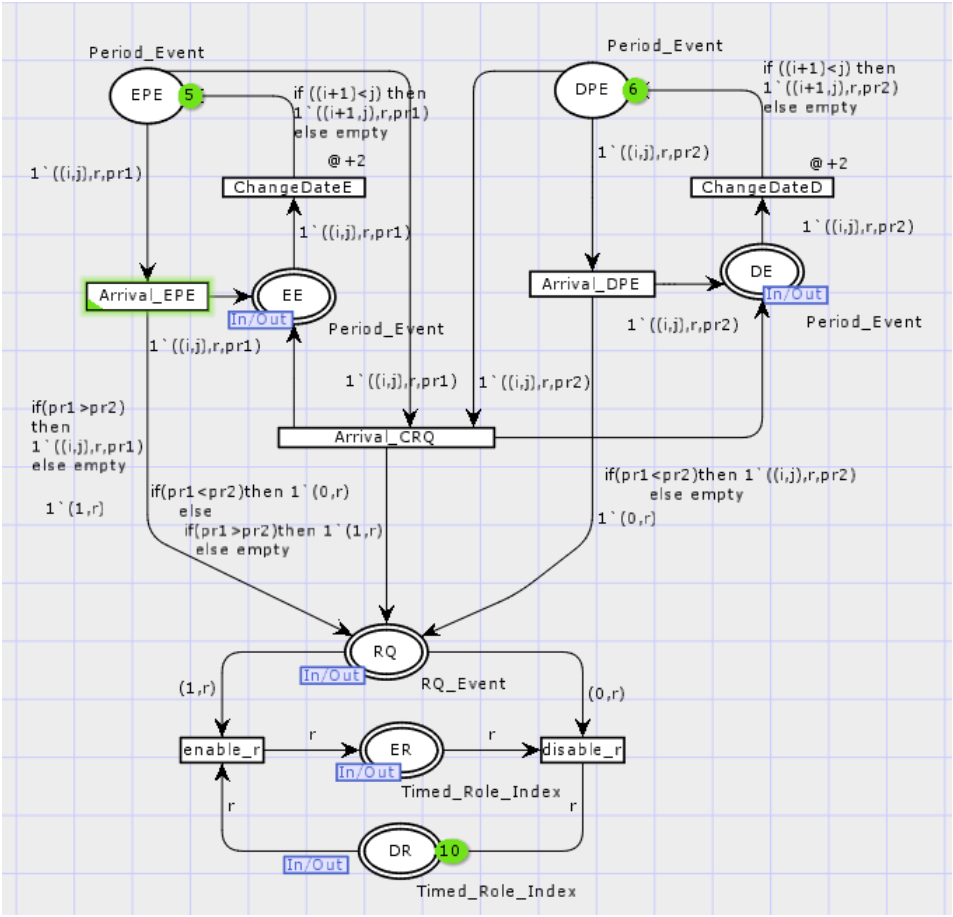


Figure II.3: Hierarchical Timed-CPN model for enabling/disabling roles

In Figure II.3, the places EPE and DPE will contain periodic (enabling or disabling) events. The expression $((i, j), pr, r)@t$ models a periodic events that will enable (in case of the arc (EPE, arrival_EPE)) or will disable (in case of the arc (DPE, arrival_DPE)) the role r, with a priority pr. The validity of these events are limited in the interval of time $[i, j]$, and they occur iteratively after a specific period at the instant $@t$. Transitions arrival_EPE, arrival_DPE, arrival_CRQ can be fired when some stamped tokens are present in EPE or|and DPE and so generate role-enabling or role-disabling requests in the place RQ (for a request). Due to the temporal aspect of HTCPN, periodic events that marked places EPE and DPE, are available only if the current time of simulation is equal to their stamp ($@t$). On the other hand, each one of these three transitions re-put the event in one of the places EE or DE to update its stamp (for its next arrival in the system). The update of the stamp of each periodic event is done by one of the two transitions ChangeDateE (for enabling events) and ChangeDateD (for disabling events).

In the case of our illustrative examples, all the roles are enabled in the journey. If we consider a day composed of two periods journey and night, and if the journey period starts at the instant $t = 1$ then the journey periods will occur at the stamps: @1, @3, @5, etc and the night periods will occur at the stamps: @2, @4, @6, etc. In Figure II.3, the two transitions ChangeDateE and ChangeDateD are timed transitions with the label $(@ + 2)$ which means that all events stamped $@t$ in the places EE or DE will be updated with a new stamp ($@t' = @(t + 2)$) and will be re-put again in the places EPE or DPE, if these events are yet valid (i.e. their intervals $[i, j]$ are always valid $(i \leqslant j)$). This is verified by the two conditional expressions labelling the arcs (ChangDateE, EPE) and (ChangeDateD, DPE).

The transition Arrival_EPE models incoming enabling requests, the transition Arrival_DPE models incoming disabling requests, and the transition Arrival_CRQ models incoming of concurrent events (enabling and disabling of the same role). The Transition Arrival_CRQ has the property **P_HIGH** which means that it will be always fired first once its preconditions are fulfilled. The transition Arrival_CRQ decides what kind of request must be put into the place RQ. The decision is based on the priority of each event (i.e. the attributes $pr1$ and $pr2$) and the expression:

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 equals to "1" for enabling request or equals to "0" for the disabling request. Finally, transitions disable_r and enable_r, will enable or disable a role. To enable a role, this role must be disabled at the current time (the role must be in the place DR), and to disable a role, this one must be enabled at the current time (the role must be in the place ER). The marking of the places depends on the scenario to be studied. In such scenario, we must identify: (i) periodic events representing the marking of places EPE and 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. As shown on models in Figure II.3 and Figure II.4, the places ER, DR, EE, DE and RQ are interface places. These places are shared between this model and the other models which will be presented below.

II.3.3.2 Trigger modelling

Triggers can be modelled as a set of transitions. Each trigger $< E_1, ..., E_n, C_1, ..., C_m \rightarrow pr : E\ after\ \Delta t >$ is modelled as a transition (time stamped with Δt) that has as inputs places DE, EE (for events: $E_1, ..., E_n$), DR, ER (for roles status: $C_1, ..., C_m$), and as output place: EPE or DPE (for enabling or disabling roles in specific period). As an example, the Figure II.4 shows an HTCPN model of the trigger:
$< (1 : enable\ R_1), (2 : disable\ R_2), R_3\ is\ enabled, R_4\ is\ disabled, > \rightarrow 3 : disable\ R_5\ after\ 4\ days.$

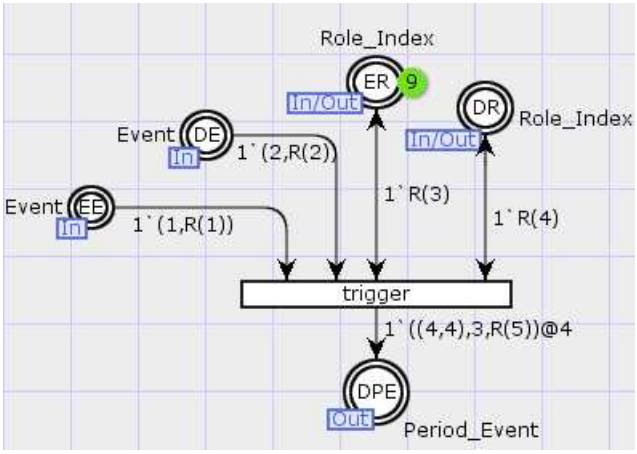


Figure II.4: Hierarchical Timed CPN model for triggers

All the places EE, DE, DPE, ER, DR are interface places. These places are shared with the model presented in Figure II.3.

II.3.4 Modelling of role-"assignment/deassignment" events and role-"activation/deactivation" events

In this section, we present the modelling of the events: *Assignment* (Figure II.5) a role to a user, *Activation* (Figure II.6) of a role by a user, *Deactivation* of a role, and *de-assignment* of a role. These events have only **non temporal** constraints. 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:

1. **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), and the dynamic cardinality (the maximum number of roles that this user can activate simultaneously). For example: (U(1), 3, 2) specifies user U_1, with a static cardinality equals to 3 and a dynamic cardinality equals to 2;

2. **Role**: to model roles. Each element in this type is a triple composed of three integer elements: the *role index* (Role_Index presented in the subsection II.3.3.1), 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(4), 5, 5) models the role R_4, with a static cardinality equals to 5 and a dynamic cardinality equals 5.

3. **Perm**: (for permission): to model permission. A permission is modelled as an integer index. For example: P(1) models the permission number 1;

Using the above three basic types, we define the new 8 following complex types.

1. **UserIndexList**, **RoleIndexList**, **PermIndexList**: three list types. An element in *UserIndexList* is a list of indices of users. Idem for the two other types *RoleIndexList* and *PermIndexList*. For example, a list of users can be [U(1), U(2), U(3)], containing three users U_1, U_2, and U_3. Idem for the list of roles and the 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. These three types (i.e., lists) will be used to define more complex types in the specification. These complex types are products of the above predefined types;

2. **UserRoles**=UserIndex $\times$ RoleIndexList. An element in this type is composed of a user index joined to a list of roles. This type is used to specify the list of roles that a user is allowed to play, is assigned to or it activates, currently. For example, the element (U(1), [R(1), R(2)]), according to its location (place) species the following: (i) in place P3, it specifies that user U_1 can play role R_1 and role R_2; (ii) in place P4, it specifies that roles R_1 and R_2 are assigned currently to user U_1; (iii) and in place P9, it specifies that user U_1 has activated role R_1 and role R_2;

3. **RoleUsers**=RoleIndex $\times$ UserIndexList. An element in this type is composed of a role index joined to a list of users. This type is used to specify either the list of users for which this role is currently assigned, or the list of users activating this role currently. For example, element (R(2), [U(1), U(2)]), according to its location (place), species the following: (i) in place P5, it specifies that role R_2 is assigned simultaneously to users U_1 and U_2; (ii) in place P12, it specifies that role R_1 is activated simultaneously by two users U_1 and U_2;

4. **RoleRoles** = RoleIndex $\times$ RoleIndexList. An element in this type is composed of a role index joined to a list of roles. This type is used to specify: (i) the inheritance hierarchy relation or the activity hierarchy relation (the set of junior roles are automatically activated by a user, when this user activates their senior role) between a senior role and the list of its junior roles; or (ii) the static or dynamic conflicts relation between a role and a list of other roles. For example, element (R(1), [R(2), R(3)]), according to its location (place), species the following: (i) in place P13, it specifies that role R_1 is the senior role for the two roles R_2 and R_3; (ii) in place P8, it specifies that roles R2 and R3, which are junior roles of R1, will be automatically activated by the user activating role R_1; (iii) in place P6, it specifies that role R1 is in static conflict with roles R_2 and R_3, which means that R_1 could not be assigned to the same users that have R_2 or R_3 as assigned role, currently; and (iv) in place P10, it specifies that role R_1 is in dynamic conflict with roles R_2 and R_3, which means that R_1 could not be activated by the same user that have activated R_2 or R_3, currently.

5. **UserRoleUsers** = UserIndex $\times$ RoleIndex $\times$ UserIndexList. An element in this type is composed of a user index joined to a role index, joined to a list of users. This type is used to specify the static or dynamic conflict relation between a user and a list of other users, on a specific role. For example, an element (U(1), R(2), [U(2), U(3)]), according to its location (place), species the following: (i) in place P7, it specifies that user U_1 is in static conflicts with two users U_2 and U_3 (on role R_2), which means that R_2 must not be assigned to U_1 if it is assigned, currently, to user U_2 or to user U_3; (ii) in place P11, it specifies that user U_1 is in dynamic conflicts with users U_2 and U_3, which means that U_1 could not activate R_2 if it is activated currently by user U_2 or by user U_3.

6. **UserPerm** = UserIndex $\times$ PermIndexList. An element in this type is composed of a user index joined to a list of permissions. This type is used to specify the list of permissions assigned to a user. For example, in this type, element (U(1),[P(1),P(2)]) specifies that user U_1 has two permissions P_1 and P_2;

7. **RolePerm** = RoleIndex $\times$ PermIndexList. An element in this type is composed of a role index joined to a list of permissions. This type is used to specify the list of permissions assigned to a role. For example, in this type, the element (R(1), [P(1), P(2)]) specifies that user R_1 defines two permissions P_1 and P_2;

8. **Role Hierarchy in Permission Conflict** = RoleIndex $\times$ RoleIndex $\times$ PermIndexList. In this type an element is composed of a first role index, a second role index and a list of permissions. This means that the first role inherits all permissions from the second role, except the permissions given in the list. For example (R(1), R(2), [P(1), P(2)]) specifies that role R_1 is the senior of the role R_2 but $R1$ cannot inherit the permissions P_1 and P_2 from R_2.

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

Based on the above types, the following paragraphs will present the modelling of two events (*role-assignment* and *role-activation*) in a TRBAC policy, as HTCPN models.

II.3.4.1 Modelling the role-assignment event

To assign a role R_i to a user U_i (see Figure II.5), the following 8 conditions must be satisfied.

1. In the system, we have a user U_i and a role R_i: this condition is satisfied by the presence of a user token $(U_i,\ SCU,\ DCU)$ in place P1 and the presence of a role token $(R_i,\ SCR,\ DCR)$ in place P2. $SCU,\ SCR,\ DCU$, and DCR are integers that represent the static cardinality and the dynamic cardinality for user U_i and role R_i, respectively;

2. User U_i is allowed to play role R_i: this condition is satisfied by the presence of a token (U_i, ARL) in place P3. ARL is a list of allowed roles and R_i must be in

the list ARL. We add a guard which implies that R_i is a member of ARL. In the CPN-tool, this guard is written: $[mem\ ARL\ R_i]$;

3. Role R_i is enabled: this condition is satisfied by the presence of role token R_i in place ER;

4. Role R_i is not yet assigned to user U_i: this is satisfied by the existence of a token $(R_i,\ RL)$ in place P4 and role R_i is not in list RL (written as a guard: $[mem\ RL\ R_i$= $false]$);

5. There is no static conflict between user U_i and users for which R_i is assigned currently: this is satisfied by: (1) the presence of a token $(R_i,\ UL)$ in place P5, where UL is the list of users for which R_i is assigned, (2) the presence of a token $(U_i,\ R_i,\ USCL)$ in place P_7 (defining users having static conflicts), where $USCL$ is a list of users that have conflicts with U_i, and finally (3) the intersection between $USCL$ and UL is empty (written as a guard: $[intersect\ USCL\ UL\ =\ nil]$);

6. There is not a static conflict between role R_i and roles assigned currently to user U_i: this is satisfied by:(1) the presence of a token $(U_i,\ RL)$ in place P4, where RL is the list of roles assigned to U_i, (2) the presence of a token $(R_i,\ RSCL)$ in place P6, where $RSCL$ is a list of roles that have conflict with R_i, and finally (3) the intersection between $RSCL$ and RL is empty (written as a guard: $[intersect\ RSCL\ RL\ =\ nil]$);

7. The number of assigned roles to U_i is less than the static cardinality of user U_i: this is satisfied by: (1) the presence of a token $(U_i,\ USC,\ UDC)$ in P1, (2) the presence of a token $(U_i,\ RL)$ in place P4, where RL is the list of roles assigned to U_i, and finally (3) the number of elements in RL is less than the static cardinality USC (written as a guard: $[length\ RL\ <\ USC]$);

8. The number of users that have R_i in their assigned roles is less than the static cardinality of role R_i: this is satisfied by: (1) the presence of a token $(R_i,\ SCR,\ DCR)$ in P1, (2) the presence of a token $(R_i,\ UL)$ in place P4, where UL is the list of users for which R_i is currently assigned, and finally (3) the number of elements in UL is less then RSC "Static Cardinality for R" (written as a guard: $[length\ UL\ <\ RSC]$).

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

1. Role R_i is assigned to user U_i: this is modelled by two effects: (1) An expression labelling the arc that updates the marking of place P4 (where roles assigned to each user are defined); This expression is written $(U_i,\ ins\ RL\ R_i)$, which means to add role R_i to the list of roles assigned currently to user U_i. (2) An expression labelling the arc that updates the marking of place P5 (where users assigned to each role are defined); This expression is written $(R_i,\ ins\ UL\ U_i)$, which means to add user U_i to the list of users assigned currently to role R_i;

2. All junior roles of role R_i become allowed to user U_i (according to **Role AHierarchy relation**): This is modelled by the updating of P3 marking, using the expression $(U_i,\ union\ RL\ RAHL)$, such that RL is the old list of roles allowed to U_i, and

$RAHL$ is the list of R_i junior roles (junior roles that are automatically activated by a user, when this last one activates R_i). The "union RL $RAHL$" is the union of the two lists. The list $RAHL$ is obtained from an input expression $(R_i,\ RAHL)$ from place P8 (P8 contains for each role the set of its junior roles);

According to the above description, the HTCPN model of the role-assignment event will be as depicted on the Figure II.5. In this Figure, types of places are near to places and their identifiers are inside the circles. The expressions labelling arcs are written near each arc. The guard associated with the transition "Assign" is $[mem\ ARL\ R_i,\ mem\ RL\ R_i = false, intersect\ ULSC\ UL = nil,\ intersect\ RLSC\ RL = nil, length\ RL < SCU, length\ UL < SCR]$.

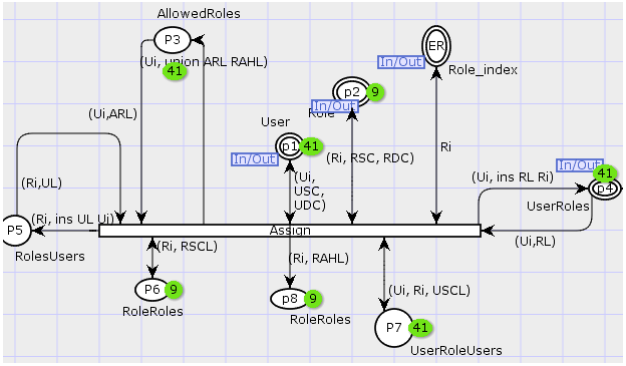


Figure II.5: HTCPN model for role-assignment event

In Figure II.5, places P1, P2, P4 and ER are interface places. ER is shared between this model and models presented in Figure II.3 and Figure II.4. P1, P2, and P4 are shared with the role-activation model (see Figure II.6, to be presented in the subsection II.3.4.2).

II.3.4.2 Modelling the role-activation event

The activation process of role R_i by user U_i is done through three steps (modelled by four transitions in the HTCPN model). Firstly, we check that all the necessary conditions to activate R_i by U_i are satisfied. If this is the case, role R_i is added to activated roles of U_i, and a first transition (**activate**) is fired. Secondly, we check if role R_i has some junior roles (with respect to the Inheritance Hierarchy relation); if this is the case then all these junior roles of R_i must be identified. This identification is done by the two transitions (**inheriting_roles_1** and **inheriting_roles_2**). Thirdly, the set of permissions of role R_i and all its junior roles are assigned to user U_i (using transition **taking_permissions**). During these three steps, user U_i and role R_i must not be used by another transition (for example transition "Assign"), and once these steps are finished, a fifth transition (**restore**) restores U_i and R_i to their original places P1 and P2.

The subnet that models the activation process shares places P1 (representing the set of users), P2 (representing the set of roles) and P4 (representing the assignment status) with the subnet modelling the assignment. There are seven necessary conditions to fire the transition "activate".

1. The system contains a user U_i and a role R_i: this is modelled by the presence of a user token $(U_i, SCU,$
$DCU)$ in place P1 and a role token (R_i, SCR, DCR) in place P2;

2. Role R_i is assigned to user U_i: this is modelled by a guard "R_i is a member in list RL", associated with transition **activate** and written as $[mem\ RL\ Ri]$. RL is the list of assigned roles to user U_i, and it is defined in token (U_i, RL), in place P4;

3. Role R_i is not yet activated by U_i: this is modelled by guard "R_i is not a member in the list ARL", written as $[mem\ ARL\ R_i = false]$. ARL is the list of activated roles by user U_i, and it is defined in token (U_i, ARL), in place P9;

4. Role R_i is not in a dynamic conflict with one of the currently activated roles by U_i: this is modelled by the guard: "the intersection between the two lists ARL and $RDCL$ is empty", written as $[intersect\ ARL\ RDCL = nil]$. $RDCL$ (the list of roles which are in dynamic conflicts with role R_i) is defined in token $(R_i, RDCL)$ in place P10. ARL (the list of activated roles by user U_i) is defined in token (U_i, ARL), in place P9;

5. User U_i is not in a dynamic conflict with one of users activating role R_i, currently: this is modelled by the guard: "the intersection between the two lists AUL and $UDCL$ is empty", written as: $[intersect\ AUL$
$UDCL = nil]$. AUL (the list of users activating role R_i) is defined in token $(R_i,\ ULAct)$, in place P12. $ULDC$ (the list of users which are in dynamic conflicts with U_i) is defined in token $(U_i, R_i, ULDC)$, in place P11;

6. The number of roles activated by user U_i is less than the dynamic cardinality of user U_i: this is modelled by the guard: "the number of elements in the list ARL is less than the value UDC", written as: $[length\ ARL < DCU]$. ARL (the list of activated roles by U_i) is defined in the token $(U_i,\ ARL)$, in place P9;

7. The number of users activating role R_i, currently, is less than the dynamic cardinality of role R_i: this is modelled by the guard: "the number of elements in AUL is less than RDC", written as: $[length\ AUL < RDC]$. AUL (the list of users activating role R_i) is defined in token $(R_i,\ AUL)$, in place P10.

After the firing of transition "**activate**", we will have the following.

1. Role R_i is added to the list of activated roles by user U_i: this is modelled by updating the marking of place P9, with token $(U_i,\ ins\ ARL\ R_i)$. The expression "$ins\ ARL\ R_i$" adds role R_i to list of activated roles (ARL) by user U_i;

2. User U_i is added to the list of users activating role R_i: this is modelled by updating the marking of place P12, with token $(R_i,\ ins\ AUL\ U_i)$;

3. User U_i takes all permissions defined for role R_i and all its juniors roles. This is modelled by: (i) firstly, extracting R_i juniors roles. This is done using transition "inheriting_roles_1". This transition requires a token $(R_i,\ IHRL)$ from place P13, where $IHRL$ is the list of junior roles of role R_i. This list is then added into place P16. (ii) Secondly, transition "inheriting_roles_2" is fired iteratively to extract all roles from list $IHRL$. When "inheriting_roles_2" is fired, it puts a role in place

P18. (iii) Transition "taking_permissions" checks place P18, and once a role R_j is found in P18, this transition uses token (R_j, RPL) (where RPL is the list of permissions defined in role R_j) from place P19 to update the marking of place P20, with token $(U_i,\ union\ UPL\ RPL)$. Expression "$union\ UPL\ RPL$" is evaluated to the union of list RPL and list UPL (the permissions assigned previously to user U_i); (iv) Finally, when list $IHRL$ is empty, transition "restore" re-puts user U_i and role R_i into their places P1, P2, and ER (enabled Roles).

According to the above description, the HTCPN model of the activation-event will be as shown in FigureII.6.

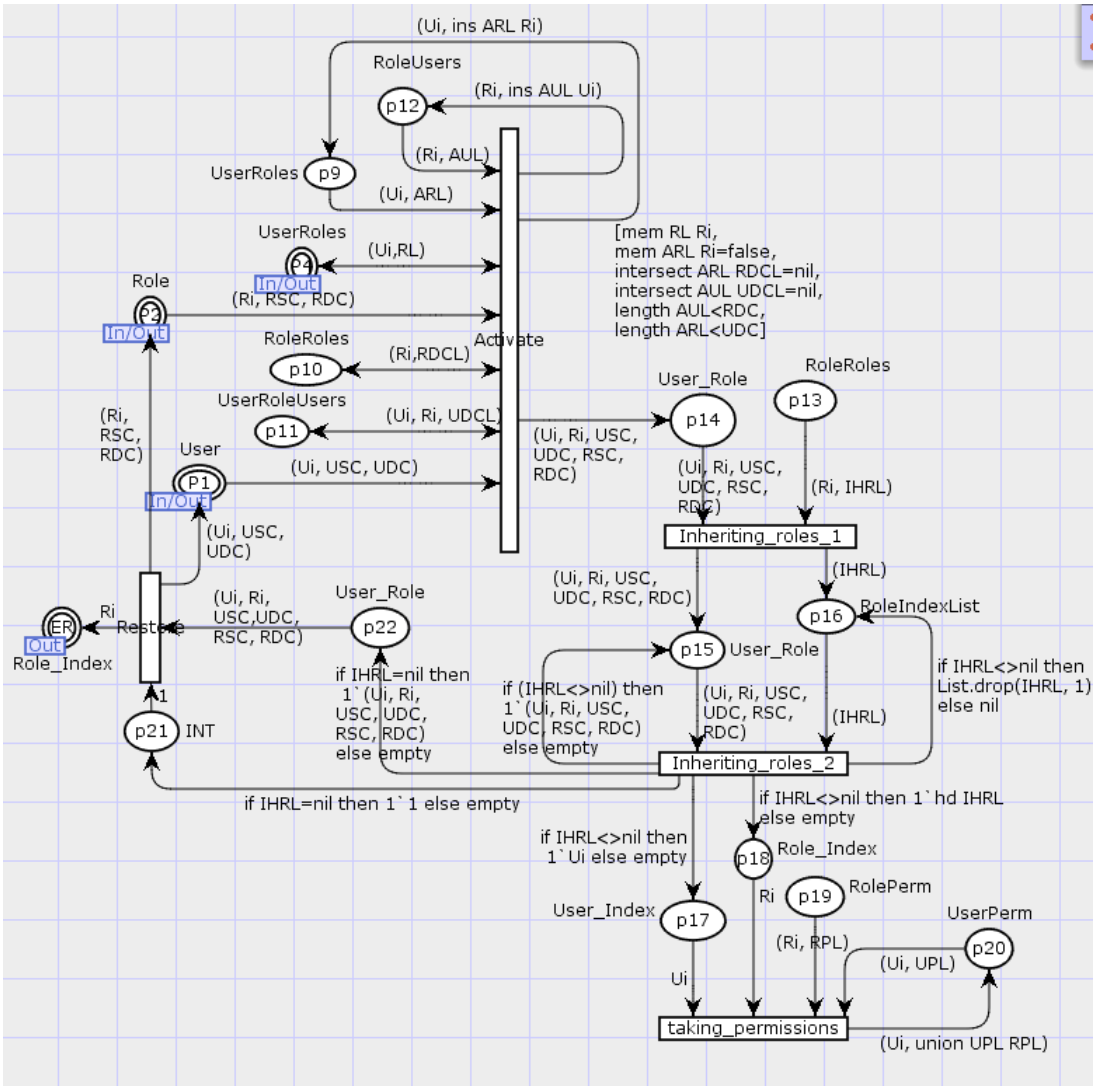


Figure II.6: HTCPN model for role-activation event

II.3.5 Formalisation and verification of properties

The verification of the policy model is realized using the CPN-tool [6]. CPN-tool allows the simulation of the model and the verification of several properties. In the case

of the HTCPN model for a TRBAC 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 a TRBAC policy. The properties, which will be verified, depending on the chosen scenario. In this section, we use the TRBAC policy presented above in the illustrative example (Section II.3.1), and we will apply the analysis by reachability graph computed by the CPN-tool. Based on this reachability graph, we build programs in the ML language [**?**] to prove several proprieties. The ML programs are executed by the CPN-tool, and the results of the verification are presented. In the following subsections, we will present three classes of properties concerning respectively (i)the role enabling/disabling model, (ii)the role-assignment event model, and finally (iii)the role-activation event model.

II.3.5.1 Formalisation and verification of properties on the role enabling/disabling model

Three temporal properties that a designer can be interested to verify on the model.

- **Property 1:** "A role $R(i)$ is enabled during its authorized period". In the illustrative example, all the roles must be enabled in the journey period. We have verified the property for a short interval of time (two days: the journey of the first day t=1 and the journey of the second day t=3). The marking of place ER (enabled roles) proves that all roles can be enabled in each journey in the specified interval (i.e., 2 days).

- **Property 2:** "A role $R(i)$ will never be enabled except during its authorized period". This is formulated as: when the time of the system is t, every enabled role $R(i)$ (in the place ER) must be allowed to be enabled at this instant t (i.e., the stamped token $(([i, j], R(i))@t)$ exists in the place EE). In the illustrative example, all roles will never be enabled in the night period. As an example, for an interval of two days (the night of the first day t=2 and the night of the second day t=4), markings prove that all enabled roles in ER are respecting the enabling constraints.

- **Property 3:** "A role $R(i)$ will never be disabled in its authorized period". This is formulated as: When the time of the system is t, every disabled role $R(i)$ ($R(i)@t$ (existing in DR) must be in place DE (i.e., DE contains the stamped token $(((i, j),$ $R(i))@t)$). In the illustrative example, all roles will never be disabled during the journey period. In a short interval of 2 days, the marking of the place DR (disabled roles) illustrates that all roles disabled are stamped with (t=2 or t=4) and proves that all roles are disabled only in the night period.

II.3.5.2 Formalisation and verification of properties on the role-assignment event model

The model in Figure II.5 is a template which can be used to verify any TRBAC policy. The example illustrated in section II.3.1 is used to define the initial marking of the model. We propose to add three anonymous users U(39), U(40) and U(41) which can be assigned to citizen's delegate role. The reachability graph report shows the markings of places. Indeed, the simulation report (with binding option) shows the sequence of the executed bindings (firing transitions), one by one with the time of each binding. Hence, the proposed approach uses the markings of places to verify non-temporal properties. To verify the temporal properties, the approach uses markings of places and executed bindings. In the temporal properties verification, we use the illustrative example enriched with a new

role $R(10)$ (denoting a Night_Security_Agent). $R(10)$ has an enable time which is "night time" and it is allowed for a new user $U(42)$. Once the model has its initial marking, one can be interested to verify the following properties.

- **Property 1:** "Can a user $U(i)$ play all its allowed roles?". This is formulated as: From each state where $U(i)$ is allowed to play the set of roles $\{R(1), ..., R(n)\}$ (i.e., a state where the marking of P3 contains $(U(i), [R(1), ..., R(n)])$), and for each $R(i)$ in $\{R(1), ..., R(n)\}$, we must find some path in the reachability graph in which there is at least one binding: $< assign, U_i = U(i), R_i = R(i) >$ (a state where the marking of P4 contains $(U(i), [R(i)])$)). Considering the policy (presented in section: II.3.1), the reachability graph report shows the markings of places P3 (Figure II.7) and P4 (Figure II.8);

```
     Assign'P3 1
 1`(U(1),[R(1)])++1`(U(2),[R(2)])++1`(U(3),[R(2)])++
1`(U(4),[R(3)])++1`(U(5),[R(4)])++1`(U(6),[R(4)])++
1`(U(7),[R(4)])++1`(U(8),[R(4)])++1`(U(9),[R(4)])++
1`(U(10),[R(5)])++1`(U(11),[R(5)])++1`(U(12),[R(5)])++
1`(U(13),[R(5)])++1`(U(14),[R(5)])++1`(U(15),[R(5)])++
1`(U(16),[R(5)])++1`(U(17),[R(5)])++1`(U(18),[R(5)])++
1`(U(19),[R(5)])++1`(U(20),[R(5)])++1`(U(21),[R(5)])++
1`(U(22),[R(5)])++1`(U(23),[R(5)])++1`(U(24),[R(5)])++
1`(U(25),[R(5)])++1`(U(26),[R(5)])++1`(U(27),[R(5)])++
1`(U(28),[R(5)])++1`(U(29),[R(5)])++1`(U(30),[R(5)])++
1`(U(31),[R(6)])++1`(U(32),[R(6)])++1`(U(33),[R(7)])++
1`(U(34),[R(7)])++1`(U(35),[R(7)])++1`(U(36),[R(8)])++
1`(U(37),[R(8)])++1`(U(38),[R(8)])++1`(U(39),[R(9)])++
1`(U(40),[R(9)])++1`(U(41),[R(9)])
```

Figure II.7: marking of P3

```
     TRBAC'p4 1
1`(U(1),[R(1)])++1`(U(2),[R(2)])++1`(U(3),[R(2)])++
1`(U(4),[R(3)])++1`(U(5),[R(4)])++1`(U(6),[R(4)])++
1`(U(7),[R(4)])++1`(U(8),[R(4)])++1`(U(9),[R(4)])++
1`(U(10),[R(5)])++1`(U(11),[R(5)])++1`(U(12),[R(5)])++
1`(U(13),[R(5)])++1`(U(14),[R(5)])++1`(U(15),[R(5)])++
1`(U(16),[R(5)])++1`(U(17),[R(5)])++1`(U(18),[R(5)])++
1`(U(19),[R(5)])++1`(U(20),[R(5)])++1`(U(21),[R(5)])++
1`(U(22),[R(5)])++1`(U(23),[R(5)])++1`(U(24),[R(5)])++
1`(U(25),[R(5)])++1`(U(26),[R(5)])++1`(U(27),[R(5)])++
1`(U(28),[R(5)])++1`(U(29),[R(5)])++1`(U(30),[R(5)])++
1`(U(31),[R(6)])++1`(U(32),[R(6)])++1`(U(33),[R(7)])++
1`(U(34),[R(7)])++1`(U(35),[R(7)])++1`(U(36),[R(8)])++
1`(U(37),[R(8)])++1`(U(38),[R(8)])++1`(U(39),[R(9)])++
1`(U(40),[R(9)])++1`(U(41),[R(9)])
```

Figure II.8: marking of P4

The marking of P4 confirms that every role allowed to a user in P3 is assigned to this user in P4. For example, the role Procurator (R(1)) is assigned to user U(1), the two users U(2) and U(3) have the role Procurator Assistant (R(2)), etc.

- **Property 2:** "A user $U(i)$ will never have a role $R(i)$ which could not be assigned to it". This is formulated as: Every role $R(i)$ assigned to user $U(i)$ in place P4 (containing assigned roles for each user), must be an allowed role to user $U(i)$ in the place P3 (containing allowed roles for each user). Indeed, the marking of P3 and P4 in the previous figures show that every role assigned to a user is an allowed role to it.

- **Property 3:** Cardinality constraint 1: "A user will never have a number of assigned roles greater than its SCU (static cardinality for a user)". This is formulated as: In place P4 (containing assigned roles for each user), each user $U(i)$ must have in its list of roles a number of roles less than or equal to its SCU (defined in place P1, see Figure II.9). Indeed, in the marking of P1, all users have SCU=1. The marking of P4 proves that every user in the system has only one assigned role. For example, users U(39), U(40) and U(41) must be assigned only to role R(9) (citizen's delegate).

```
     TRBAC'p1 1
1`(U(1),1,1)++1`(U(2),1,1)++1`(U(3),1,1)++1`(U(4),1,1)++
1`(U(5),1,1)++1`(U(6),1,1)++1`(U(7),1,1)++1`(U(8),1,1)++
1`(U(9),1,1)++1`(U(10),1,1)++1`(U(11),1,1)++1`(U(12),1,1)++
1`(U(13),1,1)++1`(U(14),1,1)++1`(U(15),1,1)++1`(U(16),1,1)++
1`(U(17),1,1)++1`(U(18),1,1)++1`(U(19),1,1)++1`(U(20),1,1)++
1`(U(21),1,1)++1`(U(22),1,1)++1`(U(23),1,1)++1`(U(24),1,1)++
1`(U(25),1,1)++1`(U(26),1,1)++1`(U(27),1,1)++1`(U(28),1,1)++
1`(U(29),1,1)++1`(U(30),1,1)++1`(U(31),1,1)++1`(U(32),1,1)++
1`(U(33),1,1)++1`(U(34),1,1)++1`(U(35),1,1)++1`(U(36),1,1)++
1`(U(37),1,1)++1`(U(38),1,1)++1`(U(39),1,1)++1`(U(40),1,1)++
1`(U(41),1,1)
```

Figure II.9: marking of P1

- **Property 4:** Cardinality constraint 2: "A role will never be assigned to a number of users greater than its SCR (static cardinality for a role)". This is formulated as: In place P5 (containing assigned users for each role), each role $R(i)$ must have in its list of users a number of users less than or equal to its SCR (defined in place P2). In fact, the marking of P5 (Figure II.11) proves that the number of users assigned to a role is equal to its SCR predefined in P2 (Figure II.10). For example, in place P2, role R6 (Secretary) has a cardinality constraint equals to 2 and in place P5, role R(6) has exactly two assigned users U(31) and U(32).

```
     TRBAC'p2 1
1`(R(1),1,1)++
1`(R(2),2,2)++
1`(R(3),1,1)++
1`(R(4),5,5)++
1`(R(5),21,21)++
1`(R(6),2,2)++
1`(R(7),3,3)++
1`(R(8),3,3)++
1`(R(9),10,10)
```

Figure II.10: marking of P2

```
     Assign'P5 1
1`(R(1),[U(1)])++1`(R(2),[U(2)])++1`(R(2),[U(3)])++
1`(R(3),[U(4)])++1`(R(4),[U(5)])++1`(R(4),[U(6)])++
1`(R(4),[U(7)])++1`(R(4),[U(8)])++1`(R(4),[U(9)])++
1`(R(5),[U(10)])++1`(R(5),[U(11)])++1`(R(5),[U(12)])++
1`(R(5),[U(13)])++1`(R(5),[U(14)])++1`(R(5),[U(15)])++
1`(R(5),[U(16)])++1`(R(5),[U(17)])++1`(R(5),[U(18)])++
1`(R(5),[U(19)])++1`(R(5),[U(20)])++1`(R(5),[U(21)])++
1`(R(5),[U(22)])++1`(R(5),[U(23)])++1`(R(5),[U(24)])++
1`(R(5),[U(25)])++1`(R(5),[U(26)])++1`(R(5),[U(27)])++
1`(R(5),[U(28)])++1`(R(5),[U(29)])++1`(R(5),[U(30)])++
1`(R(6),[U(31)])++1`(R(6),[U(32)])++1`(R(7),[U(33)])++
1`(R(7),[U(34)])++1`(R(7),[U(35)])++1`(R(8),[U(36)])++
1`(R(8),[U(37)])++1`(R(8),[U(38)])++1`(R(9),[U(39)])++1
`(R(9),[U(40)])++1`(R(9),[U(41)])
     Assign'P6 1
```

Figure II.11: marking of P5

- **SoD Constraints:** We consider the two following properties.

1. "A role must never be assigned to two conflicting users (Static Conflict between users)". Let us consider $LU1 = [U(1), ..., U(n)]$ the list of users assigned to role $R(i)$ in place P5 (containing assigned users for each role). Let us consider $LU2 = [U'(1), ..., U'(m)]$ the list of conflicting users on the role $R(i)$ in place P7. The constraint is formulated as: for all role $R(i)$, $LU2 \cap LU1 = \phi$.
2. "A user must never have two assigned conflicting roles (Static Conflict between roles)". Let us consider $LR1 = [R(1), ..., R(n)]$ the list of roles assigned to user $U(i)$ in the place P4 (containing assigned roles for each user). Let us consider $LR2 = [R'(1), ..., R'(m)]$ the list of conflicting roles on user $U(i)$ defined in place P6. The constraint is formulated as: for all user $U(i)$, $LR2 \cap LR1 = \phi$.

In our illustrative example, we have considered one conflict between roles. It is a static conflict between role "citizen's delegate" (R9) and all the other roles. The marking of place P4 proves that there is no role from (R1, R2, R3, R4, R5, R6, R7, R8) assigned to a user which has R9 as an assigned role.

Besides the above properties, the designer can be interested to verify the two following temporal properties.

- **Temporal Property 1:** "Can a user $U(i)$ play all its allowed roles during their authorized periods?". This is formulated as: From each state where $U(i)$ is allowed to play the set of roles $R(1), ..., R(n)$ (i.e., the marking of P3 contains $(U(i), [R(1), ..., R(n)])$), and for each role $R(i)$ in $R(1), ..., R(n)$, and for each period per which is an authorized period for $R(i)$ (i.e., the marking of DPE contains the stamped token $((I, J), R(i))@per$), we must find a path in the reachability graph in which there is a binding:
$< assign, Ui = U(i), Ri = R(i), time = t >$, such that t must coincide with period per. In the illustrative example, the role Procurator $R(1)$ is an allowed role for user $U(1)$, and the authorized period for it is the journey. Indeed, the role Night_Security_Agent is an allowed role for the user $U(42)$, and the authorized period for it is the Night. In a short interval of 2 days, the verification proves that: (i) Role $R(1)$ is assigned to $U(1)$ at two times t=0 and t=2; (ii) Role $R(10)$ is assigned to $U(42)$ at one time t=1. Figures II.12,II.13 and II.14 are snapshots from the simulation report. The first line in figure II.12 shows that the step 63 is the firing of the transition "Assign" at time 0. The rest is the binding of the transition's variables.

```
63    0     Assign @ (1:TimeModeling)
 - USCL = []
 - UDC = 1
 - USC = 1
 - ARL = [R(1)]
 - Ui = U(1)
 - RL = []
 - RSCL = []
 - RDC = 1
 - RSC = 1
 - Ri = R(1)
 - UL = []
64    0     Assign @ (1:TimeModeling)
```

Figure II.12: Simulation report step 63

- **Temporal Property 2:** "A user $U(i)$ will never play its allowed role $R(i)$ during a non authorized period per". This is formulated as: when the time of the system

```
76   1    Assign @ (1:TimeModeling)
 - USCL = []
 - UDC = 1
 - USC = 1
 - ARL = [R(10)]
 - Ui = U(42)
 - RL = []
 - RSCL = []
 - RDC = 1
 - RSC = 1
 - Ri = R(10)
 - UL = []
```

Figure II.13: Simulation report step 76

```
121  2    Enabling_r @ (1:TimeModeling)
 - r = R(3)
122  2    Assign @ (1:TimeModeling)
 - USCL = []
 - UDC = 1
 - USC = 1
 - ARL = [R(1)]
 - Ui = U(1)
 - RL = []
 - RSCL = []
 - RDC = 1
 - RSC = 1
 - Ri = R(1)
 - UL = []
123  2    Enabling_r @ (1:TimeModeling)
 - r = R(9)
```

Figure II.14: Simulation report step 122

is t, every role $R(i)$ assigned to a user $U(i)$ (i.e., in place P4), must be an enabled role in place ER at the same time t, such that t must coincide with period per. For example, user $U(1)$ plays $R(1)$ at $t = 0$ and $t = 2$, hence role $R(1)$ must be enable at $t = 0$ and $t = 2$. Figures II.15 and II.16 are snapshots from the simulation report which proves that the role $R(1)$ is enabled at $t = 0$ and $t = 2$.

```
18   0    Enabling_r @ (1:TimeModeling)
 - r = R(2)
19   0    Enabling_r @ (1:TimeModeling)
 - r = R(1)
```

Figure II.15: Simulation report step 19

```
118  2    desAssign @ (1:TimeModeling)
 - Ui = U(42)
 - RL = [R(10)]
 - Ri = R(10)
 - UL = [U(42)]
119  2    Enabling_r @ (1:TimeModeling)
 - r = R(1)
```

Figure II.16: Simulation report step 119

II.3.5.3 Formalisation and verification of properties on the role-activation event model

Using the model in Figure II.6, one can be interested to verify the following properties:

- **Property 1:** "Can a user U_i activate all its assigned roles?". This property is formulated as: from each state where $\{R(1), ..., R(n)\}$ are assigned to $U(i)$ (which means a state where the marking of P4 contains $(U(i), [R(1), ..., R(n)])$), and for each $R(i)$ in $R(1), ..., R(n)$, we must find some path in the reachability graph in which there is at least one binding: $< activate, U_i = U(i), R_i = R(i) >$". To prove this property, we select just users U_1 and U_2 to verify if a user can activate all its assigned roles. From place P4 we have assigned roles of U_1 and U_2: 1'(U(1),[R(1)])++ 1'(U(2),[R(2)]). So we should find that U_1 and U_2 can activated

their roles. Indeed, the marking of place P9 (place of activated roles) is equal to 1'(U(1),[R(1)])++1'(U(2), [R(2)]). This marking proves that U_1 and U_2 can activate all their roles;

- **Property 2:** "A user will never activate a role which is not assigned to it". This is formulated as the following: every role $R(i)$ activated by a user $U(i)$ in the place P9 (containing activated roles by each user), must be an assigned role to this user in the place P4 (containing assigned roles for each user). Indeed, from the marking of place P9, we see that every activated role by a user is an assigned role to this user in marking of place P4.

- **Property 3:** Cardinality constraint 1: "A user will never activate a number of roles more than its DCU (dynamic cardinality for a user)". This is formulated as: in place P9 (containing activated roles by each user), each user $U(i)$ must have in its list of roles a number of roles less than or equal to its DCU (defined in place P1). Indeed, the marking of place P1 illustrates that the DCU for all the users is one, and the marking of place P9 proves that every user activates just one role;

- **Property 4:** Cardinality constraint 2: "A role will never be activated by a number of users greater than the DCR (dynamic cardinality for a role)". This is formulated as: in place P12 (containing users activating each role), each role $R(i)$ must have in its list of users a number of users less than or equal to its DCR (defined in place P2). From the marking of P2, the DCR of R(1) is 1 and the DCR of R(2) is 2. The marking of P12 is 1'(R(1),[U(1)]) $+ +$ 1'(R(2),[U(2)]). Thus, from the marking of P12, R(1) and R(2) are activated by one user (≤ 2) and this proves that the DCR constraint is respected;

- **SoD constraints:**

  The three following constraints are considered.

  - "A role must never be activated by two conflicting users (Dynamic Conflict between users)": Let $LU1 = [U(1), ..., U(n)]$ be the list of users activating role $R(i)$ (i.e., in place P12). Let $LU2 = [U'(1), ..., U'(m)]$ be the list of users in conflicts with user $U(i)$ on the role $R(i)$ (defined in the place P11). The constraint is formulated as: for all user $U(i)$ in $LU1$: $LU2 \cap LU1 = \phi$;
  - "A user must never activate two conflicting roles (Dynamic Conflict between roles)". Let $LR1 = [R(1), ..., R(n)]$ be the list of activated roles by the user $U(i)$ (i.e., in place P9). Let $LR2 = [R'(1), ..., R'(m)]$ be the list of roles which can not be activated simultaneously by $R(i)$ (i.e., in place P10). The constraint is formulated as: for each user $U(i)$: $\mid LR2 \cap LR1 = \phi \mid \leq 1$;

- **Property 5:** "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 follows: every permission assigned to user $U(i)$ (i.e., in place P20), must be a permission defined in a role $R(j)$ (i.e., in place P19), and role $R(j)$ must be activated by user $U(i)$ (i.e., in P9). Indeed, from the marking of place P9, one can find that roles $R(1)$ and $R(2)$ are activated simultaneously by users U_1 and U_2, respectively. $R(1)$ is a senior role of $R(2)$, hence $R(1)$ should inherit $R(2)$ permissions and all permissions of $R(2)$ juniors. The marking of place P19 proves this situation.

Besides the above properties, the designer can be interested to verify the following three temporal properties.

- **Temporal Property 1:** "Can a user $U(i)$ activate all its assigned roles in their authorized periods?". This is formulated as: from each state where roles $\{R(1), ..., R(n)\}$ are assigned to $U(i)$ (i.e., the marking of P4 contains $(U(i), [R(1), ..., R(n)])$), such that for all $R(i)$ in $\{R(1), ..., R(n)\}$, $R(i)$ is enabled in period per (i.e., EPE contains $((I, j), R(i))@per$), we must find a path in the reachability graph in which there is a binding: $< activate, Ui = U(i), Ri = R(i), time = per >$.

- **Temporal Property 2:** "A user $U(i)$ will never activate a role $R(i)$ during a unauthorized period?". This is formulated as: when the time of the system is t, every role $R(i)$ activated by a user $U(i)$ (i.e., in place P9) must be an assigned role to this user (i.e., in the place P4), at a period which must coincide with t;

- **Temporal Property 3:** "A user has only permissions defined in its activated roles, and in their authorized periods". This is formulated as the following: when the time of the system is t, every permission assigned to user $U(i)$ (i.e., in place P20) must be a permission defined in role $R(j)$ (i.e., in the place P19) and role $R(j)$ must be activated by $U(i)$ (i.e., in P9), at a period which must coincide with t.

II.4 Conclusion

The use of Petri nets to model RBAC (Role Based Access Control) [81] policies is one of the ambitious axes to ensure the analysis and verification of RBAC security policies. Petri nets formalism is an event based formalism, thus specification with Petri nets requires defining events from RBAC models. Most works consider **enabling**, **disabling**, **assignment**, and **activation** of roles as the basic events that must be modelled and analysed. The analysis of Petri net models allows the designer to prove the consistency of its policy. One can use the reachability graph of Petri net model to analyse and to verify the policy. Coloured Petri Nets (CPNs) represent an extension of Petri nets with more expressive power. The modelling of RBAC using CPNs [5] is more practical than using classical Petri Nets. The CPN-tool [6] is an efficiency automatic tool that can be used to model, simulate and verify CPN models.

Some extensions of RBAC (Temporal RBAC [3], General Temporal RBAC [28]) consider temporal constraints on RBAC events. The use of a temporal formalism, like Timed CPN, is more adequate to model these constraints. Besides the temporal aspect, the use of hierarchical aspect allows the designer to have well organized models. The hierarchy helps the designer to manage and to understand its model. In this chapter, we have presented an approach (down-top) to model and to analyse TRBAC policies. This approach uses Hierarchical Timed CPNs formalism to construct models of different events. After the construction of models, we have exploited CPN-tool to verify the consistency of models. Through this chapter, we have presented how to specify several TRBAC constraints, we have presented a scenario example, and we have discussed the results of its analysis, and what kinds of properties to be checked.

The current work can be extended through several levels. On a first level, we propose to generalize the approach to deal with more extensions of RBAC as: GTRBAC, XGRP-RBAC [31], H-RBAC [32], and Tie-RBAC [33]. Another important ambition concerns the exploiting of CPN-tool abilities. The CPN-tool uses the XML (eXtended Markup

Language) to save specifications. This quality can be used to exploit the current specification as a template where the 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: (i) specify RBAC policies in XML, (ii) composition of policies, (iii) optimization of their composition, and finally (iv) formal verification of policies. By using XML specification language, this work can be integrated with the other parts of the project. Finally, we propose to tackle with the verification optimisation. Indeed, we use a state transition based model as coloured Petri nets; therefore, the verification process is based on the reachability graph using a model-checker (i.e., the CPN-tool). The major limit of the model checking process is the explosion in state space when the models are complicated and the process can be stopped due to a stack overflow in the memory. To overcome such a limit, the CPN-tool implementation uses symbolic model-checking which reduces the reachability graph based on symmetric properties in that graph. In our future work, we aim to tackle this aspect in order to optimize the verification process of large policies.

Chapter III

Second Contribution: Fine-grained Role-Attribute based Access Control model (FRABAC): A New Hybrid Access Control Model

III.1 Introduction

New technologies as cloud computing and internet of things (IoT) has expanded the range of applications. This expansion, in several computing and heterogeneous environments, makes access control an important issue. Indeed, a variety of access control models have been developed to address different aspects of security problems. The two most popular basic models are: Role Based Access Control (RBAC) and Attribute Based Access Control (ABAC). Both models RBAC and ABAC have their specific features and they can complement each other. For that, providing a hybrid model which considers both concepts "roles" as well as "attributes" has become an important research topic.

In this chapter, we propose a new access control model based principally on roles, attributes, access modes and the type of resources. An empirical method is applied to compare the new proposed model versus three existing models: RBAC, ABAC, and the hybrid model Attribute Enhanced RBAC (AERBAC). This empirical comparison is based on four metrics that are inspired from the limitations of RBAC and ABAC.

The rest of the chapter is organized as follows. Section III.2 exposes related work dealing with RBAC, ABAC and hybrid models. Section III.3 starts by presenting the requirements and needs for a new access control model, then it presents the principle and the components of the new proposed access control model. Section III.4 details the principles of the proposed empirical comparison approach, the proposed metrics, a demonstrative example, and analyses the models RBAC, ABAC, AERBAC and the new proposed model using the defined metrics. Finally, section III.5 concludes the proposed work.

III.2 Related work

In RBAC model, permissions are associated with roles that users have as a part of an organization. Thus, the user's access to resources is decided based on his role. Therefore, a role can be considered as a collection of users that have the same set of permissions. This approach has two principal advantages, on one hand, users will access only to the resources that they require to achieve their tasks, under the suitable mode. On the other hand, the system administration is made easy. However, in the basic RBAC, the access decision will be complex [26] and not adequate [27] when the contextual attributes are required to granting the access. Moreover, the permissions are referring to individual objects. This kind of referring leads to role-permission explosion problem in situations including a large number of objects. To resolve these disadvantages, ABAC [82] was proposed. The ABAC model introduces the concept of the attribute, hence an ABAC system is composed of three sets of entities: users, resources and the environment. Each of these three entities has specific attributes. An attribute consists of a pair *(key, value)* and the permissions of users depend on their attributes. Even the ABAC was proposed to facilitate the management of security, the proposed solution by ABAC can be as complicated as that of RBAC in some cases [11]. According to [50], in ABAC the role names are still associated with users, but they are no more considered as collections of permissions. In most systems, there are private objects dedicated to a particular user and where the access is qualified as "unique access" (for example, the report card of a student) versus "multiple access" in the case of shared objects. To restrict the access to these private objects, the two models resolve the situation differently. In fact, RBAC introduces a private role for each student whereas ABAC introduces a private rule for each student. In this case,

the system does not benefit from the advantages of the role of RBAC and the attributes of ABAC. Besides this problem, granting a request of a user in both models (RBAC or ABAC) requires to check the user permissions one by one to make a decision to grant or deny the access.

According to [83], RBAC and ABAC cannot be directly applied to IoT because of their limitations. However, both models still have some advantages that can be exploited in IoT applications. RBAC deals with the distribution problem of competencies where time and location change, while ABAC deals with the dynamic propagation problems of users. Both models RBAC and ABAC have their specific features and they can complement each other. The idea to merge RBAC and ABAC in one model has become an important research topic, in order to acquire advantages of these two models. However, the proposed solutions for merging both models are still insufficient. Indeed, NIST organization has announced a challenging project to define a new security model [27] based on the both existing models. Many researchers have adopted the idea and several propositions are developed. RABAC (Role-centric Attribute- Based Access Control) [84] is the first formal hybrid model which proposes an assignment of roles avoiding role-explosion problem. RABAC is an extension of RBAC with permission filtering policy (PFP) which constrains the available set of permissions based on user and object attributes by using Boolean expression (function). According to [11], the RABAC approach does not incorporate environment attributes and so that it is not suitable for systems involving frequently changing attributes. The authors in [11] combine RBAC and ABAC in one new model AERBAC (Attributes Enhanced Role-Based Access Control), by using contextual information and exploiting the contents of the resources to provide fine-grained access control mechanism. Several works as spatio-temporal RBAC [85] and context-aware RBAC [47] focus on the merge of access context in RBAC. However, these models suffer from the role-explosion problem (a big number of roles). To deal with this problem, a new spatio-temporal RBAC [86] model was proposed by introducing the concept of spatio-temporal zones to abstract location and time into one single entity. In this last model, using zones prevents the creation of new roles when spatio-temporal constraints associated with them change.

III.3 A new hybrid access control model

In this section, we present our proposed model which is a hybrid model based on both models RBAC and ABAC. The proposed model integrates the multiple accesses as well as the unique access. Before presenting the new proposed model, we start by listing a set of requirements which must be fulfilled by a suitable access control model.

III.3.1 Requirements for a suitable access control model

To deal optimally with security policies, an access control model is expected to guarantee the following needs.

- Reduce the complexity of the security policy. This requires the reduction of two metrics: *Written Permissions Number (WPN)* and the *evaluated permissions number (EPN)*. The WPN is the total number of the written permissions, by the administrator, to define what a user or a group can or can not do. The EPN is the number of permissions which will be evaluated, by the system, to decide that a user has not

the requested permission. In fact, reducing WPN leads to reduce the EPN which makes the auditing in the model easier.

- Use a suitable format of rules or permissions allowing to express the complex granularity of systems without any explosion.
- Use a suitable format of rules or permissions allowing to express the access to private objects.

To achieve the above requirements, we need to consider the following basic ideas in the new proposed model.

- Use the Role concept; thus, divide the users according to their functions.
- Each role has a set of permissions which are expressed in rules.
- In each rule, we express the object, user, and environment features.
- Divide the set of rules according to the access actions. Because there is one rule for each access actions, in each role, the decision, if a user has not the requested permission, needs to evaluate just one rule in each active role of this user. Hence, the number EPN equals to number of Active Roles (AR): $EPN = AR$.

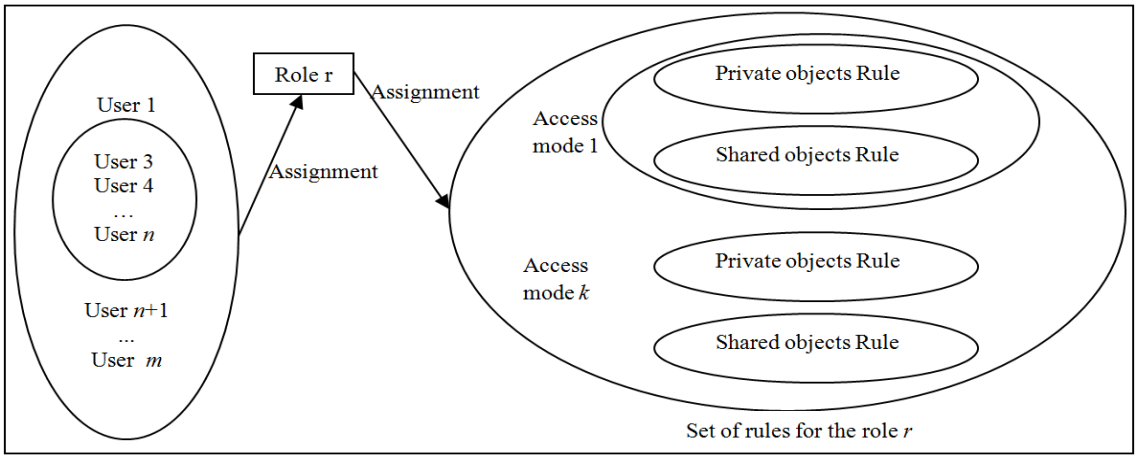
A model, which considers the above basic requirements, will be able to overcome all access control models requirements and will provide the advantages of ABAC and RBAC. In the next section, we establish a new model in order to meet these required features.

III.3.2 Principle of the proposed model

To benefit from RBAC, we define a set of roles and each role has its permissions, but rather than considering permissions as a set of permissions referring to individual objects and to one instance of the access action, we divide the permissions assigned to a role according to its access actions. In the RBAC model, if the designer wants to express the fact that a role "admin" can read papers and mails then he must define a set of permissions assigned to this role as: (admin, read, paper1), ... , (admin, read, papern), (admin, read, mail1), ... , (admin, read, mailm). However, in our proposed model, we propose to define an assignment of permissions as: (admin, read, papers and mails). This last assignment is used to express that all objects which the admin can read are: papers and mails; hence we collect the objects into sets, according to the access type (read, write, etc) by the role. In each set, we separate the objects dedicated to a particular user (i.e. unique access) from the objects dedicated to multi users (i.e. multi-access) into two subsets. To benefit from ABAC advantages, identification of permissions takes into consideration the different attributes of objects, subjects (users) and the environment. By exploiting the attributes of objects, users, environment, and the concept of ABAC rule, we assume that our model overcomes the following problems which face other models: (i) the role permissions explosion, (ii) roles explosion, and (iii) the exponential augmentation of the groups when the number of object categories increases. The use of private objects and shared objects to integrate the multi access and the unique access, in the same model, makes the model more "realistic|flexible" and respects the concept of the role in RBAC as well as the concept of attribute in ABAC (i.e. to restrict the access of each user only to his data, we do

not need to define private role or specific rule to each user). The assignment of a role to a set of users allows the designer controlling the separation of duties, list of privileges, confidentiality and to use the principle of role inheritance. FigureIII.1 depicts the principle of the proposed model. The constraints and the mechanisms required by the model will be presented in the next section.

Figure III.1: Principle of the proposed model



The figure III.1 illustrates that the permissions of users are assigned to them according to their roles (the same principle as RBAC). Each role has a set of rules defining its permissions (read an object, write the object, delete an object, execute an object, etc.). This set of rules is constructed by defining two rules for each access type (read access rules, delete access rules, etc.). The first type of rules concerns the access to shared objects and the second type of rules concerns the access to private objects. This separation between rules makes the model more efficient. Instead of checking the user query using a lot of rules, the model checks the user query using only one rule. Indeed, the syntax of rules used in this model allows the designer to divide the permission according to the access type for both object types and takes into consideration the different attributes of objects, subjects (users) and environment.

III.3.3 Collaborative Cloud Services case in the proposed model

Collaborative Cloud Services is used to achieve collaboration projects or services in a cloud environment, where there is a need to share some resources between services. The sharing of resources must be secure in this dynamic distributed environment. We distinguish between two cases of collaboration and we illustrate the principle of our policy in these two cases.

III.3.3.1 Enterprises Collaboration

To finish some tasks in a collaboration project, certain employees of an enterprise need access to some resources of another enterprise. The data owner of this last must give the required accesses to the other enterprise employees. The owner gives the accesses according to the role of employees, through his access policy. Hence, there is heterogeneity of access policies and conflicting policies in this distributed environment. Even if the other

enterprise has the same access policy model, the owner can not have trust in the other's access restrictions.

For more explanation, we use collaborative cloud services example illustrated in figure 2, where there are three issuers using the same cloud storage service. These issuers cooperate with each other to accomplish some tasks. The three issuers are the university (UN), the Auditing company (AC) and the last one is the Insurance Company (IC). The auditors in AC must be able to read-only financial reports of the (UN) (Stored in UN-Base). Insurance staffs (in IC) can only execute the search function to check if the client is a member of the UN or not.

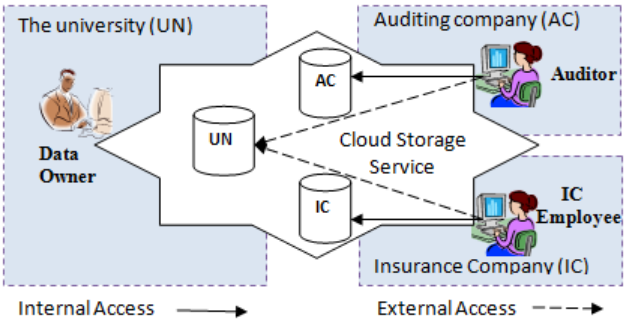


Figure III.2: Collaborative cloud services example.

The data owner of UN cannot have trust in AC "admin" to define the auditor permissions. Moreover, the admin of UN may give the auditor less permission than what he needs. Hence, the main issue is how to merge access control policies of collaborating organisations without any conflicts with their local policies? To address this last problem, we propose the following.

- **Role Permission Agreement**: it is a contract between the data owner and the other enterprise, which defines formally the following elements.
  - The set of roles and their permissions,
  - The constraints of each role access restriction (assignment constraints, activation constraints and permissions constraints),
  - The set of enabled users to play this role.

  This agreement is dynamic, where each element is elastic.

According to this agreement the data owner:

- Creates the roles and their permissions, then he adds the mentioned roles to the set of roles,
- Add the mentioned users to the set of users,
- Restricts the roles assignment, roles activation and roles permissions.

Now, whatever happens on the second side (internal/external attacks), the owner of data trusts that only the mentioned users in RPA can access its data through their mentioned roles. On the other side, the other organisation trusts that its employees have the required permissions to do their tasks. The RPA contract is built by both organizations and it defines clearly and safely the interactions between them. Hence, the organizations collaborate through inter-organisational work-flow which is free of conflict.

III.3.3.2 Enterprise and service collaboration

Collaboration with the services case is less complex than in the case of enterprises. It is enough that the data owner defines the role of the service and restricts its accesses. Thus, the service is considered a usual user.

III.3.4 The security policy under the proposed model

A policy is a set of rules that define the behaviour of a system. The system that uses this policy is expected to satisfy this set of rules in all its states. In this section, we present the security policy under the proposed model. This policy requires determining sets, functions, rules and constraints. In our proposed model, we have extended basic sets and functions that are defined in ABAC model.

III.3.4.1 The Sets

We distinguish five entities in the system: user, object, role, rule, and permission. Each of these entities yields to a specific set in the policy. Thus, the policy defines the following sets.

- S: denotes the set of subjects (users) that can manipulate or access the resources or objects in this system.
- O: denotes the set of objects. Both subjects and objects have their unique identifier u_{id} and r_{id} respectively.
- At: denotes users, resources and the environment attributes. The attribute can have a single value (*atomic value*) or multi-atomic value. The set At is composed of several subsets denoted Au_i, Ar_j and Ae. Au_i is the set of $user_i$ attributes, Ar_j is the set of $resource_j$ attributes, and Ae is the set of environment attributes (such as time and location). In the set Ar_j, we use the attribute ***Refer_To*** to define the owner of the $resource_j$. The attribute ***Refer_To*** contains the value u_{id} if the $resource_j$ belongs to the user (u_{id}) otherwise it contains "null".
- R: denotes the set of roles. The users interact with the system according to their roles.
- Ac: denotes the set of access action (i.e. read, write, view, control, etc.).
- P: denotes the set of permissions.
- RL: denotes the set of rules which assign permissions to each role.

III.3.4.2 The Functions

Three functions are introduced as follows. (i) $Vu(u,\ a)$: returns the value of the attribute a of the user ***u***, otherwise it returns *null* if the user has not this attribute; (ii) $Vr(rs,\ a)$: returns the value of attribute ***a*** for a resource ***rs***, otherwise it returns *null* if the resource has not this attribute; (iii) $Dr(u)$: returns the set of rules dedicated to a user ***u*** according to his active role.

III.3.4.3 The rules

We assume that the set of rules "RL" consists of all role rules subsets RL_r, where $r \in R$ (R is the set of roles). For each role r, the set RL_r is composed of subsets $RL_{r_{acc}}$, such that $acc \in A$ (A is the set of access actions). Lets consider R_m and R_{un} the two sets defined as follows.

- R_m: represents the set of rules which bind to the users the access acc to the multiple access objects.
- R_{un}: represents the set of rules which bind to the users the access acc to the unique access objects.

Each set $RL_{r_{acc}}$ includes one rule from the set R_m and one rule from the set R_{un}.

We use the tuple (acc, rs), containing a resource *rs* and an access mode *acc*, to express that a user has the appropriate permission to perform the action *acc* on the resource *rs*. The set UP_u of tuples (acc, rs) denotes all the permissions assigned to the user *u*.

Finally, a rule is a tuple (t, r, acc, cst), such that:

- t: is the type of the rule (unique or multiple). The value of t can be R_{un} or R_m.
- $r \in R$. r is a role.
- acc: access mode, $acc \in A$;
- cst: is a constraint. The constraint cst is a logical formula built upon the two functions $Vu(u, att)$ and $Vr(rs, att)$, such that:
  - $Vu(u, att)$ gives the attribute value of att for the user u.
  - $Vr(rs, att)$ gives the attribute value of att for the resource rs.

  The constraint cst can be written according to the following grammar.

  $cst := true$
  $cst ::= cst\ and \mid or\ Vu(u, att) = Vr(rs, att)$
  $cst ::= cst\ and \mid or\ Vu(u, att1) = Vr(rs, att2)$
  $cst ::= cst\ and \mid or\ Vu(u, att) = const$
  $cst ::= cst\ and \mid or\ Vr(rs, att) = const$
  $cst ::= cst\ and \mid or\ Vu(u, att) \supseteq Vr(rs, att)$

  such that u is a user, rs is a resource, att, $att1$ and $att2$ are attributes, and $const$ is a constant value of an attribute. Besides the above elements, a constraint may include another statements like the time or location (environment attributes). Moreover, if we have tow rules that have the same constraint cst and role r with two different access actions acc_1 and acc_1 then we write them in one rule as: $(t, r, acc_1 or acc_2, cst)$.

III.3.4.4 Constraints in the proposed model

We use the same constraints as defined in RBAC, which are: (i) Static and dynamic separation of duties (SoD), (ii) Role hierarchy, (iii) the cardinality of roles, (iv) role authorization and (v) role execution. After assigning a role to a user, we check the following elements.

- **The role authorization**: is the role authorized for this user?
- **The static separation of duties**: this role is not already assigned to some users who have static conflicts with the current user? this role is not in static conflicts with the already assigned roles to the current user?
- **The cardinality of role**: is the number of users assigned to this role less than the cardinality of the role?

After the verification of these constraints, we add the name of the role to the multi atomic values of the user's "Role-attribute".

- **Dynamic separation of duties**: we use multi atomic value attribute "Active" to express the activated roles by the user. To insert the name of a role into the "Active attribute", we should verify that the role and the user are not in dynamic conflicts. This concerns two cases: (i) this role cannot be activated by the current user because it is already activated by another user who has conflict with the current user, or (ii) this role cannot be activated by the current user because the current user activated already another role which is in conflict with the current role.
- **Role hierarchy**: to give the user a permission, we use the rules of the active role (role execution). If the role r activated by a user contains another role r' (r is a senior role of r') then this user will own the permissions assigned to r' too.

III.3.4.5 The mechanism

In this section, we present the mechanism of the access decision. It is to decide if a user u requesting access to a resource rs, through the access mode acc, is authorized or not to access rs. The mechanism is implemented by two algorithms. The first algorithm evaluates the access query of a user to a specific resource. The second algorithm evaluates the access query of a user to a set of resources sharing the same attributes (for example, three distinguished resources: "text document", "video", and "audio" which concerns all the same resource). In the second algorithm (which is not presented in the chapter), the request is denied if the user does not activate the role (role execution). The algorithm decides if the permission is granted or denied by evaluating the specific rule which is composed of the three following elements.

- The active role (role execution) or its juniors (Role hierarchy).
- The requested access mode.
- The type of the requested objects (unique or multiple).

The figure III.3 summarizes the mechanism of the access decision implemented in the algorithm 1 (i.e., when the user requires access to a specific resource).

In the algorithm of multi-access, the request is denied if the user does not activate the role (role execution). The algorithm decides if the permission is granted or denied, by evaluating the tow rules (unique and multiple) which are composed of the following two elements.

- The active role (role execution) or its juniors (Role hierarchy).

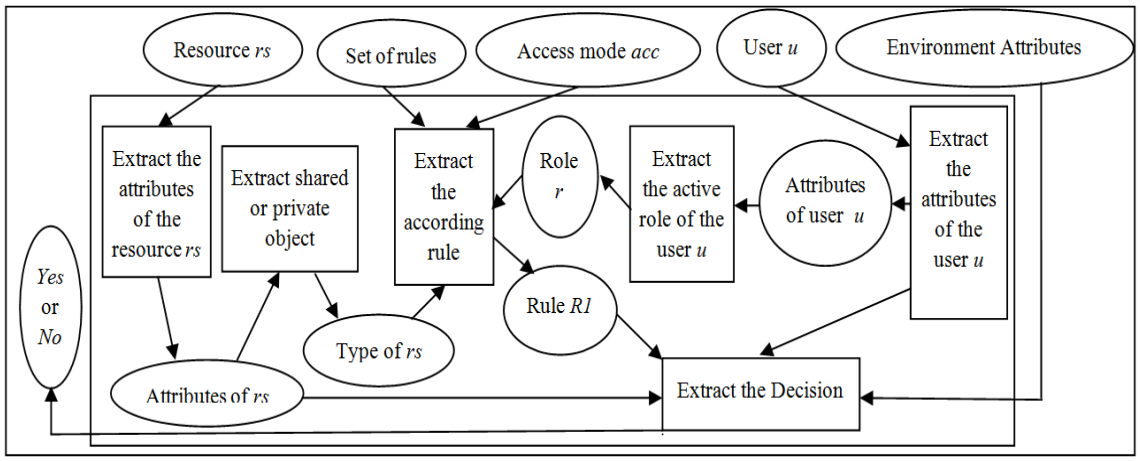
Algorithm 1 algo:1

Input: Access query ($Rq < u_{id}, acc, r_{id} >$) consisting of user identifier u_{id}, access mode acc and resource identifier r_{id} ;

Output: Access; // Access = grant if the user has the permission else Access = deny;

List_Active_Role $\leftarrow \emptyset$ // List of the user's currents activated roles.
Access $\leftarrow$ deny // The access is denied until we find that the user has the permission.
User_attributes $\leftarrow$ *get_attributes(*u_{id}*)* // Gets all user's attributes.
Active_Roles $\leftarrow$ *getvalue(User_attributes, Active)* // From the user attributes, we get the value of the attribute Active, which contains the user's currents activated roles.
if *Active_Roles* $= null$ **then**
  return *(Request denied: you not have active role)*
else
  Resource_attributes $\leftarrow$ *get_attributes(*r_{id}*);* //Return the attributes set of the object r_{id}
  Type $\leftarrow$ *getvalue(Resource_attributes, Refer_To);* //Returns the value of the attribute Refer_To **if** *Type* $= null$ **then**
    Type $\leftarrow$ *shared;*
  else
    Type $\leftarrow$ *unique;*
  $List\_Active\_Role.add(Active\_Roles)$;
  List_junior_roles $\leftarrow$ *get_junior_roles(Active_Role);* //Returns juniors of all active roles. $List\_Active\_Role.add(List\_junior\_roles)$;
  Environment_attributes $\leftarrow$ *get_att_Environment();* //Returns the set of Environment attributes.
  while $(List\_Active\_Role \neq \emptyset) \wedge (Access = deny)$ **do**
    Rule $\leftarrow$ *get_Rule(Role,Type,acc);* // Returns the rule dedicated to restrict the access (acc) of the role (Role) to (type) objects.
    Access $\leftarrow$ *evaluate(Rule, Environment_attributes, Resource_attributes, User_attributes);*//Matches the query with the rule and returns the result.
  if *Access* $= deny$ **then**
    return (Request denied);
  else
    return (Request granted);

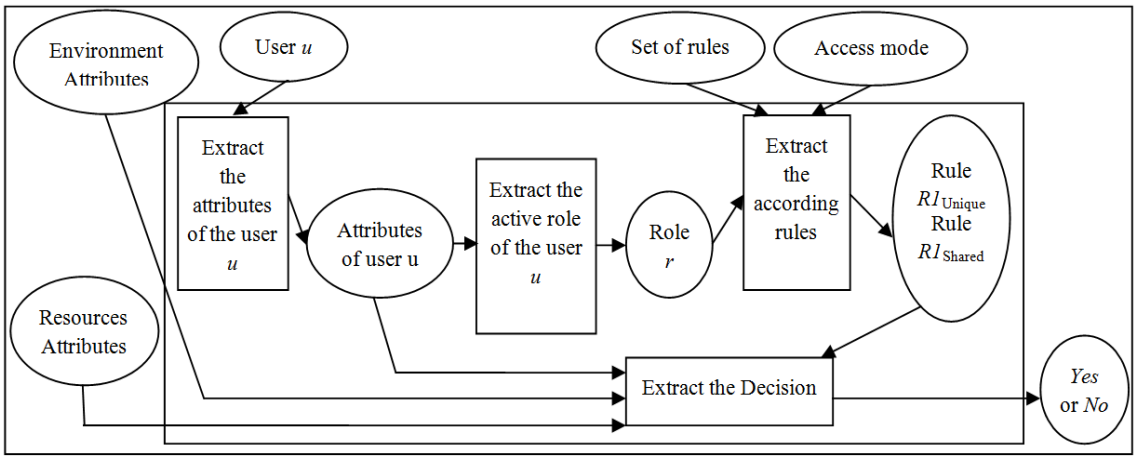
Figure III.3: Mechanism of access decision when the user requires access to a specific resource



- The requested access mode.

The figure III.4 summarizes the mechanism of the access decision when the user requires access to resources sharing the same attributes.

Figure III.4: Mechanism of the access decision for multiple resources sharing the same attributes.



III.4 Evaluation of the new proposed model: empirical comparison with existing models

In order to demonstrate the suitability of the new proposed model, this section provides an empirical comparison approach between this new proposed model and three existing models (RBAC, ABAC and the hybrid model AERBAC).

III.4.1 Metrics used in the empirical comparison approach

The proposed comparison method is based on the following set of metrics, inspired from [11].

- *Written Permissions Number (WPN)*: the total number of the written permissions, by the administrator, to define what a user or group of users can or can not do.
- *Evaluated Permissions Number (EPN)*: it is the number of permissions which will be evaluated, by the system, to decide that the user has not the requested permission.
- *Policy Modification Visualization (PMV)*: it measures how it is hard or easy to visualize the consequences of the policy modification in the access control model.
- *Context-aware Access (CaA)*: it measures if the access control model can handle the dynamic changing of attributes or not.

These four metrics will be evaluated through the policies defined under four models (the new proposed model, RBAC, ABAC and AERBAC), to show how the new proposed model is more suitable than the existing ones. The new proposed model is proved to be more able to handle efficiently security in a complicated situation where the number of entities in each set (i.e., users, roles, objects and attributes) increases highly.

III.4.2 The illustrative example

To make the comparative method easier to understand, we use the following example. In a college, students pass through three levels to get their graduate diploma. In the second level (L_2), students are divided into x specialities ($\{S_i\}_{i\in 1\ldots x}$). In the third level (L_3), students are divided into y specialities ($\{S_i\}_{i\in 1\ldots y}$). To manage students' access, the system encloses two kinds of objects: *shared* and *private*. Shared objects are dedicated to a set of users and private objects are dedicated to one user. Two "access actions" are proposed which are *read* and *download*.

The access to shared objects is managed using the following rules: (i) A student in L_1 can access to all courses of his level, (ii) A student in L_2 or L_3 can access only the courses of his speciality, (iii) Only premium users have access to *paid courses*, and (iv) Regular users have access to paid courses only during *promotional periods*. The access to private objects concerns the access to marks (i.e., marks are accessible only by the concerned student).

Resources of the system include a set of courses defined in each level L_i, for $i = 1, 2, 3$. These courses are of two kinds, regular courses and paid courses, denoted respectively as RC_{L_i} and PC_{L_i}. In levels L_2 and L_3, courses are divided into specialities. Courses of the speciality S_i for $i = 1 \ldots k$ in levels L_2 and L_3 are denoted as $RCL2_{S_i}$, $PCL2_{S_i}$, $RCL3_{S_i}$, $PCL3_{S_i}$, respectively.

Using the previous example, the following sections present a comparative evaluation between the new proposed model and three existing models which are RBAC, ABAC and hybrid model AERBAC. The policy is evaluated under each model to show the advantages of the new proposed model vs the three existing ones.

III.4.3 RBAC configuration evaluation

The RBAC policy, for the illustrative example, is defined as a set of *roles* and *access permissions*, as follows.

- *Roles*: In a regular users, $1 + x + y$ roles are required to express the conditions of levels and specialities. These roles can be denoted as: R_1 for regular students in L_1, $\{R_i\}_{i \in 2...x+1}$ for regular students in L_2, and $\{R_i\}_{i \in x+2...x+1+y}$ for regular students in L_3.
- To express the conditions of premium users, the administrator creates for each regular role a premium role. Hence, the number of roles will be $(1 + x + y) * 2$ roles.
- To express the conditions of promotional periods, the administrator creates for each regular role a promotional role. Hence, the number of roles will be $(1 + x + y) * 3$ roles. A promotional role would be available to users only during promotional periods and it inherits the premium role permissions.
- Access permissions to read regular courses: we need respectively $|RC_{L_1}|$, $|RCL2_{S_1}|$, ... $|RCL2_{S_x}|$, $|RCL3_{S_1}|$, ..., $|RCL3_{S_y}|$ permissions for roles R_1, R_2, ... R_x, R_{x+1}, ..., R_{x+1+y}. Each permission has the form (R_i, read, $\{C_j\}_j$), such that R_i is a role and $\{C_j\}_j$ is the set of regular courses accessed by role R_i.
- Access permissions to download regular courses: it is the same as for reading's permissions. However, a download permission has the form (R_i, download, $\{C_j\}_j$), such that R_i is a role and $\{C_j\}_j$ is the set of regular courses accessed by role R_i.
- Access permissions to read paid courses: we need respectively $|PC_{L_1}|$, $|PCL2_{S_1}|$,...$|PCL2_{S_x}|$, $|PCL3_{S_1}|$,..., $|PCL3_{S_y}|$ permissions for roles R_{x+2+y}, R_{x+3+y}, ... $R_2 * x + 1 + y$, $R_{2*x+2+y}$... $R_{2*(x+1+y)}$. Each permission has the form (R_i, read, $\{C_j\}_j$), such that R_i is a role and $\{C_j\}_j$ is the set of paid courses accessed by role R_i.
- Access permissions to download paid courses: The same as for read permission of paid courses. However, a permission has the form (R_i, download, $\{C_j\}_j$), such that R_i is a role and $\{C_j\}_j$ is the set of paid courses accessed by R_i.
- The RBAC does not support access to private objects because this kind of access requires the definition of a new role for each user (i.e., which makes the role concept without benefits).

In the following, we analyse each of the four metrics (WPN, EPN, PMV, CaA) separately.

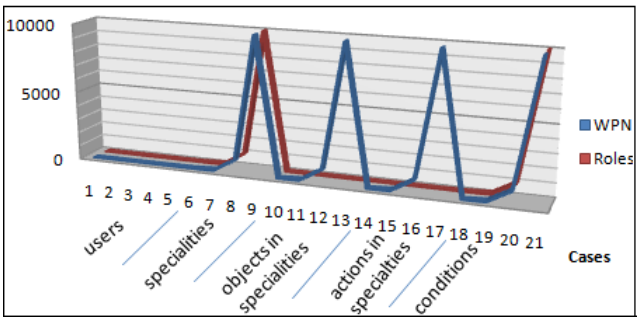
1. *WPN metric*: in RBAC, the configuration that grants permissions to roles is written in the form of direct permissions. Each permission contains an access action and the identifier of an object. We assume that R_i denotes the role identifier, where $1 \leq i \leq N/N$ is the number of roles (in the illustrative example N equals to $3 * (1 + x + y)$). The variable PN_{R_i} denotes the permissions number of the role and it is computed as: $PN_{R_i} = \sum_{acc=1}^{j} NOb_{acc}$, such that j is the number of access actions belonging to the role R_i and NOb_{acc} is the number of accessible objects by the role R_i through the access action acc.

In the illustrative example, the role R_1 has two access actions (i.e., read ($acc = 1$) and download ($acc = 2$)) and there is RL_1 courses which can be read or downloaded by role R_1. Hence, $j = 2$ and $NOb_1 = RL_1$. The permissions number PN_{R_1} for role R_1 is given as: $PN_{R_1} = \sum_{acc=1}^{2} NOb_{acc} = NOb_1 + NOb_2 = RL_1 + RL_1$. The total number of written permissions WPN is equal to the PN_{R_i} sum; hence, $WPN = \sum_{i=1}^{N} PN_{R_i}$. In fact, we will have: $WPN = 2 * RL1 + x * (2 * RL2) + y * (2 * RL3)$. To simplify the analysis, we suppose that the number of permissions is the same for all roles, hence WPN is computed using the equation III.1.

$$WPN = N * PN_{R_i} \tag{III.1}$$

We study the metric WPN and the number of roles according to five parameters: (i) the number of users, (ii) the number of specialities, (iii) the number of objects in each specialities, (iv) the number of actions in each specialities, (v) and the number of conditions. In total, we have 21 cases. In the first case, all parameters have the value 1. In each case from case 2 to case 21, four parameters are fixed to the value 1 and the fifth parameter is successively affected to the values 10, 100, 1000 and 10000. For example, case 2 affects to the number of users 10, case 3 affects to the number of users 100, case 4 affects to the number of users 1000, case 5 affects to the number of users 10000 (as depicted in III.5). The same thing is done for the four other parameters. The value of WPN and the number of required roles in each case are plotted in Figure III.1.

Figure III.5: Experimental Results in RBAC



From Figure III.5, we find that: (i) the number of users has no effect on the WPN (case 2, .., case 5), (ii) There is a WPN explosion on the systems that have a large number of objects and complex granularity (Specialities, Actions) (case 7, .., case 21). Hence, RBAC has a lack of expressiveness and does not provide fine-grained access control, (iii) and finally, the roles number increases according to the number of specialities (case 6, ..., case 9).

2. *EPN metric*: To decide that a user has not the requested permission, the RBAC evaluates all the permissions of this user's active roles (i.e., the set AR). So that, the EPN is calculated using equation III.2.

$$EPN = \sum_{i=1}^{AR} PNR_i \tag{III.2}$$

Cases	Users	Specialities	Objects in each Sp	Actions in each Sp	Conditions
1	1	1	1	1	1
2	10	1	1	1	1
3	100	1	1	1	1
4	1000	1	1	1	1
5	10000	1	1	1	1
6	1	10	1	1	1
7	1	100	1	1	1
8	1	1000	1	1	1
9	1	10000	1	1	1
10	1	1	10	1	1
11	1	1	100	1	1
12	1	1	1000	1	1
13	1	1	10000	1	1
14	1	1	1	10	1
15	1	1	1	100	1
16	1	1	1	1000	1
17	1	1	1	10000	1
18	1	1	1	1	10
19	1	1	1	1	100
20	1	1	1	1	1000
21	1	1	1	1	10000

Table III.1: The input parameters values for the evaluation of RBAC

Usually, a user is assigned to a small number of roles. This latter means that EPN is not a very big number. In fact, this is correct only if RBAC is used in systems without complex granularity. The previous metric demonstrates that RBAC is not suitable for fine grained systems. The EPN in RBAC indicates that RBAC has not complex auditing.

3. *"Policy modification visualisation" metric*: The policy is written at the role level; hence, it is easy to visualize the consequences of policy modification. If the administrator adds a permission to a role then all users assigned to this role will have the permission, automatically.

4. *"Context-aware access" metric*: The explosion of WPN in fine grained systems is due to the fact that RBAC does not use the attributes. Models that do not use attributes do not support the context-aware access as the case of RBAC.

III.4.4 ABAC configuration evaluation

The policy in ABAC is defined as a set of *rules*. According to [87], a rule is a tuple (eu, er, O, c) such that eu is a user attribute expression, er is a resource-attribute expression, O is a set of operations and c is a constraint. Therefore, in te case of the illustrative example, ABAC needs to define $2 * (x + 1 + y)$ rules. These rules are required to express the conditions on levels, specialities and access permission, as follows.

- $Rule_1$= ($true$, Role=student $\wedge$ Level=L_1, read or download, type=courses $\wedge$ level=L_1),
- $Rule_2$=($true$, Role=student $\wedge$ Level=L_2 $\wedge$ S = 1, read or download, type =courses $\wedge$ level=L_2 $\wedge$ Speciality=S_1),
  ...
- $Rule_{x+1}$=(Role=student $\wedge$ Level=L_2 $\wedge$ S=x, read or download, type=courses $\wedge$ Level=L_2 $\wedge$ Speciality=S_x),
- $Rule_{x+1+1}$=(Role=student $\wedge$ Level=L_3 $\wedge$ S=1, read or download, type=courses $\wedge$ Level= L_3 $\wedge$ Speciality=S_1),
  ...
- $Rule_{x+1+y}$ =(Role=student $\wedge$ Level=L_3 $\wedge$ S=y, read or download, type=courses $\wedge$ Level= L_3 $\wedge$ Speciality=S_y),
- $Rule_{x+1+y+1}$=($true$, Role=student $\wedge$ Level=L_1 $\wedge$Type= premium$\vee$ today $\in$ PromoDates, read or download, type=PaidCourses $\wedge$ level=L_1),
  ...
- $Rule_{2*(x+1+y)}$=(Role=student $\wedge$ Level=L_3 $\wedge$ Sp=y Type= premium $\vee$ today $\in$ PromoDates, read or download, type=PaidCourses $\wedge$ Level=L_3 $\wedge$ Speciality=S_y).

To access to a private object, the "administrator" should write a rule for each user allowing him a unique access to that object. Therefore, if we have 1000 students then the "administrator" needs to rewrite 1000 times the rule ($true$, Title = u_id, read or download, Type = Result and ReferTo = u_id), such that u_id is the identifier of a user.

In this case, the number of *rules* in ABAC is less than the number of *roles* in RBAC. We are interested to explain this decrease in the number of rules. In the following paragraphs, we study the four metrics (WPN, EPN, PMV, CaA) for ABAC model.

1. *WPN metric*: In ABAC, the configuration that grants permissions to users is written in the form of rules. The number of rules depends on the number of object groups (objects of the same group have the same attributes). As discussed above, there are $2 * (x + 1 + y)$ object sets each of which contains courses of the same type, level and speciality. We denote by NOG the number of object groups, hence the total number of the written permission WPN will be equal to NOG: $WPN = NOG$.

   We will evaluate the WPN according to six parameters: (i) the number of users, (ii) the number of specialities, (iii) the number of objects in each specialities, (iv) the number of actions in each specialities, (v) the number of object conditions, and (vi) the number of environment conditions. We define 25 cases which are similar to the previous section of RBAC. In the first case, all parameters have the value 1. In the other cases, four parameters are fixed to the value 1 and the fifth parameter is successively affected to the values 10, 100, 1000 and 10000. The metric WPN is evaluated in two cases, *with* and *without* private access. The Figure III.6 plots the values of WPN in the two cases.

Figure III.6: Experimental Results in ABAC

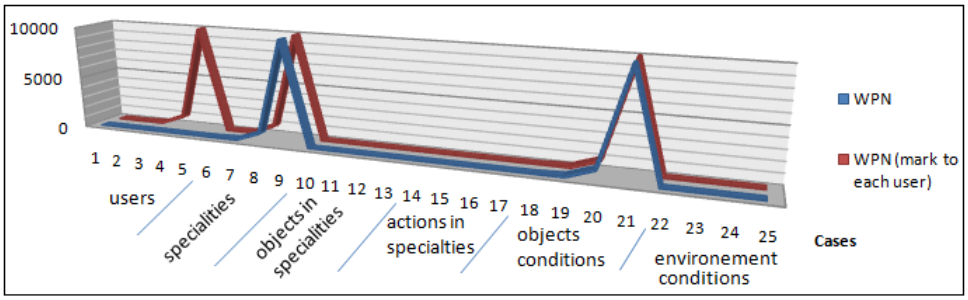


   Figure III.6 shows that: (i) WPN (in the two cases, with or without private objects) increases when the number of *specialities* or the number of *object conditions* increase (Case 9 and case 21). This is justified by the augmentation of NOG required to model the features of objects, (ii) WPN with private objects increases when the *users* number increases (case 5), (iii) finally, the two factors objects number and access actions number have no impact on WPN. Hence, ABAC provides fine-grained access but it still needs some adjustments to support the private objects and to be more suitable for fine_grained systems.

2. *EPN metric*: to decide that the user has not the requested permission, the ABAC will evaluate all the rules with an exhaustive enumeration of attributes, used in each policy rule that we denoted by A_i. Hence, the number EPN is calculated using the equation III.3.

$$EPN = \sum_{i=1}^{WPN} A_i \tag{III.3}$$

   Usually, EPN is a very big number which means that ABAC has a complex auditing.

3. *"Policy modification visualization" metric*: in ABAC, it is hard to visualise the consequences of policy modification. If the administrator changes a rule then he will not be able to know all the consequences.

4. *"Context-aware access" metric*: Unlike RBAC, ABAC supports the context-aware access, thanks to the use of the attributes.

III.4.5 AERBAC configuration evaluation

According to [11], the AERBAC policy is defined as a set of roles. Applied to the illustrative example, we require $x + 1 + y$ roles each of which has two access permissions: one with conditions and another without conditions (as described in Table III.2). AERBAC does not support the access to private objects (e.g. marks in courses).

Table III.2: AERBAC configuration

Role	**Permissions**	**Conditions**
2*R_1**:Stud**_L_1	(read, (Type(o)=Paid_Course $\wedge$ L(o)=1))	Type(u)=premium $\vee$ today $\in$ PromoDates
	(read, (Type(o)=Course $\wedge$ L(o)=1))	none
.....	.....	.....
2*R_{1+x+y}**:Stud**_L_3_S_y	(read, (Type(o)=Paid_Course $\wedge$ L(o)=3 $\wedge$ S(o)=y,))	Type(u)=premium $\vee$ today $\in$ PromoDates
	(read, (Type(o)=Course $\wedge$ L(o)=3 $\wedge$ S(o)=y))	none

The four metrics are evaluated in the following.

1. *WPN*: The AERBAC creates N roles and each role R_i has a Permission Number PN_{R_i} as in the RBAC case. Hence, the total number of written permissions WPN is equal to the sum of all PN_{R_i}, for $i = 1 \ldots N$. So that, WPN is computed as: $WPN = \sum_{i=1}^{N} PN_{R_i}$.

   Figure III.7 plots WPN depending on the same set of parameters used in the case of ABAC. Figure III.7 shows the following.

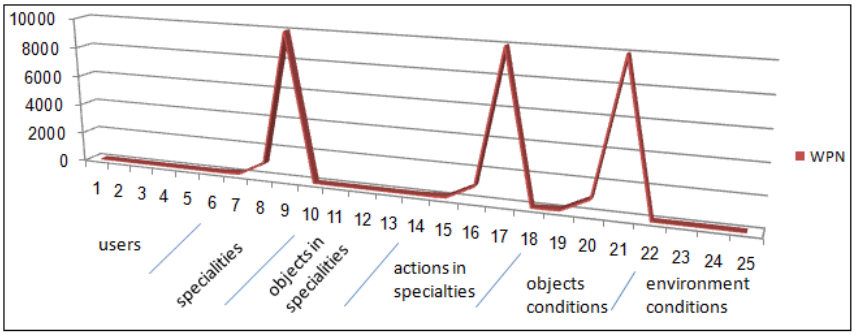
   - The number of users, the number of objects and the number of environment's conditions have no impact on WPN (as in the case of ABAC).
   - The number of access actions has an impact on WPN (as in the case of RBAC).
   - The number of specialities and the number of object conditions have an impact on WPN (as in the case of ABAC).

   As a conclusion, AERBAC provides fine grained access but it still needs some adjustments to support private objects and be more suitable in fine grained systems.

2. *EPN and "policy modification visualization" metrics*: These two metrics are similar to the case of RBAC,

3. *Context-aware access metric*: it is similar to the case of ABAC.

Figure III.7: Experimental Results in AERBAC



III.4.6 Evaluation of the new proposed model

In the new proposed model FRABAC, the policy will be defined as a set of roles and a set of access permission rules, as follows.

- *Roles*: Actually, the student at all levels has always the same role which is called "*student*". So that, unlike RBAC case, the new proposed model requires only one role to express the role "*student*".
- *Rules*: According to the format of rules defined in section III.3.4, only two rules are required to express all access conditions in the illustrative example. We define *Rule_1* to *read* or to *download* the shared objects(free courses and paid courses) and *Rule_2*to *read* or *download* the private objects (i.e., marks of a course). Let's denote by PC paid courses, C regular courses and T the type of users (which can be premium or normal) or the type of objects (which can be a mark or a course). Using the previous notations, the model requires only the following two rules to define the policy in the example.
  - $Rule\_1$:$\{true, student, [T(o) = C \vee T(o) = PC \wedge (T(u) = premium \vee today \in PromoDates) \wedge L(u) = L(o) \wedge S(u) = S(o)], readordownland\}$
  - $Rule\_2$: $\{true, student, T(o) = Note \wedge Vr(o, Refer\_to) = Vu(u, id), read$ or $downland\}$

To demonstrate the efficiency of the new proposed model, we analyse the four metrics in the following paragraphs.

1. *WPN metric*: in the proposed model, the configuration that grants permissions to roles is written in the form of role rules. The number of role rules depends on the number of access actions (in the example, there are 2 access actions). Hence, the WPN is equal to the sum of all PN_{R_i}, as presented in equation III.4.

$$WPN = \sum_{i=1}^{N} PN_{R_i} = TRu. \tag{III.4}$$

In equation III.4, we denote by TRu the total written rules number which represents the WPN metric in the comparative method. PN_{R_i} indicates the rules number of the role and it is computed using equation III.5.

$$PN_{R_i} = \sum_{acc=1}^{NAC_{R_i}} NT_{Act_{acc}}. \quad \text{(III.5)}$$

In equation III.5, NAC_{R_i} is the number of access actions belonging to the role R_i. When computing NAC_{R_i}, those access actions which have the same set of accessible objects with the same access conditions are considered as *one action*. For example, the role *student* has two access actions (read and download) which have the same access conditions and the same objects; hence, these two actions are considered as one action when computing PN_{R_i}. $NT_{Act_{acc}}$ indicates the number of "access action types" (this number can be either 1 or 2). In the example, the access action (read/download) has two types (which are: shared and private).

Using the illustrative example, we will have the following.

$$PN_{R_1} = \sum_{acc=1}^{1} NT_{Ac_{acc}} = 2. \quad \text{(III.6)}$$

$$WPN = \sum_{i=1}^{1} PN_{R_i} = PN_{R_1} = 2. \quad \text{(III.7)}$$

WPN is computed depending on the same set of parameters used in the evaluation of ABAC and AERBAC. To simplify the analysis, we suppose that the PN_{R_i} of all roles is the same, hence we will have: $WPN = N * PN_{R_i}$. We distinguish between two policy cases, case 1 (which is the middle case) and case 2 (which is the worst case).

Case 1:

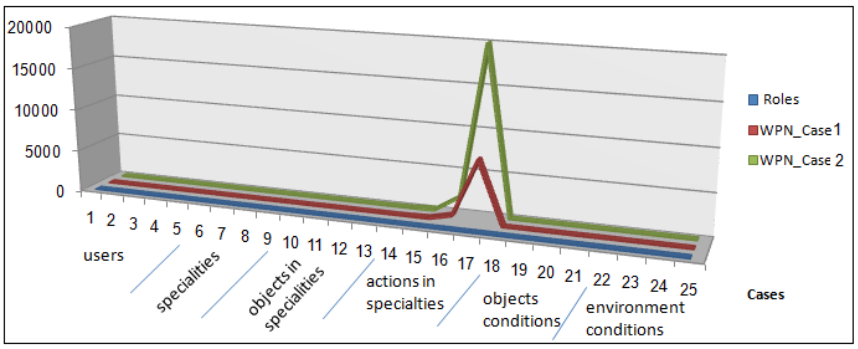
In this case, we propose that: (i) 50% of access actions have the same objects and conditions sets, (ii) and 50% of access actions have the two access types and 50% of access actions have just shared access. The equation that calculates the PN_{R_i} will be as follows.

$$PN_{R_i} = \sum_{acc=1}^{(NAC_{R_i}/2)+1} NT_{Ac_{acc}} = \sum_{acc=1}^{[(NAC_{R_i}/2)+1]/2} 2 + \sum_{acc=[(NAC_{R_i}/2)+1]/2+1}^{[(NAC_{R_i}/2)+1]} 1. \quad \text{(III.8)}$$

Case 2:

In this case, we propose that: (i) All access actions have not the same objects and conditions sets, (ii) and all access actions have the two access types. So that, $PN_{R_i} = \sum_{acc=1}^{NAC_{R_i}} 2$. The Figure III.8 plots the number of roles as well as values of WPN in the two cases, depending on the proposed input parameters. The figure III.8 shows that: (i) the number of users, objects or environment features have no effect on the WPN neither on the roles number, (ii) the WPN number increases, exponentially, in order to model the largest number of access actions in the worst

Figure III.8: Experimental Results In The Proposed Model



case. However, in reality, the access actions set is small, (iii) and finally, there is no WPN explosion on the systems that have a large number of objects or complex granularity (Specialities). Hence, the proposed model has a good expressiveness and provides fine-grained access control.

2. *EPN metric* : To decide that the user has not the requested permission, the new proposed model will evaluate at most two rules (one for shared access and one for private access) for each role of this user's active roles (AR). Hence, the number EPN is calculated using: $EPN = \sum_{i=1}^{AR} 2$. Usually, a user is assigned to a small number of roles which means that EPN is not a large number, so that FRABAC has not complex auditing even in high granularity systems.

3. *"Policy modification visualization" metric*: If the administrator adds a permission to a role then all users assigned to this role will have the permission automatically.

4. *"Context-aware access" metric*: The model supports context-aware access.

III.5 Conclusion

Existing access control models (RBAC, ABAC and AERBAC) suffer from several problems when dealing with complicated security policies in complicated systems. These previous models have several drawbacks such as: the explosion in the number of roles and rules, problems with context-awareness, problems with the visualisation of policies update, etc. These drawbacks make these models enable to provide scalability, flexibility, and fine granularity in the cloud and IoT environments. To handle these drawbacks, this work has proposed a new access control model which combines and extends, basically, the two models RBAC and ABAC. In order to demonstrate the advantages of the new proposed model, an empirical study is realised. In this study, the new proposed model is compared versus three existing models based on specific metrics. The results demonstrate that the new proposed model is more suitable than existing ones. A complete validation through simulation and formal verification is presented in the next chapter.

Chapter IV

Third Contribution: Using Hierarchical Coloured Petri Nets in the Formal Study of FRABAC Security Policies

IV.1 Introduction

To ensure that the policy is consistent and that the system will never reach an inconsistent state, formal verification using Petri Nets can be applied. In a formal specification of a FRABAC policy, we must specify the FRABAC constraints to be satisfied in all states of the system. After the specification is done, the formal verification consists to prove that all reachable states during the execution of the system are consistent with respect to the set of predefined constraints.

The first step in a modelling process requires the investigation of the set of events which change the state of the system. In our case (an access control policy using FRABAC model), besides to "*request evaluation*" event, we have the same six major events, which are identified in RBAC (Enabling of a role, Disabling of a role, Assignment of a role to a user, Activation of a role by a user, De-activation of a role, and De-assignment of a role). These six events modify the roles status. Each event in the system requires specific preconditions (a set of constraints in the FRABAC model) to be satisfied. When the events occur, specific post-conditions will be satisfied too (which are, also, a set of constraints in FRABAC model). In the Petri Net model, these events are modelled by transitions. The pre-conditions are modelled by some input-places, some expressions on input-arcs and some associated guards to the transitions. The post-conditions are modelled by some output-places and some expressions on output-arcs. The six events are modelled and analysed in [64]. Hence, in this work, we are interested in request evaluation events. In order to demonstrate the feasibility of the proposed model, we will show by details in the following paragraphs the modelling/analysis process of the FRABAC requests' evaluation.

The rest of the chapter is organized as follows: section two details the specification and the modelling of FRABAC into HTCPN. Section three details the using of CPN-tool to provide a formal verification of FRABAC. Finally, section four concludes this chapter.

IV.2 Specification of the request evaluation process

At runtime, users can access resources through requests. In the proposed access control model, a runtime request expression has the form $< u, ob, ac >$, where u is the user, ob is an object and ac is an access action. To decide if u can access or not the object ob, the proposed access control model proceeds through the two following steps.

1. Identify the type of the request and verify if the user has the right to make such request. If the user has the right then the request is saved and it will be evaluated, otherwise the request is ignored;

2. Evaluate the request.

In the following paragraphs, we present the modelling/verification of the request evaluation process using Hierarchical Coloured Petri Nets. The two previous steps (1) and (2) are modelled as a set of transitions. These transitions have input/output places and guards. To construct the model, a set of types (colours) are required.

IV.2.1 Required types for the HCPN model

We define the following four basic types, that will be used in the Hierarchical HCPN model.

1. $Object\_Index$: This type models the set of N objects (resources). $Object\_Index$ = $index\ Ob$ of $1..N$. An object Ob_1 is expressed as $Ob(1)$.

2. $User\_Index$: This type models the set of N users (subjects). $User\_Index$ = $index\ U$ of $1..N$. A user U_1 is expressed as $U(1)$.

3. $Access\_Index$: This type models the set of N access actions. $Access\_Index$ = $index\ Acc$ of $1..N$. An access action Acc_1 is expressed as $Acc(1)$.

4. $Attribute\_Index$: This type models the set of N attributes. $Attribute\_Index$ = $index\ At$ of $1..N$. An attribute At_1 is expressed as $At(1)$.

Using the above basic types, we define the new following complex types.

1. $Requests$: to model requests. A request is modelled as a triple composed of three integer elements: the $User\_Index$, $Object\_Index$ and the $Access\_Index$. As an example, the element $(U(1), Ob(2), Acc(1))$ in the type Requests specifies that the user $U(1)$ demands to operate on object $Ob(2)$ the access action $Acc(1)$.

2. Req_And_Type: specified as a product $INT * Requests$, where INT is an integer which defines the type of the request (1: if the request is shared and 0: if the request is private).

3. $Attribute$: specified as a couple of an integer (the attribute index) and a string list (values of this attribute). For example, the element $(At(1)$, [“$Algocours$”, “$DataBasecours$”]) specifies that the attribute At_1 has two values: Algocours and DataBasecours.

4. $UserIndexList$, $RoleIndexList$, $ObjectIndexList$, $AttributeIndexList$: four list
types. An element in $UserIndexList$ is a list of user indices, and it is the same for the three other types $RoleIndexList$, $ObjectIndexList$, $AttributeIndexList$. For example, the list $[U(1), U(2), U(3)]$ contains two users, the list $[Ob(1), Ob(2)]$ contains two objects, the list $[At(1), At(2)]$ contains two attributes, and finally the list $[R(1), R(2)]$ contains two roles;

5. $Resources$: specified as a tuple $(Ob(i), U(j), Att\_list)$ of the product $Object$ $\_Index$ * $UserIndexList$ * $Attributeslist$, such that $UserIndexList$ represents the value of the $refer\_to$ attribute, and $Attributeslist$ represents the other attributes and their values. For example:$(Ob(1), [], [(At(1), [\text{“basic”}]), (At(2),$ [“dynamic”])]) specifies that the object $Ob(1)$ is a shared object (because the value of $refer\_to$ attribute is nil). $Ob(1)$ has two attributes $At(1)$ and $At(2)$, such that the value of $At(1)$ is basic and the value of $At(2)$ is dynamic. The token $(Ob(1), [U(1), U(2)], [(At(1), [\text{“basic”}]), (At(2), [\text{“dynamic”}]\ )])$ specifies that the object $Ob(1)$ is dedicated only to two users U_1 and U_2.

6. $Users$: specified as a couple composed of $User\_Index$ and $Attributeslist$. Attributes list represents the attributes of this user and their values. For example, $(U(1)$, [(At(1), ["basic"]), $(At(3)$, ["computer"])]) specifies that the user $U(1)$ has two attributes $At(1)$ and $At(3)$, such that the value of $At(1)$ is basic and the value of $At(3)$ is computer.

IV.2.2 Modelling the identification process

The identification of the type of a request $< U\_i, Ob\_i, Acc\_i >$ is specified as a transition denoted $Request\_Type$. As illustrated in Figure IV.1, this transition is connected to the following places.

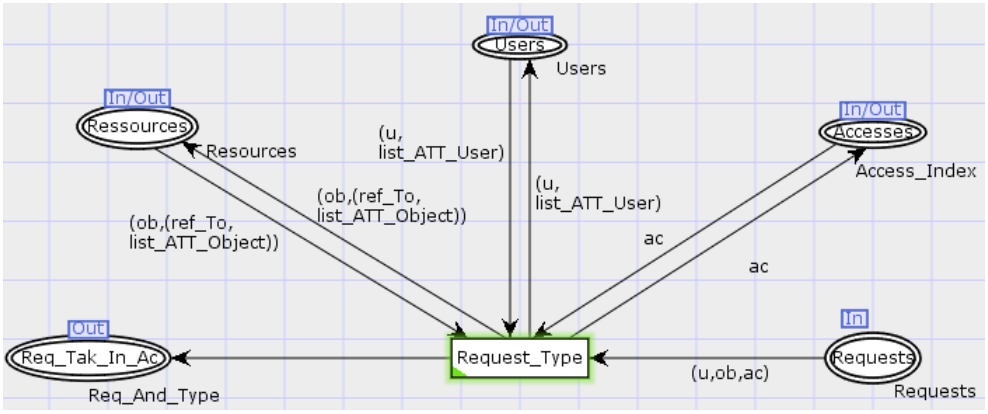


Figure IV.1: Hierarchical CPN model for the identification process

- An input place $Requests$ which type is $Requests$. It must contain the request $< U\_i, Ob\_i, Acc\_i >$.
- An in/out-put place $Users$ which type is $Users$. It must contain the user $(U\_i, User\_Attributeslist)$.
- An in/out-put place $Resources$ which type is $Resource$. It must contain the object $(Ob\_i, refer\_To, Object\_Attributes\_list)$.
- An in/out-put place $Accesses$ which type is $Access\_Index$. It must contain the access action (Acc_i).
- An output place $Req\_Tak\_In\_Ac$ which type is Req_And_Type. This place represents the requests to be evaluated. This place will contain a couple composed of an integer (equals to ȷ1ȷ for shared request or equals to ȷ0ȷ for the private request) and the request $< U\_i, Ob\_i, Acc\_i >$. The transition $Request\_Type$ decides what kind of request must be put into the place $Req\_Tak\_In\_Ac$. The decision is based on the value of the object $refer\_To$ attribute (the $Obi\ UserIndexList$) and the expression:

  $if(\ List.null\ UserIndexList)\ then\ 1`(1, (u, ob, ac))$
  $else\ if\ (mem\ UserIndexList\ u)\ then\ 1`(0, (u, ob, ac))$
  $else\ Empty$

The empty token means that the system will ignore the request if the object ob is a private object which is not dedicated to the user u.

IV.2.3 Modelling the evaluation process

As we discussed above, to evaluate the user's request and to make the decision (can or cannot access), we need to know the attributes of the user, the attributes of the resource and the access action's rule (this rule is among the active role rules). Hence, we have in the evaluation process three steps, which are: (i) get the preconditions, (ii) evaluate the request and (iii) make the decision.

IV.2.3.1 The get-precondition step:

To specify this step in HCPN, we define the two following types:

- $Activated\_Role$: specified as a tuple $(U(i), R(j))$ of the product User_Index * Role_index where the first represents the user identifier and the second represents the index of the currently activated role by this user. For example, the token $(U(1), R(1))$ means that the role R_1 is activated by the user U_1.
- Rule: specified as a tuple $(R(j), N, Acc(m), Att\_Index\_List, Att\_User, Att\_Res)$ of the product Role_Index *INT* Access_Index* AttributeIndexList * AttributeList * AttributeList. In this case, $R(j)$ is the identifier of the role, N is an integer representing the type of access (1: if the access type is shared and 0: if it is private), $Acc(m)$ is the access action identifier, Att_Index_List represents the attributes that must have the same values in the resource and the user, Att_User introduces the attributes and their values that the user must have, and the Att_Res represents the attributes and their values that the resource must have.

As illustrated in Figure IV.2, the step "Get precondition" is divided into two phases. The first phase (Take_Pre) gets preconditions to evaluate a request $< U\_i, Ob\_i, Acc\_i >$. The second phase (Sample_Format) simplifies some of these preconditions to use them in the next step.

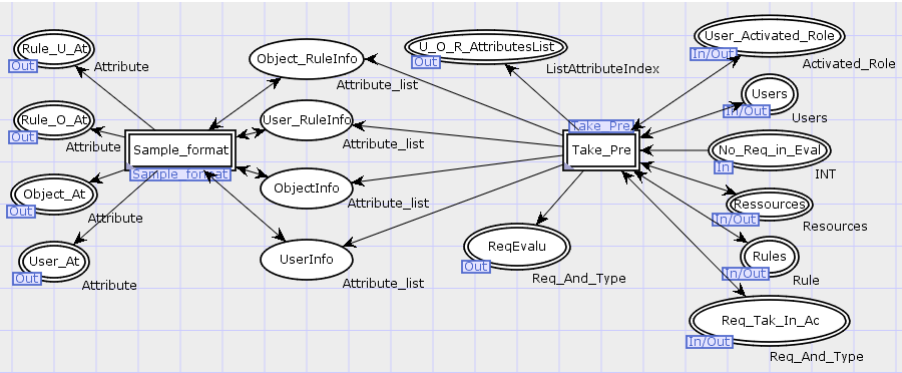


Figure IV.2: Hierarchical CPN model for Get precondition step

The phase Take_Pre is composed of one transition $Take\_Pre$. This last is connected to the following places (as depicted in Figure IV.3):

- The two in|out-put places $Users$ and $Resources$, described previously;
- An in|out-put place $UserActivatedRoles$ which has the type $Activated\_Role$. It must contain the user and his activated role (U_i, R_j);

- The in|out-put place $Req\_Tak\_In\_Ac$ described previously. It must contain the request $(N, (U\_i, Ob\_i, Acc\_m))$;
- An in|out-put place $Permission\_Rule$ which has the type $Rule$. It must contain the rule $(R(j), N, Acc(m), Att\_Index\_List, Att\_User, Att\_Res)$;
- Output places $UserInfo$, $ObjectInfo$, $User\_RuleInfo$ and $Object\_RuleInfo$ which have the type $Attribute\_list$. These places will contain user's list attributes ($User$ $\_Attributes\_list$), object's list attributes ($Object\_Attributes\_list$), the list of attributes and their values that the user must have its (Att_User) and the list of attributes and their values that the objects must have its (Att_Res), respectively;
- Output place $ReqEvalu$ which has the type is Req_And_Type. This place will contain the the selected request $(N, (U\_i, Ob\_i, Acc\_i))$ to be evaluated;
- Output place Us_Res_At which type is $ListAttributeIndex$. This place will contain the list of attributes that must have the same values on resource Ob_i and user U_i (Att_Index_List).

The transition $Take\_Pre$ cannot fire if there is a request in place $ReqEvalu$.

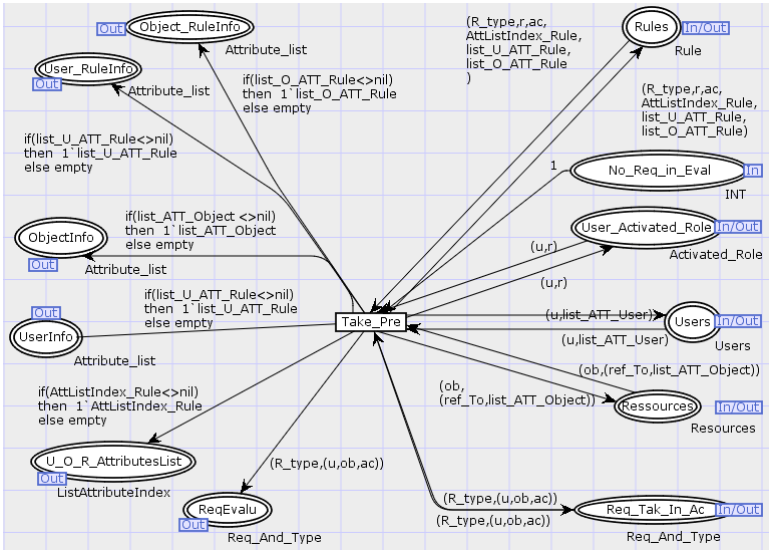


Figure IV.3: Hierarchical CPN model for Take_Pre phase

Now we have the user and the resource information (attributes and its values), also the constraints of the rule ($list\_ATT\_User\_Rule$, $AttListIndex\_Rule$ and $list\_ATT\_User$ $\_Rule$). The Figure IV.4 illustrates how we simplify the user/object/rule attributes list format into a set of couples of the form ($AttributeIndex$, $AttributeValues$). For that raison, we use four transitions T1, T2, T3 and T4, in $Sapmle\_Format$ phase. The transitions T1, T2, T3 and T4 are connected respectively to:

- The in|out-put places $UserInfo$, $ObjectInfo$, $User\_RuleInfo$, $Object\_RuleInfo$ are described previously. They contain attributes in complex format.
- The output places $User\_At$, $Object\_At$, $Rule\_U\_At$, $Rule\_O\_At$ have the type Attribute. They contain the attributes in a simple format that can be used in the evaluation process.

Those four transitions have the priority P_THIGH to avoid the firing of any other transition; before charging all attributes of the user U_i, the object Ob_i and the rule.

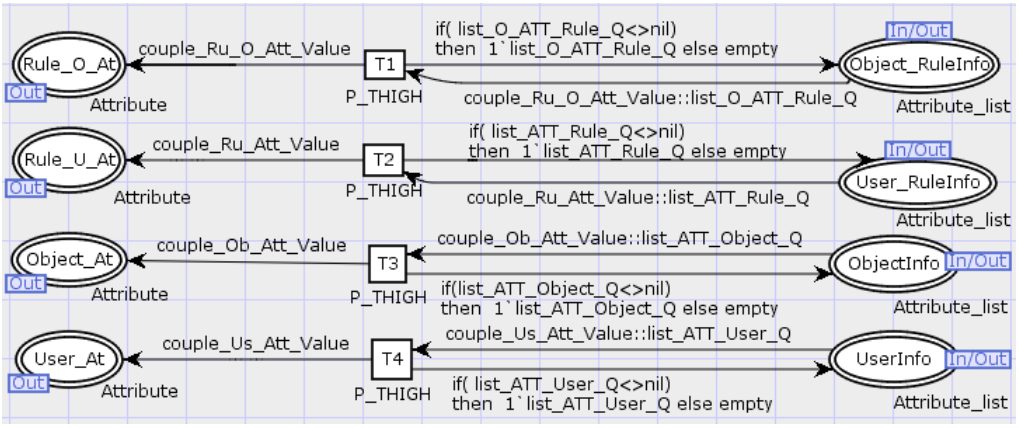


Figure IV.4: Hierarchical CPN model for Sample_Format phase

IV.2.3.2 The evaluation step:

To evaluate the request, we use the two following sub-models.

- $Attributes\_Eval$ sub-model: it contains three transitions to check that the user request respects all the required constraints .
- $Error$ sub-model: it contains three transitions to detect errors.

As illustrated in Figure IV.5, $Attributes\_Eval$ sub-model transitions are as follows.

1. The transition $User\_Res\_Cons$: it is used to check that the attributes in the resource Ob_i and the user U_i have the same values. This transition is connected to the following places.

   - The in|out-put place Us_Res_At described previously. This place must contain the list of attributes.
   - The in|out-put place $User\_attributes$ described previously. This place must contain the attribute at and its values ($couple\_U\_Att\_Value$);
   - The in|out-put place Object Attributes described previously. It must contain the attribute at and its values ($couple\_O\_Att\_Value$);
   - An output place $Values\_Error$ which has the type integer. This place will be marked $(1'0)$ if there is an error otherwise it stays empty.

   The guard [$\#1\ couple\_U\_Att\_Value = at$,$\#1\ couple\_O\_Att\_Value = at$] is used to confirm that this transition fires only when the attribute at exists in the attributes of both the resource and the user.

   The transition $User\_Res\_Cons$ does the following.

   - Take the list of attributes ($at :: AttListIndex$) from the place Us_Res_At;
   - Evaluate one of them (at the first attribute in the list);

- Compare the values of the selected attribute in user and resource;
- If the comparison result is negative then the transition puts an integer token $(1'0)$ in the place $Values\_Error$;
- Return the rest of attributes list in the place Us_Res_At;
- Fire repeatedly until it finishes handling all attributes.

2. The transition $User\_Cons$: it checks if attributes in the user have the same values as in the rule. This transition is connected to the following places.

   - The in|output place $User\_attributes$ described previously. It must contain the attribute at and its values ($couple\_U\_Att\_Value$);
   - The input place $Rule\_U\_Attributes$ described previously. It must contain the attribute at and its values($couple\_U\_Att\_Value$);

   The transition will fire repeatedly until it finishes handling all the couples in the place $Rule\_U\_Attributes$. If it is the case then the user satisfies the second constraints because he has the same attributes and with the same values required in $Rule\_U\_Attributes$.

3. The transition $Object\_Cons$: it checks that the attributes in the object have the same values as those in the rule. This transition is connected to the following places.

   - The in|out-put place $Object\_attributes$ described previously. It must contain the attribute at and its values ($couple\_U\_Att\_Value$);
   - The input place $Rule\_O\_Attributes$ described previously. It must contain the attribute at and its values($couple\_U\_Att\_Value$).

   The transition will fire repeatedly until it finishes handling all the couples in the place $Rule\_O\_Attributes$. If it is the case then the user satisfies the third constraints because the resource has the same attributes and values as required in $Rule\_O\_Attributes$.

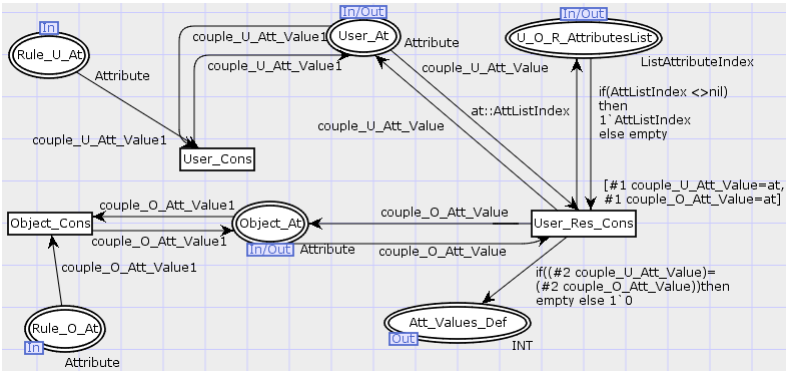


Figure IV.5: Hierarchical CPN model for Attributes_Eval sub-model

As illustrated in Figure IV.6, the transitions of the sub-model $Error$ are:

1. The transition $Attribute\_Not\_found$: If the place Us_Res_At is not empty and the transition $User\_Res\_Cons$ cannot fire, then the transition $Attribute\_Not\_found$ will fire, and it takes the $ListAttributeIndex$ as input. The priority of this last transition is P_low and the priority of the transition $User\_Res\_Cons$ is Normal, hence $Attribute\_Not\_found$ can fire only if $User\_Res\_Cons$ transition can not fire;

2. The transition $User\_couple\_Not\_Found$: If the place $Rule\_U\_Attributes$ is not empty and the transition can not fire then the user does not have an attribute or its correct values. Therefore, the transition $User\_couple\_Not\_Found$ will fire by taking a token from the place $Rule\_U\_Attributes$ as input. The priority of this last transition is P_low and that of $User\_Cons$ is $Normal$, hence it can fire only if $User\_Cons$ transition can fire;

3. The transition $Object\_couple\_Not\_Found$: If the place $Rule\_O\_Attributes$ is not empty then this transition can not fire which means that the resource does not have an attribute or the correct values. The transition $Object\_couple\_Not\_Found$ will fire by taking a token from $Rule\_O\_Attributes$ place as input. The priority of this last transition is P_low and of that of $Object\_Cons$ is Normal, hence it can fire only if $Object\_Cons$ cannot fire.

All of the previous transitions are connected to the same output place($Rejected\_R$) which has the type $integer$. This place models the rejection of the request.

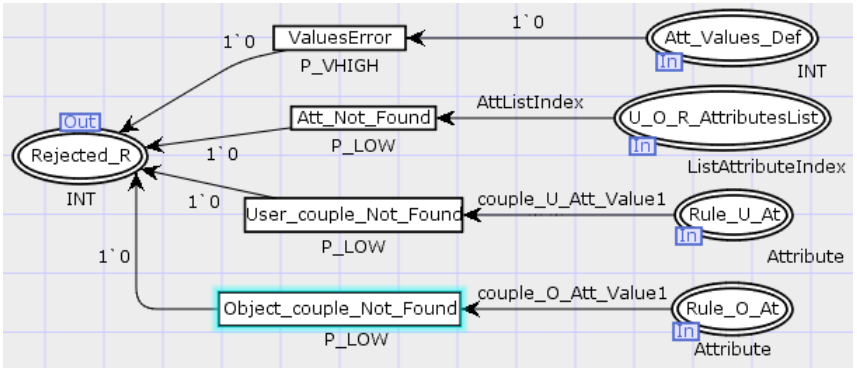


Figure IV.6: Hierarchical CPN model for Error sub-model

IV.2.3.3 Decision step

If the user satisfies all the constraints then he can access the resource, else he can not.

- $CanAccess$: As we modelled in the previous section, if the user satisfies all the constraints then the places Us_Res_At, $Values\_Error$, $Rule\_U\_Attributes$ and $Rule\_O\_Attributes$ will be empty. The transition can_access fires only if those places are empty. This transition takes as input the request from the $ReqEvalu$ place and puts the request in the place $Granted\_Permission$ which contains all granted permissions. Once the request is granted, the transition $handled\_Req$ will remove it from the place $Req\_Tak\_In\_Ac$;

- $Can'taccess$: If there is at least one constraint which is not satisfied then the evaluation of request stops. We model that as follows.

- If there is a token in $Values\_Error$ then the transition $Values\_Error$ fires immediately, because it has the priority P_Hight and it puts an integer token ($1'0$) in the place $Rejected\_Req$.
- The transitions $User\_couple\_Not\_Found$, $Object\_couple\_Not\_Found$ or $Attribute\_Not\_found$ (mentioned previously) put the integer token($1'0$) in the place $Rejected\_Req$;
- When there is a token in the place $Rejected\_Req$, the transition Can_not will fire before other enabled transitions, because its priority is P_VHIGH.

After their firing, both transitions Can_Access and Can_not will reset all places of evaluation part (i.e., there is no request in progress). This step is illustrated in the Figure IV.7.

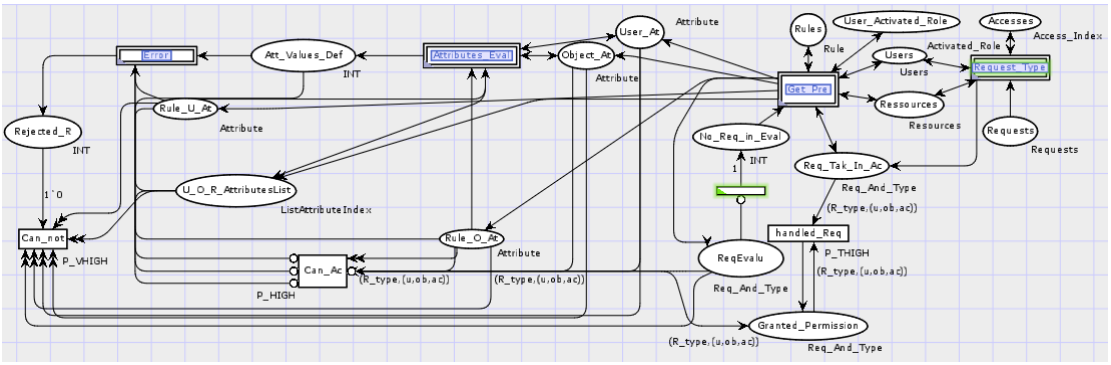


Figure IV.7: Hierarchical CPN model for Decision step

IV.3 Formal verification using CPN-tool of the HCPN models

IV.3.1 Formalization and verification of properties on the request identification

There are four properties which the designer can be interested to verify on the request identification.

- Property 1: "A User $U(i)$ can not access a private resource which is not dedicated to him". This is formulated as: for each input request $< U\_i, Ob\_i, Acc\_i >$, such that ob_i is a private resource and it does not include the U_i index in its list then we must never reach a marking where (1,$(U(i), Ob(i), Acc(i))$) or (0,$(U(i), Ob(i), Acc(i))$) are the place $Req\_Tak\_In\_Ac$;
- Property 2: "A User $U(i)$ can request to access all its private resources". This is formulated as: for each input request $< U\_i, Ob\_i, Acc\_i >$ such that ob_i is a private resource and it includes the index U_iindex in its list $UserIndexList$ then we must reach a marking where token (0,$(U(i), Ob(i), Acc(i))$) marks the place $Req\_Tak\_In\_Ac$;

- Property 3: "A User $U(i)$ can access any shared resource in the system". This is formulated as: for each input request $< U\_i, Ob\_i, Acc\_i >$ such that ob_i is a shared resource then we must reach a marking such that the token (1,$(U(i), Ob(i), Acc(i))$) marks the place $Req\_Tak\_In\_Ac$;

- Property 4: " An object $Ob(i)$ is accessed according to its type". This is formulated as: for each considered request $< Type\_i, U\_i, Ob\_i, Acc\_i >$ the type of object ob_i must be $Type\_i$. In all marking, if (1,$((U(i), Ob(i), Acc(i))$) exists in the place $Req\_Tak\_In\_Ac$ then the list $UserIndexList$ of the resource ob_i must be *nil*;

In order to show the verification of these properties, we will consider a simple policy. A Service provider provides the service Service_1 ($Ob(1)$) for only two customers Adam ($U(1)$) and Lina ($U(2)$), the service Service_2 ($Ob(2)$) for Adam and the service Service_3 ($Ob(2)$) as a public service. The customers are $U(1) \ldots U(5)$. the accesses are: storage denoted by ($Acc(1)$) and calculate denoted by ($Acc(2)$)). As initial marketing we have three requests in the input requests place: $1`(U(1), Ob(1), Acc(2))) + +1`(U(4), Ob(1), Acc(2))) + +1`(U(2), Ob(2), Acc(2))) + +1`(U(3), Ob(3), Acc(2))) + +1`(U(2), Ob(3), Acc(2)))$. After the simulation, the marking of the place $Req\_Tak\_In\_Ac$ is: $1`(0, (U(1), Ob(1), Acc(2))) ++1`(1, (U(3), Ob(3), Acc(2)) ++1`(1, (U(2), Ob(3), Acc(2))$, which illustrates that

- Adam ($U(1)$) can access Service_1 ($Ob(1)$) (proves the property 2);

- Lina($U(2)$) can not access $Service\_2$ ($Ob(2)$) and $U(4)$ can not access $Service\_1$ (proves the property 1);

- $U(3)$ and Lina can access Service_3 (improve the property 3);

- Adam can access $Service\_2$ service with private mode and $U(3)$ and Lina can access $Service\_3$ with public (shared) mode (proves the property 4);

IV.3.2 Formalization and verification of properties on the request evaluation

There are three properties which the designer can be interested to verify on the request evaluation.

- Property 1: "Each evaluated request <$(U(i), Ob(i), Acc(i))$> is considered". This is formulated as: for each request <$(U(i), Ob(i), Acc(i))$> in the place $ReqEvalu$, we must find some path in the reachability tree in which there is a binding: <$Request\_Type$, u=$U(i)$,$ob = Ob(i)$,$ac = Acc(i)$>.

- Property 2: "A request <$(U(i), Ob(i), Acc(i))$ > will never be evaluated except if the user has an active role". This is formulated as: for each request<$(U(i), Ob(i), Acc(i))$ > in the place $ReqEvalu$, we must find some path in the reachability tree in which there is a binding: < $TakePre$, $u = U(i)$,$ob = Ob(i)$,$ac = Acc(i)$,r >.

- Property 3: "A user has only permissions defined in its activated roles". This is formulated as the following: every permission (R_type,$(U(i), Ob(i), Acc(i))$) assigned to a user $U(i)$ must be a permission (R_type, $R(j)$, $Acc(i)$, $AttListIndex\_Rule$, $list\_ATT\_Rule$) defined in a role $R(j)$ and the role $R(j)$ must be activated by the user $U(i)$.

To verify these properties, we use the previous collaborative cloud services example. The UN data owner assigns to users, from both companies IC and AC, their roles as auditor role (R(1)) or insurance role (R(2)). We have two users Lina (U(1)) and Bob (U(2)). Lina is an auditor in AC and Bob is an insurance employee in IC. Any financial reports has two main attributes: its type denoted (At(1)) and its category denoted (At(2)) with the values reports and financial, respectively. To give the auditor the permission: read-only (Acc(1)) financial reports, the owner define the following rule: $< Shared(1), R(1), Acc(1), [], [], [(At(1), [\text{“Report”}]), (At(2), [\text{“Financial”}])] >$. The function that searches for members Ob(1), in the UN is a shared resource and it has the attribute name denoted (At(3)) with the value SearchMember. The owner defines the following rule for the role R(2) to give him the right (Acc(2)) to execute the function Ob(1): $< Shared(1), R(2), Acc(2), [], [], [(At(3), [\text{“SearchMember”}])] >$. We have two other objects Ob(2) and Ob(3), such that Ob(2) is shared and it is a financial report $(< Ob(2), [], , [(At(1), [\text{“Report”}]), (At(2), [\text{“Financial”}])])$ and Ob(3) is also shared and it is a statistical report $(< Ob(2), [], , [(At(1), [\text{“Report”}]), (At(2), [\text{“Static”}])] >)$. The requests are: $P1(< U(1), Acc(1), Ob(2) >)$, $P2(< U(1), Acc(1), Ob(3) >)$, $P3(< U(1), Acc(2), Ob(1) >)$ and $P4(< U(2), Acc(2), Ob(1) >)$. Initially, the role $R(1)$ is activated and the role $R(2)$ is deactivated.

- Property 1: After the execution of the CPN model, we find in the reachability graph, that the markings of place $ReqEvalu$ is: $1'(U(1), Acc(1), Ob(2)), 1'(U(1), Acc(1), Ob(3))$ and $1'(U(1), Acc(2), Ob(1))$ respectively. In the CPN Tools state space report, we find the following bindings:
  <$Request\_Type$, u=$U(1)$,$ob = Ob(2)$,$ac = Acc(1)$ >, < $Request\_Type$, u=$U(1)$, $ob = Ob(3)$,$ac = Acc(1)$ >, < $Request\_Type$, u=$U(1)$,$ob = Ob(1)$,$ac = Acc(2)$ > and < $Request\_Type$, u=$U(2)$,$ob = Ob(1)$,$ac = Acc(2)$ >. These bindings prove that each evaluated request is considered.

- Property 2: From the CPN Tools state space report, we find the following binding: <$Take\_Pre$, u=$U(1)$,$ob = Ob(2)$,$ac = Acc(1)$,$r = R(1)$>, <$Take\_Pre$, u=$U(1)$, $ob = Ob(3)$,$ac = Acc(1)$,$r = R(1)$>, <$Take\_Pre$, u=$U(1)$,$ob = Ob(1)$,$ac = Acc(2)$,$r = R(1)$>. These bindings prove that each evaluated request belong to an active role.

- Property 3: After the execution of the CPN model, we find that the markings of place $Granted\_Permission$ are: 1’(U(1), Acc(1), Ob(2)).These markings illustrate that:

  - Lina can read the financial report, because there is a rule gives the auditors this permission.
  - Lina cannot read the statical report, because there is no rule gives the auditors this permission.
  - Lina cannot execute the search function, because there is no rule gives the auditors this permission.
  - Bob cannot execute the search function, because the role R_2 is deactivated.

These markings prove that the user has only permissions defined in its activated roles.

IV.4 Conclusion

In this chapter, a formal approach using Hierarchical Coloured Petri Nets was applied to prove consistency and required properties in a security policy built on FRABAC. Indeed, the use of hierarchical aspect allows the designer to have well organized models. The hierarchy helps the designer to manage and to understand its model. After the construction of models, we have exploited CPN-tool to verify the consistency of models. Moreover, we have presented how to specify several FRABAC constraints, we have presented a scenario example, and we have discussed the results of its analysis.

CONCLUSION

Thesis aims

The design and implementation of security policies are crucial processes in the development and maintenance of systems in the computer science field. The verification, analysis and validation processes of security policies are indispensable components of high-assurance systems in such field. These three steps are used to prove the consistency of a security policy and to help the system administrators to understand the system behaviour that used a such policy. For those reasons, the main goal of this thesis was to model, analyse and verify formally Temporal Role Based Access Control (TRBAC) [3] policies.

In this thesis, we have chosen temporal RBAC model to be verified, because of the fact that role-based access control is the most predominant model for specifying security policies and access control. However, basic RBAC [1] does not model explicitly the different states of a role and does not take into account various events that are typical of an RBAC system. TRBAC is one of the most important RBAC extension, which is proposed to deal with temporal aspects and it defines necessary temporal constraints to capture the dynamic behaviour of systems that use RBAC, and for their analysis. Moreover, the TRBAC verification approach can be used as a base for other RBAC extensions verification and validation.

In order to provide a flexible, scalable and fine-grained access control model suitable to large-scale networks needs and requirements, the second aim of this thesis was to provide: (i) a hybrid access model which is based on “role” concept as well as “attribute” concept; and (ii) a formal approach to prove its consistency.

Finally, the formal approaches proposed in this thesis are an important action toward leveraging a collection of techniques and tools that help with the design of security policies.

Thesis Contributions

First Contribution

The first contribution was the using of Hierarchical Timed Coloured Petri Nets [5](HTCPNs) in the formal study of TRBAC security policies. This approach uses HTCPN formalism to model the TRBAC policy, and the CPN-tool [6] to analyse the generated models. The time aspect, in HTCPN, facilitates the consideration of temporal constraints introduced in TRBAC. The hierarchical aspect of HTCPN makes the model “manageable”, despite the complexity of TRBAC policy specification. In the HTCPN model, we defined the events that can occur in the system and their preconditions and post-conditions. These preconditions and post-conditions specify the TRBAC constraints that should be satisfied. After the specification by using the Hierarchical Timed CPN formalism, the CPN-tool was used to ensure an analysis of the policies and to verify some inherent properties. We used the Hierarchical Timed Petri Nets (HTCPNs) in a down-top approach (from an abstract model to sub-models) to model TRBAC. Besides the use of HTCPNs, we presented the analysis phase that focuses on the verification process of several temporal properties. The modelling and verification processes are all illustrated on a realistic security policy.

Second Contribution

The second contribution was the proposing of Fine-grained Role-Attribute based Access Control model (FRABAC)[9, 10], as a new Hybrid Access Control Model. The model provides scalability, flexibility, and fine granularity access control and it is suitable to large-scale networks (as cloud and IoT environments) requirements. FRABAC combines the benefits ABAC [7] [8] and RBAC and overcomes the shortcomings of both models known as the combinatorial explosion in rules and roles when the security policy becomes complicated. Besides avoiding the combinatorial explosion, the new proposed model provides a Role Permission Agreement (APA)[12] to handle the inter-organizational access decision in collaborative cloud services cases. To illustrate these advantages, an empirical study is realised. In this study, the new proposed model was compared versus three existing models based on specific metrics. The results demonstrate that the new proposed model is more suitable than existing ones.

Third Contribution

The third Contribution [12] was the use of Hierarchical Coloured Petri Nets in the formal study of FRABAC security policies. A formal approach was applied to prove consistency and required properties in a security policy built on FRABAC. In this approach, we investigated the set of events which change the state of the system. Besides to "request evaluation" event, we have the same six major events (i.e.,Enabling of a role, Disabling of a role, Assignment of a role to a user, Activation of a role by a user, De-activation of a role, and De-assignment of a role), which are identified in RBAC. These six events modify the roles status. Each event in the system requires specific preconditions (a set of constraints in the FRABAC model) to be satisfied. When the events occur, specific post-conditions will be satisfied too (which are, also, a set of constraints in FRABAC model). In the Petri Net model, these events are modelled by transitions. The pre-conditions are modelled by some input-places, some expressions on input-arcs and some associated guards to the transitions. The post-conditions are modelled by some output-places and some expressions on output-arcs. The six events are already modelled and analysed in [64]. Hence, we were interested in request evaluation events.

Thesis drawbacks and future directions

To overcome the thesis's drawbacks, the future aims and our suggestions for future works are:

- Extending the formal verification approach of TRBAC to be suitable to General TRBAC.
- Extending FRABAC: The drawback of FRABAC arises in public cloud where service providers (CSPs) themselves would be able to access the stored data of their clients. Using FRABAC, the data owner cannot prevent the service provider to access his data. In this case, the FRABAC model needs to be enhanced by cryptographic methods.

Bibliography

[1] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[2] Ravi Sandhu, David Ferraiolo, Richard Kuhn, et al. The nist model for role-based access control: towards a unified standard. In *ACM workshop on Role-based access control*, volume 2000, pages 1–11, 2000.

[3] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. Trbac: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, August 2001.

[4] Tadao Murata. *Petri Nets and their Application an Introduction*, pages 351–368. Springer US, Boston, MA, 1984.

[5] Kurt Jensen. *An introduction to the theoretical aspects of Coloured Petri Nets*, pages 230–272. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[6] Cpn-tool can be downloaded (free for academics) from:. `http://wiki.daimi.au.dk/cpntools/cpntools.wikim`. Accessed: June 3rd 2017.

[7] Eric Yuan and Jin Tong. Attributed based access control (abac) for web services. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005.

[8] Xin Jin, Ram Krishnan, and Ravi Sandhu. A unified attribute-based access control model covering dac, mac and rbac. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 41–55. Springer, 2012.

[9] Ben Attia Hasiba, Laid Kahloul, and Saber Benharzallah. A new hybrid access control model for security policies in multimodal applications environments. *Journal of Universal Computer Science*, 24(4):392–416, 2018.

[10] Ben Attia Hasiba, Laid Kahloul, and Saber Benharzallah. A new hybrid access control model for multi-domain systems. In *Control, Decision and Information Technologies (CoDIT), 2017 4th International Conference on*, pages 0766–0771. IEEE, 2017.

[11] Qasim Mahmood Rajpoot, Christian Damsgaard Jensen, and Ram Krishnan. Attributes enhanced role-based access control model. In *International Conference on Trust and Privacy in Digital Business*, pages 3–17. Springer, 2015.

[12] Ben Attia Hasiba, Laid Kahloul, and Saber Benharzallah. Frabac: a new hybrid access control model for the heterogeneous multi-domain systems. *International Journal of Management and Decision Making*, 17(3):245–278, 2018.

[13] Karl Rihaczek. The harmonized itsec evaluation criteria. *Computers and Security*, 10(2):101–110, 1991.

[14] Commission of the European Communities. *Information technology security evaluation criteria (ITSEC): Provisional harmonised criteria*. Office for Official Publications of the European Communities, 1991.

[15] Reinhardt A. Botha and Jan H. P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001.

[16] David F Ferraiolo, D Richard Kuhn, and Ramaswamy Chandramouli. Role-based access control, artech house. *Inc., Norwood, MA*, 2003.

[17] Butler W Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, 1974.

[18] Rui Zhang. *Relation based access control*, volume 5. IOS Press, 2010.

[19] G Scott Graham and Peter J Denning. Protection: principles and practice. In *Proceedings of the May 16-18, 1972, spring joint computer conference*, pages 417–429. ACM, 1972.

[20] Michael A Harrison, Walter L Ruzzo, and Jeffrey D Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[21] Donald C Latham. Department of defense trusted computer system evaluation criteria. *Department of Defense*, 1986.

[22] O Sami Saydjari. Multilevel security: reprise. *IEEE security & privacy*, 2(5):64–67, 2004.

[23] D Elliott Bell and Leonard J LaPadula. Secure computer systems: Mathematical foundations. Technical report, MITRE CORP BEDFORD MA, 1973.

[24] Mark Curphey, David Endler, William Hau, Steve Taylor, Tim Smith, Alex Russell, Gene McKenna, Richard Parke, Kevin McLaughlin, Nigel Tranter, et al. A guide to building secure web applications. *The Open Web Application Security Project*, 1(1), 2002.

[25] Alan C O 'Connor and Ross J Loomis. 2010 economic analysis of role-based access control. *NIST, Gaithersburg, MD*, 20899, 2010.

[26] Michael J Covington and Manoj R Sastry. A contextual attribute-based access control model. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 1996–2006. Springer, 2006.

[27] D Richard Kuhn, Edward J Coyne, and Timothy R Weil. Adding attributes to role-based access control. *Computer*, 43(6):79–81, 2010.

[28] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):4–23, Jan 2005.

[29] A. Abou El Kalam and Y. Deswarte. Multi-orbac: A new access control model for distributed, heterogeneous and collaborative systems. In *In 8th IEEE International Symposium on Systems and Information Security*, 2006.

[30] Abdeljebar Ameziane El Hassani, Anas Abou El Kalam, Adel Bouhoula, Ryma Abassi, and Abdellah Ait Ouahman. Integrity-orbac: A new model to preserve critical infrastructures integrity. *Int. J. Inf. Secur.*, 14(4):367–385, August 2015.

[31] Kangseok Kim and Geoffrey C. Fox. Xgsp-rbac: Access control mechanism based on rbac model in ubiquitous collaboration system. 2009.

[32] Dancheng Li, Cheng Liu, and Binsheng Liu. H-rbac: A hierarchical access control model for saas systems. *I.J.Modern Education and Computer Science*, 5:47–53, 2011.

[33] Antonio Tapiador, Diego Carrera, and Joaquín Salvachúa. Tie-rbac: An application of RBAC to social networks. *CoRR*, abs/1205.5720, 2012.

[34] Hsing-Chung (Jack) Chen, Marsha Anjanette Violetta, and Cheng-Ying Yang. Contract rbac in cloud computing. *J. Supercomput.*, 66(2):1111–1131, November 2013.

[35] Y. Zhu, D. Huang, C. J. Hu, and X. Wang. From rbac to abac: Constructing flexible data access control for cloud storage services. *IEEE Transactions on Services Computing*, 8(4):601–616, July 2015.

[36] Rahat Masood, Muhammad Awais Shibli, Yumna Ghazi, Ayesha Kanwal, and Arshad Ali. Cloud authorization: exploring techniques and approach towards effective access control framework. *Frontiers of Computer Science*, 9(2):297–321, 2015.

[37] Khair Eddin Sabri and Nadim Obeid. A temporal defeasible logic for handling access control policies. *Applied Intelligence*, 44(1):30–42, 2016.

[38] Jun Luo, Hongjun Wang, Xun Gong, and Tianrui Li. A novel role-based access control model in cloud environments. *International Journal of Computational Intelligence Systems*, 9(1):1–9, 2016.

[39] R. Steele and K. Min. Role-based access to portable personal health records. In *2009 International Conference on Management and Service Science*, pages 1–4, Sept 2009.

[40] Xiaoran Wang and C. Bayrak. Injecting a permission-based delegation model to secure web-based workflow systems. In *2009 IEEE International Conference on Intelligence and Security Informatics*, pages 101–106, June 2009.

[41] A. Jie. The realization of rbac model in office automation system. In *2008 International Seminar on Future Information Technology and Management Engineering*, pages 360–363, Nov 2008.

[42] Wesam Darwish and Konstantin Beznosov. Analysis of {ANSI} {RBAC} support in com+. *Computer Standards & Interfaces*, 32(4):197 – 214, 2010.

[43] K.Venkateswar Rao, M.Srinivasa Rao, K.Mrunalini Devi, D.Sravan Kumar, and M.Upendra Kumar. Web services security architectures using role-based access control. *International Journal of Computer Science and Information Technologies*, 1(5):402–407, 2010.

[44] R. Ranchal, B. Bhargava, R. Fernando, H. Lei, and Z. Jin. Privacy preserving access control in service-oriented architecture. In *2016 IEEE International Conference on Web Services (ICWS)*, pages 412–419, June 2016.

[45] Jun Pang and Yang Zhang. A new access control scheme for facebook-style social networks. *Computers & Security*, 54:44 – 59, 2015. Secure Information Reuse and Integration & Availability, Reliability and Security 2014.

[46] Bing-Chang Chen, Cheng-Ta Yang, Her-Tyan Yeh, and Ching-Chao Lin. Mutual authentication protocol for role-based access control using mobile rfid. *Applied Sciences*, 6(8), 2016.

[47] Nagarajan S and N P Gopalan. A dynamic context aware role based access control secure user authentication algorithm for wireless networks. *International Journal of Applied Engineering Research*, 11(6):4141–4143, 2016.

[48] Don Box. *Essential COM*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1997.

[49] NEZAR NASSR and ERIC STEEGMANS. Security service design for the rmi distributed system based on parameterized rbac. In *In the Proceeding of the International MultiConference of Engineers and Computer Scientists Vol I*, pages 1–6, 2011.

[50] Ed Coyne and Timothy R Weil. Abac and rbac: scalable, flexible, and auditable access management. *IT Professional*, 15(3):0014–16, 2013.

[51] Ricky W Butler. What is formal methods? *NASA LaRC Formal Methods Program*, 2001.

[52] K. Knorr. Dynamic access control through petri net workflows. In *Computer Security Applications, 2000. ACSAC '00. 16th Annual Conference*, pages 159–167, Dec 2000.

[53] Konstantin Knorr. Multilevel security and information flow in petri net workflows. Technical report, In: Proceedings of the 9th International Conference on Telecommunication Systems - Modeling and Analysis, Special Session on Security Aspects of Telecommunication Systems, 2001.

[54] Xin Dong, Gang Chen, Jianwei Yin, and Jinxiang Dong. Petri-net-based context-related access control in workflow environment. In *The 7th International Conference on Computer Supported Cooperative Work in Design*, pages 381–384, 2002.

[55] Yixin Jiang, Chuang Lin, Hao Yin, and Zhangxi Tan. Security analysis of mandatory access control model. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, volume 6, pages 5013–5018 vol.6, Oct 2004.

[56] Z. Liang and S. Bai. Role based workflow modeling. In *2006 IEEE International Conference on Systems, Man and Cybernetics*, volume 6, pages 4845–4849, Oct 2006.

[57] K. Juszczyszyn. Verifying enterprise's mandatory access control policies with coloured petri nets. In *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, pages 184–189, June 2003.

[58] Z. l. Zhang, F. Hong, and H. j. Xiao. Verification of strict integrity policy via petri nets. In *Systems and Networks Communications, 2006. ICSNC '06. International Conference on*, pages 23–23, Oct 2006.

[59] F. Feng, C. Lin, D. Peng, and J. Li. A trust and context based access control model for distributed systems. In *2008 10th IEEE International Conference on High Performance Computing and Communications*, pages 629–634, Sept 2008.

[60] A. Walvekar, M. Smith, M. Kelkar, and R. Gamble. Using petri nets to detect access control violations in a system of systems. In *In the Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis*, 2006.

[61] Hejiao Huang and Helene Kirchner. Secure Interoperation in Heterogeneous Systems based on Colored Petri Nets. working paper or preprint, June 2009.

[62] B. Shafiq, A. Masood, J. Joshi, and A. Ghafoor. A role-based access control policy verification framework for real-time systems. In *10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 13–20, Feb 2005.

[63] M. Song and Z. Pang. Specification of sa-rbac policy based on colored petri net. In *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, volume 3, pages 207–210, Dec 2008.

[64] Laid Kahloul, Karim Djouani, Walid Tfaili, Allaoua Chaoui, and Yacine Amirat. *Modeling and Verification of RBAC Security Policies Using Colored Petri Nets and CPN-Tool*, pages 604–618. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[65] Manachai Toahchoodee and Indrakshi Ray. On the formalization and analysis of a spatio-temporal role-based access control model. *J. Comput. Secur.*, 19(3):399–452, August 2011.

[66] Mikhail I. Gofman, Ruiqi Luo, Ayla C. Solomon, Yingbin Zhang, Ping Yang, and Scott D. Stoller. *RBAC-PAT: A Policy Analysis Tool for Role Based Access Control*, pages 46–49. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[67] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The arbac97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, February 1999.

[68] Alberto Calvi, Silvio Ranise, and Luca Vigano. Automated validation of security-sensitive web services specified in bpel and rbac. In *Proceedings of the 2010 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC '10, pages 456–464, Washington, DC, USA, 2010. IEEE Computer Society.

[69] Sadhana Jha, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. Security analysis of temporal {RBAC} under an administrative model. *Computers & Security*, 46:154 – 172, 2014.

[70] Khair Eddin Sabri. Automated verification of role-based access control policies constraints using prover9. *CoRR*, abs/1503.07645, 2015.

[71] Emre Uzun, Vijayalakshmi Atluri, Jaideep Vaidya, Shamik Sural, Anna Lisa Ferrara, Gennaro Parlato, and P Madhusudan. Security analysis for temporal role based access control. *Journal of Computer Security*, 22(6):961–996, 2014.

[72] Silvio Ranise, Anh Truong, and Alessandro Armando. Scalable and precise automated analysis of administrative temporal role-based access control. In *Proceedings of the 19th ACM symposium on Access control models and technologies*, pages 103–114. ACM, 2014.

[73] Thumrongsak Kosiyatrakul, Susan Older, and Shiu-Kai Chin. *A Modal Logic for Role-Based Access Control*, pages 179–193. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[74] Frédéric Cuppens, Nora Cuppens-Boulahia, Meriam Ben Ghorbel-Talbi, Stéphane Morucci, and Nada Essaouni. Smatch: Formal dynamic session management model for rbac. *Journal of Information Security and Applications*, 18(1):30 – 44, 2013. SETOP'2012 and FPS'2012 Special Issue.

[75] Y. Zhou, L. Ma, and M. Wen. A multi-level dynamic access control model and its formalization. In *2015 2nd International Conference on Information Science and Control Engineering*, pages 23–27, April 2015.

[76] Su Yu and Jon. J. Brewster. Formal specification and implementation of rbac model with sod. *Journal of Software*, 7(4), 2012.

[77] K. Sohr, T. Mustafa, X. Bao, and G. J. Ahn. Enforcing role-based access control policies in web services with uml and ocl. In *2008 Annual Computer Security Applications Conference (ACSAC)*, pages 257–266, Dec 2008.

[78] Samrat Mondal, Shamik Sural, and Vijayalakshmi Atluri. Security analysis of {GTRBAC} and its variants using model checking. *Computers & Security*, 30(2–3):128 – 147, 2011. Special Issue on Access Control Methods and Technologies.

[79] Antonios Gouglidis, Ioannis Mavridis, and Vincent C. Hu. Security policy verification for multi-domains in cloud systems. *Int. J. Inf. Secur.*, 13(2).

[80] Samrat Mondal and Shamik Sural. Security analysis of temporal-rbac using timed automata. In *Information Assurance and Security, 2008. ISIAS'08. Fourth International Conference on*, pages 37–40. IEEE, 2008.

[81] David Ferraiolo and Richard Kuhn. Role-based access control. In *In 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[82] David Brossard, Gerry Gebel, and Mark Berg. A systematic approach to implementing abac. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, pages 53–59. ACM, 2017.

[83] Yunpeng Zhang and Xuqing Wu. Access control in internet of things: A survey. *arXiv preprint arXiv:1610.01065*, 2016.

[84] Xin Jin, Ravi Sandhu, and Ram Krishnan. Rabac: role-centric attribute-based access control. *Computer Network Security*, pages 84–96, 2012.

[85] Devdatta Kulkarni and Anand Tripathi. Context-aware role-based access control in pervasive computing systems. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 113–122. ACM, 2008.

[86] Ramadan Abdunabi, Wuliang Sun, and Indrakshi Ray. Enforcing spatio-temporal access control in mobile applications. *Computing*, 96(4):313–353, 2014.

[87] Zhongyuan Xu and Scott D Stoller. Mining attribute-based access control policies. *IEEE Transactions on Dependable and Secure Computing*, 12(5):533–545, 2015.