

Definition of a formal framework for specifying security policies. The Or-BAC model and extensions.

Alexandre Miège

▶ **To cite this version:**

Alexandre Miège. Definition of a formal framework for specifying security policies. The Or-BAC model and extensions.. domain_other. Télécom ParisTech, 2005. English. NNT : . pastel-00001376

HAL Id: pastel-00001376
https://pastel.hal.science/pastel-00001376

Submitted on 8 Sep 2005

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Ecole Nationale Supérieure des Télécommunications
Paris

Mémoire de Thèse

en vue de l'obtention du grade de

Docteur de l'ENST

Discipline : Informatique et réseaux

Présenté et soutenu publiquement

par

Alexandre MIÈGE

le 27 juin 2005

Titre

Définition d'un environnement formel d'expression de politiques de sécurité. Modèle Or-BAC et extensions.

Jury

Ana Cavalli	Présidente	Professeur à l'INT-EVRY
Dominique Chandesris	Examinateur	Conseiller en sécurité des systèmes d'information à la DCSSI
Frédéric Cuppens	Directeur de thèse	Professeur à l'ENST-Bretagne
Thomas Jensen	Rapporteur	Directeur de recherche CNRS/IRISA
Michel Riguidel	Examinateur	Directeur du département INFRES à l'ENST Paris
Pierre Rolin	Rapporteur	Responsable du développement des compétences et des partenariats de recherche à FTR&D

Definition of a formal framework for specifying security policies. The Or-BAC model and extensions.

by

Alexandre Miège

A dissertation submitted to the Graduate Faculty of

Ecole Nationale Supérieure des Télécommunications

for the degree of

Doctor of Philosophy in Computer Science

Defense date: June 27^{th} 2005

Committee:

Ana Cavalli	Chairwoman	Professor at INT-EVRY
Dominique Chandesris	Examiner	Computer security counsellor for the DCSSI
Frédéric Cuppens	Supervisor	Professor at ENST-Bretagne
Thomas Jensen	Reporter	CNRS research director at IRISA
Michel Riguidel	Examiner	Head of department INFRES at ENST Paris
Pierre Rolin	Reporter	In charge of competence development and research parternship at FTR&D

Résumé

Nous présentons dans cette thèse un nouveau modèle de contrôle d'accès dénommé Or-BAC, *Organization Based Access Control*. Il vise à pallier les limites des modèles de sécurité existants tout en simplifiant la spécification d'une politique de sécurité. Nous proposons un modèle plus riche et plus modulaire qui permet de distinguer la rédaction de la politique de sécurité de son implantation. Ceci est rendu possible par l'abstraction des entités traditionnelles du contrôle d'accès : les *sujets* sont employés dans des *rôles*, les *objets* sont utilisés dans des *vues* et les *actions* implémentent des *activités*. De plus l'organisation dans laquelle un règlement de sécurité est défini prend une place centrale dans ce nouveau modèle. On peut ainsi analyser l'interopérabilité d'organisations ayant chacune leur politique de sécurité et par ailleurs modéliser la structure des organisations. Trois autres aspects sont détaillés dans ce mémoire. Premièrement, afin d'obtenir un règlement de sécurité dynamique, nous intégrons une large variété de contextes. De tels contextes permettent d'activer ou de désactiver des autorisations. Deuxièmement, nous offrons la possibilité d'exprimer des autorisations négatives et définissons une méthode de gestion des conflits entre autorisations positives et négatives qui a la particularité d'être paramétrable et de permettre de détecter et surtout de prévenir les conflits. Enfin, nous associons à notre modèle un modèle d'administration, AdOr-BAC, qui permet de gérer l'ensemble d'une politique de sécurité Or-BAC de façon flexible et décentralisée. Nous présentons également deux travaux de mise en œuvre : l'adaptation de notre modèle dans un environnement réseau et le développement d'OToKit, une maquette de saisie et de validation d'une politique de sécurité Or-BAC.

Mots-clés : Or-BAC, politique de sécurité, contrôle d'accès, contexte, détection des conflits, administration, AdOr-BAC, OToKit.

L'annexe B propose un plus long résumé de la thèse en français.

La thèse a été financée par France Télécom Recherche & Développement, et les travaux ont été réalisés à l'ONERA - Centre de Toulouse de mars 2002 à décembre 2004 et à l'ENST-Bretagne (campus de Rennes) de janvier 2004 à juin 2005.

Abstract

This thesis presents a new access control model called Or-BAC (*Organization-Based Access Control*). We aim at overcoming the limitations of the existing models while simplifying the security policy specification. We suggest a more expressive and modular model that enables us to make a distinction between the policy and its concrete implementation. This is obtained by making an abstraction of the traditional access control entities *subject*, *action* and *object*. Actually, *subjects* are empowered in *roles*, *objects* are used in *views* and *actions* implement *activities*. Furthermore, the concept of organization is central to our model. This makes it possible to better analyze interporability between organizations and to model an organization structure by designing hierarchies of organization. Three other features are tackled in this dissertation. First, in order to obtain dynamic security rules, we introduce the entity context. It enables us to define in which circumstances authorizations must be activated and deactivated. Second, we consider negative authorizations since it allows to more easily specify complex policies. As conflicts might occur between positive and negative authorizations, we provide a parametric conflict management strategy that allows us to detect and resolve potential conflicts. Finally, we define an administration model called AdOr-BAC. This administration model is fully compliant with Or-BAC and offers convenient and flexible means to manage Or-BAC policies. The last part of the dissertation is dedicated to two implementation works: The application to a network environment and the development of a prototype application, OToKit, used to design Or-BAC policies and to detect and solve conflicts.

Keywords: Or-BAC, security policy, access control, context, conflict management, administration, AdOr-BAC, OToKit.

This thesis was funded by France Télécom Recherche & Développement. The research took place at ONERA-Toulouse Research Center from March 2002 to December 2004, and at ENST-Bretagne (Rennes) from January 2004 to june 2005.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Frédéric Cuppens. I thank him for the opportunity he offered me to start this thesis and then for his patience and guidance during the past three years. His constant good mood as well as his friendship have made it a real pleasure to do my thesis under his supervision.

I am thankful to the committee members of my thesis defense, Michel Riguidel, Dominique Chandesris and the chairman Ana Cavalli. I would like to thank Pierre Rollin and Thomas Jensen for the time taken to review my dissertation, and for their valuable comments.

I would like to thank Jacques Cazin for having welcomed me at ONERA in the DTIM department for the first part of the thesis, and Gilbert Martineau and Xavier Lagrange for having enabled me to carry out the last year and a half at ENST-Bretagne at the RSM department.

I am especially grateful to Christine Potier from ENST Paris and Josette Brial and Monique Perron from ONERA for their determination in making it possible to start this thesis.

My thanks go to my undergraduate partners Thierry Sans, Fabien Autrel, Joaquin Garcia and Remy Delmas for the work accomplished together, and who by now have become good friends. I would like to thank them for the goods moments spent together over the past years. Many thanks to Céline Coma who was a great help in preparing my thesis presentation, and I wish her good luck for the thesis she is now starting.

I would like to thank Nora Cuppens-Boulahia. Her ideas and suggestions had a large influence on the direction the research took. Furthermore her comments on drafts of the thesis were invaluable.

I benefited from a collaboration with members of France Télécom R&D. My thanks especially go to Béatrice Renard, Sarah Nataf, Jean-Marc Hospital and Pierre Combes.

My dearest thanks go to my family and specially my Parents, Annick and Stéphane, for all their support and love during my never-ending education. Thanks to my brother, Pierre, who is a model for me and is one of the reasons why I did a Ph.D.

Finally, I could not express all my gratitude to my dearest Arline Brisemur. You constantly encouraged me during these three years and pushed me when my motivation was diminishing, even though the decision of doing a Ph.D. had meant that we were separated most of the time, and made our lives quite complicated. Thank you for having spent hours and days to carefully read the draft of this thesis and help me with your professional translation skills. Your exigent coaching during the Ph.D defence preparation highly improved my talk. Thanks you for your patience ... and for everything.

Contents

Résumé **v**

Abstract **vii**

Acknowledgments **ix**

Contents **xi**

Acronyms and Abbreviations **xvii**

List of Figures **xix**

1 Introduction **1**
1.1 Context 1
1.2 Thesis objectives 3
1.3 Outline of the dissertation 4

2 Related work **7**
2.1 Introduction 7
2.2 Entity structuring 9
2.2.1 IBAC models 9
2.2.2 Subjects structuring 11
2.2.3 Objects structuring 14
2.2.4 Actions structuring 15
2.2.5 The Concept of organization 15
2.2.6 Multiple structuring 17
2.2.7 Hierarchy and inheritance 19
2.2.8 Conclusion 22
2.3 Dynamic access control 23
2.3.1 Principle 23
2.3.2 Different kind of contexts 23
2.3.3 Rule-based models 26
2.3.4 Context enforcement 27
2.3.5 Conclusion 27
2.4 Negative authorizations and conflict management 28

2.4.1 Motivation . . . 28
2.4.2 Expression of negative authorizations . . . 29
2.4.3 Conflict management . . . 30
2.4.4 Conclusion . . . 34
2.5 Administration models . . . 34
2.5.1 Introduction . . . 34
2.5.2 Mandatory Access Control . . . 35
2.5.3 Administration of role-based models . . . 36
2.5.4 Delegation . . . 38
2.5.5 Conclusion . . . 40
2.6 Chapter conclusion . . . 40

3 The Or-BAC model 43
3.1 Introduction . . . 43
3.2 The concept of organization . . . 44
3.3 The model entities . . . 45
3.3.1 Subjects and roles . . . 45
3.3.2 Objects and views . . . 47
3.3.3 Actions and activities . . . 48
3.3.4 Attributes . . . 50
3.3.5 Organizational authorization . . . 51
3.4 Modelling contexts . . . 52
3.5 Concrete permission . . . 54
3.6 Security policy . . . 55
3.7 Conclusion . . . 57

4 Or-BAC extensions 59
4.1 Hierarchy within Or-BAC . . . 59
4.1.1 Introduction . . . 59
4.1.2 Role hierarchy . . . 60
4.1.3 View and activity hierarchies . . . 63
4.1.4 Organization hierarchy . . . 64
4.1.5 Conclusion . . . 66
4.2 Modelling Constraints . . . 67
4.2.1 Relevance constraints . . . 67
4.2.2 Cardinality constraints . . . 68
4.2.3 Separation constraints . . . 68
4.3 Conclusion . . . 69

5 Complexity and decidability 71
5.1 Introduction . . . 71
5.2 Datalog$^{\neg}$ . . . 72
5.3 Logic theory . . . 74

5.4 Conclusion . . . 75

6 Modelling contexts **77**
6.1 Introduction . . . 77
6.2 From conditions to context . . . 78
6.3 Context definition in Or-BAC . . . 79
6.3.1 Context example . . . 80
6.3.2 Context composition . . . 81
6.3.3 Compliance with Datalog . . . 81
6.4 Taxonomy and required data . . . 82
6.5 Temporal context . . . 83
6.5.1 Principle . . . 83
6.5.2 Basic temporal contexts . . . 84
6.5.3 Composed temporal context . . . 84
6.5.4 Example of rules using temporal context . . . 85
6.5.5 Decidability . . . 85
6.6 Spatial context . . . 85
6.6.1 Principle . . . 85
6.6.2 Example of spatial contexts . . . 86
6.7 User-declared context . . . 87
6.7.1 Principle . . . 87
6.7.2 Example of user-declared context . . . 88
6.8 Prerequisite context . . . 89
6.8.1 Principle . . . 89
6.8.2 Example of prerequisite context . . . 89
6.9 Provisional context . . . 90
6.9.1 Principle . . . 90
6.9.2 Example of provisional context . . . 90
6.10 Separated context . . . 91
6.11 Context hierarchy . . . 92
6.12 Conclusion . . . 92

7 Prohibitions and conflict management **95**
7.1 Introduction . . . 95
7.2 Principles of the approach . . . 96
7.2.1 Conflicts in theory T_{pol} . . . 96
7.2.2 Priority levels . . . 97
7.3 The new derivation process . . . 99
7.3.1 Step 1: Conflict management strategy . . . 99
7.3.2 Step 2: From organizational to concrete authorizations . . . 101
7.3.3 Step 3: Deriving explicit permissions . . . 101
7.4 The prioritized theory T_{Ppol} . . . 102
7.4.1 Inheritance management . . . 102

7.4.2 Specification of theory T_{Ppol} . . . 102
7.4.3 Conflict in the prioritized theory . . . 103
7.5 Conflict prevention . . . 104
7.5.1 Conflict prevention: first proposal . . . 105
7.5.2 Conflict prevention: second proposal . . . 105
7.5.3 Conflict prevention: last proposal . . . 106
7.5.4 Example . . . 107
7.6 Redundant authorization detection . . . 108
7.7 Conclusion . . . 110

8 AdOr-BAC: The administration model for Or-BAC 113
8.1 Introduction . . . 113
8.2 URA in AdOr-BAC . . . 115
8.2.1 The view URA . . . 115
8.2.2 Managing the view URA . . . 116
8.2.3 Example . . . 117
8.2.4 The prerequisite conditions . . . 118
8.3 PRA in AdOr-BAC . . . 119
8.3.1 The view PRA . . . 119
8.3.2 Managing the view PRA . . . 120
8.3.3 Example . . . 120
8.3.4 Prerequisite conditions . . . 120
8.4 UPA in AdOr-BAC . . . 121
8.4.1 UPA: granting permissions on specific actions and objects . . . 122
8.4.2 Managing the view UPA . . . 123
8.4.3 Example . . . 123
8.4.4 UPA': granting permissions on activities and views . . . 124
8.4.5 Managing the view UPA' . . . 125
8.4.6 Application to delegation . . . 125
8.5 Other administration functions . . . 126
8.6 Conclusion . . . 127

9 Enforcement of the Or-BAC model 129
9.1 Application of Or-BAC in a network environment . . . 129
9.1.1 Motivation . . . 129
9.1.2 Related work . . . 130
9.1.3 Main features of our approach . . . 131
9.1.4 Organizations . . . 132
9.1.5 Subjects and roles . . . 133
9.1.6 Activities . . . 134
9.1.7 Views . . . 134
9.1.8 Security policy . . . 135
9.1.9 Derivation of concrete firewall rules . . . 136

9.1.10 Conclusion . . . 138
9.2 OToKit: Or-BAC ToolKit . . . 139
9.2.1 Motivation . . . 139
9.2.2 Objectives . . . 139
9.2.3 Implementation choices . . . 140
9.2.4 Programming aspects . . . 140
9.2.5 OToKit graphical interface . . . 142
9.2.6 Achieved work . . . 143
9.2.7 Performance results . . . 152
9.2.8 Future works . . . 153
9.2.9 Conclusion . . . 153

10 Conclusion **155**

Bibliography **159**

A Table **1**
A.1 Basic predicates of Or-BAC . . . 1
A.2 Positive and negative authorizations specification in Or-BAC . . . 3
A.3 Derivation rules in Or-BAC . . . 3
A.4 Basic constraints in Or-BAC . . . 5
A.5 Prioritized authorizations in $T_{P_{pol}}$ . . . 6
A.6 Derivation rules in $T_{P_{pol}}$ . . . 6
A.7 Constraints in T_{Ppol} . . . 7

B French summary **9**
B.1 Introduction . . . 9
B.2 Le modèle Or-BAC . . . 10
B.2.1 Les organisations . . . 11
B.2.2 Les sujets et les rôles . . . 13
B.2.3 Les objets et les vues . . . 13
B.2.4 Les actions et les activités . . . 14
B.2.5 Une politique de sécurité à deux niveaux . . . 14
B.2.6 Conclusion . . . 16
B.3 La modélisation des contextes . . . 16
B.3.1 Motivation . . . 16
B.3.2 L'entité *Context* . . . 16
B.3.3 Taxonomie des contextes . . . 17
B.3.4 Conclusion . . . 18
B.4 La gestion des conflits . . . 19
B.4.1 Expression des interdictions . . . 19
B.4.2 Approche générale . . . 20
B.4.3 Politique de gestion des conflits . . . 21

B.4.4 Prévention de conflits 22
B.4.5 Conclusion 23
B.5 Le modèle administratif 23
B.5.1 Introduction 23
B.5.2 PRA 24
B.5.3 Délégation 25
B.5.4 Conclusion 26
B.6 Mise en œuvre 26
B.6.1 Application à un environnement réseau 26
B.6.2 OToKit 27
B.7 Perspectives 27

Acronyms and Abbreviations

ABAC	Activity Based Access Control
AC	Access Control
ACL	Access Control List
AdOr-BAC	Administration of the Or-BAC model
A-ERBAC	Administration of the ERBAC model
ARBAC	Administrative Role-Based Access Control
CBAC	Coalition-based Access Control
C-TMAC	Context-based Team Access Control
DAC	Discretionary Access Control
DRM	Digital Right Management
ERBAC	Enterpriser Role-Based Access Control
FAF	Flexible Authorization Framework
GRBAC	Generalized Role-Based Access Control
IBAC	Identity-Based Access Control
IS	Information system
IT	Information Technology
ISP	Internet Service Provider
LAMP	LogicAl Multi-Policy
MAC	Mandatory Access Control
MPEG	Moving Picture Experts Group
NIST	National Institute of Standards and Technology
OCL	Object Constraint Language
Or-BAC	Organization-Based Access Control
OS	Operating System
PDA	Portable Digital Assistant
PRA	Permission-Role Assignement
RBAC	Role-Based Access Control
RB-RBAC	Rule-Based RBAC
RCL	Role-Based Constraints Language
RRA	Role-Role Assignment
Rule-BAC	Rule-Based Access Control
SARBAC	Scoped Administration of Role-Based Access Control
SDL	Standard Deontic Logic
SOD	Separation of Duty
SQL	Structured Query Language
SSO	System Security Officer

TBAC	Task-Based Authorization Control
TRBAC	Temporal Role-Based Access Control
T–RBAC	Task–Role-Based Access Control
UCON	Usage Control
UPA	User-Permission Assignment
URA	User-Role Assignment
VBAC	View-Based Access Control
VRBAC	View–Role-Based Access Control

List of Figures

2.1 RBAC main components 12
2.2 RB-RBAC main components 12
2.3 The $RBAC_0$ model 13
2.4 The RBAC NIST model 13
2.5 The T-RBAC approach 18
2.6 Access Control in the T–RBAC model 18
2.7 An example of role hierarchy 20

3.1 The *Empower* relationship 46
3.2 The *Use* relationship 48
3.3 The *Consider* relationship 49
3.4 Abstraction of the traditional access control entities 51
3.5 The *Hold* relationship 53
3.6 The *Permission* relationship 54
3.7 The *Is_permitted* relationship 55
3.8 The Or-BAC model 56

4.1 An example of a multiple role hierarchy 60

6.1 Context taxonomy and required data 83

7.1 Permission → Is_permitted 98

9.1 Application to a network example 133
9.2 Role description 134
9.3 Permissions in organization B 136
9.4 Permissions in organization B_fw_1 136
9.5 Permissions in organization B_fw_2 137
9.6 Derivation of concrete firewall rules 137

9.7 The main components of the OToKit program 141
9.8 Example of Prolog rule 141
9.9 Java method with JPL objects 142
9.10 Class hierarchies in OToKit 143
9.11 OToKit graphical interface 144
9.12 Role-user assignment 145
9.13 Authorization expression 146
9.14 Example of authorization inheritance 147
9.15 Context expression 148
9.16 Separation constraint 148
9.17 Conflict prevention 149
9.18 Authorization request simulation 150
9.19 Actual conflict simulation 151
9.20 Performance results array 153

10.1 New Or-BAC architecture 157

B.1 Le modèle Or-BAC 12
B.2 Taxonomie des contextes et données requises 18
B.3 La gestion des conflits dans le modèle Or-BAC 21
B.4 Exemple du billet d'avion 24

Chapter 1

Introduction

1.1 Context

Information system security is a major field of research that is in constant evolution. Companies, institutions and even families increasingly use information technologies. These technologies have indeed an important place in our everyday life, in our professional life as well as in our spare time. Computers enable us to store a large amount of data and to perform a large variety of tasks quickly. New IT devices, like PDAs, appear on the market and broaden the use of digital information. Furthermore, network technologies gain speed and efficiency. This increases the desire to exchange data between enterprises and between individuals. This progress both greatly facilitates activities we carried out by other means, and provides new ways to communicate and work. In return, several risks related to security are on the increase. New practices come with new threats. As a consequence, the awareness of the importance of data protection is also growing.

We should first clarify some vocabulary. We use the term "system" in a general sense of a set of users, data, devices and procedures brought together in order to achieve some predetermined objectives. We call IT system the systems that rely on digital technologies. However, we do not limit the scope of our study to them. We rather turn our attention on information systems in general. An information system (IS) encompasses all solutions provided to store, treat, exchange and protect information in an organization.

IS security is usually defined as the ability to ensure data *confidentiality*, *integrity* and *availability*. Confidentiality corresponds to the protection against data disclosure. In other words, confidentiality is maintained if sensitive data cannot be consulted by unauthorized people. In the same way, integrity refers to the ability to protect data from unauthorized modification. Finally availability consists in the protection against withholding information or resources [TCSEC 1985]. Availability is ensured if data is available under the conditions of time, delay and performance defined beforehand. These are the main security properties. One could also add for example *auditability* (provide sufficient information related to the IT system's current activities), *non-repudiability* (a user cannot deny an action he performed) or *privacy* (keep private information secret). Securing an IS consists in maintaining these properties.

The definition of a process aiming at ensuring these properties in an IS might be divided

into three steps. The first step is the *risk analysis and management*. This consists in examining the threats which hang over the IT system, highlighting its vulnerabilities and determining which of the data is sensitive and at what degree. Thanks to this analysis, it is then possible to establish the *security policy*. Such a policy contains a set of rules that state the requirements (inferred from the risk analysis) regarding the protection of information and resources. It might specify which users are allowed to access which data for instance. The third step corresponds to the *enforcement of the security policy*. This deals with the choice and implementation of technical solutions that carry out the policy requirements. In this thesis we take an interest in the second step, that is, the security policy part.

It is not so easy to give a definition of security policy. It might address different issues, in different fields and using different formalisms. Security policies can be used to specify the physical accesses to buildings, rooms or devices; as well as the logical accesses to digital data stored in file systems or databases. Network accesses, that is, authorized information flows described as services between networks or hosts are also expressed in the security policy. Furthermore, these policies might be written in natural language, in mathematical formalisms or in particular security devices languages. Part of the policy corresponding to usual practices might even not be expressed at all. As a consequence, in most organizations, the security policy is not a single, complete and coherent document but rather a set of rules, charters and recommendations scattered in different departments, managed by several people and expressed in different languages.

In order to clarify the security policy issue, let us first consider the definition given in the european security evaluation criteria ITSEC [ITSEC 1991]. A policy is a three level concept. At the highest level is the *Corporate Security Policy* also called *Organizational Security Policy*: it is the "set of laws, rules and practices that regulate how assets, including sensitive information, are managed, protected and distributed within the organization". In accordance with the corporate security policy, an organization can then specify for each system[1] a specific *System Security Policy*. The definition of such policy is exactly the same as above but applied to systems. Finally, the technical measures to be applied to a system may be separated from the remainder of the system security policy in a separated document called *Technical Security Policy* and defined as follows: "set of laws, rules and practices regulating the processing of sensitive information and the use of resources by the hardware and software of an IT system".

This definition is general and might encompass a great number of fields. However, it brings in a significant notion. A security policy should be specified at several levels. Each level being more specialized than the one above while remaining compliant with it. This aims at obtaining a top-bottom hierarchical security policy. Afterwards we make the most of this idea: we introduce the notion of organization. Systems can then be considered as sub-organizations and are associated to specific security policies.

The first interesting works on security policies appeared at the end of the 60s and actually focused on the access control part which establishes the access granted to users over the resources of a system. Discretionary access control [DAC 1987, Harrison et al. 1976] only deals with IT systems and formulates policies at the implementation level. Mandatory access control [Bell and LaPadula 1976, Lampson 1971] provides solutions to ensure confidentiality

[1] In the ITSEC, a system is defined as "a specific IT installation with a particular purpose and known operational environment".

and is based on data labelling. Research on such models were first dedicated to military environment. In the early nineties new models were defined, among which is the so called RBAC model [Ferraiolo et al. 1993]. These models have the particularity of being adapted to civil and commercial organization needs. Moreover, they take into account requirements of modern information systems. Some of them suggest abstraction and structuring means in order to move away from implementation and to offer higher level policies. However these security policy models typically come from specific perspectives and tend to emphasize a particular system or application.

1.2 Thesis objectives

We should first motivate the definition of a model dedicated to security policies. Modelling a security policy consists in using techniques to elaborate a mathematical representation of the existing system components related to security, for the purposes of converting it into a computer model. Such a model is used to understand better the policy and its environment, to analyze and improve its functioning, and to simulate the effects of changes on the system. Modelling is used to specify and analyze a policy and prepare it to be converted into closer implementation languages. The expressivity of a security policy depends on the model it relies on.

The research in security policy modelling has been really active in the past fifteen years. Many new models have been proposed: access control models, flow control models, usage control models, administration models, network policy models, etc. Nevertheless, we claim that none of them is fully satisfactory with respect to the requirements we are attempting to address.

Security is essential and even vital to ensure the smooth running of information systems and the durability of organization relying on such systems. Our ambition is thus to provide a convenient, flexible and expressive security model that makes it possible to deal with a wide range of security issues, in two axes: horizontally by addressing many security fields like physical and logical accesses, and vertically by reasoning from the highest organization level to implementation.

Within the framework of this dissertation we design a new security model called *Organizational-based Access Control*, or Or-BAC. It relies on a fragment of first-order logic, and more precisely on Datalog. The first research works on this model was carried out in the context of the MP6 project[2]. In this thesis, we extend the Or-BAC model and build new features.

For the time being, the scope of the Or-BAC model is limited to the access control (AC) part of a security policy. Afterwards I might use the term “security policy” instead of “access control policy”. We introduce in Or-BAC several features we claim that have to be integrated in a modern AC model. Let us enumerate these features.

We first suggest a federative solution which allows us to abstract the traditional access control entities. This abstraction provides means to specify high level organizational policies, regardless of the implementation choices. We apply hierarchies and inheritance in order to

[2] http://www.telecom.gouv.fr/rnrt/rnrt/projets/res_01_59.htm

bring in facilities to manage wide and complex policies.

Since an information system evolves along several circumstances, the security rules specified in a policy must be adapted to a dynamic environment. This is carried out by defining dynamic rules that depend on given *contexts*. Or-BAC allows us to express a large number of contexts, and thereby is suitable in many application fields.

Traditional access control models are composed of rules that grant users some permissions to access data and resources of the system. These positive rules are convenient for simple systems. However, we claim that negative rules that explicitly state the forbidden accesses must also be integrated in an AC model. We made Or-BAC compatible with such a requirement. As a counterpart, expressing positive and negative rules raises an awkward problem since the rules might be conflicting. Therefore, we suggest a powerful solution to detect, and above all, to prevent any appearance of conflicts. Furthermore, the prevention mechanism is defined at the organizational security level. This ensures the absence of conflicting situations whatever are or will be the implementation choices.

Finally, the security level of an IS relies on the ability to provide a relevant policy. This must be ensured at the time of the definition of the policy; but in order to maintain its relevance, the model must provide special procedures allowing to update the policy and keeping it in adequation with the system. These administrative procedures are taken into account in our model since we define an associated administration model called AdOr-BAC. This last model has, among others, the particularity to use a formalism fully compliant with Or-BAC.

This dissertation's objectives is to present these four features. Elsewhere, I also present two works carried out on the application of the Or-BAC model on concrete policies. I first present an application of Or-BAC in a network environment. Then I describe OToKit (Or-BAC Toolkit), a software prototype designed to specify Or-BAC policies, detect potential conflicts, and simulate concrete policies.

1.3 Outline of the dissertation

In this thesis, I first develop in chapter 2 a state of the art related to security policies and more precisely regarding access control polices. My purpose is not to detail each model individually, but rather to focus on the main requirements a new security model should address and to describe existing models as I mention these requirements. In chapter 3, we shall define the basis of the Or-BAC model. We describe its components and the way they are related. We describe others features of Or-BAC in chapter 4 in order to give a complete overview. We broach the hierarchy and constraint issues, and present a formal definition of the model. The chapter 6 is dedicated to the specification of flexible and dynamic security rules to capture, through the definition of contexts, complex and dynamic security requirements. In chapter 7, we deal with the conflicting situations that might occur between permissions and prohibitions and with the solution we provide to prevent, detect and solve conflicts. The AdOr-BAC model is presented in chapter 8. The definition of this model completes the one of Or-BAC and aims at specifying administrative procedures for creating and updating a security policy. We present concrete works in the context of Or-BAC policies in chapter 9. We show that Or-BAC is flexible and strong enough to be applied to network policies. Finally, the conclusions and perspectives related to the thesis

are the subject of chapter 10. In appendix A, the reader can find all predicates and rules defined throughout this dissertation.

Chapter 2

Related work

The objective of the thesis is to design a new access control model. This chapter is dedicated to the main requirements our model has to fulfil. As a consequence, this chapter is not organized as a catalogue of existing models. Rather, we suggest developing the requirements we want to achieve, and study at the same time the solution proposed in the literature.

2.1 Introduction

A security policy, whatever definition it has, is composed of a set rules that define how users may interact with objects. Such rules may specify that some users can, cannot, or must have an access to some objects. Thus, we should first clarify the vocabulary used afterwards. If a rule states that a user can have an access to an object, it is called a "positive authorization". We might also call it a "permission". Other terms are used in the literature such as "privilege" or "right", but we consider only the first two ones. If a rule states that a user cannot have an access to an object, it is called a "negative authorization" or a "prohibition". "Negative permissions" and "denials" can also be found in the literature. The term "authorization" is used to describe equally a positive or a negative authorization. Finally if a rule states that a user must have an access to an object, it is called an "obligation".

There is a great number of existing security policy models in the literature. Some address the same issues, some address different ones. For instance some models are dedicated to activity sequence control, others to the users structuring. As mentioned earlier, we mainly focus in this thesis on models that aim at controlling the use made by the users of the information system resources. In order to browse the models we consider relevant in this work, we make a distinction between four families of security policy models: the access control models, the flow control models, the usage control models and the administration models.

Access control models enable us to specify which actions the users are allowed to carry out on which objects. Their aim is to protect resources and services from unauthorized access. If we make a distinction between an object – container – and the information – content – that this object carries, we should say that access control policies attempt to govern access to containers. Most of access control models only consider positive authorizations [Harrison et al. 1976, Sandhu et al. 1996, Thomas and Sandhu 1997,

Thomas 1997], but more recently some models attempt to integrate negative authorizations also [Jajodia et al. 2001b, Bertino et al. 2003, Halpern and Weissman 2003].

Flow control models aim at providing an efficient response to one of the main problems of access control models: in access control, if programs – more precisely processes – are considered as subjects, then a malicious process might illegally transmit some unauthorized data. Therefore in flow control models, the objective is to control the access to data – i.e. the contents – by controlling the information flow. First works on flow control [Bell and LaPadula 1976, Biba 1975, Kang et al. 2001] were mainly dedicated to securing information system in the defense area.

Usage control (UCON) models are the result of recent research works in the field of digital right management (DRM). The usage control approach is quite different from the access control approach. In access control, some permissions are granted to users to access "static" objects, in other words, objects that are usually stored within the user's organization. Usage control is based on a different paradigm in which objects no longer stand in a computer system but are also shared or sent through Internet to private computers or PDAs, MP3 players, etc., owned by numerous and unknown clients. In [Park and Sandhu 2002, Sandhu and Park 2004], the authors design a model called UCON_{ABC}, which provides authorizations, obligations and conditions management.

Administration models are usually ignored in most security policies. We claim that administration is a major part in the security policy area, mainly for two reasons. First, managing a large scale information system implies that a complex and sizeable security policy has to be administrated. This can usually not be done by a single system security officer (SSO). Next, an information system evolves over the time and according to the evolution of the organization it corresponds to. Therefore, the security policy must also evolve in order to always match the information system security requirements. For these reasons, appropriate administrative procedures must be designed in order to state which users are allowed, among other tasks, to add, modify or delete authorizations. A set of such procedures can be viewed as a meta-policy. Thus a complete administration model must be designed.

Within the framework of the thesis, we focus on access control and administration models. We do not tackle the flow control field. We assume that programs are trustful. Moreover, we cite some characteristics of usage control models, but without examining them in detail. So far, the model we propose does not address the digital right management (DRM) issue.

Information systems become every day more complex, and the awareness of the importance of data and resources protection is increasing. As a consequence, there is a larger number of more accurate security policies needs that ought to be addressed. It calls for more flexible and expressive policies. This has led to an extensive research activity these past years and the definition of a large number of models. These models deal with very different requirements. Unfortunately, they usually only address one issue. Therefore, we might regret the lack of models which aim at fulfilling several requirements at the same time.

The objective of this chapter is to browse and discuss the models we consider relevant. Nevertheless, this chapter is not organized along these models, but instead along the needs and requirements a new model should fulfil. These requirements are: entities structuring, dynamic authorization expression, negative authorization expression and conflict management, and administration. Existing models are considered further as we present these needs.

We begin with the easiest model and see how to extend it in order to obtain a more complex and more flexible model.

The remainder of this chapter is organized as following. In section 2.2 we focus on the necessity to offer means to organize and structure the traditional access control entities. This section starts with simple authorizations over the entity triplet $< subject, action, object >$. Then we examine how to structure these entities in order to make policies more general and their management easier. To obtain more expressive policies, it should be possible to bring authorizations to evolve along certain circumstances. This is addressed in section 2.3 which is dedicated to dynamic models. Dynamic authorizations correspond to a major requirement since modern information systems can definitely not be regulated with static rules. So far, we consider positive authorizations only. We show in section 2.4 that negative authorization expression might be useful in several ways. In particular, this enables the system security officer (SSO) to specify positive policies in which negative authorizations correspond to some exceptions. Models that enable us to express negative authorizations are discussed in this section as well as the mechanisms they proposed to detect and solve conflicts that may appear between permissions and prohibitions. Finally, a full security policy model must be associated with an administration model. Without such a model, the relevance of a policy cannot be maintained. Therefore, the ability to administrate a policy is a strong requirement. Section 2.5 is dedicated to this issue.

2.2 Entity structuring

In this section, we deal with the necessity to structure the traditional entities *subject*, *action* and *object*. The relevance of an access control policy relies on the modelling choices. We first consider the most simple access control model, namely IBAC or Identity-based access control. Then we shall examine several manners to design a more flexible and powerful model.

2.2.1 IBAC models

IBAC models are the first models suggested in the literature and are implemented in most operating systems (OS), such as Windows, Unix, Linux, etc. In identity-based models, authorizations are granted to users according to their identity. Let us first define the relevant vocabulary we use afterwards.

Subject, Object and Action

The notions of “subject” and “object” were first introduced by Lampson [Lampson 1969]. A subject is an active entity. In a computer system context, it includes users, that is, subjects in the form of a person, and processes which run on behalf of these users. A subject may cause some information to flow among objects and the system state to change. An object is a passive entity that contains or receives information. Accessing an object potentially implies access to the information it contains. Objects can be files, directories, programs, printers, etc. In [Harrison et al. 1976] no restriction is made regarding entities that might be both subject and object. Since an entity is active – resp. passive – it can be considered

as a subject – resp. an object. Nevertheless it is usually admitted that the set of subjects is a subset of the set of objects.

Subjects interact with objects through different modes. These modes are called "Action", "Access type" or "Access mode". In operating systems, the actions are for instance "read", "write", "execute", etc. In databases, actions are "INSERT", "CREATE", "UPDATE", etc.

Principle

In general, access control models are used to govern direct accesses made by subjects over objects using actions. As a consequence, a policy can be modelled as a set of triplets $< subject, action, object >$. All IBAC models have the distinctive feature of being based on the subject entity. For example, in UNIX systems, users are identified by their *uid* and processes by their *pid*. This identifier is the key to decide whether or not an access to an object is granted to a subject.

This is a really simple access control model. Managing a large number of subject may become difficult. To simplify the management of IBAC policies, groups of subjects can be formed. A group is merely a collection of subjects which can occur as a single entity in a triplet $< subject, action, object >$. Therefore, assigning an authorization to a group confers this authorization to all its members. Groups are used in most implementations.

IBAC is often confused with discretionary access control (DAC) models [DAC 1987]. DAC can be seen as an extension of IBAC. The control in a DAC model is 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 [TCSEC 1985]. Usually in DAC models, subjects own objects, and in general they own the objects they created. Thereby subjects are allowed to specify explicitly the actions that other subjects may have over objects under their control. We claim that DAC models should rather be considered as administration models and thereby this matter is further discussed in section 2.5.

Access matrix representation

An access matrix [Lampson 1969, Harrison et al. 1976] is a two-dimensional matrix representing subjects on the rows and objects on the columns. Each entry in the matrix represents the access types held by a subject on an object. Access types can be "read", "write" or "execute". Access matrices are a convenient way to represent access control rules.

Implementation

Several mechanisms can implement an IBAC policy. We consider here only three of them: access control list (ACL), protection bits, capabilities.

In relation to the access matrix, ACLs roughly correspond to the transcription of the matrix columns. An ACL can thus be viewed as an object attribute. It specifies which subjects can access this object. Usually an ACL refers to one or several subjects, or to a group, like in Multics [Honeywell 1984] and Trusted Minix [Donaldson et al. 1990].

Protection bits also correspond to access matrix columns. UNIX system is a well-know implementation of protection bits. For each object, there are three groups of three bits. Each bit controls the *read*, *write* and *execute* accesses to the object. The three groups

represent the object's owner, the object's group, and all others. This mechanism is a partial representation of the access matrix since the access modes of a given user cannot be defined.

Instead of ACLs and protection bits, capabilities roughly correspond to the matrix rows. Capabilities can thus be viewed as subject attributes. A subject's capabilities describes the operations that this subject can perform on given objects. A capability is a ticket described as a pair (x, r) where x is an object and r a set of access rights. This ticket offers to its owner the permission to perform accesses r on object x.

IBAC models propose a simple solution to control the interaction between subjects and objects. Nevertheless, managing a large scale information system using an IBAC model is quite heavy. Groups of subjects or sets of access rights might be useful, but inadequate in information systems where many subjects, actions and objects are defined. A modern security policy should then propose means to structure these entities and make their management easier.

2.2.2 Subjects structuring

The role-based access control (RBAC) models [Ferraiolo and Kuhn 1992, Sandhu et al. 1996, Sandhu 1998, Guiri 1995] suggest a convenient solution to manage a set of subjects. Role-based models usually consider subjects as persons, therefore the term "user" replaces the term "subject". In order to propose a way to structure a set of users, these models introduce the concept of "role".

The concept of role

The concept of role is not specific to the security policy area. This concept is of course widely used in organizations. Roughly speaking, a role is a function played by a user. In an organization, a given user is always assigned to one or several particular tasks and gets a title for that. It can be for example "head of department", "export manager", "junior engineer", etc. Thereby a role is an organizational position. In role-based models, the users structuring is based on the set of roles defined at the level of the organization and relies on the following paradigm: in order to fully accomplish his role, a user needs specific authorizations.

As simple as it seems, [Sandhu et al. 1996] points out that the concept of role is not devoid of ambiguity. A role may both represent a competence, such as a diploma, or an effective function that carries specific authority and responsibility. In access control models, roles are usually used as effective functions.

Access control principle

The security policy does not directly grant authorizations to users but rather to roles, thereby roles can be viewed as pools of authorizations. Therefore, the RBAC model [Sandhu et al. 1996] is closer to the notion of capability than to the notion of ACL[1]. Action and object concepts are taken into account in the RBAC model. Furthermore, the RBAC model only considers positive authorizations, that is "permissions".

[1] We compare RBAC permissions to capabilities. Nevertheless, [Barkley 1997] shows how ACLs may be expressed using a very simple role-based model called $RBAC_M$.

Users are assigned to roles. A user obtains all the permissions associated with the role he plays. A user can play several roles and thereby gets all the permissions of its roles. Figure 2.1 shows that the entity role is used as an intermediary entity between users and permissions.

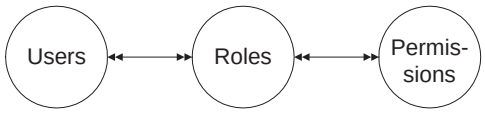


Figure 2.1: RBAC main components

The difference between groups and roles has been widely discussed in the literature. Even if groups might be used to model roles, groups and roles correspond to two really different concepts: a group is a collection of users whereas a role is a collection of permissions. It is worth mentioning that roles and groups may complete each other. For example in [Jajodia et al. 2001b], a group can play a role, with the meaning that each user of that group plays that role.

The concept of role is a really convenient manner to structure users. First because it is a quite intuitive notion, and it enables to establish a relevant matching between organization structure and access control policy. Secondly, it provides a handy solution to manage the policy. For example the SSO can define relevant roles and examine which permissions these roles need. Then, all he has to do is to assign the users to the roles. Updating the policy is easy since the modification of the roles' set of permissions automatically updates the permissions of all the users that are member of this role.

Attribute-based assignment

In [Al-Kahtani and Sandhu 2002], the authors propose an extension of RBAC called Rule-Based RBAC (RB-RBAC). Based on attributes, this model makes it possible to automatically and dynamically assign users to roles, and thereby alleviates the SSO'tasks. RB-RBAC adds the entity "attribute" to the main RBAC model as presented in figure 2.2.

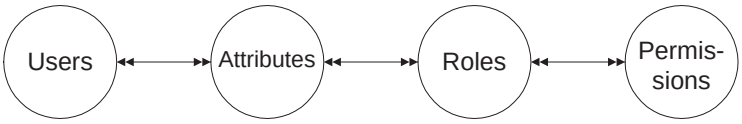


Figure 2.2: RB-RBAC main components

The assignment of users to roles is carried out through rules of the following form: $\mathrm{ae}_i \Rightarrow \mathrm{r}_g$. In other words, once a user complies with the conjunction of attribute conditions ae_i, called attribute expression, this user is assigned to role r_g.

Sessions

In the RBAC_0 model [Sandhu et al. 1996] the concept of session is introduced in order to better manage the relationship between users and roles. Indeed a user needs to open a session and activate a role to get the corresponding permissions. A user is allowed to open

several sessions. Within a session, a user is not obliged to activate all his roles but only the subset of his roles that is necessary to perform a given task. Figure 2.3 presents the complete $RBAC_0$ model.

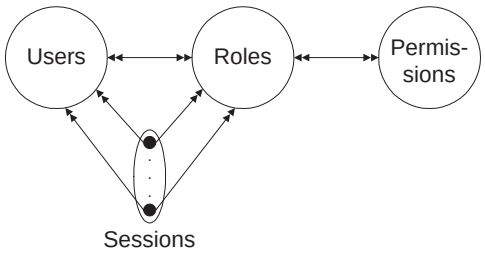


Figure 2.3: The $RBAC_0$ model

It is also possible to specify hierarchies and constraints. Hierarchies are discussed in section 2.2.7. Constraints are introduced in $RBAC_2$ and are useful to govern sessions. For instance, constraints can be used to implement the principle of separation of duty. In other words it makes it possible to specify that not any user can play two conflicting roles at the same time.

Role-based models just represent a step forward in the access control area. Many models such as [Bertino et al. 1996, Jajodia et al. 2001b] offer the possibility to express role-based policies. Nevertheless we might regret that the concept of permission is primitive. When specifying the security policy of a given application, the RBAC model must be refined so that the structure of permissions be explicit. It is argued that this is because this structure might depend on the application. For example, the NIST solution [Ferraiolo et al. 2001, Weber 1997] refined the entity Permissions into two new entities "Operations" and "Objects" (see figure 2.4). We believe that it would be better to include a generic structure of permissions in the model.

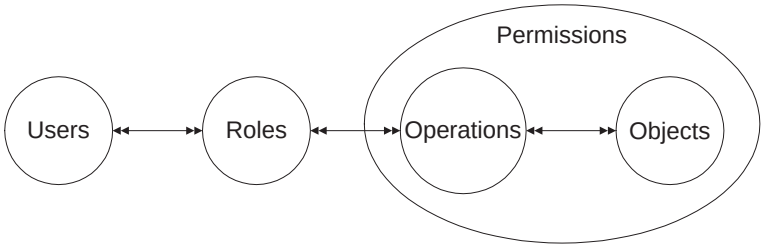


Figure 2.4: The RBAC NIST model

Generally speaking, the original RBAC model suffers from the lack of formal definition. Some work was carried out in this direction, in [Gavrila and Barkley 1998] for instance. In [Khayat and Abdallah 2003], the authors suggest a formal model of a $RBAC_0$-like model based on the specification notation Z.

A 1993 NIST study [Ferraiolo et al. 1993] concludes that role-based models offer means to meet the majority of the needs of commercial and civil organizations. We claim that some major improvement are possible.

2.2.3 Objects structuring

The presentation of the RBAC model was an opportunity to study a solution that makes it possible to organize and structure a set of users. We shall now examine the main proposals to organize and structure a set of objects.

We should first consider how most operating systems provide a simple way to organize objects: files are structured into a folder hierarchy. If an action is performed on a folder, then this action may automatically be applied to all the files inside this folder. This is the case, for example, when one change a folder's protection properties.

With some OSs, it is also possible to apply these actions on some files in accordance to their extension. It is exactly what is suggested in [Jajodia et al. 2001b, Bertino et al. 2003]. Their objective is to model authorizations like "All MPEG files are accessible to the vice-presidents". Actually this is close to the notion of view in relational database systems. In a relational database based on the Structured Query Language (SQL) norm [Grahne and Ghelli 2002], views can be used to put together some tuples. A view is a derived relation, that is, a relational expression which gets a name. For example, the following view contains all customer's accounts which have a credit balance.

```
CREATE     VIEW      Credit_account   AS
           SELECT    Number, Balance
           FROM      Customer_account
           WHERE     Customer_account.Balance > 0;
```

On the other hand, SQL makes it possible to design an access control policy by using GRANT and REVOKE instructions. The GRANT instructions allows the assignment of permissions to users. Therefore, if appropriate views are created, it is possible to design a security policy over views and not only over objects. For example, the following instruction specifies that John and Mary can select and delete any tuples from the view *Credit_account*.

```
GRANT      SELECT, DELETE
           ON      Customer_account
           TO      John, Mary;
```

SQL is a good example of language that enables to apply permissions to groups of objects, where groups are constituted in accordance with some object attributes. Models based on this idea might be called "View-based Access Control" (VBAC) models. Let us consider another solution to regroup some objects. In the TMAC model [Georgiadis et al. 2001], an object-oriented way is suggested. Objects are called "object instances" and are abstracted into "object-types". An object-type is "customer account" for example. All customer's accounts are seen as object instances of this object-type. Unfortunately, the model does not provide any relation or rule to describe object-types and object instances and bonds between them.

From these two different ways of structuring objects, we observe two really different types of collection of objects. Objects can be brought together through a *part of* relation as in in the view-based model, or through an *isa*[2] relation as in the TMAC model.

[2]The term *isa* means "is a". It refers to a relation *is an instance of* or *is a kind of*, and corresponds to a

2.2.4 Actions structuring

The two previous sections are dedicated to proposals to structure respectively subjects and objects. Following this line of reasoning we focus on the need to structure actions. Indeed, in the IBAC models, actions usually correspond to elementary commands. The Task-Based Authorization Control (TBAC) model [Thomas and Sandhu 1994] suggests an interesting proposal regarding that issue.

The authors propose to address the integrity issue in information systems. This work mainly focuses on commercial transactions control. A transaction is composed of several tasks. Each task has to be validated by an "authorization function" in order to go on to the following one. Many issues are raised such as temporal dependencies between tasks and authorizations expiration. We rather focus on the abstraction of tasks and authorization functions. A transaction cycle is divided into "approval-steps" which can be seen as atomic steps. Approval-steps are for example "prepare-check" or "billing-approval". Then the TBAC model introduces the "authorization-task-unit" which is composed of approval-steps. An authorization-task-unit gets a data structure that consists among other things, of a name, attributes and approval-steps.

The TBAC model shows that the use of these kind of concepts allows us to make an abstraction of actions and to structure a set of atomic actions by considering composed action and dependency relations.

The TBAC model is refined in [Thomas and Sandhu 1997]. By analogy with the RBAC model, a TBAC model family – from $TBAC_0$ to $TBAC_3$ – is defined. With regards to our purpose, in other words, action structuring, $TBAC_1$ is the most interesting in that it deals with composite authorizations. It relies on the same principles as the first TBAC model. $TBAC_2$ integrates constraints, and $TBAC_3$ unifies $TBAC_1$ and $TBAC_2$.

The TBAC model has opened up the way to multiple contributions on that subject. It is often claimed that a lot of security models are only focused on object-subject oriented policies, and thereby more research activities should lead towards the definition of models based on actions, tasks or activities. Such models are sometimes called Activity-Based Access Control (ABAC) model [Oh and Park 2001b]. Activity-based models are highly related to research work on "workflow" modelling, like in [Thomas 1997, Cohen et al. 2002, Cholewka et al. 2000, Crampton 2004, Botha 2001]. In business and transaction activities, access control decisions may depend on specific sequences of events. The actions or tasks that users can perform are joined up into a global process, usually called workflow. Authorizations are granted to users according to the actual activated task in that process. We come back to that issue in section 2.3.

2.2.5 The Concept of organization

In the previous sections we presented some proposals which point was to improve the simple IBAC model. In this section we introduce the notion of "organization". Actually, this notion was explicitly or implicitly suggested in the previous models. In the RBAC model for instance, a user plays a role, that is, this user is empowered by a given organization to

specialization relation.

play that role. This organization considers that this user has the ability to accomplish the corresponding tasks, and chooses to grant him the associated authority and responsibilities. In fact, we might say that "the notion of role is an enterprise or organizational concept" [Thomas 1997]. In the RBAC model, the organization is not modelled since this model allows us to only express the access control policy of a single organization. The concept of organization is explicitly introduced in the administration model of RBAC [Oh et al. 2003]. We will come back to that in section 2.5.

We claim that two requirements lead to the introduction of the concept of organization in a security policy model. First, the organization modelling provides means to better structure the subjects, actions and objects involved in a policy. Second, in the framework of collaborative activities it should be possible to model organized groups. From this standpoint, an organization can be defined as a set of users that decide to work together in order to accomplish a given activity.

Let us consider the TeaM-based Access Control (TMAC) model which was first suggested in [Thomas 1997] as a preliminary TMAC model and was then completed in [Georgiadis et al. 2001] with the C-TMAC model. This model was mentioned above regarding the abstraction made of objects into object-types. Here we focus on the abstraction made of users and roles into "teams". This model is particularly adapted to collaborative environments.

TMAC is based on RBAC and adds an entity $Team$. If a specific activity is best accomplished through an organized group of users, then a team is created and some users are assigned to this team. Some roles are also assigned to this team in order to restrict the roles of the team members. On the other hand, some permissions are granted to the teams. In the end, users obtain permissions in accordance to the role played in the team. Furthermore, the TMAC model makes it possible to express the set of objects needed by a team to accomplish its task.

The TMAC model joins the two requirements mentioned earlier: through the notion of team, this model allows to specify an organization as a collection of users and a collection of objects. It also allows to distinguish the set of permissions granted to a user in accordance to his role, or to his role within a team.

Two kinds of organization can be distinguished: the "static" and the "dynamic" organizations. In a static organization the life time of the link which joins the user, the role and the organization are undetermined. For instance, a given user is employed by a bank as a counter clerk. In a dynamic organization, that link should last the time needed to carry out a specific action. In the TMAC model, teams are dynamic organizations: by analogy to the RBAC model in which a user activates a role in a session, in the TMAC model, a user activates a team in a session. In other words, the notion of team is highly binding to the notion of activity.

The following drawbacks of the TMAC model can be pointed out. The users' permissions are the result of the addition of the activated role permissions and the activated team permissions. Therefore, a user gains more permissions playing a role in a team than playing a role only. Rather, the combination of role and team should lead to a more restrictive or more specific set of permissions. For instance, a financial adviser involved in a team that aims at auditing an entreprise's accounts should receive specific authorizations when he is

working in this team, and not hold all the authorizations corresponding to the role financial adviser and this team.

The second drawback is in fact tied up to the first one. The bond between users, roles and team, is modelled using two binary relations: a user-role and a user-team relation. Since permissions are derived from the combination of a role and a team, we believe that a ternary relation between users, roles and teams would be more suitable. If we consider the example of the previous paragraph, a single relationship like *permission* for instance, should link a user, the role financial adviser and the team. This could be written as follows, where *audit* is the privilege:

- $permission(John, financial_adviser, team, audit)$

An interesting issue is not addressed in the TMAC model. Considering that a team is constituted to carry out an activity that could not be performed by a single role, the model should propose means to verify that all roles defined in a team are actually provided in order to grant the permissions to the users. In other words, it should be possible to test the completeness of a team. This issue is not addressed in TMAC. The recent team-based model presented in [Alotaiby and Chen 2004] is very similar to the previous ones, and does not resolve these drawbacks.

The Coalition-Based Access Control (CBAC) model [Cohen et al. 2002] is an interesting work as regards the notion of organization. In this model an entity organization is created and corresponds to the concept of "static organization". An organization is composed of roles, teams, tasks, resources, functions and users. Unfortunately, the bond between the concepts of user, role, team and organization remains unclear. Nevertheless, an interesting point should be emphasized. The CBAC model makes it possible to test the completeness of a team. This is specified by the constraint $ImplementedVia$. The following constraints mean that, in a bank for example, the task "close customer account" requires the roles "counter clerk" and "head agency":

- **ImplementedVia** (close_customer_account) = {counter_clerk, head_agency}

The notion of organization is implicitly present in several security models, and explicitly in other ones, like in CBAC, and in TMAC through the notion of team. These contributions show that the concept of organization is really useful, specially in the framework of collaborative environments and large scale organizations. This work has to be further investigated.

2.2.6 Multiple structuring

Throughout the previous sections we presented models that suggest means to organize and structure the basic entities *subject*, *action* and *object* of the first IBAC models. We examined solutions entity by entity. Actually some models enable several structuring at the same time.

We mentioned earlier that SQL provides means to define some views on tuples. In SQL/3, it is also possible to create roles. Therefore, SQL/3 enables us to grant permissions to roles on views. This might be called a View–Role-based access control (VR-BAC) model. The following instructions show how to create the role *counter_clerk* and how to assign users to this role:

```
CREATE     ROLE    counter_clerk
GRANT      counter_clerk  TO  John, Mary;
```

Using the view defined in the example presented in section 2.2.3, it is now possible to grant the role *counter_clerk* the authorization to consult the view *credit_account*:

```
GRANT      SELECT ON   credit_account  TO  Counter_clerk;
SET ROLE   counter_clerk;
```

The instruction "SET ROLE" is used to activate a role.

We shall now look at models that integrate role-based and activity-based features. A large number of research activities is lead in the field of workflow modelling. Some of them try to associate role-based features. The interesting part is how to link up roles and tasks. We have already mentioned the TMAC model. In this model the relationship between role (or team) and task is unclear, but it gives the possibility to grant authorizations applied to tasks and object-types and thereby offers a double abstraction facility.

We rather focus on the Task–Role-Based Access Control (T–RBAC) model. In [Oj and Sandhu 2000] permissions are defined as pairs of objects and access modes. Tasks correspond to abstract activities like "create an account". Permissions are assigned to tasks. One should keep in mind that tasks and access modes are two different and separate entities. Like in role-based models, users are assigned to roles, and finally tasks are assigned to roles. Figure 2.5 sums up this approach.

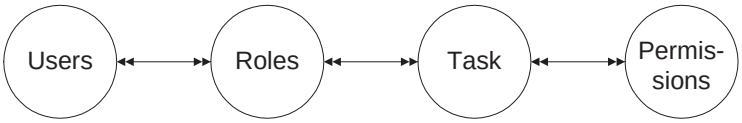


Figure 2.5: The T-RBAC approach

Elsewhere, workflows can be specified. Not all the tasks belong to the workflows. This model makes it possible to assign to roles both "single tasks" and tasks belonging to a workflow. In the first case, roles can always perform the tasks, in the second case roles can perform the task depending on the workflow state (see figure 2.6).

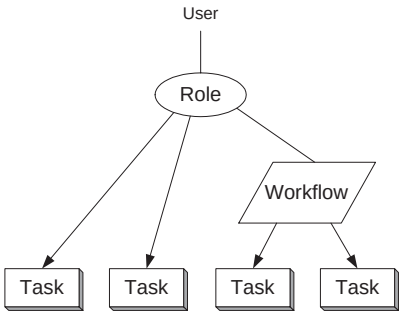


Figure 2.6: Access Control in the T–RBAC model

[Cholewka et al. 2000, Botha 2001] presents the Context-sensitive Access Control in Workflow Environments (CoSAWoEA) model. It addresses the same issue as T–RBAC in that it integrates role-based and task-based aspects. In CoSAWoEA, workflow processes are designed through the definition of several tasks which are part of a global process. Some roles are associated to each task. On the other hand, users get some roles, more precisely some deactivated roles. Users do not obtain any permissions from deactivated roles. In fact, a role is activated when a task using this role is effectively running, thereby users obtain the permissions assigned to that role. This can be compared to the role assignment in SQL, in the sense that the instruction "SET ROLE" has to be performed to activate a role.

The CBAC model incorporates elements of role-based, team-based and task-based models, and thereby become quite heavy and complicated. In fact, the authors suggest a family of CBAC models – basic, with teams, with tasks, with both of them. We rather focus on the last model $\text{CBAC}_{team+task}$. The relation between users, teams and tasks is expressed as follows:

- $\forall ta \in Tasks, \forall u \in Users, \mathbf{ContributesTo}(u, ta) \Rightarrow$
  $\exists tm \in Teams, \mathbf{CarriedOutBy}(ta, tm) \wedge \mathbf{AssignedTo}(u, tm)$

which means that a user who contributes to a task must be part of the team that performs this task. Unfortunately, there is no possibility offered to join a user, a team, a task and a role in the CBAC model.

The main point we wish to stress on is that more and more models include several kinds of entity abstractions. But it brings complexity, and specialization in the sense of application dependency. Most of these models are quite laborious and are often dedicated to a single and specific application area.

2.2.7 Hierarchy and inheritance

The previous sections where dedicated to entities structuring through the abstraction of the basic entities by introducing new high level entities such as roles, views, tasks, teams, etc. This part is dedicated to the study of hierarchies and associated inheritance mechanisms. Hierarchies provide a complementary solution to entity structuring since hierarchies are applied to high level entities. The use of hierarchy aims at mapping the access control entities structure on the organization structure. It makes it possible to design inheritance of authorizations based on the hierarchies, and therefore to mimic the authorization propagation in an organization.

Role hierarchy

The inheritance mechanism was suggested in object oriented programming as an efficient way to design an application in a modular way. From an organizational point of view, role hierarchies are a natural way of organizing roles in a way that reflects authority, responsibility, and competency in an organization. By analogy, a similar mechanism is commonly used in the access control policies based on roles. Indeed, role hierarchies are useful to structure the security policy specification. The RBAC_1 model [Sandhu et al. 1996] adds the concept of role hierarchies to the RBAC model. It is thus possible to organize a collection of roles using

specific links. A higher role in the hierarchy called "senior role", and a lower role is called "junior role". Let us consider the following example. In the bank $Trusted_bank$, the following roles are defined: *employee*, *counter_clerk*, *adviser*, $financial_adviser$, *insurance_adviser*, $chief_adviser$ and *head_agency*. Moreover, these roles are hierarchically organized as presented in figure 2.7. Role *employee* is the "junior-most role" and *head_agency* the "senior-most role".

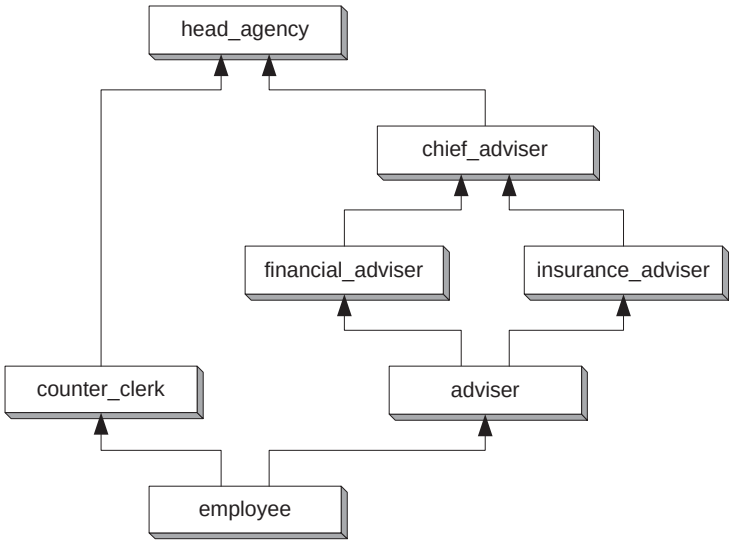


Figure 2.7: An example of role hierarchy

In RBAC_1, inheritance of permissions operates from a junior role to its senior roles. All permissions assigned to role *employee* are inherited by roles *counter_ clerk* and *adviser*. Moreover inheritance of permissions is transitive so in fact all roles in that hierarchy inherit from role *employee*. Role $financial_adviser$ only inherits the permissions of role *adviser* and passes on his own permissions to role *head_agency*. Hierarchies are defined as partial orders. They corresponds to reflexive, transitive and anti-symmetric relations.

Role hierarchies offers the facility to structure a set of role through the natural mapping of the organization structure and provides a convenient solution to simplify the permission management with the inheritance mechanism. We focused on role hierarchies, but object hierarchies can also be considered, as suggested in [Jajodia et al. 2001b, Bertino et al. 2003].

Scope of Inheritance

One could point out that senior roles get all permissions, and that there may be some cases where a role should not inherit from one of its junior role. To limit the scope of inheritance a simple solution consists in modifying the hierarchy. For example, if role *head_agency* must not inherit from role $financial_adviser$, it is enough to break the link between these two roles. This becomes more complex if only a subset of permissions must not be inherited by role *head_agency*. The RBAC_1 model suggests creating a "private role" $financial_adviser'$ as a senior role of $financial_adviser$. Then the SSO just has to assign to that new role the specific permissions that *head_agency* must not obtain. Although effective, this method can make the hierarchy become much more complex. Moreover, it does not solve the problem for the permissions of $financial_adviser$ which had been inherited from the role *adviser*:

it not possible to avoid these permissions to be inherited by the *financial_adviser*'s senior role.

This issue is also addressed in [Jajodia et al. 2001b]. The authors suggest labelling each node in the role hierarchy with a "plus" or a "minus". A "plus" – resp. "minus" – on a role indicates that this role can – resp. cannot – inherit permissions from its junior roles. Moreover this can be specified for each permission individually. The SSO can thus define fine-grained "propagation strategies".

Interpretation of hierarchies

The concept of role hierarchy is definitely helpful. However, it is not free of ambiguity. Some of these ambiguities are directly related to the concept of role itself as brought up above. In the example of figure 2.7, we can intuitively notice that the hierarchical relations between the roles are not equivalent to each other. For example, the bank hierarchy seems to indicate that a head agency is the counter clerk's boss, whereas a financial adviser is just an adviser specialized in finance. In [Moffet 1998, Moffet and Lupu 1999] the authors attempt to clarify the role hierarchy semantics. Three kinds of role hierarchies are identified: the *isa* role hierarchy, the *part of* hierarchy and the supervision hierarchy. Two issues arises from this point. First, one can wonder the relevance of a single hierarchy that mixes three kinds of hierarchical links. Second, this classification questions the legitimacy of inheritance of permissions: Does inheritance really make sense in any of these hierarchies?

The *isa* role hierarchy corresponds to specialization/generalization links between roles. For instance, the role adviser is specialized in two roles: financial and insurance adviser. This means that a user who plays the role financial adviser is an adviser. It thus seems reasonable that in the case of an *isa* link, permissions are inherited from the more generalized role to the more specialized role. In an access control model that includes prohibitions, it is also reasonable that prohibitions be inherited.

The supervision hierarchy corresponds to an authority relationship between roles. Roughly speaking, in such a hierarchy, a given role is its junior role's boss. In the above mentioned example, the user who plays the role *head_agency* is the *counter_clerk*'s boss. In such a hierarchy, a senior role is more powerful and thereby should inherit all supervision permissions of its junior roles. By contrast, supervision prohibitions might not be inherited. It would even make more sense that prohibitions be inherited from the senior roles to the junior roles. For example if a supervision prohibition is assigned to the role *counter_clerk*, it is intuitively more relevant to pass on this prohibition to the role *employee* rather than to the role *head_agency*. However, inheritance in such a hierarchy is a matter of choice, and mainly differs with respect to the considered application domain.

Finally, the *part of* hierarchy corresponds to activity aggregations. To be able to accomplish its purpose, that is, some given activities, an enterprise may have to define several sub-activities. In order to carry these sub-activities, some specific roles are created. In our bank example, the enterprise service department must include a role *financial_adviser* and a role *insurance_adviser*. Aggregation links are mostly "horizontal", as in our example, therefore the inheritance issue is not relevant.

This classification of role hierarchies shows that this issue is less simple than it seems. There is no relevant solution to this problem. By designing three parallel hierarchies, as suggested

in [Shen and Dewan 1992], we would lose the simplicity offered by a unique role hierarchy. In practice, this problem is left to one side. But copying the access control hierarchy from the organization hierarchy without a careful examination may lead to unexpected side-effects. Therefore, the role hierarchy has to be defined with the intention of obtaining the right permission heritage. In other words, the pursued inheritance determines the hierarchy, even though the resulting hierarchy mixes several types of links between roles.

The T–RBAC model provides an interesting solution in order to model several kinds of hierarchies at the same time. Two kinds of roles can be defined in it: the "job positions" and the "business roles". The job position corresponds to a supervision role and has to do with authority and management activities, whereas the business role corresponds to business work activities. There is only one hierarchy composed of these two types of role, but this hierarchy corresponds to a supervision hierarchy. Some restrictions are specified to design the hierarchy. For instance, a job position cannot exist between two business roles in the hierarchy and vice versa. The distinction between those roles comes along with the classification of the tasks into two categories. One corresponds to approval and supervision tasks which are declared as inheritable, the other one to private and workflow tasks which are declared as non-inheritable. The authors did not go into depth in this way. Indeed, there is no distinction between inheritance of these two categories of task. It would have been interesting for the business roles to inherit workflow tasks and the job positions to inherit supervision tasks. Without such distinction, the interest of the definition of two types of role is uncertain. It is indicated that it enables to limit the permission propagation in the hierarchy. Nevertheless a subtlety is introduced: all privileges that correspond to a "read" task are inherited whatever the category they belong to. This explains why the hierarchy is called supervision hierarchy: higher roles always gain the audit permissions of their junior roles.

With regards to our objective, which is to study different proposals to structure and organize the traditional access control entities, the hierarchies offer interesting perspectives. The role hierarchy presents a double level structuring: subjects are abstracted into roles, and roles are hierarchically organized.

2.2.8 Conclusion

We presented several models which suggest different means to structure the access control entities. This reduces the cost and management overheads associated with fine-grained security policy at the level of individual subjects, objects, and actions. The more information systems become complex and bigger, the more security models have to provide relevant solutions to structure their elements. This can be carried out through different kinds of abstraction like groups, or the function played in an organization (role), or how entities are arranged (file/folders), or the class/instance bonds (TMAC), etc. The notions of organization and hierarchies are really convenient since they make it possible to better model the running of organizations. Several models suggest multiple structuring, but none of them provides a complete and cohesive solution. Our objective in the next chapter is precisely to design such a solution.

2.3 Dynamic access control

2.3.1 Principle

So far, we have focused on different solutions aimed at structuring model entities. This section is dedicated to another major requirement. Indeed, in modern information systems, security policies cannot only be expressed by a set of simple authorizations without taking into account the context of the policy enforcement. Indeed, these policies are more and more complex. Thus, defining authorizations which are activated while the information system evolves, has become an important need. In most application areas, a security rule is not defined just with the subject, the action and the object, but also depends on the context in which the access is requested.

We mainly focused on static security models, this is, models in which permissions are definitive statements that can only be modified by a SSO. In a static model, it is assumed that authorizations are always activated independently of any other considerations, and thereby can be written as facts in a logic language or with n-uplets in a database. It is typically the case with the IBAC models. Actually, some models we presented in the previous sections may not be classified as static models, but we chose not to highlight their dynamic aspects as we intended to focus on abstraction and structuring. Therefore, we come back to some of these models in this section to examine how they manage dynamic authorizations. Please note that instead of "static" and "dynamic", the terms "passive" and "active" can be found in the literature. We prefer the term "dynamic" since the conditions that activate an authorization might be dynamic.

In dynamic security models, authorizations might be activated or deactivated in accordance with some events or with the information system state. The main idea is that authorization activation is conditioned by some circumstances that might evolve. Using such definition many models can be considered as dynamic models. Several terms are employed in this field to describe the authorization activation parameters, like condition, environment or context. We choose the general term of "context". We argue that authorization activation depends on the context.

The distinction between dynamic and static models may also be explained as follows. In static models, the decision to accept a security request, that is, to allow a user to access an object, is taken in accordance with the subject, the action and the object "attributes". Those models are sometimes called attribute-based models. For instance, a user's role or a file extension can be seen as attributes, and are used for the access control decision. By contrast, dynamic access control is not only based on the involved entities attributes but also on external parameters.

As the concept of context makes it possible to express different kinds of information, first we propose to classify them into four categories. Subsequently, we present the rule-based models that can be used to express some contexts.

2.3.2 Different kind of contexts

First of all, we should point out that a model is considered dynamic if the context may evolve in time. It carries an implicit notion of automatic and dynamic modification. Role-based

models could be viewed as dynamic models in that a user gains a given permission if he plays a role. In a certain sense, the assignment of a user to a role is a condition. However, the assignment is decided by a superior authority and does not change according to external events. In other words, the term "dynamic" refers to the fact that a policy dynamically evolves in accordance with the information system it is applied to.

Constraint

Constraints are a usual concept in access control policies. They are mainly used for administration tasks. For example, constraints in role-based models are used to specify the cardinality of a role (e.g., a given role can be played by more than two users). Constraints may also be useful to enforce the separation of duty principle (SOD).

Some models provide powerful language in order to specify expressive constraints which can be seen as dynamic conditions. This is the case of RCL (Role-Based Constraints Language) [Ahn and Sandhu 2000, Ahn and Shin 2001a] and OCL (Object Constraint Language) [Ahn and Shin 2001b].

Let us consider SOD in role-based models and first distinguish static SOD (SSOD) from dynamic SOD (DSOD). SSOD corresponds to the fact that conflicting users cannot be assigned the same role. Specifying such a constraint is the SSO's task. DSOD corresponds to the fact that conflicting roles cannot be activated by a single user in the same session. This last constraint might be seen as a dynamic factor that makes an access control policy becomes dynamic. Consider the case of a user who plays the role of financial adviser in a given bank and who is also a customer of this bank. This user should not be allowed to activate these two roles in a single session. Such a constraint may be viewed as a context in that it makes it possible to dynamically restrict the activation of roles.

Environment

Another kind of context corresponds to the security policy environment. Under this last term comes the system status, the time, the place where a user stands or a security request is made. In several models, like the UCON$_{ABC}$ model [Sandhu and Park 2004] such decisions factor are called "conditions".

The most obvious environment context corresponds to spatiotemporal information. [Covington et al. 2000, Covington et al. 2001] offers to broaden the role-based models by considering the security requests environment. In the Generalized Role-Based Access Control (GRBAC) model, the environment context is specified through a new type of role called "environment role", written "erole". Such a role makes it possible to express temporal conditions. The use of environment roles is quite simple. First, an *erole* must be declared. Then a temporal condition is associated to this *erole*. Let us consider the two following instructions:

- $erole(working_hours)$
- $role_rel(working_hours, 08:00 < time_of_day < 18:00)$

The first item corresponds to the definition of the environment role *working_hours*. The second item specifies that this role is activated between 8 am and 6 pm. On the other hand, permissions are not only assigned to roles, but also to a set of environment roles. Therefore, considering a given permission, a user obtains this permission if he plays the corresponding

role and if the corresponding *eroles* are activated. In other words, the activation of a permission depends on the activation of some environment roles.

The GRBAC model can also be used to model spatial contexts. Such contexts are defined and specified the same way as temporal contexts. Moreover the environment role hierarchy makes the contextual information management easier. In particular for temporal environment roles the fact that *erole*2 is a senior role of *erole*1 means that the time period condition of *erole*2 is included in the one of *erole*1.

Elsewhere, the Ponder policy language [Damianou et al. 2001] can also be used to express environment contexts. This declarative and object-oriented language is specifically designed for network policies in distributed object systems. The definition of context is done through the creation of filters. The previous temporal context is expressed as follows:

- **inst auth** filter1 {
  subject john; **target** account_n°21; **action** read;
  if (time.between("08:00","18:00")}

In [Bertino et al. 2000], the authors propose another extension of the RBAC model, the Temporal Role-Based Access Control (TRBAC) model. On the contrary to GRBAC where the time factor is applied to the permissions, in TRBAC it is only defined on roles, or role activation events. This model suggests a formal semantics to support periodic activations/deactivations of roles (e.g., roles which can access object o are only activated between 9 am and 6 pm) or thanks to a trigger (e.g., the role counter clerk is active whenever the role head agency is active). This model cannot express timing constraints applied to objects or actions.

Provisional context

In some cases, a user is authorized to carry out an action provided he accomplished some given actions beforehand. For example, a counter clerk is allowed to open a new customer account if he checked this customer's financial situation at the central bank. This last action is called a "provision". So provisions are those conditions or *preconditions* that have to be satisfied for an access control decision to be rendered. In other words, a permission is activated by the realization of required actions. This issue is addressed in [Kudo and Hada 2000, Jajodia et al. 2001a, Bettini et al. 2002].

[Bettini et al. 2002] provides a first-order language to write some provisional contexts. Let us consider the following example:

- $access(account, s, read) \leftarrow in(s, Counter_clerk), Register(s)$

This rule says that any counter clerk can read the customers' account provided that he has registered at the bank information system.

Using provisional contexts implies specific mechanisms. First, actions performed by users must be stored into an history log. Second, a module must be designed to have an access to the history and check whether an action has been realized yet or not. In [Jajodia et al. 2001a], this is realized in a the Provision-based Authorization program. When an access request is launched, it is passed on to a *provision evaluation module* that finds the weakest conditions under which the request is accepted. Then an *order specifica-*

tion module yields a set of conditions. Finally, a *provision verification module* verifies that the requested conditions were previously fulfilled by the user.

Workflow management

In most works regarding context, the context corresponds to the ongoing activities. Expressing such contexts is useful in the case of business and transaction activities. In these areas, the access control decision depends on specific sequences of events. In these ABAC models [Thomas 1997, Cholewka et al. 2000, Botha 2001, Cohen et al. 2002, Crampton 2004, Oh and Park 2003] a workflow is defined as sets of tasks and dependencies between these tasks. Thus, the workflow makes it possible to "filter" or to activate the privileges granted to the users. From this standpoint, ABAC models fall within the dynamic model category. The set of activated authorizations evolves while the workflow status passes from one step to the other.

ABAC models make it possible to take into account the need-to-know requirement, and the notion of just-in-time permission activation. The expression of a workflow environment through the context can also be considered as a means to respond to the principle of least privilege.

2.3.3 Rule-based models

The previous models had not defined language or defined too specific languages. We focus here on security policy expression frameworks based on rules. We commonly distinguish these models with the term "rule-based" access control (Rule-BAC) model. [Halpern and Weissman 2003, Bertino et al. 2003, Jajodia et al. 2001b] for example can be considered as rule-based models. Using a first order logic notation, authorizations are written as rules. In a simple access control model, these rules would appear under the following form:

- $\forall s \in Subject, \forall a \in Action, \forall o \in Object, condition \rightarrow permission(s, a, o)$

Modelling the access control rules in this manner is a natural and intuitive way to translate natural language rules into computable rules. *condition* allows us to express any type of conditions as long as the associated language provides the appropriate predicates. Therefore, in a rule-based model, there is no limitation in the expression of new conditions. As a consequence, there is also no structuring of these conditions.

Rule-based models, despite their name, are not access control models in the same way as RBAC, TBAC, etc. They are high-level models, in that they are policy expression frameworks that support multiple security models. They provide a language and a semantics rather than new concepts like roles or tasks.

[Halpern and Weissman 2003] provides a good example of such a formalization. The authors suggest an access control expression framework where a policy is a set of first-order formulas of the following form:

- $\forall x_1 ... \forall x_2 (f \Rightarrow \mathbf{Permitted}(t, t'))$,

where f is a first-order formula, t and t' are terms of the *Subject* and *Action* type respectively. Actually, an action brings together an access mode and an object. For example, the

following formula means that a counter clerk is allowed to consult companies accounts if all financial advisers are away:

- $\forall x, \forall y, \forall z(\mathbf{Counter_clerk}(x) \wedge \mathbf{financial_adviser}(z) \wedge \mathbf{absent}(z) \wedge$
  $\mathbf{Company_account}(y) \Rightarrow \mathbf{Permitted}(x, consult(y)))$

Using this kind of formalization makes it possible to express any kind of authorization rule, and thereby it can be used to express any kind of context. In the previous example, the specific part that corresponds to a context is the term $\mathbf{absent}(z)$ since it activates or deactivates the permission whether z is absent or not. Elsewhere, the authors show the limitation of such open formalization. Indeed the tractability may not be guaranteed. Therefore, a certain amount of restrictions must be respected for a policy to be solved in a low-order polynomial time.

2.3.4 Context enforcement

An expressive and convenient access control model must allow the expression of contexts. It is also important to be able to evaluate the contexts specified in a policy. For example, in the case of temporal context, the information system must provide the policy enforcement modules with the ability to consult the current time. As the implantation of policies is out of our research scope, we do not go into detail and just mention two examples of works related to this issue.

[Covington et al. 2001] describes a Context Toolkit able to collect the environment state using sensors and aggregators. It gathers information and forwards relevant ones to the applications. Through the Antigone Condition Framework (ACF), [McDaniel 2003] provides means to specify, implement and evaluate the context which is seen as a set of external conditions.

2.3.5 Conclusion

The notion of dynamic access control encompasses several types of contexts, namely constraints, environment, provisions and workflows. Some models allow the expression of high-level policies by allowing the specification of such contexts which act as conditions in order to activate and deactivate the security policy authorizations.

On the one hand, some models design well defined contexts, however they are limited to specific problems – like GRBAC or TRBAC. On the other hand, some languages provides powerful means to express any kind of context but without specifying any restrictive framework. It is thus appropriate to notice that the UCON models offer interesting solutions with regard to the context issue. We do not go into more details on usage control, yet we just call the reader's attention on the UCON$_{ABC}$ model [Sandhu and Park 2004, Park 2003] since it makes it possible to specify constraints, environment contexts and provisions in an expressive framework.

2.4 Negative authorizations and conflict management

So far, we considered models that enable us to express positive authorizations only. Some of the models we examined offer means to specify negative authorizations – also called prohibitions – but we chose to leave this matter aside until now.

Negative authorizations expression is another major requirement for the development of a flexible and powerful security policy model. We will first examine the reasons for that. Then we shall present different ways to model prohibitions. Finally, since specifying a security policy that includes both permissions and prohibitions may lead to conflicting situations, we discuss several solutions to detect and solve such conflicts.

2.4.1 Motivation

Negative authorizations are often ignored in access control. It is claimed that the absence of a positive authorization corresponds to a negative authorization. To clarify this point, let us first recall the definition of open and closed policies.

In a **closed policy**, a user is granted the right to access an object if there exists a corresponding positive authorization, otherwise this access is forbidden. By contrast, in an **open policy**, a user is denied the access to an object if there is a corresponding negative authorization, and is authorized otherwise.

One should notice that the concept of open/closed policy is different from the concept of positive/negative authorization policy. A positive authorization policy defines the actions that subjects are permitted to perform on objects. Therefore a positive policy is composed of positive authorizations only. A positive authorization policy is usually associated to the closed policy assumption. For instance, the RBAC model considers permissions only. By contrast, a negative authorization policy specifies the actions that subjects are forbidden to perform, and thereby is composed of negative authorizations only. Access control is traditionally based on positive authorization policies.

We claim that expressing negative authorizations is essential in order to specify high-level access control policies. The main question is thus: if a policy is only composed of permissions and if unspecified requests are forbidden, then why should we consider some negative authorizations, this is, some explicit denials of authorization?

Consider the following arguments:

- Using both permissions and prohibitions retains the natural way people express policies. If a SSO thinks about an important action that a specific user must not be allowed to carry out, he will prefer to explicitly specify a prohibition.
- In a large scale system, the audit of the policy is much faster if the permissions and the prohibitions can be monitored immediately.
- Negative authorizations can also be used to remove access rights from subjects if the need arises. In other words, prohibitions can be used as exceptions.
- In the case of inheritance due to the definition of hierarchies, the propagation of permissions can be limited by addition of prohibitions.

- If the security policy management is decentralized, that is, if more than one SSO administrate the policy without consulting each other, negative authorizations are really useful. Each SSO might not have a global vision of the whole policy and thus will not able to infer negative authorizations resulting from unspecified requests. Therefore, each SSO must be allowed to express negative authorizations.
- Many systems – like firewalls – support negative authorizations.

For these reasons, we integrate negative authorizations in the Or-BAC model. Let us first consider several solutions provided in the literature to express such authorizations.

2.4.2 Expression of negative authorizations

We shall look at different manners to express negative authorizations. Naturally, this depends on the chosen languages. We mainly focused on logic-based languages. However, let us first consider the Ponder language cited in section 2.3.2, as it represents a good example of object-oriented language. In such a language, expression of negative authorization is easy. One just has to create an authorization with **auth+**:

- **inst (auth+** | **auth-)** policyName {
  subject subject; **target** target; **action** action;}

The same kind of notation is used in [Lupu and Sloman 1999]. We examine now different ways to write negative authorizations in security policies based on logic languages. Many works on policy languages are based on the Authorization Specification Language (ASL) [Jajodia et al. 1997], like [Jajodia et al. 2001a] and the Flexible Authorization Framework (FAF) [Jajodia et al. 2001b]. ASL provides a complete first-order logic language. In ASL, given a set of action *Action*, a set of signed authorization types SA is defined as {+a, -a | a ∈ *Action*}. Then, an authorization is a triplet of the form:

- $(o, s, < sign > a)$

where $s \in Subject, o \in Object$ and $a \in Action$. This idea is also used in [Rabitti et al. 1991, Shen and Dewan 1992]. The prohibition "John is forbidden to read the file account12.pdf" is written: $(account12.pdf, John, -read)$.

More recently, [Bertino et al. 2003, Bertino et al. 2004] suggests an expressive framework, LAMP (LogicAl Multi-Policy), for reasoning about policies which give the possibility to express negative authorizations as well. As in ASL, positive and negative authorizations are distinguished by a sign. Authorizations are defined as follows:

- $Auth(O : object, S : subject, P : privilege, G : grantor, \varepsilon : string)$

where ε is in $\{+, -\}$. The previous prohibition is written as follows:

- $Auth(O : account12.pdf, S : John, P : read, G : SSO, \varepsilon : -)$.

FAF is based on Datalog [Ullman 1989] and LAMP on an object-oriented variant of Datalog, namely C-Datalog [Greco et al. 1992]. This provides tractable programs but forbids the use of functions and negations. By contrast, the first-order logic language suggested in [Halpern and Weissman 2003] and briefly presented above is not based on Datalog in order to avoid the previous restrictions. However, other limitations have to be laid down to get

tractable solutions. Since this language is based on rules, the restrictions are related to the conditions that conclude on authorizations. We just describe one of them here. These conditions are formulas. They must be ground literals. While this is enough to capture the information in databases for instance, it is not possible to express conditions like "All subjects playing role adviser..." since these formulas must be quantifier-free. The authors also define the specific situations where such a restriction can be relaxed and thereby specify more expressive conditions. In [Halpern and Weissman 2003], the expression of a negative authorization is seen as the negation of a positive authorization. A prohibition is expressed as follows:

- $\forall x_1 ... \forall x_2 (f \Rightarrow \neg \mathbf{Permitted}(t, t'))$,

where f is a first-order formula, t and t' are terms of the *Subject* and *Action* kinds respectively. The previous prohibition might be written as follows:

- $. \Rightarrow \neg \mathbf{Permitted}(John, read(account12.pdf))$

In [Cholvy and Cuppens 1997], the authors suggest a methodology based on SDL (Standard Deontic Logic) for analyzing consistency of security policies. Negative authorizations are expressed through a dedicated predicate F. Finally [Al-Kahtani and Sandhu 2004] introduces an extension of the RB-RBAC model called RB-RBAC-ve (RB-RBAC with negative authorizations). This model presents a noticeable difference. The negation is not applied to an action or an authorization, but rather to a role. Instead of receiving a negative authorization through the role he plays, a user is forbidden to play the role. In practice, this is expressed as a "negative authorization" of the form:

- $\mathrm{ae}_i \Rightarrow \neg\ \mathrm{r}_g$,

meaning that once a user satisfies the attribute expression ae_i, this user is not allowed to play role r_g. The point of presenting the negative authorizations this way is to be compatible with the RBAC model. But it provides a rough solution since it is not possible to give just one prohibition to a role, otherwise the definition of the roles must be revised.

2.4.3 Conflict management

We examine now different ways to detect and resolve conflicting situations between permissions and prohibitions. A conflict occurs when a positive and a negative authorization involve the same entities (e.g., the same subjects, actions and objects). These authorizations may be explicitly defined by the SSO, but in many cases, one or both result from the authorization propagation in a hierarchy, in a grouping or in a class/instance framework.

Conflict management is not an easy task, and there has not been much research made in this field. The way conflicts are managed is often called a *conflict management strategy*, or just *strategy*. Some strategies are simple but do not allow fine-grained decisions, some are complex but raise the tractability issue. The next subsection provides some strategies from the easiest to the most complex.

Simple precedence strategy

The easiest strategies are the well-known "Denials Takes Precedence" (DTP) and "Permission Takes Precedence (PTP)". In the first strategy negative authorizations are always

chosen when a conflict occurs. In PTP, positive authorizations always override negative authorizations when there is a conflict. One should notice that DTP – resp. PTP – makes more sense in a closed – resp. open – policy. In the case of DTP for instance, prohibitions can be considered as exceptions: the general policy is written using permissions, and exceptions using prohibitions. But it is not possible to specify exceptions to prohibitions.

Such strategies are deterministic: they guarantee that whatever the security policy is (or will be), no conflict can never happen. In the following, we say that a strategy that fulfils this property is "strong". By contrast, a strategy that might not resolve all conflicts is "weak".

DTP and PTP are not flexible as they do not allow specification of special cases, and thereby are only appropriate in specific situations. In particular, if negative authorizations are used as exceptions, or if they can only be specified by a most powerful SSO, then the DTP strategy is advisable. Let us consider the example presented in [Lupu and Sloman 1999].

- **R1 A-** @/users { reboot() } @/workstations
  Users are forbidden to reboot the workstations
- **R2 A+** @/users/sys_admin { reboot() } @/workstations
  System administrators are authorized to reboot the workstations

These policies only make sense if permissions take precedence. In this case, this means that administrators are *only* allowed to reboot the workstations.

Expressive access control models should not be restricted to these strategies. Indeed, the relevance of negative authorizations is bounded by the conflict resolution strategy. In Argos [Jonscher and Dittrich 1996] for instance, which provides a configurable access control system in the area of identity-based access control, only the DTP strategy can be applied. The solution in terms of conflict resolution in [Jajodia et al. 2001b] is also disappointing. The authors suggest a powerful and flexible support to multiple policies. However, the strategy to manage conflicts is "hard-coded" in the sense that there is no clear separation between the strategy for conflict management and the remainder of the policy specification. Moreover, the authors only consider four possible conflict management strategies: No conflict – in this case, a conflict is viewed as a constraint violation – DTP, PTP and nothing takes precedence – means that there is actually no conflict resolution. A final derivation rule guarantees that no conflict will persist, since this rule specifies that the lack of a positive authorization over a triplet corresponds to a negative authorization. But, this weakens the use of conflict management strategy.

Several models enables us to choose the strategy. DTP and PTP are usually proposed, like in [Al-Kahtani and Sandhu 2004, Lupu and Sloman 1999, Jajodia et al. 2001b] for example.

First/Last matching strategy

When a conflict occurs between a permission and a prohibition, a simple way to take a decision is to give precedence to the first, or the last matching authorization. Of course, this implies that authorizations are ordered. Most firewalls use this strategy. Firewall policies are good examples of policies mixing positive and negative authorizations. For instance, NetFilter adopts a first matching strategy, whereas IPFilter adopts a last matching strategy. In the framework of databases, [Shen and Dewan 1992] suggests several conflict management strategies, but in the last resort, implements a first matching strategy.

Strategies based on specificity

Many models consider hierarchies of entities associated with inheritance mechanisms. The result of this is the propagation of authorizations through the hierarchy, and above all, a heightened risk of conflict situations.

[Lupu and Sloman 1999] explains the notion of "distance" between an authorization and the entities it refers to. This notion can be interpreted in different ways, but the main idea is that the more an authorization is specific, the more it takes precedence. Let us consider the role hierarchy in figure 2.7. Assume that the role *employee* is not allowed to open the safe, and, on the other hand, the role $financial_adviser$ obtains the permission to open the safe. A conflict appears at the level of the role $financial_adviser$ since this role inherits from the negative authorization given to the role *employee*. However, the distance between the positive authorization and $financial_adviser$ is less than between the negative authorization and $financial_adviser$ since the permission is explicitly granted. As a consequence, the permission takes precedence. [Shen and Dewan 1992] use exactly this interpretation of the distance with role hierarchies.

Strategies based on weak and strong authorizations

[Rabitti et al. 1991] introduces the notions of "strong" and "weak" authorizations. One should observe that these two terms are used in different meanings as above. A strong authorization overrides a weak authorization whereas a strong authorization cannot be overridden. Strong and weak authorizations are inherited through the hierarchy. It is possible to combine strong/weak authorizations with positive/negative authorizations. For example, strong negative authorizations can be used as exceptions in a hierarchy of weak positive authorizations. In fact, this is not completely true since strong authorizations are inherited, and thereby cannot be assigned to only one entity of the hierarchy. [Bertino et al. 1996] is based on the same idea but defines more clearly the conflict management. Conflicts may occur between two strong authorizations or two weak authorizations. If there are conflicts between strong authorizations, the policy is *inconsistent*. In fact, the authors assume there are no such conflicts as long as the semantics of strong authorizations is always respected. They only provides a mechanism to insert in a consistent policy new strong authorizations without creating "strong conflicts". In the case of weak conflicts, the default strategy is DTP. However, the SSO can add a new authorization. A conflict between two weak authorizations has a resolution if it is possible to specify an authorization overriding one of these authorizations. So, these two models offer a quite limited conflict management strategy.

Strategies based on priorities

The distinction between strong and weak authorizations may be viewed as a two priority levels strategy, where "strong" is higher than "weak". Therefore, one can imagine the definition of strategies based on explicit priorities assigned to authorizations. In [Bradshaw et al. 2003] for example, each authorization is associated with a value. The authorization of higher priority takes precedence on the authorization of lower priority. In the case of equality, the SSO is required to specify which authorization takes precedence.

Such priorities are difficult to manage since it is not easy to assign priorities that relate to the importance of the authorizations. Moreover, as stated above, conflicts remain in the

case of equal priorities. However, strategies with explicit priorities may be useful in specific policy frameworks to avoid most conflicting situations.

Parametric strategy

Many conflict management strategies can actually be defined. The selection of a strategy depends on the model, language, semantics and application area. Furthermore, it is convenient in some cases to mix different strategies. Therefore, the best solution would be to enable the SSO to define his own strategy. [Bertino et al. 2003] suggests a logical framework that makes it possible to express several kinds of policies. It is associated with a "parametric conflict resolution policy". The SSO can exploit any information to specify the strategy. Unfortunately, no strategy language expression is provided. So, let us consider an example of strategy written in natural language.

- The authorization with the most-senior role takes precedence;
- otherwise, the authorization with the most-senior grantor takes precedence;
- otherwise, denial takes precedence.

In this framework, the SSO can create or mix strategies, and may test the resulting policy for each one. However, the semantics is not free of ambiguity. In conflicting situation, an authorization rule is kept, and the other one is "discarded". It appears that the discarded rule is merely deleted from the policy, which might lead to the so called "drowning" problem: in relation to other conflicting situations, the discarded rule could have taken precedence over another one. Therefore, it modifies the policy without any control. This issue is explained with the following example. Authorizations are specified using a simplified language.

- $Auth(o, Dev_1, write, +)$
  "Members of group Dev_1 are granted permission to perform action *write* on object o"
- $Auth(o, Dev_2, write, -)$
  "Members of group Dev_2 are forbidden to perform action *write* on object o"
- $Auth(o, John, read, +) \leftarrow Auth(o, Dev_1, write, +)$
  "John is granted permission to perform action *read* on object o provided members of group Dev_1 are granted permission to perform action *write* on object o"

Assume that Ann is a member of groups Dev_1 and Dev_2. Therefore, Ann obtains the two first authorizations. As a consequence, a conflict occurs between these authorizations. If the first one is discarded for instance, thereby all members of group Dev_1 are loosing the authorization to perform action *write* on object o. John also loose the authorization to read object o. This might not be what was expected.

Other strategies

[Halpern and Weissman 2003] designs a first-order logic language for security policies. As indicated before, the authors choose not to rely on Datalog to avoid some of its restrictions. A theorem guarantees that as long as those restrictions are respected, the policy is consistent, and as a consequence, no conflict can occur. So there is actually no conflict management strategy.

In [Cholvy and Cuppens 1997], a security policy is modelled using modal logic, where permissions, prohibitions and obligations are represented using deontic modalities. This provides a richer model in which it is possible to specify, for instance, disjunctive obligations

or conjunctive prohibitions. However, this paper only suggests managing conflicts using priority between roles. In [Cuppens et al. 2001], the approach is refined and the concept of strategy to manage conflicts is introduced. However, this strategy is used to define priority between roles and its specification is separated from the remainder of the policy specification.

Finally, [Benferhat et al. 2003] suggests an approach based on possibilistic logic to handle conflicts in prioritized security policies. The priority is implicitly derived from the format of rules. This is used to construct a stratified policy efficiently.

2.4.4 Conclusion

Negative authorizations expression is an important requirement in the design of powerful and flexible policies. Indeed, most modern access control models and languages provide such authorizations. However, the expression of prohibitions must be associated to a conflict resolution strategy. As we presented, these strategies can be very simple or more complex, they can guarantee the resolution of all conflicts or just decide for some of them. We claim that strategies should be parametric in order to let the SSO choose or design his own strategy.

Some issues have not be addressed yet, since they are not treated in most models. First, in accordance with the strategy, and in the case of hierarchies, some authorizations may become useless. We call them "redundant authorizations". Let us consider the strategy with weak/strong authorizations and the role hierarchy of figure 2.7. If a weak positive authorization to open the safe is granted to the role $financial_adviser$, and if the equivalent strong negative authorization is assigned to the role *employee*, then the first positive authorization is always overridden. The positive authorization never applies and is thus redundant. This problem is cited in [Rabitti et al. 1991] but no solution is suggested. Second, in the context of dynamic access control, some conflicts might be detected in the policy without occurring during the policy enforcement. We call it the "potential conflict" issue. If we have two authorizations, a negative and a positive one, involving the same entities, but which depend on different contexts, then a conflict only occurs provided the contexts of these authorizations are valid at the same time. It would be useful to be able to determine whether a conflict will happen or not, and what is more, to know in which situations it might occur. In chapter 7, we suggest solutions for these two issues.

2.5 Administration models

So far, we described some major requirements: entities structuring, contexts and negative authorizations expression. However an access control model would not be complete without addressing the administration issue. In this section we first explain what administration is made of, and the reasons for which it has to be taken into account.

2.5.1 Introduction

The administration consists of creating and maintaining the policy entities such as users, actions, objects, groups, roles, etc., and authorizations. Specifying the security policy is the first administration task. Some specific users must have the authority to deal with it.

However, it is not sufficient to define the policies at once. Some mechanisms for updating all the elements of the policy must be defined, in order to maintain the policy in accordance with the information system. For instance, if a new user is employed by an organization, it should be possible to add this user, grant him some authorizations, and above all, preserve the policy consistency. Without strong administration procedures, the quality of a policy is degraded as the information system evolves.

Administration consists in distributing authorizations in order to update the security policy. It must be distinguished from the entities structuring such as the definition of roles. The role-based permission assignment enables us to reduce the number of operations, when a new user is employed. Compared to an IBAC model in which authorizations are given at the user level, role-based models decrease the administrative overhead. Therefore, entities structuring increases the security management scalability, but does not solve the administration issue.

A distinction can be made between two types of administration. Administration tasks can be performed by one or several administrators, also called system security officers (SSO). SSO corresponds to a specific function with specific authority and responsibility. In many organizations, SSO is a full-time job. Administration tasks can also be taken on by basic users. Indeed, an organization can choose to delegate fully or partially the administration activity to its users. It is known as "delegation". Delegation also encompasses the situation where a user grants some of his authorizations (or all of them) to another user.

The importance of administration was ignored for a long time in the field of access control research activities. It has become a major concern over the past few years. In many contributions, the administration issue is addressed as an inevitable aspect of access control, like in [Botha 2001, Bertino et al. 1996]. However only a few models are associated with a complete administration model. The most famous administration model is ARBAC (see section 2.5.3).

Subsequently, the first subsection is dedicated to the mandatory access control. In the second subsection we discuss the administration of role-based models, and the last one deals with delegation. Discretionary access control models are well-known. We broach these models only from the delegation point of view, in the last section.

2.5.2 Mandatory Access Control

In a Mandatory Access Control (MAC) policy, a clearance is assigned to each subject, and a classification is assigned to each object. The administration of a MAC policy consists of managing the ressource classification and the subject clearance. A unique authority first defines the criteria used for the classification and the clearance, and then determines the security level for each subject and object. It is a fully centralized administration model in which the subjects have no rights to administer. The MAC administration is very rigid. What is more, the transposition to IT systems is complex especially regarding networks. Finally, this administration mode does not seem to be convenient for commercial needs [Ferraiolo et al. 1993]. Regarding the administration of a MAC policy, the main problem is the object classification management. Since the classification rules are often difficult to apply, automatic support tools are suitable to assist the users when choosing the initial classification of an object. The classification of an object is not static but evolves during its lifetime, and leads to the downgrading of the initial classification when the object content

becomes less sensitive. In this case, tools that manage the object classification and apply rules to automatically downgrade object classifications become useful. The general principles for such tool were presented in [Cuppens and Gabillon 1996].

See also [Carrère et al. 1999] that presents the design and implementation of SACADDOS, a support tool to automatically manage object classification. Finally, in [Solworth and Sloan 2004], the authors suggest a MAC administration model called SPBAC (Security Property Based Administrative Controls). Objects are associated to labels. The information flow is controlled with the relation $mayFlow$: $mayFlow(l, l')$ means that it is possible to write an object with label l' after having read an object with label l. This aims at ensuring confidentiality. For integrity, another relation, $didflow$, is added. Unfortunately, the description of this model is quite vague. Furthermore, the suggested method involves a great amount of information related to each user's accesses and thereby generates a significant processing overhead.

The administration of MAC shows an interesting aspect of a security policy administration as it is fully centralized and thereby rigid.

An organization may not want to delegate its administration prerogatives to any users. This does not mean that all administration tasks have to be held by a unique SSO, or a single authority. Implementation of a security model and high security needs encourage us to look for a compromise. In this case, the RBAC administration model turns out to be an interesting option.

2.5.3 Administration of role-based models

The Role-Based Access Control (RBAC) model consider the role as a central concept. The ARBAC model is dedicated to the management of RBAC policies. This model is a recognized solution for decentralized administration. It authorizes administrative roles by means of "role ranges" and "prerequisite conditions".

ARBAC

ARBAC97 [Sandhu et al. 1997] (Administrative Role-Based Access Control) is the first RBAC administration model. ARBAC has three main features. First, it provides the possibility of administrating an RBAC policy in a decentralized way, but without loosing control over rights' propagation. Second, ARBAC is an RBAC auto-administration model. Third, though the administrative roles and permissions are based on RBAC, they are completely separated from the regular roles and permissions. ARBAC97 provides three sub-models:

- URA97 [Sandhu and Bhamidipati 1997]: This model describes how to assign users to the predefined roles. The assignment by an administrative role of a user to a regular role is based on a ternary relation "can_assign" between the administrative role, the prerequisite roles and the regular role. A member of an administration role can assign a user to a regular role. It can be specified that this user must be a member of one or several given roles (prerequisite roles) in order to be assigned to this regular role.
- PRA97 [Sandhu et al. 1999]: This model is the dual of URA97 and it describes the assignment of permissions to roles. It is also based on a ternary relation "can_assignp" with prerequisite conditions.

- RRA97 [Sandhu and Munawer 1998]: This last model proposes rules for the role-role assignment, that is, the construction of the role hierarchy.

ARBAC97 offers a proper administration model which is not exactly the case for the other security models. In order to obtain a decentralized administration of an RBAC policy, ARBAC could be used this way: the management of the role hierarchy and the assignment of the permissions are carried out by a centralized authority on the one hand. On the other hand, the assignment and the revocation of users can be left under the responsibility of the chiefs of the different departments or units, through the assignment of these chiefs to administrative roles. However, it is noteworthy to point out the following shortcomings. ARBAC is not a real auto-administrated model since it does not use the RBAC permissions but rather creates new assignment and revocation rules (such as can_assign and can_revoke) used by the administrative roles.

As we mentioned before, the assignment relation is ternary. Thus the prerequisite conditions depend on the administrative role and the regular role. However, it seems that the prerequisite conditions generally only depend on the regular role and should rather be considered as a constraint on the regular role. Moreover, ARBAC does not give any information on the creation of the roles, and does not offer any delegation mechanism.

ARBAC does not offer means to express contextual conditions. Thus, it is not possible to specify that a given administrative role is allowed to assign a permission to a regular role only at working hours or only from his own terminal. This kind of restriction can be useful to detect misuses of power for instance.

ARBAC extensions

Two ARBAC extensions have been proposed. With ARBAC99 [Sandhu and Munawer 1999], the authors present a way to manage the *mobile* and *immobile* users and permissions. Unlike a mobile user, an immobile user can be seen as a non-permanent user such as a user under training, a visitor, a consultant, etc. In this case, the user can be a member of a role and get the corresponding permissions. But an administrative role cannot use this membership to put the immobile user into other roles. That is, an immobile user cannot move up through the hierarchy. The same idea is used for immobile permissions.

The objective of ARBAC02 [Oh et al. 2003] is different. Several drawbacks of ARBAC97 have been pointed out. With ARBAC02, some improvements have been proposed to resolve, among others, the multi-step user assignment which generates a lot of work for the SSO and which causes redundant tuples in the User-Role Assignment (URA) management. The main modification made in ARBAC02 affects the prerequisite conditions for the user and the permission assignment. An organization structure of user pools and an organization structure of permission pools are created. The first one is managed by the human resources group, the second one by the IT group. We obtain two hierarchies which are independent from the role hierarchy. User and permission assignment are made by the security officers by picking users and permissions in these pools, thereby simplifying the assignment processes.

These two extensions are interesting but do not resolve the shortcomings we have just mentioned. Moreover, ARBAC02 simplifies the assignment process, but transfers the problem of the prerequisite conditions onto the human resources group and the IT group.

SARBAC

SARBAC (Scoped Administration of Role-Based Access Control) [Crampton and Loizon 2002] is an extension of RHA4 [Crampton and Loizou 2003] and an alternative to ARBAC97. SARBAC relies on administrative scope which changes dynamically as the role hierarchy changes. Thus, update operations over RBAC96 and SARBAC relations become easier and can not lead to inconsistent rules. In particular, SARBAC makes it possible to delete a role without any restriction. Unlike in ARBAC97, it is possible to assign administrative roles to users since SARBAC does not make any distinction between regular and administrative roles.

A-ERBAC

[Kern and Moffet 2003] presents an administration model dedicated to the Enterpriser Role-Based Access Control (ERBAC) model. It is focused on large scale policies and thereby on a widely decentralized administration. Like in ARBAC, it distinguishes regular and administrative permissions and roles. All permissions are seen as pairs $< operations, objects >$ like in the NIST Solution [Ferraiolo et al. 2001]. This model suggests a more traditional approach than ARBAC, since it is based on ACLs. Each administrative role is associated to a *scope*, that is a part of the organization. It obtains administrative permissions, which are valid within its scope, and which are defined as a conjunction of an operation (*view, insert, change,* or *delete*) and an object (*User, User-Role Assignment, Role, Role-Role Assignment,* or *Role-Permission Assignment*). For instance the permission $< insert, User\text{-}Role\ Assignment >$ means that the administrative role is allowed to assign a user to a role. There is no possibility offered to express more specific authorizations. This administration seems to be robust, and has been tested on large systems. However, its expressiveness is rather limited.

Graph-based

We mention some graph-based approaches. Such approaches provide user-friendly notations and may be used to design effective tools. However we prefer language-based approaches which offer means for consistency verifications and logic programming.

[Koch et al. 2002] presents a formalization of RBAC using graph transformations. This graphical technique provides means to specify the ARBAC operations, as the *can_assign* relation and the prerequisite conditions. [Koch et al. 2004] extends graph-based administration to the SARBAC model. It enables us to graphically model the notions of scope and constraint.

In [Oh and Park 2001a], the authors suggest an administration method for T–RBAC. Through a semi-automatic process, this method makes it possible to design the T–RBAC model from the enterprise model using graph transformation. It offers interesting solutions to design a new policy, but does not address the administration issue once the transformation process is performed.

2.5.4 Delegation

Delegation is a difficult issue in the field of access control, and only few works are dedicated to this point. It is difficult to even define the concept of delegation. As a broad definition,

delegation is the process whereby a user without any administrative prerogatives obtains the ability to grant some authorizations. The user who delegates an authorization is the "grantor", the user who receives this authorization is the "grantee". It is generally admitted that a user can only delegate authorizations he has himself, or authorizations on objects he owns. Delegation is usually related to the notion of ownership. Some models, like Ponder, make it possible for a user to grant an authorization that he does not possess. In this case, the distinction we have made between administration and delegation is thin. Afterwards, we assume that one can only delegate something one owns.

Delegation is also defined as granting an authorization on one's behalf. In other words, the grantee is delegated a specific authority and responsibility [Goh and Baldwin 1998]. As a consequence, the grantor loses the authorization he delegates, that is, the corresponding activity cannot be carried out by the grantor and the grantee at the same time. If this property is respected, [Bertino and Ferrari 1997] talks about "transfer" rather than delegation.

Actually, many properties related to the notion of delegation can be studied. We just stated two of them in the previous paragraph. [Goh and Baldwin 1998, Barka 2002] suggest other characteristics. As delegation is not our main focus, and could be the subject of an entire thesis, we shall not go into further detail. One can just mention three other points: can a grantee delegate the authorization he receives? Who is allowed to revoke an authorization that has been delegated, the grantor or the SSO? Does it make sense to delegate a negative authorization? etc. In the following section, we discuss the discretionary access control model, then the role-based delegation model.

Discretionary Access Control

Discretionary access control (DAC) models are fully decentralized. In the context of IBAC models, the administration tasks are a set of operations that are used to update the access matrix. In most implementations, right management is based on the notion of object ownership: if a user owns an object, he is allowed to give another user rights to access it. Thus, the main authority fully delegates the management of the user's rights. The administration activity is reduced to its minimum. The main consequence is the risk of losing control over the rights' propagation [Jones et al. 1976]. Moreover, since access is distributed at the discretion of the object owners, there is the risk that end-users sets of rights become uniform.

Some models have been proposed in order to obtain tractable solutions to control the consequences of the cascaded grants, like the Take-Grant protection model and the Schematic Protection Model [Sandhu 1988]. The TAM (Typed Access Matrix) model [Sandhu 1992] is close to DAC but introduces strong typing of subjects and objects. It guarantees a decidable and monotonic solution.

Administration based on delegation has been widely used. For instance, [Bertino et al. 1996] applies a DAC approach in database systems.

Role-based delegation model

The role-based models suffer from the impossibility to express delegation as the ability to grant an authorization to a user. Let us use our previous bank example, and assume that John plays the role *adviser* and that the head agency wants to delegate him one of his permissions. The head agency cannot delegate directly to John since the role-based

model imposes the role as an interface between permissions and users. If the head agency delegates a permission to the role adviser, John receives it and so do all the users member of role *adviser*, which is not what the head agency wants. The Role-Based Delegation (RDBM) model [Barka and Sandhu 2000] was conceived in order to solve this problem. In RDBM, instead of delegating his permissions, a user delegates the role he plays, and with it all the permissions it carries. RDBM has a major characteristic: during the time a user has delegated his role, he cannot play this role. RDBM also addresses the problem of revocation, that can be activated with a time-out, or by any member of the delegated role. In [Barka and Sandhu 2004], the authors extend the RDBM by considering the role hierarchy. [Moffet and Lupu 1999] also discuss the delegation inheritance in accordance with the semantics associated to hierarchy. The authors also point out what delegation inheritance has in common with rights' propagation in DAC models.

The RDBM model does not seem to be an adequate manner to deal with delegation. Delegating roles may be convenient when a user is absent. In all other cases, the solution suggested is too rough. For instance, if the head agency needs to delegate the permission to open the safe to an adviser, he has to delegate his role. During that time, the head agency cannot assume his functions neither his authority.

2.5.5 Conclusion

To be complete, an access control model has to include a full set of administration procedures. It is easier to administrate a well-built administration model. Indeed entities structuring reduces the security administration overhead.

We claim that administration tasks should be addressed in a separated but fully compliant model. A flexible administration model must provide means to design either centralized or decentralized administrative procedures. Small policies are better administrated by a single trusty SSO, whereas large scale policies must be taken in charge by a set of SSOs to whom the administration activities are distributed.

Elsewhere, delegation is a convenient manner to make administration more flexible. It is a natural means of decentralizing control, and the way in which most organizations work in practice. Delegation also enables fine-grained security administration. But it must be controlled tightly to guard against hazardous rights propagation.

Finally, the more administration tasks can be automated, the less errors and malicious actions will happen. From this stand point, [Al-Kahtani and Sandhu 2002] suggests an interesting solution. The automatic user-role assignment based on attributes reduces the manual interventions on the system.

2.6 Chapter conclusion

Throughout this chapter, we have presented many access control models with more or less details. Above all, we studied some essential requirements in order to design a new, efficient and convenient access control model. In this chapter, we looked into the four following requirements: entities structuring, dynamic authorizations, negative authorizations and conflict management, and administration.

From the simple IBAC model based on the entities triplet $< subject, action, object >$ we studied several suggestions to bring more abstraction. We introduced the role-based, the activity-based and the view-based models; each one offering the possibility to model at a higher level one of the three entities. In our new model we attempt to federate them within a single model. We showed that hierarchies provide powerful means to manage a security policy, and to mimic the organization structure. Entities structuring implies, explicitly or implicitly, the notion of organization. Therefore we argue in favor of the creation, in a new model, of an entity "organization".

The second requirement corresponds to the necessity to express dynamic security rules in order to obtain a dynamic access control model. Such models are more expressive, and allow to specify a security policy that sticks better to information system evolutions. A dynamic authorization may be viewed as a set of conditions, this is the context, that concludes on an authorization. The context can be used to express the system state, the current workflow state, some provisions, etc. Unfortunately, each dynamic model is focused on one type of context.

The use of negative authorizations in addition to positive authorizations is a useful and natural way to write policies. However, negative authorization modelling must go with conflict management mechanisms. We presented several ones, but none of them satisfy all the requirements. A model should provide parametric conflict management strategies in order to specify simple or complex strategies, and deal with redundant rules and potential conflicts as well.

Administration becomes an important field of investigation in the area of security policy research. It is essential, indeed, to provide administrative procedures to update the policy along with the organization and the information system evolution. We claim that administrative activities should be specified through an auto-administered model independent from the policy model (that is optional) but fully compliant with it. Such an administration model should also take into account the delegation issue.

Afterwards, we propose a new access control model, called Or-BAC, which meets these four requirements. Chapters 3 and 4 are dedicated to the definition of the Or-BAC model based on the conclusion we have just drawn about entity structuring. The next three chapters are the subject of the other three requirements. In chapter 6 we show how the Or-BAC model offers a convenient solution to express several types of context. We focus on the conflict management issue in Or-BAC in chapter 7. Finally, we go on to the definition of an administration model in chapter 8.

Of course, some other issues should be studied. Let us consider two of them now. First, some recent models make it possible to express obligations. Obligations state actions that must be fulfilled by users, after a given event, or before another one. Obligation expression raises many problems, such as the verification that obligations are not violated. This issue is tackled in [Cuppens et al. 2005].

Second, we left aside the flow control models. Afterwards we assume that processes that work on the behalf of users correspond to trustworthy applications. We are aware that it is a strong assumption, and thereby future works should be done in the direction of flow access control.

Chapter 3

The Or-BAC model

3.1 Introduction

The previous chapter was an occasion to browse a large scale of existing security policy models and to define the requirements for the definition of a new, expressive and convenient security model.

In this chapter we introduce the main features of the *Organization-Based Access Control* model [Or-BAC 2003, Cuppens and Miège 2004c]. This model stems from the work carried out in the MP6 project framework[1] (Modèles et Politiques de Sécurité pour les Systèmes d'Informations et de Communications en Santé et Social). MP6 is a RNRT project (Réseau National de Recherche en Télécommunications) funded by the French Ministry of Research. It ended in November 2003.

Through the Or-BAC model we attempt to make good use of the existing model advances, but we also try to overcome their limitations. We aim at proposing a model that offers means to structure the security policy in such way that it is possible to define complex and flexible policies that match IT systems' reality. In the previous chapter we brought up four major requirements. The first one is the necessity to provide solutions to structure the security policies components. We concluded that a new model should attempt to federate the role-based, activity-based and view-based models, and should introduce the concept of organization as well. The working-out of our model is based on these objectives. Actually, we integrate three high-level components *role*, *activity* and *view* as abstraction of *subject*, *action* and *object*, in order to design implementation-independent policies.

Furthermore, the concept of organization is brought in as the central component of our model. In this manner, the policy specification is completely parameterized by the organization so that it is possible to handle simultaneously several security policies associated with different organizations. In other words, the security policy does not directly apply to subjects, actions and objects. Instead, it defines authorizations that apply within an organization to control the activities performed by roles on views. This is our main focus in this chapter.

In the previous chapter, we also mentioned that the modelling of hierarchies provides convenient solutions to structure the set of entities and the set of authorizations defined in

[1] http://www.telecom.gouv.fr/rnrt/rnrt/projets/res_01_59.htm

a security policy. We examine hierarchies and define inheritance mechanisms in the next chapter at section 4.1.

The second and the third requirements raised in the previous chapter respectively correspond to the context and the negative authorization expression. The Or-BAC model fulfils these requirements. In order to give a complete overview of the model, we briefly touch on these issues, but they are fully examined later on. Chapter 6 is dedicated to context expression, and chapter 7 to negative authorization expression and conflict management.

In this chapter, we give a semi-formal presentation of Or-BAC. Chapter 5 is dedicated to the formal presentation and tackles the decidability issue. We introduce our model using a diagrammatic language based on the entity-relationship model as it is more convenient to introduce the concepts used in Or-BAC. We then express the security rules with the first-order logic syntax [Shoenfield 2001]. Indeed, a relationship can ben viewed as a fact, and an entity as a term. We call "Or-BAC policy" any security policy based on the Or-BAC model.

The reader should be aware our objective is to design a security policy model. We do not attempt to examine how a policy based on this model should be implemented.

Section 3.2 to section 3.4 are dedicated to the definition of the model entities. Section 3.6 gives the main derivation rule and an overview of the model. All the relations and rules defined in this chapter and the following ones are summed up in the tables in appendix A.

3.2 The concept of organization

In the Or-BAC model, we introduce the concept of organization. Hence, the first entity we insert in Or-BAC is the entity *Organization*. This is the most important one. It gives its name to our model. The main objective is to offer a convenient way to structure the whole security policy.

We first defined an organization as an organized group of active entities, that is, subjects playing some role or other [Or-BAC 2003]. That group of subjects does not necessarily correspond to an organization. More precisely, the fact that each subject plays a role within the organization corresponds to some agreement between the subjects to form an organization. Following this definition, an organization can be a company, an institution, or any group of users like a project or a team. Nevertheless, the notion of organization in the Or-BAC model is quite different from the notion of group of users as it is usually interpreted in access control models.

We later refined this definition in order to obtain a more accurate and general one [Cuppens et al. 2004a]. An organization is now defined as any entity in charge of the security policy. In other words, any association of subjects that encompasses a security policy and is in charge of enforcing this policy can be considered as an organization. This notion can be extended to security components for instance. In chapter 9, we see that in the context of a local network, a firewall is viewed as an organization.

[Grossi and Dignum 2004] suggests the following definition of organization: "set of agents with specific roles, private and common objectives, the activities of which are procedurally

determined". Although interesting, we prefer to restrict this definition to the scope of the security policies.

Definition 3.2.1. Organization
"An organization is an association of subjects that is in charge of defining and enforcing the security policy applied to the subjects. The set of organizations defined in an Or-BAC policy is written *Org*".

The organization is a parameter of all the security rules of the policy. In this manner, we can manage simultaneously several policies associated with several organizations. It also offers means to make organizations cooperate better as long as they specify compatible security policies.

As we shall see in section 4.1.4, Or-BAC makes it possible to break down an organization into several sub-organizations. Thus, we can define an organization hierarchy. If we consider a bank called $Trusted_bank$, it is most likely that this organization is divided into departments, and the departments into units for example. The different agencies of this bank may also be viewed as sub-organizations. Or-BAC allows to model such structures.

Using inheritance mechanism, if a security policy is defined for an organization, we can express that the sub-organizations may inherit of that policy. In each sub-organization, it is possible to add specific authorizations such as the lower we are in the organization hierarchy, the more specific the security policy is. This is explained in section 4.1.4.

Before describing how authorizations are modelled within the Or-BAC model, we give the definition of some other entities. All the entities presented in the following sections are always defined in a given organization. Hence the entity organization is central in a security policy based on the Or-BAC model.

3.3 The model entities

The Or-BAC model does not give up the traditional entities *Subject*, *Action* and *Object* used in most access control models. It builds an expressive model instead, using these entities. The set of subjects – resp. actions and objects – are written S – resp. A and O.

The static authorization triplet $< subject, action, object >$ is suited for traditional environments and applications but is less appropriate to meet requirements of the rising systems and their applications and so we introduce three new entities described in the three following subsections. Our objective is to bring in more abstractness in order to hold a large number of policies and to obtain stable policies for a long period of time, as suggested in [Grossi and Dignum 2004]. We also aim at specifying general policies, that is, policies independent from the implementation issue, that might be implemented in several organizations.

3.3.1 Subjects and roles

Subject

The entity *Subject* is used differently from one security model to another. In the Or-BAC model, the entity *Subject* is considered as the traditional active entity. Therefore, a subject

can be either a user, i.e. a human, or an organization. In fact, we shall assume that the set of organizations is a subset of the set of subjects, so that $Org \subseteq S$. Examples of subjects therefore include users such as $John$, $Mary$, $Peter$, etc., or organizations such as $Trusted_bank$ or its financial department called $Trusted_finance$.

In other respects, we do not address the flow control issue. Programs are considered as reliable, and thereby are treated the same way as users. In order to avoid confusion, we restrict the set of subjects to users and organizations, and do not consider programs in this dissertation.

As in role based access models, we chose to consider a new entity *Role* as an abstraction of entity *Subject*.

Role

The concept of role was explained in section 2.2.2. In Or-BAC, the entity *Role* is used to structure the link between subjects and organizations.

$\mathcal{R}$ is the set of roles defined in a policy. We should remind the reader that the notion of role is different from the notion of "group". A group consists of a set of users whereas a role consists of a set of authorizations. Assigning a subject to a role amounts to grant a set of privileges to this subject.

The role is an organizational concept. In other words, a role is defined within an organization. To model such a relation we introduce the relationship *Relevant_role*:

Definition 3.3.1. Relation Relevant_role
"*Relevant_role* is a relation over domains $Org \times \mathcal{R}$. If *org* is an organization and r a role, then $Relevant_role(org, r)$ means that r is a relevant role in organization *org*".

Therefore, a role can be considered as relevant in several organizations. The relation is useful in particular with respect to the hierarchies and the associated inheritance rules that we study in section 4.1.2. Since subjects play roles in organizations, we need a relationship that joins up these entities together: the relationship *Empower* (see figure 3.1).

Definition 3.3.2. Relation Empower
"*Empower* is a relation over domains $Org \times S \times \mathcal{R}$. If *org* is an organization, s a subject and r a role, then $Empower(org, s, r)$ means that *org* empowers subject s in role r".

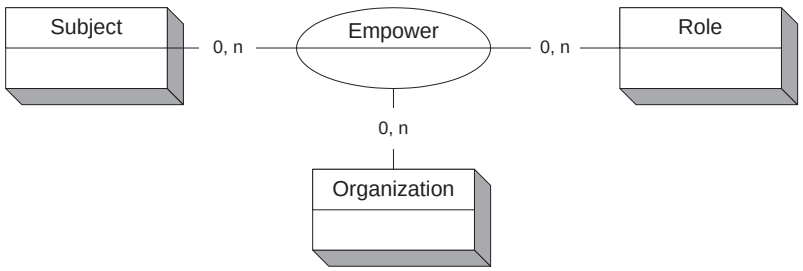


Figure 3.1: The *Empower* relationship

Unlike the TMAC and the RBAC models which consider binary relations between organizations and subjects or between subjects and roles, notice that our model considers a ternary

relation between organizations, subjects and roles. Thanks to this ternary relation, we are able to specify that a given role carries different sets of authorizations according to the organization in which this role is defined. In the context of our bank example, a subject can play the role counter clerk in two banks and get two different sets of authorizations for instance.

Example

To illustrate this, let us look at the following examples. We assume that roles *counter_clerk* and $financial_department$ are relevant roles in organization $Trusted_bank$. This is expressed this way:

- $Relevant_role(Trusted_bank, counter_clerk)$
- $Relevant_role(Trusted_bank, financial_department)$

The two following relations correspond to the fact that organization $Trusted_bank$ empowers some subjects in these roles:

- $Empower(Trusted_bank, John, counter_clerk)$
  "$Trusted_bank$ empowers John as a counter clerk".
- $Empower(Trusted_bank, trusted_finance, financial_department)$
  "$Trusted_bank$ empowers organization $trusted_finance$ as its financial department".

3.3.2 Objects and views

Object

In our model, the entity *Object* will mainly cover inactive entities such as data files, e-mails, printed forms, etc. In the banking domain, we can consider for example objects like customer accounts or company accounts. We assume that the set of subjects is a subset of the set of objects, so that $S \subseteq O$. By means of the entity *Role*, we are able to structure the subjects and to update easily the security policies when new subjects are added to the system. Since we will also have to structure the objects and to add new objects to the system, we believe that a similar entity regarding objects is necessary: the entity $View$.

View

Roughly speaking, as in relational databases, a view corresponds to a set of objects that satisfy a common property. This explains the reason why we chose the term "view". In practice, there is no more possible comparison between views in the relation databases area and in the Or-BAC model. A view in a relation database results from an operation made on tables, and above all, is just a temporary set of elements used to build more complex SQL requests. In the Or-BAC model, once a view is defined in a policy, it is considered as an element of this policy, as it is for a role. A view can also be defined as the "role played" by an object, or a set of objects, within a given organization. This idea was already introduced in the GRBAC model [Covington et al. 2000]. However, we prefer to define a view from the access control standpoint only:

Definition 3.3.3. View
"A *view* is an access control entity defined within an organization, and used to put together objects to which apply the same authorizations".

$\mathcal{V}$ is the set of views defined in a policy. The view, like the role, is an organizational concept. Therefore we introduce a new relation, $Relevant_view(org, v)$, that brings together a view with an organization.

Definition 3.3.4. Relation Relevant_view
"$Relevant_view$ is a relation over domains $Org \times \mathcal{V}$. If org is an organization and v a view, then $Relevant_view(org, v)$ means that v is a relevant view in organization org".

In a file management system of a bank, for instance, the view *customer_account* corresponds to the customer accounts whereas the view *company_account* corresponds to the company accounts. Seeing that views characterize the ways objects are used in organizations, we need a relationship that links together these entities: the relationship Use (see figure 3.2).

Definition 3.3.5. Relation Use
"Use is a relation over domains $Org \times O \times \mathcal{V}$. If org is an organization, o is an object and v is a view, then $Use(org, o, v)$ means that org uses object o in view v".

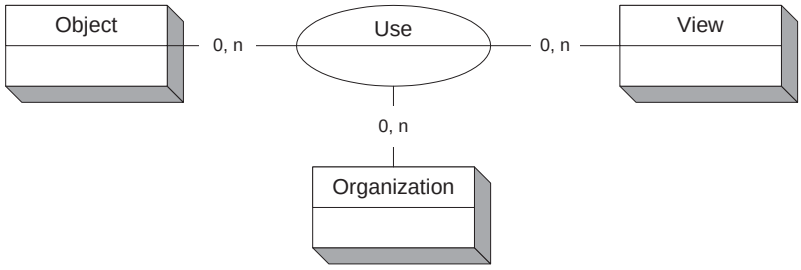


Figure 3.2: The Use relationship

We wish to draw the attention on the fact that our model considers a ternary relation between organizations, objects and views. Our aim is to make ourselves able to characterize organizations that give different definitions to the same view.

Example

Take the case of the view *customer_account* relevant in organization $Trusted_bank$ and $Gold_bank$, and defined as a set of Excel documents in $Trusted_bank$ and as a set of XML documents in $Gold_bank$:

- $Use(Trusted_bank, customer_12.xls, customer_account)$
  "$Trusted_bank$ uses object $customer_12.xls$ as a customer account".
- $Use(Gold_bank, customer_15.xml, customer_account)$
  "$Gold_bank$ uses object $customer_15.xml$ as a customer account".

3.3.3 Actions and activities

Action

Security policies specify the authorized accesses granted to active entities on inactive entities and regulate the actions carried out in the system. In our model, the entity *Action* will

mainly contain computer actions such as *read*, *write*, *send*, etc. In the Or-BAC model, the set of actions A is a subset of the set of objects O, such that $A \subseteq O$.

Activity

Following the line of reasoning suggested in section 3.3.1 where subjects are abstracted by means of roles, a new entity will also be used to abstract actions: the entity *Activity*. An activity is a task, whereas an action is the way this task is implemented in the organization. As for the notion of view, an activity can be seen as the "role played" by an action, or a set of actions, within an organization. Intuitively, an *activity* is an operation that can be fulfilled by some actions defined in this organization. However, we define the entity *Activity* from the security policy point of view only.

Definition 3.3.6. Activity
"An *activity* is an access control entity defined within an organization, and used to put together some actions on which apply the same authorizations".

$\mathcal{A}$ is the set of activities defined in the policy. The activity, like the role and the view, is an organizational concept. Therefore we introduce a new relation, *Relevant_activity*, that joins up an activity with an organization:

Definition 3.3.7. Relation Relevant_activity
"*Relevant_activity* is a relation over domains $Org \times \mathcal{A}$. If org is an organization and a an activity, then $Relevant_activity(org, a)$ means that a is a relevant activity in organization org".

In our model, activities like *reading*, *writing*, *consulting*, etc., will be of the utmost interest. Since different organizations may decide that a single action comes under distinct activities, we introduce a new relation that will be used to join up the entities *Organization*, *Action* and *Activity*: the relationship *Consider* (see figure 3.3).

Definition 3.3.8. Relation Consider
"*Consider* is a relation over domains $Org \times A \times \mathcal{A}$. More precisely, if org is an organization, α is an action and a is an activity, then $Consider(org, \alpha, a)$ means that org considers that action α falls within the activity a".

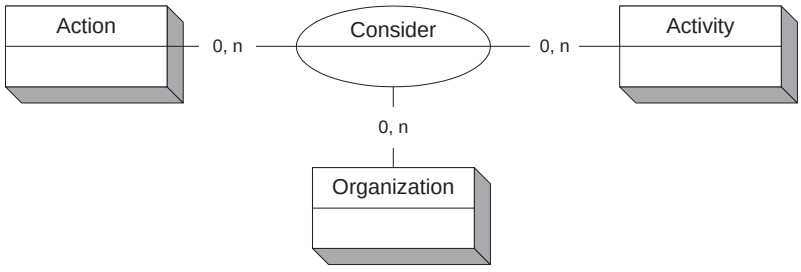


Figure 3.3: The *Consider* relationship

Since *Consider* is a ternary relation, different organizations may decide that a single action comes under distinct activities or that different actions come under the same activity. What we have in mind is to be able to characterize organizations that structure similar activities in different ways.

Example

We could consider, for instance, the activity *consulting* which is relevant in organizations $Trusted_bank$ and $Gold_bank$. In the first bank, this activity might correspond to an action *read* run on data files whereas it might correspond, in the other bank, to action *select* performed on relational databases:

- $Consider(Trusted_bank, read, consulting)$
  "$Trusted_bank$ considers *read* as an activity consulting" and
- $Consider(Gold_bank, select, consulting)$
  "$Gold_bank$ considers *select* as an activity consulting".

3.3.4 Attributes

We introduce attributes in the Or-BAC model. Attributes enable us to express further information about the entities involved in a security policy. This is indeed convenient to specify entity attributes in order to design expressive rules. More precisely, as shown in the following example, attributes can be used to design automatic assignment. Furthermore, attributes will be used to design contexts (see chapter 6).

In Or-BAC every object may have some attributes. Since $S \subseteq O$ and $A \subseteq O$, attributes can actually be assigned to the concrete entities *subject* and *action*. Let us consider the following relation and assume that $account_12$ is an object:

- $client_type(account_12, company)$

This relation corresponds to an attribute "client type" and means that object $account_12$ is of the type *company*. Thanks to this attribute, we are able to express the following rule: "every object used in the view *account* having attributes *client_type* equal to *company* is automatically used in the view *company_account*. This is modelled with the following rule:

- $\forall s \in S,$
  $Use(Trusted_bank, s, company_account)$
  $\leftarrow Use(Trusted_bank, s, account) \wedge$
  $client_type(s, company)$

It might be useful for example to create an attribute *account_balance* in order to automatically assign credit accounts in a given view and debit accounts in another view. With respect to subjects, one might want to indicate the age or the gender of the customers. Afterwards, we use, among others, the following attribute examples:

- $account_number(number, account)$ where *account* is an account and *number* is a number account.
- $account_owner(subject, number)$ where *subject* is a customer and *number* is an account number.

The definition of attributes depends of the application domain of the security policy. This is the reason why we do not specify predefined attributes.

3.3.5 Organizational authorization

Using the entities and the relationships introduced in the previous sections, we are now in a position to define security policies applying to organizations. Figure 3.4 shows the abstraction made of the entities *Subject*, *Action* and *Object* into the entities *Role*, *Activity* and *View*. We call "concrete entities" the first three entities and "organizational entities" the last three entities.

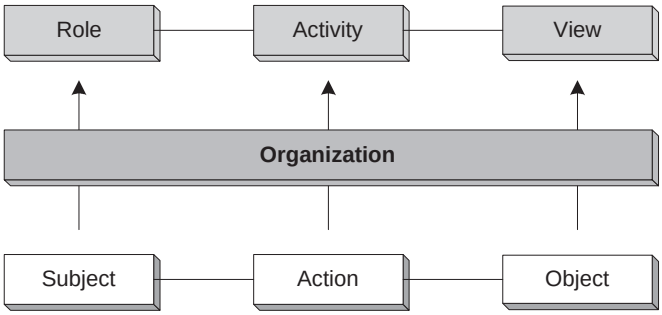


Figure 3.4: Abstraction of the traditional access control entities

We have introduced almost all the Or-BAC model entities so far. It is time now to consider the way we model authorizations. What we have in mind is to extend our model with two new relations *Permission* and *Prohibition* for the purpose of being able to join together organizations, roles, views and activities and thereby expressing authorizations. The first relation corresponds to a positive authorization, the second relation to a negative authorization.

As indicated in section 2.4, the use of explicit negative authorizations in security policies is often discussed, mainly for two reasons. Firstly, if we go on the closed policy assumption, explicit negative authorizations might seem to be useless. Secondly, some conflicting pair of positive and negative authorizations may appear in a policy which enables to express positive and negative authorizations at the same time. The negative authorization expression, together with our proposal to manage, detect and solve conflicts, are explained later on. Actually, chapter 7 is exclusively dedicated to these issues. From now and until that chapter, we might omit prohibitions when definitions and properties apply to prohibitions in the same way as they do to permissions. We explicitly distinguish prohibitions from permissions if needed.

Here is the definition of relation *Permission*. Relation *Prohibition* is defined in the same manner.

Definition 3.3.9. Relation Permission (first definition)
"*Permission* is a relation over domains $Org \times \mathcal{R} \times \mathcal{A} \times \mathcal{V}$. More precisely, if org is an organization, r is a role, v is a view and a is an activity, then $Permission(org, r, a, v)$ means that organization org grants role r the positive authorization to perform activity a on view v".

Inasmuch as such permissions – resp. prohibitions – handle organizational entities (roles, activities and views), they are called "organizational permissions" – resp. "organizational prohibitions".

Example

Let us take the case of bank $Trusted_bank$ that grants role $financial_adviser$ permission to perform activity *creation* on view *customer_account*. This is expressed by the following fact:

- $Permission(Trusted_bank, financial_adviser, creating, customer_account)$

Otherwise, the following fact:

- $Permission(Trusted_bank, counter_clerk, consulting, company_account)$

says that bank $Trusted_bank$ grants role *counter_clerk* permission to perform activity *consulting* on view *company_account*.

So far, the introduction of the entity organization and the three organizational entities allow to express high level security rules. Let us consider again the previous permission which shows the benefit of such modelling. With only one security rule, $Trusted_bank$ can express that in all agencies and whatever IT system is used, all counter clerks are allowed to realize any action that falls within activity *consulting* on any company account.

3.4 Modelling contexts

As presented in chapter 2, an expressive policy model should provide means to define dynamic security policies, that is, policies which may depend on some specific circumstances. More precisely, it should be possible to make each security rule constrained by the policy context. In order to address this issue, we propose to extend our model with a new entity: $Context$. Roughly speaking, this entity will be used to specify the concrete circumstances in which organizations grant roles some permissions to perform activities on views.

Let us first consider the following permission:

- $Permission(Trusted_bank, customer, consulting, customer_account)$
  "$Trusted_bank$ allows its customers to consult the costumer accounts".

Thanks to this privilege, the bank can express that its customers are granted the right to consult the bank accounts. On the other hand, it is clear that this security requirement is not exactly what $Trusted_bank$ wants to specify: a customer should be allowed to consult only his own bank account. The truth of the matter is that the Or-BAC model described above simply cannot cope with such security requirement: given an organization, users inherit permissions from the roles they play in that organization. The use of the context entity will allow the bank to express the privilege it wants.

Definition 3.4.1. Context
"A *context*, within an organization, is a set of constraints that must be satisfied in order to activate an authorization".

In this section, we introduce briefly the entity $Context$ in order to get a global vision of the Or-BAC model. In chapter 6, we go into further detail about the context definition and the different contexts Or-BAC enables to model.

$\mathcal{C}$ is the set of contexts defined in the policy. The context is an organizational notion. A context is actually defined within an organization. In order to model this link, we introduce a new relation *Relevant_context*:

Definition 3.4.2. Relation Relevant_context
"*Relevant_context* is a relation over domains $Org \times \mathcal{C}$. If org is an organization and c a context, then $Relevant_context(org, c)$ means that c is a relevant context in organization org".

As explained in chapter 6, which is dedicated to contexts in the Or-BAC model, each context is seen as a ternary relation between subjects, objects and actions defined within such or such organization. Therefore, entities *Organization*, *Subject*, *Object*, *Action* and *Context* are linked together by the relationship $Hold$ (see figure 3.5).

Definition 3.4.3. Relation Hold
"$Hold$ is a relation over domains $Org \times S \times A \times O \times \mathcal{C}$. If org is an organization, s is a subject, o is an object, α is an action and c a context, then $Hold(org, s, \alpha, o, c)$ means that within organization org, context c is true between subject s, action α and object o".

The conditions required for a given context to be linked, within a given organization, to subjects, objects and actions is formally specified by logic rules (see section 6.3).

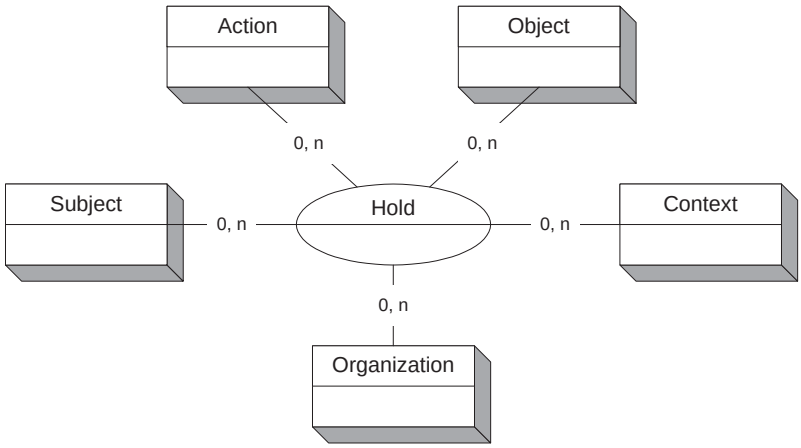


Figure 3.5: The $Hold$ relationship

In order to model contextual authorizations, we have to modify the relation *Permission* as presented at figure 3.6.

Definition 3.4.4. Relation Permission (final definition)
"*Permission* is a relation over domains $Org \times \mathcal{R} \times \mathcal{A} \times \mathcal{V} \times \mathcal{C}$. More precisely, if org is an organization, r is a role, v is a view, a is an activity and c is a context then $Permission(org, r, a, v, c)$ means that organization org grants role r the positive authorization to perform activity a on view v in context c".

Example

Let us get back to our previous example. $Trusted_bank$ can define a new context, called $personal_account$:

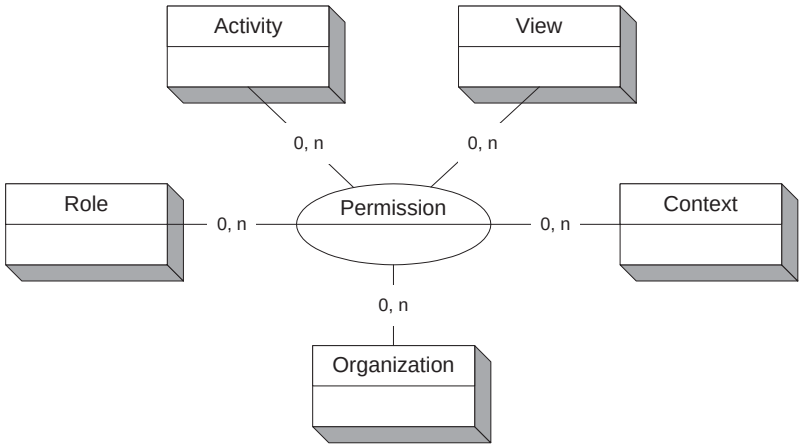


Figure 3.6: The *Permission* relationship

- $\forall s \in S, \forall \alpha \in A, \forall o \in O$
  $Hold(Trusted_bank, s, \alpha, o, personal_account)$
  $\leftarrow account_number(number, o) \wedge account_owner(s, number)$

The definition of context *personal_account* is explained at section 6.3. Using this context definition the bank is able to fully express the privilege corresponding to the permission granted to a customer to consult his personal bank account:

- $Permission(Trusted_bank, customer,$
  $consulting, customer_account, personal_account)$.

3.5 Concrete permission

The relationship *Permission* enables a given organization to specify permissions granted in a given context. Such permissions correspond to a relation between roles, views and activities. However, down-to-earth access control must provide a framework to describe the concrete actions that may be performed by subjects on objects. For the purpose of modelling concrete permissions, we introduce in our model the new relation $Is_permitted$ as a relationship between subjects, objects and actions (see figure 3.7).

Definition 3.5.1. Is_permitted
"$Is_permitted$ is a relation over domains $S \times A \times O$. If s is a subject, o is an object and α is an action then $Is_permitted(s, \alpha, o)$ means that subject s is permitted to perform action α on object o".

We define in the same way the relationship $Is_prohibited$ which corresponds to a negative authorization.

Our relationship $Is_permitted$ is similar to the notion of permission suggested in the IBAC models (see section 2.2.1). There is, however, a difference of major importance. In IBAC models, each authorized triplet $< s, \alpha, o >$ must be explicitly stated. In our model, triplets that are instances of the relationship $Is_permitted$ are logically derived from organizational permissions granted to roles, views and activities by the relationship *Permission*. This is modelled by a general rule RG_1 presented in the following section.

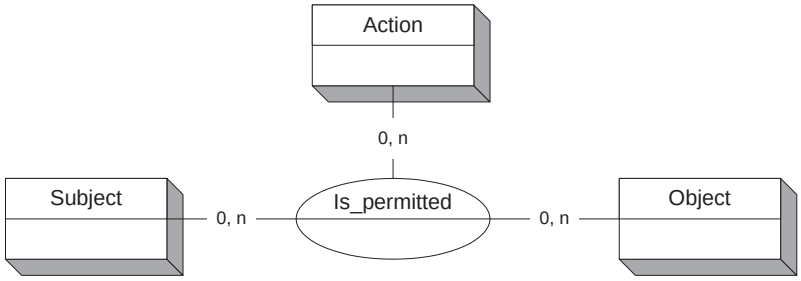


Figure 3.7: The *Is_permitted* relationship

Example

Let us consider the following concrete permission example:

- $Is_permitted(John, ATM.consult, account_n°428)$
  "John is allowed to consult the bank account n°428 in an ATM machine."

Explicit instances of the relationship *Is_permitted* may also be viewed as exceptions to the general security policy specified by the relationship Permission. Let us consider the following example. Usually subjects who are empowered as counter clerk are not allowed to consult the company accounts. Therefore, no organizational authorization is stated. However, and whatever could be the reason, if the bank wants to grant the permission to a specific user, for example *Paul*, to consult a specific object company account, for example $society12.act$, it can add the corresponding concrete authorization:

- $Is_permitted(Paul, read, society12.act)$

Thereby, this concrete authorization can be viewed as an exception to the organizational policy. This is called an "explicit concrete authorization". Actually, this notion is close to the one of delegation. In chapter 8 we discuss the way explicit concrete authorizations can be used as delegation authorizations. Furthermore, the use of such concrete authorizations relies on how the Or-BAC policy is enforced: explicit concrete authorizations should have priority over derived concrete authorizations.

3.6 Security policy

In this section we give an overview of an Or-BAC security policy. As we introduced earlier, an Or-BAC policy can be seen as a two level security policy: on the one hand an organizational level compound of organizational entities – *Organization*, *Role*, *Activity*, *View*, *Context* – and of organizational authorizations – *Permission*, *Prohibition* – and on the other hand a concrete level compound of concrete entities – *Subject*, *Action*, *Object* – and of concrete authorizations – *Is_permitted*, *Is_prohibited*.

These two levels are related as follows. In a given organization *org*, a subject s is permitted to perform an action α on an object o if:

i. s is empowered to play a given role r in *org* and

ii. α implements a given activity a in *org* and

iii. o is used in a given view v by org and

iv. a context c holds in org between s, α and o, and

v. the organization org grants to role r the permission to perform the activity a on the view v in the context c.

If these conditions are fulfilled, the request made by the subject s to perform the action α on the object o is accepted. This is modelled by the following derivation rule which makes it possible to derive concrete positive authorizations from organizational positive authorizations:

- RG_1: $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Permission(org, r, v, a, c) \wedge$
  $Emplower(org, s, r) \wedge$
  $Consider(org, \alpha, a) \wedge$
  $Use(org, o, v) \wedge$
  $Hold(org, s, \alpha, o, c)$
  $\rightarrow Is_permitted(s, \alpha, o)$

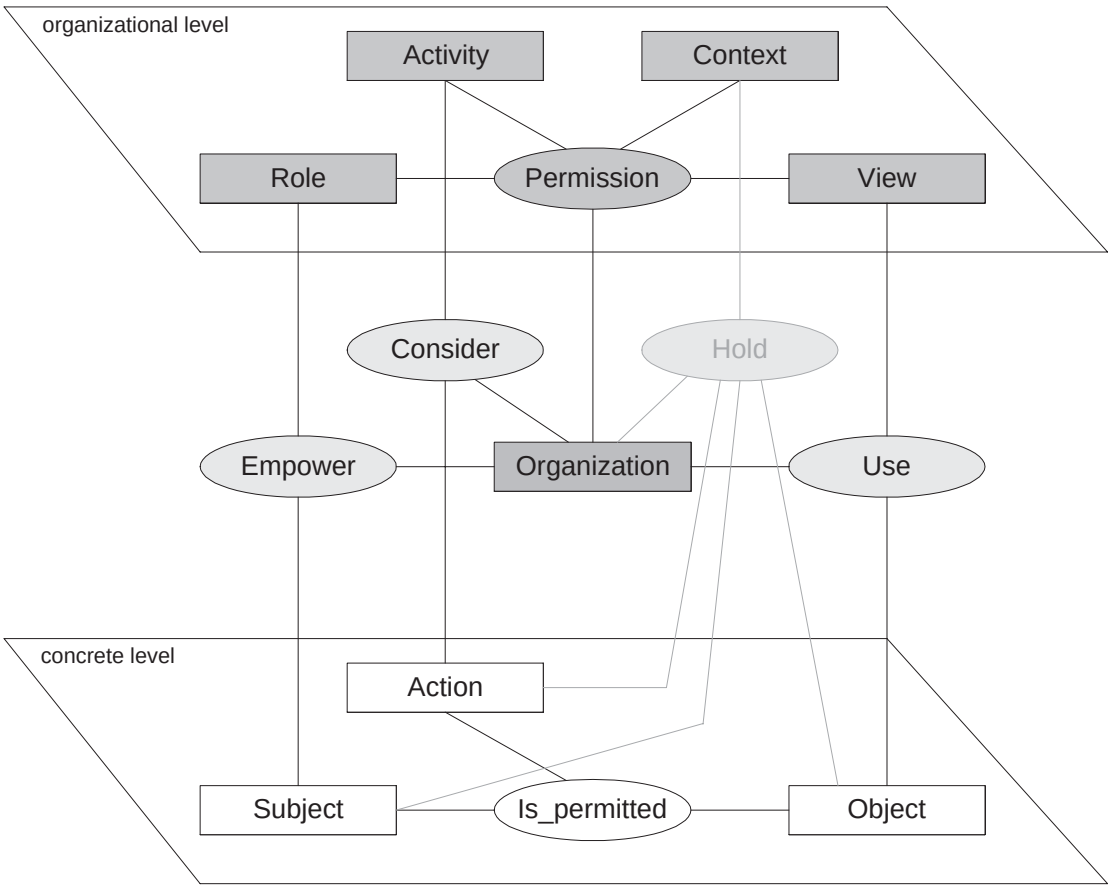


Figure 3.8: The Or-BAC model

The rule RG_2 that enables to get the concrete negative authorizations from the organizational negative authorizations is stated in the same manner as rule RG_1. Figure 3.8 resumes the Or-BAC security model. To give a comprehensive description of Or-BAC, relations *Prohibition* and *Is_prohibited*, as well as the relevance relations, are absent of figure 3.8.

As indicated in the introduction of this chapter, we do not aim at considering the implementation of an Or-BAC policy. Nevertheless, we suggest that concrete permissions are security rules dynamically derived. On the one hand, organizational permissions compose the written security policy, on the other hand, concrete permissions are viewed as security requests made by subjects. In this perspective, the derivation rules previously stated are only used when security requests are launched. The specific decision process in case of the use of negative authorizations is discussed in chapter 7.

Example

Let us consider the following example where John, a $Trusted_bank$ customer, wants to consult his cash account balance on an ATM machine. The corresponding security request made after John has logged on the machine could be expressed as follows:

- $Is_permitted(John, ATM.consult, account_n°428)$

John is a bank customer:

- $Empower(Trusted_bank, John, customer)$

We assume that John's account number is 428:

- $Use(Trusted_bank, account_n°428, customer_account)$
- $Hold(Trusted_bank, John, ATM.consult, account_n°428, personal_account)$

We also assume that in the IT system, $ATM.consult$ is the name given to the action of consulting using an ATM machine:

- $Consider(Trusted_bank, ATM.consult, consulting)$

Finally, we assume that the following organizational permission is part of the $Trusted_bank$ security policy:

- $Permission(Trusted_bank, customer, consulting, customer_account, personal_account)$

Using these facts and the derivation rule RG_1, we are able to conclude that John is allowed to consult his account balance on an ATM machine.

3.7 Conclusion

We browsed the main features of a new access control model called Or-BAC. This model stems from the analysis of existing access control models. In order to make this chapter lighter, we only described the basis of the Or-BAC model. Others important features are presented in the next chapter, namely the hierarchies and the constraints.

We use the traditional access control entities *Subject*, *Action* and *Object*. Furthermore Or-BAC makes it possible to assign some attributes to such entities. A major contribution of Or-BAC consists in the abstraction made of these entities into entities *Role*, *Activity* and *View*, called "organizational entities". We integrated within a single model the notions of the role based, activity based and the view based, with the view to propose a kind of unified model which offers the advantages of these models. Using these six entities, the Or-BAC model allows us to define a two-level security policy. On the one hand, the organizational

policy corresponds to authorizations that would be written by a security officer. On the other hand, the concrete policy is the one implemented in the IT system. We have presented the derivation rule to get the concrete policy from the organizational policy.

An organizational policy, as defined in the Or-BAC model, enables to design a security policy independently of the implementation choices made in the definition of the IT system. For example, whatever action is chosen to implement the activity *consulting* – it could be *acroread*, *SELECT*, etc. – the security rules in which this activity is involved remain relevant. Therefore, a single organizational policy can be applied to different IT systems.

Another entity, *Organization*, is also introduced. In the Or-BAC model, all authorizations are defined within a given organization. In practice, the organization is a parameter for all security rules of an organizational policy. The introduction of this entity meets the three following objectives. Firstly, we are able to model the structure of an organization through the definition of an organization hierarchy. Secondly, it offers the possibility to obtain more specific policies as we go down into an organization hierarchy. Thirdly, it increases the compatibility between security policies of several organizations that collaborate.

The tractability and decidability issues are discussed in chapter 5. As represented in this chapter, the formal definition of the Or-BAC model relies on Datalog. Therefore, Or-BAC policies are actually specified using logic facts and rules.

In section 2.2.5, we introduced the notions of static and dynamic organizations. So far, organizations in Or-BAC are static. We plan to integrate the concept of dynamic organization in future works. Such organizations correspond to a set of roles and views joined together during the execution of given activities. The notion of dynamic organization is more general than the notion of "session". A session, as defined in RBAC for instance, is indeed a dynamic organization that involves only one subject. Dynamic organizations would enable to model teams, projects and dynamic separation of duty, etc.

In other respects, we did not go into much detail with regards to the meaning of the abstraction of actions into activities. As in the ABAC models, we would like to propose several kinds of decomposition in order to offer means to model workflows and activity aggregation. These issues will be the subject of future works.

An Or-BAC policy, as it can be specified for the time being, does not yet meet all the requirements listed in chapter 2. At this point, we are able to express positive and negative authorizations but not to detect and solve conflicts. Moreover, we still have to explain which kind of contextual authorizations can be modelled. These issues are addressed in the following chapters, as well as the administration of our model.

Chapter 4

Or-BAC extensions

We described the main features of the Or-BAC model in the previous chapter. Here, we introduce some other issues in order to complete our model.

As we pointed out before, the Or-BAC model offers means to specify hierarchies of entities. The particularity of this model is that it is possible to define not only role hierarchies, but also activity, view, context and organization hierarchies. Associated with inheritance mechanisms, hierarchies make it possible to design structured policies, and simplify the management of the set of authorizations. Hierarchies are addressed in section 4.1.

Moreover, as in most access control models, Or-BAC provides means to specify some constraints over the definition of a policy. In section 4.2, we show how to express constraints in Or-BAC, and more noticeably, we specify the basic constraints that an Or-BAC policy must satisfy.

4.1 Hierarchy within Or-BAC

4.1.1 Introduction

The definition of a new security policy model like Or-BAC aims at modelling complex and relevant policies. The more a policy is complex, the more difficult it is to manage. It can thus be useful to structure the entities of a policy into hierarchies. We introduced hierarchies in section 2.2.7 and we mainly focused on role hierarchies. We saw that hierarchies ease the policy management. Indeed, hierarchies offer a clear way to structure the set of roles. Furthermore, the authorization inheritance makes the update of the set of authorizations easier. We suggest to introduce in the Or-BAC model the role hierarchies in the same way, but we also consider hierarchies of activity, view, context and organization [Cuppens et al. 2004a]. Actually, context hierarchies are examined at section 6 since a complete definition of context has first to be presented in order to tackle this issue. Notwithstanding hierarchy and inheritance might be useful, the use of hierarchies is prone to semantics ambiguities, particularly when negative authorizations are introduced.

The first subsection is dedicated to role hierarchies. In particular, we separately analyze positive and negative authorizations inheritance. Next, we present the view and activity hierarchies. Finally, we provide some useful inheritance mechanisms between organizations

and sub-organizations.

4.1.2 Role hierarchy

Principle

Let us first address the case of the role hierarchy. In every organization, it is possible to associate a set of roles with a hierarchy. For this purpose, we introduce the following relation:

Definition 4.1.1. Relation sub_role
"*sub_role* is a relation over domains $Org \times \mathcal{R} \times \mathcal{R}$. If *org* is an organization, if r_1 and r_2 are roles, then $sub_role(org, r_2, r_1)$ means that r_2 is a sub-role of r_1 in organization *org*".

The term "sub-role" in Or-BAC has the same meaning as the term "senior role" in RBAC [Sandhu 1998]. The relation corresponding to *sub_role* is transitive, reflexive and anti-symmetric. Notice that the role hierarchy depends on the organization. This means that the hierarchy may vary from one organization to another. Let us consider the hierarchy of figure 4.1 and assume that the roles are defined in organization $Trusted_bank$. This hierarchy is modelled in Or-BAC with a set of facts. We just give here two of them:

- $sub_role(Trusted_bank, head_agency, chief_adviser)$
- $sub_role(Trusted_bank, chief_adviser, financial_adviser)$

Let us now model inheritance principles associated with this hierarchy. A first, but incorrect, way to model inheritance through the role hierarchy would be to consider that if r_1 is a sub-role of r_2 in organization *org*, then every subject empowered in role r_1 is also empowered in role r_2. This is modelled by the following rule:

- $\forall org \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}, \forall s \in S,$
  $sub_role(org, r_2, r_1) \wedge Empower(org, s, r_1) \rightarrow Empower(org, s, r_2)$

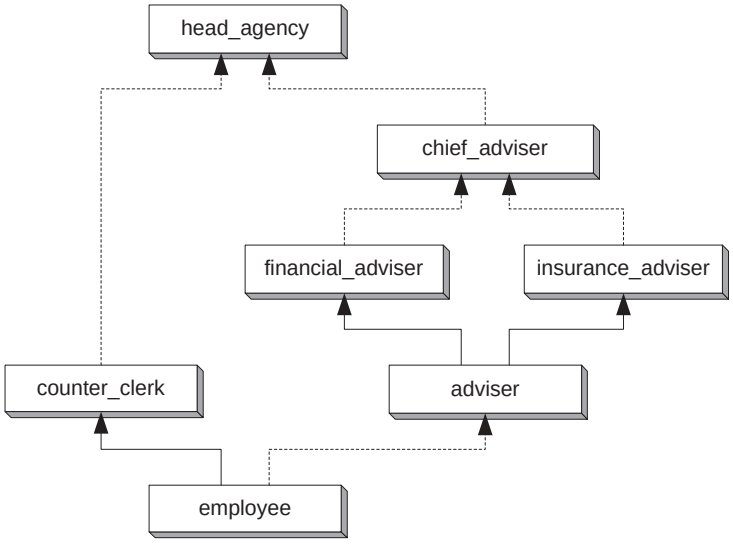


Figure 4.1: An example of a multiple role hierarchy

According to this rule, every subject empowered as a counter clerk is also empowered as an employee. By doing so, every subject empowered as a counter clerk "inherits" all authorizations assigned to role employee. However, this approach is not satisfactory for at least three reasons:

1. Our objective is to specify an authorization inheritance mechanism, while the suggested rule corresponds to inheritance of role. This underpins really different concepts.
2. It may happen that some roles in the hierarchy are only "virtual roles", that is, roles introduced in the hierarchy as a convenient way to specify permissions or prohibitions that are common to every sub-role of this virtual role. A virtual role is a pool of authorizations to which no subject is assigned. In our example, we might create the role *loan_grantor* as a junior role of role *adviser*. We could thereby choose to assign all authorizations related to loans to this role without assigning any subject to it.
3. As mentioned in [Sandhu et al. 1996], the role hierarchy may correspond to an "organizational" hierarchy, that is, r_1 is a sub-role of r_2 if each subject empowered in role r_1 is hierarchically higher in the organization than the subjects empowered in role r_2. For instance, role *head_agency* may be defined as a sub-role of role *chief_adviser*. In this case, it would be clearly incorrect to conclude that a subject empowered in role *head_agency* is also empowered in role *chief_adviser*.

A better way to model permission inheritance through the role hierarchy is specified by the following rule:

- RH_1: $\forall org \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $sub_role(org, r_2, r_1) \wedge Permission(org, r_1, a, v, c) \rightarrow Permission(org, r_2, a, v, c)$

This rule means that if role r_2 is a sub-role of role r_1 in organization org, then every permission assigned to role r_1 in organization org is also assigned to role r_2. It perfectly translates the idea that role hierarchy is associated with inheritance of permissions. Rule RH_2 corresponding to the prohibition inheritance is written in the same way.

Actually, the relation *sub_role* is a general solution to create a role hierarchy but is not associated to a specific semantics. As discussed in section 2.2.7, three kinds of role hierarchies can be distinguished: specialization hierarchy, supervision hierarchy and aggregation hierarchy. We concluded that according to the hierarchy type, the inheritance direction for permission and prohibition might differ. In fact, whatever is the hierarchy, permissions are always inherited from a role to his sub-roles and thereby rule RH_1 always applies. Contrariwise, some authors consider that prohibition inheritance may not always have the same direction.

We claim that managing several kinds of hierarchies with different inheritance directions leads to non-resolvable problems. We justify this in the next subsection. Therefore, we choose to consider only inheritance from juniors to seniors role, for positive and negative authorizations. The specific cases of contrary inheritance are treated as exceptions.

Discussion of prohibition inheritance

Let us discuss how a specific inheritance direction for negative authorization could be taken into account, and see why we do no chose this option. A solution consists in differen-

tiating the prohibition inheritance direction according to the hierarchy links between roles, as suggested in the T–RBAC model [Oj and Sandhu 2000].

Among the three kinds of hierarchy, we consider only the two first ones – i.e. specialization hierarchy and supervision hierarchy – since the third one is rather related to the activity aggregation issue than to the inheritance issue. Assume that prohibitions should be inherited from a role to his sub-roles in case of a specialization hierarchy, and on the opposite direction in case of a supervision hierarchy. To do so, we still consider predicate *sub_role* and rule RH_1 and RH_2, but we introduce two more relations in order to make a distinction between the specialization and the supervision hierarchies:

- $specialized_role(org, r_2, r_1)$: in organization org, role r_2 is a more specialized role than role r_1.
  e.g.: $specialized_role(Trusted_bank, financial_adviser, adviser)$
- $senior_role(org, r_2, r_1)$: in organization org, role r_2 is a supervision role of role r_1.
  e.g.: $senior_role(Trusted_bank, head_agency, counter_clerk)$

We consider that relationship *specialized_role* is included in *sub_role*:

- RH_3: $\forall org \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}$,
  $specialized_role(org, r_2, r_1) \rightarrow sub_role(org, r_2, r_1)$

The consequence is that rules RH_1 and RH_2 apply to the specialization role hierarchy and thus permissions and prohibitions are inherited through this hierarchy. By contrast, the relationship *senior_role* is not included in *sub_role*. We need to introduce two more rules:

- RH_4: $\forall org \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $senior_role(org, r_2, r_1) \wedge Permission(org, r_1, a, v, c)$
  $\rightarrow Permission(org, r_2, a, v, c)$
- RH_5: $\forall org \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $senior_role(org, r_2, r_1) \wedge Prohibition(org, r_2, a, v, c)$
  $\rightarrow Prohibition(org, r_1, a, v, c)$

We obtain the inheritance mechanism we wanted. Permissions are always inherited from a role to its sub-roles, as for prohibitions in the case of a specialization hierarchy. Prohibitions are inherited from a role to his junior roles in the case of a supervision hierarchy.

However, designing a supervision hierarchy with two different inheritance directions raises a major issue in the event that the security policy is specified with positive authorizations and if negative authorizations are only used as exceptions, in a DTP conflict management strategy for instance (see section 2.4.3). Let us consider the following example. Let r_1 and r_2 be two relevant roles in organization org such that $senior_role(org, r_2, r_1)$. r_2 inherits all the positive authorizations granted to r_1. Assume that a specific authorization should not be inherited from r_1 to r_2, and so, a corresponding prohibition is given to r_2 (as an exception). Using the inheritance specified above, r_1 will inherit this prohibition and therefore "loose" also its permission. In other words, such inheritance in a supervision hierarchy limits the use of prohibitions as exceptions. It would actually be possible with a conflict management strategy based on priority levels, but it would be really difficult to manage.

Mixing hierarchy types. If the role hierarchy is only a specialization hierarchy or only a supervision hierarchy, therefore inheritance does not present any problem. On the contrary,

if the role hierarchy mixes specialization and supervision relations between roles, some side-effects may appear in the inheritance process. Let us consider our example of hierarchy. It is presented at figure 4.1. Now, specialization links are represented with plain arrows whereas supervision links with dotted arrows.

If a given prohibition p is assigned to role *adviser*, then roles *financial_adviser*, *insurance_adviser* and *employee* will receive this prohibition. This is reasonable. As opposed to that, role *counter_clerk* will also obtain p, which might not be what the SSO wants. Furthermore, in more complex hierarchies, some loops might appear, which can be disturbing from a computational point of view. In fact, mixing hierarchies would complicate the authorizations management in huge organizations.

We do not go into more details concerning this issue, but it clearly raises some major problems that will have to be studied in the future. For instance, it may be suitable to distinguish "specialization authorizations" from "supervision authorizations".

4.1.3 View and activity hierarchies

Principle

The Or-BAC model makes it possible to consider inheritance between roles, but also between views and activities. To do so, we introduce two new relationships. The hierarchy relation between two views is defined using relationship *sub_view* whereas the hierarchy relation between two activities is defined using relationship *sub_activity*:

Definition 4.1.2. Relation sub_view
"*sub_view* is a relation over domains $Org \times \mathcal{V} \times \mathcal{V}$. If org is an organization, if v_1 and v_2 are views, then $sub_view(org, v_2, v_1)$ means that v_2 is a sub-view of v_1 in organization org".

Definition 4.1.3. Relation sub_activity
"*sub_activity* is a relation over domains $Org \times \mathcal{A} \times \mathcal{A}$. If org is an organization, if a_1 and a_2 are activities, then $sub_activity(org, a_2, a_1)$ means that a_2 is a sub-activity of a_1 in organization org".

These relations are transitive, reflexive and asymmetric. The positive authorization inheritance associated with view and activity hierarchies are defined with rules AH_1 and VH_1 as follows:

- AH_1: $\forall org \in Org, \forall r \in \mathcal{R}, \forall a_1 \in \mathcal{A}, \forall a_2 \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Permission(org, r, a_1, v, c) \wedge sub_activity(org, a_2, a_1)$
  $\rightarrow Permission(org, r, a_2, v, c)$
- VH_1: $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v_1 \in \mathcal{V}, \forall v_2 \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Permission(org, r, a, v_1, c) \wedge sub_view(org, v_2, v_1)$
  $\rightarrow Permission(org, r, a, v_2, c)$

In other words, an activity – resp. view – inherits all the positive authorizations of its junior activities – resp. views. Rules AH_2 and VH_2 for negative authorization inheritance are defined in a similar way, and have the same inheritance direction.

Discussion about semantics

Let us discuss the meaning of such hierarchies. Actually, concerning views and activities, we only specify specialization hierarchies since supervision in this case does not make sense.

Saying that activity a_2 is a sub-activity of a_1, that is, one of its senior activity, means that a_2 is a more specialized, or a more specific activity than a_1. It corresponds to a relation "isa". In other words, if a role is granted the right to carry out activity a_1, then, thanks to the inheritance mechanism, it is also granted the right to perform activity a_2. The activity hierarchy should essentially be designed in order to get appropriate inherited authorizations. For example, if in a given organization org, it is considered that allowing a role to delete a view supposes that this role has the right to modify this particular view, then activity $modifying$ should be defined as a sub-activity of activity $deleting$:

- $sub_activity(org, modifying, deleting)$

The view hierarchy is apprehended in the same way. A sub-view is a more specialized view. In our bank example, we consider views *account*, *customer_account* and *company_account*. An account is indiscriminately a costumer account or a company account. Therefore, the views *customer_account* and *company_account* can be seen as sub-views of the view *account*. This way, granting the role $financial_adviser$ the authorization to consult any account, enables to derive the same authorization applied to the views *customer_account* and *company_account*:

- $sub_view(Trusted_bank, customer_account, account)$
- $sub_view(Trusted_bank, company_account, account)$

4.1.4 Organization hierarchy

Principle

Previous sections showed how to define hierarchies between roles, activities and views within a given organization. In this section, we study how to define organization hierarchies. Our objective is to model an organization structure through an organization hierarchy. Most companies and institutions are composed of departments, units, services, etc. All those entities are considered as sub-organizations in the Or-BAC model. Organization hierarchies might also be used to model an IT system. For this purpose, we introduce the following relation:

Definition 4.1.4. Relation sub_organization
"$sub_organization$ is a relation over domains $Org \times Org$. If org_1 and org_2 are organizations, then $sub_organization(org_2, org_1)$ means that org_2 is a sub-organization of org_1".

We may actually require that every sub-organization org_2 of a given organization org_1 is assigned to a role. This requirement is modelled by the following constraint (constraints are introduced in section 4.2).

- C_{10}: $\forall org_1 \in Org, \forall org_2 \in Org, \forall r \in \mathcal{R}$,
  $sub_organization(org_2, org_1) \wedge \neg Empower(org_1, org_2, r)$
  $\rightarrow error()$

Let us consider the following example, where $Trusted_bank$ is a bank and $trusted_finance$ is the financial department of this bank, that is, $trusted_finance$ is empowered as the financial department. We have:

- $sub_organization(trusted_finance, Trusted_bank)$
- $Empower(Trusted_bank, trusted_finance, financial_department)$

Thereby, C_{10} is satisfied. Using organization hierarchies, Or-BAC makes it possible to model the structure of organizations. In the following, we examine the inheritance mechanisms that can be associated with an organization hierarchy. In fact, we consider three inheritance types: organizational entities inheritance, hierarchy inheritance, and authorization inheritance.

Organizational entities inheritance

The organizational entities inheritance corresponds to inheritance of role, activity, view and context from an organization to another. Actually, here again, we consider that it is a matter of choice, and it depends on the organization hierarchy semantics. Let us consider the roles only. We suggest three possibilities:

1. One may consider that the roles defined in an organization are inherited in all sub-organizations. For example, assume that organization $Trusted_bank$ is a compound of sub-organizations which are local agencies. All roles specified at figure 4.1 can be defined within $Trusted_bank$ and then inherited in all local agencies. This would be modelled by the following rule:

- $\forall org_1 \in Org, \forall org_2 \in Org, \forall r \in \mathcal{R},$
  $sub_organization(org_2, org_1) \wedge Relevant_role(org_1, r)$
  $\rightarrow Relevant_role(org_2, r)$

2. By contrast, roles might be inherited from an organization to its junior organizations. For example, if a given local agency defines a new role, it might be useful for this role to be automatically defined as relevant in organization $Trusted_bank$. This would be modelled by the following rule:

- $\forall org_1 \in Org, \forall org_2 \in Org, \forall r \in \mathcal{R},$
  $sub_organization(org_2, org_1) \wedge Relevant_role(org_2, r)$
  $\rightarrow Relevant_role(org_1, r)$

3. The third solution consists in considering that roles are not inherited.

The same discussion holds for activities, views and contexts, but different solutions can be chosen in accordance with the entity type. We claim that the SSO has to define an "entities inheritance strategy" in which he/she defines all inheritance rules.

Hierarchy inheritance

If entities are inherited from an organization to another, we have to take into account the inheritance of hierarchical links between inherited entities. Broadly speaking, the hierarchy inheritance issue happens when relevant entities hierarchically linked in an organization are also relevant in the sub-organizations. It can be the result of an inheritance mechanism or it can be specified by the SSO. We claim that if some roles are relevant in an organization

and are also relevant in the sub-organizations, then the hierarchy links between these roles must be inherited from the organization to its sub-organizations.

Let us consider roles only. Assume that org_b is a sub-organization of org_a. For those roles of org_a that are relevant in org_b, we consider that the role hierarchy defined in org_a also applies in org_b. This is modelled by the following rule:

- HH_1: $\forall org_a \in Org, \forall org_b \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R},$
  $sub_organization(org_b, org_a) \wedge$
  $sub_role(org_a, r_1, r_2) \wedge$
  $Relevant_role(org_b, r_1) \wedge Relevant_role(org_b, r_2)$
  $\rightarrow sub_role(org_b, r_1, r_2)$

Similar principles apply to inheritance of activity and view hierarchies through the organization hierarchy. Thus, we obtain two other rules (respectively called HH_2, HH_3) by replacing the *sub_role* predicate in rule HH_1 by the *specialized_role* predicate (resp. the *sub_activity* and *sub_view* predicates).

Authorizations inheritance

The last inheritance mechanism corresponds to the authorizations inheritance. We accept similar principles for inheritance of permissions and prohibitions through the organization hierarchy provided that the role, activity, view and context in the scope of the permission or prohibition are relevant in the sub-organization. This is modelled by the following rule:

- OH_1: $\forall org_1 \in Org, \forall org_2 \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$
  $sub_organization(org_2, org_1)$
  $Permission(org_1, r, a, v, c) \wedge$
  $Relevant_role(org_2, r) \wedge$
  $Relevant_activity(org_2, a) \wedge$
  $Relevant_view(org_2, v) \wedge$
  $Relevant_context(org_2, c)$
  $\rightarrow Permission(org_2, r, a, v, c)$

A similar rule applies to inheritance of prohibition (rule called OH_2).

4.1.5 Conclusion

As seen in this chapter, the definition of hierarchies and associated inheritance mechanisms offer convenient means to structure the policy entities and to manage authorizations. It is easy to specify some inheritance rules, but as we showed, numerous side-effects might appear, especially in the case of multiple inheritance. We proposed several inheritance mechanisms, for roles, activities, views and organizations. Inheritance of context is considered in chapter 6. Inheritance is mainly the concern of hierarchy semantics and enforcement choices. We suggested some inheritance mechanisms that appear to be pertinent. As in [Bertino et al. 2003], we let the SSO free to choose the relevant inheritance rules for its security policy. Therefore, future works have to be led in the direction of the definition of a parametric inheritance strategy, in which all derivation rules would be specified. Furthermore major studies ought to be realized to analyze an verify the coherence of a security policy on which such a strategy would be applied.

4.2 Modelling Constraints

So far, the Or-BAC model makes it possible to assign some authorizations to roles to carry out activities on views. Thanks to the entity *Context*, we are able to specify some conditions over these authorizations. Nevertheless, certain types of conditions can hardly be expressed that way. Actually, this is the case for instance for conditions that involve the relation *Empower*. As we will consider just later, the principle of separation of duty, for instance, is implemented through some constraints over the way subjects are empowered into roles. This cannot be done using entity *Context* and relation *Hold*. Therefore, conditions over relation *Empower*, but also over relations *Consider* and *Use* are modelled in a different way, and are called "constraints".

Constraints that apply to an access control policy were first suggested in the RBAC models, more precisely, in the $RBAC_2$ sub-model [Sandhu et al. 1996] and further analyzed in [Ahn and Sandhu 2000, Ahn and Shin 2001a, Ahn and Shin 2001b]. Constraints are special rules that have to be respected when a security policy is designed. Thus, constraints definition can be viewed as an administrative activity. However, constraints are usually kept in the policy definition part. To specify constraints in the Or-BAC model, we introduce a predicate $error()$ as done in [Bertino et al. 2003, Jajodia et al. 2001b].

Definition 4.2.1. Constraint
"A constraint is a rule whose conclusion is $error()$".

In the following, we give some examples of traditional constraints usually expressed in access control policies and we show how to model them using Or-BAC. Let us first start with the relevance constraints.

4.2.1 Relevance constraints

We first consider roles. In section 3.3.1, we introduced the relations *Empower* and *Relevant_role* without specifying the way they are associated. Actually, we require that a subject can be empowered in a role within an organization provided this role is relevant in this organization. This is modelled by the constraint C_1:

- C_1: $\forall org \in Org, \forall s \in S, \forall r \in \mathcal{R}$,
  $Empower(org, s, r) \wedge \neg Relevant_role(org, r)$
  $\rightarrow error()$

We similarly define the equivalent constraints C_2, C_3 and C_4 respectively for the views, activities and contexts. Furthermore, since authorizations are defined within an organization, entities involved in authorizations must be relevant in the corresponding organization. For positive authorizations, this is modelled with constraint C_5:

- C_5: $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Permission(org, r, a, v, c) \wedge$
  $\neg(Relevant_role(org, r) \wedge$
  $Relevant_activity(org, a) \wedge$
  $Relevant_view(org, v) \wedge$
  $Relevant_context(org, c))$
  $\rightarrow error()$

The similar constraint C_6 is defined for relevant entities in negative authorizations.

4.2.2 Cardinality constraints

A cardinality constraint enables to specify that only a given number of users – resp. action, object – can be assigned to a given role – resp. activity, view. We consider the specific case where only one user can be assigned to role. More precisely, in $Trusted_bank$, only one user can be appointed general manager. That constraint would be expressed as follows:

- $Empower(Trusted_bank, subject_1, general_manager) \wedge$
  $Empower(Trusted_bank, subject_2, general_manager) \wedge$
  $subject_1 \neq subject_2$
  $\rightarrow error()$

4.2.3 Separation constraints

Separation constraints correspond to the notion of separation of duty. Let us consider first the role separation. These constraints correspond to the idea that some roles cannot be played by the same user. In a bank, there would be a conflict of interests if the role "loan officer" and the role "customer" (that may ask for a loan) were played by the same user. The role separation constraint is used to address this issue. Saying that two roles are separated means that a given subject cannot simultaneously play these two roles. For this purpose, we first introduce the following definition:

Definition 4.2.2. Role separation constraint
"$separated_role$ is a relation over domains $Org \times \mathcal{R} \times Org \times \mathcal{R}$. If org_1 and org_2 are organizations and r_1 and r_2 are relevant roles in org_1 and org_2 respectively, then $separated_role(org_1, r_1, org_2, r_2)$ means that no subject s is permitted to be at the same time empowered into role r_1 in org_1 and into r_2 in org_2".

Using this relation, we can define the following constraint C_7:

- C_7: $\forall org_1 \in Org, \forall org_2 \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}, \forall s \in S,$
  $separated_role(org_1, r_1, org_2, r_2) \wedge$
  $Empower(org_1, s, r_1) \wedge Empower(org_2, s, r_2)$
  $\rightarrow error()$

This constraint says that, if role r_1 in organization org_1 is separated from role r_2 in organization org_2, then a subject cannot play both role r_1 in organization org_1 and role r_2 in organization org_2. Such a constraint is strong. For example, if the role loan officer and the role customer are conflicting roles, then the subjects empowered by the bank as loan officers cannot be customers of that bank.

By using relations $separated_view(org_1, v_1, org_2, v_2)$ and $separated_activity(org_1, a_1, org_2, a_2)$, we can similarly define view and activity separation constraints saying that an object - resp. an action - cannot be simultaneously used in views v_1 in org_1 and v_2 in org_2 – resp. cannot implement activities a_1 in org_1 and a_2 in org_2. Constraints C_8 and C_9 respectively corresponding to activity and view separation are defined by analogy with C_7. We also suggest to define a notion of context separation.

The definition of such constraints, as well as some examples of separated contexts are given in section 6.10. We assume that *seperated_role*, *separated_activity*, *separated_view* are respectively irreflexive and symmetrical relations between roles, activities, views and contexts. These relations will be more specifically used in the conflict management part in chapter 7.

4.3 Conclusion

The Or-BAC model incorporates hierarchies. We actually showed how to express role, activity, view and organization hierarchies and described the associated inheritance mechanisms. These hierarchies enable to model the structure of an organization and to simplify the management of the set of authorizations.

Furthermore, as in most access control models, it is possible to express constraints over the definition of a policy. In Or-BAC, constraints are specific rules that conclude on predicate *error*(). In this chapter, we specified the constraints that an Or-BAC policy must respect and take into account.

Chapter 5

Complexity and decidability

5.1 Introduction

So far, we presented the Or-BAC model using an entity/relationship language since it was more convenient to introduce the concepts of this model. Afterwards, we specify Or-BAC policies using a fragment of first-order logic, and more precisely using Datalog. Entities and relations described in the previous chapter can easily be translated into predicates and facts.

Using Datalog ensures decidability over Or-BAC policies. On the other hand, some Datalog specific restrictions have to be satisfied. We examine these points in this chapter.

With respect to syntax, we consider constants, variables and predicates, but not functions since Datalog do not admit them. As regards grammar, we consider terms, literals, atoms and Horn formulas in order to remain compliant with Datalog.

In this chapter, we first introduce Datalog and Datalog$^{\neg}$ and examine the expressivity and decidability issues in section 5.2. Then, in section 5.3 we define the logic theory that underpins the Or-BAC model.

5.2 Datalog$^{\neg}$

In this section, we present the main feature of Datalog. Or-BAC could be modelled with first-order logic [Shoenfield 2001]. However, this would not ensure decidability for policies expressed using Or-BAC. For this reason, we choose Datalog. Datalog actually defines a sub-set of first-order logic which is expressive enough and which enables us to obtain decidable policies. Moreover, it provides programs with reasonable complexity and tractable in polynomial time.

Deductive databases. We consider the Or-BAC model in the context of deductive databases (DDB). A deductive database is composed of a finite set of clauses divided in two parts: the extensional database (EDB) which consists of ground facts only, and the intentional database (IDB) which is derived using logic rules. Therefore, part of the predicates belong to the EDB (and are called extensional predicates), and the other predicates belong to the IDB (and are called intentional predicates). We consider Datalog [Ullman 1989] in particular since it is the standard language for DDBs. The main advantage of Datalog is that its semantics guarantees tractable logic programs.

Pure Datalog. Pure Datalog consists in facts and Horn clauses where the premise is a conjunction of non-negated and relational literals. It does not admit recursive rules. Furthermore, a Datalog program must be "safe". This is ensured if a specific condition, known as the safety condition, is always satisfied for all rules of the program. A rule fulfills the safety condition if each variable that appears anywhere in this rule, also appears at least in one non-negated extensional literal of the premise. This guarantees that the rule does not have an infinite set of solutions. Let us illustrate this with the following example. We consider the Richter scale used to measure the magnitude of an earthquake. This is an open ended scale since it is based on measurements not descriptions. Therefore, there is no limit to the magnitude value. Each value corresponds to a damage description:

- R_1: $destruction(x) \leftarrow x = 7$
- R_2: $great_destruction(x) \leftarrow x = 8$
- R_3: $major_damage(x) \leftarrow x > 8$

According to the safety condition, rules R_1 and R_2 are safe. On the other hand rule R_3 is not safe since $major_damage(x)$ is true for an infinite set of magnitude values.

Note that the safety condition also prevents appearance of rules of the form:

- $P(x, y) \leftarrow Q(x)$

Such rules, where a variable is in the conclusion without appearing in the premise, also have an infinite set of solutions.

Elsewhere, non-recursive rules are equivalent to the core relational algebra, thereby it ensures a unique solution to a Datalog program. However, pure Datalog clearly provides a too restrictive language. For example, the derivation rules expressing inheritance due to hierarchies cannot be written.

Datalog with recursion. Pure Datalog can be extended to Datalog with recursion. In this case, it does not correspond anymore to a relational algebra and the fixpoint semantics is usually applied: using a *forward-chaining* mechanism and starting with the EDB facts, one can iteratively derive the IDB facts. Since, there is a limited number of facts in the EDB and that the process is monotone, it converges towards a unique solution called the fixpoint (provided the safety condition is satisfied).

Datalog with negation. Datalog can also be extended with negation. It is then denoted Datalog¬. Negation is evaluated using the *closed world assumption*: if one cannot prove P, then ¬P is derived.

As we will see in chapters 6 and 7, Datalog¬ will be useful for us. But in this case, the fixpoint semantics do not ensure anymore a Datalog program to converge towards a unique solution. In order to guarantee a unique solution, a Datalog¬ program must be *stratified*.

The stratification constraint can be tested by enforcing the dependency graph. In such a graph, the nodes are the intentional predicates. Two predicates P_i and P_j are linked with an arc going from P_i to P_j if there exits a rule in which P_j is the conclusion and P_i is in the premise. If there is no cycle in this graph, then the corresponding program is stratified.

In fact, stratifying a Datalog program consists in separating the program into several sub-programs called *strata*. A stratum is thus a set of intentional predicates such that for all rules in this stratum:

- if $P \leftarrow ..., \neg Q, ...$ then $stratum(P) < stratum(Q)$
- if $P \leftarrow ..., Q, ...$ then $stratum(P) \leq stratum(Q)$

The breaking up of a Datalog¬ program into strata enables to determine the resolution order of the rules. As long as a program can be stratified, this program has a unique tractable solution.

So far, we have not introduced any logic rules with negative literals. Therefore, for the time being, Or-BAC policies are always stratified for the time being. Afterwards, stratification, as well as the safety condition, will be the main restrictions while specifying new rules in the Or-BAC model.

5.3 Logic theory

The following definition describes the logic theory that underpins an Or-BAC policy.

Definition 5.3.1. Theory T_{pol}
"In the Or-BAC model, a security policy *pol* is modelled as a logic theory T_{pol} defined as follows:

- (1) Basic model: Sets of facts using predicates *Empower*, *Consider*, *Use*, *Permission* and *Prohibition* (see chapter 3).
- (2) Attributes: Sets of facts using attributes predicates (see section 3.3.4).
- (3) Contexts: A set of context definition rules. These rules conclude on the predicate *Hold* (see chapter 6).
- (4) Hierarchies: Sets of facts using predicates *sub_role*, *sub_view* and *sub_activity*, *sub_context* and *sub_organization* (see section 4.1)
- (5) Inheritance: Rules RH_1, RH_2, VH_1, VH_2, AH_1 and AH_2 for inheritance of permissions and prohibitions (see section 4.1).
- (6) Separated entities: Sets of facts using predicates *separated_role*, *separated_view*, *separated_activity* and *separated_context* (see section 4.2).
- (7) Derivation: Rules RG_1 and RG_2 for deriving concrete permissions and prohibitions (see section 3.6). These rules respectively conclude on the predicate *Is_permitted* and *Is_prohibited*.
- Constraints: A set of constraints. These rules conclude on the predicate *error*(). In particular, the set of constraints includes the rules C_1 to C_{10} (see sections 4.2)"

According to the definition of intentional and extensional databases, points (3) and (5) belong to the EDB, whereas points (1), (2), (4), (6) and (7) belong the IDB.

Let us remind the definition of closed and open policies. In a closed policy a user is granted the right to access an object if there exists a corresponding positive authorization, and is forbidden otherwise. By contrast, in an open policy a user is forbidden to access an object if there exists a corresponding negative authorization and is allowed otherwise. In the framework of the Or-BAC model, the policy designer can chose between these two assumptions.

Furthermore, rules RG_1 and RG_2, defined in section 3.6, are Datalog compliant rules: they are safe rules, and since there is no recursion, or negation, there is no need to verify stratification or to check the dependency graph.

As mentioned in chapter 6, any context can be specified in the Or-BAC model as long as it satisfies Datalog restrictions. We discuss the cases from which problems can arise.

Finally, the logic theory presented above will be refined in the context of the conflict management, in order to introduce prioritized authorizations. This extended theory is defined in section 7.4.2.

Constraint violation. In section 4.2 we introduced several constraints. Thanks to the definition of the logic theory we just gave, we are able to define constraint violations:

Definition 5.3.2. Constraint violation
The security *pol* violates a constraint if it is possible to derive $error()$ from T_{pol}:

$$T_{pol} \vdash error()$$

Note that the violation resolution is decidable in polynomial time. Afterwards, we assume that all constraints are satisfied.

5.4 Conclusion

We gave a formal definition of the Or-BAC model. Because we use Datalog which is a fragment of first-order logic, policies have a clear syntax; and reasoning about policies is made easier. Moreover, we assume that Or-BAC policies are stratified Datalog$^\neg$ programs. This provides flexible means (recursion, negation) to specify expressive policies, while this is a guarantee to obtain unique and decidable programs which are tractable in polynomial time. A polynomial complexity is satisfactory. However, the compute time to resolve an Or-BAC policy might grow quickly as the number of authorizations increases. With respect to this issue, we present, in section 9.2.7 some results of performance tests.

In the following of the dissertation, rules are written in first-order logic for reasons of readability. Nevertheless, all these rules can be fully rewritten using Datalog's Horn formulas. Two points might not be Datalog compliant. They are discussed in sections 6.5 and 8.4.6.

Chapter 6

Modelling contexts

6.1 Introduction

As computer infrastructures become more and more complex, security models ought to handle more flexible and dynamic security policies. Furthermore, the need for security becomes bigger and bigger as the business world realizes the risk information technologies represent. Hence security policies must provide means to express accurate and dynamic security rules. It should be possible, for instance, to express that a given user is allowed to access a specific resource only on some given days, if that user holds a given position and if he accesses the resource via a specific network from a specific computer. Increased mobility, wireless connections, decentralization, etc., are just of few arguments among others that claim for highly expressive and dynamic security rules.

In this chapter, we shall show how the Or-BAC model is useful to deal with some of these new requirements. The Or-BAC model allows administrators to specify more complex set of authorizations since each authorization might only be valid in given *contexts*.

In section 2.3 we discussed the difference between static and dynamic models. Static models are just able to express static authorizations. On the contrary, dynamic models allow us to specify contextual authorizations, that is, authorizations that are activated according to the context status. In particular, the rule-based models make it possible to model contextual authorizations as logic rules. The premise corresponds to the context expressed as a conjunction of conditions. Its conclusion corresponds to a "static authorization" like a triplet $< subject, action, object >$ for instance. This is a convenient way to insert the kind of extra conditions similar to the ones we have just described above. Nevertheless, we suggest that these conditions can be better structured in the Or-BAC model thanks to the abstraction made of the concrete entities into some organizational entities, namely the entities *Role*, *Activity* and $View$. Our objective is to show that part of the contextual conditions can merely be expressed thanks to this abstraction. The other conditions are expressed using the entity *Context*. Indeed in Or-BAC, each authorization only applies in a given context that must be satisfied to activate the authorization.

The definition of context in the Or-BAC model was already introduced in the previous chapter. Our objective is now to further investigate it. Actually, as studied in section 2.3, we consider several kinds of contexts. Based on this analysis, we shall present five different contexts,

namely temporal, spatial, prerequisite, user-declared and provisional contexts. We present a taxonomy of different types of contexts and investigate the data the information system must handle in order to deal with these different contexts [Cuppens and Miège 2003c].

The remainder of this chapter is organized as follows. Section 6.2 explains how to go from the notion of condition in the Rule-BAC models to the notion of context in the Or-BAC model. Section 6.3 presents a taxonomy of different types of contexts and how to model them in the Or-BAC model. Each section from 6.5 to 6.9 is dedicated to one type of context. Sections 6.10 and 6.11 are respectively dedicated to the specific separation constraint applied to context, and to the context hierarchies.

6.2 From conditions to context

As presented in section 2.3, several access control models consist of authorizations that involve a subject, an action and an object. A static access control model only considers this type of triplet to design policies. For instance, let us consider the following triplet where $Is_permitted$ is a positive authorization:

- $Is_permitted(John, acroread, file234.pdf)$

In this example, user $John$ is allowed to read object $file2.pdf$ using Acrobat Reader. A static policy would only comprise this kind of authorization. However, a more expressive access control policy should provides means to express authorizations that depend on certain circumstances.

Otherwise, in rule-based models [Halpern and Weissman 2003, Bertino et al. 2003, Jajodia et al. 2001b], a policy is viewed as a set of access control rules in which some conditions drive some authorizations:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O, condition \rightarrow Is_permitted(s, \alpha, o)$

In the Or-BAC model, the expression of an access control policy is further structured than in a rule-based model. After analyzing the structure of *condition* of a rule-based policy rule, we suggest structuring this *condition* as follows:

- $condition \leftarrow cond_subject(s) \wedge cond_action(\alpha) \wedge cond_object(o) \wedge constraint(s, \alpha, o)$

where $cond_subject(s)$, $cond_action(\alpha)$ and $cond_object(o)$ are respectively the conditions the subject s, the action α and the object o must separately satisfy so that the corresponding rule applies. $constraint(s, \alpha, o)$ is an additional condition that joins subject s, action α and object o. Satisfying the constraint is necessary to activate the rule.

For instance, let us consider the rule "a bank customer is granted permission to consult his or her personal bank account". In this case, $cond_subject(s)$, $cond_action(\alpha)$ and $cond_object(o)$ respectively correspond to the conditions that s is a customer, α is an action of consulting and o is a bank account. $constraint(s, \alpha, o)$ is an extra condition that joins subject s and object o, namely o must be an account of subject s. In this example, action α is absent from the constraint. The main idea is to define a language that makes it possible to structure the set of conditions in which an authorization is granted. This is actually one of the main purposes of the Or-BAC model, and this work has been done in part.

In Or-BAC, instead of defining security rules that directly apply to subject, action and object, the access control policy is defined at the organizational level. Using the concepts, entities and relationships defined at chapter 3, it is easy to see that:

- $cond_subject(s)$ corresponds to the condition that a subject is empowered in a role, that is, corresponds to the relation $Empower$,
- $cond_object(o)$ corresponds to the condition that an object is used in a view, that is, corresponds to the relation Use,
- $cond_action(\alpha)$ corresponds to the condition that an action implements a given activity, that is, corresponds to the relation $Consider$.

Of course, the way we suggest to structure the rule based language *condition* is not a matter of chance. It is a manner to show that the abstraction made of the concrete entities into organizational entities is used to express some conditions and requirements.

In the remainder of this chapter we focus on the conditions that can not be taken into account with the previous relations, in other words, the conditions expressed through the fact $constraint(s, \alpha, o)$. It specifies that a subject performs an action on an object in a given $Context$, and is modelled in Or-BAC through predicate $Hold$. As we explain in the next section, the conditions required for a given context to be true are formally specified by logic rules.

condition can now be written this way:

- $condition \leftarrow Empower(Org, s, r) \wedge Consider(Org, \alpha, a) \wedge$
  $Use(Org, o, v) \wedge Hold(Org, s, \alpha, o, c)$

However, this is not exactly the way an access control policy is specified in the Or-BAC model. The exact derivation rule DR_1 in which all these relations are involved has been defined earlier in section 3.6. One should notice the introduction of the predicate $Permission$ in the Or-BAC model enables us to have single general rule, whereas in rule-based approaches, several rules have to be specified.

6.3 Context definition in Or-BAC

In the previous section, we introduced the predicate $constraint(s, \alpha, o)$ to model extra conditions that a subject, an action and an object must satisfy in order to activate an authorization. In Or-BAC, these extra conditions are modelled through the notion of $Context$. Each context has a name and its definition depends on the organization. Notice that we use the term "context" in a broad sense since it actually corresponds to any constraint that relates a subject, an action and an object. For instance, in the banking domain, the entity $Context$ will cover circumstances such as "agency customer", "transaction", "personal account", etc. $\mathcal{C}$ is the set of contexts defined in the policy. From a modelling point of view, entities $Organization$, $Subject$, $Object$, $Action$ and $Context$ are linked together by the relationship $Hold$ (see figure 3.5).

The conditions required for a given context to be linked, within a given organization, to subjects, objects and actions are formally specified by logic rules. For instance, we introduced the context $personal_account$ in section 3.4. This context may be defined as follows:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O,$
  $Hold(Trusted_bank, s, \alpha, o, personal_account)$
  $\leftarrow account_number(number, o) \wedge account_owner(s, number)$

that is, in $Trusted_bank$, the context "personal account" holds between subject s, action α and object o if and only if the account number o corresponds to one account of customer s. Attributes *account_number* and *account* are introduced in section 3.3.4.

Notice that in our approach, a context actually corresponds to a constraint that joins a subject, an object and an action. However, one or several of these parameters may be optional. This means that we can consider constraints that only apply to a subject, an object or an action. In this case, the constraint may be modelled by respectively defining a sub-role, a sub-view or a sub-activity. For instance, if we have an authorization that applies to role customer in the context "debit balance account", we can create a sub-role "debit balance customer" and express the previous authorization with this new role and without any context. However, in the general case, this approach leads to quite artificial roles. Moreover, it does not apply when the constraint joins several parameters, for instance a subject and an object. Hence, we suggest to use the context even if a sub-entity could be created.

As discussed in chapter 5, an Or-BAC policy is considered as a Datalog program. Therefore every policy has to be stratified, and must comply with the safety rule. In other words, these restrictions have to be fulfilled while defining new contexts.

Afterwards, we consider in particular the special context $default$. This context always holds with any subject, action, object and organization. In other words, this context is used for static authorizations, that is, authorizations that do not depend on any context:

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O,$
  $Hold(org, s, \alpha, o, default) \leftarrow .$

6.3.1 Context example

Let us consider the following rule: "Within $Trusted_bank$, customers are allowed to consult their personal bank accounts". This authorization is modelled as follows:

- $Permission(Trusted_bank, customer,$
  $consulting, customer_account, personal_account)$

where the context $personal_account$ is defined as previously. Then, a subject can access a given file if he is a customer, if this file corresponds to one of his accounts, and if the action he uses is considered by $Trusted_bank$ as falling within the activity *consulting*. This is expressed by the following rule (in fact the rule RG_1 partially instanciated):

- $\forall s \in S, \forall \alpha \in A, \forall o \in O,$
  $Empower(Trusted_bank, s, customer) \wedge$
  $Use(Trusted_bank, o, bank_account) \wedge$
  $Consider(Trusted_bank, \alpha, consulting) \wedge$
  $Hold(Trusted_bank, s, \alpha, o, personal_account)$
  $\rightarrow Is_permitted(s, \alpha, o)$

So far, we explained how the Or-BAC model makes it possible to express some contexts, this is, some conditions that must be fulfilled to activate some authorizations. In the following section we go into further detail about the different kinds of contexts we are able to express.

6.3.2 Context composition

We also consider conjunctive, disjunctive and negative contexts. For this purpose, we introduce function &, + and $^-$. If c_1 and c_2 are two contexts, then $\&(c_1, c_2)$ is a conjunctive context, $\oplus(c_1, c_2)$ is a disjunctive context and $\bar{c_1}$ is a negative context. We shall use the infix notations $c_1 \& c_2$ and $c_1 \oplus c_2$ in place of the prefix notations $\&(c_1, c_2)$ and $\oplus(c_1, c_2)$. Conjunctive, disjunctive and negative contexts are defined by the following rules:

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall c_1 \in \mathcal{C}, \forall c_2 \in \mathcal{C},$
  $Hold(org, s, \alpha, o, c_1 \& c_2) \leftarrow Hold(org, s, \alpha, o, c_1) \wedge Hold(org, s, \alpha, o, c_2)$
- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall c_1 \in \mathcal{C}, \forall c_2 \in \mathcal{C},$
  $Hold(org, s, \alpha, o, c_1 \oplus c_2) \leftarrow Hold(org, s, \alpha, o, c_1) \oplus Hold(org, s, \alpha, o, c_2)$
- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall c \in \mathcal{C},$
  $Hold(org, s, \alpha, o, \bar{c}) \leftarrow \neg Hold(org, s, \alpha, o, c)$

6.3.3 Compliance with Datalog

The way we have just presented context composition is not compliant with the Datalog restrictions explained in section 5.2. In order to integrate context composition in a Datalog program, the following alterations must be made.

Let us consider the conjunctive contexts. In order to ensure that, with the *backward-chaining* method used in Datalog, we have a finite set of solutions, we introduce the predicate $Use_context$:

- $Use_context(c_1 \& c_2) \rightarrow Use_context(c_1)$
- $Use_context(c_1 \& c_2) \rightarrow Use_context(c_2)$

Then, we redefined the first definition of a conjunctive context:

- $Use_context(c_1 \& c_2) \wedge$
  $Hold(org, s, \alpha, o, c_1) \wedge Hold(org, s, \alpha, o, c_2)$
  $\rightarrow Hold(org, s, \alpha, o, c_1 \& c_2)$

Therefore, only the contexts actually used in the policy will be evaluated. Secondly, functions are not allowed in Datalog, and thereby the infix notations must be replaced. In order to remove functions, you just need to add a new predicate. For conjunctive contexts, we introduce predicate *conjunctive_context*:

- $Use_context(c) \wedge conjunctive_context(c_1, c_2, c) \rightarrow Use_context(c_1)$

Modified in this way, all rules become compliant with Datalog. The same line of reasoning holds for disjunctive and negative contexts.

6.4 Taxonomy and required data

In section 2.3 we concluded that the context might be used to express different types of conditions that control activation of access control rules. We highlighted four types: the dynamic constraints, the environment, the provisions, the workflow related constraints.

Several models make it possible to capture information related to time, or to the user location, or to the team to which the user belongs, or to the current workflow, etc. Even though these models are interesting, they do not allow to express different types of contexts within a single framework. It is precisely our objective.

We defined a context as any constraint that relates a subject, an action and an object. Thus, we consider the following contexts:

- The *Temporal context* that depends on the time at which the subject is requesting an access to the system;
- the *Spatial context* that depends on the subject location;
- the *User-declared context* that depends on the subject objective (or purpose);
- the *Prerequisite context* that depends on characteristics that join the subject, the action and the object;
- the *Provisional context* that depends on previous actions the subject has performed in the system.

As a comparison between our taxonomy and the one suggested in section 2.3, we might say that the temporal and spatial contexts as well as the prerequisite context roughly correspond to the environment; the provisional context is related to the provisions and the workflow management. The dynamic constraints are not addressed in Or-BAC. As mentioned earlier, future works will be led to integrate the notion of dynamic organization, which encompasses the notion of session. With such organizations, we will be in position to express constraints such as dynamic separation of duty. The user-declared context is, as far as we know, a type of context which was never modelled until now.

We assume that each organization manages an information system that stores and manages different types of information. To control context activation, each information system must provide the information required to check that conditions associated with the context definition are satisfied or not. The following list gives the kind of information – or information container – related to the contexts we have just mentioned:

- A *global clock* to check the temporal context;
- the *subject environment* and the *software and hardware architecture* to check the spatial context;
- the *subject purpose* to check the user-declared context;
- the *system database* to check the prerequisite context;

- an *history* of the action carried out, to check the provisional context.

Figure 6.1 presents the correspondence between the contexts and the required data. The context modelling in access control policy implies that mechanisms are implemented to allow the context assessment, in order to determine if the corresponding conditions are true or not. If the information system does not provide some information in the list above, then the corresponding context cannot be managed by the access control policy. We shall not focus furthermore on this question. We assume that the IT system provides means to evaluate context validity.

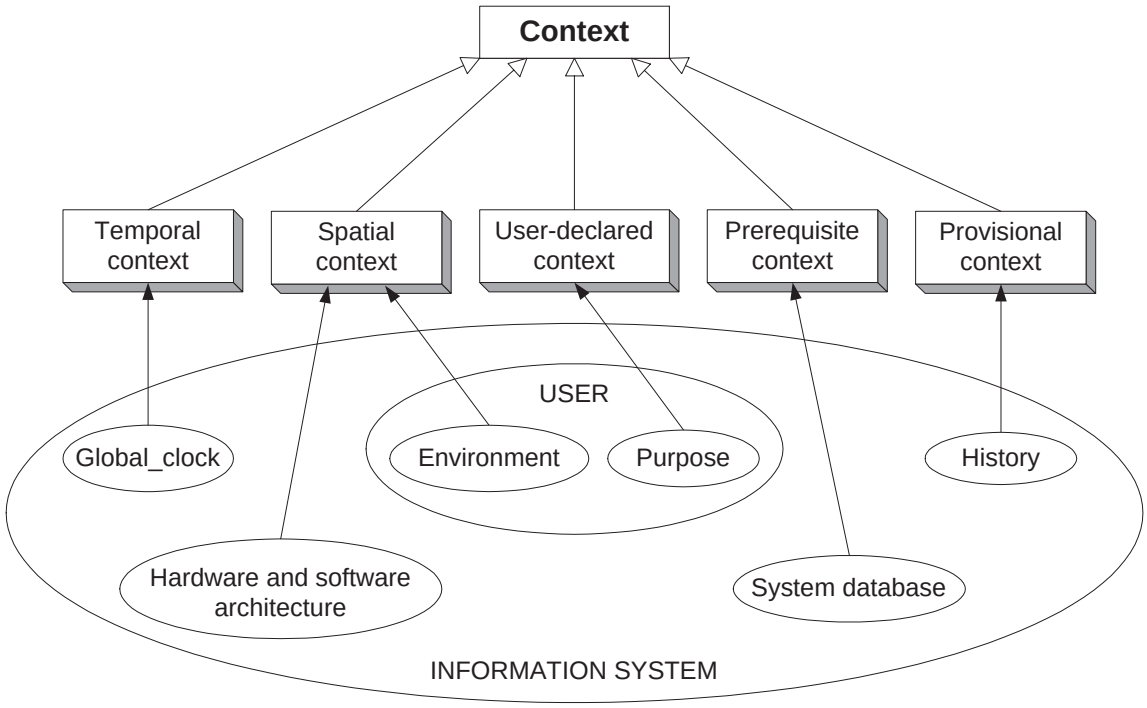


Figure 6.1: Context taxonomy and required data

These five types of context are detailed in the following subsections.

6.5 Temporal context

6.5.1 Principle

It is easy to understand that accessing some resources may only be granted when the request is made at a certain time. With temporal contexts, it should be possible to express that a given action made by a given user on a given object is authorized only at a given time or during a given time interval. The temporal conditions can correspond to a day of the week, or to a time of the day, etc.

For instance, a user within a company may be allowed to access the file server of its network only during the working hours, between 8:00 am and 19:00 pm. It is really useful to be able to express such time conditions in the access control policy. We show in this section the use of the context concept of the Or-BAC model for this purpose.

The temporal context corresponds to the time the security request is made. To validate a given query for an access, it is necessary to be able to evaluate the current time. Thus, we assume that the information system has a clock, and that this clock can be queried at any time to assess the temporal context of the query. We consider the entity clock as an object called $GLOBAL_CLOCK$.

We associate the following attributes to $GLOBAL_CLOCK$: *time*, *day*, *week*, *month*, *year*. The corresponding predicates give the current time, the current day, the current week, etc. For example:

- $time(GLOBAL_CLOCK, '11:00')$
- $date(GLOBAL_CLOCK, '05/31/2005')$

6.5.2 Basic temporal contexts

We define two predicates $before_time$ and $after_time$ that apply to the set of $Time$ and return a temporal context defined as follows:

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall t \in Time,$
  $Hold(org, s, \alpha, o, after_time(t))$
  $\leftarrow time(GLOBAL_CLOCK, current_time) \wedge current_time \geq t$
- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall t \in Time,$
  $Hold(org, s, \alpha, o, before_time(t))$
  $\leftarrow time(GLOBAL_CLOCK, current_time) \wedge current_time \leq t$

We can similarly define two predicates $before_date$ and $after_date$ that apply to the set $Date$ and return a temporal context. We also consider a predicate *on_day* that applies to the set Day and return a temporal context defined as follows:

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall d \in Day,$
  $Hold(org, s, \alpha, o, on_day(d))$
  $\leftarrow day(GLOBAL_CLOCK, current_day) \wedge current_day = d$

6.5.3 Composed temporal context

Using the basic temporal contexts, we can define more complex temporal contexts, for instance:

- $night = after_time(23:00) \oplus before_time(8:00)$
- $weekend = on_day(saturday) \oplus on_day(sunday)$
- $working_hours = after_time(08:00)\ \&\ before_time(19:00)\ \&\ \overline{weekend}$

Notice that a context definition actually depends on the organization. For instance, in an organization in which employees work on Saturday but not on Monday, context *working_hours* would be defined as follows:

- $working_hours = after_time(08:00)\ \&\ before_time(19:00)\ \&$
  $\overline{on_day(sunday)}\ \&\ \overline{on_day(monday)}$

6.5.4 Example of rules using temporal context

Let us consider the following rule: "Within $Trusted_bank$, a financial adviser is allowed to consult the company account database $CADB$ during working hours only". This is expressed by the following fact:

- $Permission(Trusted_bank, financial_adviser, consulting, CADB, working_hours)$

Let us now assume that the role head agency is permitted to consult $CADB$ also on Sunday. This is expressed by the following fact:

- $Permission(Trusted_bank, head_agency, consulting, CADB,$
  $working_hours \oplus on_day(sunday))$

If we assume that $head_agency$ is a senior role of role $financial_adviser$, it would be actually sufficient to specify that $head_agency$ is permitted to consult $CADB$ on Sunday, since the authorization in temporal context $working_hours$ will be inherited from the role $financial_adviser$.

6.5.5 Decidability

The definition of temporal contexts raises a problem with respect to the decidability issue. The validity of a temporal context depends on the current time, and thereby cannot be pre-computed. This is due to the Datalog bottom-up evaluation. By contrast, temporal contexts can be evaluated in Prolog which has a top-down approach. We did not study this issue in detail on that issue. [Becker and Sewell 2004] presents *Cassandra*, a language for policies based on Datalog with constraints. The authors suggest a solution to guarantee tractable solutions in the situation of temporal contexts. We have to carry out further works on this issue.

6.6 Spatial context

6.6.1 Principle

Knowing the location from where the user makes the request can be useful to specify the access control policy. For example the head of a $Trusted_bank$ agency is granted the right to read all employees' payrolls, however he must read those payrolls in his own office, and not anywhere else in the company. It thus reduces the possibility of curious employees being able to read their colleagues' payrolls over the manager's shoulder. Spatial context is used to express this kind of condition.

We can distinguish two different types of spatial context. The physical spatial context and the logical spatial context. The first one corresponds to the physical location of the user, namely his office, a security area, a specific building, the country, etc. Validation of such a context may rely on GPS modules, employee's badge tracking, etc. The logical spatial context corresponds to the "logical location" he stands in. For example, it can be the computer, the network or the sub-network, the cell in the case of radio communication such as in UMTS, etc.

In some cases, physical and logical spatial contexts are highly correlated. The network IP address from which a user is connected probably corresponds to a specific physical place such as a department area. Note that due to the expanding use of Global Position System (GPS) tools, it could be possible to locate a user or a terminal independently of the network.

If an organization allows its employees the use of Mobile IP, it is necessary to take into account from which network a request is emitted. A user will probably get reduced permissions if he is connected from a customer's office. Moreover the development of wireless technologies such as Wi-Fi motivates this work. The security policy must make it possible to take into account the fact that a user is connected through a regular network or a wireless network, and in this last case, on which Wi-Fi access point he is attached to.

In this section we show that the entity concept in the Or-BAC model can be used to express such contexts.

6.6.2 Example of spatial contexts

Consider that $Trusted_bank$ has a secured area SA in which specific security requirements are enforced. For example, there is no possibility of optical eavesdropping [Kuhn 2002]. Users are allowed to consult certain documents on their laptop only in this area. If a given subnetwork address is allocated to this area, then the IP address of the terminal that is making a request is enough to locate it. Thus SA corresponds to the name of a specific subnetwork. We consider that the subject entity has the attribute $host_IP$ which provides the IP address of the terminal on which a user is connected. The attribute IP_range is allocated to networks and subnetworks. In this example the context in_secure_area can be defined as follows:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O,$
  $Hold(Trusted_bank, s, \alpha, o, in_secure_area)$
  $\leftarrow host_IP(s, ip) \wedge IP_range(ip, SA)$

A similar idea can be used in the case of Mobile IP. When using this protocol, the so-called local agent must manage the network where the mobile hosts are.

Let us consider another example. In a wireless network, some user is allowed to access a specific resource from everywhere but only with his own laptop. We assume that attribute $laptop_owner$ indicates the laptop number of a subject, that attribute $laptop_MAC$ indicates the MAC address corresponding to a laptop number and finally that $host_MAC$ states the MAC address of the received packets. The context on_own_laptop is then defined as follows:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O,$
  $Hold(Trusted_bank, s, a, o, on_own_laptop)$
  $\leftarrow laptop_owner(s, laptop) \wedge laptop_MAC(laptop, mac) \wedge host_MAC(s, mac)$

Notice that specific security mechanisms must be implemented to prevent a malicious user to bypass access control requirements by forging his own packets, and choosing the appropriate IP address or MAC address. However, we shall not further discuss these specific security issues here.

6.7 User-declared context

6.7.1 Principle

In some circumstances, a subject according to the role he plays in the organization may be allowed to declare that he performs some activities in a given context. When declaring a context, a subject will obtain some specific permissions and possibly also some prohibitions. For instance, a subject playing the role financial adviser may be permitted to declare that he is performing a statistical analysis about the loan given in his agency. By doing so, this subject will be permitted to have access to all accounts.

In our approach, user declared contexts are modelled as follows. We shall consider a view called *Purpose*. Objects belonging to the view *Purpose* have an attribute *recipient*. If p is an object belonging to the view *Purpose*, then predicate $recipient(p, rcpt)$ represents the subject *rcpt* who takes advantage of the declared purpose. Notice that it is possible to consider sub-views of view *Purpose* that may be associated with other specific attributes (see below for an example). The access control policy can specify that some roles are permitted to insert some objects in the view *Purpose*. Of course, the policy can also specify that the inserted objects must satisfy constraints, for instance conditions related to the attributes of the objects. This is useful to specify that a financial adviser is permitted to declare the statistical analysis purpose but not another purpose.

By inserting an object in the view *Purpose*, a subject will declare that another subject will perform some activity in a given context. Notice that in our model, there are two subjects involved in the process of context declaration: the subject who is declaring the context and the subject who takes advantage of this declaration. The policy can specify that these two subjects must be identical. For instance, in the example above, the financial adviser may be only permitted to declare a context that applies to himself. However, it is also possible that the policy specifies that these two subjects may be different, provided that these subjects satisfy some constraints. For instance, a financial adviser may be permitted to declare that a subordinate counter clerk will perform some activity in some given context. In this case, the subjects are different but the declarant must be a financial adviser and the recipient must be a subordinate counter clerk.

The definition of a user-declared context has three parts:

1. Definition of the context associated with objects belonging to the view *Purpose*
2. Specification of roles who are permitted to declare some given purpose.
3. Specification of roles that are permitted to perform some activities in the associated user-declared context.

The user-declared context enables a subject who plays a role to declare a given context. This notion is really useful in the case where the context assessment can only be done by this subject. For example, if the IT administrator – who is usually not allowed to access the account database – thinks there may be a virus in the IT system, he must get the permission to open any files and any database. But only the IT administrator is able to evaluate that the context “virus detection” is true.

Of course, the notion of user-declared context is to put together with the notion of trust and accountability.

6.7.2 Example of user-declared context

Let us illustrate the three steps of definition of a user-declared context through the following example: In $Trusted_bank$, the role "financial adviser" is granted the right to declare the context "statistical analysis". In this context, users who play this role are allowed to have access to all account files.

We consider a sub-view *statistical_analysis* of view *Purpose*. View *statistical_analysis* has two attributes: *recipient* (inherited from view *Purpose*) and *topic*. If p is an object belonging to view *statistical_analysis* then we assume that the fact $topic(p, tpc)$ in which *tpc* represents the topic associated with this analysis (for instance, loan repayment, debit balance account, withdraw amount, etc.) is defined in the policy.

First step: We define a context *loan_repayment_analysis*. This context is associated with objects belonging to sub-view *statistical_analysis* having topic equal to *loan_repayment*. This is represented by the following rule:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O, \forall p \in O,$
  $Hold(Trusted_bank, s, \alpha, o, loan_repayment_analysis)$
  $\leftarrow use(Trusted_bank, p, statistical_analysis) \wedge$
  $recipient(p, s) \wedge topic(p, loan_repayment)$

that is, in $Trusted_bank$, subject s performs action α on object o in context *loan_repayment_analysis* if there is an object p belonging to view *statistical_analysis* having s as recipient and *loan_repayment* as topic.

Second step: We specify that subjects playing role $financial_adviser$ are permitted to declare the purpose *statistical_analysis* that applies to themselves:

- $Permission(Trusted_bank, financial_adviser, declare,$
  $statistical_analysis, My_purpose)$

In this *Permission*, *declare* is an activity and $My_purpose$ is a context defined as follows:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O, \forall p \in O,$
  $Hold(Trusted_bank, s, \alpha, o, My_purpose)$
  $\leftarrow use(Trusted_bank, p, Purpose) \wedge recipient(p, s)$

that is, a subject s is in context $My_purpose$ if there is a purpose p having s as a recipient.

The reader should notice that if such an authorization is granted to role financial adviser, then if only one subject playing that role declares this context then all subjects playing that role get the associated permission - written at next step. There is no problem, if in a bank agency the role financial adviser is only played by one user. Otherwise, it can be relevant to limit the possibilities of declaring this context. This can be done in the definition of context *loan_repayment_analysis* for instance.

Third step: We specify that subjects playing role $financial_adviser$ are permitted to consult all objects belonging to view *account* in context *loan_repayment_analysis*:

- $Permission(Trusted_bank, financial_adviser, consult, account,$
  $loan_repayment_analysis)$

In the end, role financial adviser obtains the authorization to consult all accounts if he declared the context loan repayment analysis.

6.8 Prerequisite context

6.8.1 Principle

In many cases, an authorization is granted to a subject, but only if some specific constraints are satisfied. For instance, let us turn back to the example presented in section 6.3.1. This example says that a customer is permitted to consult a bank account. However, a specific constraint must be satisfied: this account must be one of the customer's accounts.

We assume that the information required to check this constraint, namely the set of accounts of each customer, is stored into the *system database*. Thus, the evaluation of such a constraint is done by querying the database. This kind of constraint is called *prerequisite context*.

6.8.2 Example of prerequisite context

Let us consider the following example. Each company that has an account at $Trusted_bank$ has an attending financial adviser. Only this financial adviser is allowed to consult the company account. This authorization is modelled as follows:

- $Permission(Trusted_bank, financial_adviser, consulting,$
  $company_account, attending_adviser)$

where context attending_advise is defined this way:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O,$
  $Hold(Trusted_bank, s, \alpha, o, attending_adviser)$
  $\leftarrow company_account(company, o) \wedge adviser_of(s, company))$

We assume that attribute $company_account(company, account)$ states the company name corresponding to the account $account$, and that predicate $adviser_of(subject, company)$ indicates the subject $subject$ who is in charge of the accounts of the company $company$.

Now, consider the following rule: "A counter clerk is granted the permission to consult a company account in the context where the financial adviser of the corresponding company is absent". We consider the following attribute, $status$, that indicates the user's status – for instance $absent$.

The prerequisite context $absent_financial_adviser$ is expressed as follows[1]:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O,$
  $Hold(Trusted_bank, s, \alpha, o, absent_financial_adviser)$
  $\leftarrow (Use(Trusted_bank, o, company_account) \wedge$
  $\neg \exists s' \in S, (Empower(Trusted_bank, s', financial_adviser) \wedge$
  $company_account(company, o) \wedge$
  $adviser_of(s', company) \wedge \neg status(s', absent)))$

that is, subject s performs action α on object o if o is a company account and if there is no subject s' such that s' is the attending adviser of this company and s' is not absent.

[1] This rule should be written in two parts in order to be compliant with Datalog. We keep it this way since it is easier to understand.

Notice that we decide to define the context $absent_financial_adviser$ as a prerequisite context. It is evaluated by querying the database in order to check if the financial adviser of a given company is absent. This is possible if the database actually stores such information. If this is not the case, then another possibility would be to define the context $absent_financial_adviser$ as a user-declared context. For instance, the counter clerk may be permitted to declare this context for a given company account. Of course, the two policies will not be identical since, in this second case, the counter clerk will be responsible for declaring the context $absent_financial_adviser$.

This means that the fact that a given context will be defined as a prerequisite context or as a user-declared context strongly depends on the data stored in the system database.

6.9 Provisional context

6.9.1 Principle

The notion of provision, or provisional authorization, is addressed in [Jajodia et al. 2001a, Kudo and Hada 2000, Bettini et al. 2002]. A provision is described as a kind of obligation, that is, a specific action, that has to be fulfilled before the decision is taken. As suggested in [Jajodia et al. 2001a], provisions and obligations can actually be distinguished in the following way: provisions are specific actions that have to be performed before the decision is taken, whereas obligations are actions that have to be performed after the decision, and more generally in the future.

We suggest modelling this notion of provision using another type of context called provisional context. For this purpose, we shall first assume that the information system manages a log, that stores data about previous activities of users in the system. This is modelled by a view called *Log*. Objects belonging to view *Log* have six attributes: *actor*, *action*, *target*, *activity*, *context* and *date* that respectively correspond the subject (*actor*) who is performing an action (*action*) on an object (*target*) within an activity (*activity*) in a context (*context*) at a given date (*date*).

6.9.2 Example of provisional context

To illustrate the approach, let us show how to model the following authorization: "In $Trusted_bank$, an adviser can create a new customer account if he checked before the customer's financial situation with the central bank". To model this rule, we first define a provisional context called $new_account$ as follows:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O, \forall l \in O,$
  $Define(Trusted_bank, s, \alpha, o, new_account)$
  $\leftarrow Use(Trusted_bank, l, Log) \wedge$
  $actor(l, s) \wedge$
  $activity(l, consult_central_bank) \wedge$
  $customer_account(customer, o) \wedge customer_situation(customer, situation) \wedge$
  $target(l, situation)$

In this context, we assume that the attribute *customer_account* states the account of a customer and that the attribute *customer_situation* points the financial situation of a future customer. The authorization is modelled as follows:

- $Permission(Trusted_bank, adviser, creating, customer_account, new_account)$

In the future, we plan to extend the notion of provisional context in two directions. First, if a sequence of provisional contexts is defined in this manner, we can model some simple workflows. Indeed, if an action, once it is realized, triggers one or several authorizations, it is thus possible to specify the set of steps of a workflow. However, managing a workflow through an history log can be used to activate some authorizations, but does not permit to deactivate them. Therefore, some major work must be realized with respect to this issue. Second, we also want to incorporate provisional obligations, that is, obligations activated thanks to the carrying out of some activities, and thus incorporate provision and obligation as done in [Bettini et al. 2002].

6.10 Separated context

In section 4.2.3, we introduced some constraints in the Or-BAC model and in particular the separation constraints. Here, we tackle the specific case of the context separation constraint. To do this, we add a new relation *separated_context* as follows:

Definition 6.10.1. Context separation constraint
"If c_1 and c_2 are two relevant contexts in organizations org_1 and org_2 respectively, then $separated_context(org_1, c_1, org_2, c_2)$ means that contexts c_1 and c_2 are separated".

As for roles, activities and views, the separation constraint for contexts is associated to a rule that concludes on predicate $error()$:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O,$
  $separated_context(org_1, c_1, org_2, c_2) \wedge$
  $hold(org_1, s, a, o, c_1) \wedge$
  $hold(org_2, s, a, o, c_2)$
  $\rightarrow error()$

Actually, the specific case of separation constraint applied to contexts is less simple as for roles, activities and views. For example for roles, a separation constraint is explicitly stated by the SSO in order to prevent subjects to be empowered into two conflicting roles. By contrast, a context separation constraint may be explicitly stated by the SSO, but it also stems from the two context definitions.

Assume that the temporal context *working_hours* holds for all the days of the week, and that context *week-end* holds on Saturday and Sunday. These contexts are in fact separated. It is also easy to find examples of de facto separated spatial contexts. Prerequisite contexts and provisional contexts are also separated in accordance to their definition, to the database and to the history log content. Therefore the meaning of a context separation constraint is quite different from the ones of the other separation constraints, since the context separation constraint should be in adequation with the contexts definition.

Nevertheless, expressing an explicit context separation constraint in these case has an advantage. Let us get back to our bank example, and assume that part of the security policy is specified at the senior-most organization level. For instance, the SSO of the parent company might specify that contexts *working_hour* and *week-end* are separated, in the sense that they *must* be separated. Then the SSOs of each agency are in charge of defining the working hours and the week-end days that apply in their agency keeping in mind that these contexts must be separated.

This raises another issue. From the implementation standpoint, the separation constraint poses some problems. Verifying that two contexts are actually separated supposes that the IT system is able to understand the contexts definitions. Let us take first the case of physical spatial context separation constraint. In order to verify such a constraint, the security policy enforcement module must know the plan of the organization building. This becomes even more complicated in case of conjunctive contexts. Actually, as we will see in chapter 7, these issues are highly significant with respect to conflict management.

6.11 Context hierarchy

We introduced hierarchies in section 4.1, but we let the context hierarchy aside since we needed to define the context taxonomy beforehand.

The context hierarchy raises the same discussion as for the context separation constraints. In fact, the role, activity, view and organization hierarchies are stated by the SSO. The context hierarchy can also be stated by the SSO, but it is most likely that a context hierarchy stems from the definitions of the contexts defined in the policy. In order to model a context hierarchy, we introduce a new relationship *sub_context*:

Definition 6.11.1. Relation sub_context
"*sub_context* is a relation over domains $Org \times \mathcal{C} \times \mathcal{C}$. If org is an organization, if c_1 and c_2 are contexts, then $sub_context(org, c_2, c_1)$ means that c_2 is a sub-context of c_1 in organization org".

Roughly speaking, context c_2 is a sub-context of context c_1 if c_1 always holds between a subject, an action and an object in an organization when c_2 holds for the same subject, action object and organization. For example, if context *week-end* is true on Saturday and Sunday, and if context *saturday* is true on Saturday, then *saturday* can be viewed as a sub-context of *week-end*.

For the time being, the context hierarchy had not been investigate in detail. So, for the moment, we consider that the context hierarchy is explicitly stated by the SSO, and assume that the SSO takes care of the coherence between the hierarchy and the definition of the contexts.

6.12 Conclusion

We presented how a wide variety of contexts can be modelled using Or-BAC. First of all, the policy structuring, with the help of the organizational entities, is used to express some

conditions over the set of authorizations. For instance, assume that an authorization is activated provided the object it is applied to is of a given type, there is no need to specify a context, one just has to rely on relation Use. For the other conditions that cannot be expressed that way, we introduce the entity $Context$. A context is defined by a logic rule, therefore the only limitation comes from the compliance to Datalog. Nevertheless, we suggest a taxonomy of different types of context. Starting from elementary contexts, we also define conjunctive, disjunctive and negative contexts.

The activation conditions which are the concern of the policy *environment* – as defined in [Covington et al. 2000, Sandhu and Park 2004] – are expressed in Or-BAC using the temporal, spatial and prerequisite contexts. More precisely, the temporal context corresponds to the time at which a security request is launched, the spatial context to the physical or the logical location of a user when he makes a security request, and finally, the prerequisite context is used to express conditions regarding all information stored in the organization databases.

We brought in a new kind of context called *user-declared* context. This makes it possible to model the contexts that can only be activated by users themselves, this is, activated by the users who make the security requests. To activate a user-declared context, a user must be authorized to declare some objective (or purpose) of his or her activity. This is modelled by views; a given user-declared context is thus activated by inserting a given object in this view. The user-declared context is related to the notions of trust and responsibility.

Finally, we define the provisional context. This is used to model authorizations that depend on previous actions performed by the user. To control activation of provisional context, the information system must store the actions carried out by the users. Information systems generally provide such historical data through audit trail. Provisional context may also be useful to model situations where users obtain permissions as their work proceeds. Similarly, provisional prohibition is another useful context to model situations where the user's previous activity leads to prohibition.

There were several other proposals to model contexts within an access control model but this is the first time that all the different contexts are expressed within a unique homogeneous framework. Some works are currently carried out in order to investigate in more detail the notion of provisional context, in particular to model management of rights in workflow systems. We are also applying this model in the framework of relational database administration.

As for the other organizational entities of the Or-BAC model, we introduce the context hierarchy, and the context separation constraints. Nevertheless, these notions, when applied to the context lead to complex problems that have to be further analyzed.

Chapter 7

Prohibitions and conflict management

7.1 Introduction

As described in section 3.3.5, the Or-BAC model enables to express negative authorizations, also called prohibitions. The motivations for integrating negative authorizations in a security policy are expressed in section 2.4.1. In this chapter, we focus on the conflict management between positive and negative authorizations, and in particular on the solutions proposed within the Or-BAC model framework [Cuppens and Miège 2003b, Cuppens and Miège 2004b].

Specifying a security policy that includes both permissions and prohibitions may lead to conflicting situations. This corresponds to situations in which a subject is both permitted and prohibited to perform a given action on a given object. Hence, the system might not be able to decide either to allow or deny the access. In section 2.4.3, I exposed several solutions that can be found in the literature, like PTP (permission takes precedence), DTP (denial takes precedence) or "the more specific takes precedence", among others. I concluded that a security policy model which enables us to express prohibitions should permit to specify a parametric "conflict management strategy". A conflict management strategy consists of a set of rules that enable the system to decide, in the event of a conflict, to discard either the positive or the negative authorization. As a consequence, the resulting access control policy will depend on the chosen strategy. So the security officer should have the possibility to define his or her own conflict management strategy in order to obtain a relevant access control policy.

I also concluded that such a strategy should take into account potential conflicts. We actually make a distinction between "actual" and "potential" conflicts. During the time the policy is enforced in the information system, some conflicts might actually happen between positive and negative authorizations when a security request is made. For instance, user John tries to read a given file, and the access control module concludes on both a permission and a prohibition for this specific request. This is an actual conflict. On the other hand, we might want to detect conflicts before they occur, and more precisely during the policy specification process. This consists in detecting the coexistence of rules that may lead to

some conflicts if their associated conditions are simultaneously satisfied. Such conflicts are called potential conflicts. As we show in the following, the Or-BAC model is particulary adapted for such a distinction between actual and potential conflicts since actual conflicts correspond to conflicts between concrete authorizations and potential conflicts to conflicts between organizational authorizations.

Let us illustrate this with an example. Assume that in a given bank, the role *counter_clerk* is forbidden to modify the customer's accounts, whereas the role *adviser* has the permission to do it. These two authorizations lead to a potential conflict since if a user plays these two roles, a conflict might occur. Note that if an explicit constraint states that these roles are separated, then there is no potential conflict anymore. Therefore, the potential conflict detection intervenes before the actual conflict detection is carried out. With regard to actual conflicts, assume that user John is empowered in both roles *counter_clerk* and *adviser*. As a consequence an actual conflict will occur when John attempts to modify an account.

A conflict management strategy should also offer means to detect redundant authorizations. When applying a conflict management strategy, some authorizations might become useless because they are always in conflict with other higher prioritized authorizations, and thus never take precedence. Such authorizations are called redundant authorizations. Since detecting the redundant authorizations in complex security policies without an adapted mechanisms might be really difficult, we suggest an effective solution to this issue.

In the Or-BAC framework, the aim of considering negative authorizations is to provide expressive and powerful means to specify conflict management strategies. Furthermore, an Or-BAC policy is in fact defined at two levels: the organizational level and the concrete level. As mentioned above, from the information system standpoint, conflicts occur at the concrete level, that is, between concrete authorizations. However, we aim at defining a conflict management strategy at the organizational level, mainly for two reasons. First, it should allow to detect potential conflicts. Second, if we are able to ensure that the organization policy does not include any conflicting pairs of positive and negative authorizations, thereby we are able to give the undertaking that no conflict will appear in the concrete policy implemented in the information system. Moreover, we propose a solution for the redundant authorizations problem.

In section 7.2 we describe the principles of our approach. In section 7.3 we introduce some new predicates and rules in order to define a new derivation process to derive the concrete authorizations from the organizational authorizations. In section 7.4 we define the new logic theory for managing conflicts in Or-BAC. Section 7.5 is dedicated to the prevention of conflicts. Finally, we tackle the redundant authorizations issue in section 7.6.

7.2 Principles of the approach

7.2.1 Conflicts in theory T_{pol}

Let us consider again the logic theory T_{pol} described before in section 5.3. In the Or-BAC model, when a user makes a security request, that is, when a user asks for the permission to access an object, the information system has to take a decision in accordance with the possibility to derive concrete authorizations (positive and/or negative) related to this subject,

this action and this object. Therefore, a conflict occurs when a concrete permission and a prohibition are derived for the same subject, action and object. In order to characterize such a situation we introduce the predicate $conflict()$. The following rule RC specifies a conflicting situation:

- RC: $\forall s \in S, \forall \alpha \in A, \forall o \in O,$
  $Is_permitted(s, \alpha, o) \wedge Is_prohibited(s, \alpha, o)$
  $\rightarrow conflict()$

Let us add this rule in the logic theory T_{pol}. We are now able to give the definition of conflicting policy:

Definition 7.2.1. Conflicting policy in T_{pol}
There is a conflict in the security *pol* if it is possible to derive $conflict()$ from T_{pol}:

$$T_{pol} \vdash conflict()$$

One might argue that conflicts can be defined at the organizational policy level, that is, between organizational Or-BAC predicates $Permission$ and $Prohibition$. Let us assume that the two following authorizations are defined in the policy:

- $Permission(org, r, a, v, c)$ and $Prohibition(org, r, a, v, c)$

The fact we have such authorizations is not sufficient to derive conflicting pairs of concrete authorizations. For instance, if the context c is equal to "night", then there is no conflict between concrete authorizations during the day. More formally, a conflict occurs only if a subject is empowered in role r, an action is considered as an activity a, an object is used in view v and context c holds between these concrete entities in organization org. Furthermore, this condition is sufficient but not necessary. There is actually no need to have a permission and a prohibition applied exactly to the same role, activity, view and context for a conflict to happen. Consider the following authorizations:

- $Permission(org, r_1, a, v, c)$ and $Prohibition(org, r_2, a, v, c)$

If a subject s plays both roles r_1 and r_2, and if an action is considered as an activity a, an object is used in view v and context c holds between these concrete entities in organization org, then it is possible to derive a conflicting pair of concrete authorizations. As a consequence, actual conflicts can only be detected at the concrete level. On the other hand, the organization level enables us to determine the potential conflicts. This issue is examined in section 7.5. Let us present how to define conflict management strategies in Or-BAC.

7.2.2 Priority levels

When a conflict occurs between two facts $Is_permitted$ and $Is_prohibited$, our proposal to solve such a conflict is to associate these facts with priority levels. Then the fact with the highest priority takes precedence over the other fact. For this purpose, we introduce a set of priority levels denoted $\mathcal{L}$. We assume that $\mathcal{L}$ is associated with a partial order relation denoted $\prec$ such as:

- if l_1 and l_2 are two priorities, then $l_1 \prec l_2$ means that l_2 is higher than l_1

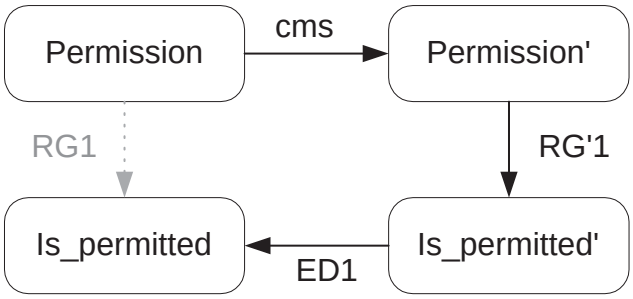


Figure 7.1: Permission → Is_permitted

- $l_1 \prec\!\succ l_2$ means than the priorities l_1 and l_2 are not comparable, that is:
  $l_1 \prec\!\succ l_2 \leftrightarrow \neg(l_1 \prec l_2) \wedge \neg(l_2 \prec l_1)$

The definition of $\mathcal{L}$ is application-dependent in the sense that it depends on the strategy used to manage conflicts. The set of priority levels is used to prioritize permissions and prohibitions both at the concrete and the organizational levels. Thus, we consider the following four new predicates:

Definition 7.2.2. Predicate Permission'
"$Permission'$ is a relation over domains $Org \times \mathcal{R} \times \mathcal{A} \times \mathcal{V} \times \mathcal{C} \times \mathcal{L}$. If org is an organization, r a role, v a view, a an activity, c a context and l a priority level, then $Permission'(org, r, v, a, c, l)$ means that $Permission(org, r, a, v, c)$ is assigned to priority level l".

Definition 7.2.3. Predicate Prohibition'
"$Prohibition'$ is a relation over domains $Org \times \mathcal{R} \times \mathcal{A} \times \mathcal{V} \times \mathcal{C} \times \mathcal{L}$. If org is an organization, r a role, v a view, a an activity, c a context and l a priority level, then $Prohibition'(org, r, v, a, c, l)$ means that $Prohibition(org, r, a, v, c)$ is assigned to priority level l".

Definition 7.2.4. Predicate Is_permitted'
"$Is_permitted'$ is a relation over domains $S \times A \times O \times \mathcal{L}$. If s is a subject, α an action, o an object and l a priority level, then $Is_permitted'(s, \alpha, o, l)$ specifies that $Is_permitted(s, \alpha, o)$ is assigned to priority level l".

Definition 7.2.5. Predicate Is_prohibited'
"$Is_prohibited'$ is a relation over domains $S \times A \times O \times \mathcal{L}$. If s is a subject, α an action, o an object and l a priority level, then $Is_prohibited'(s, \alpha, o, l)$ specifies that $Is_prohibited(s, \alpha, o)$ is assigned to priority level l".

In a theory T_{pol} that represents a given security policy pol, rule RG_1 was used to derive concrete authorizations $Is_permitted$ from organizational authorizations $Permission$. In the remainder of this section, we shall model a prioritized security policy $Ppol$. In $Ppol$, rule RG_1 is no longer available. The new derivation process is structured into three steps:

1. Derivation of $Permission'$ from $Permission$. This corresponds to the definition of a Conflict Management Strategy (cms).

2. Derivation of $Is_permitted'$ from $Permission'$. This is modelled with a new derivation rule RG'$_1$.
3. Derivation of $Is_permitted$ from $Is_permitted'$. This is modelled with an explicit permission derivation rule ED$_1$.

The new derivation process is presented in figure 7.1. The derivation process used so far is represented with a dotted arrow. The new process is composed of three steps and is presented with plain arrows. The process used to derive concrete prohibitions $Is_prohibited$ from organizational prohibitions $Prohibition$ is similar. We now present a formalization of this process.

7.3 The new derivation process

In this section, we first explain each step of the new derivation process.

7.3.1 Step 1: Conflict management strategy

First, let us focus on the specification of priority levels. When a new permission or prohibition is inserted in the security policy, a first possibility consists in assuming that the administrator will manually associate this permission or prohibition with a priority level. Such an administrative approach is quite complex to manage and security administrator dependent. Instead, we suggest defining a set of rules that is automatically used to derive a priority level for each authorization. This is what we call a conflict management strategy. Notice that we do not assume that a conflict management strategy is complete, in the sense that it provides means to assign a priority level to *every* permission or prohibition. If some permissions or prohibitions cannot be automatically prioritized, then the administrator will be asked to manually assign a priority.

Definition 7.3.1. Conflict Management Strategy
"A conflict management strategy is a set of rules that concludes on the predicates $Permission'$ or $Prohibition'$".

Notice that the priority levels do not necessarily correspond to priority numbers. On the contrary, priority levels enable to model a large number of strategies. Consider the following examples:

Example 1: Denial takes precedence (cms_1)

In DTP, when a situation of conflict occurs, prohibition takes precedence. Let us call this strategy cms_1. This is a very simple example of strategy. It is modelled as follows:

- $\mathcal{L} = \{0, 1\}$
- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Permission(org, r, a, v, c) \rightarrow Permission'(org, r, a, v, c, 0)$
- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Prohibition(org, r, a, v, c) \rightarrow Prohibition'(org, r, a, v, c, 1)$

The strategy PTP is defined by analogy with DTP by reversing the positive and the negative authorizations.

Example 2: First matching (cms_2)

We mentioned the conflict management strategy implemented in firewalls. We considered in particular the first matching strategy. In such a strategy, the first authorization wins. This includes the assumption that the set of authorizations is ordered. In order to model this strategy, we suppose that $\mathcal{L} = \mathbb{N}$ and that two authorizations cannot have the same priority level. In this strategy, the priority level assignment must be carried out by the SSO. If a conflict appears, the authorization having the higher level takes precedence. This strategy is denoted cms_2.

Example 3: Priority based on role (cms_3)

In this strategy, a priority relation between roles is defined to solve conflicts [Cholvy and Cuppens 1997]. It is close to the notion of distance described in sections 2.4.3. This relation of priority is generally compatible with the hierarchy defined for permission and prohibition inheritance between roles. Let us call this strategy cms_3. It is modelled using a set of priority levels $\mathcal{L} = \mathcal{R}$ where $\mathcal{R}$ represents the set of roles; and the set $\mathcal{R}$ is associated with a partial order relation. Then if a permission or a prohibition is assigned to role r, then r also represents the priority level of this permission or prohibition.

- $\mathcal{L} = \mathcal{R}$ and the set $\mathcal{R}$ is associated with a partial order relation.
- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Permission(org, r, a, v, c) \rightarrow Permission'(org, r, a, v, c, r)$
- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Prohibition(org, r, a, v, c) \rightarrow Prohibition'(org, r, a, v, c, r)$

Example 4: Combination of cms_1 and cms_3 (cms_4)

This strategy uses in the first place priority between roles to solve conflicts. If there is still a conflict that cannot be solved using role priority, then the prohibition takes precedence (DTP). Let us call this strategy cms_4. This strategy is modelled as follows:

- $\mathcal{L} = \mathcal{R} \times \{0, 1\}$
- The partial order relation over $\mathcal{L}$ is defined as follows:
  - $\forall r_1 \in \mathcal{R}, \forall l_1 \in \mathcal{L}, \forall r_2 \in \mathcal{R}, \forall l_2 \in \mathcal{L}$,
    $(r_1 \prec r_2 \wedge l_1 \in \{0, 1\} \wedge l_2 \in \{0, 1\}) \rightarrow < r_1, l_1 > \prec < r_2, l_2 >$
  - $\forall r_1 \in \mathcal{R}, \forall l_1 \in \mathcal{L}, \forall r_2 \in \mathcal{R}, \forall l_2 \in \mathcal{L}$,
    $(r_1 = r_2 \wedge l_1 \prec l_2) \rightarrow < r_1, l_1 > \prec < r_2, l_2 >$
  - $\forall r_1 \in \mathcal{R}, \forall l_1 \in \mathcal{L}, \forall r_2 \in \mathcal{R}, \forall l_2 \in \mathcal{L}$,
    $(r_1 \prec \succ r_2 \wedge l_1 \prec l_2) \rightarrow < r_1, l_1 > \prec < r_2, l_2 >$
- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Permission(org, r, a, v, c) \rightarrow Permission'(org, r, a, v, c, < r, 0 >)$
- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Prohibition(org, r, a, v, c) \rightarrow Prohibition'(org, r, a, v, c, < r, 1 >)$

7.3.2 Step 2: From organizational to concrete authorizations

Here, we focus on the derivation rule that enables us to derive the authorization $Is_permitted'$ – resp. $Is_prohibited'$ – from the organizational prioritized authorization $Permission'$ – resp. $Prohibition'$. Rules RG_1 and RG_2 (see section 3.6) used to derive concrete authorizations from organizational authorizations in theory T_{pol} do not hold anymore. RG_1 is replaced by the following rule RG'$_1$:

- RG'$_1$: $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}, \forall l \in \mathcal{L}, \forall s \in S, \forall \alpha \in A, \forall o \in O,$
  $Permission'(org, r, a, v, c, l) \wedge$
  $Empower(org, s, r) \wedge$
  $Consider(org, \alpha, a) \wedge$
  $Use(org, o, v) \wedge$
  $Hold(org, s, \alpha, o, c)$
  $\rightarrow Is_permitted'(s, \alpha, o, l)$

This rule specifies that $Is_permitted'$ may be derived with the same priority level as $Permission'$, provided that other conditions in the premises are satisfied. A similar rule called RG'$_2$ applies to derive $Is_prohibited'$ from $Prohibition'$.

7.3.3 Step 3: Deriving explicit permissions

Let us now consider the last step of the derivation process which corresponds to the derivation of the concrete authorizations $Is_permitted$ – resp. $Is_prohibited$ – from the prioritized concrete authorizations $Is_permitted'$ – resp. $Is_prohibited'$. The general idea of this derivation is the following one: we derive a concrete authorization $Is_permitted$ provided that it is not possible to derive a concrete authorization $Is_prohibited$ having a higher priority level. This is modelled with the rule ED_1:

- ED_1: $\forall s \in S, \forall \alpha \in A, \forall o \in O, \forall l_1 \in \mathcal{L},$
  $Is_permitted'(s, \alpha, o, l_1) \wedge$
  $\neg \exists l_2 \in \mathcal{L}, (l_1 \prec l_2 \wedge Is_prohibited'(s, \alpha, o, l_2))$
  $\rightarrow Is_permitted(s, \alpha, o)$

This rule says that a concrete permission can be derived in order to allow subject s to perform action α on object o if this permission is labelled at a priority level l_1 and if there is no prohibition for s to perform α on o with a priority level l_2 strictly higher than l_1. The derivation rule ED_2 for concrete negative authorizations is defined in a similar way:

- ED_2: $\forall s \in S, \forall \alpha \in A, \forall o \in O, \forall l_1 \in \mathcal{L},$
  $Is_prohibited'(s, \alpha, o, l_1) \wedge$
  $\neg \exists l_2 \in \mathcal{L}, (l_1 \prec l_2 \wedge Is_permitted'(s, \alpha, o, l_2))$
  $\rightarrow Is_prohibited(s, \alpha, o)$

So far, we explained the new derivation process used to derive concrete authorizations from organizational authorizations. The derivation process provides means to specify conflict management strategies. In the next section, we give a formal definition of the new logic theory.

7.4 The prioritized theory T_{Ppol}

New authorizations as well as new predicates were introduced with the objective to manage conflicts in an Or-BAC policy. As a consequence the logic theory specified in section 5.3 is no longer valid. In this section we define a new theory, denoted T_{Ppol}, and we give the new definition of a conflicting policy.

7.4.1 Inheritance management

In the first place, since new predicates are introduced, some other rules than the one specified in the previous section have to be modified. Such modifications are applied to the inheritance rules.

The Rule RH_1 (section 4.1.2) that was used to model authorization inheritance between roles is no longer available in the prioritized theory T_{Ppol}. It is replaced by the following rule:

- RH'$_1$: $\forall org \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}, \forall l \in \mathcal{L}$,
  $Permission'(org, r_1, a, v, c, l) \wedge$
  $sub_role(org, r_2, r_1)$
  $\rightarrow Permission'(org, r_2, a, v, c, l)$

Rule RH'$_1$ says that if role r_2 is a sub-role of r_1 and if there is a permission associated with r_1 at a priority level l, then r_2 inherits this permission *with the same priority level* l. If we were using rule RH_1, r_2 would inherit permission from r_1 but possibly with a different priority level (for instance apply the strategy priority based on role) which is generally not satisfactory.

Rules RH_2, VH_1, VH_2, AH_1 and AH_2 are similarly replaced by rules RH'$_2$, VH'$_1$, VH'$_2$, AH'$_1$ and AH'$_2$ in the prioritized theory. The rule OH_1 (section 4.1.4) must also be modified. It is replaced by the following rule OH'$_1$. Similarly, the rule OH'$_2$ stands in for the rule OH_2.

- OH'$_1$: $\forall org_1 \in Org, \forall org_2 \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}, \forall l \in \mathcal{L}$,
  $sub_organization(org_2, org_1)$
  $Permission(org_1, r, a, v, c, l) \wedge$
  $Relevant_role(org_2, r) \wedge$
  $Relevant_activity(org_2, a) \wedge$
  $Relevant_view(org_2, v) \wedge$
  $Relevant_context(org_2, c)$
  $\rightarrow Permission(org_2, r, a, v, c, l)$

Furthermore, constraints C_5 and C_6 must also be modified. We define C'$_5$ and C'$_6$ by replacing the organizational authorizations by the corresponding prioritized organizational authorizations (see section A.7).

7.4.2 Specification of theory T_{Ppol}

Thanks to the rules defined in section 7.3 and in section 7.4.1, we are now in position to give the new logic theory for managing the Or-BAC model.

Definition 7.4.1. Prioritized theory T_{Ppol}
"A prioritized security policy *Ppol* associated with a conflict management strategy *cms* is modelled as a logic theory T_{Ppol} – called prioritized theory – that is similar to theory T_{pol} except for the following points:

- Rules RG_1 and RG_2 are replaced by rules RG'$_1$ and RG'$_2$;
- rules ED_1 and ED_2 belong to theory T_{Ppol};
- rules RH_1, RH_2, VH_1, VH_2, AH_1, AH_2, OH_1 and OH_2 are replaced by rules RH'$_1$, RH'$_2$, VH'$_1$, VH'$_2$, AH'$_1$, AH'$_2$, OH'$_1$ and OH'$_2$;
- constraints C_5 and C_6 are replaced by constraints C'$_5$ and C'$_6$;
- there is a set of rules that defines *cms*: (1) rules or facts that define the partial order relation on priority levels, and (2) a set of rules that concludes on predicates $Permission'$ and $Prohibition'$".

The conflict definition remains the same as in T_{pol}:

Definition 7.4.2. Conflicting policy in T_{Ppol}
There is a conflict in security policy *Ppol* if it is possible to derive $conflict()$ from T_{Ppol}:

$$T_{Ppol} \vdash conflict()$$

7.4.3 Conflict in the prioritized theory

Notice that all conflict management strategies do not prevent conflicts from occurring in the prioritized theory T_{Ppol}. Using rules ED_1, ED_2 and RC, we can actually prove that a conflict in theory T_{Ppol} occurs when the following condition is satisfied:

- $C_{Conflict}$: $\exists s \in S, \exists \alpha \in A, \exists o \in O, \exists l_1 \in \mathcal{L}, \exists l_2 \in \mathcal{L},$
  $Is_permitted'(s, \alpha, o, l_1) \wedge$
  $Is_prohibited'(s, \alpha, o, l_2) \wedge$
  $\neg \exists l_3 \in \mathcal{L}, (l_2 \prec l_3 \wedge Is_permitted'(s, \alpha, o, l_3)) \wedge$
  $\neg \exists l_4 \in \mathcal{L}, (l_1 \prec l_4 \wedge Is_prohibited'(s, \alpha, o, l_4))$
- **Proof:** The proof is trivial. We merely have to observe that if condition $C_{Conflict}$ holds, then we can apply rules ED_1 and ED_2 to derive $Is_permitted(s, \alpha, o)$ and $Is_prohibited(s, \alpha, o)$. Next, we can derive $conflit()$ by rule RC.

Let us comment the negative conditions in $C_{Conflict}$. Let us assume that there is a level l_3 such that $l_2 \prec l_3$ and that subject s is permitted to perform action α on object o at level l_3. In this case, the conflict is solved since this permission at level l_3 overwrites prohibition at level l_2. This explains the negative condition $\neg \exists l_3 \in \mathcal{L}, (l_2 \prec l_3 \wedge Is_permitted'(s, \alpha, o, l_3)$. Explanation of the other negative condition is similar but for prohibition.

It is easy to prove that this condition can never be satisfied in strategies cms_1 (denial takes precedence), cms_2 (first matching) and cms_4 (combination of cms_1 and cms_3). However, this is not the case of strategy cms_3 in which condition $C_{Conflict}$ can be satisfied. In order to characterize these two different situations we define the notion of "effective" and "weak" strategies:

Definition 7.4.3. Effective Strategy
"A conflict management strategy is *effective* provided condition $C_{Conflict}$ cannot be satisfied when this strategy is used".

Definition 7.4.4. Weak Strategy
"A conflict management strategy is *weak* if it is not effective".

Hence, strategies cms_1, cms_2 and cms_4 are effective whereas strategy cms_3 is weak. From now on, there are two possible behaviors. The first one is to only accept effective strategies. However, we guess that, in many situations, this would be too restrictive to oblige the policy administrator to use only effective strategies. Another attitude consists in accepting a weak strategy and controlling that a conflict does not occur in the security policy to be enforced. We further investigate this approach in the following section.

7.5 Conflict prevention

Let us recapitulate what we have defined so far. We presented a parametric conflict management strategy which makes it possible to specify effective or weak strategies. We defined a new logic theory T_{Ppol}. Furthermore, we established that if the condition $C_{Conflict}$ is satisfied then a conflict occurs between a concrete permission and a concrete prohibition. In the case of an efficient strategy the management we proposed is reliable. On the contrary, if the chosen strategy is weak, we have to further investigate the conflict issue.

When a weak strategy is used, condition $C_{Conflict}$ can be satisfied. This condition involves some subjects, actions and objects. This is not satisfactory for two reasons. First, the security policy is not stable in the sense that it can be free of conflict at a given time, when in fact, a conflict might occur when the concrete level is updated (that is, when a subject is newly-empowered in a new role; or an object is newly-used in a new view; or an action is newly-considered as an activity). Second, this is not compliant with the basic principle of the Or-BAC model. Indeed, Or-BAC is designed to provide means to specify a security policy at an organizational level, that is, independently of the actual subjects, objects and actions used in the system. Using condition $C_{Conflict}$, conflicts are managed at the concrete level. A better approach would be to manage conflicts at the organizational level, with the following objective: if there is no conflict in the security policy at the organizational level, then this will guarantee that there is no conflict at the concrete level even if a weak strategy is used.

We suggest to specify a new condition that will stand in for condition $C_{Conflict}$, and which will apply at the organizational policy level. In other words, we want to detect conflicts between organizational authorizations. The objective of such a condition is to prevent conflicting situations and thus to detect the potential conflicts. This new condition is not easy to specify. Indeed, if this condition is too strong, we might create some "false negatives", this is, we might detect some conflicts that would never occur. Afterwards, we suggest a first condition. This condition will then be refined in the forthcoming sections.

7.5.1 Conflict prevention: first proposal

We suggest a first condition to prevent conflicts at the organizational level of Or-BAC. It is obtained by simply replacing concrete authorizations by organizational authorizations in condition $C_{Conflict}$ presented in section 7.4.3. It is thus defined as follows:

- $C'_{Conflict1}$: $\exists org_1 \in Org, \exists r_1 \in \mathcal{R}, \exists a_1 \in \mathcal{A}, \exists v_1 \in \mathcal{V}, \exists c_1 \in \mathcal{C}, \exists l_1 \in \mathcal{L},$
  $\exists org_2 \in Org, \exists r_2 \in \mathcal{R}, \exists a_2 \in \mathcal{A}, \exists v_2 \in \mathcal{V}, \exists c_2 \in \mathcal{C}, \exists l_2 \in \mathcal{L},$
  $Permission'(org_1, r_1, a_1, v_1, c_1, l_1) \wedge$
  $Prohibition'(org_2, r_2, a_2, v_2, c_2, l_2) \wedge$
  $\neg\exists l_3 \in \mathcal{L}, (l_1 \prec l_3 \wedge Prohibition'(org_1, r_1, a_1, v_1, c_1, l_3)) \wedge$
  $\neg\exists l_4 \in \mathcal{L}, (l_2 \prec l_4 \wedge Permission'(org_2, r_2, a_2, v_2, c_2, l_4))$

We can then prove the following theorem:

Theorem 7.5.1. Let *Ppol* be a prioritized security policy. We have:

$$T_{Ppol} \vdash conflict() \Rightarrow T_{Ppol} \vdash C'_{Conflict1}$$

Proof. The only way to derive $conflict()$ in T_{Ppol} is by applying rule RC. This means that there is a subject s, an action α and an object o such that $Is_permitted(s, \alpha, o) \wedge Is_prohibited(s, \alpha, o)$. In T_{Ppol}, the only way to derive these facts is by applying rules ED_1 and ED_2. So, we conclude that there is a priority level l_1 such that $Is_permitted'(s, \alpha, o, l_1)$ and there is no l_2 such that $(l_1 \prec l_2 \wedge Is_prohibited'(s, \alpha, o, l_2))$. But the only way to derive $Is_permitted'(s, \alpha, o, l_1)$ is by applying rule RG'$_1$. So the premises of rule RG'$_1$ are true and we can conclude that there is an organization org_1, a role r_1, an activity a_1, a view v_1 and a context c_1 such that $Permission'(org_1, r_1, a_1, v_1, c_1, l_1)$. We can similarly derive $Prohibition'(org_2, r_2, a_2, v_2, c_2, l_2)$. Now let us assume that there is a priority level l_3 such that $(l_1 \prec l_3 \wedge Prohibition'(org_1, r_1, a_1, v_1, c_1, l_3))$. Since the premises of rule RG'$_1$ are true, we can derive that $Is_prohibited'(s, \alpha, o, l_3)$. But this is in contradiction with the conclusion that there is no l_2 such that $(l_1 \prec l_2 \wedge Is_prohibited'(s, \alpha, o, l_2))$. Assuming that there is a priority level l_4 such that $(l_1 \prec l_4 \wedge Permission'(org_1, r_1, a_1, v_1, c_1, l_4))$ leads to a similar contradiction. Thus, we can conclude that $Conflict'_1$ is true.

Lemme 7.5.1. From theorem 7.5.1, by contraposition, we also have:

$$T_{Ppol} \nvdash C'_{Conflict1} \Rightarrow T_{Ppol} \nvdash conflict()$$

This means that if we cannot derive $Conflict'_1$ from T_{Ppol} then there is no conflict in the security policy defined by *Ppol*. Therefore, it is sufficient to check condition $C'_{Conflict1}$ to prevent a conflict in the policy. This is interesting because condition $C'_{Conflict1}$ can be checked at the organizational level.

However, $C'_{Conflict1}$ is a very strong condition because it requires that there is no conflict between *every* pair of organizations, roles, activities, views and contexts. We can actually suggest a weaker condition by considering separation constraints.

7.5.2 Conflict prevention: second proposal

This section suggests a weaker condition than $C'_{Conflict1}$ to prevent conflicts. It is defined as follows:

- $C'_{Conflict2}$: $\exists org_1 \in Org, \exists r_1 \in \mathcal{R}, \exists a_1 \in \mathcal{A}, \exists v_1 \in \mathcal{V}, \exists c_1 \in \mathcal{C}, \exists l_1 \in \mathcal{L},$
  $\exists org_2 \in Org, \exists r_2 \in \mathcal{R}, \exists a_2 \in \mathcal{A}, \exists v_2 \in \mathcal{V}, \exists c_2 \in \mathcal{C}, \exists l_2 \in \mathcal{L},$
  $Permission'(org_1, r_1, a_1, v_1, c_1, l_1) \wedge$
  $Prohibition'(org_2, r_2, a_2, v_2, c_2, l_2) \wedge$
  $\neg \exists l_3 \in \mathcal{L}, (l_1 \prec l_3 \wedge Prohibition'(org_1, r_1, a_1, v_1, c_1, l_3)) \wedge$
  $\neg \exists l_4 \in \mathcal{L}, (l_2 \prec l_4 \wedge Permission'(org_2, r_2, a_2, v_2, c_2, l_4)) \wedge$
  $\neg separated_role(org_1, r_1, org_2, r_2) \wedge$
  $\neg separated_activity(org_1, a_1, org_2, a_2) \wedge$
  $\neg separated_view(org_1, v_1, org_2, v_2) \wedge$
  $\neg separated_context(org_1, c_1, org_2, c_2)$

Predicates *separated_role*, *separated_activity* and *separated_view* were already presented in section 4.2.3 and *separation_context* in section 6.10.

Theorem 7.5.2. Let *Ppol* be a prioritized security policy. We have:

$$T_{Ppol} \vdash conflict() \wedge T_{Ppol} \nvdash error() \Rightarrow T_{Ppol} \vdash C'_{Conflict2}$$

Proof. Proof is similar to theorem 7.5.1. We have simply to observe that if a conflict occurs in theory *Ppol* for a given subject s, then this means that this subject is empowered in a given role r_1 by a given organization org_1 (to derive that this subject is permitted to perform a given action on a given object) and this subject is empowered in another role r_2 by another organization org_2 (to derive that this subject is prohibited to perform the same action on the same object). However, if we have $separation_role(r_1, org_1, r_2, org_2)$, then we could derive that a separation constraint is violated. But this is in contradiction with the assumption that *Ppol* does not violate any separation constraint. We can similarly derive the negation of other separation constraints that appear in $C'_{Conflict2}$.

Theorem 7.5.3. Let *Ppol* be a prioritized security policy. We have:

$$T_{Ppol} \vdash Conflict'_2 \Rightarrow T_{Ppol} \vdash Conflict'_1$$

Proof. Trivial.

$C'_{Conflict2}$ provides a weaker condition than $C'_{Conflict1}$ since it is sufficient to check pairs of roles, activities, views and contexts for which no separation occurs. In the following, we can further weaken this condition.

7.5.3 Conflict prevention: last proposal

Our last proposal for conflict prevention is defined as follows:

- $C'_{Conflict3}$: $\exists org_1 \in Org, \exists r_1 \in \mathcal{R}, \exists a_1 \in \mathcal{A}, \exists v_1 \in \mathcal{V}, \exists c_1 \in \mathcal{C}, \exists l_1 \in \mathcal{L},$
  $\exists org_2 \in Org, \exists r_2 \in \mathcal{R}, \exists a_2 \in \mathcal{A}, \exists v_2 \in \mathcal{V}, \exists c_2 \in \mathcal{C}, \exists l_2 \in \mathcal{L},$
  $Permission'(org_1, r_1, a_1, v_1, c_1, l_1) \wedge$
  $Prohibition'(org_2, r_2, a_2, v_2, c_2, l_2) \wedge$
  $\neg separated_role(org_1, r_1, org_2, r_2) \wedge$
  $\neg separated_activity(org_1, a_1, org_2, a_2) \wedge$
  $\neg separated_view(org_1, v_1, org_2, v_2) \wedge$
  $\neg separated_context(org_1, c_1, org_2, c_2)$
  $\neg \exists l_3 \in \mathcal{L}, (i, j, k, l, m) \in \{1, 2\},$
  $((l_1 \prec l_3 \wedge Prohibition'(org_i, r_j, a_k, v_l, c_m, l_3)) \vee$
  $(l_2 \prec l_3 \wedge Permission'(org_i, r_j, a_k, v_l, c_m, l_3)))$

Theorem 7.5.4. Let *Ppol* be a prioritized security policy. We have:

$$T_{Ppol} \vdash conflict() \wedge T_{Ppol} \nvdash error() \Rightarrow T_{Ppol} \vdash C'_{Conflict2}$$

Proof. Proof starts like theorem 7.5.1 to conclude that both premises of rules RG'$_1$ and RG'$_2$ are true. Now let us assume that there is a level l_3 such that $l_1 \prec l_3 \wedge Prohibition'(org_i, r_j, a_k, v_l, c_m, l_3)$ with values of subscripts i, j, k, l, m belonging to $\{1, 2\}$. Then we can still apply rule RG'$_1$ (using premises of rules RG'$_1$ or RG'$_2$ when subscript is respectively equal to 1 or 2) to derive $Is_prohibited(s, \alpha, o, l_3)$ which is a contradiction. We obtain a similar contradiction if we assume that there is a level l_3 such that $l_2 \prec l_3 \wedge Permission'(org_i, r_j, a_k, v_l, c_m, l_3)$.

7.5.4 Example

In order to compare the various conditions suggested in the previous sections, let us consider the policy defined by the following facts. This policy is prioritized; this means that the first step of the derivation process is already realized.

Organizational policy:

- A$_1$: $Permission'(Bank, adviser, consulting, customer_account, Default, l_1)$
  "In *Bank*, advisers are allowed to consult the customer accounts with a priority level l_1"
- A$_2$: $Prohibition'(Bank, counter_clerk, consulting, company_account, Default, l_2)$
  "In *Bank*, clerks are prohibited to consult the company accounts with a priority level l_2"
- A$_3$: $Permission'(Bank, adviser, consulting, company_account, Default, l_3)$
  "In *Bank*, counter advisers are allowed to consult the company accounts with a priority level l_3"
- $l_2 \prec l_3 \wedge l_2 \prec\!\succ l_1$
  "Priority l_3 is higher than l_2. l_2 and l_1 are not comparable. The conflict management strategy is weak"
- $\neg separated_role(Bank, counter_clerk, Bank, adviser)$
  "In *Bank*, roles counter clerk and adviser are not separated (meaning that a given subject can be empowered in both roles)"

- $\neg separated_view(Bank, customer_account, Bank, company_account)$
  "In *Bank*, views customer account and company account are not separated (meaning that a given object can be used in both views)"

Concrete policy:

- $Empower(Bank, John, adviser) \wedge Empower(Bank, John, counter_clerk)$
  "In *Bank*, John is assigned to both roles adviser and clerk"
- $Consider(Bank, SELECT, consulting)$
  "In *Bank*, action SELECT partake to activity consult"
- $Use(Bank, doc_1, customer_account) \wedge Use(Bank, doc_1, company_account)$
  "In *Bank*, doc_1 is used in both views customer account and company account"

Consider now step 2 of the derivation process. From this policy, and using rules RG'$_1$ and RG'$_2$ (section 7.3.2), we derive the following concrete authorizations:

- $Is_permitted'(John, SELECT, doc_1, l_1)$
- $Is_permitted'(John, SELECT, doc_1, l_3)$
- $Is_prohibited'(John, SELECT, doc_1, l_2)$

We go now on step 3. Using ED$_1$ (section 7.3.3), we can derive $Is_permitted(John, SELECT, doc_1)$ whereas we cannot apply rule ED$_2$, because $l_2 \prec l_3$. Even though the chosen strategy is weak, there is actually no conflict in this example.

Let us examine our potential conflict conditions $C'_{Conflict1}$, $C'_{Conflict2}$ and $C'_{Conflict3}$. Since $l_2 \prec l_1$, condition $C'_{Conflict1}$ is satisfied, due to the conflict between the two first authorization A$_1$ and A$_2$. Therefore, when applying this condition, we would detect a potential conflict. This condition is actually too strong.

Although more restrictive, $C'_{Conflict2}$ is also satisfied. Indeed, the roles adviser and counter clerk are not separated. $C'_{Conflict2}$ is still a too strong requirement.

Finally, $C'_{Conflict3}$ cannot be derived. This is due to the third authorization A$_3$. This is satisfactory since no conflict occurs in this policy.

This example shows that our approach is useful to check that a prioritized security policy is free from conflicts before implementing it in a concrete system.

7.6 Redundant authorization detection

A consequence of introducing some priority levels is that some redundant authorizations may appear. In order to explain when redundant rules occur, let us consider the following policy:

- A$_1$: $Permission'(Bank, adviser, consulting, account, Default, l_1)$
  "In *Bank*, advisers are allowed to consult the accounts with a priority level l_1"
- A$_2$: $Prohibition'(Bank, adviser, consulting, account, Default, l_2)$
  "In *Bank*, advisers are prohibited to consult the accounts with a priority level l_2"

- A_3: $Permission'(Bank, employee, consulting, account, Default, l_3)$
  "In $Bank$, employees are allowed to consult the accounts with a priority level l_3"
- $l_1 \prec l_2 \prec l_3$
  "Priority l_3 is higher than l_2, which is higher than l_1"
- $sub_role(Bank, adviser, employee)$
  "In $Bank$, role adviser is a sub-role of role employee"

Clearly A_1 is redundant beside A_2 since these authorizations apply to the same entities and A_2 is associated to a higher priority level than A_1. As a consequence, A_1 is useless in the policy. There is actually no need to have authorizations involving the same entities to obtain a redundant authorization. Consider for example A_2 and A_3. The role adviser is a sub-role of role employee, so the role adviser inherits authorization A_3. Since A_3 is associated to a higher priority level than A_2, A_2 is redundant beside A_3. The same line of reasoning holds between A_1 and A_3. As a consequence, just one authorization out of three is useful in this security policy example. Even though we consider only a role hierarchy in this example, the same phenomenon would appear with activity, view and context hierarchy.

This is useful for the SSO to detect the redundant rules. Indeed, due to the inheritance mechanisms, some important authorizations might be discarded. This can be considered as a side-effect of the use of hierarchies and priority levels. Therefore, in order to detect the redundant authorizations we define a new constraint C_{11}.

Redundant authorizations cannot be detected by comparing two derived authorizations, this is, authorizations resulting from one of the derivation rules specified in Or-BAC, otherwise all authorizations involved in a potential conflict and which do not take precedence would be declared as redundant. As a consequence, we introduce two new predicates for the derived authorizations $D_Permission'$ and $D_Prohibition'$. In the context of the redundant authorizations detection, we assume that all derivation rules defined in the Or-BAC that conclude on predicates $Permission'$ and $Prohibition'$, conclude now on predicates $D_Permission'$ and $D_Prohibition'$. This holds for inheritance derivation rules for instance. We also add the following rules:

- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}, \forall l \in \mathcal{L}$
  $Permission'(org, r, a, v, c, l) \rightarrow D_Permission'(org, r, a, v, c, l)$
- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}, \forall l \in \mathcal{L}$
  $Prohibition'(org, r, a, v, c, l) \rightarrow D_Prohibition'(org, r, a, v, c, l)$

The redundant authorizations detection only considers explicit authorizations, that is, predicates $Permission'$ and $Prohibition'$.

Notice that the redundant authorization issue does not take care of the authorizations' type: an authorization is redundant beside another one, wether these authorizations are positive or negative. In order to model this feature and to better specify the constraint C_{11}, we introduce a new predicate $Authorization'$. It enables us not to make any distinction between positive and negative authorizations. It is defined as follows:

- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}, \forall l \in \mathcal{L}$
  $Permission'(org, r, a, v, c, l) \rightarrow Authorization'(org, r, a, v, c, l)$
- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}, \forall l \in \mathcal{L}$
  $Prohibition'(org, r, a, v, c, l) \rightarrow Authorization'(org, r, a, v, c, l)$

We are now in position to specify the constraint C_{11}:

- C_{11}: $\exists org \in Org, \exists r_1 \in \mathcal{R}, \exists a_1 \in \mathcal{A}, \exists v_1 \in \mathcal{V}, \exists c_1 \in \mathcal{C}, \exists l_1 \in \mathcal{L},$
  $\exists r_2 \in \mathcal{R}, \exists a_2 \in \mathcal{A}, \exists v_2 \in \mathcal{V}, \exists c_2 \in \mathcal{C}, \exists l_2 \in \mathcal{L},$
  $Authorization'(org, r_1, a_1, v_1, c_1, l_1) \wedge$
  $Authorization'(org, r_2, a_2, v_2, c_2, l_2) \wedge$
  $sub_role(org, r_2, r_1) \wedge$
  $sub_activity(org, a_2, a_1) \wedge$
  $sub_view(org, v_2, v_1) \wedge$
  $sub_context(org, c_2, c_1) \wedge$
  $l_2 \prec l_1$
  $\rightarrow error()$

Note that relation *sub_role*, *sub_activity*, *sub_view* and *sub_context* are reflexive and transitive. This constraint handles the redundant authorizations due to hierarchies within a single organization. Now, redundant authorizations might also appear as a result of inheritance between organizations which is modelled with OH'$_1$ and OH'$_2$ (see section 4.1.4). Thereby, we must add another constraint:

- C_{12}: $\exists org_1 \in Org, \exists org_2 \in Org, \exists r \in \mathcal{R}, \exists a \in \mathcal{A}, \exists v \in \mathcal{V}, \exists c \in \mathcal{C}, \exists l_1 \in \mathcal{L}, \exists l_2 \in \mathcal{L},$
  $Authorization'(org_1, r, a, v, c, l_1) \wedge$
  $Authorization'(org_2, r, a, v, c, l_2) \wedge$
  $sub_org(org_2, org_1) \wedge$
  $l_2 \prec l_1$
  $\rightarrow error()$

7.7 Conclusion

In this paper, we presented a logical approach to manage conflicts in an access control policy modelled in Or-BAC. Since a policy in Or-BAC is defined at an organizational level (i.e. independently from actual implementation of subjects, objects and actions in the system), we also suggest managing conflicts at the organizational level. Our approach is based on defining a parametric conflict management strategy that is used to assign priority levels to the organizational permissions or prohibitions. In order to do so, we redefine the derivation process between organizational and concrete authorizations. Two different situations may arise when using a strategy. (1) The strategy is effective; this guarantees that the strategy will solve every conflict that may possibly occur in the policy. (2) The strategy is weak; in this case, we defined a condition and proved that this condition, when it is satisfied, prevents conflicts from occurring at the concrete level. This means that the security administrator can actually use any strategy. As far as we know, we guess it is the first time such an approach is defined and formally modelled to manage conflict. Furthermore, since redundant authorizations might appear using a conflict management strategy, we establish two constraints to detect such authorizations.

One might argue that context validation raises a difficulty with respect to the conflict detection issue. Since contexts enable us to activate or deactivate authorizations, some conflicts might occur between pairs of authorizations provided their contexts are valid at the same time. However, the predicate *separated_context* is precisely used to specify the pairs of

contexts that can never be "conflicting". We actually assume that the fact that a separation constraint between two contexts matches the definitions of these contexts (that is, the rules concluding on predicate $Hold$) is carried out beforehand. As a consequence, the context validation does not interfere with the conflict prevention.

Chapter 8

AdOr-BAC: The administration model for Or-BAC

8.1 Introduction

In section 2.5 we focused on administration and discussed existing models. In the access control area, administration consists in distributing some authorizations in order to update the security policy. This is indeed essential to specify some administrative procedures. Otherwise, the security policy will not remain in adequation with the information system. In an organization, if a new subject is empowered, it must be possible for instance to assign some authorizations or some roles to this subject. We concluded that an access control model should be associated with a complete administration model, and that this model should be fully compliant with the access control model. Furthermore, administration tasks can be centralized and managed by a single SSO, or distributed to several SSOs, each one being in charge of a part of the policy. Therefore, an administration model should make it possible to specify several administration strategies. We also mentioned that delegation is a really important issue, but difficult to handle in role-based models.

This chapter presents AdOr-BAC [Cuppens and Miège 2003a, Cuppens and Miège 2004a], an administration model for Or-BAC. This model is fully homogeneous with the remainder of Or-BAC and is used to define decentralized administration procedures as well as delegation procedures.

For the time being, AdOr-BAC is defined in the context of non-prioritized policies. Therefore we assume we stand in the logic theory T_{pol} afterwards.

Objectives

Administration encompasses all tasks related to the security policy update process. As a consequence, a complete Or-BAC administration model should provide means to control the following purposes:

1. Management of organizations
2. Management of roles, activities, views and contexts
3. Assignment (and revocation) of users to roles
4. Assignment (and revocation) of authorizations to roles

5. Assignment (and revocation) of users to permissions

In the above list, "management" includes creation, modification, deletion and hierarchy definition. Actually, I could not go into all these points during the thesis. Therefore, we rather focus on the last three issues. In fact, the five tasks above can be classified in another way. On the one hand, some administrative tasks (1 and 2) provide means to create the set of organizations, roles, activities, etc., and hierarchies, in other words, the organization structure. On the other hand, some tasks (3, 4 and 5) are used to "fill" this structure by assigning authorizations to roles and subjects, and roles to users. We focus first on the latter tasks. Part of the tasks 1 and 2 are briefly examined in the last section.

We discussed the ARBAC model in section 2.5. Actually, the RBAC model is associated with ARBAC [Sandhu et al. 1999, Sandhu and Munawer 1999, Oh et al. 2003]. ARBAC provides interesting solutions with respect to the centralization / decentralization issue and is one of the most complete administration models. ARBAC includes, among others, two main components:

- The URA (User-Role Assignment) component which controls who is permitted to assign a user with a new role and who is permitted to revoke a user from an existing role.
- The PRA (Permission-Role Assignment) component which controls who is permitted to assign a role with a new permission and who is permitted to revoke a role from an existing permission.

By analogy with ARBAC we also introduce a user-role assignment (URA) and a permission-role assignment (PRA[1]) component. Elsewhere, we also introduce a new component, called UPA:

- The UPA (User-Permission Assignment) component which controls who is permitted to assign a user with a new permission and who is permitted to revoke a user from an existing permission.

As it is discussed in section 8.4, UPA component is highly related to the delegation issue. Actually, this facility is useful to control fine grained permission assignment, and is more precisely used when a user wants to grant a specific authorization to another user.

Principle

The approach we suggest in AdOr-BAC is to define these administration functions by considering different Or-BAC views respectively called URA, PRA and UPA. Every organization manages such views. Objects belonging to these views have specific semantics; namely they will be respectively interpreted as an assignment of user to a role, an authorization to a role and a permission to a user. Intuitively, inserting an object in these views will enable an authorized user to respectively assign a user to a role, assign an authorization to a role and assign a permission to a user. Conversely, deleting an object from these views will enable a user to perform a revocation.

Defining the administration functions in AdOr-BAC thereby consists in specifying which roles are permitted to get an access to views URA, PRA and UPA, or to more specific views when the role does not have a complete access to one of these views.

[1]Contrary to the RBAC model, in the Or-BAC model, the PRA component is used to assign not only permissions but also prohibitions. However, we keep the component name "permission"-role assignement.

The approach we suggest is homogeneous with the remainder of the Or-BAC model: the language we use in AdOr-BAC to define permission to administer the policy is completely similar to the one in Or-BAC. Actually, strictly speaking, it is even incorrect to consider that AdOr-BAC is a distinct model from Or-BAC. Since we merely have to consider three new views, namely URA, PRA and UPA in the Or-BAC model, it would be more appropriate to say that Or-BAC is an auto-administered model. In the following we shall present the structure of these three views and further analyze the administration functions associated with the management of these views.

Notice that, in the ARBAC model, there are two types of fully separated roles called regular roles and administration roles. Administration roles are only allowed to perform administration functions and regular roles are only permitted to perform other functions excluding administration functions. In some circumstances this separation is superfluous. For instance the role $chief_adviser$ (see figure 2.7) may hold a plurality of administrative and non administrative permissions. In such case, it is not necessary to create two roles. The AdOr-BAC model does not impose to create these two roles. But, a security policy designer could legitimately want to separate them anyway, because of separation of duty and least privilege questions. The AdOr-BAC model makes it possible to do so. Thus, we leave such separation optional in the AdOr-BAC model. Keeping this separation makes AdOr-BAC compliant with ARBAC.

The remainder of this chapter is organized as following. Each following section corresponds to one of the three components of AdOr-BAC: section 8.2 is dedicated to URA, section 8.3 to PRA, and finally section 8.4 to UPA. In section 8.5 we suggest other components which describe the organization structure.

8.2 URA in AdOr-BAC

8.2.1 The view URA

The aim of the user-role administration activity is to determine who is allowed to assign a user to a role and under which conditions. Assigning a user to a role equals to inserting a new objet in a given view called URA. Three attributes are associated with this view. More precisely, an object ura belonging to view URA has three attributes:

- $subject(ura, s)$: designates the subject s that is related to the assignment
- $role(ura, r)$: designates the role r to which the subject will be assigned
- $org(ura, org)$: designates the organization org in which the subject is assigned

If a given role is allowed to insert any object in view URA under any condition, then this means that this role is actually allowed to assign any user to any role in any organization under any condition. However, this would generally be not restrictive enough. To enforce further restrictions, we have to define sub-views of view URA. For example, in organization org_a, in order to assign subject s to role r_2 within organization org_b, we have to create the following view called $myURA$:

- $\forall ura \in O$,
  $Use(org_a, ura, myURA)$
  $\leftarrow Use(org_a, ura, URA) \wedge$
  $subject(ura, s) \wedge$
  $role(ura, r_2) \wedge$
  $org(ura, org_b)$

In the Or-BAC model, an organization empowers a user in a role. It is characterized by the relationship *Empower*. Therefore, there is a link between the object belonging to the view URA and the relationship *Empower*. This link is modelled through the following rule:

- $\forall org_a \in Org, \forall ura \in O$,
  $Use(org_a, ura, URA) \wedge$
  $org(ura, org) \wedge subject(ura, s) \wedge role(ura, r)$
  $\rightarrow Empower(org, s, r)$

This rule means that from each object ura belonging to the view URA (or to any sub-views of URA), we can derive that a given subject (corresponding to $subject(ura, s)$) is empowered in a given role (corresponding to $role(ura, r)$) in a given organization (corresponding to $org(ura, org)$). Notice that in this rule, there are actually two different organizations, namely org_a and org. This means that a user empowered in a given organization corresponding to org_a can actually manage the user-role administration activity of another organization org. For instance, org_a might be the human resources department of a given company and org might be the different departments of this company.

8.2.2 Managing the view URA

So far, we described how to create a view URA. We have to determine how to use such a view in order to grant permission to a role to assign users to roles. There are three different activities that apply to view URA: *manage*, *assign* and *revoke*. The activity *assign* corresponds to assigning a user to a role through the view URA. For instance, the permission granted to the role r_1 to assign a user to the role r_2 in the sub-organization org_b of org_a is expressed as follows:

- $Permission(org_a, r_1, assign, myURA, context)$

This means that a subject playing the role r_1 is allowed to insert an object in the view $myURA$ if this object corresponds to the definition of this view. This is similar to the "WITH CHECK OPTION" used in relational databases to control that an object can be inserted in a view only if this object matches the definition of the view.

Up to now, we have only dealt with assignment but not with revocation. For this purpose, we use the activity *revoke*. For instance, we may specify:

- $Permission(org_a, r_1, revoke, myURA, context)$

When a role is authorized to both assign and revoke users to a specific role, we create the activity *manage*, and consider the activities *assign* and *revoke* as two sub-activities of *manage*. This means that, if a given role is permitted to *manage* a given view in a given context, then this role inherits the right to perform the activities *assign* and *revoke* on this view in the same context.

Note that these administrative authorizations can become dynamic authorizations by defining a context. Furthermore, remember that in Or-BAC we make a distinction between the activity and the action that actually implements this activity. This means that implementation of activities *assign* and *revoke* may change from one organization to another. For instance, in a relational database, activity *assign* may be implemented by the action "INSERT" (of objects in view URA) and activity *revoke* by the action "DELETE" (of objects in view URA). If this is the case in organization B, it is specified by the following facts:

- $Consider(B, INSERT, assign)$
- $Consider(B, DELETE, revoke)$

Furthermore, one can notice that it is possible to specify that a given role r_1 is granted permission to assign a user to role r, and that another role r_2 is granted permission to revoke the user from role r. This might be useful to specify that revocation is a matter of a higher role than r_1 in the role hierarchy.

8.2.3 Example

Let us use again our banking example and assume that the bank $Trusted_bank$ is composed, among others, of a financial department called $trusted_finance$. If the role $Chief_financial_adviser$ is only allowed to assign a user to the role $financial_adviser$ in the financial department, we have to create a specific view, called $URA_financial_adviser$ for example. This view is a sub-view of URA and is defined as follows:

- $\forall ura \in O$
  $Use(Trusted_bank, ura, URA_financial_adviser)$
  $\leftarrow Use(Trusted_bank, ura, URA) \wedge$
  $role(ura, financial_adviser) \wedge$
  $org(ura, trusted_finance)$

Therefore, the permission granted to the role $Chief_financial_adviser$ to assign a user to the role *adviser* in the financial department $trusted_finance$ of $Trusted_bank$ is expressed as follows:

- $Permission(Trusted_bank,$
  $Chief_financial_adviser, assign, URA_financial_adviser, Default)$

Let us consider another example: in $Trusted_bank$, the head staff of the human resources department is granted the permission to assign user John in the role $financial_adviser$ in the department $trusted_finance$:

- $Permission(Trusted_bank, head_staff, assign, URA_John_financial_adviser, Default)$

where the view $URA_John_financial_adviser$ is defined as follows:

- $\forall ura \in O,$
  $Use(Trusted_bank, ura, URA_John_financial_adviser)$
  $\leftarrow Use(Trusted_bank, ura, URA_financial_adviser) \wedge$
  $subject(ura, John)$

8.2.4 The prerequisite conditions

In the ARBAC model, there is the following ternary relation *can_assign*:

- $can_assign(admin_role, regular_role, prerequisite_role)$

This relation is used to specify that an administrative role *admin_role* is permitted to assign users to regular role *regular_role* provided these users are empowered in some *prerequisite* roles. In AdOr-BAC, no such ternary relation exists. However, it is possible to specify a similar requirement as follows. First, a permission can be expressed as follows:

- $Permission(org, admin_role, assign, URA_regular_role, context)$

It is then possible to include the prerequisite condition when specifying the view $URA_regular_role$ as follows:

- $\forall org \in Org, \forall ura \in O,$
  $Use(org, ura, URA_regular_role)$
  $\leftarrow Use(org, ura, URA) \wedge$
  $role(ura, regular_role) \wedge$
  $subject(ura, s) \wedge$
  $Empower(org, s, prerequisite_role)$

To illustrate this approach, let us consider the following example. In $Trusted_bank$, the general manager is permitted to assign a user as the chief of the financial department, but only if this user is empowered in the role $financial_adviser$ (prerequisite condition):

- $Permission(Trusted_bank,$
  $general_manager, assign, URA_chief_financial_dpt, Default)$

where the view $URA_chief_financial_dpt$ is defined as follows:

- $\forall ura \in O,$
  $Use(Trusted_bank, ura, URA_chief_financial_dpt)$
  $\leftarrow Use(Trusted_bank, ura, URA) \wedge$
  $role(ura, Chief_financial_adviser) \wedge$
  $org(ura, trusted_finance) \wedge$
  $subject(ura, s) \wedge$
  $Empower(Trusted_bank, s, financial_adviser)$

Remember that a subject can only insert an object in a given view if this object matches the definition of this view. This means that a subject empowered in the role *general_manager* can only assign another subject in view $URA_chief_financial_dpt$ if this subject is empowered in role $financial_adviser$. We can thus specify that for the financial department, the chief must be a financial adviser.

The user-role assignment in AdOr-BAC is very flexible. A large number of conditions can be expressed such as the prerequisite conditions of ARBAC, thanks to the use of views which make it possible to model the assignments. Moreover, contrary to AdOr-BAC, the relation *can_assign* in ARBAC does not enable us to specify the organization and the context. It is also important to mention that in the AdOr-BAC model there is no distinction between the regular roles and the administrative roles as suggested in ARBAC. Since, there are

no specific permissions for the administrative tasks, such as *can_assign* and *can_revoke* in ARBAC, permissions corresponding to the activities *manage*, *assign* and *revoke* can be given to any role, and not only to specific roles such as $senior_security_officer$ or $project_security_officer$.

8.3 PRA in AdOr-BAC

In the previous section we dealt with the user-role administration. We discuss here the permission-role administration. The PRA component is used to assign positive and negative authorizations to roles. As we have just seen, we modelled user assignment to role with the view URA. Here, the authorization assignment to role is modelled with another view called PRA. Giving a new authorization to a role corresponds to inserting a new object that complies with the view PRA.

8.3.1 The view PRA

Five attributes are associated with the view PRA:

- *issuer*: the organization where the authorization applies
- *grantee*, *privilege*, *target*: designates the role, the activity and the view concerned by the authorization
- *context*: designates the context in which the authorization can be applied
- *type*: indicates if the granted authorization is a permission or a prohibition. *type* is in $\{positive, negative\}$

For example, in organization org_a, in order to grant role r permission to perform activity a on view v in context c within organization org_b, we define the following view $myPRA$:

- $\forall org_a \in org, \forall pra \in O,$
  $Use(org_a, pra, myPRA)$
  $\leftarrow Use(org_a, pra, PRA) \wedge$
  $issuer(pra, org_b) \wedge$
  $grantee(pra, r) \wedge$
  $privilege(pra, a) \wedge$
  $target(pra, v) \wedge$
  $context(pra, c) \wedge$
  $type(pra, positive)$

The link between a permission and an object belonging to view PRA and which has *type* equals to *positive* is modelled as follows:

- $\forall org_a \in Org, \forall pra \in O,$
  $Use(org_a, pra, PRA) \wedge$
  $issuer(pra, org_b) \wedge grantee(pra, r) \wedge privilege(pra, a) \wedge$
  $target(pra, v) \wedge context(pra, c) \wedge type(pra, positive)$
  $\rightarrow Permission(org_b, r, a, v, c)$

This rule means that from each object pra belonging to the view PRA (or to any sub-views of PRA) in any organization org, we can derive that, in a given organization (corresponding to $issuer(pra, org_b)$), a given role (corresponding to $grantee(pra, r)$) obtains the permission to perform a given activity (corresponding to $privilege(pra, a)$) on a given view (corresponding to $target(pra, v)$) in a given context (corresponding to $context(pra, c)$). The following rule enables us to model the link between objects that belong to the view PRA and which has the attribute $type$ negative, and prohibitions:

- $\forall org_a \in Org, \forall pra \in O,$
  $Use(org_a, pra, PRA) \wedge$
  $issuer(pra, org_b) \wedge grantee(pra, r) \wedge privilege(pra, a) \wedge$
  $target(pra, v) \wedge context(pra, c) \wedge type(pra, negative)$
  $\rightarrow Prohibition(org_b, r, a, v, c)$

8.3.2 Managing the view PRA

The same activities $assign$, $revoke$ and $manage$ defined in the previous section are used to express the authorization given to a role to assign, revoke and manage positive or negative authorizations to other roles.

8.3.3 Example

In this section, we go on with the example of section 3.6, where a customer of bank $Trusted_bank$ has the permission to consult his own accounts:

- $Permission(Trusted_bank, customer, consulting, customer_account, own_account)$

Assume that the permission is managed by role $counter_clerk$. First, we have to define the correct sub-view of view PRA. Let us call it $PRA_customer_consult$. This view is defined as follows:

- $\forall pra \in O,$
  $Use(Trusted_bank, pra, PRA_customer_consult)$
  $\leftarrow Use(Trusted_bank, pra, PRA) \wedge$
  $issuer(pra, Trusted_bank) \wedge$
  $grantee(pra, customer) \wedge$
  $privilege(pra, consulting) \wedge$
  $target(pra, customer_account) \wedge$
  $context(pra, own_account) \wedge$
  $type(pra, positive)$

Second, the following authorization will enable a subject who plays the role $counter_clerk$ to manage the authorization for customer to consult their account:

- $Permission(Trusted_bank, counter_clerk, manage, PRA_customer_consult, Default)$

8.3.4 Prerequisite conditions

We can consider many examples of constraints that apply to permission-role assignment. As a special case, let us consider the prerequisite condition suggested in the ARBAC model.

There is a ternary relation *can_assignp* defined as follows:

- $can_assignp(admin_role, regular_role, prerequisite)$

The relation *can_assignp* is used to specify that an administrative role is permitted to assign permissions to a regular role provided these permissions are already assigned to some *prerequisite* role. Notice that the meaning of the prerequisite condition differs from *can_assign* to *can_assignp*. In the *can_assign* relation, the *subject* must be empowered in the prerequisite role before being empowered in the regular role. In the *can_assignp* relation, the *permission* must be assigned to the prerequisite role before being assigned to the regular role.

In AdOr-BAC, it is possible to specify a similar prerequisite requirement as follows. To do so, we first need to specify a permission having the following form:

- $Permission(org, admin_role, assign, PRA_regular_role, context)$

We then include the prerequisite condition when specifying the view $PRA_regular_role$ as follows:

- $\forall org_a \in Org, \forall pra \in O, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$
  $Use(org_a, pra, PRA_regular_role)$
  $\leftarrow Use(org_a, pra, PRA) \wedge$
  $grantee(pra, regular_role) \wedge$
  $issuer(pra, org_b) \wedge privilege(pra, a) \wedge$
  $target(pra, v) \wedge context(pra, c) \wedge type(pra, positive) \wedge$
  $Permission(org_b, prerequesite_role, a, v, c)$

Notice that our approach makes explicit the different meanings of the prerequisite condition between *can_assign* and *can_assignp*. The same line of reasoning stands for prohibitions.

In AdOr-BAC, this prerequisite condition is just a special case of constraint we can consider in the PRA assignment. We can actually find many other examples of such constraint. For instance, we can specify that a given role is only permitted to assign particular *activities* to some regular role.

8.4 UPA in AdOr-BAC

The URA and PRA components respectively allow an authorized user to assign users to roles and organizational authorizations to roles. Thus, these components indirectly enable this authorized user to assign concrete authorizations to users. We argue that sometimes a more direct process should enable a user to grant a concrete permission to another user.

Actually this is related to the delegation issue presented in section 2.5. Delegation in access control is of utmost importance. It enables to delegate to certain users the ability to grant authorizations to other users. As described before, many delegation (and transfer) mechanisms can be specified in accordance with the ownership issue or the loss of a delegated permission issue for instance.

In role-based models, as in Or-BAC, authorizations are assigned to users through some roles. As a consequence, a user cannot give an authorization to another user as described in

section 2.5.4. The solution suggested in the RDBM model [Barka and Sandhu 2000] is not satisfactory. Actually, to solve this problem, it is suggested in the RDBM model to delegate roles, and thereby the permissions associated to these roles. This raises two problems. First, it does not allow us to specify some fine-grained delegation mechanisms, that is, delegation of permission one by one. Furthermore, a user loses his ability to play a role during the time his role is delegated. Therefore, if a user delegates his role in order to grant only one permission to another user, he looses all his permissions.

In the Or-BAC model, we introduce the new component UPA to grant permissions directly to users. Actually, Or-BAC provides two components:

- UPA is used to grant permissions to users on specific actions and objects
- UPA' is used to grant permissions to users on activities and views

We shall present first these two components and then see they can intervene in the context of delegation. Indeed, UPA and UPA' are proposals for delegation in role-based models but not only. Indeed, we do not have to assume that a user possesses a permission to delegate it, and a user does not loose a permission he delegates. One should notice that delegation is a matter of authority and responsibility. Therefore delegation of prohibition does not make sense. As a consequence we only consider permissions in UPA and UPA'.

8.4.1 UPA: granting permissions on specific actions and objects

We consider a view UPA with five attributes which have the same names as PRA but with slightly different meanings:

- *issuer*: represents the organization which is issuing the permission
- *grantee*: designates the subject who is receiving the permission
- *privilege*: represents the action the grantee is authorized to perform
- *target*: represents the object the grantee is authorized to have an access to
- *context*: designates the context in which the permission applies.

Therefore, in order to allow a role within org_a to grant permission in org_b to subject s to perform action α on object o within context c, we have to create a new view UPA called $myUPA$ and defined as follows:

- $\forall org_a \in Org, \forall upa \in O,$
  $Use(org_a, upa, myUPA)$
  $\leftarrow Use(org_a, upa, UPA) \wedge$
  $issuer(upa, org_b) \wedge$
  $grantee(upa, s) \wedge$
  $privilege(upa, \alpha) \wedge$
  $target(upa, o) \wedge$
  $context(upa, c)$

The following rule specifies that we can derive, from objects belonging to the view UPA, the fact that a subject is permitted to perform an action on an object. It is modelled as follows:

- $\forall org_a \in Org, \forall upa \in O, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall c \in \mathcal{C}$
  $Use(org_a, upa, UPA) \wedge$
  $issuer(upa, org_b) \wedge grantee(upa, s) \wedge privilege(upa, \alpha) \wedge$
  $target(upa, o) \wedge context(upa, c) \wedge Hold(org_b, s, \alpha, o, c)$
  $\rightarrow Is_permitted(s, \alpha, o)$

that is, if an object *upa* is used by a given organization *org* in the view UPA and the issuer of *upa* defines that the context holds between the grantee, the privilege and the target specified by *upa*, then the grantee is permitted to invoke his privilege on the target.

The concrete permissions derived from this rule may be viewed as exceptions to the general permissions defined by the predicate *Permission*. This is exactly the purpose of the UPA component: provide means to specify such exceptions.

8.4.2 Managing the view UPA

The same activities *assign*, *revoke* and *manage* defined in the previous sections are used to express the authorization given to a role to assign, revoke and manage permissions to users.

8.4.3 Example

Let us now show how to use this material. Assume that in $Trusted_bank$, advisers are allowed to consult the company accounts and the counter clerks are allowed to consult the customers' accounts. Also assume that an adviser is allowed to delegate to a given user empowered as a counter clerk the permission to consult the company account of one of his clients. We have first to consider a sub-view $UPA_Consult_Company_Account$ of view UPA defined as follows:

- $\forall upa \in O,$
  $Use(Trusted_bank, upa, UPA_Consult_Company_Account)$
  $\leftarrow Use(Trusted_bank, upa, UPA) \wedge$
  $grantee(upa, s) \wedge Empower(Trusted_bank, r, counter_clerk) \wedge$
  $privilege(upa, \alpha) \wedge Consider(Trusted_bank, \alpha, consulting) \wedge$
  $target(upa, o) \wedge Use(Trusted_bank, o, company_account)$

that is, object *upa* is used in view $UPA_Consult_Company_Account$ if it is used in view UPA and the values of attributes *grantee*, *privilege* and *target* respectively correspond to a user empowered as a counter clerk, an action considered as a consulting activity and an object used as a company account. The permission is then specified as follows:

- $Permission(Trusted_bank,$
  $adviser, assign, UPA_Consult_Company_Account, Default)$

We might want to constrain the use of such a permission. For example, an adviser can grant this permission only if this counter clerk had been designated to work in collaboration with this adviser. To do so, we define a new context $Adviser_Counter_clerk$:

- $\forall s \in S, \forall \alpha \in A, \forall upa \in O,$
  $Hold(Trusted_bank, s, \alpha, upa, Adviser_Counter_clerk)$
  $\leftarrow Use(Trusted_bank, upa, UPA_Consult_Company_Account) \wedge$
  $grantee(upa, s') \wedge Empower(Trusted_bank, s', counter_clerk) \wedge$
  $adviser_clerk(s, s')$

where attribute $adviser_clerk(s_1, s_2)$ states the collaborating pairs of an adviser s_1 and a counter clerk s_2.

This example shows how to give permissions to a set of users. It is also possible to give a permission to a single user. For instance, if an adviser is allowed to give the above permission only to John, one just has to create a sub-view $UPA_John_Consult_Company_Account$ of view $UPA_Consult_Company_Account$:

- $\forall upa \in O,$
  $Use(Trusted_bank, upa, UPA_John_Consult_Company_Account)$
  $\leftarrow Use(Trusted_bank, upa, UPA_Consult_Company_Account) \wedge$
  $grantee(upa, John)$

The same line of reasoning holds for the action and the object.

8.4.4 UPA': granting permissions on activities and views

The UPA component enables to grant permissions on specific actions and objects. In some cases, it is indeed more convenient to grant permissions on organizational entities, namely activities and views. Thereby UPA' is more general. UPA' has the same attributes than UPA but with different meanings:

- *issuer*: represents the organization which is issuing the permission
- *grantee*: designates the subject who is receiving the permission
- *privilege*: represents the activity the grantee is authorized to perform
- *target*: represents the view the grantee is authorized to have an access to
- *context*: designates the context in which the permission applies.

Therefore, in order to allow a role within org_a to grant permission in org_b to subject s to perform activity a on view v within context c, we have to create a new view UPA called $myUPA'$ and defined as follows:

- $\forall org_a \in Org, \forall upa' \in O, \forall s \in S, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$
  $Use(org_a, upa', myUPA')$
  $\leftarrow Use(org_a, upa', UPA') \wedge$
  $issuer(upa', org_b) \wedge$
  $grantee(upa', s) \wedge$
  $privilege(upa', a) \wedge$
  $target(upa', v) \wedge$
  $context(upa', c)$

The following rule describes the relation between an object in a view UPA' and a permission:

- $\forall org_a \in Org, \forall upa' \in O, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$
  $Use(org_a, upa', UPA') \wedge$
  $issuer(upa', org_b) \wedge grantee(upa', s) \wedge$
  $privilege(upa', a) \wedge Consider(org, \alpha, a) \wedge$
  $target(upa') \wedge Use(org, o, v) \wedge$
  $context(upa', c) \wedge Hold(org_b, s, \alpha, o, c)$
  $\rightarrow Is_permitted(s, \alpha, o)$

that is, if an object upa' is used by a given organization org in the view UPA', and the issuer of upa defines that the context holds between the grantee, an action that falls within the activity $privilege(upa', a)$ and an object uses in the view target specified by upa', then the grantee is permitted to perform this action on this object.

8.4.5 Managing the view UPA'

The same activities *assign*, *revoke* and *manage* defined in the previous sections are used to express the authorization given to a role to assign, revoke and manage permissions to users.

8.4.6 Application to delegation

As mentioned before, modelling delegation is a complex problem. The aim of this section is not to fully investigate this problem. We shall simply show that the expressiveness of AdOr-BAC is sufficient to model several of these subtleties. In AdOr-BAC, permission to delegate may be represented by facts having the following forms:

- $Permission(org, role, delegate, view, context)$

meaning that, in organization org, $role$ is permitted to delegate a permission on $view$ in a given $context$. $view$ is a sub-view of UPA or UPA' (depending on if the delegation is applied to specific actions and objects, or activities and views).

Ownership

It is generally assumed that to delegate a permission to a user, the grantor must first hold the permission he wants to delegate. In AdOr-BAC, this constraint is modelled by a context AG (for Authorized Grantor) in the context of UPA and by a context AG' in the context of UPA':

- UPA: $\forall org \in Org, \forall s \in S, \forall d \in A, \forall upa \in O,$
  $Hold(org, s, d, upa, AG)$
  $\leftarrow Use(org, upa, UPA) \wedge$
  $Consider(org, d, delegate) \wedge$
  $privilege(upa, \alpha) \wedge$
  $target(upa, o) \wedge$
  $Is_permitted(s, \alpha, o)$

  that is, the delegator must be granted the permission to perform the action addressed in upa on the object addressed in upa. Notice that action d implements activity *Delegate* and is different from action α addressed in upa.

- UPA': $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall upa' \in O,$
  $Hold(org, s, \alpha, upa', AG')$
  $\leftarrow Use(org, upa', UPA') \wedge$
  $grantee(upa', s) \wedge Empower(org, s, r) \wedge$
  $privilege(upa', a) \wedge target(upa', v) \wedge context(upa', c) \wedge$
  $Permission(org, r, a, v, c)$

  that is, the subject s who delegate must be empowered in a role that has the permission over the activity, the view and the context addressed in the object upa'.

Notice that the definition of context AG might raise a problem with respect to the decidability issue. Assume that AdOr-BAC is applied to a prioritized security policy as defined in chapter 7. The concrete conflict condition $C_{Conflict}$ defined in section 7.4.3 introduces negative predicates $Is_permitted$. As a consequence, if the context AG is applied to an authorization, the policy cannot be stratified since it introduces a cycle. We have to carry out further works on this issue.

Authorization transfer

Another possible restriction is that the grantor will loose the permission he has delegated. In AdOr-BAC, this means that delegation is not an elementary activity but the combination of assigning a permission (as modelled in UPA or UPA$'$) and self-revoking this permission on the grantor (this may be also modelled in UPA or UPA$'$). We do not further develop this analysis of the delegation concept in this paper. We plan to keep on investigating this issue in the future.

Temporary delegation

In some circumstances, we may also specify that the delegation only applies temporarily and will be automatically revoked after a given deadline. In AdOr-BAC, this may be modelled by a temporal context (see section 6.5)

8.5 Other administration functions

In the previous sections, we showed how to model user-role assignment (by the URA view), permission-role assignment (by the PRA view) and user-permission assignment (by the UPA view). We model other administrations in Or-BAC by using the following views:

- *Organization* and *Subject*.

  Inserting or deleting objects in these views enables a subject empowered in an authorized role to respectively create or delete organizations and subjects. In our model, views *Organization* and *Subject* have at least the attribute *name* to respectively represent the organization or subject name and possibly other application specific attributes to model other organization or subject attributes.

- ROA (for Role-Organization assignment), AOA (for Activity-Organization assignment) and VOA (for View-Organization assignment).

  Inserting objects in these views respectively enables a subject empowered in an authorized role to assign roles (resp. activities) (resp. views) to a given organization. These

views have two attributes: *role* (resp. *activity*) (resp. *view*) and *org*. From objects belonging to these views, we can respectively derive instances of relations *Relevant_role*, *Relevant_activity* and *Relevant_view*. This is modelled by the following rule:

$\forall org_a \in Org, \forall roa \in O,$
$Use(org_a, roa, ROA)$
$\leftarrow org(roa, org_b) \wedge role(roa, r) \wedge$
$\quad Relevant_role(org_b, r)$

and the same applies to for relations *Relevant_activity* and *Relevant_view*.

The Or-BAC model also includes the possibility to specify hierarchies of roles, activities, views and contexts. We can manage these hierarchies using the following views:

- RHA (for Role-Hierarchy Administration), AHA (for Activity-Hierarchy Administration), VHA (for View-Hierarchy Administration) and CHA (for Context-Hierarchy Administration).

  View RHA is used to specify that in a given organization, a given role is a sub-role of another role. This view has three attributes: *org*, *sub_role* and *super_role*. The relation between an object in view RHA and the *sub_role* relation is modelled as follows:

  $\forall org_a \in Org, \forall rha \in O, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R},$
  $Use(org_a, rha, RHA) \wedge$
  $org(rha, org_b) \wedge sub_role(rha, r_2) \wedge super_role(rha, r_1)$
  $\rightarrow sub_role(org_b, r_2, r_1)$

  Views AHA, VHA and CHA are similarly defined by respectively replacing role by activity, view and context.

8.6 Conclusion

In this chapter, we presented AdOr-BAC, an administration model for the Or-BAC model. Using AdOr-BAC, the definition of an administration policy is defined in a similar way as the remainder of the security policy specified in Or-BAC. Thus, Or-BAC is a fully auto-administered model. We suggest a logic-based model to express AdOr-BAC.

AdOr-BAC provides a good compromise between fully centralized (and too rigid) administration as in the MAC model, or fully decentralized (but uncontrolled) administration as in the DAC model. When creating a new Or-BAC policy, we suggest starting with a unique user (the creator of the policy) and a unique predefined role *policy-designer* assigned to the creator. The role *policy-designer* has permissions to define roles to administer the organizations and to specify permissions associated with these roles. Thus, using AdOr-BAC, one can specify a decentralized administration. However, it is always possible to control and limit the administration possibilities associated with the different roles created.

We develop three main components for AdOr-BAC called URA for User-Role Assignment, PRA for Permission-Role Assignment and UPA for User-Permission Assignment. The UPA component is useful to control Permission *delegation* when a user wants to grant another user a specific permission. Administration in Or-BAC consists in first creating some relevant views URA, PRA and UPA, next in adding compliant objects in these views. For example,

adding an object in view URA will have the effect of assigning a user in a role. From then on, one just has to grant permissions to add objects in these views in order to give administrative permissions.

Elsewhere, we suggest two variations of the UPA component: UPA that enables a user to delegate a permission to perform a specific action on a specific object and UPA$'$ to delegate a permission to perform an activity on a view. Applying the UPA component to model delegation still requires further work. There are several different characteristics related to delegation such as permanence, monotonicity, totality, levels of delegation or cascading revocation. We have started modelling some of these criteria in the context of the AdOr-BAC model through context definitions. We plan to continue this work, in particular to model how to refine non-elementary activities (such as non monotonic delegation) into elementary ones (such as permission assignment and self-revocation). We have not here taken into account the role hierarchy and the inheritance cascading revocation issues which might appear then. Multi-step delegations also require further investigation.

Chapter 9

Enforcement of the Or-BAC model

In this last chapter, we present some pieces of work carried out in the Or-BAC model framework. So far, we browsed several aspects of Or-BAC: abstraction, hierarchies, contexts, conflict management and administration. However these issues were examined from a rather theoretical standpoint. We now focus on two different applications of the Or-BAC model. In section 9.1 I show how to apply our model in a network environment with the aim of specifying a network security policy using Or-BAC, more exactly we aim at configuring firewalls. Elsewhere, during the thesis, a prototype application, called OToKit, was developed. This program provides a user-friendly interface. It enables us to specify Or-BAC policies and to detect and prevent conflicts. Section 9.2 is dedicated to OToKit.

9.1 Application of Or-BAC in a network environment

9.1.1 Motivation

In the previous chapters we showed that the Or-BAC model provides efficient means to derive the access control specification of an organization. We now illustrate this approach in the context of a network security policy, and present a way to use Or-BAC in order to configure firewalls [Cuppens et al. 2004a]. Two reasons motivate this work.

First, in the general introduction of this dissertation (section 1) we explained that a security policy, in the broadest sense, encompasses all laws, rules, practices, and charters of an organization, and is applied to many application domains such as physical security, operating system access control, database access control, network security, etc. So far, we rather turned the Or-BAC model towards access control. However, our aim with Or-BAC is much wider and ambitious. Indeed, the definition of such a model based on organizations and organizational components, which provides means to express contextual authorizations, to manage conflicts and which offers administration procedures, has for final purpose to express the whole security policy using a single model. More precisely, we aim at defining the top organizational policy using Or-BAC, then refining this policy in order to produce the policies of each application domain until obtaining the final technical policies which will be enforced by the security components. Here, we focus on the network security policy.

Second, one of the problems encountered with firewalls is the difficulty administrators have to configure properly these firewalls. There is a real lack of methodology and corresponding

supporting tools to help them set the network security policy part, and generate and deploy the rules derived from this policy. There are actually no intermediary levels between the policy requirement formulated in the form of an English sentence and its equivalent set of firewall rules, in the form of computable scritps. Moreover, current firewall configuration languages have no well founded semantics. Each firewall implements its own algorithm that parses specific proprietary languages. Even if the firewall administrator is proficient in many configuration languages and tools, this expertise does not avoid from making mistakes. Without a clear methodology and some corresponding supporting tools, this may lead to the generation of configuration rules that are not consistent with the intended network security policy. We claim that the use of a high level language to specify a network security policy will avoid such mistakes and will help to consistently modify the firewall rules when necessary. We also notice that there is not a global security policy specification. The following hypothesis is always assumed: a single security component is used, in other words, a single firewall. Now, it is sometimes more convenient to deploy security rules on several security components. In particular, access security rules can be separated into relevant packages and enforced by more than one firewall on the same LAN.

The remainder of this section is organized as follows. Section 9.1.2 briefly broaches the main contributions related to our topic. Section 9.1.3 introduces the main features of our approach. Sections 9.1.4 to 9.1.7 studies the use of the Or-BAC entities *Organization*, *Subject*, *Role*, *Activity* and $View$ in the context of a network policy. Based on interpretations, we are able in section 9.1.8, to specify a set of network security rules. Finally, in section 9.1.9, we lay the foundations of the automatic derivation of firewall rules from an Or-BAC network policy.

9.1.2 Related work

There are some tools that help administrators to build their security policy and to translate it to the actual configuration language (for instance Cisco PIX [Degu and Bastien 2003], FireHOL [Checkpoint 2004], Ipfilter [Russell 2002], ...) but these tools, for example Firewall builder [Kurland 2003], are bottom-up approaches. In other words, they deal with the particular problem of producing the code in the configuration language of the target firewall. It is quite easy to set filtering rules using the configuration tool included in the offering. However, they do not give a way to think and specify an access control policy before deriving firewall configuration rules that enforce this policy. As a consequence, the security rules can be inconsistent with each other or/and with the global security policy leading to security holes.

There are some works coming under the same topic as ours. Hence, firewall management toolkit Firmato [Bartal et al. 1999] uses an entity-relationship model to specify both the access security policy and the network topology, and uses of the concept of roles to define network capabilities. In this approach there is some mixing between the net topology – a particular concrete level – and the access security policy to be enforced so that the role concept becomes ambiguous. Indeed, the authors are bounded to introduce the "group" concept with an unclear semantics; sometimes group is used to design a set of hosts and sometimes it stands for a role. This can lead to some difficulties to assign network entities to the model entities. In this connection, Firmato's authors use privileges inheritance through

hierarchy of groups to derive automatically permissions. They also make use of tricks to avoid permission leakage. Hence, they introduce notions of "open group" to authorize inheritance of permissions and "closed group" to prohibit it. The reason is the fact that the concept of group is not well defined and we claim that this concept is not needed at the access control policy specification level. Another work of which the motivations are close to ours is the RBNS model [Hassan and Hudec 2003]. Although authors claim that their work is based on the RBAC model [Sandhu et al. 1996], it seems that they keep from this model only the concept of role. Indeed, the specification of network entities and the assignement of roles and permissions are not rigorous and does not fit any reality. In particular, (1) all RBNS relations are binary even though an access control security goal and its equivalent filtering rule are always a triplet $< source, service, target >$. This leads to a loss of information: permissions are missing in RBNS model even though authors consider the assignment of a service to an IP address as a permission, which is semantically weak. (2) Hosts are of two kinds, client or server and roles are assigned to hosts thanks to the pre-declared type of hosts. This is a wrong assignment since, at the abstract level, the role of a given host is service-dependent. (3) The approach makes an excessive use of the concept of role, hence this leads authors to introduce a role-to-role assignment which is a "limping" use of a role based access control model since it means assigning a permission package to another permission package.

9.1.3 Main features of our approach

In this section, we model a local area network as well as its security architecture and its connectivity to the Internet. We choose to re-use the example (figure 9.1) presented in Firmato [Bartal et al. 1999] so as to bring out how Or-BAC provides a natural statement of various entities and concepts used in the security architecture.

Our approach avoids the administrator pondering on access security using filtering rules. The specification of the access control policy is done at a more abstract level and problems like inconsistency are solved before generating the concrete filtering rules.

Organization

In section 3.2, we introduced the definition of entity *Organization* (definition 3.3.8). In accordance with this definition, any component in charge of managing and enforcing a set of security rules can be considered as an organization. Therefore, a concrete security component, such as a firewall, can be viewed as an organization. Afterwards, firewalls are indeed considered as sub-organizations of the institution that manages these firewalls.

Zone

To handle a network security policy, the network topology of the organization local area network has to be captured. Hence, the LAN is parcelled out into *zones*. The network security policy consists in securely managing communications between these zones. We show in the following that view and role definitions, in the Or-BAC model, allow a fine grained specification of zones.

Hierarchy

The hierarchies defined in the Or-BAC model (see section 4.1) can be applied in the context of a network and reveal themselves to be useful in such context. Most noticeably, hierarchies avoid the use of artifices like open and closed groups as suggested in [Bartal et al. 1999]. Actually, we enforce organization, role, activity and view hierarchies.

Permission

In most firewalls, administrators use dual security policy: they specify both positive and negative rules. In this case, the selection of the appropriate rule is based on a first matching or a last matching procedure. In both cases, the decision depends on how the security rules are sorted. Hence, administrators have to find out the correct and efficient rule order, which depends on the filtering procedure. This is a complex task to manage especially when the security policy has to be updated. Moreover, in some cases, it is even not always possible to sort the rules. As a consequence, a closed access control policy that only includes permissions may be an alternative.

We choose indeed to use permissions only and we make the assumption of a closed policy. In practice, we assume a default "deny rule" is stated. This way, we get out of ordering the firewall rules derived to enforce the policy.

Context

This section presents preliminary works on the application of Or-BAC in a network environment. We do not take into account the context for the time being. Thereby, we omit the parameter context in the predicate $Permission$ afterwards. To remain compliant with the Or-BAC model, one should only have to add context $Default$ (see section 6.3) in each permission.

9.1.4 Organizations

We want to model the access control policy of a corporate network used in the organization $Trusted_bank$ which is denoted B in the following. B has a two-firewalls network configuration, as shown in figure 9.1. As presented in [Bartal et al. 1999], the external firewall guards the corporation's Internet connection. Behind is the DMZ, which contains the corporation's externally visible servers. In our case these servers provide HTTP/HTTPS (Web), FTP, SMTP (e-mail) and DNS services. The corporation actually only uses two hosts to provide these services, one for DNS and the other (called Multi_server) for all other services. Behind the DMZ is the internal firewall which guards the corporation's intranet. This firewall actually has three interfaces: one for the DMZ, one for the private network zone, and a separate interface connecting to the firewall administration host. Within the private network zone, there is one distinguished host, *Admin_serv*, which provides the administration for the servers in the DMZ.

In Or-BAC, we can introduce several organizations to model such a configuration. First, there is an organization B and to simplify, we shall actually identify B with its corporate network. B has two sub-organizations denoted B_fw_1 corresponding to the external firewall and B_fw_2 corresponding to the internal firewall. Notice that we could also use an

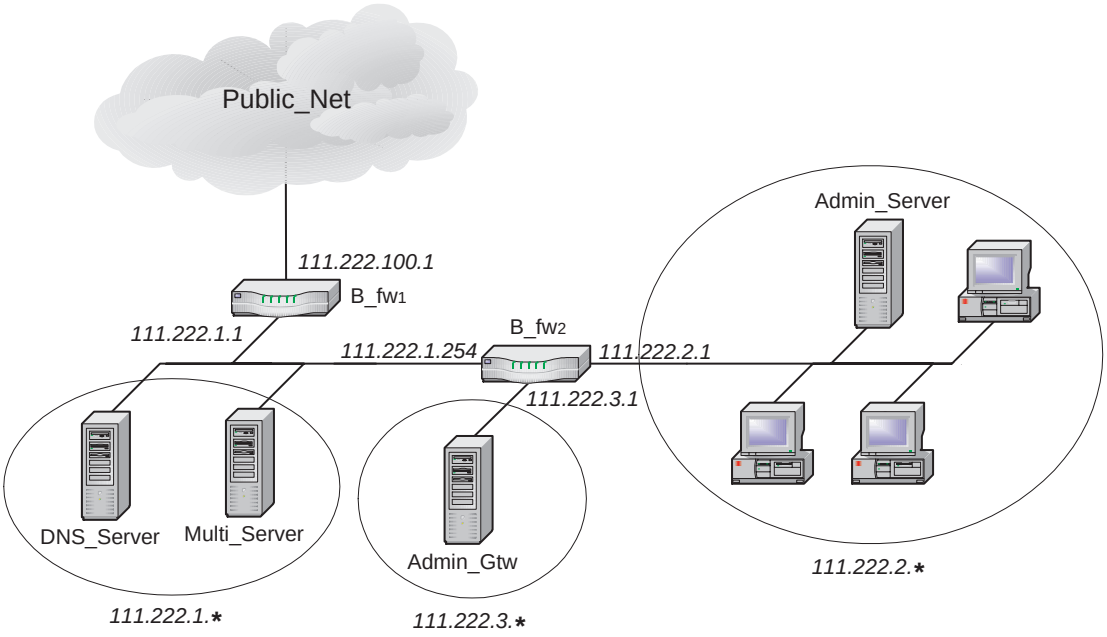


Figure 9.1: Application to a network example

organization called *Internet* if we had to specify an explicit policy to be enforced by the Internet.

9.1.5 Subjects and roles

In this example, subjects correspond to hosts identified by their IP address. We introduce the predicate *address* such as $address(h, ip)$ joins up host h with its IP address ip. Roles are assigned to hosts. For this purpose, predicate *Empower* enables us to assign a role to a given host. However, it would be quite fastidious to assign a given role to every host belonging to a given network. To avoid this, we let the possibility to write the relations concluding on predicate *Empower* in the form of rules:

- $\forall h \in S$,
  $address(h, ip) \wedge partOf(ip, 111.222.3.*)$
  $\rightarrow Empower(B, h, Admin_gtw)$
- $\forall h \in S$,
  $address(h, ip) \wedge partOf(ip, 111.222.3.*) \wedge \neg Empower(B, h, Firewall_interface)$
  $\rightarrow Empower(B, h, Private_net)$

This provides a flexible manner to define some roles. Figure 9.2 specifies those roles that are respectively relevant in organizations B_fw_1 and B_fw_2. Notice that role $Firewall$ is relevant in B_fw_2 (for administration purpose) but not in B_fw_1. For each role, figure 9.2 also presents the sub-roles of this role. In this example, the sub-role hierarchy actually corresponds to a specialization role hierarchy.

Role name	**Hosts assigned to role**	***sub_role***	B_fw_1	B_fw_2
Public_host	Hosts in view *Public_Net*	-	×	
Private_host	Hosts in view *Private_Net*	-		×
Firewall	Firewall interfaces	*External_firewall* *Internal_firewall*		×
External_firewall	External firewall interfaces	-	×	×
Internal_firewall	Internal firewall interfaces	-		×
DNS_server	DNS server	-	×	×
Ftp_server	Ftp server	$Multi_server$	×	×
$Mail_server$	Mail server	$Multi_server$	×	×
Web_server	Web server	$Multi_server$	×	×
$Multi_server$	Multi-server	-	×	×
Adm_fw_host	Hosts in view *Admin_gtw*	-	×	×
Adm_serv_host	Hosts in view *Admin_serv*	-		×

Figure 9.2: Role description

9.1.6 Activities

Activities correspond to various services available in the corporate network B. Activities enable us to join together services to which are applied some common authorizations. We define a first activity *all_tcp* with different tcp activities (such as *smtp*, *ssh* and *https*) as sub-activities. Similarly, we define an activity *all_icmp* with different icmp activities (such as *ping*) as sub-activities. We also define two other activities. *admin_to_gtwy* has to sub-activities: *ssh* and *ping*. *gtwy_to_admin* has also two sub-activities: *ssh* and *https*. All these activities are relevant in organizations B, B_fw_1 and B_fw_2.

9.1.7 Views

Views are used to structure objects on which network services apply. We define a view called *target* having two attributes: $content(o, ctt)$ designates the content *cct* of an object o belonging to the view *target*, and $dest(o, d)$ that corresponds to the destination host d. The destination host is identified by its role. Actually, the *content* attribute is not used in the example because we shall only consider filtering rules on the destination host. However, it would be useful to filter messages depending on their content.

We can then define sub-views of view *target* according to the role assigned to the destination host. For instance, we can define sub-view *to_dns* as follows:

- $\forall o \in O$,
  $Use(B, o, to_dns)$
  $\leftarrow Use(B, o, target) \wedge dest(o, DNS_server)$

This would lead to define as many views as there are roles. This would be quite fastidious. Instead, we suggest defining this kind of views as compound atoms having the form $to_target(r)$. Views created with *to_target* are defined as follows:

- $\forall o \in O, \forall r \in \mathcal{R}$,
  $Use(B, o, to_target(r))$
  $\leftarrow Use(B, o, target) \wedge dest(o, r)$

We consider that every view defined as $to_target(r)$ is relevant in one of the organization of our example if r is a role relevant in this organization. We also consider that if role r_1 is a sub-role of role r_2, then the view $to_target(r_1)$ is a sub-view of view $to_target(r_2)$.

9.1.8 Security policy

We can now specify several permissions that apply to organization B. These permissions correspond to the security policy presented in [Bartal et al. 1999]. It is a rather simple policy, which nonetheless covers many of the aspects which occur in more complex, real-life policies. Its premise is that internal corporate users are basically trusted and thereby are relatively unrestricted, whereas external users are only allowed to access information that is explicitly specified as public. In more detail, the policy has the following goals:

1. Internal corporate hosts can access all the resources on the Internet.
2. External hosts can only access the servers in the *DMZ*.
3. The *DMZ servers* can be updated only by the web administrator host *admin server*. Other corporate hosts have the same privileges as Internet hosts with respect to the *DMZ servers*.
4. The firewall gateway interfaces are only accessible from the *fw admin* host and are otherwise inaccessible to any host (this practice is usually called "stealthing" the gateways).

Figure 9.3 lists how these permissions are modelled in Or-BAC. One significant advantage of our approach is that it enables to automatically derive permissions that respectively apply to B_fw_1 and B_fw_2. This is done by using rule OH_1 (see section 4.1.4) for deriving permissions in sub-organizations of B. The results we obtain for B_fw_1 are presented in figure 9.4. To illustrate this derivation process let us consider the following permission:

- $Permission(B, adm_fw_host, admin_to_gtwy, to_target(firewall))$

Since role adm_fw_host, activity $admin_to_gtwy$ and view $to_target(firewall)$ are relevant in B_fw_2, we can apply rule OH_1 to derive:

- $Permission(B_fw_2, adm_fw_host, admin_to_gtwy, to_target(firewall))$

However, since view $to_target(firewall)$ is not relevant in B_fw_1, we cannot derive a similar permission for B_fw_1. But, view $to_target(external_firewall)$ is a sub-view of $to_target(firewall)$. Since $to_target(external_firewall)$ is a relevant view in B_fw_1, we can apply rules VH_1 and OH_1 to derive:

- $Permission(B_fw_1, adm_fw_host, admin_to_gtwy, to_target(external_firewall))$

Notice that one permission, namely:

$Permission(B, adm_fw_host, admin_to_gtwy, to_target(firewall))$
$Permission(B, firewall, gtwy_to_admin, to_target(adm_fw_host))$
$Permission(B, private_host, all_tcp, to_target(public_host))$
$Permission(B, adm_server_host, all_tcp, to_target(dns_server))$
$Permission(B, adm_server_host, all_tcp, to_target(multi_server))$
$Permission(B, public_host, smtp, to_target(mail_server))$
$Permission(B, public_host, dns, to_target(dns_server))$
$Permission(B, public_host, ftp, to_target(ftp_server))$
$Permission(B, public_host, https, to_target(web_server))$
$Permission(B, private_host, smtp, to_target(mail_server))$
$Permission(B, private_host, dns, to_target(dns_server))$
$Permission(B, private_host, ftp, to_target(ftp_server))$
$Permission(B, private_host, https, to_target(web_server))$
$Permission(B, dns_server, dns, to_target(public_host))$
$Permission(B, ftp_server, ftp, to_target(public_host))$
$Permission(B, dns_server, dns, to_target(private_host))$
$Permission(B, ftp_server, ftp, to_target(private_host))$

Figure 9.3: Permissions in organization B

$Permission(B_fw_1, adm_fw_host, admin_to_gtwy, to_target(external_firewall))$
$Permission(B_fw_1, external_firewall, gtwy_to_admin, to_target(adm_fw_host))$
$Permission(B_fw_1, public_host, smtp, to_target(mail_server))$
$Permission(B_fw_1, public_host, dns, to_target(dns_server))$
$Permission(B_fw_1, public_host, ftp, to_target(ftp_server))$
$Permission(B_fw_1, public_host, https, to_target(web_server))$
$Permission(B_fw_1, dns_server, dns, to_target(public_host))$
$Permission(B_fw_1, ftp_server, ftp, to_target(public_host))$

Figure 9.4: Permissions in organization B_fw_1

- $Permission(B, private_host, all_tcp, to_target(public_host))$

is not inherited by B_fw_1 nor B_fw_2. This is because role *private_host* is only relevant to B_fw_2 whereas view $target(public_host)$ is only relevant to B_fw_1. So, no firewall alone can manage this permission. In this case, our proposal is to use this permission to configure both firewalls.

9.1.9 Derivation of concrete firewall rules

The application of the Or-BAC model to the configuration of firewalls has been further investigated in [Cuppens et al. 2004b]. In this section, we just give the main features of this work. We suggest an XML syntax to specify an organizational level network security policy based on Or-BAC and independent of the implementation of this policy in a given firewall. All entities, concrete and organizational, are expressed using XML. These entities are used all top-down specification long to properly generate firewall configuration rules.

This derivation process consists of two steps, as shown in figure 9.6. In the first step, we generate, from the Or-BAC policy and using an XSL transformation, rules expressed

$Permission(B_fw_2, adm_fw_host, admin_to_gtwy, to_target(firewall))$
$Permission(B_fw_2, firewall, gtwy_to_admin, to_target(adm_fw_host))$
$Permission(B_fw_2, private_host, all_tcp, to_target(public_host))$
$Permission(B_fw_2, adm_server_host, all_tcp, to_target(dns_server))$
$Permission(B_fw_2, adm_server_host, all_tcp, to_target(multi_server))$
$Permission(B_fw_2, private_host, smtp, to_target(mail_server))$
$Permission(B_fw_2, private_host, dns, to_target(dns_server))$
$Permission(B_fw_2, private_host, ftp, to_target(ftp_server))$
$Permission(B_fw_2, private_host, https, to_target(web_server))$
$Permission(B_fw_2, dns_server, dns, to_target(private_host))$
$Permission(B_fw_2, ftp_server, ftp, to_target(private_host))$

Figure 9.5: Permissions in organization B_fw_2

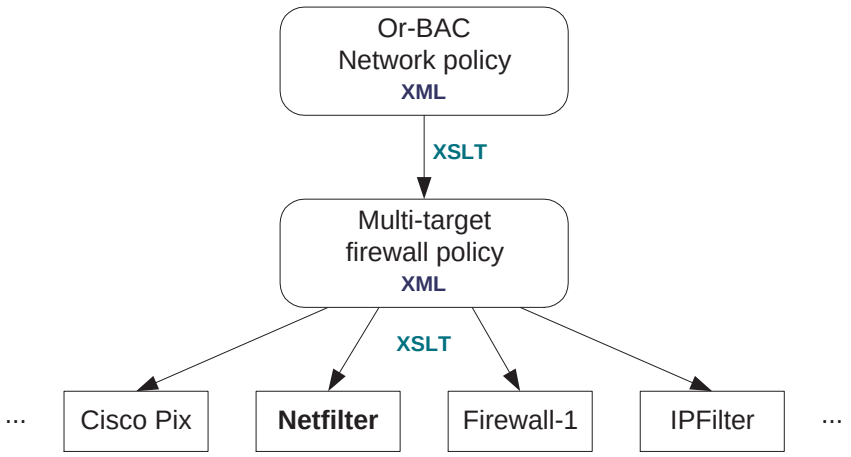


Figure 9.6: Derivation of concrete firewall rules

in an intermediary multi-target firewall language which also uses an XML syntax. In the second step, we derive, from this intermediary language and using an XSL transformation, concrete configuration rules expressed in the native target firewall language. In the next two sub-sections, we give details about these two-steps derivation process.

From organizational policy to generic firewall rules

The aim of this first XSL transformation is to derive generic firewall rules from the organizational network security policies expressed with the Or-BAC formalism. This process uses hierarchy mechanisms to derive concrete rules relevant for each firewall involved in the security architecture. For this purpose, once the network security policy of a given organization is specified (organization B in our example), we merely have to specify which entities (roles, activities and views) are relevant in the sub-organizations (firewalls B_fw_1 and B_fw_2 in our example). The derivation process is then able to automatically distribute every permission that each firewall has to manage. To generate the security rules in the multi-target firewall language, the derivation process parses organizational security rules specified in the Or-BAC model. Notice that the organizational policy is not order-sensitive: rules can be written in any order.

From generic rules to specific firewall rules

In order to validate our approach, we have chosen to derive concrete rules for NetFilter [Russell 2002]. So we have designed an XSL transformation which enables to derive NetFilter rules from the multi-target firewall language. It is noticeable that, as for the organizational security rules, the order of the generated rules does not matter (with the exception of the "deny rule" corresponding to a closed policy). Thanks to the chain mechanism of NetFilter, we can derive rules without ordering exclusion conditions. But for firewall languages that do not provide such mechanism, exclusion conditions will correspond to deny rules. These deny rules have to be interleaved with "accept rules" to obtain the same result as with NetFilter chains. Ordering the rules has to be done by the XSL transformation process.

We plan to create some new XSL transformations associated to other firewall configuration languages for the purpose of dealing with heterogeneous networks that integrate at the same time different firewalls, like Cisco PIX, Check-Point Firewall-1, etc.

For further details about the Or-BAC expression using XML and the derivation process from an organizational policy to specific firewall rules, see [Cuppens et al. 2004b].

9.1.10 Conclusion

We presented the application of the Or-BAC model in a network environment. We showed that Or-BAC provides a convenient layer of abstraction which is revealed to be suitable for network policies. In order to fit our model to a network environment we suggested to extend the interpretation of the Or-BAC entities: firewalls are considered as organizations, hosts as subjects and services as activities. Furthermore, hosts can play some roles (like DNS server), and targets are captured using views. Therefore, using Or-BAC to model network policies allows the SSO to make a clear separation between network entities and organizational model entities like roles, services, groups of hosts having the same role, hosts concerned by the source host query, and so on.

Or-BAC can be used to (1) specify a network security policy which is not topology-dependent, (2) specify a network security policy which is independent of a particular firewall product.

Moreover we showned how to use inheritance to distribute the network policy specification over several security components. We illustrated this approach by an example of security architecture based on two firewalls. We also investigated the automatic generation of the target firewall rules from the formal specification of a network security policy.

The application of Or-BAC in a network environment enabled us to extend the interpretation of the Or-BAC concepts. These new interpretations are integrated in Or-BAC and therefore extend the scope of our model. We actually proved that Or-BAC is perfectly suitable to design network policies.

9.2 OToKit: Or-BAC ToolKit

9.2.1 Motivation

This section is dedicated to OToKit (Or-BAC ToolKit), an application prototype designed to manage Or-BAC policies. The working out of such a prototype aims, by enforcing concrete examples, at showing the Or-BAC model. We present the results and the advantages of the various derivation rules related to the hierarchies, the contexts, the conflict management. Furthermore, OToKit enables people who do not possess knowledge in logic programming to exploit all Or-BAC facilities. Finally, OToKit permits us to validate our approach.

The next section describes the features I wanted to implement in OToKit. Then, section 9.2.3 presents the technology used, and section 9.2.4 some programming aspects. Section 9.2.5 shows and details the graphical interface. In section 9.2.6 we come back on the points of the following list to see what I have achieved so far and what does actually work. In section 9.2.7, we discuss the performance of OToKit. Finally, we shall see in section 9.2.8 the future works we plan to realize with OToKit.

9.2.2 Objectives

1. Design a user-friendly interface. The Or-BAC model is based on a logic language. Therefore a simple request interface is enough to design Or-BAC policies. However, a graphical user interface (GUI) is much more convenient to consult a complex policy composed of numerous entities.

2. Capture the organization structure. OToKit must enable to manage organizations, roles, activities, views and contexts (sections 3.2, 3.3 and 3.4), and to specify hierarchies of such entities (section 4.1).

3. Specify authorizations. OToKit must provide means to specify the policy authorizations, that is, positive and negative organizational authorizations as defined in section 3.6.

4. Implement inheritance mechanisms. Since OToKit has to allow to define hierarchies, It must be possible to associate appropriate inheritance mechanisms to these hierarchies, as it is described in section 4.1.

5. Specify contexts. OToKit shall provide means to define contexts, that is, to specify the logic rules that conclude on contexts (chapter 6).

6. Specify constraints. All along this report we introduced some basic constraints that must be implemented in order to ensure that written policies are consistent. These basic constraints (section 4.2) can be hard-coded since they are essential. In contrast, other constraints, such as separation constraints, are specified by the SSO. As a consequence, OToKit should enable us to capture user-defined constraints.

7. Capture conflict management strategies. In chapter 7, we designed a solution to define parametric strategies, which are based on sets of priority levels and sets of rules to take decisions when conflicts occur. OToKit should capture such strategies, and specify for each authorization the corresponding priority level.

8. Detect potential conflicts. OToKit must detect potential conflicts between organizational authorizations as described in section 7.5.

9. Concrete policy simulation. The Or-BAC model is designed with a view to separate the organizational policy from the implementation issues. However, in order to simulate concrete policies, it would be convenient to define users, actions and objects. Simulating a concrete policy consists in deriving the concrete authorizations and detecting conflicts between these authorizations, as described in section 7.4.3.

10. Administration. Since an administrative model is associated with Or-BAC (chapter 8), OToKit should enable to capture administrative authorizations and implementing these authorizations in order to enforce the administrative procedures.

9.2.3 Implementation choices

The implementation of OToKit relies on three different technologies: SWI-Prolog for Or-BAC policies, Java to design the graphical interface and the API JPL to make the Prolog program and the interface communicate.

SWI-Prolog[1]. The Or-BAC model is based on logical facts and rules, therefore, Prolog enables us to directly implement our model. Moreover, Prolog is more flexible than Datalog. For example, as discussed in section 6.5, were are not able to evaluate temporal contexts with Datalog for the time being, whereas Prolog makes this possible. However, since Or-BAC is compliant with Datalog, we have a guarantee that Or-BAC policies remain decidable. I chose SWI-Prolog in particular, rather than GNUProlog for instance, because SWI-Prolog provides some interesting features such as module management (for clearer and more flexible programming), threads (for the power of parallel processes), interface ODBC (for future developments), availability for both Windows and Linux systems. All facts corresponding to a policy and all derivation rules are coded using SWI-Prolog. I worked with SWI-Prolog v5.0.0.

Java[2]. Java programming is a good choice to design nice and convenient GUIs. The *Swing* API provides solutions to quickly develop strong programs. It is claimed that the computation of a java program is slow. However, it appears sufficient for a prototype. Moreover, Java programs are portable. This offers an indisputable advantage. The interface was programmed with Java2 SDK build 1.4.2_05-b04.

API JPL[3]. The Java API JPL (Java-Prolog) provides a set of classes to launch easily Prolog requests from a Java program, and collect results in a suitable and convenient format.

OToKit is designed using the Eclipse[4] Project development software version 2.1.3 for the Java programming part, and SWI-Prolog-Editor[5] version 2.12g for the Prolog programming part.

9.2.4 Programming aspects

Figure 9.7 shows the main components of the OToKit program. One might notice that the communication module consists of two elements: the JPL part and a "Rules" part. It is

[1] http://www.swi.prolog.org
[2] http://www.java.sun.com
[3] http://sourceforge.net/projects/jpl/
[4] http://www.eclipse.org/
[5] http://www.bildung.hessen.de/abereich/inform/skii/material/swing/index.htm

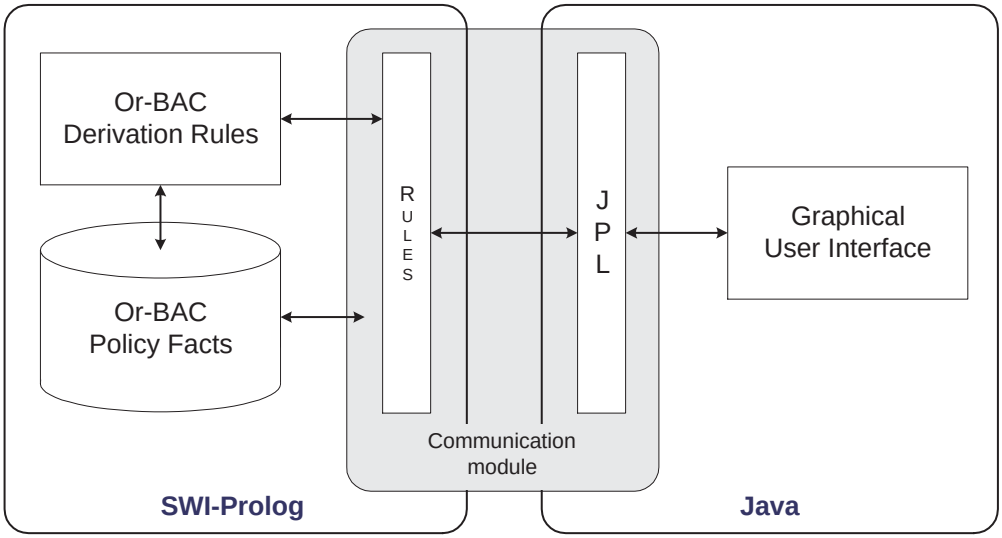


Figure 9.7: The main components of the OToKit program

```
%%% RG'1: Concrete positive authorization derivation in TPpol
%% Permission' --> Is_permitted'
is_permittedL(Subject, Action, Object, Level) :-
    permissionL(Org, Role, Activity, View, Context, Level),
    empower(Org, Subject, Role),
    consider(Org, Action, Activity),
    use(Org, Object, View),
    hold(Org, Subject, Action, Object, Context).
```

Figure 9.8: Example of Prolog rule

indeed necessary to provides some Prolog rules only dedicated to the communication with the GUI. Afterwards we present some programming aspects for each of OToKit's components.

SWI-Prolog

All facts and rules described in this report are easily translated in Prolog. Contrary to Datalog, Prolog queries are evaluated in *backward-chaining* (or *backtracking*), that is, from rule conclusions to rule premises. Thereby, several points must be carefully considered. In particular the transitive closure must always be ensured. Figure 9.8 shows how the derivation rule RG'$_1$ (section A.6) is coded.

JPL

Figure 9.9 presents a method example which enables to construct a Prolog request. A Prolog request is launched with the method $unify$ and returns a hashtable array. The request predicate is captured with a $String$. The parameters are stored in an array of JPL class $Term$. Free variables are captured with the JPL class $PrologVariable$[6]. Instantiated

[6]The original class is $Variable$. I extended it in order to indicate the variable type, namely *Org*, *Role*, etc.

```
/**
 * Get all the prioritized authorizations within organization <b>org</b>.<br>
 * Get positive or negative authorizations according to <i>authorizationType<i>
 *
 * @param authorizationType equals to <i>permission</i>, <i>prohibition</i> (mandatory)
 * @param org               an organization
 * @return a list of prioritized authorizations
 */
public Hashtable[] getAuthorizationWithLevel (String authorizationType, String org)
{
    // init local variables
    String predicate;

    // set predicate
    if (authorizationType.equals("permission"))
        predicate = "permissionL";
    else
        predicate = "prohibitionL";

    // set the variable array
    Term arg[] = {  new Atom(org),
                    new PrologVariable("Role"),
                    new PrologVariable("Activity"),
                    new PrologVariable("View"),
                    new PrologVariable("Context"),
                    new PrologVariable("Level") };

    return prolog.unify(predicate,arg);
}
```

Figure 9.9: Java method with JPL objects

variables are captured with the JPL class *Atom*. The method example of figure 9.9 is used to get all authorizations (positive or negative) in a given organization.

Java

With regard to the Java programming part, we just present part of the class structure. Figure 9.10 shows the hierarchies of the Or-BAC entities and authorizations. The class *Entity* is the parent class of all the other class entities. It is actually an abstract class. Two classes inherit from *Entity*: *OrganizationalEntity* and *ConcreteEntity*. Notice that class *RAVEntity* encompasses all variables and methods that the entities *Role*, *Activity* and *View* have in common. These methods are for example *add*, *delete*, *modify*, *addSubEntity*, etc. The other hierarchy is related to the organizational authorizations. The class *AuthorizationL* corresponds to prioritized organizational authorizations. Elsewhere, since concrete authorizations are dynamically derived, there is no need to create a dedicated class.

9.2.5 OToKit graphical interface

Figure 9.11 presents an overview of the graphical interface. The numbers that appear in this figure correspond to the different GUI components. There are described in the list below:

1. Pull-down menu which enables to select an organization.
2. Entity tabbed pane. Each tab corresponds to an entity type, except the first one which displays all the organizational entities in the form of a tree. The tabs corresponding to

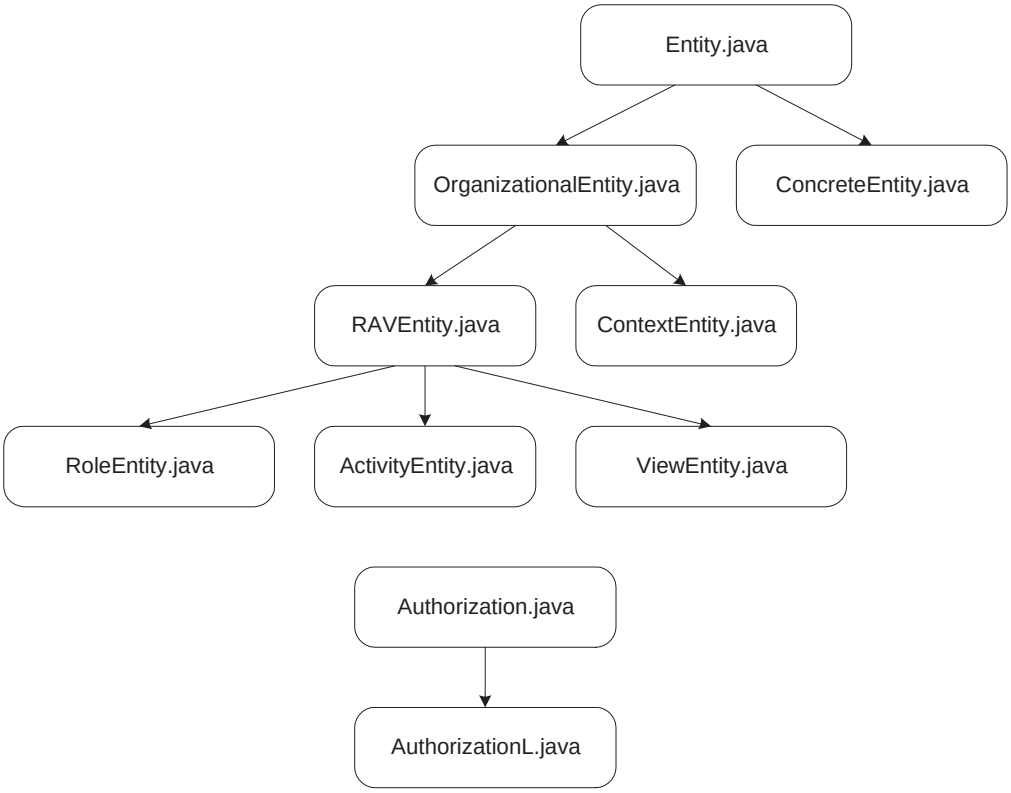


Figure 9.10: Class hierarchies in OToKit

organizational entities also contain trees in order to show the hierarchies. By contrast, the tabs corresponding to concrete entities contain lists.

3. Non-editable text panel which displays several kinds of information corresponding to the entity selected in 2.

4. This table shows all organizational authorizations (positive and negative) associated to the entity selected in 2.

5. This pane also contains a table. It displays the conflicting authorization couples. It can actually display conflicts between organizational authorizations (figure 9.17) or concrete authorizations (figure 9.19) in accordance with the mode of conflict detection selected in 8.

6. By pressing this button, the user launches the conflict detection.

7. This button displays a dialog box which enables the user to test if a given user is granted permission to perform a given action on a given object (figure 9.18).

8. Conflict detection mode. The user can choose between organizational or concrete detection.

9.2.6 Achieved work

We browsed the main goals to achieve with the development of OToKit in section 9.2.2. We shall go over these objectives point by point and describe what I have implemented so far and point out the main difficulties.

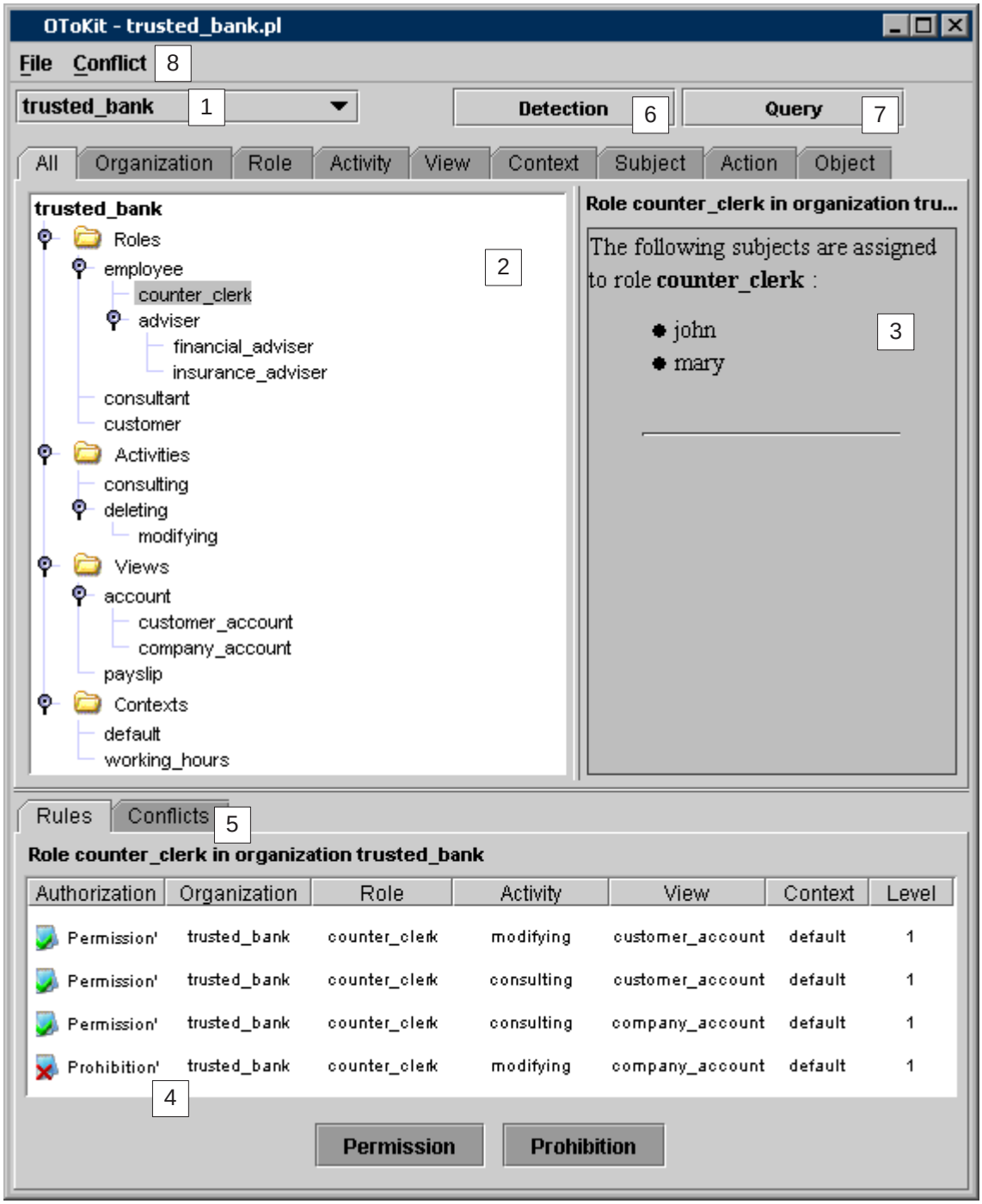


Figure 9.11: OToKit graphical interface

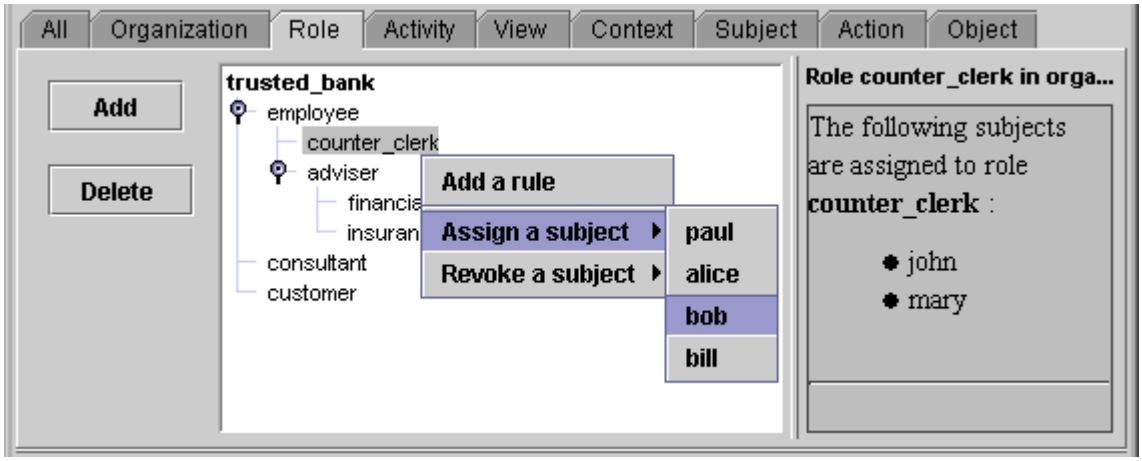


Figure 9.12: Role-user assignment

1. Designing a user-friendly interface

The OToKit graphical user interface is shown in figure 9.11 and all along the figures 9.12 to 9.17. It is convenient to audit and create Or-BAC security policies. Even if, with use, it appears that some minor improvements have to be made (like the way some functionalities are accessed), it is suitable for its purpose.

2. Specifying the structure of organizations

OToKit provides means to create and manage all the organizational entities of the Or-BAC model, namely the organizations, roles, activities, views and contexts. Furthermore, we can create hierarchies of each of these entities, with the exception of the context hierarchy since this is work in progress. I choose to use the Java class $JTree$ to model these hierarchies. Although convenient for simple hierarchies, and for a first version of the prototype, it rather limits the expression of hierarchies since it does not permit to model multiple inheritance hierarchies. For example, in figure 2.7, the role $chief_adviser$ inherits from the roles $financial_adviser$ and $insurance_adviser$. One can note in figure 9.11 that it can not be expressed using simple trees. Another graphical representation has to be found.

OToKit provides a simple way to assign users to roles as presented in figure 9.12. The same method is available to assign actions into activities and objects into views.

3. Specifying authorizations

Once enough elements are specified, it is rather simple to grant authorizations. One just has to choose the corresponding functionality in the contextual menu of any element. Once this is done, the dialog box of figure 9.13 appears. Each authorization parameter is selected in a pull-down menu which is dynamically filled with the elements stored in the policy. Authorizations are then displayed in the component 4 of the GUI. Note that authorizations explicitly stated by the policy designer appear in black font, whereas derived authorizations resulting from inheritance appear in gray font (figure 9.14).

Only prioritized authorizations can be keyed in at present. I aim at providing the possibility to let the policy designer choose whether to create a prioritized policy or not.

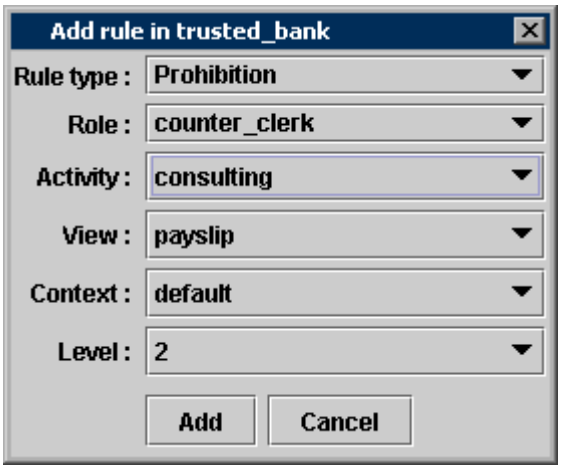


Figure 9.13: Authorization expression

4. Inheritance mechanisms

All inheritance mechanisms specified in section 4.1 are implemented in OToKit, with again the exception of the context hierarchies. Let us consider the role, activity and view hierarchies which can be observed in figure 9.11. This example implements role and view hierarchies. The first panel of figure 9.14 shows the authorizations granted to role *employee*. Actually, only one authorization is explicitly stated. The two others result from the view hierarchy since the view *account* has two sub-views *customer_account* and *company_account*. The second and the third panels display respectively the authorizations granted to the role *counter_clerk* and the role *adviser*. These roles inherit authorizations of the role *employee*. An explicit negative authorization is added to each of these roles.

5. Specifying contexts

At the moment, since OToKit aims at validating our approach, we only provide means to specify temporal contexts. Future works must be led to identify the recurrent contexts and to provide appropriate graphical tools. Furthermore, the solutions suggested in section 6.3.2 to specify expressive contexts using compositions, as well as the context taxonomy described in section 6.4, will be useful to develop such tools. For example, complex contexts can be specified as a conjunction of more elementary contexts. Figure 9.15 shows the definition of context *working_hours*.

6. Specifying constraints

The basic constraints of the Or-BAC model are coded in built-in predicates in the Prolog derivation rule part. The other constraints have to be specified by the policy designer. For the moment, the OToKit graphical interface makes it possible to add some separation constraints. In figure 9.16, one can notice that the activities *consulting* and *modifying* are separated. The same figure shows how to add a new separation constraint, a view separation constraint to be specific. In practice, since a separation constraint exists between the activities *consulting* and *modifying*, if the policy designer tries to assign a given action to both activities *consulting* and *modifying*, OToKit will stop him and show an alert.

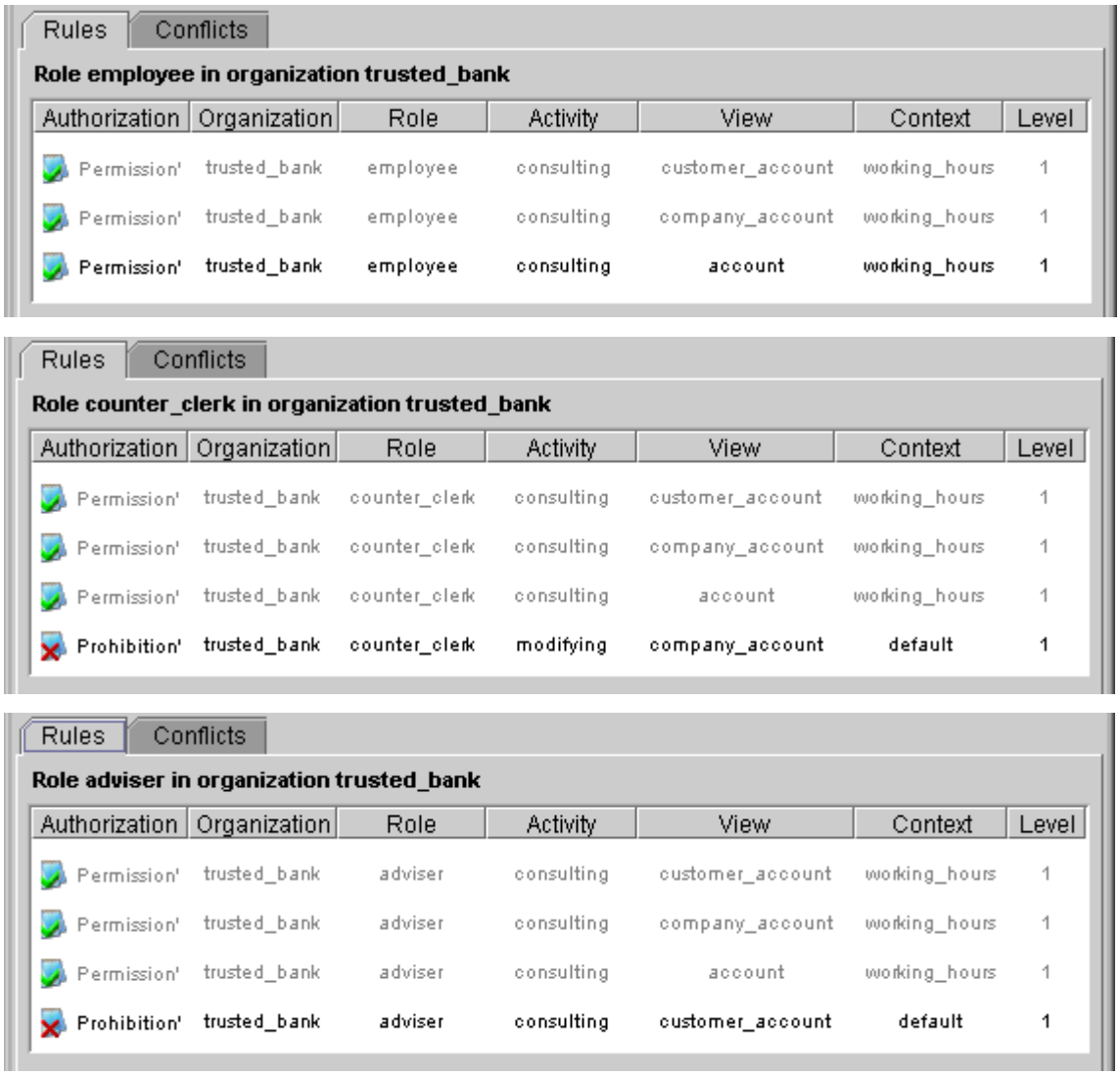


Figure 9.14: Example of authorization inheritance

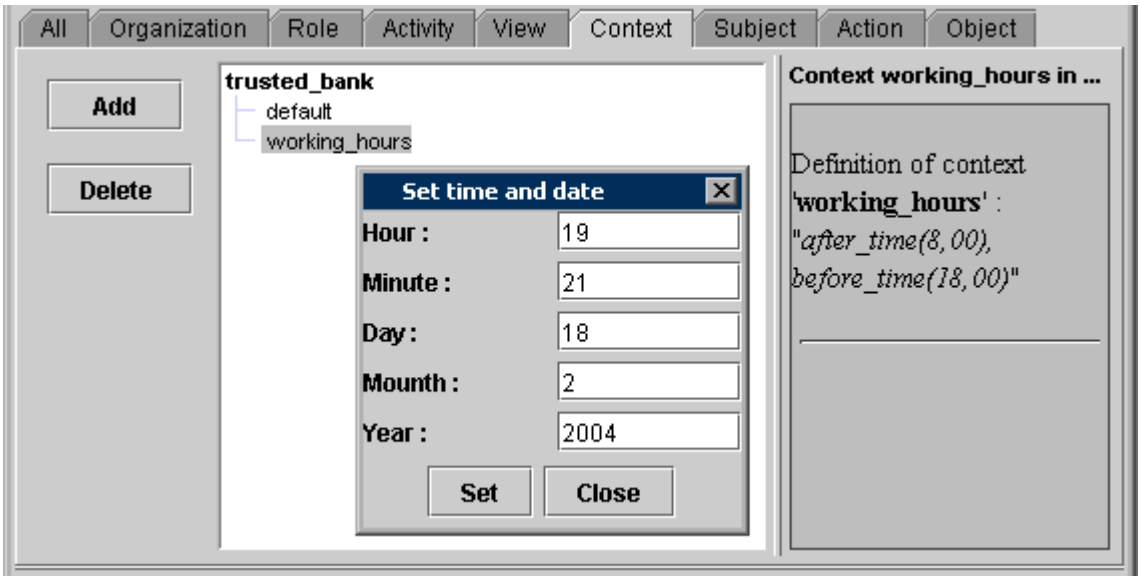


Figure 9.15: Context expression

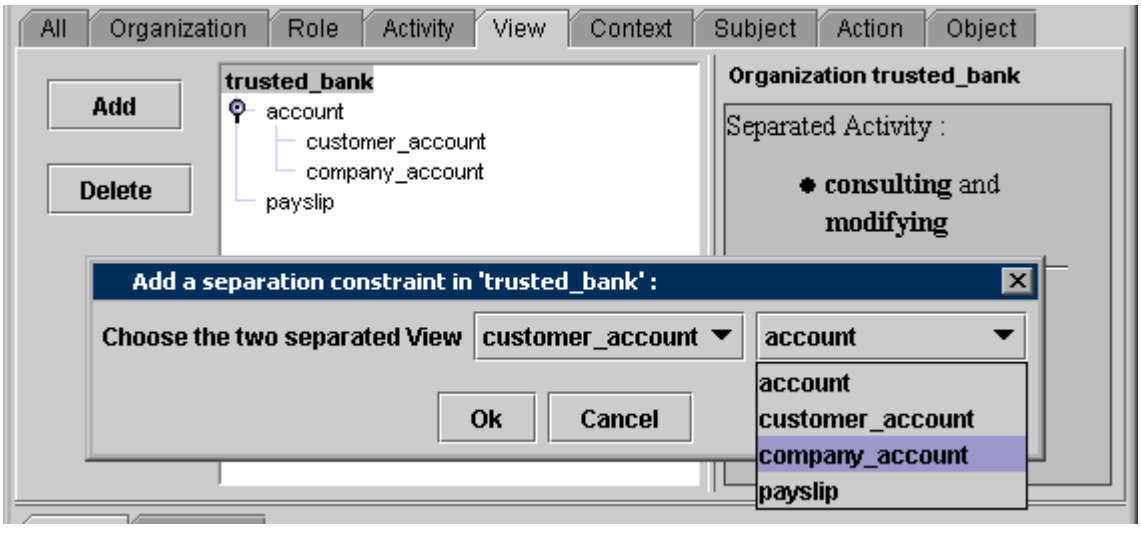


Figure 9.16: Separation constraint

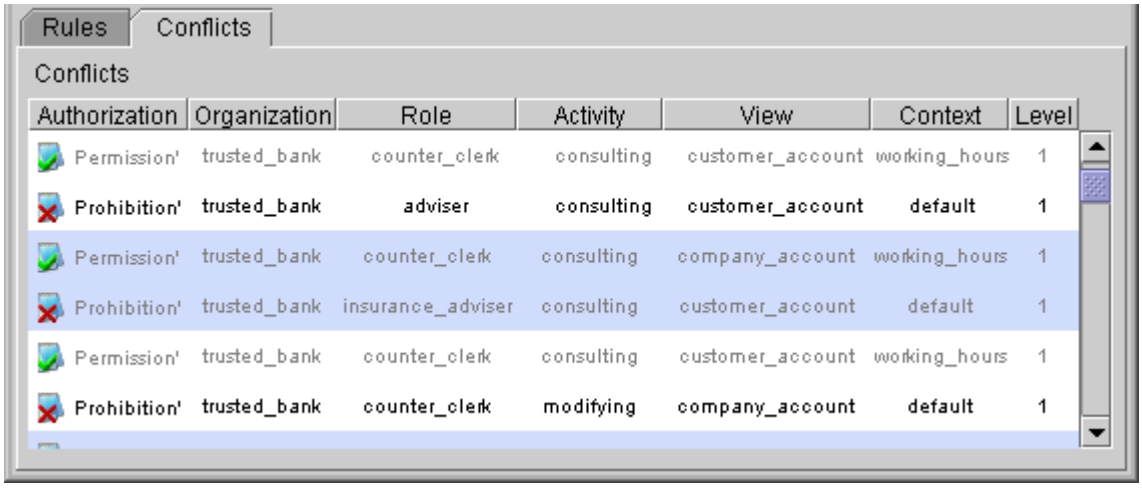


Figure 9.17: Conflict prevention

Furthermore, OToKit will not accept to add a separation constraint between two activities, for example, if an action is yet assigned in these activities.

7. Specifying conflict management strategies

The development of graphical tools dedicated to the specification of conflict management strategies had not been studied in depth due to time limitation. For the time being, a strategy is fixed. The set of priority levels is defined as a fixed set of intergers. The authorization with the higher priority level takes precedence.

8. Detecting potential conflicts

OToKit implements the conflict prevention condition $C'_{Conflict3}$ exposed in section 7.5.3. The first panel of figure 9.17 shows part of the potential conflicts detected in the policy described above in figure 9.14. The first pair of conflicting authorizations corresponds to a positive authorization granted to role *counter_clerk* and an equivalent negative authorization assigned to role *adviser*. A potential conflict is raised since if a user is empowered in both roles, and if at least one object is used in the view *customer_account* and one action in the activity *consulting*; then a concrete conflict will occur when this user wants to use this action on this object. Note that, at the organizational level, contexts are considered as simple labels. Therefore, without an explicit separation constraint between two contexts, conflicts are detected between permissions and prohibitions that use these contexts. The SSO has several possibilities to resolve this potential conflict:

- delete one of these conflicting authorizations;
- increase one of the authorizations priority level;
- add a separation constraint between the roles *customer_account* and *adviser*.

The third conflicting pair of our example corresponds to a situation where an object would be used in both views *customer_account* and *company_account*, and an action would be considered both as activities *consulting* and $modifying$.

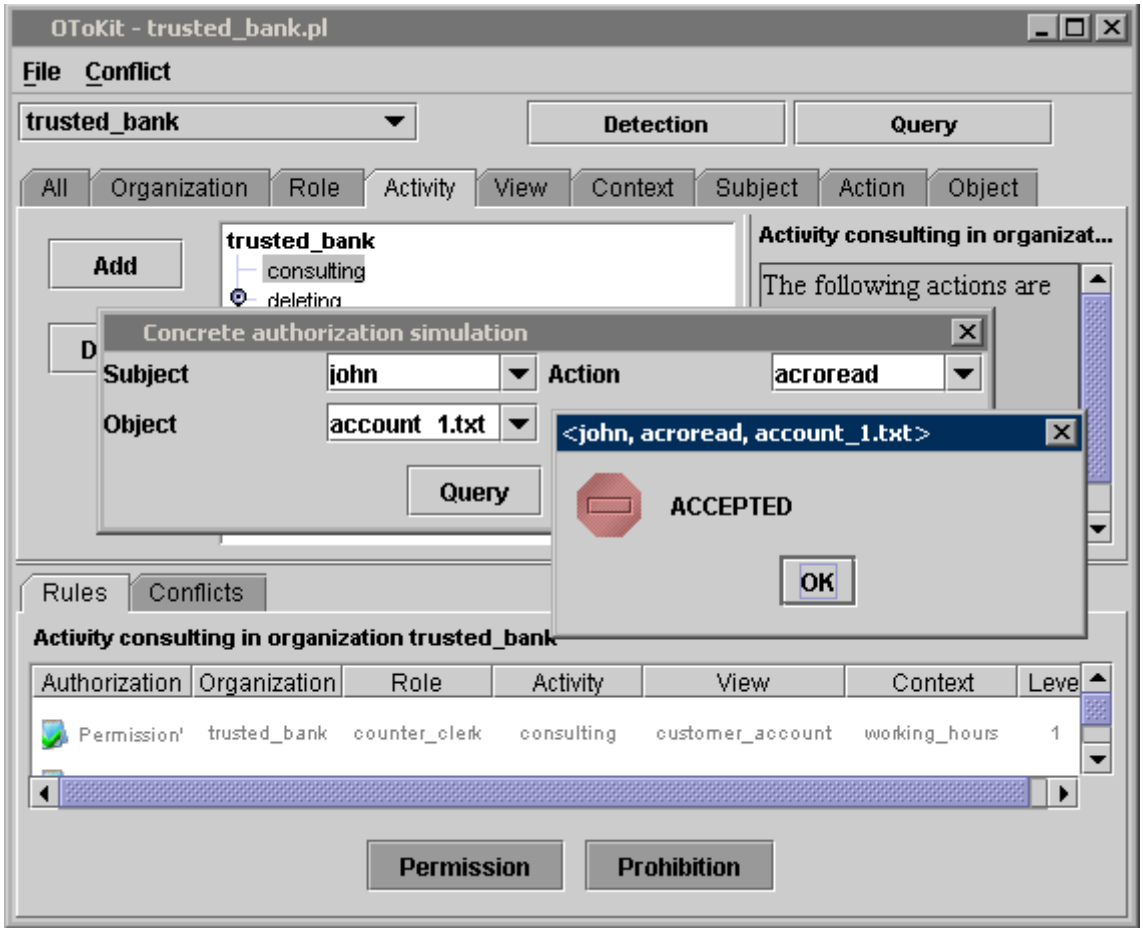


Figure 9.18: Authorization request simulation

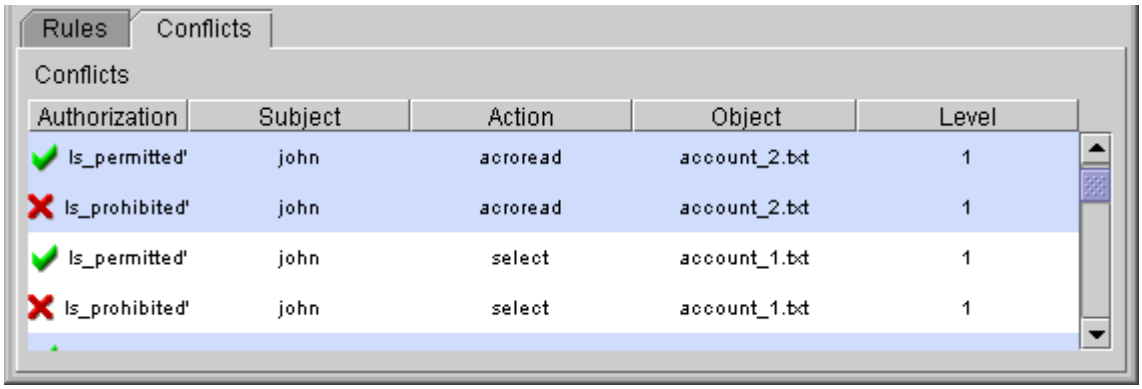


Figure 9.19: Actual conflict simulation

9. Simulation of concrete policies

The main purpose of OToKit is to design organizational policies, in order to verify that conflicts will not happen, and to administrate these policies. Then, these policies are enforced in the information system, like in OSs, databases, security components, etc. However, I integrated several possibilities to simulate an Or-BAC policy. First, subjects – resp. actions, objects – can be added and assigned to roles – resp. activities, views.

Second, OToKit enables us to simulate an access request. This is done using the button *Query*. Figure 9.18 shows the dialog box which is displayed. Thanks to this box, the policy designer can select a subject, an action and an object and checks if the corresponding access is accepted. This is done by applying the rules ED_1 and ED_2 defined in section 7.3.3.

Third, OToKit makes it possible to detect conflicts between concrete authorizations, by verifying conditions $C_{Conflict}$ exposed in section 7.4.3. Actual conflicts example is presented in figure 9.19. Concrete authorizations depend on contexts. Only temporal contexts are implemented in OToKit for the time being. Therefore, concrete conflicts depend on the time at which a request is made. OToKit enables to set a simulation time, as shown in figure 9.15.

Consider the following example. Assume we have the following positive organizational authorization:

- $Permission'(trusted_bank,$
  $counter_clerk, consulting, customer_account, working_hours, 1)$

This authorization is displayed in the second panel of figure 9.14. Assume that the context *working_hours* is defined as in figure 9.15, that this context is valid between 8 am and 6 pm. As shown in figure 9.19, we might derive the following concrete authorization:

- $Is_permitted(john, acroread, account_2.txt)$

Actually, the fact that this authorization is derived depends on the context. If the policy designer set the simulation time at 7:21 pm as shown in figure 9.15, then we get this concrete authorization, and thereby we obtain the first concrete conflict shown in figure 9.19. By contrast, if the simulation time is set to 9 pm, then the concrete permission is not derived and this conflict does not occur.

10. Administration

The enforcement of the AdOr-BAC model (chapter 8) is a work in progress. The objective is to use AdOr-BAC to control accesses to OToKit so that only authorized users are allowed to create new organizations and subjects, assign roles, activities and views to organizations, and in each organization, assign permissions to roles, subjects to roles and permissions to subjects.

If the administration mode is not selected, then any user connected to OToKit is considered as a super-user, like in UNIX. Otherwise, when an authorized user is connected, he will be able to perform only the administrative tasks that he received authorizations for.

9.2.7 Performance results

Several derivation rules are implemented in OToKit. We indicated in section 5.2 that the complexity of an Or-BAC policy is polynomial. However, it is relevant to perform several performance tests. The array of figure 9.20 shows the results. All tests were conducted on the following computer:

- Processor: Pentium-M 1,4 Ghz
- RAM: 256 Mhz
- Operating System: Windows XP SP2

Actually, displaying the entity trees as well as the authorizations is instantaneous. Only the conflict detection shows a visible processing time. Therefore, the tests are realized by timing the detection of potential conflicts. Let us comment on the results array. The first column indicates the number of roles (R), activities (A), views (V), subjects (S), actions (Ac) and objects (Ob) defined in the policy. The second columns shows hierarchy types enforced in the tested policy. For example, at line 3, all entities are “flat” except roles. The third column indicates the number of couples of authorizations that have to be tested in order to find the conflicts. The fourth column contains the number of potential conflicts that have to be detected. Finally, the last column shows the time needed to end the detection of potential conflicts. All lines correspond to detection of potential conflicts with exception of the last one which corresponds to the detection of actual conflicts between concrete authorizations.

This array shows that the conflict detection might last a long time in large policies. However, many optimizations can be introduced in the Prolog program. Furthermore, Prolog is not the most suitable logic tool for our purpose since we based our model on Datalog. Finally, the prevention conflict process is carried out during the security policy conception only. It does not have to be performed afterwards. So it does not lead to any processing overhead of the information system when the policy is actually implemented.

With regard to the actual conflict detection of the last line, the long time shown in the results array might be frightened. Remember that we search here for all conflicting couples. In contrast, when a single security request is made by a user, the derivation process is instantaneous in all of our tests.

Nb of entities	Hierarchies	Nb of tests	Nb of conflicts	Delay
10R 10A 10V		100	100	150ms
20A 20A 20V		400	400	600ms
20R 10A 10V	R	400	400	700ms
20R 20A 10V	R A V	1600	1600	1.6sec
20R 20A 10V	R A V	6400	6400	17.3sec
20R 20A 10V	R A V	3200	3200	11.7sec
20R 20A 10V	R A V	3200	0	6.3sec
20R 20A 10V	O R A V	3200	0	6.3sec
20R 20A 20V 20S 20Ac 20Ob	O R A V	4800	0	15.6sec

Figure 9.20: Performance results array

9.2.8 Future works

I have developed OToKit during my Ph.D. studies in order to implement the concepts, notions and ideas I have put forward. However, OToKit is becoming a collective work since other students are currently working on it. Many improvements and new functionalities can be introduced in OToKit. I just indicates here the most important one:

- Design a convenient and flexible solution to specify logic rules, with the specific purpose of defining new contexts, new constraints and above all new conflict management strategies.
- Manage other types of context than the temporal context. Our priority is to integrate provisional and user-declared contexts.
- Modify the graphical management of hierarchies, in order to integrate multiple inheritance hierarchies.
- Detect the redundant authorizations resulting from the chosen conflict management strategy.
- Save an Or-BAC security policy written with OToKit in a clear format based on XML.
- Integrate a "network policy" mode in which specific attributes, such as IP_address, will enable the policy designer to write network policies, and then to derive automatically firewall rules scripts.
- Continue to implement administration. In particular, we want to enforce more complex views URA, PRA, etc.

The work on the three last items are currently in progress.

9.2.9 Conclusion

The development of OToKit is not finished and several objectives

that it is easy to integrate extensions. The graphical interface offers a convenient way to specify security policies based on the Or-BAC model. We tested OToKit on several policies

and noticed that the concepts of role, activity and view greatly reduce the number of authorizations. This ensures clearer policies. Furthermore, OToKit enables to prove that conflict prevention mechanism is really useful; it indeed helps to find unexpected conflicts. Finally, the ongoing implementation of the AdOr-BAC model in order to administrate OToKit proves that Or-BAC is an auto-administered model. We hope OToKit has a bright future ahead.

Chapter 10

Conclusion

This thesis first gave us the opportunity to examine a large number of existing security models. I described these models according the following four requirements: the structuring of a policy entities, the ability to design dynamic security rules, the expression of negative authorizations and thereby conflict management, and the necessity to provide administrative procedures.

Based on these issues I presented a new security policy model that aims at solving several limits of these previous models. This model, called Or-BAC, is centered on the concept of *Organization*. It makes it possible to manage a security policy in conformity with the organization structure. Furthermore, it enables to provide specific security policies for each part of an organization and to design common policies between organizations which collaborate.

From the traditional triplet $< subject, action, object >$, we introduced three new entities, namely *Role*, *Activity* and $View$ in order to provide organizational policies totally independent of the implementation choices and the target application. More precisely, within a given organization, roles are used to model how subjects are empowered; activities allow us to model what actions are used for; and views allow us to model how objects are used. In other words, the Or-BAC model federates the role-based, activity-based and view-based models. Moreover, we introduced hierarchies. All organizational entities can be structured into hierarchies in such a way that this provides clear and expressive policies. Associated to inheritance mechanisms, hierarchies offer convenient means to manage the policy, and reduce the administration overheads as well as the number of authorizations. Therefore, Or-BAC makes it possible to express security rules, but also to model the structure of an organization and the structure of the entities this organization considers, even in the case of huge and complex organizations.

Elsewhere, modern information systems might not be governed by security rules expressed as simple and definitive facts. In order to match the complexity and evolution of a system, the security policy authorizations have to be dynamic. This is done in the Or-BAC model by introducing a new entity *Context*. Authorizations are associated to contexts and are activated/deactivated according to their contexts validity. We provide solutions to express several types of contexts. Using Or-BAC, it is indeed possible to design temporal and spatial contexts, contexts based on information stored in the information system base, provisional contexts (i.e. actions that must be performed before activating authorizations) and user-

declared contexts that are helpful when only the user is able to evaluate the context validity. Therefore, Or-BAC offers an expressive and flexible framework to design dynamic security policies.

We also offer the possibility, in the Or-BAC model, to express prohibitions, also called negative authorizations. Specifying a policy that includes both permissions and prohibitions may lead to conflict. We suggest to manage conflicting situation by defining a conflict management strategy. This is a major contribution in that this issue is seldom addressed, and that we propose three important features: the conflict management strategy is parametric and can thus be designed by the SSO. Second, such a strategy can detect but also prevent conflicts. We defined a specific condition for this purpose. It guarantees that no conflict will ever happen provided this condition is fulfilled. Third, specifying a conflict management strategy may lead to redundant authorizations, that is, authorizations that will never take precedence and are thereby useless. Or-BAC enables us to detect such authorizations.

We developed an administration model, called AdOr-BAC, associated to Or-BAC. Administrating a security policy is actually essential in order to make sure that the policy remains in adequation with the information system. With AdOr-BAC, we provide a complete administration model which exhibits two characteristics. First, AdOr-BAC is fully compliant with Or-BAC in the sense that administrative authorizations are Or-BAC authorizations. This makes it possible to manage the security and the administration policies using the same tools, and to apply the context expression and the conflict management on the administration policy. Second, updating an Or-BAC policy using AdOr-BAC amounts to adding objects in views and removing object from views. This is a powerful and flexible way to administrate a security policy.

Finally, we exposed two pieces of work carried out with the purpose of putting into practice the Or-BAC model. The first one corresponds to the application of Or-BAC to a network environment. We show that Or-BAC can be used to specify network policies and to automatically derive appropriate firewall configuration scripts. This had been applied on a test network that includes NetFilter firewalls. I also developed a prototype application dedicated to the Or-BAC model. This software, called OToKit, is based on Prolog and offers a user-friendly graphical interface designed in Java. Most of Or-BAC features are integrated in OToKit. OToKit enables to specify Or-BAC policies. One can define organizations, roles, activities, views, contexts and hierarchies, as well as prioritized positive and negative authorizations. Furthermore, conflict detection and conflict prevention are implemented.

Recently, we suggested to define Or-BAC as a more modular model. For this purpose, we suggest first to consider the minimal model, called Or-BAC core, which only integrates organization, roles, activities and views. Then, the other features presented in this dissertation, as well as recent works, are considered as extensions and can be used according to the security needs related to a given policy. This is presented in figure 10.1.

Several research activities can be led on the Or-BAC model. We are currently going deeper into the activity decomposition. We aim at relying on the entity *Activity* and on the hierarchies of activities in order to model workflows. Some activities can be defined as sets of sub-activities that must be performed in series or in parallel. This enables to model commercial transaction for instance. To do so, we need to enlarge the semantics and the expression of the entity *Activity*.

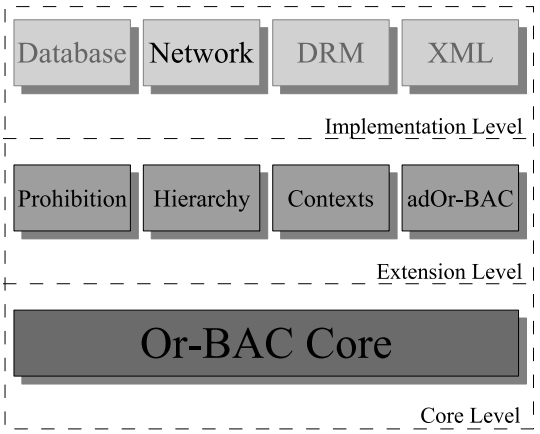


Figure 10.1: New Or-BAC architecture

We are also working on *obligations*. In order to define obligations, let us consider the distinction made in [Jajodia et al. 2001a] between provisions and obligations. Provisions are actions that must be fulfilled to obtain authorizations. Therefore provisions are actions performed $before$ the access decision is taken. In Or-BAC, provisions are modelled as provisional contexts to be specified to get an authorization. On the other hand obligations are actions that must be performed after the access, and more generally, in the future. We are currently working at integrating such obligations in Or-BAC. [Cuppens et al. 2005] describes the NOMAD (Non Atomic Actions and Deadlines) model that pursues this objective.

Current works are also in progress to adapt the Or-BAC model to the Web-services area, and in particular, to define digital right management (DRM) policies. We already suggested an XML version of the Or-BAC model in [Cuppens et al. 2004b]. Notice that more generally, we focus on DRM and obligations with the ultimate objective of extending the Or-BAC model to usage control and not only access control.

Finally, further developments are being carried out on OToKit. We are integrating AdOr-BAC so that administrative authorizations related to the use of OToKit can be specified. Moreover, a specific module for network policies is being designed. We also plan to reconsider OToKit's architecture to provide a more modular use as suggested in figure 10.1. The SSO might then chose to activate only the extensions he needs.

Bibliography

[Ahn and Sandhu 2000] G.-J. Ahn and R. Sandhu. Role-Based Authorization Constraints Specification. *ACM Transactions on Information and System Security*, 3(4), November 2000.

[Ahn and Shin 2001a] G.-J. Ahn and M. E. Shin. Role-based Authorization Constraints Specification Using Object Constraint Language. In *Proceedings of the 10^{th} IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2001)*, Massachusetts, USA, 2001.

[Ahn and Shin 2001b] G.-J. Ahn and M. E. Shin. Role-based Authorization Constraints Specification Using Object Constraint Language. In *Proceedings of 6^{th} IEEE International Workshop on Enterprise Security (WETICE 2001)*, pages 157–162, June 2001.

[Al-Kahtani and Sandhu 2002] M. A. Al-Kahtani and R. Sandhu. A Model for Attribute-Based User-Role Assignment. In *Proceedings of the 18^{th} Annual Computer Security Applications Conference (ACSAC'02)*, page 353, San Diego, California, USA, December 2002.

[Al-Kahtani and Sandhu 2004] M. A. Al-Kahtani and R. Sandhu. Rule-Based RBAC with Negative Authorization. In *Proceedings of the 20^{th} Annual Computer Security Applications Conference (ACSAC 2004)*, pages 405–415, Tucson, Arizona, USA, December 2004.

[Alotaiby and Chen 2004] F. T. Alotaiby and J. X. Chen. A Model for Team-based Access Control (TMAC04). In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)*, Las Vegas, Nevada, USA, April 2004.

[Barka and Sandhu 2000] E. Barka and R. Sandhu. A Role-Based Delegation Model and Some Extensions. In *Proceedings of the 23^{rd} National Information Systems Security Conference*, Baltimore, Maryland, USA, October 2000.

[Barka and Sandhu 2004] E. Barka and R. Sandhu. Role-Based Delegation Model/ Hierarchical Roles (RBDM1). In *Proceedings of the 3^{rd} 20th Annual Computer Security Applications Conference (ACSAC 2004)*, Tucson, Arizona, USA, December 2004.

[Barka 2002] E. S. Barka. Framework for Role-Based Delegation Models. Master's thesis, Graduate Faculty of George Mason University, Fairfax, Virginia, USA, 2002.

[Barkley 1997] J. Barkley. Comparing Simple Role Based Access Control Models and Access Control Lists. In *Proceedings of the 2^{nd} ACM workshop on Role-based Access Control (RBAC 1997)*, Fairfax, Virginia, USA, November 1997.

[Bartal et al. 1999] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. In *20^{th} IEEE Symposium on Security and Privacy*, pages 17–31, Oakland, California, May 1999.

[Becker and Sewell 2004] M. Y. Becker and P. Sewell. Cassandra: Flexible Trust Management, Applied to Electronic Health Records. In *Proceedings of the 17^{th} IEEE Computer Security Foundations Workshop (CSFW 2004)*, Pacific Grove, California, USA, June 2004.

[Bell and LaPadula 1976] D. E. Bell and L. J. LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical Report ESD-TR-75-306, MTR-2997, Rev. 1, MITRE Corporation, Bedfort, MA, March 1976.

[Benferhat et al. 2003] S. Benferhat, R. E. Baida, and F. Cuppens. A Stratification-Based Approach for Handling Conflicts in Access Control. In *Proceedfins of the 8^{th} ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, Lake Come, Italy, June 2003.

[Bertino et al. 2000] E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: a temporal role-based access control model. In *Proceedings of the 5^{th} ACM workshop on Role-based access control*, pages 21–30, July 2000.

[Bertino et al. 2003] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A Logical Framework for Reasoning about Access Control Models. *ACM Transactions on Information and System Security*, 6(1), February 2003.

[Bertino et al. 2004] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. On comparing the Expressing Power Of Access Control Model. In *Proceedings of 3^{rd} Workshop on Foundations of Computer Security (FCS'04)*, Turku, Finland, July 2004.

[Bertino and Ferrari 1997] E. Bertino and E. Ferrari. Administration Policies in a Multipolicy Authorization System. In *Proceedings of the 11^{th} Annual IFIP WG 11.3 Working Conference on Database Security*, pages 341–355, Lake Tahoe, California, USA, August 1997.

[Bertino et al. 1996] E. Bertino, S. Jajodia, and P. Samarati. Supporting Multiple Access Control Policies in Database Systems. In *IEEE Symposium on Security and Privacy*, Oakland, USA, 1996.

[Bettini et al. 2002] C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera. Provisions and Obligations in Policy Management. In *Proceedings of the 28^{th} Very Large Data Bases (VLDB) Conference*, pages 502–513, Hong Kong, China, August 2002.

[Biba 1975] K. J. Biba. Integrity consideration for secure computer systems. Number MTR-3153, June 1975.

[Botha 2001] R. A. Botha. CoSAWoE - A Model for Context-sensitive Access Control in Workflow Environments. Master's thesis, Faculty of Natural Sciences of the Rand Afrikaans University, Johannesburg, South African, November 2001.

[Bradshaw et al. 2003] J. Bradshaw, A. Uszok, R. Jeffers, N. Suri, P. Hayes, M. Burstein, A. Acquisti, B. Benyo, M. Breedy, M. Carvalho, D. Diller, M. Johnson, S. Kulkarni, J. Lott, M. Sierhuis, and R. V. Hoof. Representation and Reasoning for DAML-Based Policy and Domain Services in KAoS and Nomads. In *Proceedings of the Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2003)*, pages 835–842, Melbourne, Australia, July 2003.

[Carrère et al. 1999] J. Carrère, F. Cuppens, and C. Saurel. SACADDOS: a support tool to manage multilevel documents. In *Database Security, 12: Status and Prospects. Results of the IFIP WG 11.3 Workshop on Database Security*, pages 173–188, Fairfax, Virginia, USA, July 1999. Kluwer Academic Press.

[Checkpoint 2004] Checkpoint. Firewall-1. 2004. In http://www.checkpoint.com/.

[Cholewka et al. 2000] D. G. Cholewka, R. A. Botha, and J. H. P. Eloff. A Context-sensitive Access Control Model and Prototype Implementation. In *IFIP TC 11 16^{th} Annual Working Conference on Information Security*, Beijing, China, August 2000.

[Cholvy and Cuppens 1997] L. Cholvy and F. Cuppens. Analyzing Consistency of Security Policies. In *IEEE Symposium on Security and Privacy*, Oakland, California, USA, May 1997.

[Cohen et al. 2002] E. Cohen, R. K. Thomas, W. Winsborough, and D. Shands. Models for Coalition-based Access Control (CBAC). In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, Monterey, California, USA, June 2002.

[Covington et al. 2001] M. J. Covington, W. Long, S. Srinivasan, A. Dey, M. Ahamad, and G. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001)*, Chantilly, Virginia, USA, May 2001.

[Covington et al. 2000] M. J. Covington, M. J. Moyer, and M. Ahamad. Generalized role-based access control for securing future applications. In *Proceedings of the 23rd National Information Systems Security Conference, (NISSC)*, Baltimore, MD, USA, October 2000.

[Crampton 2004] J. Crampton. An algebraic approach to the analysis of constrained workflow systems. In *Proceedings of the 3rd Workshop on Foundations of Computer Security (FCS'04)*, pages 61–64, Turku, Finland, July 2004.

[Crampton and Loizon 2002] J. Crampton and G. Loizon. SARBAC: A New Model for Role-Based Administration. Technical Report BBKCS-02-09, Birkbeck College, University of London, July 2002.

[Crampton and Loizou 2003] J. Crampton and G. Loizou. Administrative scope: A foundation for role-based administrative models. *ACM Transactions on Information and System Security (TISSEC)*, 6(2):201–231, May 2003.

[Cuppens et al. 2001] F. Cuppens, L. Cholvy, C. Saurel, and J. Carrère. Merging Regulations: analysis of a practical example. *International Journal of Intelligent Systems*, 16(11), November 2001.

[Cuppens et al. 2004a] F. Cuppens, N. Cuppens-Boulahia, and A. Miège. Inheritance hierarchies in the Or-BAC Model and application in a network environment. In *Proceedings of the 3rd Workshop on Foundations of Computer Security (FCS'04)*, Turku, Finland, July 2004.

[Cuppens et al. 2005] F. Cuppens, N. Cuppens-Boulahia, and T. Sans. NOMAD: A Security Model With Non Atomic Actions and Deadlines. In *Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW 2005)*, Aix-en-Provence, France, June 2005.

[Cuppens et al. 2004b] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège. A formal approach to specify and deploy a network security policy. In *Second Workshop on Formal Aspects in Security and Trust (FAST)*, Toulouse, France, August 2004.

[Cuppens and Gabillon 1996] F. Cuppens and A. Gabillon. Modelling a Multilevel Database with Temporal Downgrading Functionalities. In *Database Security IX: Status and Prospects, Proceedings of the Ninth Annual IFIP WG11 Working Conference on Database Security (DBSec)*, pages 145–164, Rensselaerville, New York, USA, 1996.

[Cuppens and Miège 2003a] F. Cuppens and A. Miège. Administration Model for Or-BAC. In *International Federated Conferences (OTM'03), Workshop on Metadata for Security*, pages 754–768, Catania, Sicily, Italy, November 2003.

[Cuppens and Miège 2003b] F. Cuppens and A. Miège. Conflict management in the Or-BAC model. Technical report, ENST Bretagne, December 2003.

[Cuppens and Miège 2003c] F. Cuppens and A. Miège. Modelling contexts in the Or-BAC model. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003)*, pages 416–427, Las Vegas, Nevada, USA, December 2003.

[Cuppens and Miège 2004a] F. Cuppens and A. Miège. AdOrBAC: An Adminsitration Model for Or-BAC. *Special issue of the International Journal of Computer Systems Science and Engineering*, 19(3), May 2004.

[Cuppens and Miège 2004b] F. Cuppens and A. Miège. High level conflict management strategies in advanced access control models. Technical report, ENST Bretagne, November 2004.

[Cuppens and Miège 2004c] F. Cuppens and A. Miège. Or-BAC: Organization Based Access Control. In *Distribution des Données à Grande Echelle (DRUIDE 2004)*, Le Croisic, France, May 2004.

[DAC 1987] DAC. A Guide to Understanding Discretionary Access Control in Trusted Systems. 1987.

[Damianou et al. 2001] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *International Workshop, Policies for Distributed Systems and Neworks (Policy 2001)*, Bristol, UK, January 2001.

[Degu and Bastien 2003] C. Degu and G. Bastien. CCP Cisco Secure PIX firewall Advanced Exam Certification Guide. April 2003.

[Donaldson et al. 1990] A. Donaldson, J. W. Taylor, and D. M. Chizmadia. Trusted MINIX: A Worked Example. In *Proceedings of the 13th National Computer Security Conference*, Washington DC, USA, October 1990.

[Ferraiolo et al. 1993] D. F. Ferraiolo, D. M. Guilber, and N. Lynch. An examination of federal and commercial access control policy needs. In *Proceedings of the 16th NIST-NSA National Computer Security Conference*, pages 107–116, Baltimore, Maryland, USA, September 1993.

[Ferraiolo and Kuhn 1992] D. F. Ferraiolo and R. Kuhn. Role-Based Access Controls. In Z. Ruthberg and W. Polk, editors, *Proceedings of the 15th NIST-NSA National Computer Security Conference*, pages 554–563, Baltimore, MD, 13-16 October 1992.

[Ferraiolo et al. 2001] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):222–274, August 2001.

[Gavrila and Barkley 1998] S. I. Gavrila and J. F. Barkley. Formal specification for role based access control user/role and role/role relationship management. In *Proceedings of 3rd ACM workshop on Role-based access control*, pages 81–90, Fairfax, Virginia, USA, November 1998.

[Georgiadis et al. 2001] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas. Flexible team-based access control using contexts. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001)*, Chantilly, Virginia, USA, May 2001.

[Goh and Baldwin 1998] C. Goh and A. Baldwin. Towards a More Complete Model of Role. In *Proceedings of the 3rd ACM workshop on Role-based access control (RBAC 1998)*, Fairfax, Virginia, USA, October 1998.

[Grahne and Ghelli 2002] G. Grahne and G. Ghelli. *Database Programming Language*. Springer, ISBN 3-54044-080-1, October 2002. 1st edition.

[Greco et al. 1992] S. Greco, N. Leone, and P. Rullo. COMPLEX: An Object-Oriented Logic Programming System. *IEEE Transaction on Knowledge and Data Engineering*, 4(4):344–359, august 1992.

[Grossi and Dignum 2004] D. Grossi and F. Dignum. Abstract and Concrete Norms in Institutions. Technical report, Institute of Information and Computing Science, Utrecht Univesity, 2004.

[Guiri 1995] L. Guiri. A new model for role-based access control. In *Proceedings of the 11th Annual Computer Security Applications Conference*, pages 249–255, New Orleans, LA, December 1995.

[Halpern and Weissman 2003] J. Halpern and V. Weissman. Using First-order Logic to Reason about Policies. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW 2003)*, Pacific Grove, California, USA, June 2003.

[Harrison et al. 1976] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *Communication of ACM*, 19(8):461–471, August 1976.

[Hassan and Hudec 2003] A. Hassan and L. Hudec. Role Based Network Security Model: A Forward Step towards Firewall Management. In *Workshop On Security of Information Technologies*, Algiers, December 2003.

[Honeywell 1984] Honeywell. Multics Programmer's Manual-Reference Guide. volume AG91, 1984. Honeywell Informations Systems, Inc.

[ITSEC 1991] ITSEC. Information Technology Security Evaluation Criteria (ITSEC) v1.2. Technical report, 1991.

[Jajodia et al. 2001a] S. Jajodia, M. Kudo, and V. Subrahmanian. Provisional Authorizations. In A. Ghosh, editor, *E-commerce Security and Privacy*, pages 133–159, 2001. Kluwer Academic Publishers.

[Jajodia et al. 2001b] S. Jajodia, P. Samarati, M. Sapino, and V. Subrahmanian. Flexible Support for Multiple Access Control Policies. *ACM Transactions on Database Systems*, 26(2), June 2001.

[Jajodia et al. 1997] S. Jajodia, S. Samarati, and V. S. Subrahmanian. A logical Language for Expressing Authorizations. In *IEEE Symposium on Security and Privacy*, Oakland, California, USA, May 1997.

[Jones et al. 1976] A. K. Jones, R. Lipton, and L. Snyder. A linear time algorithm for deciding security. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, pages 33–41, Houston, Texas, USA, October 1976.

[Jonscher and Dittrich 1996] D. Jonscher and K. Dittrich. Argos - A Configurable Access Control System for Interoperable Environments. pages 43–60, January 1996.

[Kang et al. 2001] J.-M. Kang, W. Shin, C.-G. Park, and D.-I. Lee. Extended BLP Security Model Based on Process Reliability for Secure Linux Kernel. In *Proceedings of the 8th Pacific Rim International Symposium on Dependable Computing (PRDC 2001)*, page 299, Seoul, Korea, December 2001.

[Kern and Moffet 2003] A. Kern and A. S. J. Moffet. An Administration Concept for the Enterprise Role-Based Access Control Model. In *Proceedings of the 9th Symposium on Access Control Models and Technologies (SACMAT 2003)*, pages 3–11, June 2003.

[Khayat and Abdallah 2003] E. Khayat and A. Abdallah. A Formal Model for Flat Role-Based Access Control. In *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2003)*, Tunis, Tunisia, July 2003.

[Koch et al. 2002] M. Koch, L. V. Mancini, and F. Parisi-Presicce. A graph-based formalism for RBAC. *ACM Transactions on Information and System Security (TISSEC)*, 5(3):332–365, August 2002.

[Koch et al. 2004] M. Koch, L. V. Mancini, and F. Parisi-Presicce. Role-based Authorization Constraints Specification Using Object Constraint Language. In *Proceedings of the 9th Symposium on Access Control Models and Technologies (SACMAT 2004)*, pages 97–104, June 2004.

[Kudo and Hada 2000] M. Kudo and S. Hada. XML Document Security based on Provisional Authorization. In *Proceedings of the 7th ACM Conference on Computer and Communication Security (CCS 2000)*, Athens, Greece, 2000.

[Kuhn 2002] M. G. Kuhn. Optical Time-Domain Eavesdropping Risks of CRT Displays. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Oakland, California, USA, 2002.

[Kurland 2003] V. Kurland. Firewall Builder. *White paper*, 2003.

[Lampson 1969] B. Lampson. Dynamic protection structures. In *AFIPS Conf. Proc.*, pages 27–38, March 1969.

[Lampson 1971] B. Lampson. Protection. In *5th Princeton Symposium on Information Sciences and Systems*, pages 437–443, March 1971.

[Lupu and Sloman 1999] E. C. Lupu and M. Sloman. Conflicts in Policy-Based Distributed Systems Management. *IEEE Transactions on Software Engineering*, 5(6):852–869, November 1999.

[McDaniel 2003] P. McDaniel. On Context in Authorization Policy. In *Proceedings of the 8th ACM Symposium On Access Control Models and Technologies (SACMAT 2003)*, Como, Italy, June 2003.

[Moffet 1998] J. D. Moffet. Control Principles and Role Hierarchies. In *Proceedings of the 3rd ACM Workshop on Role-Based Access Control*, Nicosia, Cyprus, October 1998.

[Moffet and Lupu 1999] J. D. Moffet and E. C. Lupu. The use of role hierarchies in access control. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, pages 107–116, Fairfax, Virginia, USA, October 1999.

[Oh and Park 2001a] S. Oh and S. Park. An Improved Administration Method on Role-Based Access Control in the Enterprise Environment. *Journal of Inforamtion Science and Engineering*, 17(6):921–944, November 2001. Elsevier Science Ltd., Oxford, UK.

[Oh and Park 2001b] S. Oh and S. Park. Enterprise Model as a Basis of Administration on Role-Based Access Control. In *Proceedings of the 3rd International Symposium on Cooperative Database Systems for Advanced Applications (CODAS 2001)*, pages 165–174, Beijing, China, 2001.

[Oh and Park 2003] S. Oh and S. Park. Task–role-based access control model. *Information Systems*, 28(6):533–562, 2003. Elsevier Science Ltd., Oxford, UK.

[Oh et al. 2003] S. Oh, R. Sandhu, and X. Zhang. An Effective Role Administration Model Using Organization Structure. *ACM Transactions on Information and System Security (TISSEC)*, 2003.

[Oj and Sandhu 2000] S. Oj and R. Sandhu. An Integration Model of Role-Based Access Control and Activity-Based Access Control Using Task. In *Proceedings of the 14th Annual IFIP WG 11.3 Working Conference on Database Security*, August 2000.

[Or-BAC 2003] Or-BAC. A. Abou El Kalam and R. El Baida and P. Balbiani and S. Benferhat and F. Cuppens and Y. Deswarte and A. Miège and C. Saurel and G. Trouessin. Organization Based Access Control. In *Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, pages 120–134, Lake Come, Italy, June 2003.

[Park 2003] J. Park. Usage Control: A Unified Framework for Next Generation Access Control. Master's thesis, Graduate Faculty of George Mason University, Fairfax, Virginia, USA, 2003.

[Park and Sandhu 2002] J. Park and R. Sandhu. Originator Control in Usage Control. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, pages 60–66, Monterey, California, USA, June 2002. IEEE Computer Society.

[Rabitti et al. 1991] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A Model of Auhtorization for Next-Generation Database Systems. *ACM Transactions on Database Systems*, 16(1):88–131, March 1991.

[Russell 2002] R. Russell. Linux 2.4 Packet Filtering. 2002. In http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.html.

[Sandhu et al. 1999] R. Sandhu, Bhamidipati, and Q. Munawer. *The ARBAC97 Model for Role-Based Administration of Roles*, volume 2. ACM Press, February 1999.

[Sandhu and Bhamidipati 1997] R. Sandhu and V. Bhamidipati. The URA97 Model for Role-Based User-Role Assignment. In *Proceedings of IFIP WG 11.3 Workshop on Database Security*. North-Holland, Lake Tahoe, California, 1997.

[Sandhu et al. 1997] R. Sandhu, V. Bhamidipati, E. Coyne, S. Ganta, and C. Youman. The arbac97 model for role-based administration of roles: Preliminary description and outline. In *Proceedins of the 2nd ACM Workshop one Role-Based Access Control*, Fairfax, Virginia, USA, November 1997.

[Sandhu and Munawer 1998] R. Sandhu and Q. Munawer. The RRA97 Model for Role-Based Administration of Role Hierarchies. In *Proceedings of the 14th Annual Computer Security Applications Conference (ACSAC'98)*. Phoenix, Arizona, USA, December 7-11 1998.

[Sandhu and Munawer 1999] R. Sandhu and Q. Munawer. The ARBAC99 Model For Administration of Roles. In *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC'99)*, Phoenix, Arizona, December 1999. IEEE Computer Society Press.

[Sandhu and Park 2004] R. Sandhu and J. Park. The UCON_{ABC} Usage Control Model. *ACM Transactions an Information and System Security (TISSEC)*, 7(1), February 2004.

[Sandhu 1988] R. S. Sandhu. The schematic protection model: its definition and analysis for acyclic attenuating schemes. *Journal of the ACM*, 35(2):404–432, April 1988.

[Sandhu 1992] R. S. Sandhu. The Typed Access Matrix Model. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, Oakland, California, USA, May 1992.

[Sandhu 1998] R. S. Sandhu. Role-Based Access Control. *Advances in Computers*, 46, 1998.

[Sandhu et al. 1996] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.

[Shen and Dewan 1992] H. Shen and P. Dewan. Access Control for Collaborative Environments. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 51–58, Toronto, Ontario, Canada, November 1992.

[Shoenfield 2001] J. R. Shoenfield. *Mathematical logic*. AK Peters, ISBN 1-56881-135-7, 2001.

[Solworth and Sloan 2004] J. A. Solworth and R. H. Sloan. Security Property Based Administrative Controls. In *Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS 2004)*, September 2004.

[TCSEC 1985] TCSEC. Trusted Computer System Evaluation Criteria. Technical report, DoD 5200.28-STD, 1985.

[Thomas and Sandhu 1997] R. Thomas and R. Sandhu. Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management. In *Proceedings of the 11th IFIP Working Conference on Database Security*, Lake Tahoe, California, USA, August 1997.

[Thomas 1997] R. K. Thomas. Team-Based Access Control (TMAC): A Primitive for Applying Role-Based Access Controls in Collaborative Environments. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control (RBAC 1997)*, pages 13–19, Fairfax, Virginia, USA, November 1997.

[Thomas and Sandhu 1994] R. K. Thomas and R. S. Sandhu. Conceptual Foundations for a Model of Task-based Authorizations. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop (CSFW 1994)*, pages 66–67, Franconia, New Hampshire, USA, June 1994.

[Ullman 1989] J. D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1,2. Computer Science Press, edit. W. H. Freeman, 1989.

[Weber 1997] H. A. Weber. Role-Based Access Control: The NIST Solution. Technical report, Fairfax, Virginia, USA, November 1997.

Appendix A

Table

A.1 Basic predicates of Or-BAC

Predicate name	Domain	Description
Relevant_role	$Org \times \mathcal{R}$	If *org* is an organization and r a role, then $Relevant_role(org, r)$ means that playing role r is defined in organization *org*.
Relevant_activity	$Org \times \mathcal{A}$	If *org* is an organization and a is an activity, then $Relevant_activity(org, a)$ means that performing activity a is defined in organization *org*.
Relevant_view	$Org \times \mathcal{V}$	If *org* is an organization and v is a view, then $Relevant_view(org, v)$ means that using view v is defined in organization *org*.
Relevant_context	$Org \times \mathcal{C}$	If *org* is an organization and v is a view, then $Relevant_view(org, v)$ means that using view v is defined in organization *org*.
Empower	$Org \times S \times \mathcal{R}$	If *org* is an organization, s a subject and r a role, then $Empower(org, s, r)$ means that *org* empowers subject s in role r.
Consider	$Org \times A \times \mathcal{A}$	If *org* is an organization, α is an action and a is an activity, then $Consider(org, \alpha, a)$ means that *org* considers that action α falls within the activity a.
Use	$Org \times O \times \mathcal{V}$	If *org* is an organization, o is an object and v is a view, then $Use(org, o, v)$ means that *org* uses object o in view v.
Hold	$Org \times S \times A \times O \times \mathcal{C}$	If *org* is an organization, s a subject, α an action, o an object and c a context, then $Define(org, s, \alpha, o, c)$ means that within organization *org*, context c holds between subject s, action α and object o.
sub_role	$Org \times \mathcal{R} \times \mathcal{R}$	If *org* is an organization, if r_1 and r_2 are roles, then $sub_role(org, r_2, r_1)$ means that r_2 is a sub-role of r_1 in organization *org*.
sub_activity	$Org \times \mathcal{A} \times \mathcal{A}$	If *org* is an organization, if a_1 and a_2 are activities, then $sub_activity(org, a_2, a_1)$ means that a_2 is a sub-activity of a_1 in organization *org*.

sub_view	$Org \times \mathcal{V} \times \mathcal{V}$	If org is an organization, if v_1 and v_2 are views, then $sub_view(org, v_2, v_1)$ means that v_2 is a sub-view of v_1 in organization org.
sub_context	$Org \times \mathcal{C} \times \mathcal{C}$	If org is an organization, if c_1 and c_2 are contexts, then $sub_context(org, c_2, c_1)$ means that c_2 is a sub-context of c_1 in organization org.
sub_organization	$Org \times Org$	If org_1 and org_2 are organizations, then $sub_organization(org_2, org_1)$ means that org_2 is a sub-organization of org_1.
separeted_role	$Org \times \mathcal{R} \times Org \times \mathcal{R}$	If org_1 and org_2 are organizations, if r_1 and r_2 are roles, then $separeted_role(org_1, r_1, org_2, r_2)$ means that that no subject s in S is permitted to be at the same time empowered into role r_1 in org_1 and into r_2 in org_2.
separeted_activity	$Org \times \mathcal{A} \times Org \times \mathcal{A}$	If org_1 and org_2 are organizations, if a_1 and a_2 are activities, then $separeted_activity(org_1, a_1, org_2, a_2)$ means that no action α in A is can be at the same time considered as activity a_1 in org_1 and as a_2 in org_2.
separeted_view	$Org \times \mathcal{V} \times Org \times \mathcal{V}$	If org_1 and org_2 are organizations, if v_1 and v_2 are views, then $separeted_view(org_1, v_1, org_2, v_2)$ means that no object o in O can be at the same time used into view v_1 in org_1 and into v_2 in org_2.
separeted_context	$Org \times \mathcal{C} \times Org \times \mathcal{C}$	If c_1 is a context defined in organization org_1 by condition $Cond1$ and if c_2 is a context defined in organization org_2 by condition $Cond2$. $separeted_context(org_1, c_1, org_2, c_2)$ means that condition $Cond1 \wedge Cond2$ is inconsistent.

A.2 Positive and negative authorizations specification in Or-BAC

Predicate name	Domain	Description
Permission	$Org \times \mathcal{R} \times \mathcal{A} \times \mathcal{V} \times \mathcal{C}$	If *org* is an organization, r a role, a an activity , v a view and c a context, then $Permission(org, r, a, v, c)$ means that organization *org* grants to role r the permission to perform activity a on view v in context c.
Prohibition	$Org \times \mathcal{R} \times \mathcal{A} \times \mathcal{V} \times \mathcal{C}$	If *org* is an organization, r a role, a an activity , v a view and c a context, then $Prohibition(org, r, a, v, c)$ means that organization *org* prohibits role r from performing activity a on view v in context c.
Is_permitted	$S \times A \times O$	If s is a subject, α an action, o an object, then $Is_permitted(s, \alpha, o)$ means that s is concretely permitted to perform action α on object o.
Is_prohibited	$S \times A \times O$	If s is a subject, α an action, o an object, then $Is_prohibited(s, \alpha, o)$ means that s is concretely prohibited to perform action α on object o.

A.3 Derivation rules in Or-BAC

Rule name	Derivation rule definition	Description
RG_1	$\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$ $Permission(org, r, v, a, c) \wedge$ $Emplower(org, s, r) \wedge$ $Consider(org, \alpha, a) \wedge$ $Use(org, o, v) \wedge$ $Hold(org, s, \alpha, o, c)$ $\rightarrow Is_permitted(s, \alpha, o)$	if organization *org*, within context c, grants role r permission to perform activity a on view v, if *org* empowers subject s in role r, if *org* considers that action α falls within the activity a, if *org* uses object o in view v and if, within *org*, the context c is true between s, α and s, then s has permission to perform α on o.
RG_2	$\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$ $Prohibition(org, r, v, a, c) \wedge$ $Emplower(org, s, r) \wedge$ $Consider(org, \alpha, a) \wedge$ $Use(org, o, v) \wedge$ $Hold(org, s, \alpha, o, c)$ $\rightarrow Is_prohibited(s, \alpha, o)$	if organization *org*, within context c, prohibits role r to perform activity a on view v, if *org* empowers subject s in role r, if *org* considers that action α falls within the activity a, if *org* uses object o in view v and if, within *org*, the context c is true between s, α and s, then s is forbidden to perform α on o.
RH_1	$\forall org \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$ $sub_role(org, r_2, r_1) \wedge$ $Permission(org, r_1, a, v, c)$ $\rightarrow Permission(org, r_2, a, v, c)$	If role r_2 is a sub-role of role r_1 in organization *org*, then every permission assigned to role r_1 in organization *org* is also assigned to role r_2.

RH_2	$\forall org \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$ $sub_role(org, r_2, r_1) \wedge$ $Prohibition(org, r_1, a, v, c)$ $\rightarrow Prohibition(org, r_2, a, v, c)$	If role r_2 is a sub-role of role r_1 in organization org, then every prohibition assigned to role r_1 in organization org is also assigned to role r_2.
AH_1	$\forall org \in Org, \forall r \in \mathcal{R}, \forall a_1 \in \mathcal{A}, \forall a_2 \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$ $sub_activity(org, a_2, a_1) \wedge$ $Permission(org, r, a_1, v, c)$ $\rightarrow Permission(org, r, a_2, v, c)$	If activity a_2 is a sub-activity of activity a_1 in organization org, then every permission assigned to activity a_1 in organization org is also assigned to activity a_2.
AH_2	$\forall org \in Org, \forall r \in \mathcal{R}, \forall a_1 \in \mathcal{A}, \forall a_2 \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$ $sub_activity(org, a_2, a_1) \wedge$ $Prohibition(org, r, a_1, v, c)$ $\rightarrow Prohibition(org, r, a_2, v, c)$	If activity a_2 is a sub-activity of activity a_1 in organization org, then every prohibition assigned to activity a_1 in organization org is also assigned to activity a_2.
VH_1	$\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v_1 \in \mathcal{V}, \forall v_2 \in \mathcal{V}, \forall c \in \mathcal{C},$ $sub_view(org, v_2, v_1) \wedge$ $Permission(org, r, a, v_1, c)$ $\rightarrow Permission(org, r, a, v_2, c)$	If view v_2 is a sub-view of view v_1 in organization org, then every permission assigned to view v_1 in organization org is also assigned to view v_2.
VH_2	$\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v_1 \in \mathcal{V}, \forall v_2 \in \mathcal{V}, \forall c \in \mathcal{C},$ $sub_view(org, r_2, r_1) \wedge$ $Prohibition(org, r, a, v_1, c)$ $\rightarrow Prohibition(org, r, a, v_2, c$	If view v_2 is a sub-view of view v_1 in organization org, then every prohibition assigned to view v_1 in organization org is also assigned to view v_2.
HH_1	$\forall org_a \in Org, \forall org_b \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$ $sub_organization(org_b, org_a) \wedge$ $sub_role(org_a, r_2, r_1) \wedge$ $Relevant_role(org_b, r_1) \wedge$ $Relevant_role(org_b, r_2)$ $\rightarrow sub_role(org_b, r_2, r_1)$	If organizations org_b is a sub-organization of organization org_b, if r_1 and r_2 are roles, if r_2 is a sub-role of r_1 in org_a and if r_1 and r_2 are relevant in org_b, then r_2 is a sub-role of r_1 in org_b in org_a also applies in org_b.
OH_1	$\forall org_1 \in Org, \forall org_2 \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$ $sub_organization(org_2, org_1)$ $Permission(org_1, r, a, v, c) \wedge$ $Relevant_role(org_2, r) \wedge$ $Relevant_activity(org_2, a) \wedge$ $Relevant_view(org_2, v) \wedge$ $Relevant_context(org_2, c)$ $\rightarrow Permission(org_2, r, a, v, c)$	If organization org_2 is a sub-organization of organization org_1, if org_1 grants role r_1 permission to perform activity a on view v in context c within in context c, and if r, a, v and c are relevant in org_2, then this permission applies in org_2.
OH_2	$\forall org_1 \in Org, \forall org_2 \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$ $sub_organization(org_2, org_1)$ $Prohibition(org_1, r, a, v, c) \wedge$ $Relevant_role(org_2, r) \wedge$ $Relevant_activity(org_2, a) \wedge$ $Relevant_view(org_2, v) \wedge$ $Relevant_context(org_2, c)$ $\rightarrow Prohibition(org_2, r, a, v, c)$	If organization org_2 is a sub-organization of organization org_1, if org_1 prohibits role r_1 to perform activity a on view v in context c within in context c, and if r, a, v and c are relevant in org_2, then this prohibition applies in org_2.

A.4 Basic constraints in Or-BAC

Rule name	Constraint definition	Description
C_1	$\forall org \in Org, \forall s \in S, \forall r \in \mathcal{R}$, $Empower(org, s, r) \wedge \neg Relevant_role(org, r) \rightarrow error()$	An organization org should not empower a subject s in role r if r is not relevant in org.
C_2	$\forall org \in Org, \forall o \in O, \forall v \in \mathcal{V}$, $Use(org, o, v) \wedge \neg Relevant_view(org, v) \rightarrow error()$	An organization org should not use an object o in view v if v is not relevant in org.
C_3	$\forall org \in Org, \forall \alpha \in A, \forall a \in \mathcal{A}$, $Use(org, \alpha, a) \wedge \neg Relevant_activity(org, v) \rightarrow error()$	An organization org should not consider an action α as an activity a if a is not relevant in org.
C_4	$\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall c \in \mathcal{C}$, $Hold(org, s, \alpha, o, c) \wedge \neg Relevant_context(org, c) \rightarrow error()$	An organization org should not specify a context c if c is not relevant in org.
C_5	$\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$, $Permission(org, r, a, v, c) \wedge \neg(Relevant_role(org, r) \wedge Relevant_activity(org, a) \wedge Relevant_view(org, v) \wedge Relevant_context(org, c)) \rightarrow error()$	All entities involved in a positive authorization specified in organization org must be relevant in org.
C_6	$\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$, $Prohibition(org, r, a, v, c) \wedge \neg(Relevant_role(org, r) \wedge Relevant_activity(org, a) \wedge Relevant_view(org, v) \wedge Relevant_context(org, c)) \rightarrow error()$	All entities involved in a negative authorization specified in organization org must be relevant in org.
C_7	$\forall org_1 \in Org, \forall org_2 \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}, \forall s \in S$, $separated_role(org_1, r_1, org_2, r_2) \wedge Empower(org_1, s, r_1) \wedge Empower(org_2, s, r_2) \rightarrow error()$	if role r_1 in organization org_1 is separated from role r_2 in organization org_2, then a subject s cannot play both role r_1 in org_1 and role r_2 in org_2.
C_8	$\forall org_1 \in Org, \forall org_2 \in Org, \forall a_1 \in \mathcal{A}, \forall a_2 \in \mathcal{A}, \forall \alpha \in A$, $separated_activity(org_1, a_1, org_2, a_2) \wedge Consider(org_1, \alpha, a_1) \wedge Consider(org_2, \alpha, a_2) \rightarrow error()$	if activity a_1 in organization org_1 is separated from activity r_2 in organization org_2, then an action α cannot be considered as an activity a_1 in org_1 and an activity a_2 in org_2.

C_9	$\forall org_1 \in Org, \forall org_2 \in Org, \forall v_1 \in \mathcal{V}, \forall v_2 \in \mathcal{V}, \forall o \in O,$ $separated_view(org_1, v_1, org_2, v_2) \wedge$ $Use(org_1, o, v_1) \wedge Use(org_2, o, v_2)$ $\rightarrow error()$	if view v_1 in organization org_1 is separated from view v_2 in organization org_2, then an object o cannot be in the view v_1 in org_1 and in the view v_2 in org_2.
C_{10}	$\forall org_1 \in Org, \forall org_2 \in Org, \forall r \in \mathcal{R},$ $sub_organization(org_1, org_2) \wedge$ $\neg Empower(org_2, org_1, r)$ $\rightarrow error()$	An organization should empower all its sub-organizations in roles.

A.5 Prioritized authorizations in $T_{P_{pol}}$

Predicate name	**Domain**	**Description**
$Permission'$	$Org \times \mathcal{R} \times \mathcal{A} \times \mathcal{V} \times \mathcal{C} \times \mathcal{L}$	If org is an organization, r is a role, v is a view, a is an activity, c is context and l a priority level, then $Permission'(org, r, v, a, c, l)$ means that $Permission(org, r, a, v, c)$ is assigned to priority level l.
$Prohibition'$	$Org \times \mathcal{R} \times \mathcal{A} \times \mathcal{V} \times \mathcal{C} \times \mathcal{L}$	If org is an organization, r is a role, v is a view, a is an activity, c is context and l a priority level, then $Prohibition'(org, r, v, a, c, l)$ means that $Prohibition(org, r, a, v, c)$ is assigned to priority level l.
$Is_permitted'$	$S \times A \times O \times \mathcal{L}$	If s is a subject, α an action, o an object and l a priority level, then $Is_permitted'(s, \alpha, o, l)$ specifies that $Is_permitted(s, \alpha, o)$ is assigned to priority level l.
$Is_prohibited'$	$S \times A \times O \times \mathcal{L}$	If s is a subject, α an action, o an object and l a priority level, then $Is_prohibited'(s, \alpha, o, l)$ specifies that $Is_prohibited(s, \alpha, o)$ is assigned to priority level l.

A.6 Derivation rules in $T_{P_{pol}}$

Rule name	**Derivation rule definition**	**Description**
RG'$_1$	$\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}, \forall l \in \mathcal{L},$ $Permission'(org, r, v, a, c, l) \wedge$ $Emplower(org, s, r) \wedge$ $Consider(org, \alpha, a) \wedge$ $Use(org, o, v) \wedge$ $Hold(org, s, \alpha, o, c)$ $\rightarrow Is_permitted'(s, \alpha, o, l)$	$Is_permitted'$ may be derived with the same priority level as $Permission'$, provided other conditions in the premises are satisfied.

RG'$_2$	$\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}, \forall l \in \mathcal{L}$, $Prohibition'(org, r, v, a, c, l) \wedge$ $Emplower(org, s, r) \wedge$ $Consider(org, \alpha, a) \wedge$ $Use(org, o, v) \wedge$ $Hold(org, s, \alpha, o, c)$ $\rightarrow Is_prohibited'(s, \alpha, o, l)$	$Is_prohibited'$ may be derived with the same priority level as $Prohibition'$, provided other conditions in the premises are satisfied.
ED$_1$	$\forall s \in S, \forall \alpha \in A, \forall o \in O, \forall l_1 \in \mathcal{L}$, $Is_permitted'(s, \alpha, o, l_1) \wedge$ $\neg \exists l_2 \in \mathcal{L}$, $(l_1 \prec l_2 \wedge Is_prohibited'(s, \alpha, o, l_2))$ $\rightarrow Is_permitted(s, \alpha, o)$	A concrete permission can be derived for allowing subject s to perform action α on object o if this permission is labelled at a priority level l_1 and there is no prohibition for s to perform α on o with a priority level l_2 strictly higher than l_1.
ED$_2$	$\forall s \in S, \forall \alpha \in A, \forall o \in O, \forall l_1 \in \mathcal{L}$, $Is_prohibited'(s, \alpha, o, l_1) \wedge$ $\neg \exists l_2 \in \mathcal{L}$, $(l_1 \prec l_2 \wedge Is_permitted'(s, \alpha, o, l_2))$ $\rightarrow Is_prohibited(s, \alpha, o)$	A concrete prohibition applied to subject s, action α and object o ca be derived if this prohibition is labelled at a priority level l_1 and there is no permission for s to perform α on o with a priority level l_2 strictly higher than l_1.
RH'$_1$	$\forall org \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}, \forall l \in \mathcal{L}$, $sub_role(org, r_2, r_1) \wedge$ $Permission'(org, r_1, a, v, c, l)$ $\rightarrow Permission'(org, r_2, a, v, c, l)$	If role r_2 is a sub-role of r_1 and if there is a permission associated with r_1 at a priority level l, then r_2 inherits this permission with the same priority level l.
RH'$_2$, AH'$_1$, AH'$_2$, VH'$_1$, VH'$_2$, OH'$_1$, OH'$_2$	Similarly to RH'$_1$ which is obtained from RH$_1$, these rules are respectively obtained from RH$_2$, AH$_1$, AH$_2$, VH$_1$, VH$_2$, OH$_1$, OH$_2$	

A.7 Constraints in T_{Ppol}

Rule name	**Constraint definition**	**Description**
C'$_5$	$\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}, \forall l \in \mathcal{L}$, $Permission(org, r, a, v, c, l) \wedge$ $\neg (Relevant_role(org, r) \wedge$ $Relevant_activity(org, a) \wedge$ $Relevant_view(org, v) \wedge$ $Relevant_context(org, c))$ $\rightarrow error()$	All entities involved in a prioritized positive authorization specified in organization org must be relevant in org.

C'$_6$	$\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}, \forall l \in \mathcal{L}$, $Prohibition(org, r, a, v, c, l) \wedge$ $\neg(Relevant_role(org, r) \wedge$ $\quad Relevant_activity(org, a) \wedge$ $\quad Relevant_view(org, v) \wedge$ $\quad Relevant_context(org, c))$ $\rightarrow error()$	All entities involved in a prioritized negative authorization specified in organization org must be relevant in org.

Appendix B

French summary

B.1 Introduction

L'emploi des technologies de l'information est de plus en plus fréquent, aussi bien dans le monde professionnel que dans la sphère privée. En effet, l'émergence de nouveaux outils, l'amélioration des performances et l'augmentation des débits des réseaux nous poussent à nous appuyer de plus en plus sur des données numériques pour le traitement, le stockage et l'échange d'information. Si les bénéfices sont reconnus par tous, les risques liés à la sécurité des données (confidentialité, intégrité, disponibilité) augmentent parallèlement. La mise en place d'outils et de méthodes de protection suppose au préalable la définition d'un règlement de sécurité indiquant les actions permises et interdites. Un tel règlement fait partie de la *politique de sécurité*. En particulier, la politique de contrôle d'accès consiste, dans le cadre d'un système d'information et à travers la définition d'autorisations, à accorder des privilèges à des sujets afin qu'ils puissent réaliser des actions sur des ressources. Les sujets peuvent être des utilisateurs, c'est-à-dire des personnes physiques, mais aussi les processus que ces personnes utilisent. Les ressources, aussi appelées objets, sont par exemple les fichiers du système, mais peuvent être également des relations dans une base de données relationnelles, des imprimantes, etc. Enfin, les actions possibles dans un système d'information sont par exemple "lire", "écrire", "exécuter", ou dans une base de données "select", "update", etc. Cette thèse est centrée sur le domaine du contrôle d'accès.

De nombreux modèles de contrôle d'accès ont été proposés depuis plus de trente ans, et la recherche s'est accélérée en particulier depuis le début des années quatre-vingt-dix. Tous ces modèles présentent des avantages et procurent à leur manière des avancées significatives. Néanmoins, tous ces modèles sont trop souvent spécifiques à un domaine d'application et ne tentent de résoudre qu'une seule des problématiques liées au contrôle d'accès. De ce constat, nous concluons qu'il nécessaire de définir un nouveau modèle qui à la fois fédère ces différentes avancées et qui soit, en même temps, adapté à de nombreux domaines d'application. Ce modèle que nous définissons tout au long de la thèse est appelé Or-BAC (*Organization-Based Access Control*).

Dans le chapitre 2 de la thèse, nous présentons les modèles existants en même temps que nous définissons les besoins auxquels notre modèle doit pouvoir répondre. La première exigence correspond à la nécessité de structurer les entités traditionnelles du contrôle d'accès, à savoir les entités *sujet*, *action* et *objet*, afin de permettre la définition de politiques de sécurité

d'un haut niveau d'abstraction et d'offrir la possibilité de gérer des systèmes d'information comportant un grand nombre d'utilisateurs et de ressources. En effet, une politique de sécurité ne doit pas se restreindre à la définition d'un ensemble d'autorisations, mais doit aussi permettre de spécifier la structure de l'organisation. Notre solution consiste à intégrer, au sein d'un même modèle, les notions présentes dans les modèles basés sur les rôles, dans les modèles basés sur les activités et dans les modèles basés sur les vues. Nous présentons notre solution dans la section B.2 à travers la définition de notre modèle de contrôle d'accès.

Dans la mesure où les systèmes d'information sont de plus en plus complexes et les actions qu'ils doivent permettre de réaliser de plus en plus diversifiées, il est essentiel de proposer un formalisme pour la rédaction des règles de sécurité qui soit suffisamment souple et expressif pour spécifier des politiques de contrôle d'accès adaptées. En particulier, il doit être possible d'indiquer dans quelles circonstances une autorisation est activée, afin de ne pas se limiter à l'expression d'autorisations statiques. Nous verrons dans la section B.2 comment à travers la définition de *contextes* notre modèle permet de satisfaire cette exigence.

Le troisième besoin énoncé se rapporte à la nécessité d'offrir au concepteur de la politique de contrôle d'accès la possibilité d'exprimer des permissions (ou autorisations positives) mais aussi des interdictions (ou autorisations négatives). Nous évoquons dans la section B.4 les raisons d'une telle exigence. Nous présentons également comment résoudre les conflits qui peuvent apparaître entre des permissions et des interdictions. Cette solution présente l'originalité de pouvoir détecter et surtout prévenir ces conflits.

Enfin, un modèle de contrôle d'accès ne serait être complet s'il n'était pas associé à des procédures administratives permettant dans un premier temps la création d'une nouvelle politique, puis sa mise à jour au fur et à mesure des modifications apportées au système d'information visé. Plus qu'une liste de procédures, nous définissons un véritable modèle d'administration qui a la particularité d'être totalement compatible en terme de formalisme avec le modèle Or-BAC. Ce nouveau modèle est présenté dans la section B.5.

La dernière section est consacrée à une rapide présentation de travaux réalisés en vue de l'application du modèle Or-BAC à des cas concrets.

B.2 Le modèle Or-BAC

Les premiers travaux sur le modèle Or-BAC ont été réalisés dans le cadre du projet MP6 (Modèles et Politiques de Sécurité pour les Systèmes d'Informations et de Communications en Santé et Social) et ont abouti à une première formulation du modèle [Or-BAC 2003]. MP6[1] était un projet RNRT (Réseau National de Recherche en Télécommunications) financé par le Ministère de la Recherche. Les travaux ultérieurs ont été réalisés dans le cadre de cette thèse.

Un des objectifs majeurs du modèle Or-BAC est de définir une politique de sécurité à deux niveaux, un niveau concret et un niveau organisationnel. Cette notion est d'ailleurs déjà présente dans les ITSEC [ITSEC 1991] où sont définis trois niveaux de politique de sécurité : La politique de sécurité interne (ou organisationnelle), la politique de sécurité système et la politique de sécurité technique. Dans le modèle Or-BAC, le niveau concret correspond aux

[1] http://www.telecom.gouv.fr/rnrt/rnrt/projets/res_01_59.htm

entités concrètes du système comme les utilisateurs, les différentes ressources, les opérations, etc. Afin de définir une politique de sécurité générique, ou organisationnelle, nous réalisons une abstraction de toutes ces entités. L'objectif est double : obtenir une politique de sécurité de haut niveau, indépendante des choix d'implémentation, et rédiger une politique de sécurité organisationnelle sur laquelle un certain nombre de vérifications peuvent être effectuées. La politique de sécurité concrète est ensuite automatiquement déduite de la politique de sécurité organisationnelle.

Les sections suivantes sont consacrées à la définition des entités et des prédicats du modèle Or-BAC. Ces différents éléments sont représentés à la figure B.1. On remarquera la position centrale de l'entité organisation et la présence du contexte.

Le modèle Or-BAC est formalisé à l'aide d'un sous-ensemble de la logique du premier ordre. En effet, une politique de sécurité Or-BAC est envisagée comme un programme Datalog [Shoenfield 2001]. Ainsi, les fonctions ne sont pas admises et les formules doivent être des clauses de Horn. Dans la mesure où nous désirons obtenir des politiques suffisamment expressives, nous considérons en particulier Datalog avec négations, ou Datalog$^{\neg}$. L'emploi de clauses récursives avec des littéraux négatifs ne garantit pas l'obtention de solutions uniques. Nous adoptons alors la sémantique de la stratification[2]. Par conséquent nous imposons dans la rédaction de politiques Or-BAC les deux restrictions suivantes : la contrainte des clauses sûres[3] doit toujours être satisfaite et, de plus, toute politique doit être stratifiable [Ullman 1989].

B.2.1 Les organisations

Le concept d'organisation est central dans ce nouveau modèle. D'abord défini comme un groupe organisé d'entités actives [Or-BAC 2003], c'est-à-dire de sujets jouant certains rôles, nous avons élargi le concept d'organisation à toute entité en charge d'une politique de sécurité [Cuppens et al. 2004a]. Ainsi, une entreprise, mais également un firewall ou un système sont considérés comme des organisations. L'organisation est l'un des paramètres des règles de sécurité, de sorte qu'il est possible de gérer simultanément plusieurs politiques de sécurité associées à différentes organisations. Or-BAC définit des règles spécifiques à l'organisation. En particulier, l'organisation peut être structurée en plusieurs sous-organisations qui ont chacune leur propre politique de sécurité. Il est également possible de spécifier une politique de sécurité générique au niveau d'une organisation mère. Ces sous-organisations peuvent alors hériter en partie de sa politique de sécurité mais aussi ajouter ou supprimer des règles et ainsi définir leur propre politique de sécurité. La définition d'une organisation et de la hiérarchie des sous-organisations qui la composent permet ainsi de faciliter l'administration de la politique de sécurité. D'un point de vue pratique, cette hiérarchie permet de modéliser la structure des organisations réelles qui peuvent être constituées de départements, d'entités, d'unités, etc. Un projet ou un groupe de travail peut également être modélisé par une organisation.

[2]La stratification d'un programme Datalog$^{\neg}$ consiste à définir l'ordre de résolution des règles logiques. Si un programme est stratifiable, il existe alors une solution unique au programme, et celle-ci peut être calculée en un temps polynômial.

[3]Cette contrainte garantie qu'une règle a un nombre fini de solutions. Cette contrainte est respectée pour une règle si toute variable utilisée dans cette règle apparaît dans au moins un prédicat extensionnel positif de la prémisse, ou dans un prédicat qui est lui-même conclusion d'une règle sûre.

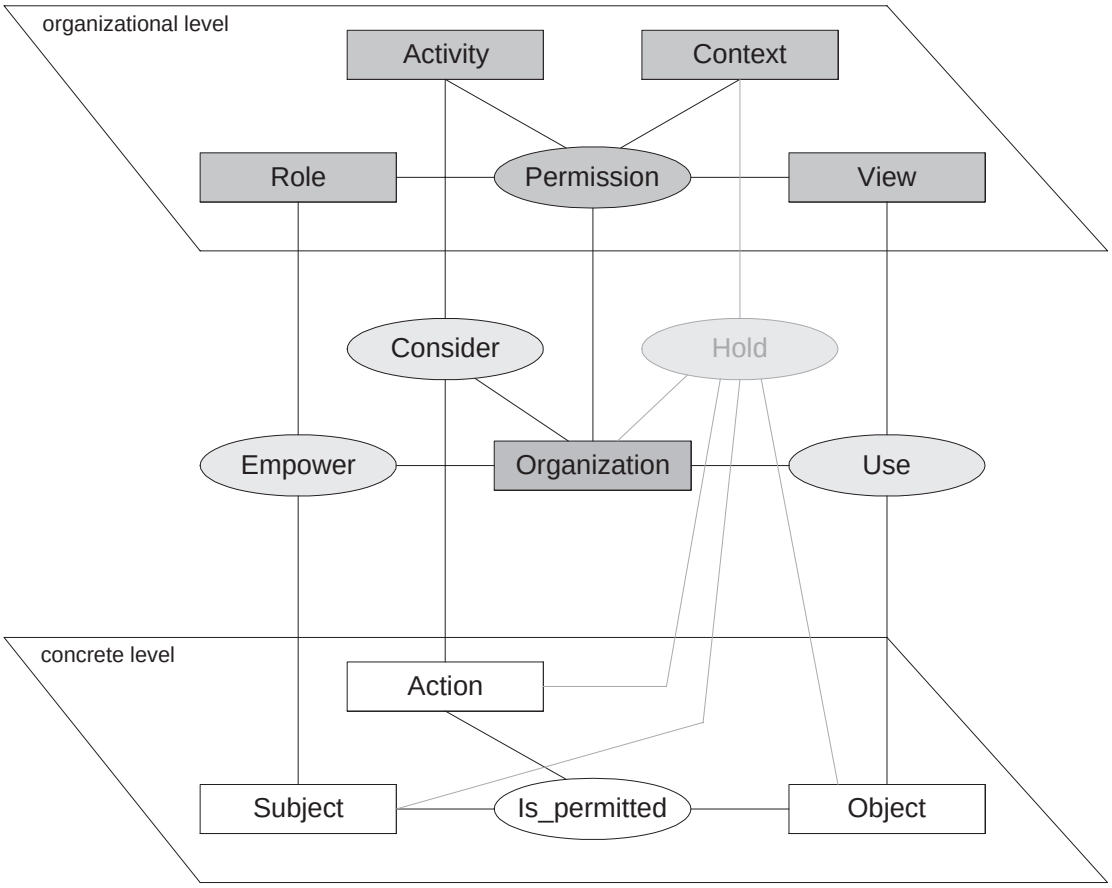


Figure B.1: Le modèle Or-BAC

La hiérarchie d'organisation est exprimée à l'aide du prédicat *sub_organisation* : si org_1 et org_2 sont deux organisations, alors $sub_organisation(org_2, org_1)$ signifie que org_2 est une sous-organisation de org_1.

B.2.2 Les sujets et les rôles

L'entité *Subject* est utilisée différemment selon les modèles de sécurité. Dans le modèle Or-BAC, un sujet peut être soit une entité active, c'est-à-dire un utilisateur, soit une organisation. La notion de rôle a déjà été introduite entre autres dans le modèle RBAC [Sandhu et al. 1996], mais elle diffère quelque peu dans le modèle Or-BAC. Ici, nous considérons qu'un sujet joue un rôle dans une organisation. Ainsi, l'entité *Role* est utilisée pour structurer le lien entre les sujets et les organisations. Nous dirons par exemple que l'utilisateur "Pierre" joue le rôle "administrateur" dans l'organisation "département informatique". Les permissions obtenues par Pierre dépendent ainsi de son rôle et de l'organisation dans laquelle il l'exerce. La relation qui lie le sujet, le rôle et l'organisation est appelée *Empower*. L'exemple précédent peut alors s'écrire de la manière suivante :

- $Empower(departement_informatique, Pierre, administrateur)$

Comme nous l'avons précédemment signalé, un sujet peut également être une organisation. Par exemple l'organisation "Wanadoo" joue le rôle de "fournisseur d'accès" dans l'organisation "France Télécom".

Par ailleurs, des permissions sont accordées aux rôles. Les sujets obtiennent alors les permissions accordées aux rôles qu'ils jouent, sachant qu'un sujet peut jouer plusieurs rôles. Ainsi, comme dans tous les modèles basés sur les rôles, les rôles sont une interface entre l'ensemble des utilisateurs et l'ensemble des permissions.

Enfin, il est possible de hiérarchiser l'ensemble des rôles définis dans une organisation. Un mécanisme d'héritage permet alors d'accorder toutes les permissions d'un rôle à ses sous-rôles. Nous définissons le prédicat *sub_role* (c.f. section A.1) pour modéliser la hiérarchie de rôles. La règle de dérivation RH_1 liée à l'héritage est donnée dans la section A.3.

La définition de rôles, l'affectation de rôles aux sujets et l'héritage des permissions à travers la hiérarchie de rôles ont pour objectif de structurer l'ensemble des sujets d'une organisation et de simplifier ainsi la gestion de la politique de sécurité.

B.2.3 Les objets et les vues

Dans notre modèle, l'entité *Object* représente principalement les entités non actives, en d'autres termes, toutes les ressources de l'organisation, comme les fichiers, les courriers électroniques, les formulaires imprimés, etc. Comme nous venons de le voir, les rôles nous permettent de structurer les sujets et de faciliter la mise à jour de la politique de sécurité. Dans la mesure où il est également nécessaire de structurer les objets et d'ajouter de nouveaux objets au système, une entité comparable au rôle pour les sujets est nécessaire pour les objets. Nous l'appelons entité *View*. De manière intuitive, une vue correspond, comme dans les bases de données relationnelles, à un ensemble d'objets qui satisfont une propriété commune. Prenons l'exemple des fichiers clients d'un fournisseur d'accès. Chaque organisation peut choisir la manière dont ces fichiers sont implantés. Ces informations peuvent être gérées

par des fichiers papier ou stockées dans une base de données. L'organisation peut ainsi avoir à manipuler des objets de nature diverse. Dans ce cas, nous créons une vue "fichier clients". Cette vue regroupe l'ensemble des objets correspondants aux fichiers clients quelle que soit leur nature. Une vue est donc une abstraction d'un ensemble d'objets. Dans la mesure où les vues caractérisent la manière dont les objets sont utilisés dans l'organisation, nous avons besoin d'une relation qui lie ces trois entités : la relation Use. Nous pourrons alors écrire qu'une certaine organisation "FAI" (Fournisseur d'Accès Internet) utilise un certain fichier "fichier245.xls" dans la vue "fichier clients" :

- $Use(FAI, fichier245.xls, fichier_client)$

Enfin, comme pour les rôles, il est possible de définir des hiérarchies de vues, et ceci à l'aide du prédicat *sub_view* (c.f. sections A.1 et A.3).

B.2.4 Les actions et les activités

Les politiques de sécurité spécifient les actions que les entités actives (sujets) ont la permission ou l'interdiction de réaliser sur les entités passives (objets). Dans notre modèle, l'entité Action englobe principalement les actions informatiques comme "lire", "écrire", "envoyer", etc. De la même manière que les rôles et les vues sont des abstractions des sujets et des objets, nous définissons une nouvelle entité utilisée comme abstraction des actions : l'entité *Activity*. Ainsi, les rôles associent des sujets qui remplissent les mêmes fonctions, les vues regroupent des objets qui satisfont une propriété commune, et les activités correspondent à des actions qui ont un même objectif pour la politique de sécurité. Nous pouvons par exemple définir l'activité "consulter". L'action "acroread" (utilisation d'Acrobat Reader) pourra être considérée par une certaine organisation comme implantant l'activité "consulter". Dans la mesure où différentes organisations peuvent considérer qu'une même action est employée pour réaliser différentes activités, la relation *Consider* sera utilisée pour associer les entités *Organization*, *Action* et *Activity*. Nous pourrons alors écrire une relation du type :

- $Consider(departement_informatique, acroread, consulter)$

Comme pour les rôles et les vues, le modèle Or-BAC offre la possibilité de définir une hiérarchie d'activités, en utilisant le prédicat *sub_activity* (c.f. sections A.1 et A.3).

B.2.5 Une politique de sécurité à deux niveaux

Nous venons de voir que les sujets, les actions et les objets sont respectivement abstraits en rôles, en activités et en vues, comme le représente la figure B.1. Nous obtenons alors une politique de sécurité à deux niveaux. Le modèle Or-BAC permet ainsi d'établir une politique de sécurité organisationnelle (rôle, activité, vue) indépendante des choix d'implémentation (sujet, action, objet).

Nous devons à présent définir les prédicats correspondant aux permissions. Soit respectivement Org, S, A, O, $\mathcal{R}$, $\mathcal{A}$ et $\mathcal{V}$ l'ensemble des organisations, des sujets, des actions, des objets, des rôles, des activités et des vues. Afin de représenter les règles de contrôle d'accès au niveau concret, nous introduisons le prédicat *Is_permitted*. Ce prédicat permet d'exprimer une permission accordée à un sujet de réaliser une action sur un objet :

- $Is_permitted(Subject, Action, Object)$

Ainsi, par exemple, la permission accordée à l'utilisateur Jean de lire le fichier "fiche_client_21.pdf" est exprimée de la manière suivante :

- $Is_permitted(Jean, lire, fiche_client_21.pdf)$

De telles règles de sécurité sont dites concrètes, et sont semblables aux règles de contrôle d'accès obtenues dans les modèles [Harrison et al. 1976, Lampson 1971, DAC 1987]. Une politique de sécurité concrète dans le modèle Or-BAC est un ensemble d'autorisations de cette forme.

Nous appelons *Permission* la relation entre un rôle, une activité et une vue. L'organisation dans laquelle une permission est valide est aussi indiquée dans cette relation :

- $Permission(Organisation, Role, Activity, View)$

Cette relation signifie que l'organisation donne la permission à un rôle de réaliser une activité sur une vue. Une telle permission est dite organisationnelle. L'objectif dans le modèle Or-BAC est de rédiger la politique de sécurité à l'aide de permissions organisationnelles. Les permissions concrètes sont alors dérivées des permissions abstraites. La règle de dérivation RG_1 est la suivante :

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V},$
  $Permission(org, r, v, a, c) \wedge$
  $Empower(org, s, r) \wedge$
  $Consider(org, \alpha, a) \wedge$
  $Use(org, o, v) \wedge$
  $\rightarrow Is_permitted(s, \alpha, o)$

Pour illustrer cette règle, considérons l'exemple suivant. L'utilisateur "Jean" désire ouvrir le fichier "fiche_client_21.pdf" à l'aide d'Acrobat Reader. Le contrôle d'accès associé à cette requête correspond à la permission suivante : $Is_permitted(Jean, acroread, fiche_client_21.pdf)$.

Nous dirons alors que si nous avons la permission organisationnelle $Permission(departement_informatique, administrateur, consulter, fiche_client)$, et que l'organisation "département informatique" habilite Jean dans le rôle "administrateur", que cette organisation considère l'action "acroread" comme une activité "consulter", et que cette organisation utilise l'objet "fiche_client_21.pdf" dans la vue "fiche_client", alors Jean obtient l'accès demandé. Si l'on considère à nouveau la figure de la page 12 de ce chapitre, il nous reste, pour être complet, à introduire une entité très importante, l'entité *Context*, que nous détaillons dans la section B.3.

Comme dans de nombreux modèles, il est possible de spécifier des contraintes sur les différentes entités et prédicats définis. Les contraintes dans le modèle Or-BAC sont des règles qui concluent sur le prédicat $error()$. Dans la section A.4 figure l'ensemble des contraintes prédéfinies et nécessaires à la rédaction d'une politique de sécurité Or-BAC. D'autres contraintes peuvent être définies par le concepteur de la politique.

B.2.6 Conclusion

Le modèle Or-BAC présente l'originalité de proposer une abstraction des sujets en rôles, mais également des objets en vues et des actions en activités. Nous obtenons alors une politique de sécurité à deux niveaux. Le niveau organisationnel permet de définir un règlement de sécurité et de modéliser la structure d'une organisation, grâce notamment aux hiérarchies. La politique de sécurité concrète qui est effectivement mise en œuvre au niveau du système d'information est déduite de la politique organisationnelle. Ceci nous assure une totale indépendance de la politique de sécurité vis à vis de son implémentation. Ainsi, la politique organisationnelle reste inchangée, et garde alors toutes ses propriétés lorsque des sujets, des actions ou des objets sont ajoutés ou supprimés du système.

B.3 La modélisation des contextes

B.3.1 Motivation

Il est courant de distinguer les modèles de contrôle d'accès dynamiques des modèles de contrôle d'accès statiques. Dans les modèles passifs, le règlement est spécifié à travers la définition de faits, comme ceux exprimés dans la section précédente. Ces faits sont des autorisations permanentes qui sont ajoutées, supprimées et surtout modifiées par l'administrateur de sécurité uniquement. Or, il est souvent très utile de définir qu'une autorisation n'est valable que dans telle ou telle circonstance, correspondant par exemple à l'heure de la journée, au lieu où se trouve l'utilisateur, à l'état du système d'information, etc. Nous appelons *contexte* l'ensemble des circonstances appliquées à une autorisation. C'est la validation du contexte qui permet alors d'activer et de désactiver les autorisations. Les modèles dits dynamiques offrent la possibilité de spécifier de tels contextes.

Il existe une grande diversité de contextes qu'un administrateur de sécurité pourrait vouloir exprimer. De nombreux modèles actifs sont définis dans la littérature, mais chacun d'entre eux est centrés sur un seul type de contexte. Au contraire, à travers le modèle Or-BAC [Cuppens and Miège 2003c], nous nous attachons à modéliser une large variété de contextes.

L'intégration du contexte dans le modèle Or-BAC oblige à redéfinir quelque peu ce dernier. Nous nous attachons à cette tâche à dans la sous-section suivante. Nous présentons ensuite à la sous-section B.3.3 les différents types de contextes pris en compte dans notre modèle.

B.3.2 L'entité *Context*

Nous introduisons l'entité *Context*, et l'ensemble des contextes $\mathcal{C}$ définis dans une organisation. Les contextes sont utilisés pour spécifier les circonstances concrètes dans lesquelles les organisations accordent aux sujets des permissions de réaliser des actions sur les objets. Par conséquent, les entités *Organization*, *Subject*, *Object*, *Action* et *Context* sont liées par une nouvelle relation appelée *Hold*.

Les contextes sont définis par des règles concluant sur la relation *Hold*. Par exemple, le contexte *heures_travail* dans l'organisation "département informatique" peut être défini de la façon suivante :

- $\forall s \in S, \forall \alpha \in A, \forall o \in O,$
  $Hold(departement_informatique, s, \alpha, o, heures_travail)$
  $\leftarrow 09:00 \leq GLOBAL_CLOCK \leq 19:00$

Afin de prendre en compte le contexte dans les autorisations, nous modifions la forme du prédicat *Permission* qui prend maintenant la forme :

- $Permission(Organisation, Role, Activity, View, Context)$

On peut alors spécifier la règles suivante :

- $Permission(departement_informatique, administrateur,$
  $consulter, fiche_client, heures_travail)$

Cette règle signifie que les sujets jouant le rôle "administrateur" pourront consulter les objets appartenant à la vue "fiches clients" uniquement pendant les heures de travail.

Enfin, la règle de dérivation des permissions concrètes à partir des permissions organisationnelles doit également être modifiée. Elle permet de montrer comment le contexte est vérifié avant d'accorder une autorisation :

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C},$
  $Permission(org, r, v, a, c) \wedge$
  $Empower(org, s, r) \wedge$
  $Consider(org, \alpha, a) \wedge$
  $Use(org, o, v) \wedge$
  $Hold(org, s, \alpha, o, c)$
  $\rightarrow Is_permitted(s, \alpha, o)$

B.3.3 Taxonomie des contextes

La liste suivante présente les contextes que le modèle Or-BAC permet de modéliser :

- Le contexte temporel
- Le contexte spatial
- Le contexte déclaré par l'utilisateur
- Le contexte prérequis
- Le contexte provisionnel

Le contexte temporel permet de contraindre la date et la durée de validité d'une permission. Le contexte spatial correspond au lieu d'où un utilisateur peut effectuer une activité. La localisation d'un utilisateur peut être physique (dans un bâtiment par exemple) ou logique (sur un réseau local par exemple). Le contexte déclaré par l'utilisateur n'est, à notre connaissance, jamais exprimé dans les modèles de contrôle d'accès. Il correspond à un contexte dans lequel l'utilisateur décide de se placer pour effectuer une activité. En effet, dans certains cas seul l'utilisateur est en mesure d'évaluer si un contexte est vrai ou non. C'est le cas par exemple si un administrateur à besoin d'accéder à l'ensemble des informations du système pour réaliser une recherche de virus. L'activation d'un contexte déclaré donnera au sujet des permissions supplémentaires mais, en contrepartie, la politique de sécurité peut spécifier que ce sujet aura des obligations à remplir. Le contexte prérequis correspond aux contraintes

spécifiques au domaine d'application, et enfin, le contexte provisionnel permet d'activer des permissions ou des interdictions en fonction des actions précédemment réalisées.

La décision d'accepter une requête nécessite l'évaluation du contexte. Ceci requiert d'avoir à notre disposition un certain nombre d'informations pour tester l'activation du contexte. La liste suivante décrit l'ensemble des informations que le système doit pouvoir fournir :

- Une horloge interne, appelée GLOBAL_CLOCK, pour évaluer le contexte temporel;
- l'environnement des utilisateurs et des informations relatives à l'architecture logicielle et matérielle, pour évaluer le contexte spatial;
- l'objectif de l'utilisateur pour évaluer le contexte déclaré par l'utilisateur;
- une base d'informations, pour évaluer le contexte prérequis;
- l'historique des actions, pour évaluer le contexte provisionnel.

La figure B.2 récapitule la taxonomie des contextes et les informations nécessaires à leur évaluation.

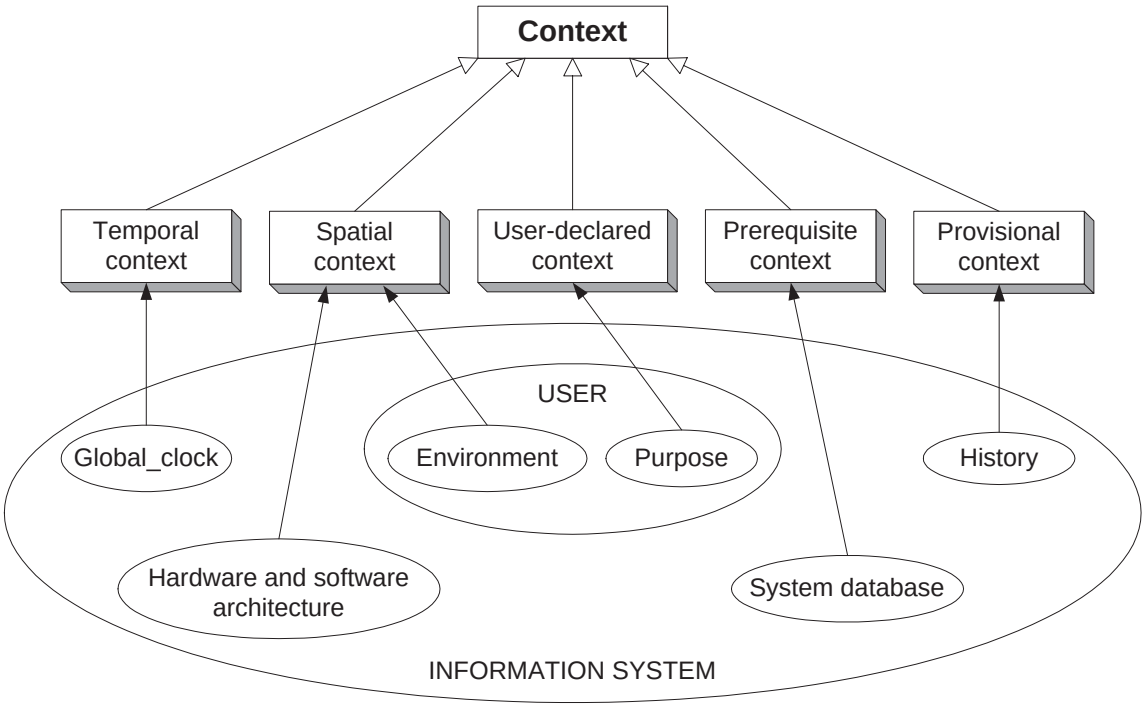


Figure B.2: Taxonomie des contextes et données requises

B.3.4 Conclusion

Nous pouvons, avec le modèle Or-BAC, répondre à une exigence très importante : définir des autorisations dynamiques, en d'autres termes, des autorisations qui sont activées et désactivées en fonction de certaines circonstances. Ces dernières sont modélisées par l'entité *Context*. Contrairement à la plupart des modèles existants, Or-BAC offre la possibilité d'exprimer de nombreux types de contextes : le contexte temporel et spatial, les conditions liées au domaine d'application (prérequis), le contexte provisionnel qui dépend des actions déjà réalisées, et le contexte déclaré par l'utilisateur.

B.4 La gestion des conflits

Jusqu'à présent, nous avons considéré uniquement des permissions, c'est-à-dire des autorisations positives. C'est en fait le cas de la plupart des modèles de contrôle d'accès. On appelle politique permissive, les politiques qui n'intègrent que des autorisations positives. Le modèle Or-BAC permet d'exprimer également des autorisations négatives, appelées également interdictions [Cuppens and Miège 2003b, Cuppens and Miège 2004b]. Une politique de sécurité qui mêle permissions et interdictions est potentiellement conflictuelle. Un conflit apparaît lorsqu'une permission et une interdiction sont appliquées au même triplet $< sujet, action, objet >$. Il est alors nécessaire de définir des méthodes de gestion des conflits. Comme nous le verrons par la suite, il est possible avec Or-BAC de détecter et de gérer les conflits au niveau des autorisations concrètes, mais aussi au niveau des autorisations organisationnelles. Le deuxième cas nous intéresse tout particulièrement et constitue le principal objectif de notre travail sur ce thème. En effet, notre but est de résoudre les conflits au niveau organisationnel, en d'autres termes, d'obtenir la garantie qu'une politique de sécurité organisationnelle n'est pas conflictuelle, et de montrer ensuite que si tel est le cas, alors peu importe les choix d'implémentation, aucun conflit ne pourra apparaître dans la politique de sécurité concrète. Ainsi, une même politique de sécurité organisationnelle pourrait être appliquée à des organisations différentes dans des domaines différents tout en ayant l'assurance qu'aucun conflit n'est possible. De plus, le modèle Or-BAC permet de spécifier une *politique de gestion des conflits* paramétrable par l'administrateur de sécurité.

Nous voyons dans un premier temps à quels objectifs répondent les interdictions et comment elles s'expriment dans notre modèle. Puis, nous expliquons dans la section B.4.2 l'approche générale pour la gestion des conflits. La définition d'une politique de gestion des conflits est abordée dans la section B.4.3. Enfin, nous donnons les grandes lignes de la prévention des conflits dans la dernière section.

B.4.1 Expression des interdictions

Dans le cas de politiques de sécurité simples, l'emploi des autorisations négatives n'est pas nécessaire. En effet, on suppose alors que la politique de sécurité est fermée (*closed policy*), autrement dit, que tout ce que n'est pas permis est interdit. Néanmoins, nous considérons que les interdictions sont essentielles pour rédiger des politiques de sécurité à la fois claires et expressives. Tout d'abord, l'emploi de permissions et d'interdictions est assez naturel pour exprimer un règlement de sécurité. Deuxièmement, dans une politique comportant un très grand nombre de règles, il est intéressant de pouvoir visualiser immédiatement les interdictions plutôt que de les déduire des permissions. De plus, les interdictions peuvent être utilisées comme des exceptions dans une politique de sécurité globalement permissive. En particulier, dans le cas de l'héritage des permissions, il est possible de se servir des interdictions pour arrêter la propagation de certaines permissions. Enfin, de nombreux systèmes supportent les interdictions.

Nous ajoutons au modèle Or-BAC deux nouveaux prédicats : l'interdiction organisationnelle $Prohibition$ et l'interdiction concrète $Is_prohibited$ telles que :

- $Prohibition(Organisation, Role, Activity, View, Context)$
- $Is_prohibited(Subject, Action, Object)$

Comme pour les permissions, les interdictions concrètes sont déduites des interdictions organisationnelles grâce à une règle de dérivation RG_2. Cette règle est obtenue en remplaçant dans RG_1 les permissions par des interdictions (c.f. section A.3). Enfin, il est également possible de définir un héritage des interdictions.

B.4.2 Approche générale

La détection des conflits effectifs ne peut se faire qu'au niveau concret, c'est-à-dire entre les autorisations $Is_permitted$ et $Is_prohibited$. Un conflit existe s'il existe un sujet s, une action α et objet o tels que :

- $Is_permitted(s, \alpha, o) \wedge Is_prohibited(s, \alpha, o)$

Néanmoins, nous considérons cette approche insuffisante pour deux raisons. D'abord car il n'est pas possible de donner la priorité à l'une ou l'autre des autorisations sur des critères choisis. On peut tout juste spécifier que les interdictions l'emportent sur les permissions ou inversement. Pour cette raison nous associons aux autorisations des niveaux de priorité. De plus, dans la mesure où les autorisations concrètes varient en fonction des éléments concrets du système, les conflits ne sont détectés qu'au moment de leur occurrence. Ainsi, nous associons les niveaux de priorité aux autorisations organisationnelles afin de prévenir les conflits avant la dérivation des autorisations concrètes.

Nous ajoutons de nouveaux types d'autorisation afin d'introduire les niveaux de priorité. Nous appelons $Permission'$ et $Prohibition'$ les nouvelles autorisations organisationnelles et $Is_permitted'$ et $Is_prohibited'$ les nouvelles autorisations concrètes. Elles sont de la forme :

- $Permission'(Organisation, Role, Activity, View, Context, Level)$
- $Is_permitted'(Subject, Action, Object, Level)$

La détermination des niveaux de priorité permet de définir une stratégie de gestion des conflits. Nous expliquons ce point dans la section B.4.3. La relation entre les anciennes et les nouvelles autorisations, ainsi que la démarche générale de la gestion des conflits dans le modèle Or-BAC sont présentées dans la figure B.3. Le schéma est équivalent pour les interdictions. La dérivation des autorisations concrètes sans la gestion des conflits est représentée par la flèche en pointillés. Les flèches pleines indiquent le nouveau chemin.

Notre approche est constituée des trois étapes suivantes :

- Définition d'une politique de gestion des conflits (cms) afin de déterminer les autorisations primées.
- Dérivation des autorisations concrètes primées (règle RG_1).
- Dérivation explicite des autorisations concrètes effectives (règle ED_1).

Le principe est le suivant : la politique de gestion des conflits doit permettre d'éviter tout ou partie des conflits. Il est ensuite possible de prévenir les conflits résiduels entre les autorisations organisationnelles primées. Ceci est expliqué dans la section B.4.4.

Etudions tout d'abord les deux dernières étapes. La règles RG'$_1$ permet de dériver les permissions concrètes primées. Cette règle est présentée dans la section A.3. Les permissions concrètes primées sont obtenues en appliquant la règle RG'$_1$ de la même manière qu'avec

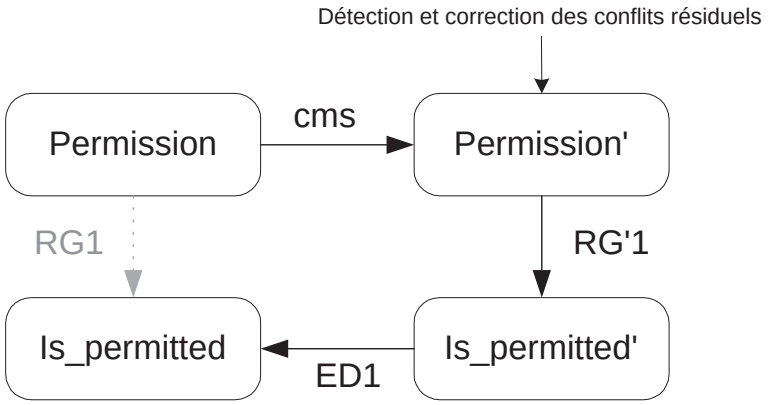


Figure B.3: La gestion des conflits dans le modèle Or-BAC

la règle RG_1 mais en conservant le niveau de priorité. La règle RG'$_2$ est équivalente à RG_1 pour les interdictions.

Finalement la règle ED_1 permet de dériver les permissions concrètes effectives :

- ED_1: $\forall s \in S, \forall \alpha \in A, \forall o \in O, \forall l_1 \in \mathcal{L}$,
  $Is_permitted'(s, \alpha, o, l_1) \wedge$
  $\neg \exists l_2 \in \mathcal{L}, (l_1 \prec l_2 \wedge Is_prohibited'(s, \alpha, o, l_2))$
  $\rightarrow Is_permitted(s, \alpha, o)$

Cette règle signifie qu'il est possible de dériver une permission concrète à partir d'une permission concrète primée s'il n'existe pas d'interdiction concrète primée ayant un niveau de priorité supérieur, et qui soit appliquée au même triplet sujet, action, objet. La règle ED_2 est équivalente à ED_1 pour les interdictions concrètes.

B.4.3 Politique de gestion des conflits

Le modèle Or-BAC offre la possibilité à l'administrateur de sécurité de définir sa propre politique de gestion des conflits. Celle-ci permet de déterminer comment doivent être résolu les conflits. Afin d'offrir une grande flexibilité nous avons introduit des niveaux de priorité dans les autorisations. C'est en s'appuyant sur ces niveaux que la *cms* est définie. En pratique, créer une politique de gestion des conflits consiste à définir un ensemble $\mathcal{L}$ de niveaux de priorités associé à un ordre partiel et à déterminer comment sont obtenus les autorisations primées.

Considérons les deux exemples suivants :

cms1 :

- $\mathcal{L} = \{0, 1\}$
- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Permission(org, r, a, v, c) \rightarrow Permission'(org, r, a, v, c, 0)$
- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Prohibition(org, r, a, v, c) \rightarrow Prohibition'(org, r, a, v, c, 1)$

cms2 :

- $\mathcal{L} = \mathcal{R}$ et l'ensemble de rôles $\mathcal{R}$ est associé à un ordre partiel.
- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Permission(org, r, a, v, c) \rightarrow Permission'(org, r, a, v, c, r)$
- $\forall org \in Org, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall c \in \mathcal{C}$,
  $Prohibition(org, r, a, v, c) \rightarrow Prohibition'(org, r, a, v, c, r)$

La première politique signifie que l'interdiction l'emporte toujours sur la permission. Dans la seconde politique, le niveau de priorité d'une autorisation est le rôle impliqué dans cette autorisation. Ainsi, dans le cas d'une hiérarchie de rôle, c'est l'autorisation associée au rôle de plus haut niveau dans la hiérarchie qui sera privilégié. Ces deux exemples sont assez simples, mais cette méthode permet de définir des politiques plus complexes. Une politique de gestion des conflits doit permettre de dériver les autorisations organisationnelles primes automatiquement, afin d'alléger le travail de l'administrateur de sécurité. Ce dernier peut néanmoins affecter manuellement un niveau de priorité autorisation par autorisation.

Il faut distinguer deux types de politiques de gestion des conflits. Si la politique assure qu'il n'y aura pas de conflit, elle est dite efficace ($effective$) sinon elle est dite faible ($weak$). Ainsi, $cms1$ est une politique efficace, alors que $cms2$ est faible. En effet, dans ce dernier cas, un conflit entre une permission et une interdiction portant sur le même rôle n'est pas résolu.

B.4.4 Prévention de conflits

Si la politique est efficace, alors la gestion des conflits est terminée. En revanche, si elle est faible, des conflits peuvent encore apparaître. Ces conflits pourront être détectés au niveau des autorisations concrètes. Ceci n'est pas satisfaisant pour deux raisons. En premier lieu, une telle politique de sécurité n'est pas stable dans la mesure où une mise à jour au niveau concret, comme par exemple l'affectation d'un nouvel utilisateur dans un rôle, peut entraîner l'apparition de nouveaux conflits. En second lieu, la gestion des conflits au niveau des autorisations concrètes est en contradiction avec l'esprit du modèle Or-BAC. En effet, Or-BAC est réalisé pour permettre la gestion de la politique de sécurité à un haut niveau d'abstraction, et ceci afin de s'affranchir des choix d'implémentation. Ainsi, il serait plus dans l'esprit du modèle Or-BAC de gérer les conflits entre les permissions et les interdictions au niveau abstrait, autrement dit, entre les faits $Permission'$ et $Prohibition'$. L'objectif étant dans ce cas de pouvoir certifier que s'il n'existe pas de conflits au niveau des autorisations organisationnelles, alors il n'y aura pas de conflits entre les autorisations concrètes, et ce, quels que soient les sujets, les actions et les objets.

Ceci est réalisé par la détermination d'une condition particulière. Si cette conditions est satisfaite pour une permission et une interdiction organisationnelles, alors aucun conflit ne pourra apparaître au niveau concret. L'apparition effective d'un conflit dépend des sujets, des actions et des objets affectés dans les entités organisationnelles de la permission et de l'interdiction. Une telle méthode présente des contreparties. En effet, cette condition permet de prévenir la potentialité d'un conflit. Ainsi, si elle est trop forte, son respect restreint de manière trop importante la rédaction de la politique de sécurité. Par conséquent, nous nous sommes attachés dans cette thèse à déterminer et à affiner une condition qui soit la plus faible possible.

B.4.5 Conclusion

Ces travaux ont permis d'enrichir le modèle Or-BAC de l'expression des interdictions. De plus, nous avons défini une méthode de gestion des conflits. Cette méthode présente deux avantages considérables : elle permet de spécifier une politique de gestion des conflits, paramétrable par l'administrateur de sécurité, ce qui offre une grande souplesse. Nous définissons également une méthode de prévention des conflits qui permet de garantir l'absence de tout conflit dans une politique de sécurité, et ceci quels que soient les choix d'implémentation.

B.5 Le modèle administratif

B.5.1 Introduction

Au sens large, l'administration d'une politique de contrôle d'accès consiste à créer et mettre à jour l'ensemble de cette politique. Prévoir des procédures administratives permettant de mettre à jour la politique de sécurité est tout à fait essentiel. En effet, une fois créée, la politique perd de sa pertinence au fur et à mesure des modifications apportées au système d'information. Ainsi, la politique doit évoluer en même temps que celui-ci, afin de toujours rester en adéquation avec les exigences initiales. Dans le cas d'une politique Or-BAC, il s'agit par exemple d'ajouter (ou de modifier) des entités organisationnelles (organisations, rôles, activités, etc), de les structurer en hiérarchies, d'affecter des entités concrètes et de spécifier des autorisations, etc.

L'administration d'une politique peut être plus ou moins décentralisée. En effet, la gestion de la politique peut être laissée à la charge d'un unique administrateur de sécurité. Elle peut aussi être distribuée aux différents responsables de département d'une organisation par exemple. La délégation est un autre aspect important de l'administration, et n'est que très peu prise en compte dans les modèles fondés sur les rôles.

Nous présentons ici l'administration dans le modèle Or-BAC, et montrons en quoi elle est suffisamment flexible pour permettre de résoudre ces difficultés. Nous avons défini un modèle dédié à l'administration, appelé AdOr-BAC [Cuppens and Miège 2004a], pour *Administration model for Or-BAC*. Il comporte entre autres les composants suivants :

- PRA (Permission-Role Assignment) : création et suppression de permissions.
- URA (User-Role Assignment) : habilitation des sujets dans des rôles.
- UPA (User-Permission Assignment) : affectation de permissions à des utilisateurs (Délégation).
- RHA (Role-Hierarchy Assignment) : création de liens hiérarchiques entre les rôles.

Un des objectifs du modèle AdOr-BAC est de permettre d'administrer une politique de sécurité Or-BAC en utilisant les entités et les prédicats déjà décrits. En effet, il nous paraît important qu'un modèle de sécurité soit auto-administré, autrement dit qu'il ne soit pas nécessaire d'introduire de nouveaux concepts. Ainsi, les permissions qui autorisent un rôle à mettre à jour la politique de sécurité doivent avoir la même forme que les permissions standards. Nous ne rentrerons pas ici dans le détail de tous les composants cités précédemment. En effet, le principe reste le même d'un composant à un autre. Ainsi, nous nous concentrons

uniquement sur PRA. En revanche nous reviendrons sur la délégation rendue possible avec le composant UPA.

B.5.2 PRA

Le composant PRA a pour objectif de créer et de supprimer des autorisations organisationnelles "administratives". Ces autorisations permettent de spécifier les rôles qui ont la permission de créer et de supprimer des autorisations (positives ou négatives). Dans PRA, comme dans les autres composants, l'administration du modèle Or-BAC consiste à distribuer des *capacités*[4]. Commençons notre explication par une comparaison, celle du billet d'avion.

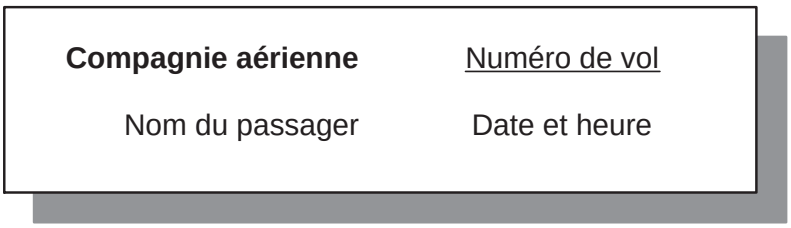


Figure B.4: Exemple du billet d'avion

La permission de monter dans un avion se traduit par la possession d'un billet d'avion correspondant à un certain vol (figure B.4). Une agence de voyage a la permission de délivrer des billets d'avion ; elle a le droit de délivrer des permissions de prendre des avions. La permission créée par l'agence de voyage se traduit par la création d'un objet (le billet).

De la même manière, nous considérons dans Or-BAC que la création d'une autorisation correspond à la création d'un objet au sens d'entité passive du contrôle d'accès. Dans la mesure où nous désirons qu'AdOr-BAC soit totalement compatible avec Or-BAC, cet objet est inséré dans une vue créée à cet effet, la vue PRA. Ainsi, nous considérons que la création d'une autorisation est équivalente à l'ajout d'un objet dans cette vue.

Le billet d'avion comporte plusieurs attributs : le nom de la compagnie aérienne qui reconnaîtra le droit de voler, le nom du passager qui bénéficie de la permission de prendre l'avion, le numéro du vol, la date et l'heure qui permettent d'identifier sur quel vol s'applique la permission, etc.

De la même manière, les objets de la vue PRA doivent comporter plusieurs attributs :

- *issuer*: désigne l'organisation qui reconnaît l'autorisation.
- *grantee*, *privilege*, *target*: désignent le rôle, l'activité et la vue concernés par l'autorisation.
- *context*: correspond au contexte dans lequel l'autorisation sera activée.
- *type*: indique le type de l'autorisation. *type* est dans $\{positive, negative\}$.

L'équivalence entre la création d'une permission par exemple et l'ajout d'un objet dans la vue PRA est exprimée selon la règle suivante :

[4]Le terme capacité renvoie au mode d'implémentation d'une matrice de contrôle d'accès qui, contrairement aux listes de contrôle d'accès (ACL), consiste à attacher les accès autorisés aux sujets plutôt qu'aux objets.

- $\forall org_a \in Org, \forall pra \in O,$
  $Use(org_a, pra, PRA) \wedge$
  $issuer(pra, org_b) \wedge grantee(pra, r) \wedge privilege(pra, a) \wedge$
  $target(pra, v) \wedge context(pra, c) \wedge type(pra, positive)$
  $\rightarrow Permission(org_b, r, a, v, c)$

En d'autres termes, accorder la permission à un rôle de créer une autorisation consiste à donner la permission à ce rôle d'ajouter un objet à la vue PRA. Lorsqu'un rôle r n'a pas la permission de créer n'importe quelle permission, il faudra créer des sous-vues de PRA adaptées et n'autoriser r qu'à insérer des objets dans ces vues particulières. On peut remarquer que l'organisation dans laquelle est émise l'autorisation n'est pas nécessairement celle dans laquelle l'autorisation sera valable.

Considérons l'exemple suivant où nous désirons permettre à un administrateur *admin* le droit d'accorder la permission suivante :

- $Permission(departement_informatique, operateur,$
  $consulter, fiche_client, heures_travail)$

On définit alors la sous-vue $PRA_operateur$ de la manière suivante :

- $\forall pra \in O,$
  $Use(departement_informatique, pra, PRA_operateur)$
  $\leftarrow Use(departement_informatique, pra, PRA) \wedge$
  $grantee(pra, operateur) \wedge$
  $issuer(pra, departement_informatique) \wedge privilege(pra, consulter) \wedge$
  $target(pra, fiche_client) \wedge context(pra, heures_travail) \wedge type(pra, positive)$

Il suffit ensuite de donner la permission d'ajouter des objets dans cette vue. Ceci est réalisé avec la permission suivante:

- $Permission(departement_informatique, admin, assign, PRA_operateur, default)$

Nous pouvons ainsi administrer les autorisations en utilisant les entités et les prédicats déjà définis dans le modèle Or-BAC.

De même, l'affectation d'un sujet à un rôle se fait par l'insertion d'un objet dans une vue, appelée URA. Ce principe est utilisé pour toute l'administration du modèle Or-BAC. En distribuant les permissions administratives adéquates, il est alors possible de définir finement le mode d'administration (plus ou moins décentralisé).

B.5.3 Délégation

Le composant UPA est particulièrement intéressant car il permet de mettre en œuvre des mécanismes de délégation. Cette tâche n'est pas aisée. En effet, tous les modèles fondés sur les rôles souffrent de l'impossibilité d'affecter des permissions directement aux utilisateurs. Dans de tels modèles, Or-BAC en fait partie, les utilisateurs obtiennent des permissions en fonction du ou des rôles qu'ils jouent. Il est donc impossible d'exprimer par exemple que le rôle "administrateur" peut déléguer, quand il est absent, une partie de ses prérogatives à un utilisateur particulier, et non à un rôle. S'il délègue une permission au rôle chef de département par exemple, tous les utilisateurs jouant ce rôle recevront la permission, ce qui

n'est pas souhaitable. Pour ce faire, nous créons une vue UPA qui permet ainsi à un rôle d'accorder des autorisations à un sujet particulier. Cette vue comporte les mêmes attributs que la vue PRA mais l'attribut *grantee* désigne cette fois-ci un utilisateur et non plus un rôle.

B.5.4 Conclusion

Nous avons associé au modèle Or-BAC un modèle d'administration qui permet pour l'heure de gérer les principaux aspects d'une politique de sécurité. Chaque tâche administrative est réalisée en ajoutant ou supprimant des objets dans des vues. Ceci offre un double avantage : le mode de répartition des tâches administratives devient très flexible, et le modèle administratif est alors totalement compatible avec le modèle Or-BAC. Les permissions administratives peuvent alors être gérées par le même système informatique.

B.6 Mise en œuvre

La thèse a également été l'occasion de mettre en œuvre le modèle Or-BAC. Nous évoquons ici deux réalisations. La première consiste en l'application du modèle Or-BAC dans un environnement réseau, la seconde au développement d'une maquette permettant de saisir une politique Or-BAC et d'appliquer la méthode de gestion des conflits.

B.6.1 Application à un environnement réseau

Ce travail [Cuppens et al. 2004a, Cuppens et al. 2004b] consiste à appliquer le modèle Or-BAC à un environnement réseau avec comme objectif de proposer une méthodologie de haut niveau pour rédiger des politiques de sécurité réseau. En effet, nous sommes parti du constat que pour la configuration des firewalls il n'existe que des outils et des langages spécifiques à chaque matériel.

Nous avons donc interprété les différents concepts de notre modèle dans un environnement réseau. Ainsi, dans la mesure où une organisation est définie comme une entité qui gère un règlement de sécurité nous pouvons considérer les firewalls d'un réseau comme des organisations. Les ordinateurs sont vus comme des sujets qui jouent des rôles (serveur Web, serveur DNS, machine utilisateur, etc), les services sont comparables à des activités et les hôtes cibles des flux d'information sont considérés comme des vues. Ceci nous a permis d'écrire une politique de sécurité réseau à l'aide du modèle Or-BAC.

Cette réutilisation du modèle offre plusieurs avantages. Entre autres, le nombre de règles est diminué grâce à la définition de rôles. De plus, il est possible de définir des hiérarchies de rôles et d'activités et ainsi de mettre en place des mécanismes d'héritage permettant de simplifier la gestion de la politique.

Nous avons ensuite défini, par transformation XSL (XSLT), une méthodologie permettant de traduire automatiquement la politique de sécurité d'un réseau en script de configuration pour les différents firewalls qui le composent. Nous nous sommes concentrés pour le moment sur le firewall Netfilter. Nous avons choisi ce firewall en particulier car le système de branchement (*jump*) qu'il offre permet de rédiger une politique en n'utilisant que des permissions et

ainsi de s'affranchir de l'ordre des règles de sécurité. L'étape consistant à ordonner les règles positives (*permit*) et négatives (*deny*) d'un firewall n'est alors plus nécessaire. Nous projetons de créér d'autres XSLT pour dériver les scripts de configuration d'autres firewalls.

B.6.2 OToKit

Nous avons développé une maquette appelée OToKit (*Or-BAC ToolKit*) adaptée au modèle Or-BAC. L'objectif consistait à proposer une interface conviviale permettant de saisir et de consulter une politique de sécurité Or-BAC et également de détecter et résoudre les conflits.

Le programme est constitué de deux parties. La politique de sécurité est stockée sous forme de faits et de règles Prolog. Prolog permet également d'implémenter toutes les règles de dérivation liées à l'héritage, aux contraintes et à la gestion des conflits. L'interface graphique, dont une impression écran se trouve à la page 144, est réalisée en Java.

OToKit permet ainsi de saisir et de visualiser l'ensemble des éléments qui constituent une organisation (rôles, activités, vues, contextes) et également de saisir des autorisations positives et négatives associées aux différents éléments de la politique de sécurité. Grâce à notre maquette, nous pouvons mettre en évidence tous les bénéfices de l'héritage mais aussi les effets de bord qui peuvent apparaître. OToKit permet également de gérer les conflits en associant des niveaux de priorités aux autorisations et en détectant, soit les conflits potentiels, soit les conflits effectifs.

D'autres développements sont en cours comme l'intégration des procédures administratives. Ceci doit permettre d'administrer une politique de sécurité Or-BAC, en d'autres termes, d'administrer l'accès même à OToKit.

B.7 Perspectives

De nombreux travaux et études sont encore à réaliser sur le modèle Or-BAC. Nous envisageons dans un premier temps d'approfondir la notion d'activité et la sémantique de la hiérarchie d'activités. L'objectif est de raffiner les activités de haut niveau afin d'offrir la possibilité de modéliser les flux d'activités ($workflow$). L'idée est que la réalisation d'une certaine activité nécessite la réalisation en série ou en parallèle de plusieurs sous-activités constituant autant d'étapes d'un processus global. Une autre piste d'étude consiste à modéliser les $workflows$ non pas par décomposition des activités mais par le développement de contextes provisionnels.

Nous travaillons actuellement à l'incorporation des obligations dans le modèle Or-BAC. Nous pouvons distinguer provisions et obligations. Les provisions sont les actions qui doivent être réalisées pour activer certaines autorisations. Ceci est modélisé par le contexte provisionnel. Les obligations, quant à elles, sont des activités qui doivent être réalisées dans le futur. Ceci implique l'emploi de la logique temporelle, et vise à transformer le modèle Or-BAC non plus en un modèle de contrôle d'accès uniquement mais également en un modèle de contrôle d'usage.

D'autres travaux sont également menés pour adapter Or-BAC au monde des Web-services et en particulier pour définir des politiques de gestion de droits (DRM). Une adaptation à XML a déjà été mise au point.

Concernant la gestion des conflits, nous devons approfondir l'étude des "conflits contextuels", autrement dit, des conflits entre des autorisations présentant des contextes différents. Il s'agit de déterminer le dénominateur commun entre deux contextes afin d'établir dans quelles circonstances un conflit a effectivement lieu.

Enfin, du point de vue de l'implémentation, de nombreuses extensions peuvent être développées autour de l'outil OToKit. En particulier l'intégration du modèle AdOr-BAC, et l'adaptation d'OToKit pour la rédaction de politiques de sécurité réseau sont en cours de réalisation.