Université Saâd DAHLEB de Blida

Faculté des sciences

Département d'Informatique

Mémoire Présenté par :

DAHMANI Maria

TSAMDA Hanane

En vue d'obtenir le diplôme de Master

Domaine : Mathématique et Informatique.

Filière : Informatique.

Spécialité : Ingénierie des Logiciels.

Option : Sécurité d'informatique.

Sujet :

LA GESTION DES CONFLITS DANS LE CONTRÔLE D'ACCES

Devant le jury composé de :

- Président
- Rapporteur
- Examinateur

Sujet proposé par :

Mlle N. BOUSTIA.

Promotion 2012-2013

Remerciements

Cet opuscule serait incomplet si nous ne témoignerons pas de notre profonde gratitude envers le grand **DIEU** qui nous a conduits à réaliser l'un de nos rêves par sa bienveillance et sa générosité. Toutes les expressions de remerciement ne suffiront jamais pour traduire notre reconnaissance et notre profonde gratitude.

Nous exprimons nos remerciements à notre promotrice la demoiselle ***BOUSTIA.Narhimene*** *pour l'assistance qu'elle nous a témoignée, pour sa disponibilité, pour ces orientations et conseils sans lesquels ce travail ne verra pas le jour, qu'elle trouve ici l'expression de notre gratitude.*

Nous remercions les membres du département d'informatique.

Que tous les professeurs qui ont contribué à notre formation trouvent ici notre profonde gratitude

Que les membres de ce distingué jury soient assurés de notre gré pour nous avoir fait l'honneur d'évaluer notre travail.

Nous remercions tous ceux et celles qui ont participé de près ou de loin l'élaboration du présent travail.

Nous tenons à remercier tous nos amis et collègues pour leur soutien moral tout au long de la préparation de ce mémoire.

Spécialement à tous les étudiants de la promotion sortante sans exception.

Dédicace

A qui a consacré sa vie à faire mon bonheur,
et qui a su guider mes pas d'enfance vers ce que je suis devenue aujourd'hui, mon très cher père.

A celle qui m'abreuve d'amour et d'affection intarissable,
source de mon bonheur qui m'a enseigné que l'amour et la patience sont bien la clé du bonheur et de la réussite et ma raison d'être, ma très chère mère.

A mon très cher mari Mohamed. Tes sacrifices, ton soutien moral, ta gentillesse sans égal, ton profond attachement m'ont permis de réussir mes études et a toute sa famille.

A mes adorables grandes parents que Dieu me les garde encore aussi longtemps et A la mémoire de ma très chère grande mère.

A mes oncles, tantes, cousins et cousines surtout Maya...et toute ma famille.

A toutes mes amies, pour les meilleurs moments que j'ai passé avec eux

Surtout ma binôme Maria et toutes sa famille.

Je n'oublierai pas pour autant, tous ceux qui sont absents sur cette feuille
Mais toujours présents dans mon cœur.

HANANE

Dédicace

Je dédie ce mémoire de fin d'études

A

Mon très cher père et ma très chère mère

en témoignage de ma reconnaissance envers le soutien, les sacrifies

et tous les

efforts qu'ils ont fait pour mon éducation ainsi que ma formation

A

Mon adorable grand-mère que Dieu me la garde encore aussi longtemps et

A la mémoire de **ma grand-mère.**

A mes oncles, tantes, cousins, cousines et toute ma famille.

A

Mon chère frères *Hamza*, et ma chère sœur *F.Zohra*

Pour leur affection, compréhension et patience

A

Ma binôme *Hanane* qui a partagé avec moi les moments les plus

durs et les plus beaux de ce voyage et a toutes sa familles

Aux merveilleuses : *Imene, Soumia, Lyna, Hadjer et Khadidja ...*

Je n'oublierai pas pour autant, tous ceux qui sont absents sur cette feuille

mais toujours présents dans mon cœur.

Je dédie ce modeste travail

MARIA

Sommaire

Résumé.

Introduction générale.

Chapitre I: Le modèle OrBAC

I.1.Introduction.....5

I.2. Modèle OrBAC : 5

I.2.1. Objectifs et avantages du modèle OrBAC 5

I.2.2. Eléments du modèle : 6

I.2.2.1.Organisation. 7

I.2.2.2.Sujets et rôles. 7

I.2.2.3.Objets et vues. 7

I.2.2.4.Actions et activités. 8

I.2.3. Notion de contexte. 9

I.2.4. Politique de sécurité. 10

I.2.5.Les autorisations concrètes. 10

I.3. Hiérarchies dans une organisation :.....11

I.3.1. Hiérarchie de rôles. 11

I.3.2. Hiérarchie d'activités. 11

I.3.3. Hiérarchie de vues.....11

I.4. Conclusion. 12

Chapitre II: la logique de description

II.1. Introduction. 14

II.2. Origines de la logique de description : 14

II.2.1. La première génération de logique de description (1980-1990). 14

II.2.2. La deuxième génération de logique de description (1990-Aujourd'hui). 15

II.3. Présentation de la logique de description : 15

II.3.1. Objets de la logique de description. 15

II.3.2. Niveaux de description : 16

II.3.2.1. Niveau terminologique (T-Box). 16
II.3.2.2. Niveau factuel (A-Box). 17
II.3.3.La logique minimale AL: 17
II.3.3.1. Syntaxe du langage AL. 18
II.3.3.2. La sémantique formelle d'AL. 19
II.3.4. Services d'inférence : 19
II.3.4.1. Subsomption : 19
II.3.4.1.1. Subsomption extensionnelle. 20
II.3.4.1.2. Subsomption structurelle (ou intentionnelle). 20
II.3.4.2. Inférences terminologiques : 20
II.3.4.2.1.Classification des concepts. 20
II.3.4.2.2. Complétion. 20
II.3.4.2.3. Héritage. 21
II.3.4.2.4. Détection d'incohérence. 21
II.3.4.3. Inférences assertionnelles : 21
II.3.4.3.1. Reconnaissance d'instance. 21
II.3.4.3.2. Propagation. 21
II.4. la logique de description ALδε : 21
II.4.1.Syntaxes d'ALδε :..... 22
II.4.2. Propriétés des connecteurs δ et ε. 22
II.4.3. Caractéristiques de la logique ALδε : 23
II.4.3.1. Héritage. 23
II.4.3.2. Points forts. 23
II.4.3.3. Points faibles. 23
II.5. Conclusion. 23

Chapitre III: Gestion des conflits dans Or-BAC .

III.1.Introduction : 25
III.2. Types de conflit. 26

III.3.3. Conflit entre obligation et interdiction. ……28
III.4. Détection des conflits. ……28
III.5. Spécification de la théorie logique. ……29
III.5.1. Modèle de base. ……30
III.5.2. Les sujets et les rôles. ……30
III.5.3. Les objets et les vues. ……30
III.5.4. Les actions et les activités. ……31
III.5.5. La hiérarchie. ……31
III.5.6. Contraintes de séparations : ……33
III.5.6.1. Séparation rôle. ……33
III.5.6.2. Séparation vue. ……33
III.2.6.3. Séparation activité. ……34
III.5.7. Spécification des règles de sécurité. ……34
III.6. Conflit dans la théorie. ……35
III.7. Résolution de Conflit. ……37
III.8. Principe de l'approche : ……37
III.8.1. Niveau de priorité : ……38
III.8.1.1. Le concept Permission'. ……39
III.8.1.2 Le concept Interdiction'. ……39
III.8.1.3. Le concept Est-permis'. ……39
III.8.1.4 Le concept Est-interdit'. ……39
III.9. Nouveau processus de dérivation: ……40
III.9.1. Premier étape: Politique de gestion de conflit « cms »……40
III.9.2. Deuxième étape: Dérivation des autorisations concrètes à partir des autorisations abstraites (organisationnelles)……41
III.9.3. Troisième étape: Dérivation des permissions concrètes non primées ED1…42
III.10. Politique de gestion de conflits. ……43
III.11. Prévention de conflits : ……45
III.11.1. Prévention du conflit : Première proposition. ……45
III.11.2. Prévention du conflit: Dernière proposition. ……46
III.11.3. Prévention du conflit: Deuxième proposition. ……46
III.12. Conclusion. ……47

Chapitre IV: Implémentation.

IV.1. Introduction 49

IV.2. Outil de développement : 49

IV.2. Langage de programmation. 49

IV.2.1. Java. 49

IV.2.2.NetBeans 49

IV.3.Présentation de l'application 50

IV.3.1.La fenêtre principale 50

IV.3.2.Enrichissement de la base de connaissance à travers le DL-OrBac..... 51

IV.3.3.Processus de dérivation..... 58

IV.3.4. Gestion de conflit..... 59

IV.4. Conclusion..... 61

Conclusion général.

Bibliographie.

Annexe.

Liste des figures

Figure I.1 : *Le modèle OrBAC*.....6
Figure I.2 : *La relation Habilite*.....7
Figure I.3 : *La relation Utilise*.....8
Figure I.4 : *La relation Considére*.....9
Figure I.5 : *La relation Définit*.....10
Figure II.1 : Structure générale des logiques de description.....16
Figure II.2 : La grammaire des expressions conceptuelles selon AL.....18
Figure II.3 : La sémantique formelle d'AL.....19
Figure III.1 : La gestion des conflits dans Or-BAC.....39
Figure IV.1 : La fenêtre principale du DL_OrBAC..... 50
Figure IV.2 : Menu principal.....50
Figure IV.3: La fenêtre hiérarchie des organisations.....52
Figure IV.4 : La fenêtre hiérarchie de rôle.....53
Figure IV.5 : La fenêtre de séparation de rôles.....53
Figure IV.6 : La fenêtre séparation de vues.....54
Figure IV.7 : La fenêtre séparation d'activité.....54
Figure IV.8 : La fenêtre utilise.....55
Figure IV.9 : La fenêtre habilite.....55
Figure IV.10 : La fenêtre considère.....56
Figure IV.11 : La fenêtre mode abstrait défaut 56
Figure IV.12 : La fenêtre d'ajouter une permission exception 57
Figure IV.13 : La fenêtre mode concret..... 57
Figure IV.14 : La fenêtre processus de dérivation..... 58
Figure IV.15 : La fenêtre gestion des conflits..... 59
Figure IV.16 : La fenêtre solution..... 60
Figure IV.17 : La fenêtre dérivation en mode concret..... 61

Résumé :

Le but de ce travail est de définir une politique de gestion de conflits afin d'augmenter l'efficacité du modèle OrBAC (Organization Based Access Control).

L'idée principale est d'attribuer au modèle lui-même, un ensemble de règles pouvant aider le système dans le cas d'un conflit de trancher en donnant des priorités aux permissions et interdictions.

Afin de rendre la politique de gestion de conflits plus efficace, nous proposons de résoudre le conflit au niveau abstrait, pour qu'ils ne se répercutent pas sur le niveau concret.

Notre politique sera formalisée avec un langage formel, à savoir, la logique de description non monotone, ceci est dû à tous ces avantages tels que l'expressivité, la décidabilité…

Mots clés : Modèle de contrôle d'accès OrBAC, logique de description non monotone.

Abstract :

The aim of this search is to lay down a policy for management conflict, in order to increase the efficiency of OrBAC (Organization Based Access Control).

The basic idea is to award to this OrBAC model, a set of rules bring able to help a system in case of a conflict to settle giving priority to the permissions and prohibitions.

We suggest to resolve the conflict in abstract standard, so that it does not reflect on the concrete standard.

Our policy will be formalized with one formal language, namely, the not monotonous logic of description; this is advantage such as expressivity and the decidability...

Key words: *Access Control Model OrBAC, Description logic DL.*

تلخيص :

الهدف من هذا العمل متمثل في وضع برنامج إدارة التناقضات بغية زيادة فعالية النموذج **OrBAC.**

إن الفكرة الرئيسية هي أن نخصص للنموذج نفسه مجموعة قواعد لكي تساعد النظام في حالة التناقض, أوعلى أخذ القرار بإعطاء أولويات للمسموحات و الممنوعات.

لقد اقترحنا حل التناقضات على الصعيد النظري لكي لا يؤثر سلبا على الصعيد العملي و هذا بهدف جعل برنامج إدارة التناقضات أكثر فعالية .

الكلمات الرئيسية : نموذج التحكّم في طلبات الدخول(OrBAC), منطق الوصف(DL non monotone).

Introduction générale

Introduction générale

Aujourd'hui, de nombreux composant de sécurité sont disponibles. Cependant, pour que ces composants soient efficaces, il convient de définir une politique de sécurité globale du système d'information (SI) à protéger. Dans la grande majorité des cas, cette tache de déploiement est actuellement réalisée de façon artisanale, les administrateurs doivent configurer manuellement les différents composants de sécurité de l'architecture du SI dont ils ont la responsabilité.

Cette méthodologie doit permettre de garantir que pour chaque composant, l'ensemble des règle de configuration généré est à la fois :

- ✓ Valide, dans le sens ou toutes les règles sont nécessaires à la mise en œuvre de la politique.
- ✓ Complet, dans le sens ou la réunion des règles doit permettre de réaliser les activités autorisée.
- ✓ Globalement cohérent.

Ces règles positives sont commandes pour les systèmes simples. Cependant, les règles négatives doivent être explicitement définies et intégrée dans un modèle de contrôle d'accès. Le modèle OrBAC nous donne cette opportunité.

L'objectif d' **OrBAC (Organization Based Access Control)** est de permettre la modélisation d'une variété de politiques de sécurité basées sur le contexte de l'organisation. Pour arriver à ce but, et afin de réduire la complexité de gestion des droits d'accès, le modèle OrBAC repose sur quatre grands principes : déroule

- ➢ L'organisation est l'entité centrale du modèle
- ➢ Il y a deux niveaux d'abstraction :
  - Un niveau concret : sujet, action, objet
  - Un niveau abstrait : rôle, activité, vue
- ➢ La possibilité d'exprimer des permissions, des interdictions, et des obligations.
- ➢ La possibilité d'exprimer les contextes.

OrBAC prend en compte les contextes, les hiérarchies et la délégation.

OrBAC permet d'exprimer aussi bien des permissions, que des interdictions, que des obligations. Il permet donc de spécifier une politique mixte.

La politique mixte pose de nombreux problèmes liés à la gestion des conflits potentiels et des règles redondantes.
Par exemple, si un même utilisateur possède deux rôles et que l'un de ces rôles lui permet de faire une activité et l'autre lui interdit. On est sûr qu'il y aura conflit.

Pour résoudre les conflits, on ajoute un certain nombre de règle au modèle lui-même, permettant de trancher en cas de conflit.

Le But de ce travail est de proposer des règles de gestion de conflits dans le modèle OrBAC en utilisant une logique de description non monotone. On propose aussi de résoudre le conflit au niveau abstrait, pour qu'ils ne se répercutent pas sur le niveau concret. On décide pour cela de donner des priorités aux interdictions et permissions du niveau abstrait.

Le plan du mémoire s'articule autour d'une introduction générale qui définit le contexte de cette étude, d'un premier chapitre dans lequel nous définissons le modèle de contrôle d'accès Or-BAC, d'un deuxième chapitre consacré à la compréhension de la logique de description (DL) et particulièrement les DLs non monotones, Proposition d'une extension du modèle OrBac avec le module de gestion de conflits dans le troisième chapitre et d'un quatrième chapitre pour illustrer le fonctionnement de notre modèle Or-BAC et finalement d'une conclusion générale.

Chapitre I

Le modèle OrBAC

I.1.Introduction :

La sécurité d'un système d'information est assurée via l'application d'un ensemble de règles et de recommandations s'appliquant à divers niveaux tant physiques que logiques.

Une des approches possibles est d'établir un contrôle d'accès qui régit les autorisations d'accès de sujets (des entités du système) à des objets (les ressources du système). Divers modèles ont été proposés comme les modèles d'accès discrétionnaires et les modèles de flux d'information, jusqu'au modèle RBAC **[1]** et ses dérivés. Un problème classique est alors d'assurer que le système est conforme à la politique exprimée.

Les ITSEC **[2]** définissent une politique de sécurité comme étant «l'ensemble des lois, règles et pratiques qui régissent la façon dont l'information sensible et les autres ressources sont gérées, protégées et distribuées à l'intérieur d'un système spécifique ». D'une manière générale, les règles de sécurité sont souvent spécifiées en termes de permissions et d'interdictions, mais aussi en termes d'obligation.

I.2. Modèle OrBAC :

OrBAC (*Organization Based Access Control*) est un modèle de politique de sécurité. Il est issu des travaux réalisés dans le cadre du projet RNRT MP6 en France **[3]** (Modèles et Politiques de Sécurité des Systèmes d'Informations et de Communication en Santé et en Social).

Les différents éléments qui composent ce modèle sont : l'*organisation*, les *sujets* et les *rôles*, les *objets* et les *vues* et enfin les *actions* et les *activités*.

I.2.1. Objectifs et avantages du modèle OrBAC :

L'objectif d'OrBAC est de permettre la modélisation d'une variété de politiques de sécurité basées sur le concept de l'organisation. Pour arriver à ce but, et afin de réduire la complexité de gestion des droits d'accès, le modèle OrBAC repose sur quatre grands principes :

- L'organisation est l'entité centrale du modèle.
- Il y a deux niveaux d'abstraction (Les interactions d'OrBAC) :
  - ✓ un niveau concret : sujet, action, objet.

- ✓ un niveau abstrait : rôle, activité, vue.
- La possibilité d'exprimer des permissions, des interdictions, et des obligations.
- La possibilité d'exprimer les contextes.

L'introduction d'un niveau abstrait organisationnel permet aussi la structuration des entités comme la Figure I.1 le montre.

Ainsi dans OrBAC, un rôle est un ensemble de sujets sur lesquels sont appliquées les mêmes règles de sécurité. Identiquement, une activité est un ensemble d'actions sur lesquelles sont appliquées les mêmes règles de sécurité. Une vue est un ensemble d'objets sur lesquels sont appliquées les mêmes règles de sécurité.

Il existe de plus un outil complet permettant de spécifier une politique de sécurité OrBAC, de la simuler, de l'analyser (trouver les conflits entre permission et interdiction par exemple), de l'administrer et de la déployer : **Motorbac [4].**

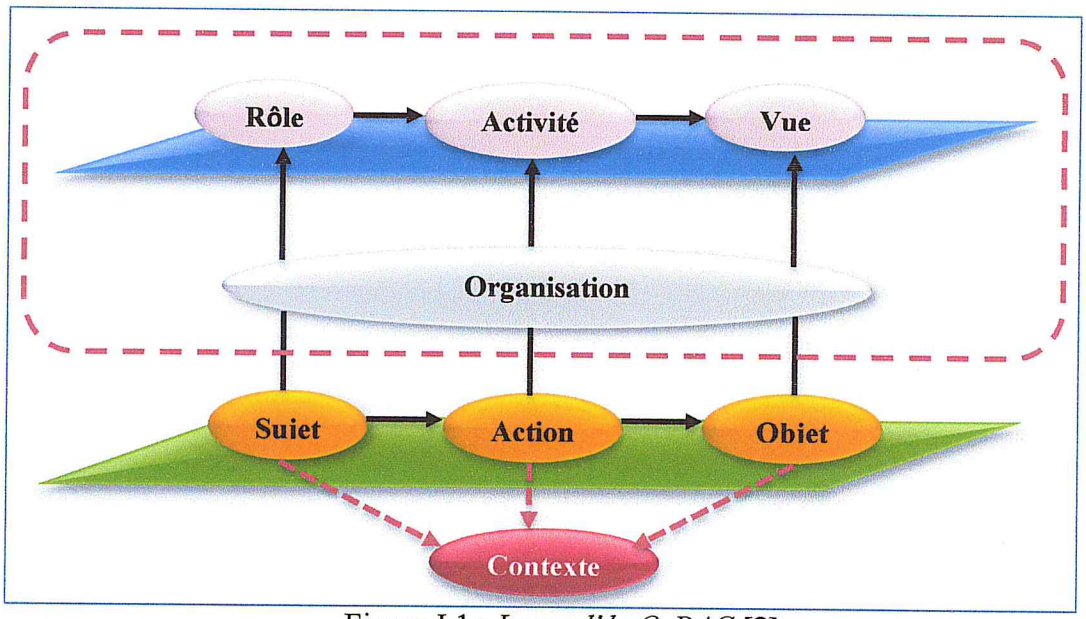


Figure I.1 : *Le modèle OrBAC* **[3]**

Ce modèle de contrôle d'accès nous permet de structurer l'ensemble des sujets, l'ensemble des objets ainsi que l'ensemble des actions au sein d'une organisation. La possibilité d'exprimer des permissions, des interdictions, et des obligations, peut être utile pour faire le filtrage au sein des catégories des utilisateurs définies dans une organisation.

I.2.2. Elément du modèle :

Cette section rappelle les principaux concepts du modèle OrBAC **[5]** présentés sous forme d'entités-associations.

I.2.2.1. Organisation:

L'entité centrale du modèle OrBAC est l'*organisation*. Elle peut être vue comme un groupe de sujets jouant certains rôles. Dans le domaine médical, un exemple d'organisation peut être l'unité de soins intensifs de l'hôpital *Mustapha*.

I.2.2.2. Sujets et rôles :

L'entité *Sujet* est utilisée différemment selon les modèles de sécurité. Dans le modèle OrBAC, un sujet peut être soit une entité active, c'est-à-dire un utilisateur, soit une organisation. Par exemple, " *Omar*", "*Jean*", etc., peuvent être des sujets, tout comme les organisations " *le service des urgences de l'hôpital Mustapha*", etc.

Dans OrBAC, l'entité *Rôle* est utilisée pour structurer le lien entre les sujets et les organisations. Dans le domaine médical, les rôles " *cardiologue*", " *infirmière*" ou "*médecin*", sont joués par des utilisateurs alors que les rôles " *service des urgences*" ou " *unité des soins intensifs*" sont joués par des organisations. Comme les sujets jouent des rôles dans des organisations, nous introduisons une relation entre ces entités : la relation *Habilite* (figure I.2). Si *org* est une organisation, *s* est un sujet et *r* est un rôle, alors ***habilite* (*org*, *s*, *r*)** signifie que *org* habilite le sujet *s* à jouer le rôle *r*.

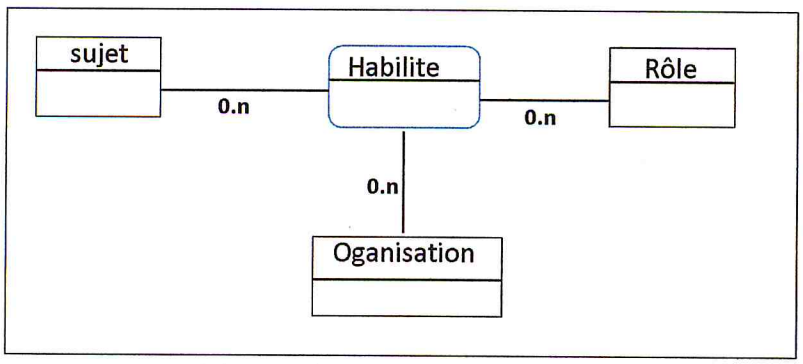


Figure I.2 : *La relation Habilite* **[3]**

I.2.2.3. Objets et vues :

Dans le modèle OrBAC, l'entité *Objet* représente principalement les entités non actives comme les fichiers, les courriers électroniques, les formulaires imprimés, etc.

Dans le domaine médical, nous aurons ainsi à considérer l'objet « les dossiers administratifs des patients».

Les rôles nous permettent de structurer les sujets et de faciliter la mise à jour de la politique de sécurité quand un nouvel utilisateur est ajouté. Dans la mesure où il est également nécessaire de structurer les objets et d'ajouter de nouveaux objets au système, nous considérons qu'une entité comparable au rôle pour les sujets est nécessaire pour les objets. Nous l'appelons : entité *Vue*. De manière intuitive, une vue correspond, comme dans les bases de données relationnelles, à un ensemble d'objets qui satisfait une propriété commune. Par exemple dans un système de fichier administratif, la vue " *dossiers administratifs*" correspond à l'ensemble des dossiers administratifs des patients, alors que la vue " *dossiers médicaux*" correspond aux dossiers médicaux des patients.

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 *Utilise* (figure I.3). Si *org* est une organisation, *o* est un objet et *v* est une vue, alors ***Utilise (org, o, v)*** signifie que *org* utilise l'objet *o* dans la vue *v*.

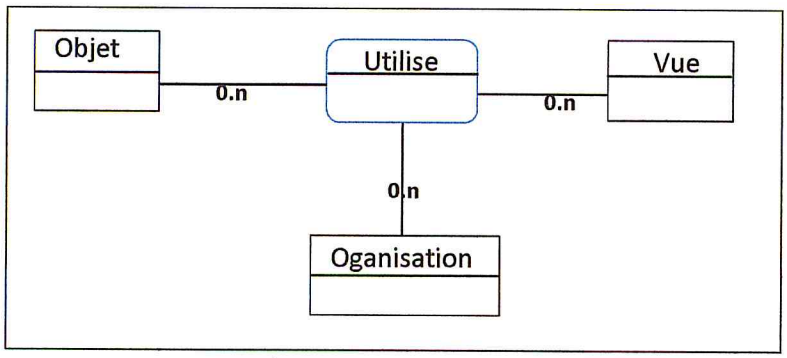


Figure I.3 : *La relation Utilise* **[3]**

I.2.2.4. Actions et activités :

Les politiques de sécurité spécifient les accès autorisés aux entités passives par des entités actives et régulent les actions opérées sur le système. Dans le modèle OrBAC, l'entité *Action* englobe principalement les actions informatiques comme "*lire*", " *écrire*", " *envoyer*", etc.

De la même manière que dans les sections I.2.2 et I.2.3 où 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 *Activité*. Ainsi, les rôles associent

des sujets qui remplissent les mêmes fonctions, les vues regroupent des objets qui satisfont une propriété commune et par analogie les activités correspondent à des actions qui ont un objectif commun.

Dans OrBAC, la mesure où des organisations différentes peuvent considérer qu'une même action est employée à la réalisation d'activités différentes, la relation *Considère* (figure 1.4) sera utilisée pour associer les entités *Organisation*, *Action* et *Activité*. Plus précisément, si *org* est une organisation, *a* est une action et *a* est une activité, alors *Considère (org, a, a)* signifie que l'organisation *org* considère l'action *a* comme faisant partie de l'activité *a*.

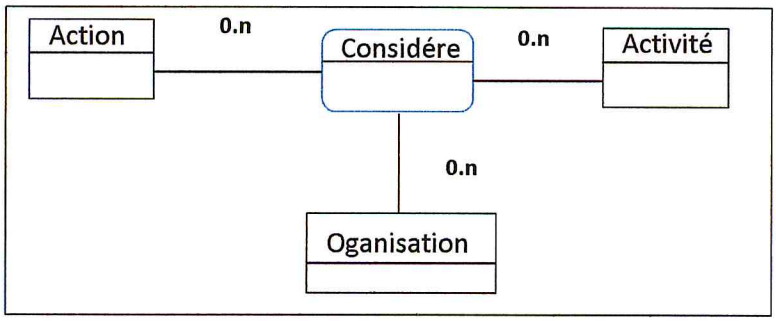


Figure I.4 : *La relation Considére* **[3]**

Si nous considérons l'activité " consultation". Cette activité peut correspondre, dans l'organisation hôpital « Mustapha Bacha », à l'action " lire" un fichier, mais peut tout aussi bien correspondre à l'action " select" sur une base de données dans l'hôpital « Ibn Badis ».

- *Considère* (*Mustapha_Bacha*, *lire*, *consultation*) :

L'hôpital *Mustapha_Bacha* considère *lire* comme une *consultation*.

I.2.3. Notion de contexte :

Les contextes sont utilisés pour spécifier les circonstances concrètes dans lesquelles les organisations accordent des permissions de réaliser des activités sur des vues. Dans le domaine médical, une nouvelle entité Contexte permettra d'exprimer des circonstances telles que " urgence", "médecin traitant", etc. Les contextes peuvent être vus comme des relations ternaires entre les sujets, les objets et les actions définis dans une certaine organisation. Par conséquent, les entités, Organisation, Sujet, Objet, Action et Contexte sont liées par une nouvelle relation appelée Définit (figure I.5) telle que : si *org* est une organisation, *s* est un sujet, *a* est

une action, *o* est un objet et *c* est un contexte, alors *Définit* (*org*, *s*, *a*, *o*, *c*) signifie qu'au sein de l'organisation *org*, le contexte *c* est vraie entre le sujet *s*, l'objet *o* et l'action *a*.

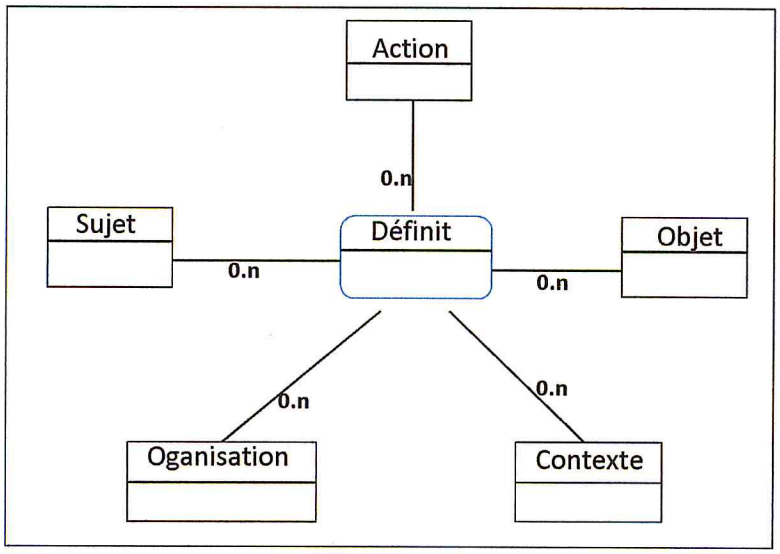


Figure I.5 : *La relation Définit* **[3]**

I.2.4. Politique de sécurité :

Dans le modèle OrBAC, les relations *Permission, Interdiction, Obligation et recommandation* correspondent à des relations entre les organisations, les rôles, les vues, les activités et les contextes. La relation *Permission (org, r, a, v, c)* signifié que l'organisation *org* accorde au rôle *r* la permission de réaliser l'action *a* sur la vue *v* dans le contexte *c*. Les relations *Interdiction (org, r, a, v, c), Obligation (org, r, a, v, c) et recommandation (org, r, a, v, c)* sont définies de façon similaire.

Les relations *Permission, Interdiction, Obligation et recommandation* sont introduites comme des faits. Elles ne sont pas directement associées aux utilisateurs, actions et objets mais à leur abstraction respectives rôles, activités et vues.

Par exemple, la politique de sécurité de l'hôpital Purpan peut contenir le fait suivant : *Permission (Purpan, médecin, consulter, dossier médical, médecin traitant)* qui signifie que l'hôpital Purpan accorde aux médecins la permission de consulter les dossiers médicaux des patients dont ils sont les médecins traitants.

I.2.5. Les autorisations concrètes :

La relation *Permission,* décrite précédemment, permet à une organisation donnée de spécifier les permissions accordées suivant le contexte. De telles permissions correspondent à une relation entre les rôles, les vues et les activités. Pour autant, le contrôle d'accès bas niveau doit permettre de décrire les actions concrètes que réalisent les sujets sur les objets.

Dans le but de modéliser des permissions concrètes, OrBAC introduit la relation *Est_permis* entre les sujets, les objets et les actions : si *s* est un sujet, *a* est une action et *o* est un objet, alors *Est_permis*(*s*, *a*, *o*) signifie que le sujet *s* a la permission de réaliser l'action *a* sur l'objet *o*. Dans le modèle OrBAC, les triplets, qui sont des instances de la relation *Est_permis (resp Est_interdit, Est_obligatoire et Est_recommandé)*, sont dérivés logiquement des permissions accordées aux rôles, aux vues et aux activités par la relation *Permission(resp Interdiction, Obligation et Recommandation)* .

I.3. Hiérarchies dans une organisation :

Dans le modèle Or-BAC, il est possible de définir des hiérarchies de rôles mais aussi des hiérarchies de vues et d'activités. Chaque hiérarchie définit respectivement une relation d'ordre partiel sur l'ensemble des rôles, des vues et des activités. Afin de modéliser ces différentes hiérarchies, nous présentons dans cette section les règles générales d'héritage des permissions qui leur sont associées **[6].**

I.3.1. Hiérarchie de rôles :

Nous nous attachons dans un premier temps à étudier la hiérarchie de rôles. Pour cela nous introduisons le prédicat *sub_role (org, r1, r2)* qui signifie : dans l'organisation *org*, le rôle *r1* est un sous-rôle du rôle *r2*.

Remarquons que la hiérarchie de rôles dépend de l'organisation. Ainsi, chaque organisation peut définir sa propre hiérarchie de rôles.

I.3.2. Hiérarchie d'activités :

Nous définissons dans cette section l'héritage entre les activités. Dans chaque organisation, les activités sont structurées sous forme de hiérarchies. La modélisation de ce type de relation hiérarchique est faite au moyen du prédicat *sub_activité (org, a1, a2)* qui signifie : dans l'organisation *org*, l'activité *a1* est une sous-activité de *a2*.

I.3.3. Hiérarchie de vues :

Comme pour les rôles et les activités, l'ensemble des vues est structuré par des hiérarchies dépendant de l'organisation. Cette hiérarchisation est modélisée par le prédicat *Sub_vue (org, v1, v2)* qui signifie : dans l'organisation *org*, la vue *v1* est une sous-vue de la vue *v2*.

La sémantique que nous attribuons à cette relation hiérarchique est la spécialisation. Cette structuration est en fait proche de la hiérarchie d'héritage de classes utilisée dans les approches orientées objet (la relation *Is a*).

I.4. Conclusion :

Nous avons présenté dans ce chapitre le modèle de politique de sécurité OrBAC, son but est d'apporter des réponses aux limites des modèles existants.
OrBAC est centré sur le concept organisation. En effet, tous les autres concepts que nous avons présenté et qui permettent de spécifier une politique de sécurité dépondent d'une organisation donnée **[5].**

En s'appuyant sur les concepts rôle, activité, vue, organisation, nous pouvons exprimer une politique de sécurité comme un ensemble de permissions, d'interdictions, d'obligations et de recommandations. Nous n'avons présenté que les permissions.

Nous avons également décrit les différents types de hiérarchie qui peuvent exister lors de la définition des entités du modèle telle que la hiérarchie de rôles, de vues ou d'activités.

Notre travail s'inspire du modèle OrBAC. La politique de sécurité est dynamique mais le formalisme utilisé est différent.

OrBAC est basé sur la logique du premier ordre où le contexte est un simple argument. Le modèle sur lequel nous nous sommes basées repose sur la logique de description non monotone. Avant de présenter les particularités de cette logique, nous présentons dans le chapitre suivant les caractéristiques de base d'une logique de description.

II.1. Introduction :

Un système à base de connaissances est un programme capable de raisonner sur un domaine d'application pour résoudre un certain problème, en utilisant des connaissances relatives au domaine étudié. Les connaissances du domaine sont représentées par des entités qui ont une description syntaxique à laquelle est associée une sémantique.

Il n'existe pas de méthode universelle pour concevoir de tels systèmes, mais un courant de recherche très actif s'est développé, qui s'est nourri d'études effectuées sur la logique des prédicats, les réseaux sémantiques et les langages de frames, a donné naissance à une famille de langages de représentation appelées logiques de description.**[7]**

Ce chapitre présente les logiques de description (LD), une famille de langages de représentation de connaissances qui peut être utilisée pour représenter la connaissance d'un domaine d'application par un moyen clair, formel et structuré. Ces langages exploitent, en général, des sous-ensembles décidables de la logique de premier ordre. Ils ont été largement étudiés et utilisés dans plusieurs systèmes à base de connaissances.

Les logiques de description décrivent les concepts d'un domaine en utilisant des concepts atomiques, correspondant à des prédicats unaires, et des rôles atomiques, correspondant à des prédicats binaires et décrivant les relations entre les objets/concepts du domaine. Les rôles sont spécifiés à l'aide de constructeurs fournis par le langage formel des logiques de description.

II.2. Origines de la logique de description :

Le développement des LD fut fortement influencé par les travaux sur la logique des prédicats et les réseaux sémantiques.

II.2.1. La première génération de logique de description (1980-1990) :

Les premiers travaux sur les LD commencèrent au début des années 1980 avec des systèmes à base de connaissances. Ces premières implantations résolvent des

Chapitre II

La logique de description

problèmes d'inférence en temps souvent polynomial, par le biais d'une catégorie d'algorithmes de vérification ,de subsomption et de type normalisation/comparaison. Ces algorithmes ne s'appliquent qu'à des LD peu expressives, donc ils sont incomplets, c'est-à-dire qu'ils sont incapables de prouver certaines formules vraies**[8].**

II.2.2. La deuxième génération de logique de description (1990-Aujourd'hui) :

Dans les années 1990, une nouvelle classe d'algorithmes est apparue : les algorithmes de vérification de satisfiabilité à base de tableaux. Ces derniers raisonnent sur des LD dites expressives ou très expressives, mais en temps exponentiel. Cependant, en pratique, le comportement des algorithmes est souvent acceptable. L'expressivité accrue a ouvert la porte à de nouvelles applications telles que le Web sémantique. Le terme logiques de description expressive (LDE) désigne l'ensemble des LD qui ont émergé pendant cette période**[8].**

II.3. Présentation de la logique de description :

Les logiques de description appelées aussi logiques descriptives (LDs) sont une famille de langages de représentation de connaissance qui peuvent être utilisés pour représenter la connaissance terminologique d'un domaine d'application d'une manière formelle et structurée. Le nom de logique de description se rapporte, d'une part à la description de concepts utilisée pour décrire un domaine et d'autre part à la sémantique basée sur la logique qui peut être donnée par une transcription en logique des prédicats du premier ordre. La logique de description a été développée comme une extension des frames et des réseaux sémantiques, qui ne possédaient pas de sémantique formelle basée sur la logique.

II.3.1. Objets de la logique de description :

Dans une logique de description Les éléments qui sont définis et manipulés sont : les concepts, les individus et les rôles.

- **Les concepts :** peuvent être vus comme des prédicats logiques unaires.
- **Les individus :** sont les instances des concepts.

- **Les rôles :** sont similaires aux prédicats logiques binaires. Les restrictions des rôles portent généralement sur le co-domaine, i.e., les concept avec lequel le rôle établit une relation, et la cardinalité, i.e., le nombre minimal et maximal que peut prendre un rôle **[9].**

II.3.2. Niveaux de description :

La modélisation des connaissances d'un domaine avec les LD se réalise en deux niveaux. Le premier, le niveau terminologique (ou TBox) et le niveau factuel (ou ABox). Plusieurs ABox peuvent être associés à une même TBox, chacune représente une configuration constituée d'individus, et utilise les concepts et rôles de la TBox pour l'exprimer.

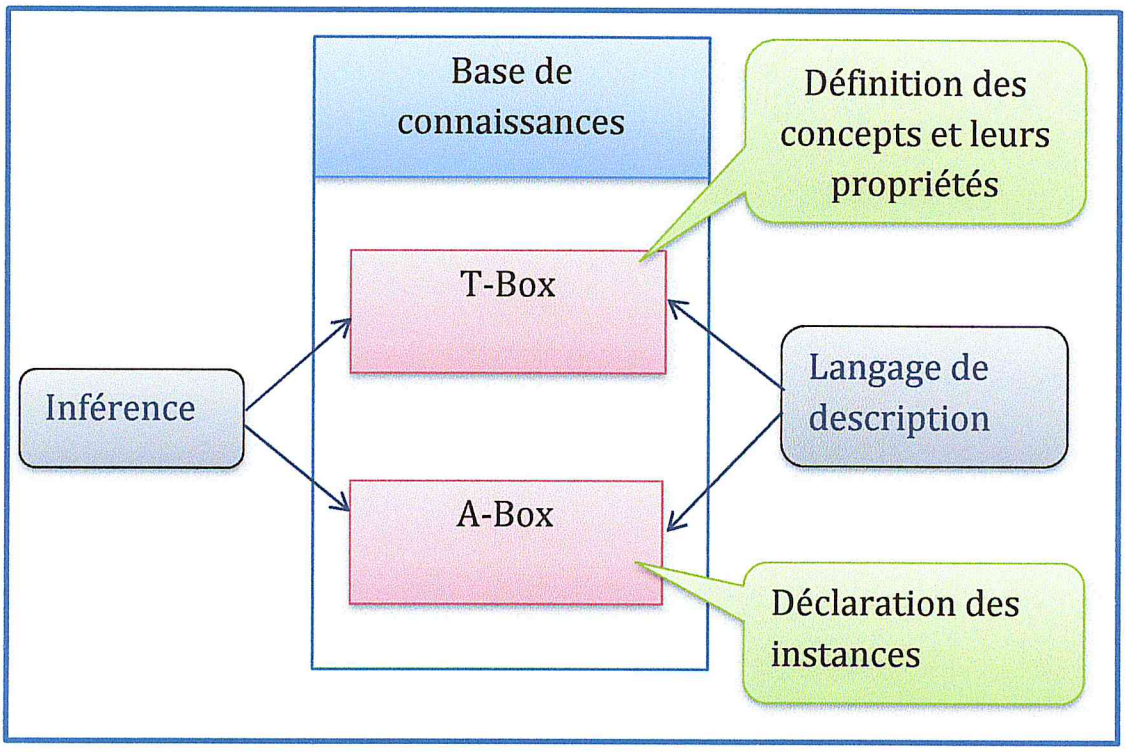


Figure II.1 : Structure générale des logiques de description **[10].**

II.3.2.1. Niveau terminologique (T-Box) :

La TBox comprend la définition des concepts et des rôles. Dans la T-Box, on est généralement intéressés à savoir si tous les concepts définis sont consistants, c'est à-dire si, pour chaque concept, il peut exister au moins un individu membre de cette

classe. Par exemple, si on définit une classe comme étant à la fois une sous-classe des classes *homme* et *femme* et que la T-Box spécifie aussi que ces deux classes sont disjointes (c'est-à-dire qu'aucune entité ne peut à la fois être un homme et une femme), on se retrouve alors avec un concept inconsistant. Un autre type d'inférence réalisé avec la T-Box est la *subsomption*, qui consiste à déduire qu'une classe est une sous-classe d'une autre classe, même si cela n'est pas déclaré explicitement dans la base de connaissances.

Nous allons nous intéresser dans ce projet à un autre type d'inférence réalisé avec la T-Box qui est *l'héritage*, qui permet, pour une sous-classe, la réutilisation des propriétés de sa superclasse, l'extension où une sous-classe ajoute ses propres propriétés, mais également la redéfinition, par la sous-classe, de propriétés de sa superclasse.

II.3.2.2. Niveau factuel (A-Box) :

Nommé aussi le langage assertionnel. Dans la A-Box, décrit les individus en les nommant et en spécifiant en termes de concepts et de rôles, des assertions qui portent sur ces individus nommés. En d'autres mots, on y spécifie quelles sont les entités du monde et à quelle classe elles appartiennent. La ABox contient aussi des énoncés spécifiant les relations qui existent entre les individus.

Les inférences avec la A-Box visent normalement à déterminer si un ensemble d'assertions est *consistant*, c'est-à-dire si un individu déclaré comme instance d'une classe peut réellement être une instance de cette classe et, similairement, si une relation déclarée entre deux individus est réellement possible. Supposons par exemple qu'une T-Box déclare qu'un célibataire est une personne non mariée. La A-Box sera inconsistante si elle contient un célibataire qui est marié avec une autre personne.

II.3.3.La logique minimale AL:

La plupart des langages utilisés découlent du langage *AL* (*Attributive Language*), dont l'expressivité est plutôt limitée **[11].**

II.3.3.1. Syntaxe du langage AL :

Dans ce langage, les axiomes sont construits à partir d'un ensemble de concepts et des constructeurs pour l'édification de concepts composés.
Les descriptions possibles dans le langage *AL* sont les suivantes (on suppose que A est un concept atomique et C, D sont des concepts atomiques ou complexes) :

C, D	A (Concept atomique)
	I T (Concept universel)
	I (Le concept le plus spécifique)
	I A (La négation atomique)
	I C Π D (L'intersection)
	I ∃R.T (Quantification existentielle limitée)
	I ∀ R.C (Quantification universelle complète)

Figure II.2 : La grammaire des expressions conceptuelles selon AL **[11].**

- Le constructeur C Π D permet de faire la conjonction de deux concepts composés, ce qui représente l'ensemble des individus, membres à la fois du concept C et du concept D pour une interprétation.
- Le constructeur A est utilisé pour évoquer la négation d'un concept atomique, c'est-à dire les individus pour une interprétation qui n'appartiennent pas au concept atomique A.
- Le quantificateur existentiel non typé R. désigne l'ensemble des individus, membres du domaine d'un rôle R pour une interprétation donnée.
- Le quantificateur universel R.C évoque l'ensemble des individus du domaine d'un rôle R qui sont en relation, par le biais de R, avec un individu du concept C, pour une interprétation donnée.

AL ne permet pas la spécification de rôles à l'aide de constructeurs (rôles composés). La sous-section qui suit décrit la sémantique formelle d'***AL***.

II.3.3.2. La sémantique formelle d'AL:

La sémantique du langage *AL* fait appel à la théorie des ensembles. Essentiellement, à chaque concept est associé un ensemble d'individus dénotés par ce concept. Une interprétation suppose donc l'existence d'un ensemble non vide Δ qui représente des entités du monde décrit.

Une fonction d'interprétation *I* qui associe à chaque description un sous-ensemble de Δ.

$I(T) = \Delta$
$I(\ \ ) = \{\}$
$I(\neg A) = I(\Delta) \setminus I(A)$
$I(C \sqcap D) = I(C) \sqcap I(D)$

Figure II.3 La sémantique formelle d'AL **[11]**

Deux concepts C et D d'une T-Box ***AL*** s'équivalent si et seulement si I (C) $= I$ (D) pour toute interprétation *I*.

II.3.4. Mécanismes d'inférence :

Des mécanismes d'inférence sont associés aux DLs, ils permettent la déduction des connaissances qui ne sont pas représentées explicitement dans la base.

Les deux inférences principales dans les DLs sont la classification de concepts qui s'effectue au niveau de la T-Box, et la reconnaissance d'instances qui s'effectue au niveau de la A-Box.

Ces deux opérations sont fondées sur la relation de subsomption.

II.3.4.1. Subsomption :

La subsomption permet de répondre à la question suivante : étant donné deux concepts C et D, lequel des deux est-il plus spécifique que l'autre ?

Autrement dit, La subsomption est la relation qui permet d'organiser les concepts par niveau de généralité (en hiérarchie). Intuitivement, un concept C subsume un concept D si C'est plus générale que D, i.e., l'ensemble des individus représentés par C contient l'ensemble d'individus représentés par **W. A Woods** décrit cinq types de subsomptions et les relations qui existent entre ces différents types**[12]**.

Nous ne définissons que deus types qui sont la subsomption extensionnelle et structurelle.

II.3.4.1.1. Subsomption extensionnelle:

Un concept *C subsume* un concept *D* si et seulement si l'ensemble des individus *(instances)* dénotés par *C* contient l'ensemble des individus *(instances)* dénotés par *D*. Par exemple, l'ensemble des instances du concept Enfant est inclus dans l'ensemble des instances du concept Etre-Humain.

II.3.4.1.2. Subsomption structurelle (ou intentionnelle) :

Un concept *C subsume* un concept *D* si et seulement si l'ensemble de ses *propriétés* est inclus dans l'ensemble des *propriétés* de *D*. Par exemple, l'ensemble des propriétés du concept Enfant contient l'ensemble des propriétés du concept Etre-Humain.

II.3.4.2. Inférences terminologiques :

Nous présentons dans ce qui suit les principales opérations que l'on trouve au niveau de la T-Box des DLs et que l'on appelle les opérations terminologiques.

II.3.4.2.1. Classification des concepts :

La classification de concepts est l'opération qui permet de placer un concept donnée à la place la plus appropriée dans la hiérarchie. Le processus de classification permet de découvrir les relations de subsomption qui existent entre un nouveau concept et les concepts présents dans une taxonomie.

Ce processus s'effectue en deux phases qui sont :

- Trouver les concepts les plus spécifiques qui subsument le concept *D* (ce sont les subsumeurs).
- Rechercher les concepts les plus généraux que *D* subsume (ce sont les *subsumés* de *D*).

II.3.4.2.2. Complétion :

Ce terme désigne un ensemble d'inférences qui permettent de retrouver toutes les propriétés d'un concept. La complétion permet de retrouver les propriétés héritées du concept mais aussi des propriétés déduites logiquement.

II.3.4.2.3. Héritage :

Un concept hérite des propriétés des concepts qui le subsument. Le mécanisme d'héritage consiste à retrouver toutes les propriétés d'un concept à partir des propriétés des concepts qui le subsument.

II.3.4.2.4. Détection d'incohérence :

Cette opération consiste à détecter les concepts incohérents (i.e. qui possèdent une définition incohérente). D'un point de vue extensionnel, un concept incohérent est dénoté par l'ensemble vide (aucun individu n'est instance de ce concept).

II.3.4.3. Inférences assertionnelles :

Les opérations assertionnelles varient d'une DL à une autre. La plupart des ABox comprennent au moins l'opération de reconnaissance d'instances.

II.3.4.3.1. Reconnaissance d'instance :

La reconnaissance d'instances consiste à trouver pour un individu donné les concepts les plus spécifiques dont il est instance. L'opération élémentaire utilisée dans cette recherche est l'opération de *test d'instance*, qui consiste à vérifier si un objet *O* est une instance d'un concept *C*.

La méthode *abstraction- classification* est l'une des méthodes employées pour faire de la reconnaissance d'instances. Elle consiste à calculer l'abstraction d'un objet (i.e. transformer un objet en un concept appelé *"concept abstrait"*) et de classer le concept obtenu.

Pour calculer le concept abstrait d'un individu, on retrouve toutes les informations liées à l'individu et on les rassemble sous la forme d'une définition de concept la plus spécifique possible.

II.3.4.3.2. Propagation :

Une assertion sur un individu *o1* peut avoir des conséquences logiques sur des individus en relation avec *o1*. Les DL qui bénéficient de l'opération de propagation propagent les nouvelles informations déduites sur les individus concernés.

II.4. la logique de description AL δε :

Dans cette section nous présentons la logique AL δε qui est une extension des logiques de descriptions classiques.

La logique de description ***ALδε*** permet d'introduire les notions défauts et exception, qui sont très importantes pour la représentation non monotone de connaissances, symbolisées par les deux connecteurs **δ** et **ε**.

II.4.1. Syntaxes d'ALδε :

La logique ***ALδε*** **[14][15]** est définie à partie d'un ensemble r de rôles primitifs, d'un ensemble P de concepts primitifs, des constantes ⊤ et ⊥ et de la règle syntaxique suivante : (C et D sont des concepts).

C, D → T Le concept le plus général
| ⊥ Le concept le plus spécifique
| *P* Concept primitif
| C∏ D Conjonction de concepts
| ¬ *P* Négation de concept primitif
| ∀ *r* : C Restriction de valeurs pour les rôles (R>0)
| δ C Concept défaut
| Cε Exception sur un concept

II.4.2. Propriétés des connecteurs δ et ε :

Voici les axiomes ci-dessous qui représentent les propriétés des connecteurs δ et ε qui sont d'une grande importance dans le raisonnement basé sur une logique qui introduit les notions de défaut et exception :

a. (δ A)ε ≡ Aε
b. δ(A ∏ B) ≡ (δA) ∏ (δB)
c. A ∏ δA ≡ A
d. Aε ∏ δA ≡ Aε
e. δδ A ≡ δA

On exprime dans l'axiome **a** le fait qu'une exception n'a de sens que si elle porte sur une propriété défaut.

Dans l'axiome **b**, on représente la propriété de distributivité du connecteur δ par rapport à la conjonction de deux concepts.

On définit dans les axiomes **c** et **d** les relations de subsomption qui existent entre A et δA (δA subsume A) et entre Aε et δA (δA subsume Aε) .

Dans l'axiome est_permet la suppression de chaînes de défauts redondantes.

II.4.3. Caractéristiques de la logique ALδε :

II.4.3.1. Héritage :

Dans une logique de description qui ne contient pas de connecteurs δ et ε, un concept hérite toutes les propriétés du concept qui le subsume on peut dire alors que le mécanisme de subsomption dans un cadre stricte. Or, dans un cadre d'***ALδε*** où il y a des connaissances de type défaut et exception, un concept ne pourra pas hériter de toutes les propriétés d'un concept qui le subsume.

II.4.3..2. Points forts :

L'un des points forts de la logique ***ALδε*** est d'introduire les défauts et les exceptions au sein même d'une définition d'un concept et non en tant que règle incidente comme c'est le cas dans les autres approches.

II.4.3.3. Points faibles :

Le but fixé par les concepteurs des DLs est la plus part du temps « la puissance d'expressivité ». La logique de description ***ALδε*** n'est pas assez expressive d'une part et d'autre part elle n'est pas dotée d'algorithme d'inférences qui permettent l'évolution de la base de connaissance, c'est les raisons pour les quelles elle n'est pas utilisée dans le cadre pratique.

II.5. Conclusion :

Nous avons présenté dans ce chapitre les logiques de descriptions en donnant leurs origines, leurs langages (terminologique /assertionnel) ainsi que la logique minimale avec ses extensions et les services d'inférence.

Les DLs se sont développées pour devenir une clé importante dans l'histoire de la représentation de la connaissance. La famille des langages DLs est probablement l'ensemble le plus familier de toutes les représentations de la Connaissance.

III.1.Introduction :

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 politiques permissives, les politiques qui n'intègrent que des autorisations positives. Or, comme nous l'avons décrit dans le chapitre I, le modèle Or-BAC permet d'exprimer des autorisations négatives, aussi appelées *Interdictions* **[13].**

Dans ce chapitre, nous nous intéressons à la Gestion des conflits, et nous allons voir les solutions proposées. Dans le modèle Or-BAC, des permissions et aussi des interdictions et des obligations peuvent être exprimées. Le fait de pouvoir exprimer ces trois types de règles répond à un objectif précis : ne pas limiter la politique de sécurité au simple contrôle d'accès aux objets mais de contrôler également l'*usage* qui est fait de ces objets.

Néanmoins, nous considérons que les interdictions sont essentielles pour rédiger des politiques de sécurité à la fois claires et expressives.
Spécifier une politique de sécurité qui inclut à la fois des permissions, des interdictions et des obligations peut mener à des situations incompatibles (situations de *Conflits*) **[16]** et cela correspond aux situations dans lesquelles : « *le sujet est en même temps permit et interdit d'exécuter une action donnée sur un objet donne. D'où le système ne peut pas décider de permettre ou de nier d'accès* », autrement dit : «*Un conflit apparait lorsqu'une permission et une interdiction sont appliquées au même triplet < sujet, action, objet >* » **[5]** On remarque qu'un modèle de sécurité qui permet d'exprimer des interdictions devrait permettre de spécifier une « *Politique de gestion des conflits* », cette politique consiste en un ensemble de règles qui permettent au système de décider dans le cas d'un conflit, d'abandonner soit la permission ou bien l' Interdiction. En conséquence, la politique de sécurité devrait avoir la possibilité de définir sa propre politique de gestion de conflits, afin d'obtenir une politique de contrôle d'accès pertinente.

Nous voyons dans un premier temps à quels objectifs répondent les interdictions et comment elles s'expriment dans notre modèle. Puis, nous expliquerons dans la section III.7 l'approche générale pour la gestion des conflits. La définition d'une politique de gestion des conflits est abordée dans la section III.8. Enfin, nous

donnerons les grandes lignes de la prévention des conflits dans la dernière section.

III.2. Types de conflit :

On peut conclure qu'une politique de gestion de conflits devrait prendre en considération des conflits potentiels. Nous faisons distinction entre conflits «*Effectif*» et conflits «*Potentiels* », pendant le temps de la mise en vigueur de la politique dans le système d'information, quelques conflits apparaissent entre permission et interdiction quand une demande de sécurité est faite.

Par exemple, un utilisateur *Mohamed* essaye de lire un dossier donné, et la politique de contrôle d'accès conclut une permission et une interdiction pour cette demande de lecture, c'est un conflit « *effectif* ».

En revanche, nous pouvons détecter des conflits avant qu'ils se produisent, et plus précisément pendant que de spécification la politique. Cela consiste à détecter la coexistence des règles qui peuvent mener à quelques conflits si leurs conditions associées sont satisfaites simultanément. De tels conflits sont appelés des *«Conflits potentiels* ».

Le modèle Or-BAC est particulièrement adapté pour une telle distinction entre conflits effectifs et conflits potentiels puisque les *conflits effectifs* correspondent aux conflits entre *les autorisations concrètes* et les *conflits potentiels* correspondent aux conflits entre *les autorisations abstraites*. Nous allons illustrer ceci par un exemple.

Exemple de conflit potentiel :

Supposons que dans une banque donnée, le rôle « *Employé du comptoir* » a l'interdiction de modifier les comptes des clients, alors que le rôle « *conseiller* » a la permission de le faire.

Ces deux autorisations mènent à un « *conflit potentiel* » si l'utilisateur joue ces deux rôles.

Notons que si une contrainte explicite déclare que ces deux rôles sont séparés, alors il n'y a plus de « *Conflit potentiel* ». Par conséquent, puisque le conflit potentiel est détecté avant le conflit effectif, donc le conflit effectif ne peut pas avoir lieu.

Exemple de conflit effectif:

Supposons que l'utilisateur *Mohamed* est autorisé aux deux rôles « *Employé du comptoir* » et « *conseiller* ». Comme résultat, « *un conflit effectifs* » se produira quand *Mohamed* essaie de modifier un compte.

III.3. Les différents conflits possibles :

Dans le modèle OrBAC, il est possible de spécifier une politique de sécurité en accordant *Permission, interdiction, Obligation* et *Recommandation.* Cela fournit une structure très expressive et flexible pour exprimer plusieurs exigences de la sécurité. Par exemple, il est possible de spécifier les règles de sécurité générales correspondantes aux permissions (resp. obligations), et alors spécifier des exceptions possibles à ces dispositions générales qui correspondent aux interdictions (resp. recommandations).

Cependant, quand un modèle de politique de sécurité inclut la possibilité de spécifier les permissions, interdictions, obligations et recommandations, quelques conflits peuvent se produire. Nous identifions les conflits possibles suivants:

III.3.1. Conflit entre permission et interdiction :

- **Conflit effectif:** Ce conflit se produit quand il est possible de dériver, au niveau concret, pour un *sujet s, action a* et *objet o*, les deux relations suivantes:

  Est_permis (s, a, o) and Est_interdit (s, a, o).

Par exemple, dans un contexte donnée le sujet « *Mohamed* » est-permis de réaliser l'action « *lire* » sur l'objet « *dossier chirurgicale* » et au moment même il est interdit d'effectuer cette même action sur ce même objet donc un conflit apparait.

- **Conflits potentiel :** Ce conflit se produit quand il est possible de dériver au niveau abstrait, pour *un rôle r, une activité a, une vue v et une organisation org* les deux relations :

  δ Permission (org, r, a, v) and δ Interdiction (org, r, a, v).

III.3.2. Conflit entre obligation et recommandation :

- **Conflit effectif :** Ce conflit se produit quand il est possible de dériver, au niveau concret, pour un *sujet s, action a et objet o*, les deux relations suivantes :

  Est_obligatoire (s, a, o) and Est_recommandé (s, a, o).

Par exemple, le sujet « *Hanane* » qui est un médecin est-obligé de réaliser l'action «*Informer le malade de son état* » sur l'objet « *Malade* », et au même moment est recommandé de réaliser cette même action sur ce même objet et dans ce cas un conflit apparait.

- **Conflits potentiel :** Ce conflit se produit quand il est possible de dériver au niveau abstrait, pour un *rôle r,* une *activité a,* une *vue v et* une *organisation org*,les deux relations suivantes :

  δ Obligation (org, r, a, v) and δ Recommandation (org, r, a, v).

III.3.3. Conflit entre obligation et interdiction :

- **Conflit effectif :** Ce conflit se produit quand il est possible de dériver, au niveau concret, pour un *sujet s, action a* et *objet o*, les deux relations suivantes:

  Est_obligatoire (s, a, o) and Est_interdit (s, a, o).

Par exemple, le sujet « *Maria* » a l'obligation de réaliser l'action « *envoyer* » sur l'objet « *document* » et elle reçoit l'interdiction de réaliser cette action sur cet objet, dans ce cas un conflit apparait.

- **Conflits potentiel :** Ce conflit se produit quand il est possible de dériver au niveau abstrait, pour un *rôle r,* une *activité a,* une *vue v et* une *organisation org*, les deux relations :

  δ Obligation (org, r, a, v) and δ Interdiction (org, r, a, v).

Cependant, puisqu'une *obligation* implique une *permission* (s'il y a un conflit entre une obligation et une interdiction, alors il y a aussi un conflit entre une permission et une interdiction).

Donc un conflit entre une obligation et une interdiction n'est pas réellement un conflit primaire.

De ce fait, nous avons deux conflits primaires à diriger dans Or-BAC:

- conflit entre permission et interdiction et,
- conflit entre obligation et recommandation.

Quand quelques conflits se produisent, nous avons besoin d'une politique pour résoudre ces éventuels conflits. Le principe de base pour définir une telle politique est d'assigner des priorités aux règles de sécurité. Pour simplifier notre propos, nous ne considérons ici que les conflits entre les permissions et les interdictions**[17].**

III.4. Détection des conflits :

Comme nous avions vu auparavant, dans le modèle Or-BAC, il existe des autorisations concrètes et des autorisations abstraites. Ainsi, des conflits peuvent

apparaitre au niveau concret comme au niveau abstrait. Nous pourrions avoir par exemple :

- *Est_permis (Mohamed, Lire, fiche_client_21_pdf)* et *Est interdit (Mohamed, Lire, fiche client21_pdf)*
- *δ Permission (département_informatique, administrateur, consulté, fiche _client)* et *δ Interdiction (département_informatique, administrateur, consulté, fiche_client)*

La politique de gestion de conflits doit offrir des moyens pour détecter les autorisations (permissions ou interdictions) redondantes. Quand une politique de gestion de conflits est appliquée, quelques autorisations peuvent devenir inutiles parce qu'elles sont toujours en conflit avec d'autres autorisations plus prioritaires, et ainsi ne prennent jamais la priorité. De telles autorisations sont appelées «*Autorisations redondantes* », mais cela n'est pas l'objectif de notre travail, car nous allons traiter que les autorisations simples.

Il est possible avec Or-BAC de détecter et de gérer les conflits au niveau concret (entre autorisations concrètes), mais également au niveau abstrait (entre des autorisations abstraites). Le deuxième cas nous intéresse tout particulièrement et constitue le principal objectif de notre travail. En effet, notre but est de résoudre les conflits au niveau abstrait, 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 apparaitre dans la politique de sécurité concrète. Ainsi, une même politique de sécurité abstraite 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.

Donc, l'objectif est de pouvoir garantir que s'il n'existe pas de conflit au niveau abstrait, alors il ne pourra pas en exister au niveau concret.

III.5. Spécification de la théorie logique:

Le but de la spécification des règles de fonctionnement et des règles de sécurité est de définir les différents flux d'information et les contrôles d'accès. II ne s'agit de représenter que le fonctionnement pertinent vis-à-vis de la sécurité afin de pouvoir,

Dossier_chirurgical est une instance instances du concept Vue.

Frantz_fanon est une instance du concept Organisation.

u1 est une instance instances du concept Utilise.

III.5.4. Les actions et les activités :

Dans l'exemple suivant nous considérons les activités correspondantes à des accès directs aux dossiers, par exemple dans l'hôpital Frantz_Fanon, «*écrire*», est considèré comme une création.

Considère(C1) $\subseteq$ *and* *((*$\ni$ considèreAc.activité (création))

($\ni$considèreA.Action(écrire)) ($\ni$ considèreOr.Organisation (Frantz_Fanon))).

Donc, création est une instance du concept Activité.

Ecrire du concept Action.

Frantz_franon est une instance du concept Organisation.

Cl est une instance du concept Considère.

III.5.5. La hiérarchie :

L'héritage est un mécanisme qui permet de maitriser la complexité.

Dans l'hôpital Frantz Fanon le directeur joue aussi le rôle d'anesthésiste, de ces faits le rôle de directeur doit hériter l'ensemble des permissions du rôle d'anesthésiste.

La règle qui exprime ce fait s'écrit sous la forme :

(δ Permission (Frantz__Fanon, anesthésiste, a, v)
$\subseteq \delta$ Permission (Frantz fanon, directeur, a, v)).

Dans notre modélisation nous avons utilisé la relation *sous_rôle*, c'est plus intéressant d'utiliser cette relation pour exprimer la hiérarchie des rôles et l'héritage de permission entre ces rôles.

A titre d'exemple, supposons que si dans un hôpital, un rôle *r2* est un sous rôle de *r1*, alors le rôle *r2* hérite de toutes les permissions de *r1*, ce fait est exprimé à l'aide de la règle **RH1** suivante :

RH1 :

δ Permission(Org,r2,a,v) $\subseteq$ (Sous_role(org, r1, r2) $\cap \delta$ Permission(Org,r1,a,v)).

Tandis que l'héritage des interdictions entre rôles est exprimé par la règle **RH2** qui

suit:

RH2 :

δ *Interdiction(Org,r2,a,v)* $\subseteq$ *(Sous_role(org, r1, r2)* $\cap$ δ *Interdiction(Org,r1,a,v)).*

Prenons par exemple, le rôle *secrétaire_médicale* est considère comme un sous rôle de *professionnelle_de_santé* dans l'hôpital, a ce fait le professionnel_de_santé héritera de toutes les permissions de la secrétaire_médicale. Ceci est présenté comme suit :

δ *Permission(*hôpital, secrétaire_médicale, a, v*)* $\subseteq$ *(Sous_role(*hôpital, professionnelle_santé, secrétaire_médicale*)* $\cap$ δ *Permission(*hôpital, professionnel de santé, a, v*))*

Nous pourrions même exprimer la hiérarchique des rôles dans une organisation, si un *rôle r2 est un sous rôle de rl, et r3* est un *sous rôle de r2*, alors *r3* est un *sous rôle de r1*, ce fait est exprimé à l'aide de la règle :

RH :

Sous_ rôle(SR1) $\subseteq$ *(Sous_ rôleR.r1* $\cap$ *Sous_ rôleR.r2* $\cap$ *Sous_ rôleOr.org)*

Sous_ rôle(SR2) $\subseteq$ *(Sous_ rôleR.r2* $\cap$ *Sous_ rôleR.r3* $\cap$ *Sous_ rôleOr.org)*

Sous_ rôle $\subseteq$*(Sous_ rôle(SR1)* $\cap$ *Sous_ rôle(SR2))*

L'expression de la hiérarchie des organisations se fera de la même manière, mais cette fois en utilisant la relation Sous_organisation, si une organisation org2 est une sous organisation de org1 et org3 est une sous organisation de org2 alors org3 est une sous organisation de org1, ce fait est exprimé à l'aide de la règle:

OH:

Sous _organisation(SO1) $\subseteq$*(Sous _organisationOr.org1* $\cap$*Sous_organisationOr.org2).*

Sous _organisation(SO2) $\subseteq$*(Sous _organisationOr.org2* $\cap$*Sous_organisationOr.org3).*

Sous _organisation $\subseteq$ *(Sous _organisation(SO1)* $\cap$ *Sous _organisation(SO2)).*

III.5.6. Contraintes de séparations :

Lors de la création des entités organisationnelles, il est également possible de définir des contraintes que doivent respecter ces entités. Toute mise à jour de la politique de sécurité qui viole une de ces contraintes est rejetée dans Or-BAC, une contrainte est modélisée par une règle qui dérive sur une erreur. **[16]**

III.5.6.1. Séparation rôle :

Certaines contraintes comme la séparation de rôle ont un format générique prédéfini dans Or-BAC. Cette contrainte utilise le prédicat *Séparation_rôle* qui correspond à la règle suivante :

C1:

$$Habilite1 \subseteq (HabiliteS.s \cap HabiliteR.r1 \cap HabiliteOr.org1).$$

$$Habilite2 \subseteq (HabiliteS.s \cap HabiliteR.r2 \cap HabiliteOr.org2).$$

$$Séparation\_rôle \subseteq (Séparation\_rôleOr.org1 \cap Séparation\_rôleR.r1 \cap Séparation\_rôle\ Or.org2 \cap Séparation\_\ rôle\ R.r2).$$

$$\mathbf{Erreur \subseteq (\ Habilite1 \cap\ Habilite2 \cap Séparation\_rôle).}$$

Cette règle signifie que si le rôle *r1* dans l'organisation *Org1* est séparé du rôle *r2* dans l'organisation *Org2*, alors un sujet ne peut jouer les deux rôles, r1 dans l'organisation *Org1* et le rôle *r2* dans l'organisation *Org2*, alors la contrainte est fausse ce qui donne le prédicat erreur, autrement dit aucun sujet s ne peut être simultanément affecté au rôle *r1* dans l'organisation *Org1* et au rôle *r2* dans l'organisation *Org2*.

III.5.6.2. Séparation vue:

C2:

$$Utilise1 \subseteq (UtiliseO.o \cap Utilise\ V.v1 \cap UlitiseOr.org1)$$

$$Utilise\ 2 \subseteq (UtiliseO.o \cap Utilise\ V.v2 \cap UlitiseOr.org2)$$

$$Séparation\_vue \subseteq (Séparation\_vueOr.org1 \cap Séparation\_vueV.v1 \cap Séparation\_vueOr.org2 \cap Séparation\_vueV.v2)$$

$$\mathbf{Erreur \subseteq (\ Ulilise\ U1 \cap Ulilise\ U2 \cap Séparation\_vue).}$$

Cette règle spécifie que si une règle de séparation de vues existe entre *v1* et *v2* dans les organisations *Org1* et *Org2*, alors aucun objet o ne peut être simultanément utilise

dans la vue *v1* dans l'organisation *Org1* et dans la vue *v2* dans l'organisation *Org2*.

III.5.6.3. Séparation activité :

C3:

$$\forall org1 \in Org,\ \forall org2 \in Org,\ \forall ac1 \in AV,\ \forall ac2 \in AV,\ \forall a \in A,$$

$$Séparation\_activité \subseteq (Séparation\_activitéOr.org1 \cap Séparation\_activitéAV.ac1 \cap Séparation\_activitéOr.org2 \cap activitéAV.ac2).$$

$$Considére1 \subseteq (ConsidéreA.a \cap ConsidéreAv.ac1 \cap ConsidéreOr.org1).$$

$$Considere2 \subseteq (ConsidéreA.a \cap ConsidéreAv.ac2 \cap ConsidéreOr.org2).$$

$$Erreur \subseteq (Considére1 \cap Considére2 \cap Séparation\_activité).$$

Cette règle spécifie que si une règle de séparation d'activités existe entre *ac1* et *ac2* dans les organisations *Org1* et *Org2*, alors aucune action *a* ne peut être simultanément affectée à l'activité *ac1* dans l'organisation *Org1* et à l'activité *ac2* dans l'organisation *Org2*.

III.5.7. Spécification des règles de sécurité :

Du point de vue formel, une règle de sécurité est une formule. Cette règle reflète la manière dont l'état de sécurité est en relation avec les différentes permissions, obligations, interdictions et recommandations qui existent dans le système. **[16]**

Les règles de dérivation :

- δ *Permission (Organisation, Rôle, Activité, Vue)*

Cette relation signifie que l'organisation donne la permission à un rôle de réaliser une *activité* sur une *vue*. Une telle permission est dite abstraite. L'objectif dans le modèle Or- BAC est de rédiger la politique de sécurité à l'aide de permissions abstraites. Les permissions concrètes sont alors dérivées des permissions abstraites. La règle de dérivation **RG1** est la suivante

RG1:

$$Habilite \subseteq (HabiliteS.s \cap HabiliteR.r \cap HabiliteOr.org).$$

$$Considére \subseteq (ConsidéreAc.a \cap ConsidéreAv.ac \cap ConsidéreOr.org).$$

$$Utilise \subseteq (UtiliseO.o \cap Utilise\ V.v \cap UlitiseOr.org).$$

$$\delta\ Permission(Org, r, a, v) \subseteq (PermissionR.r \cap PermissionAv.ac \cap PermissionV.v \cap PermissionOr.org).$$

$$\mathbf{Est\_permis(s, a, o) \subseteq (Habilite \cap Utilise \cap Considére \cap \delta\ Permission).}$$

Pour toute organisation *org*, si *org* accorde la permission au rôle *r* de réaliser l'acivité *a* sur la vue *v*, et si *org habilite* le sujet *s* dans le rôle *r*, et si *org utilise* l'objet *o* dans la vue *v*, et si *org considère* l'action *a* comme faisant partie de l'activité *a*, alors le sujet *s* a la permission de réaliser l'action a sur l'objet *o*.

Le passage à la relation "Est_interdit" se fait de façon similaire.
Comme pour les permissions, les interdictions concrètes sont déduites des interdictions organisationnelles (abstraites) grâce à une règle de dérivation **RG2**. Cette règle est obtenue en remplaçant dans **RG1** les permissions par des interdictions. Enfin, il est également possible de définir un héritage des interdictions. Donc la règle de dérivation **RG2** est la suivante :

RG2:

$Habilite \subseteq (HabiliteS.s \cap HabiliteR.r \cap HabiliteOr.org).$

$Considére \subseteq (ConsidéreAc.a \cap ConsidéreAv.ac \cap ConsidéreOr.org).$

$Utilise \subseteq (UtiliseO.o \cap Utilise\ V.v \cap UlitiseOr.org).$

$\delta\ Interdiction(Org, r, a,v) \subseteq (Interdiction\ R.r \cap Interdiction\ Av.ac \cap Interdiction\ V.v \cap Interdiction\ Or.org).$

$\mathbf{Est\_interdit(s, a, o) \subseteq (Habilite \cap Utilise \cap Considére \cap \delta\ Interdiction).}$

RG2 signifie que : Pour toute organisation *org*, si *org* accorde l'interdiction au rôle *r* de réaliser l'activité *a* sur la vue *v*, et si *org habilite* le sujet *s* dans le rôle *r*, et si *org utilise* l'objet *o* dans la vue *v*, et si *org considère* l'action *a* comme faisant partie de l'activité *a*, alors le sujet *s* a l'Interdiction de réaliser l'action *a* sur l'objet *o*.
Ainsi, la dérivation des autorisations concrètes à partir des autorisations abstraites permet de simplifier le processus de détection de conflits car cela garantie le non détection de situation conflictuelle dans le mode concret Si cette situation n'existe pas dans le mode abstrait.

III.6. Conflit dans la théorie :

Dans le modèle Or-BAC, quand un utilisateur demande la permission d'accéder à un objet, le système doit prendre une décision conformément à la possibilité de dériver des permissions concrètes (positives et/ou négative) en relation avec ce sujet, cette action et cet objet. Par conséquent, un conflit se produit quand une permission et une interdiction concrète sont dérivées pour le même sujet, action et objet. Pour

caractériser une telle situation nous introduisons l'attribut *conflit ()*. La règle suivante spécifie une situation de conflit :

RC: *Est_permis(s, a, o) and Est_interdit(s, a, o).*
→ *Conflit ().*

Laissez-nous supposer que les deux relations suivantes sont définies dans la politique:

δ Permission (org, r, a, v) and δ Interdiction (org, r, a, v).

En réalité de telles relations ne sont pas suffisantes pour dériver des paires conflictuelles d'autorisations concrètes. Plus cérémonieusement, un conflit se produit seulement si un sujet est autorisé dans le rôle *r*, une action est considérée comme une activité *a*, un objet *o* est utilisé dans une vue *v* dans l'organisation *org*. En outre, cette condition est suffisante mais pas nécessaire. Il n'y a réellement aucun besoin d'avoir une permission et une interdiction à solliciter exactement au même rôle, activité et vue pour qu'un conflit se dérive. Considérons les autorisations suivantes:

δ Permission (org, r1, a, v) and δ Interdiction (org, r2, a, v).

Si un sujet *s* joue les deux rôles *r1* et *r2*, et si une action est considérée comme une activité *a* et un objet *o* est utilisé dans une vue *v*, dans l'organisation org, alors il est possible de dériver une paire conflictuelle de prédicats concrets. En conséquence, les conflits effectifs peuvent être détectés seulement au niveau concret. En revanche, le niveau abstrait nous permet de déterminer les conflits potentiels.

Nous présentons une fonction pour analyser les conflits au niveau abstrait. Pour détecter ces conflits, nous introduisons d'abord le concept conflit. La présence de ce dernier est ensuite détectée en appliquant une règle de sécurité **[19].**

Cette règle doit indiquer que si:

1) une permission et une interdiction organisationnelles (abstraites) existent dans les organisations Orgl et Org2, et

2) il n'existe pas de contraintes de séparation entre les rôles, activités et vues, et

3) les priorités associées respectivement à la permission et à l'interdiction ne sont pas comparables, **alors un conflit est dérivé**.

III.7. Résolution de Conflit:

Supposons qu'un ou plusieurs conflits apparaissent au niveau de la politique de sécurité abstraite. L'utilisateur doit tout d'abord identifier ces conflits. Une fois le problème identifie, l'utilisateur a plusieurs solutions :

- **Modifier une des règles conflictuelles :** L'utilisateur peut considérer que le conflit détecté est dû à une erreur dans la spécification de la politique. Dans ce cas, il peut mettre à jour la politique en modifiant une des règles conflictuelles.
- **Ajouter une (ou plusieurs) contrainte(s) de séparation :** L'utilisateur peut ajouter des contraintes de séparation. Par exemple, en introduisant une contrainte de séparation entre les rôles *infirmière* et *médecin*, on a l'assurance que les privilèges de ces deux rôles ne peuvent plus rentrer en conflit.
- **Modifier le niveau de priorité d'une des règles conflictuelles :** Changer le niveau de priorité d'une règle est un moyen simple pour résoudre le conflit. Cependant, il faut s'assurer que cette modification n'a pas pour conséquence de rendre l'une des règles redondante ou inapplicable. Nous ne développons pas dans notre travail le problème des règles redondantes. Il s'agit d'un problème complexe et nous renvoyons à **[20]** où est défini un algorithme pour détecter ce type d'anomalie dans une politique de sécurité réseau.
- **Ignorer le conflit :** L'utilisateur peut tout simplement ignorer le conflit. Cependant tout conflit non résolu au niveau abstrait peut générer des conflits au niveau concret.

Cependant, on ne peut choisir cette dernière solution que si les trois premières sont impossible où elles n'ont pas mené à une résolution du conflit. Dans la section suivante nous allons voir l'approche utilisé pour résoudre le conflit, et nous allons utiliser la théorie définit auparavant (Voir section III.5). **[19]**

III.8. Principe de l'approche :

La détection des conflits effectifs ne peut se faire qu'au niveau concret, c'est-à-dire entre les autorisations *Est-permis* et *Est-interdit*, Un conflit existe s'il existe un *sujet s*, une *action a* et *objet o* tels que :

And ((Est_permisS.sujet)
(Est_permis0.objet) (Est_permisA.action)
(Est_interditS.sujet) (Est_interdictO.objet).

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 que moment de leurs occurrence. Ainsi, nous associons les niveaux de priorité aux autorisations abstraites afin de prévenir les conflits avant la dérivation des autorisations concrètes.

III.8.1. Niveau de priorité :

Quand un conflit se produit entre deux autorisations est-permis et est_interdit, notre proposition pour résoudre un tel conflit est d'associer ces faits avec les niveaux de priorité. Alors l'autorisation avec la plus haute priorité prend la précédence sur d'autres autorisations.

Pour ce but, nous introduisons un ensemble de niveaux de priorité dénote **P.** Nous assumons que **P** est associé avec une relation d'ordre partiel dénotée < tel que: Si **p1** et **p2** sont deux priorités, alors

- **p1 < p2** veut dire que **p2** est **plus haut** que **p1**.
- **P1 <> p2** signifie que les priorités **p1** et **p2** sont incomparables, c'est équivalent à:

  P1 <> p2 ↔ ¬ (p1 < p2) ∧ ¬ (p2 < p1)

La définition de **P** est une application dépendante, dans le sens où elle dépend de la stratégie utilisée pour dériver les conflits. L'ensemble de niveaux de priorité est utilisé pour donner la priorité aux permissions et aux interdictions dans le niveau abstrait et concret.

Donc, Nous ajoutons de nouveaux types d'autorisation afin d'introduire les niveaux de priorité.

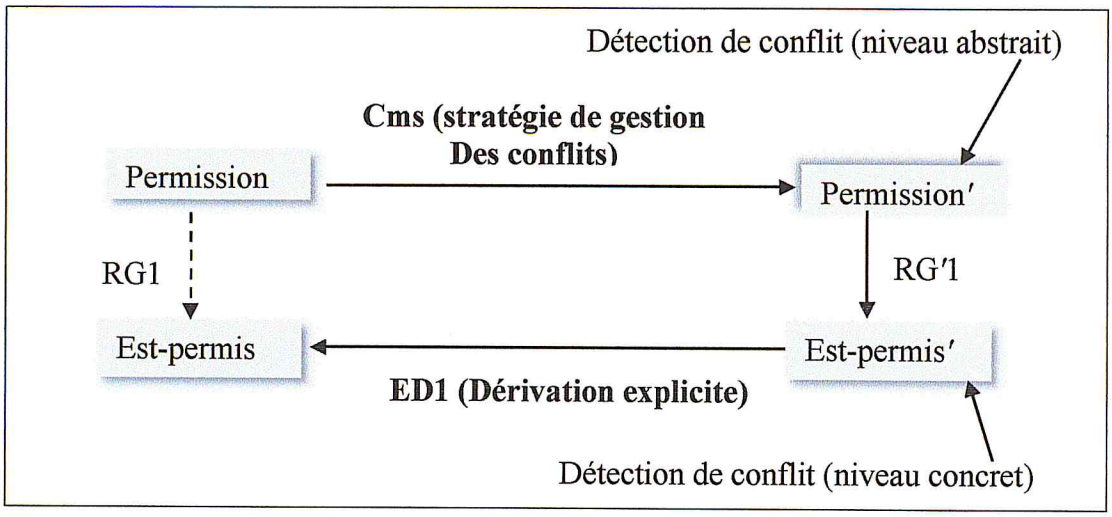


Figure VI.1 : La gestion des conflits dans Or-BAC.

Nous appelons Permission' et Interdiction' les nouvelles autorisations organisationnelles et *Est_perrmis'* et *Est_interdit'* les nouvelles autorisations concrètes. Elles sont de la forme :

III.8.1.1. Le concept Permission' :

"Permission' est une relation sur les domaines Org x R x A x V x P. Si org est une organisation, *r* un rôle, *v* une vue, *a* une activité et *p* un niveau de priorité, alors *δ Permission' (org, r, v, a, p) signifie que δ Permission (org, r, a, v)* est assigné au niveau de priorité *p*".

III.8.1.2 Le concept Interdiction' :

"*Interdiction'* est une relation sur les domaines Org x R x A x V x P. Si org est une organisation, *r* un rôle, *v* une vue, *a* une activité et *p* un niveau de priorité, alors *δ Interdiction' (org, r, v, a, p)* signifie que *δ Interdiction (org, r, a, v)* est assignée au niveau de priorité *p*".

III.8.1.3. Le concept Est-permis' :

"*Est_permis'* est une relation sur les domaines S x A x O x P. Si s est un sujet, *a* une action, *o* un objet et *p* un niveau de priorité, alors *Est_permis' (s, a, o, p)* spécifie que *Est_permis (s, a, o)* est assigné au niveau de priorité *p*."

III.8.1.4 Le concept Est-interdit':

" *Est_interdit'* est une relation sur les domaines S x A x O x P. Si s est un sujet, *a* une action, *o* un objet et *p* un niveau de priorité, alors *Est_ interdit' (s, a, o, p)* spécifie

que *Est_ interdit (s, a, o)* est assigné au niveau de priorité *p*."

III.9. Nouveau processus de dérivation:

Le processus de dérivation est composé de trois étape et représenter en pointillée dans **la figure III.1**.Le processus de dérivation des interdictions concrètes (*Est_interdit*) et des interdictions abstraites (*Interdictions*) est le même.

Nous expliquons en premier chaque étape du nouveau processus :

III.9.1. Premier étape: Politique de gestion de conflit « cms »

En premier, nous nous concentrons sur la spécification des niveaux de priorité. Quand une nouvelle permission (ou interdiction) est insérée dans la politique de sécurité, une première possibilité consiste à supposer que l'administrateur associe manuellement cette permission (Ou interdiction) avec un niveau de priorité. Une telle approche administrative est très complexe à gérer pour un administrateur de sécurité. Alors, nous suggérons de définir un ensemble de règles qui sont utilisées automatiquement pour dériver un niveau de priorité pour chaque permission (interdiction). Si quelques permissions ou interdictions ne peuvent pas avoir automatiquement une priorité, alors l'administrateur demandera d'assigner une priorité manuellement. **[16]**

Définition : Politique de gestion de conflit :

Une politique de gestion de conflit est un ensemble de règles appliquées sur des attributs de permission ou interdiction, il faut mentionner que les niveaux de priorité ne correspondent pas nécessairement aux nombres de priorité. Au contraire, les niveaux de priorité permettent de modéliser un grand nombre de politiques.

Considérons les exemples suivants:

- ✓ **Exemple 1: L'interdiction prend le privilège sur la permission (cms1)**

Dans ce cas quand une situation de conflit se produit, l'interdiction prend le privilège. **DTP (Denial Takes Precedence).** Laissez-nous désigner cela par **cms1**. C'est un exemple très simple de politique. Il est modélisé comme suit:

- P = {0, 1}
- $\forall$org $\in$Org, $\forall$r $\in$ R, $\forall$ a $\in$A, $\forall v \in$V,

δ Permission (org, r, a, v) => δ Permission' (org, r, a, v, 0)

- P = {0, 1}
- $\forall$org $\in$Org, $\forall$r $\in$ R, $\forall$ a $\in$A, $\forall v \in$V,

δ Interdiction (org, r, a, v) => δ Interdiction' (org, r, a, v, 1)

Il y a aussi le cas où la permission prend le privilège **PTP (permission takes precedence).**Ce cas est modélisé de la même façon que **cms1,** mais en inversant les priorités de permissions et d'interdictions.

✓ **Exemple 2: La première autorisation prend le privilège (cms2)**

Nous considérons particulièrement la première politique. Dans une telle politique, la première autorisation gagne. Cela inclut la supposition que l'ensemble d'autorisation est ordonné. Pour modéliser cette politique, nous supposons que **P=N** et que deux autorisations ne peuvent pas avoir le mème niveau de priorité. Dans cette politique, le niveau de priorité assigné doit être porté dehors par l'officier de la sécurité ou bien l'administrateur du systeme **SSO** (*System Security officer*) Si un conflit apparaît, l'autorisation ayant le plus haut niveau dans le sens de l'ordonnancement pris en considération, prend la précédence. Cette politique est dénotée **cms2**.

✓ **Exemple 3: La priorité basée sur le rôle (cms3)**

Dans cette politique, une relation de priorité entre rôles est définie pour résoudre des conflits **[21].** Cette relation de priorité est généralement compatible avec l'hiérarchie définie pour l'héritage de permission et d'interdiction entre rôles. Nous allons appeler cette politiques **cms3**. Cela est modélisé en utilisant un ensemble de niveaux de priorité **P = R** où **R** représente l'ensemble de rôles et l'ensemble **R** est associé à une relation d'ordre partiel. Alors si une permission ou une interdiction est assignée au rôle *r*, alors *r* représente aussi le niveau de priorité de cette permission ou interdiction.

- **P= R** et l'ensemble **R** est associé avec une relation d'ordre partiel.
- $\forall$org $\in$Org, $\forall$r $\in$ R, $\forall$ a $\in$A, $\forall v \in$V,

δ Permission (org, r, a, v) => δ Permission' (org, r, a, v, r)

- $\forall$org $\in$Org, $\forall$r $\in$ R, $\forall$ a $\in$A, $\forall v \in$V,

δ Interdiction (org, r, a, v) => δ Interdiction' (org, r, a, v, r)

III.9.2. Deuxième étape: Dérivation des autorisations concrètes à partir des autorisations abstraites (organisationnelles) :

Ici, nous nous concentrons sur la règle de dérivation qui nous permet de dériver l'autorisation Est_permis' (resp. Est_interdit ') de l'autorisation abstraite ayant une priorité (autorisation primée) Permission' (resp. Interdiction '). La règle **RG1**(resp. **RG2**) (voir. **III.5.7**) utilisée pour dériver des autorisations concrètes à partir d'autorisations abstraites dans la théorie logique ne sont plus valides.
RG1 et remplacé par la règle **RG1'** dans le nouveau processus de dérivation.

RG1' :

Habilite $\subseteq$ *(HabiliteS.s* $\cap$ *HabiliteR.r* $\cap$ *HabiliteOr.org).*
Considére $\subseteq$ *(ConsidéreAc.a* $\cap$ *ConsidéreAv.ac* $\cap$ *ConsidéreOr.org).*
Utilise$\subseteq$ *(UtiliseO.o* $\cap$ *Utilise V.v* $\cap$ *UlitiseOr.org).*
δ *Permission' (Org, r, a ,v, p)*
Est_permis' (s, a, o, p) $\subseteq$ ***(Habilite*** $\cap$ ***Utilise*** $\cap$ ***Considére*** $\cap$ δ ***Permission').***

Cette règle indique que *Est_permis'* peut être dérivé par le même niveau de priorité comme *Permission'*, pourvu que les autres conditions soient satisfaites. Une règle semblable appelée **RG**2' est appliquée pour dérivé *Est_interdit'* de *Interdiction'.*

RG2':

Habilite $\subseteq$ *(HabiliteS.s* $\cap$ *HabiliteR.r* $\cap$ *HabiliteOr.org).*
Considére $\subseteq$ *(ConsidéreAc.a* $\cap$ *ConsidéreAv.ac* $\cap$ *ConsidéreOr.org).*
Utilise$\subseteq$ *(UtiliseO.o* $\cap$ *Utilise V.v* $\cap$ *UlitiseOr.org).*
δ *Interdiction' (Org, r, a ,v, p)*
Est_interdit' (s, a, o, p) $\subseteq$ ***(Habilite*** $\cap$ ***Utilise*** $\cap$ ***Considére*** $\cap$ δ ***Interdiction').***

III.9.3. Troisième étape: Dérivation des permissions concrètes non primées ED1 :

Laissez-nous maintenant considérer la dernière phase du processus de dérivation qui correspond à la dérivation des autorisations concrètes Est-permis (resp. Est_interdit) des autorisations concrètes ayant une priorité Est_permis' (resp. Est_interdit'). L'idée générale de cette dérivation est la suivant:
Nous dérivons une autorisation concrète Est-permis pourvu que ce ne soit pas

possible de dériver une autorisation concrète Est-interdit ayant le plus haut niveau de priorité. Cela est modélisé par la règle **ED1.**

ED1 :

$$\forall s \in S,\ \forall a \in A,\ \forall o \in O,\ \forall p \in P,$$

$$Est\_permis'(s, a, o, p1) and \neg\exists\ p2 \in P\ ,\ (p1 < p2 \text{ and } Est\_interdit'(s, a, o, p2)) \Rightarrow Est\_permis(s, a, o)$$

Cette règle dit qu'une *Permission* concrète peut être dérivé pour permettre à un sujet *s* d'exécuter l'action *a* sur l'objet *o* si cette permission a un niveau de priorité *p1* et s'il n'y aucune *interdiction* pour *s* pour exécuter *a* sur *o* avec un niveau de priorité *p2* strictement plus haut que *p1*.La règle **ED2** pour les interdiction concrètes est définie dans un chemin semblable sauf qu'à la place de permis on met interdit et vice versa.

ED2 :

$$\forall s \in S,\ \forall a \in A,\ \forall o \in O,\ \forall p \in P,$$

$$Est\_interdit'(s, a, o, p1) and \neg\exists\ p2 \in P\ ,\ (p1 < p2 \text{ and } Est\_permis'(s, a, o, p2)) \Rightarrow Est\_interdit(s, a, o)$$

III.10. Politique de gestion de 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 (Conflict Management Strategy)** est définie. En Pratique, créer une politique de gestion des conflits consiste à définir un ensemble **P** de niveaux de priorités associe à un ordre partiel et à déterminer comment sont obtenues les autorisations primées.

Considérons les deux exemples suivants :

Politique 1 :

P = {10, 1}

- *δ Permission (org, r, a, v) → δ Permission' (org, r, a, v, 0)*
- *δ Interdiction (org, r, a, v) → δ Interdiction' (org, r, a, v, 1)*

Politique 2:

P =R, l'ensemble de rôles R est associe à un ordre partiel.

- *δ Permission (org, r, a, v) → δ Permission' (org, r, a, v, r)*
- *δ Interdiction (org, r, a, v) → δ Interdiction' (org, r, a, v, r)*

La première politique signifie que l'interdiction prend toujours le privilège **DTP** (***Denial Takes Precedence***). Il existe aussi le cas où les permissions prennent le privilège **PTP** (***Permission Takes Precedence***), celle-ci est définie par analogie avec **DTP** en inversant les permissions et les interdictions.

Dans la seconde politique, le niveau de priorité d'une autorisation est le *rôle* impliqué dans cette autorisation **[21]**, cette relation de priorité est généralement compatible avec la hiérarchie définie pour l'héritage des permissions et des interdictions entre rôles. Ceci est modélisé en utilisant un ensemble de niveaux de priorité **P = R** où **R** représente l'ensemble de rôles, associé à une relation d'ordre partielle.

Alors si une permission ou une interdiction est assignée au rôle *r*, donc *r* représente aussi le niveau de priorité de cette permission ou bien l'interdiction, qui veut dire que le rôle le plus prioritaire est assigné à la permission ou bien à l'interdiction.

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 de conflits doit permettre de dériver les autorisations abstraites primées (permission', interdiction') 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 de 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, **La Politique 1** est une politique efficace, alors que **La Politique2** 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.

III.11. Prévention de conflits :

Si la politique est efficace, alors la gestion de conflits est parfaite. En revanche, si elle est faible, des conflits peuvent encore apparaitre. 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 entrainer 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 en 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 *Interdiction'*.

L'objectif étant dans ce cas de pouvoir certifier que s'il n'existe pas de conflits au niveau abstrait, alors il n'y aura pas de conflits au niveau concret, et ce, quels que soient les sujets, les actions et les objets.

Ceci est réalisé par la détermination d'une condition particulière. Si cette condition 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 abstraites de la permission et de l'interdiction. **[16]**

III.11.1. Prévention du conflit : Première proposition

Nous suggérons une première condition pour prévenir des conflits au niveau Abstrait d'OrBAC. II est obtenu en remplaçant simplement les autorisations concrètes par les autorisations abstraites dans la condition $C_{conflit}$ C'est donc défini comme suit **[16]** :

$C'_{conflit} \subseteq$

δ Permission' (org1, r1, a1, v1, p1) and δ Interdiction' (org2, r2, a2, v2, p2) and

$\exists p3 \in P$, (p1 < p3 and δ Interdiction' (org2, r2, a2, v2, p3)) and

$\exists p4 \in P$, (p2 < p4 and δ Permission' (org1, r1, a1, v1, p4)).

III.11.2. Prévention du conflit: Deuxième proposition :

Cette section suggère une condition plus faible que $C'_{conflit\ 1}$ pour prévenir des conflits. Elle est définie comme suit **[16]:**

$C'_{conflit\ 2} \subseteq$

δ Permission' (org1, r1, a1, v1, p1) and δ Interdiction' (org2, r2, a2, v2, p2) and

$\exists p3 \in P$, (p1 < p3 and δ Interdiction' (org2, r2, a2, v2, p3)) and

$\exists p4 \in P$, (p2 < p4 and δ Permission' (org1, r1, a1, v1, p4))and

$\exists$ Séparation_activité $\subseteq$ (Séparation_ activitéOr.org1 $\cap$ Séparation_activitéAv.a1 $\cap$ Séparation_activitéOr.org2 $\cap$ activitéAv.a2).

$\exists$ Séparation_rôle $\subseteq$ (Séparation_rôleOr.org1 $\cap$ Séparation_rôleR.r1 $\cap$ Séparation_rôle Or.org2 $\cap$ Séparation_ rôle R.r2).

$\exists$ Séparation_vue $\subseteq$ (Séparation_vueOr.org1 $\cap$ Séparation_vueV.v1 $\cap$Séparation_vueOr.org2 $\cap$ Séparation_vueV.v2)

III.11.3. Prévention du conflit: Dernière proposition :

Notre dernière proposition pour la prévention de conflit est définie comme suit **[16]:**

$C'_{conflit\ 3} \subseteq$

δ Permission' (org1, r1, a1, v1, p1) and δ Interdiction' (org2, r2, a2, v2, p2) and

$\exists$ Séparation_activité $\subseteq$ (Séparation_ activitéOr.org1 $\cap$ Séparation_activitéAv.a1 $\cap$ Séparation_activitéOr.org2 $\cap$ activitéAv.a2).

$\exists$ Séparation_rôle $\subseteq$ (Séparation_rôleOr.org1 $\cap$ Séparation_rôleR.r1 $\cap$ Séparation_rôle Or.org2 $\cap$ Séparation_ rôle R.r2).

$\exists$ Séparation_vue $\subseteq$ (Séparation_vueOr.org1 $\cap$ Séparation_vueV.v1 $\cap$Séparation_vueOr.org2 $\cap$ Séparation_vueV.v2)

$\exists p3 \in P, (I, j, k, l)$

((p1 < p3 and δ Interdiction' (orgi, rj, ak, vl, p3)) and

(p2 < p3 and δ Permission' (orgi, rj, ak, vl, p3))).

Ainsi, pour détecter et résoudre le conflit dans notre modèle Or-BAC, nous avons utilisé cette règle car elle est la plus récapitulative des règles et des conditions que nous avons vue jusqu'à maintenant, elle traite aussi le cas général, et alors nous

III.12. Conclusion :

Dans ce chapitre, nous avons défini une méthode pour gérer les éventuels conflits. Cette méthode présente un avantage considérable car elle permet de spécifier *une politique de gestion de conflits* paramétrable par l'administrateur de sécurité, ce qui offre une grande souplesse.

Nous avons définis 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.

Nous avons aussi présenté une approche logique pour gérer des conflits dans une politique de contrôle d'accès modélisée dans Or-BAC. Puisque une politique dans Or-BAC est définie au niveau organisationnel (c.-à-d, indépendamment de la mise en œuvre réelle des sujets, des objets et des actions dans le système), nous suggérons aussi de gérer des conflits au niveau abstrait. Notre approche est basée sur la définition d'une politique de gestion de conflits paramétrable qui est utilisée pour assigner les niveaux de priorité aux permissions ou aux interdictions abstraites. Pour le faire, nous redéfinissons le processus de dérivation entre les autorisations abstraites et concrètes. Deux différentes situations peuvent survenir en utilisant notre politique :

(1) La politique est dite efficace, cela garantit que la stratégie résoudra chaque conflit qui peut se produire dans la politique.

(2) si la politique est dite faible, la résolution de conflit est impossible.

Chapitre IV

Implémentation

IV.1.Introduction :

Après avoir achevé la phase théorique qui englobe les notions nécessaires à la réalisation de notre modèle DL_OrBACδε (Les principes de fonctionnement du processus de détection et de réalisation de conflits dans le modèle OrBAC en utilisant la logique de description non monotone), dans ce chapitre nous allons aborder la phase pratique (implémentation), c'est la partie où nous allons faire une description de toutes les étapes de l'étude, en les illustrant à l'aide des interfaces graphiques.

Mais avant cela nous présentons brièvement les différents outils de développement utilisés.

IV.2. Outil de développement :

Notre travail a été conçu dans un environnement Windows 7, en utilisant le langage de programmation java pour implémenter les différents algorithmes présentés dans le chapitre précédent, et Microsoft Office Access 2010 pour la persistance de notre base de connaissance.

IV.2.1. Java :

Le langage java est un langage de programmation informatique orienté objet créé par **James Gosling** et **Patrick Naughton** employés de Sun Microsystems avec le soutien de **Bill Joy** (cofondateur de Sun Microsystems en 1982), présenté officiellement en 1995.

Le langage Java a la particularité principale que les logiciels écrits avec ce dernier sont très facilement portables sur plusieurs systèmes d'exploitation tels que Linux, avec peu ou pas de modifications… C'est la plate-forme qui garantit la portabilité des applications développées en Java.

Nous avons choisi NetBeans (**Netbeans 6.8**) comme IDE pour java.

IV.2.2. NetBeans :

NetBeans est un environnement de développement intégré (IDE) open source. Il est développé par Sun. En plus de Java, NetBeans permet également le développement avec d'autres langages tels que: C++, XML, PHP...

Il comprend toutes les caractéristiques d'un IDE moderne (coloration syntaxique, éditeur graphique d'interfaces et des pages web, etc.).

IV.3.Présentation de l'application :

Notre programme fournit une interface conviviale. Il nous permet de spécifier les politiques du modèle OrBAC, Détecter et résoudre des conflits.

Ci-dessous nous présentons les diverses figures qui schématisent notre application :

IV.3.1.La fenêtre principale :

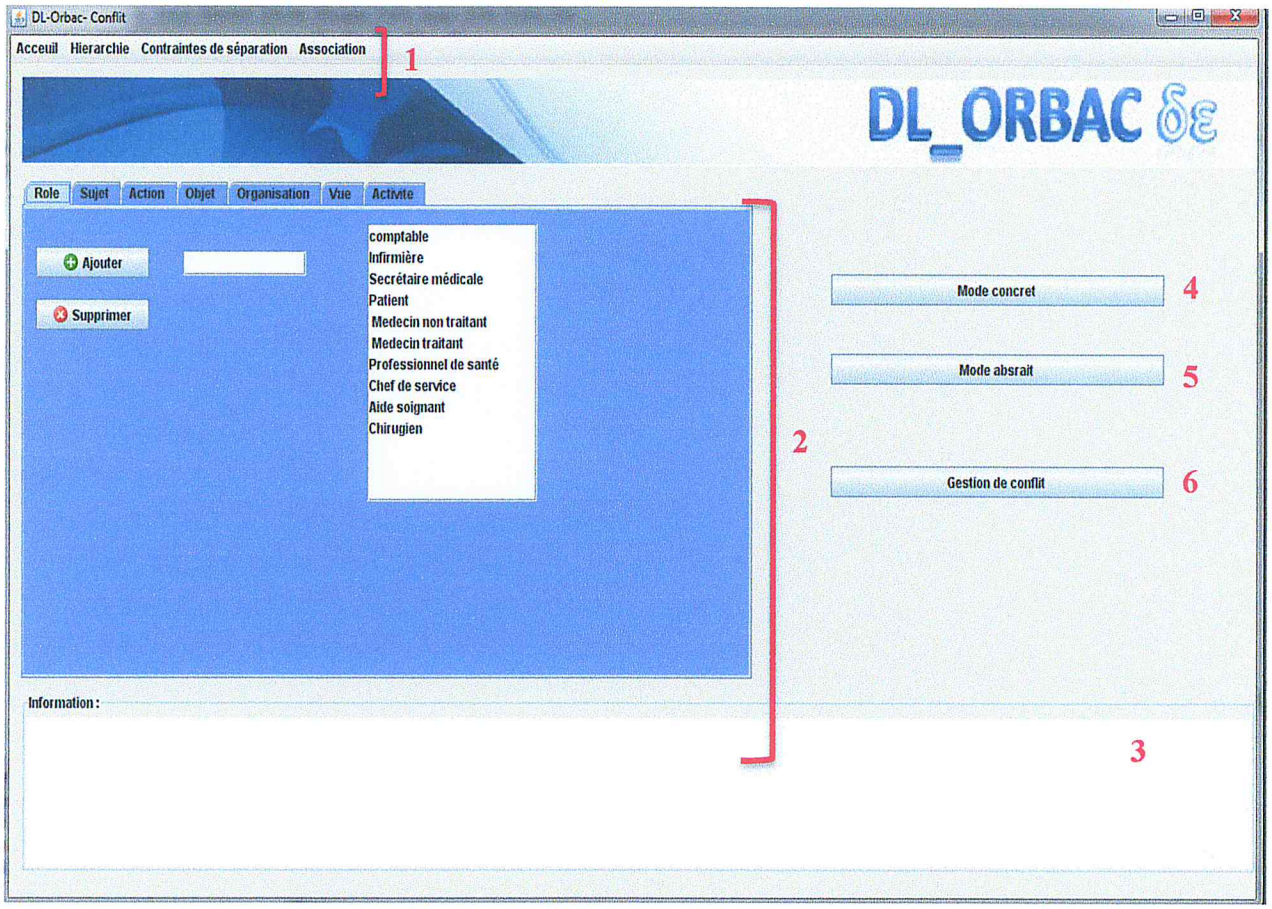


Figure IV.1 : La fenêtre principale du DL_OrBAC.

(1) Le Menu Principale :

Figure IV.2 : Menu principal.

- **Hiérarchie :** Permet à l'administrateur d'établir les hiérarchies de rôles et d'organisations dans le système.
- **Contrainte de séparation :** Permet à l'administrateur d'établir les séparations de rôles et d'organisations dans le système.
- **Association :** Permet d'instancier les différentes relations d'OrBAC (Habilite, Considère et Utilise) en liant les différents acteurs de la base de connaissances d'OrBAC.

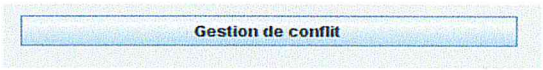
(2) **Base Abox editor :** Ce composant permet d'instancier les composants de base d'OrBAC, d'afficher les instances de chaque concepts, et de de supprimer des instances.

(3) **Information :** afficher les déférentes opérations effectuées au cours de la session en cours sur la base de connaissance.

(4) **Mode concret :** Se composant est utilisé pour afficher les permissions et les interdictions concrètes inférées est_permis et est_interdit.

(5) **Mode abstrait :** Se composant est utilisé pour ajouter et afficher les permissions et les interdictions abstraites inférées permission et interdiction.

(6) **Processus de gestion de conflit :** se composant sera en mode activé lorsqu'un conflit se détecte.



IV.3.2.Enrichissement de la base de connaissance à travers le DL-OrBac :

L'enrichissement de la base de connaissance se fait de manière directe ou indirecte, directement à travers l'interface du DL-OrBac, et indirectement à travers le module d'inférence.

L'enrichissement direct peut être :

- L'instanciation d'un concept d'OrBac.
  - L'établissement des relations entre les individus d'Orbac.
  - La définition d'une permission abstraite.
  - L'établissement d'une hiérarchie de rôle ou d'une hiérarchie d'organisation (la hiérarchie peut être construite indirectement).

- L'établissement d'une séparation de rôle, de vue ou d'activités (la séparation peut être construire indirectement).

➢ L'enrichissement se fait à travers :

- La Base Abox editor pour les concepts de base.
- Les fenêtres d'associations pour l'instanciation des relations(habilite, considère, utilise).
- La fenêtre permission abstrait pour l'édition des permissions.
- Les fenêtres hiérarchie de rôle, hiérarchie d'organisation pour l'établissement des hiérarchies.
- Les fenêtres de séparations de rôle, séparation de vue, séparation de d'activité pour l'établissement des séparations.

La hiérarchie d'organisation :

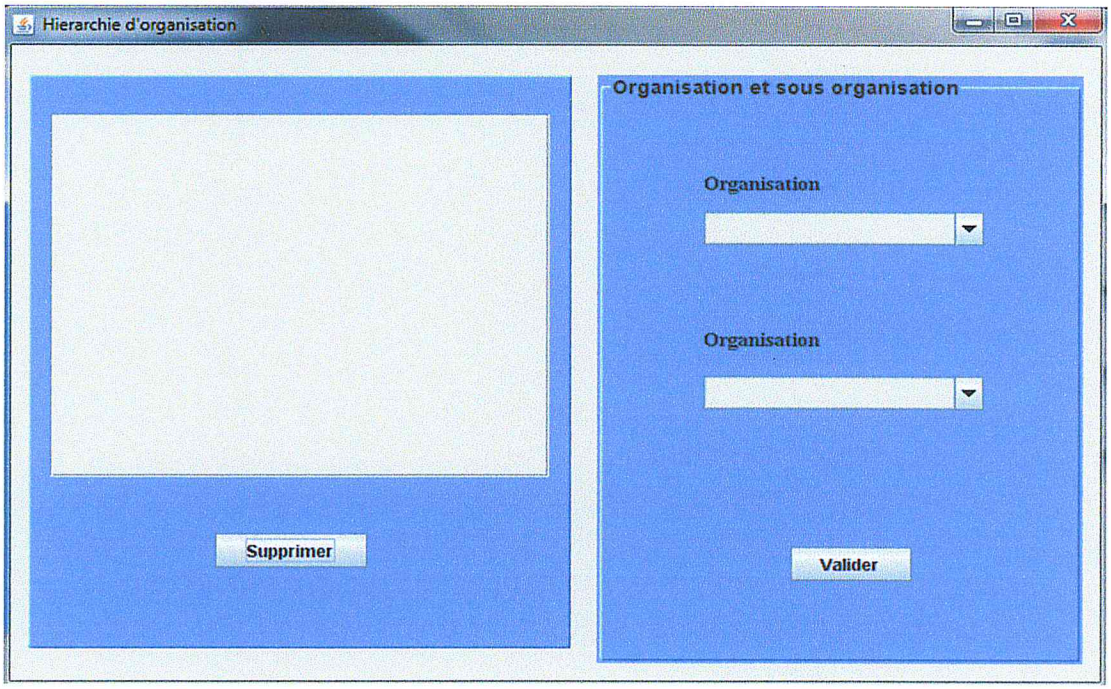


Figure IV.3 : La fenêtre hiérarchie des organisations

La hierarchie de role :

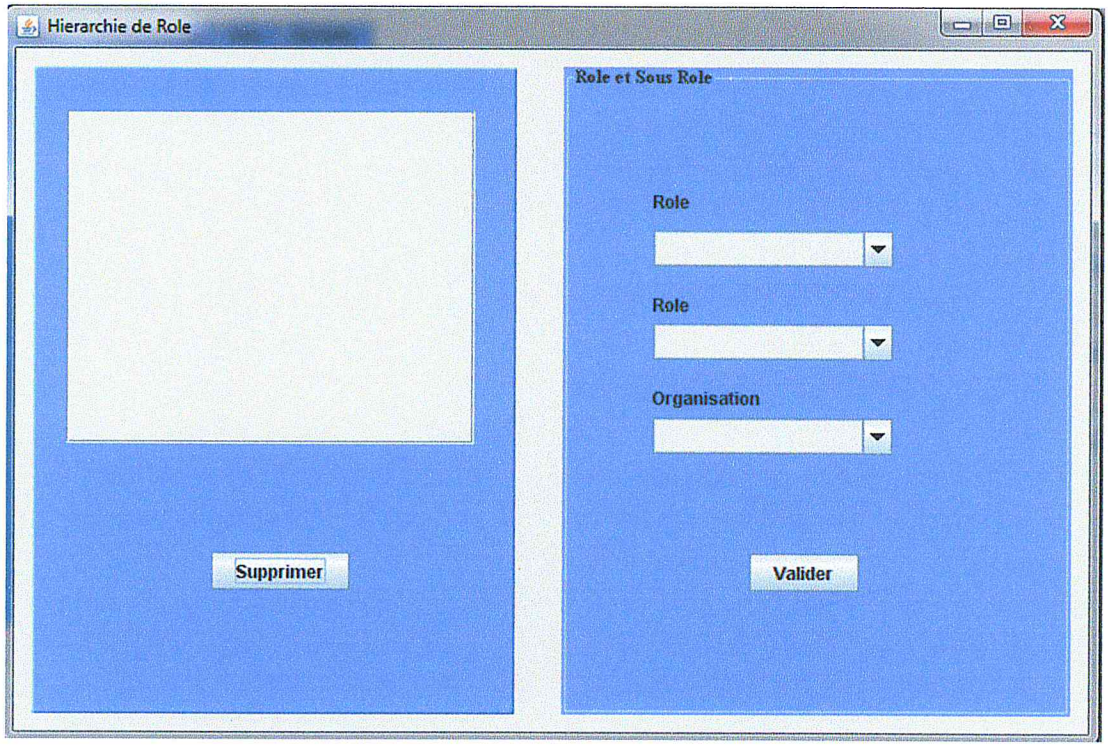


Figure IV.4 : La fenêtre hiérarchie de rôle

La séparation de rôle :

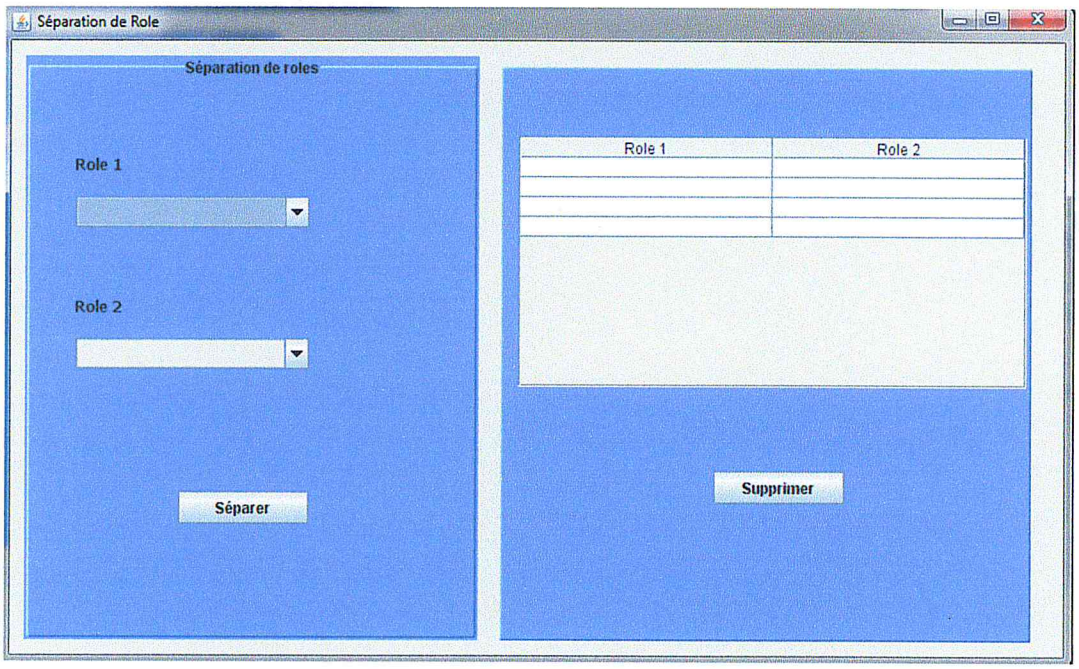


Figure IV.5 : La fenêtre de séparation des rôles.

La séparation de Vue :

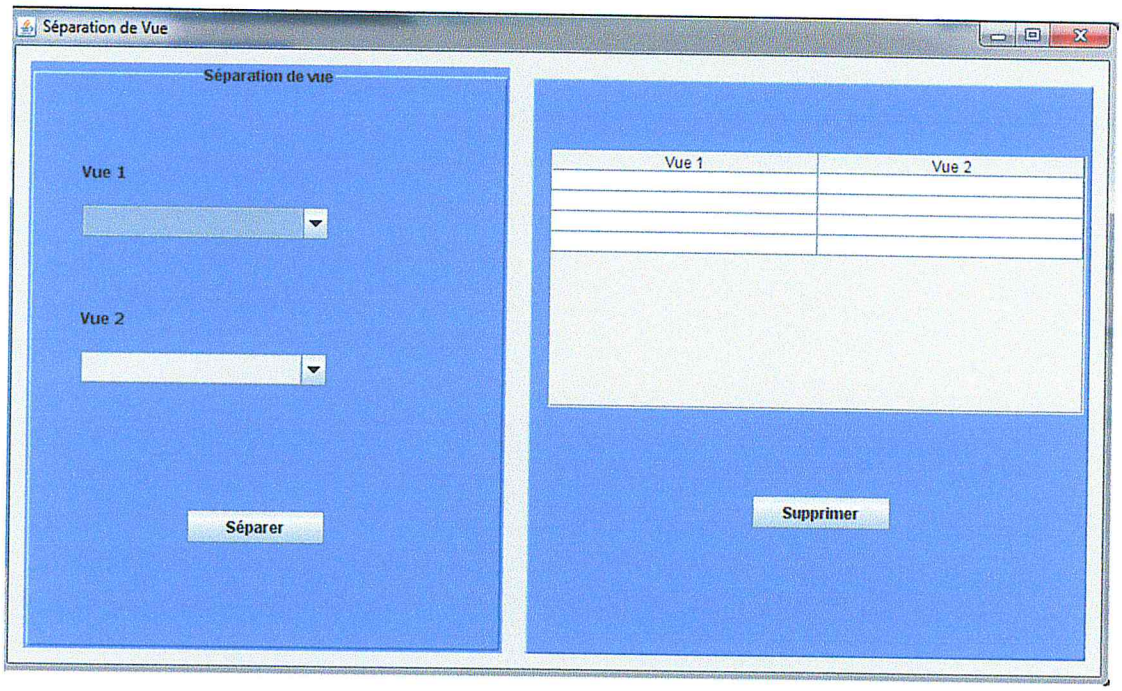


Figure IV.6 : La fenêtre séparation des vues.

La séparation d'activité :

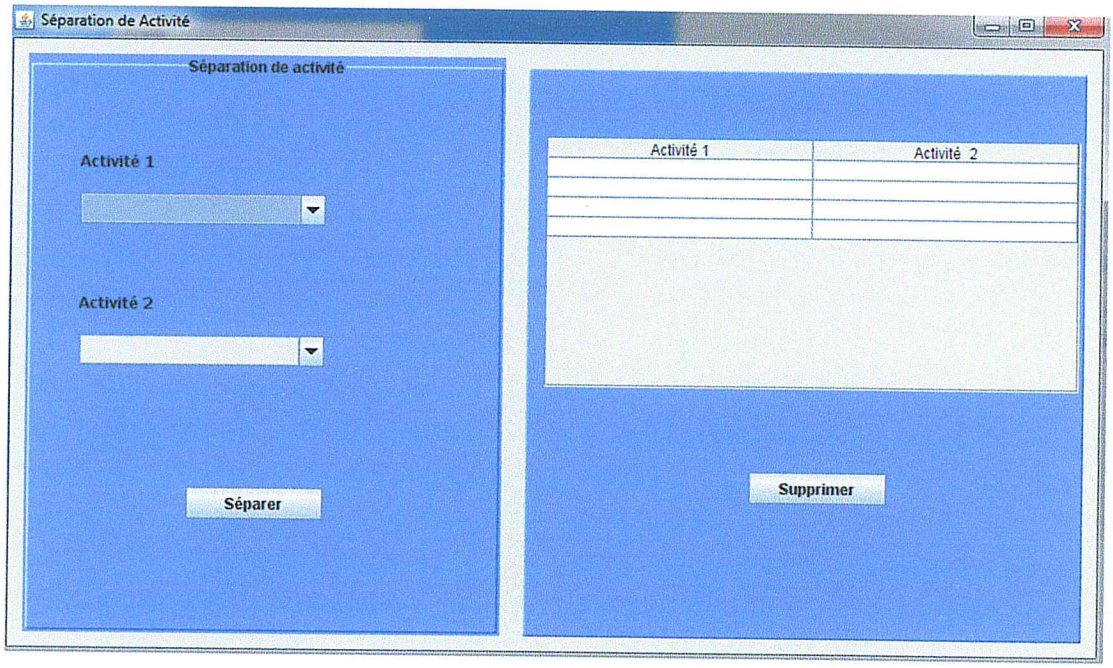


Figure IV.7 : La fenêtre séparation des activités

La relation utilise :

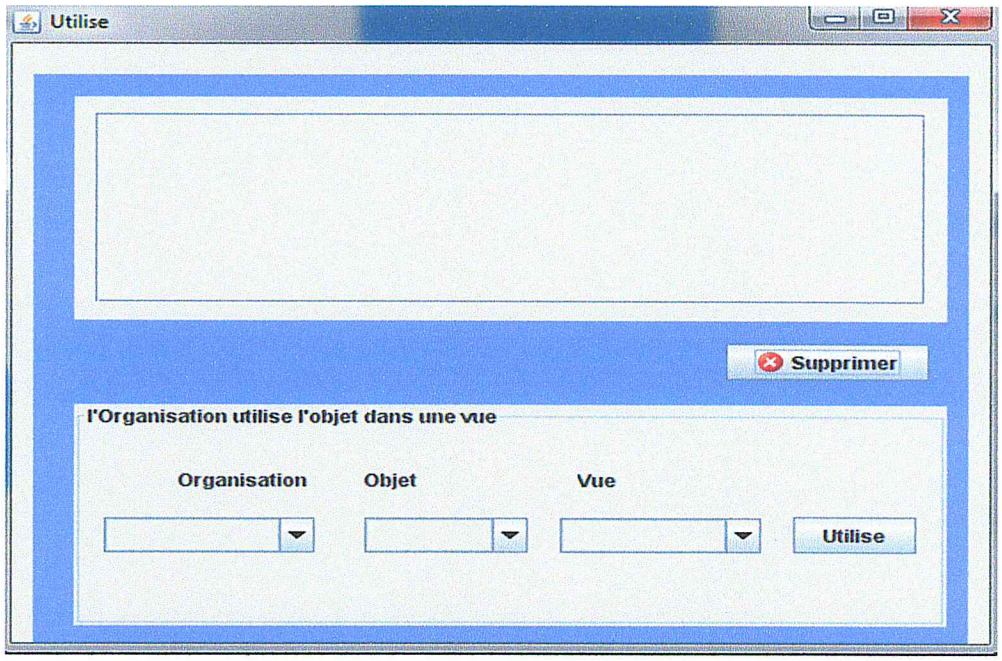


Figure IV.8 : La fenêtre Utilise.

La relation habilite :

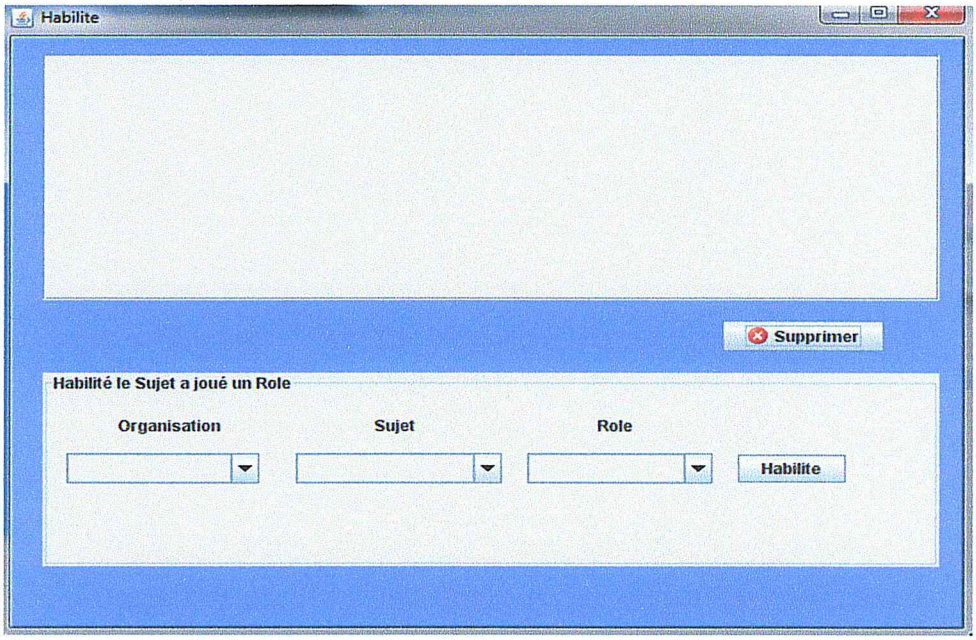


Figure IV.9 : La fenêtre Habilite.

La relation considère :

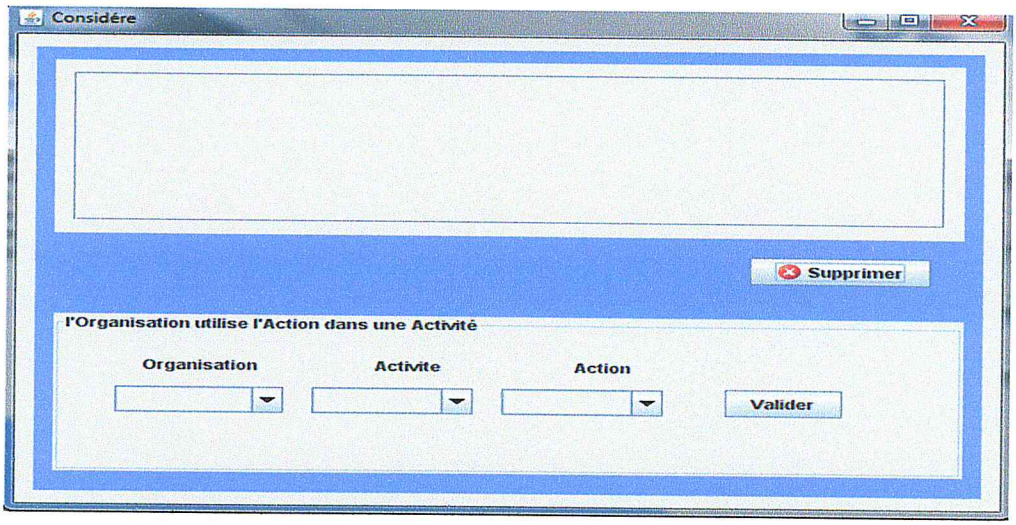


Figure IV.10 : La fenêtre Considère.

Le mode abstrait « défaut » :

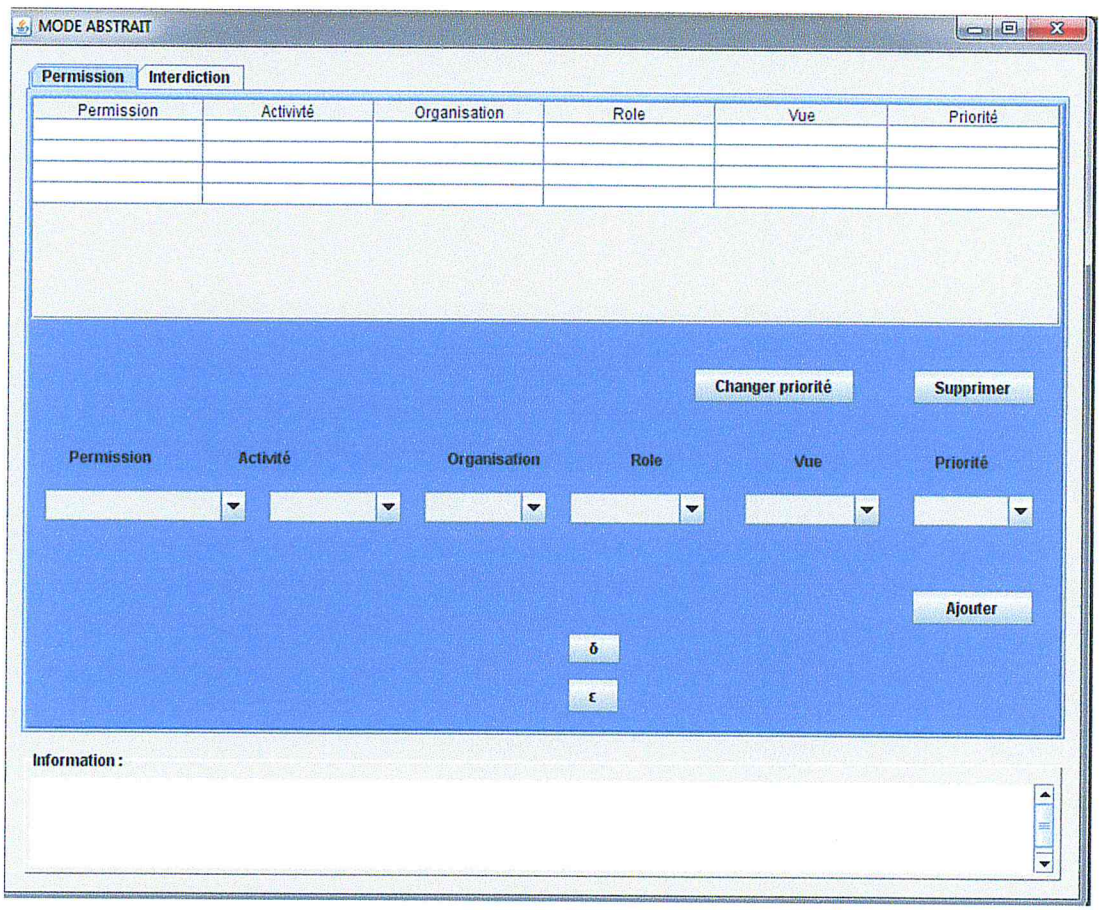


Figure IV.11 : La fenêtre mode abstrait défaut .

Le mode abstrait « exception » :

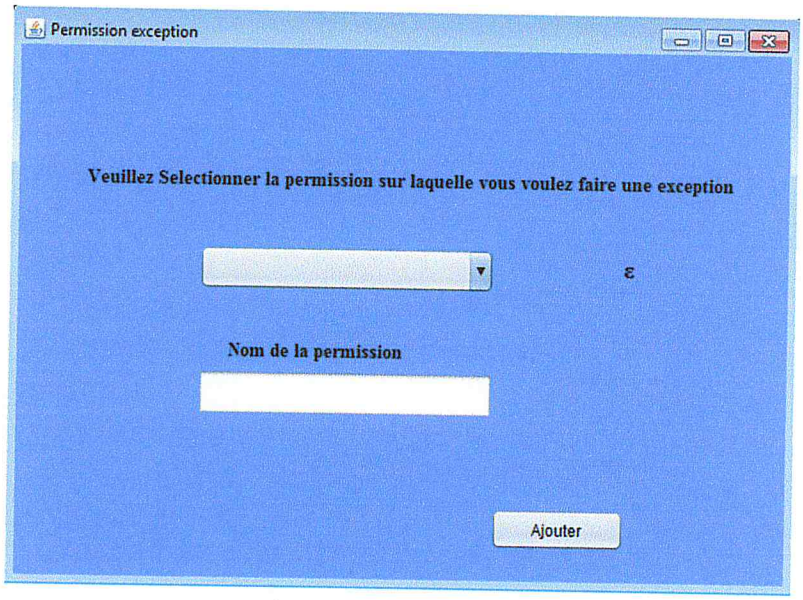


Figure IV.12 : La fenêtre d'ajouter une permission exception.

Le mode concret :

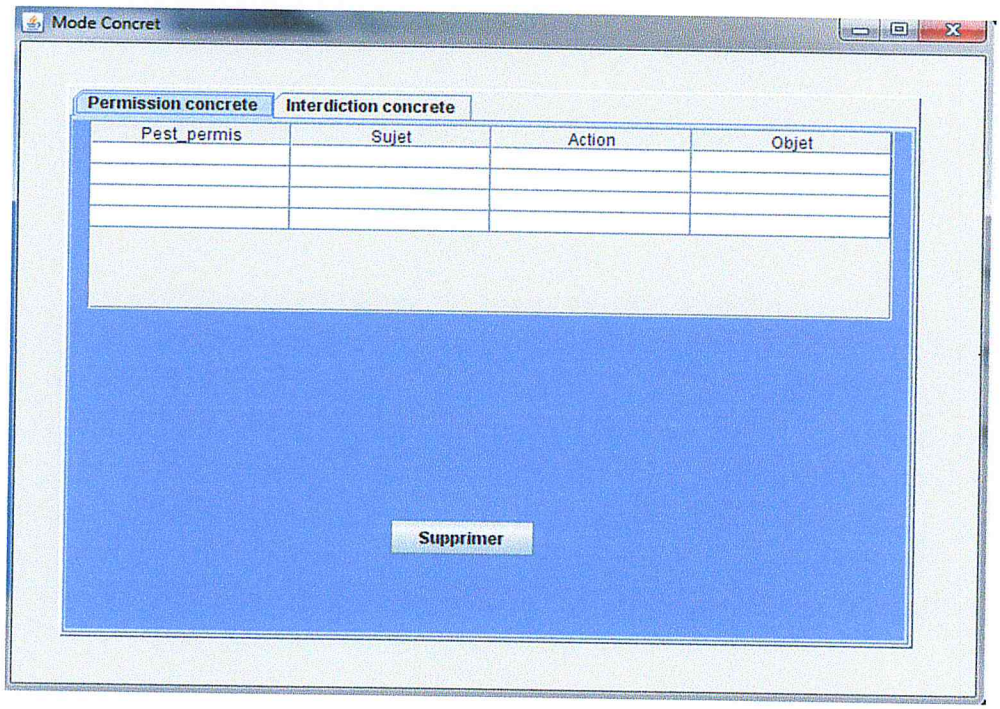


Figure IV.13 : La fenêtre mode concret

IV.3.3.Processus de dérivation :

Nous présentons ici le nouveau processus de dérivation, lorsque la condition de la règle de dérivation **RG'1** (resp **RG'2**) décrite dans les chapitres précédents est satisfaite, des éléments d'une permission (resp interdiction) concrète (sujet, Action, objet, priorité).

S'affichent sur le tableau (1) en cliquant sur le bouton (2) de la figure, une permission (resp interdiction) concrète se dérive et s'affiche le tableau (3) de la figure.

Quand un conflit se détecte avec une interdiction concrète (permission concrète), Une information de ce conflit s'affiche sur l'onglet (6) de la figure, ainsi pour résoudre cet éventuel conflit, l'administrateur de sécurité change le niveau de priorité en choisissant une priorité et en cliquant sur le bouton « changer priorité ».Lorsque la politique de sécurité est saine, l'administrateur pourra dérivé Est_pemris (resp Est_interdit) par la régle ED1 en cliquant sur le bouton « Dériver est_permis » (resp « Dériver est_interdit) qui sera utilisée pour le control d'accès.

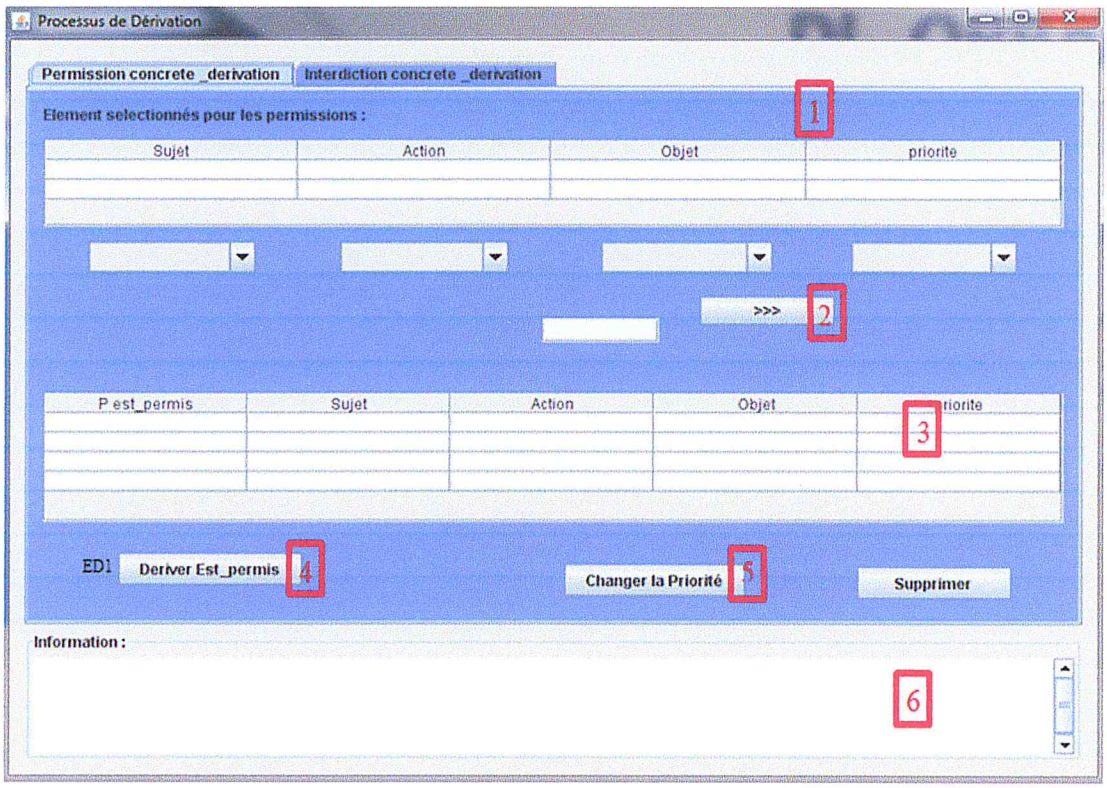


Figure IV.14 : La fenêtre Processus de dérivation.

IV.3.4. Gestion de conflit :

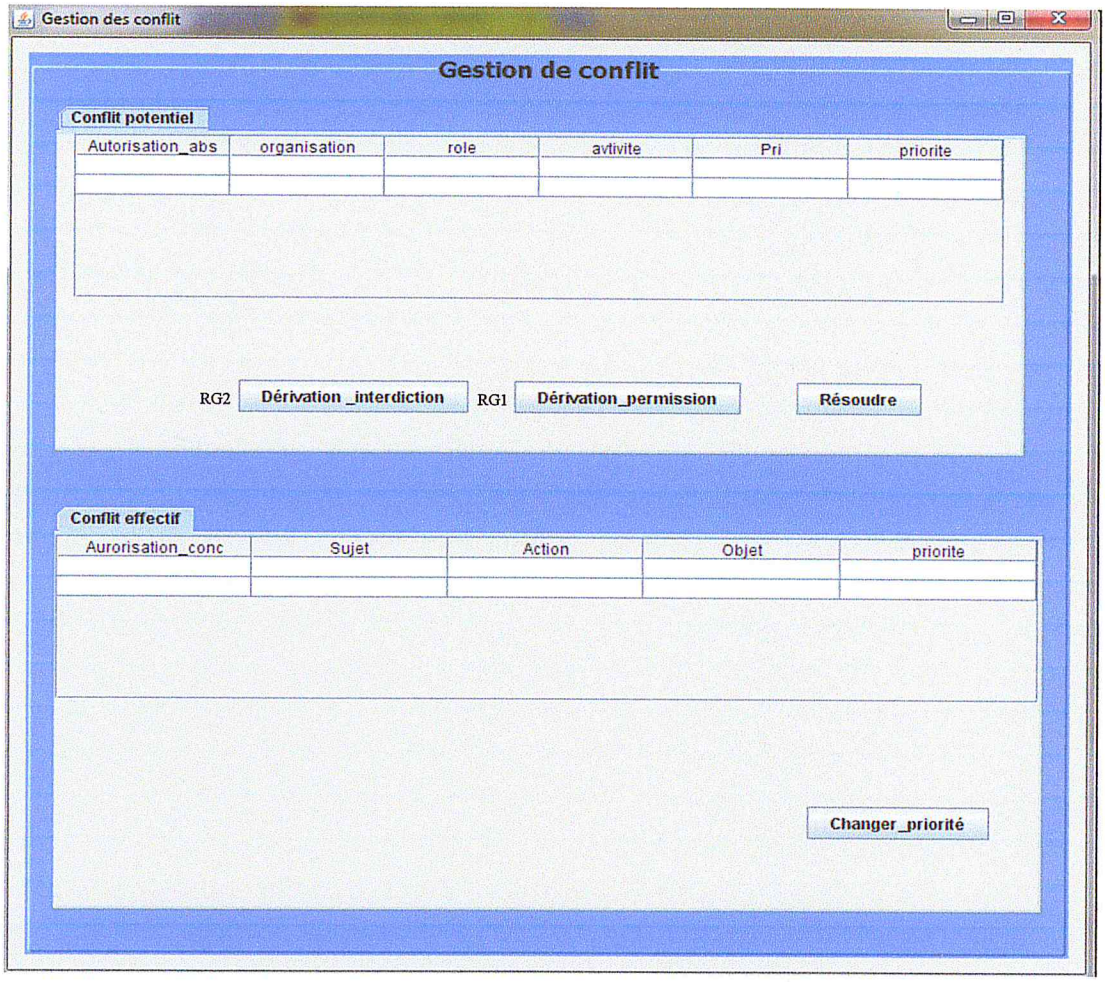


Figure IV.15 : La fenêtre Gestion des conflits.

La figure représente les processus de getions de conflits,le tableau supérieur représente les éventuels conflits détectés entre les autorisations abstraites dans le système. Pour résoudre ces conflit, l'administarteur clique sur le bouton « **résoudre** », la fenetre « **Solution** » s'affiche en demandant à l'administrateur de sécurité d choisir une ou plusieurs des solutions proposées.

Solution :

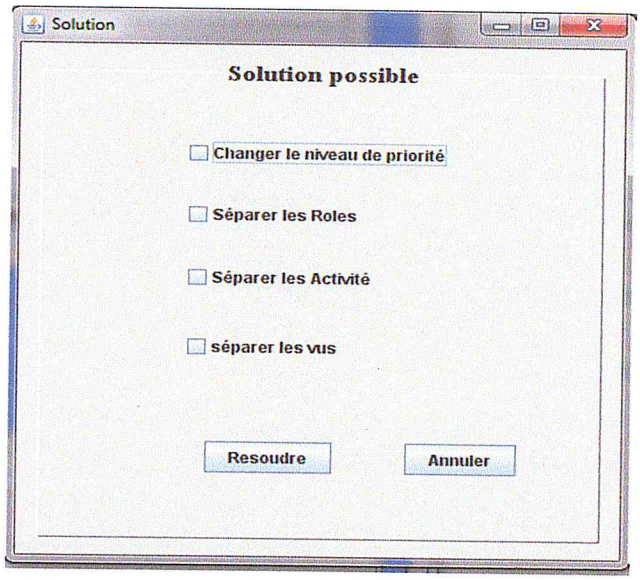


Figure IV.16 : La fenêtre solution

*Si l'administrateur choisis de changer de priorité, la fenetre des autorisations abstarites s'ouvre avec l'avtivation de bouton « **Changer priorité** » et du ComboBox « **priorité** ».

*Si l'administrateur choisis de séparer les roles , les vues ou les activités, les fenetres des différents séparations apparaissent.

Si les derniers solutions ne résolent pas de conflit détecté, il sera indispensable de dériver une des autorisations abstraites qui introduisent le conflit vers le mode concret en cliquant sur le bouton « **Dérivation interdiction** » ou le bouton « **Dérivation permission** » de la **figure *IV.15***.

En cliquant sur un de ces boutons, la fenêtre de la dérivation s'ouvre :

L'administrateur sélectionne un sujet, un objet, et une action pour dériver une autorisation concrète, en cliquant sur le bouton « **habilite+utilise+considère** », une dérivation s'active par l'aplication des régles **RG'1** et **RG'2**, en appliquant le processus de dérivation, le conflit potentiel se résolus.

Dérivation en mode concret :

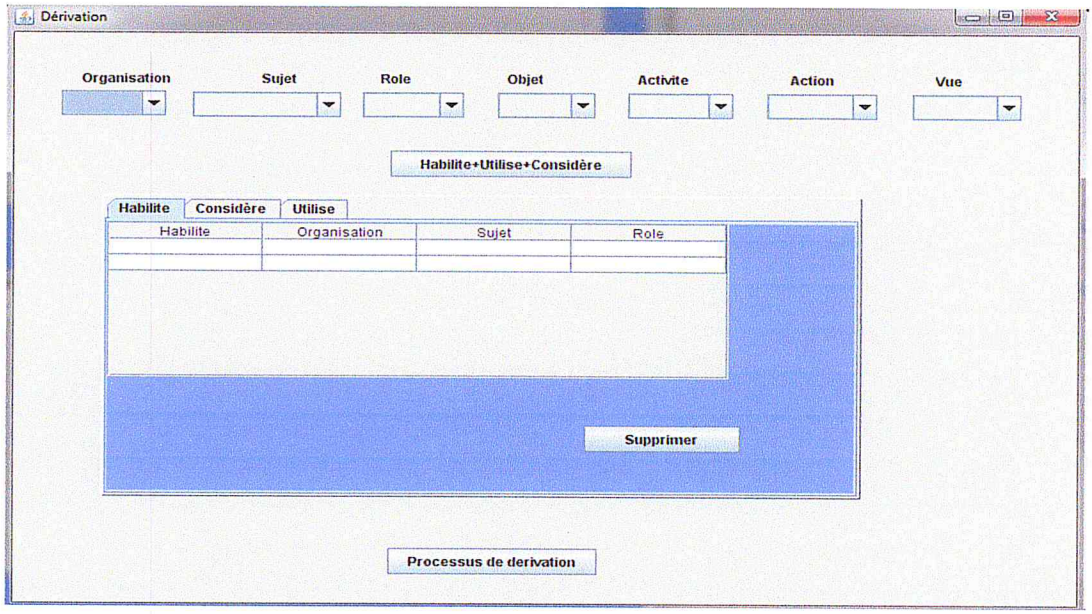


Figure IV.17 : La fenêtre Dérivation en mode concret.

Si le système détecte un conflit effectif , ce dernier s'affiche sur le tableau en bas de la **Figure *VI.15***,il n'y a qu'une seule solution que l'administrateur doit effectuer est celle du changement de priorité qui se réalise de la même manière que le conflit potentiel.

IV.4. Conclusion :

Dans ce chapitre nous avons appliqué le modèle de contrôle d'accès DL-OrBACδε qui regroupe deux parties :

La première consiste en la création et la manipulation des différentes entités qui composent le système d'information. Nous avons détaillé les différentes opérations d'ajout et suppression ainsi que les relations *Habilite, Considère, Utilise*.

La deuxième partie *« Gestions des conflits »* qui résoudre les conflits (Processus de dérivation, Séparations des entités et Niveau de priorité).

Conclusion générale

Conclusion général

Assurer la sécurité est une des plus importantes préoccupations des chercheurs. Il est important de pouvoir contrôler les flots d'informations dans les réseaux et dans les systèmes d'information. Il convient de développer au sein des systèmes informatiques des mécanismes permettant de filtrer les accès afin de ne laisser passer que ceux autorisés. Il s'agit pour cela de définir une politique de sécurité, c'est-à-dire la caractérisation des accès. Sa conception et son développement doivent être menés de manière à garantir sa fiabilité et sa sûreté. En effet, toute faille au sein de ce programme pourrait entrainer des violations de la politique de sécurité.

Le travail présenté dans ce manuscrit concerne la réalisation d'un modèle de contrôle d'accès dynamique contextuel formalisé avec la logique de description augmentée avec les opérateurs défaut (δ) et exception (ε) appelé DL-OrBACδε.

Les deux principales phases qui constituent notre modèle ont été détaillées, elles concernent :

- ✓ La création de la base de connaissances du modèle et les inférences sur cette base. Nous nous sommes intéressées aux deux mécanismes d'inférence : la subsomption qui permet la classification des concepts dans la hiérarchie et l'héritage qui permet de retrouver les propriétés héritées pour chaque concept à partir des concepts qui le subsument.
- ✓ La manipulation des différents entités créées dans la phase précédentes dans un cadre précis, dans notre cas notre modèle DL-OrBACδε.

Le modèle DL-OrBACδε s'est inspiré d'un modèle qui existe déjà qui est la modèle OrBAC.

Le modèle OrBAC est centré sur le concept d'organisation. En effet, tous les autres concepts de celui-ci (rôle, activité, vue, sujet, action, objet) dépondent d'une organisation donnée. En s'appuyant sur ces concepts, nous pouvons exprimer une politique de sécurité comme un ensemble de permissions, d'interdictions, d'obligations et de recommandations. DL-OrBACδε traite que les permissions en supposant que tous ce qui n'est pas permis est interdit.

La logique de description a été choisie comme formalisme de représentation.

DL-OrBACδε est basé sur les LD non monotones $AL\delta\varepsilon$. Cette logique a le pouvoir de définir des propriétés défauts au sein même d'une description d'un concept. Par contre, $AL\delta\varepsilon$ possède quelques inconvénients qui l'ont empêché d'être utilisée dans le cadre pratique qui

sont la faible puissance d'expressivité et le manque d'algorithmes permettant la subsomption et l'héritage qui font évoluer la base de connaissances. Ce dernier a été le point essentiel de notre projet.

Or-BAC offre une structure expressive et flexible pour concevoir des politiques de sécurité dynamiques, il permet aussi d'exprimer des interdictions, aussi appelées autorisations négatives. Spécifier une politique qui inclut des permissions et des interdictions peut mener à des situations de conflit.

Notre travail consistait à développer un module de gestion et de résolution de conflit et de l'introduire à l'application développée précédemment dans un travail de PFE.

Notre travail permet de détecter un conflit dès son apparition dans la base de connaissance et propose aussi de le résoudre selon différentes stratégies pour assurer une cohérence de la base de connaissance. La résolution du conflit se fait au niveau abstrait mais si on se trouve dans l'impossibilité de le résoudre à ce niveau, les autorisations concrètes sont dérivées et la résolution du conflit se fera au niveau concret.

Pour illustrer notre modèle de contrôles d'accès, nous avons choisi une application médicale.

Cette étude nous a permis de mieux comprendre l'utilité de la logique de description non monotone (défaut (δ) et exception (ε)) et son application dans le cadre pratique tel qu'un modèle de contrôle d'accès.

Bibliographie

Bibliographie

[1] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, "*Role-Based Access Control Models*", IEEE Computer, vol. 29, n° 2, pp.38-47, Février, 1996.

[2] ITSEC, "*Information Technology Security Evaluation Criteria*", v 1.2, 136 pp., ISBN 92-826-3005-6, Office des publications officielles des Communautés Européennes, Luxembourg, 1991.

[3] A. Abou El Kalam, R.El Baida, P.Balbiani, S.Benferhat, F.Cuppens, Y. Deswarte, A.Miège, C.Saurel et G.Trouessin, "*Organization Based Access Control*". IEEE 4th International Workshop on Policies for Distributed Systems and Networks (Policy 2003), Lake Come, Italy, June 4-6, 2003.

[4] F.Cuppens and A. Miège, "*Administration model for OrBAC*". In International Federated Conferences (OTM'03), Workshop on Metadata for Security. Catania. Sicily,Italy, November 3-7, 2003.

[5] A. Abou El Kalam, P.Balbiani, S.Benferhat, F.Cuppens, Y.Deswarte, R.El-Baida, A.Miège, C.Saurel, G.Trouessin, "*Organization-Based Access Control* ", 4th International workshop on Policies for Distributed Systems and Networks (Policy'03),Côme, Italie, 4-6 juin 2003, IEEE Computer Society Press, pp. 120-131.

[6] F.Cuppens, N. Cupens-Boulahia & A.Miège, "*Héritage de privilèges dans le modèle OrBAC: application dans un environnement réseau*" Journées SSTIC, juin2004.

[7] D. Nardi & R.J. Brachman, "*An intrroduction to description logics*" dans F. Baader, D.Calvanese, D. McGuinness, D. Nardi & P. Patel Schneider (éditeurs), "*The Description Logic Handbook : Theory, Implementation and Applications*". Cambridge University Press, pp. 5- 44, 2008.

[8] Bernard ESPINASSE, « Introduction aux Logiques de, Description (LD) », l'Université d'Aix-Marseille, pp 8- 9, 20 Janvier 2009.FRANCE.

[9] Nhan LE THANH, « Introduction à la Logique de, Description (résumé) », Labo I3S, Cours B7, Master 2 PMLT, Ecole doctorale STIC, UNSA.

[10] F.Baader, D.L. McGuiness, D. Nardi and P.F. Schneider, "*The Description logic handbook: Theory, Implementation and Applications*", Cambridge university press (ISBN-13:9780521781763), 2008.

[11] D. Nardi & R.J. Brachman, "*An intrroduction to description logics*" dans F. Baader, D. Calvanese, D. McGuinness, D. Nardi & P. Patel Schneider (éditeurs), "*The Description Logic Handbook : Theory, Implementation and Applications*". Cambridge University Press, pp 5- 44, 2008.

[12] W.A. Woods, "*Understanding subsomption and taxonomy: a framework for progress*", In Morgan-Kaufmann, editor, Principals of Semantic Networks, pp 45- 94. J. Sowa, 1991.

[13] F.Cuppens et A. Miège, "High level Conflit management in the *OrBAC model*", Technical rapport, ENST Bretagne, Novembre 2004.

[14] N.Boustia, "*Un formalisme de sécurisation des bases de données multi-niveaux*", thèse de doctorat, Université des Sciences et de la Technologie Houari Boumediene (USTHB), Alger, 2011.

[15] N. Boustia et A. Mokhtari, "*A dynamic access control model*". In Applied Intelligence Journal, DOI 10.1007 /s 10489-010-0254-z, 2010.

[16] A.Miège, "*définition d'un environnement formel d'expression de politiques de sécurité*", Modèle OrBAC et extensions, Mémoire de thèse de Doctorat.

[17] F.Cuppens, N.Cuppens-Boulahia and Ben Ghorbel, "*High level Conflit management in advanced access control models*", In Workshop on Information and Computer Security (ICS),Timisoara, Romania, Septembre 2006.

[18] A.Toufik, B.Nassim, "*La modelisation du modèle de sécurité OrBAC en logique de description*", **Mémoire pour l'obtention d'un** Diplôme d'Ingénieur d'Etat en Informatique, Université Saâd Dahleb de Blida 2007-2008.

[19] F.Cuppens, N.Cuppens-Boulahia and C.Coma, "*MotOrBAC : un outil d'administration et de simulation de politiques de sécurité*".

[20] F.Cuppens et A. Miège, "High level Conflit management in the *advanced access control models*", Technical rapport, ENST Bretagne, 2005.

[21] F.Cuppens and L.Cholvy, "*Analyzing Consistency of Security Polices*",in IEEE symposium on security and privacy, Oaklan, California, USA, May 1997.

Annexe

Annexe

Algorithme 1 Algorithme Sub δε

En entrée: C et D deux descriptions de concepts de CClassicδε.

En sortie: Répondre par "OUI" ou "NON" à la question "C est – il subsumé par D?"

```
/*Calcul de formes normales*/
fn(C) ←Normalisation(C)
fn(C Π D) ←Normalisation(C Π D)

/*Traitement Bottom*/
si fn(C)=b0
    alors
    Response←"OUI"
    sinon
        si fn(C Π D)=b0
        alors
         Réponse←"NON"
        sinon
        /*Comparaison des formes normales obtenues*/
        Compare (fn(C)θ , fn(CΠ D)θ , rep1)
        si rep1="OUI"
            alors
             Compare (fn(C)δ, fn(C Π D)δ , rep1)
            Réponse←rep2
            sinon
            Réponse←"NON"
            Fin si
         Fin si
Fin si
```

Algorithme 2 Procédure de normalisation

En entrée : D, une description du concept de CClassicδε

En sortie : fn(D), la forme normale de D. D a la forme D≡T1 Π T2 Π…Π Tn, avec n ≥ 1

/**Termes simples*/*

Si D est T

Alors fn(D) ← ⟨ (Ø, Ø, Ø), (Ø, Ø, Ø) ⟩

Fin si

Si D est ⊥

Alors fn(D) ← b0

Fin si

Si d est un concept primitif

Alors fn(D) ← ⟨ (P, Ø, Ø), (P, Ø, Ø) ⟩

Fin si

Si D est ∀ R : T

Alors fn (D) ← t

Fin si

Si D est ∀ R : ⊥

Alors fn(D) ← ⟨ (Ø, ⟨R, b0⟩, Ø), (Ø, ⟨ R, b0⟩, Ø) ⟩

Fin si

/**Termes composés*/*

Si D est ∀ R : C

Alors fn (C) ← normalisation (C)

Si fn (C) =t

Alors fn (D) =t

Sinon

Si fn(C)= b0

Alors fn(D) ← ⟨ (Ø, ⟨R, b0⟩, Ø), (Ø, ⟨ R, b0⟩, Ø) ⟩

Sinon

Fn(D) ← ⟨ (Ø, ⟨R, C⟩, Ø), (Ø, ⟨ R, C⟩, Ø) ⟩

Fin si

Fin si

Fin si

Si D est δC **alors**

Fn(C) ← normalisation (C)

Fn(D) ← ⟨ (P, Ø, Ø), fn(D)δ ⟩

Fin si

Si D est Cε **alors**

fn(D)σπ ← Ø

fn(D)σr ← Ø

fn(D)σε ← Index-Calcul (fn(C)δ)

fn(D)δπ ← fn(C)δπ

fn(D)δr ← fn(C)δr

fn(D)δε ← fn(D)σε È fn(C)δε

Fin si

Si D est A Π B

Alors fn(A) ← normalisation (A)

fn(B) ← normalisation (B)

fn(D) ← fn(A) Ä fn(B)

Fin si

Algorithme 3 Procédure Calcul-Index (s) (s est un 3-uplets)

Si une paire (n, s) dans la table d'index

Alors Calcul-Index ← n

Sinon

Créer un nouveau couple dans la table d'index (m, s) où m : le nombre de couple dans la

table d'index + 1.

Calcul-Index (m, s)

Fin si

Algorithme 4 Procédure Compare (t1, t2, Réponse)

Si (t1$\pi \neq$ t2π) ou (t1$\varepsilon \neq$ t2ε)

Alors Réponse ← "NON"

Sinon

/* *comparaison des champs rôles*/

Comparer-rôle (t1r, t2r, Reps)

Réponse ← Reps

Fin si

Algorithme 5 Comparer- rôle (Ar, Br, Reps)

Reps ← "OUI"

Si | Ar | ≠ | Br |

Alors Reps ← "NON"

Fin si

Tant que (Ar ≠ Ø) et (Reps ="OUI")

Faire

Laisser e Î (e=⟨ R, c1 ⟩ ; chercher e' dans Br de même nom R

Si e' Ï Br

Alors Reps ← "NON"

Sinon

Laisser e' Î Br (e' =⟨ R, c2 ⟩)

Fin si

/**comparaison des 3-uplets stricts et défauts des formes normales de c1 et c2*/*

Comparer (c1σ, c2σ, repStrct)

Si reps ="OUI"

Alors Comparer (c1δ, c2δ, repDft)

repStrct ← repDft

Sinon

repStrct ← "NON"

Fin si

Ar ← Ar – e

Br ← Br – e'

Fin Pour

Algorithme Héritage

En entrée: le concept C.

En sortie: le tableau des groupes disjoints hérités du concept C est mis à jour.

Début

Pour tous Ci du tableau des SPS du concept C

Faire

- ajouter les groupes disjoints et les groupes disjoints hérités du Ci au tableau des groupes disjoints hérités du concept C

  Si Ci est incohérent **alors** C est incohérent

Fait

Pour tous GRi du tableau des groupes disjoints de C

Faire

- ajouter le concept C dans la table des concepts des groupes disjoints.
- Appel la procédure EstIncohérent **{*Pour vérifier la cohérence de la BC*}**

Si les SPG du C sont différents de BOTTOM

alors

Pour chaque Ci du tableau des SPG du C

Faire appel Héritage pour le concept Ci **{*Appel récursif*} Fait**

Fsi

Fait

FIN

Algorithme EstIncoherent

En entrée: le tableau des SPS du concept C à vérifier sa cohérence

En sortie: Booléen

Début

pour tous élément Ci du tableau des SPS du concept C

Faire

pour tous élément Cj du tableau des SPS du C tel que $i \neq j$

Faire

l'intersection des deux tableaux du groupe disjoint(Groupe Disjoint, Groupe Disjoint hérité) de Ci et Cj

Si l'intersection est nulle

alors return (false) **{*le concept est cohérent*}**

Sinon return (true) **{*le concept est incohérent*}**

FinSi

Fait

Fait

FIN