République algérienne démocratique et populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université SAAD DAHLEB Blida

Faculté des sciences
Département d'Informatique

Mémoire de fin d'études en vue d'obtention du diplôme master
En Informatique
Spécialité :Ingénierie des logiciels

La délégation des permissions dans Or-BAC avec la logique de description

---

Sujet proposé par :

M^lle^ N.BOUSTIA

Présenté par :

Mr. SAHEL Mehdi
Mr. BOUDJEMA Yacine

Devant le jury composé de :

Mme Bensttiti : Présidente

Mlle Toubaline : Examinatrice

Mlle Ameur : Examinatrice

Mlle Boustia : Rapporteur

2011-2012

Remerciements

Nous remercions tout d'abord dieu de nous avoir apporté la volonté, le courage et la patience pour accomplir ce travail. الحمد لله.

Je remercie vivement Mlle BOUSTIA, notre promotrice, pour son encadrement, son aide et ses conseils pendant ces mois de travail qui furent parfois joyeux parfois déprimants. Nous la remercions de nous avoir fait profiter de son expérience de recherche dans le domaine. Ses conseils sur l'aspect théorique et pratique qui nous ont été utiles et permis de mener à bien cette thèse.

Nous adressons aussi notre profonde reconnaissance à tous nos enseignants qui ont contribué à notre formation.

Nos sincères remerciements sont adressés également à nos amis pour leurs aides, leurs soutiens et leurs conseils.

Enfin, nous remercions nos parents, nos frères et nos sœurs qui ont toujours été un soutien déterminant dans le déroulement de mes études.

Dédicaces

J'ai le grand honneur de dédier ce travail A mes très chers parents, qui ont consacre leur vie pour m'assurer l'amour et la tendresse et qui n'ont jamais cesse de m'encourager et m'aider dans mes études

A mon frère et ma sœur : Lyesse et Marwa

A mes grands pères et grand-mère

A tous mes oncles et mes tantes

A toute ma famille et mes proches

A tous mes amis, a tous ceux qui sèment le bonheur dans mon chemin

A tous les étudiants d'informatique

A tous ceux que j'aime

Yacine

Dédicaces

Je dédie ce modeste travail à ma mère qui a veillé à ce que je sois ce que je suis devenu maintenant, à mon père, mes sœurs: Amina, Soumia et Sara et mon frère yacine. Sans oublier mes amis: Abdelhak, Amine, Yacine et Youcef et surtout mes deux frères: Hamza et Mehdi.

A ma grand-mère, mon grand père; mes tantes: Djazia, Faiza, Aicha, Nadjia, Wassila.

A Maa, tata aicha, tata malika.

Une dédicace spéciale à amou Mamar.

Mehdi

Résumé :

Ce projet a pour but la conception et la réalisation d'un modèle de contrôle d'accès dynamique en utilisant la logique de description. DL-OrBAC s'est inspiré du modèle Or-BAC. L'attribution des autorisations aux utilisateurs se fait en fonction de ses rôles dans une organisation et dans un contexte donné.

Mots clés : *Or-BAC, Délégation, Logique de description.*

Abstract:

This project aims to design and conduct a dynamic access control model using on a descriptive logic. DL-OrBAC is inspired from Or-BAC model. Assigning authorizations to users depends on their roles into the organization in a given context.

Key words: *Or-BAC, delegation, descriptive logic.*

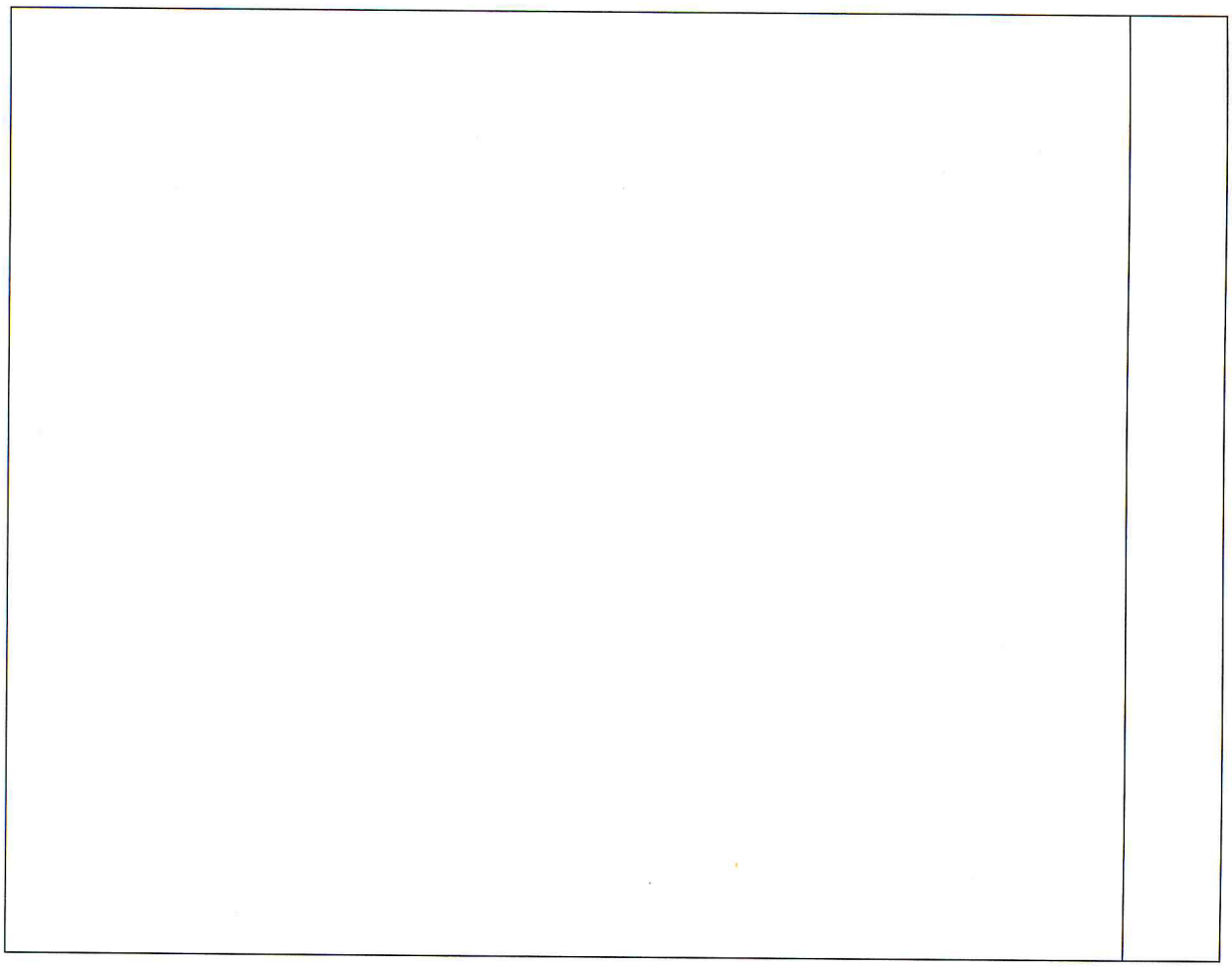
ملخص

يهدف هذا المشروع إلى تصميم وانجاز نموذج للتحكم في الدخول باستخدام منطق الوصف. واستلهم هذا النموذج من النموذج . ويستند تعيين الاذن للمستخدمين على دورهم في المنظمه وفي سياق معين.

كلمات البحث: ، منطق الوصف, تفويض , OR-BAC.

Sommaire

Table de figure



Introduction

Introduction générale :

De nos jours, l'utilisation des systèmes d'information s'accroit avec le temps et les risques d'utilisation malveillante s'accroissent avec, d'où vient la nécessité de protéger notre système contre toute utilisation virulente.

Pour répondre aux besoins croissants de sécurité, des logiciels et des systèmes d'information et des approches basées sur les politiques de sécurité sont largement utilisées.

La sécurité des systèmes est assurée par l'application d'un ensemble de règles à divers axes : physique, administratif et logique.

- L'axe *physique* définit l'ensemble de procédures et moyens qui protègent les locaux et les biens contre les risques tels que l'incendie, l'inondation, etc.
- L'axe *administratif* définit un ensemble de procédures et moyens qui traitent la sécurité d'un point de vue organisationnel au sein de l'entreprise, tels que la structure de l'organigramme et la répartition des tâches.
- L'axe *logique* fait référence aux aspects informatique et technique des systèmes et définit les actions légitimes qu'un utilisateur peut effectuer.

L'axe logique repose sur deux aspects :

- Le premier consiste à l'*identification* et l'*authentification* des utilisateurs.
- Le second à l'***autorisation*** et le ***contrôle d'accès*** qui signifie la vérification de la légitimité des opérations demandées.

Un système de contrôle d'accès doit intercepter toutes les tentatives d'accès aux ressources et informations critiques en définissant des règles qui régissent les accès et décident lesquels sont autorisés.

Le modèle de contrôle d'accès appelé ***Or-Bac*** **(Organisation Based Access control)** est le dernier résultat des recherches dans le domaine de contrôle d'accès, il s'est apparu pour généraliser les modèles qui l'ont précédé et palier à leurs insuffisances.

Dans le modèle Or-BAC, il est possible de définir des hiérarchies de rôles, de vues et d'activités. Chaque hiérarchie définit respectivement une relation d'ordre partiel sur l'ensemble des rôles, des vues et des activités.

Grâce à cette Hiérarchie les chercheurs dans le domaine de la sécurité des systèmes et dans le domaine du contrôle d'accès en particulier ont pu définir la notion de **délégation des droits**.

La délégation d'un droit et donner une permission par le détenteur de ce droit à un tiers pour agir à sa place ou à la place d'un autre. Donc la délégation permet de donner à un utilisateur particulier un privilège, sans donner ce privilège à toutes les personnes ayant le même rôle que lui.

Cependant la délégation pose beaucoup de problèmes liés à **la gestion des conflits** ; un conflit c'est qu'un utilisateur jouant un rôle qui lui interdit de faire une activité, reçoit une permission de la faire d'un autre utilisateur par la délégation.

Notre travail consiste en l'étude et la réalisation du modèle Or-Bac et résoudre les conflits en utilisant la logique de description. Cette logique est bien adaptée aux représentations des systèmes d'informations qui ont besoin de représenter la hiérarchie, l'héritage, les rôles ... etc. Nous nous sommes intéressés particulièrement dans cette étude à l'implémentation de l'opération de délégation des permissions et la notion d'héritage qui permet calculer les propriétés héritées lors d'une délégation.

Pour atteindre cet objectif nous allons franchir deux étapes : La création de la base de connaissances du modèle et les inférences sur cette base, et L'expression de la délégation des permissions avec le modèle Or-Bac en utilisant la logique de description et enfin l'appliquer à un système d'information médical.

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 l'étude de la délégation, d'un troisième chapitre pour l'étude de la logique de description, d'un quatrième chapitre pour illustrer le fonctionnement de notre modèle Or-BAC et finalement d'une conclusion générale.

Chapitre 1

Etude du modèle de contrôle d'accès Or-BAC

Au sein de l'organisation, un utilisateur demande d'effectuer une action sur un objet, l'objectif d'un modèle de contrôle d'accès est de décider si cet utilisateur a ce droit ou pas.

Un système de contrôle d'accès doit intercepter toutes les tentatives d'accès aux ressources et informations critiques en définissant des règles qui régissent les accès et décident lesquels sont autorisés[13].

Nous présentons dans ce chapitre le Modèle Or-Bac qui est le dernier résultat des recherches dans le domaine de contrôle d'accès, il s'est apparu pour généraliser les modèles qui l'ont précédé en les corrigeant et en évitant les failles dans lesquelles ils sont tombés.

1. Le Modèle Or-bac :

Or-BAC est un modèle de politique de sécurité issu des travaux réalisés dans le cadre du projet RNRT MP6 en France (Modèles et Politiques de Sécurité des Systèmes d'Informations et de Communication en Santé et en Social) [1].

L'objectif d'Or-BAC est de permettre la modélisation d'une politique de sécurité basée sur le concept de l'organisation [2] (par exemple un hôpital, une banque, ou ça peut même être un département d'une organisation). Pour arriver à ce but, et afin de pouvoir gérer les droits d'accès, le modèle Or-BAC repose sur les entités suivantes :

- *L'organisation :* l'entité centrale du modèle.
- Les entités concrètes : sujets, actions, objets.
- Les entités abstraites : rôles, activités, vues.
- Le contexte.

Ainsi 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.

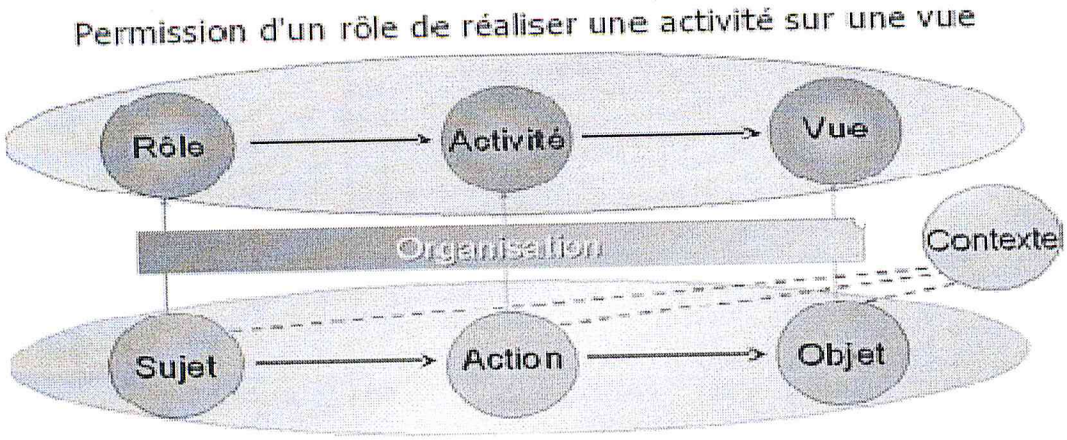


Figure I.1 : Le modèle Or-BAC

2. Les entités d'Or-BAC :

2.1. Organisation :

L'*organisation* est l'entité centrale du modèle Or-BAC, elle peut être vue comme un groupe d'entités actives, c'est-à-dire un ensemble de sujets jouant certains rôles. Notons qu'un groupe de sujets ne représentent pas une organisation que s'il y'a un accord entre ces sujets pour former cette organisation [3].
Nous pouvons considérer comme organisations : « Une clinique médicale », « Une banque », «Le service des urgences dans un hôpital » ...etc.

2.2. Les entités concrètes :

- *Les sujets :*

Dans le modèle Or-BAC un *sujet* est soit une entité active au sein de l'organisation, c'est-à-dire un utilisateur. Par exemple, « Mehdi », «Yacine », etc. soit une organisation.
Par exemple, « Service comptable d'une banque », « Service des urgences d'un hôpital », etc.

- *Les actions :*

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 Or-BAC, l'entité *Action* englobe principalement les actions informatiques comme " lire", " écrire", " envoyer", etc.

- *Les objets :*

Dans le modèle Or-BAC l'entité *objet* représente principalement les entités passives comme les fichiers ou les courriers électroniques, etc. Dans le domaine médical, On considère aussi les dossiers administratifs, les dossiers médicaux des patients comme étant des objets.

2.3. Les entités abstraites :

- *Les rôles :*

Dans le modèle Or-BAC, l'entité Rôle est utilisée pour structurer le lien entre les sujets et les organisations.
Dans le domaine médical, les rôles « infirmier » ou « médecin » sont joués par des utilisateurs, alors que les rôles « service des urgences » ou « unité des soins intensifs » sont joués par des organisations.

- *Les activités :*

Dans le modèle Or-BAC, les activités correspondent à des actions qui ont un objectif commun. Les activités sont « consulter », « modifier », etc.

- *Les vues :*

Dans la mesure où il est 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.*
Dans le modèle Or-BAC, une vue correspond, comme dans les bases de données relationnelles, à un ensemble d'objets qui satisfont 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.

2.4. Le 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[14].

3. Les relations :

Comme chaque entité concrète doit être représentée par une entité abstraite, nous définissons les relations suivantes :

3.1. La relation « Habilite » :

Nous introduisons la relation « *habilite* » entre les entités Sujet, Rôle et Organisation Comme présenté dans la figure 1.2

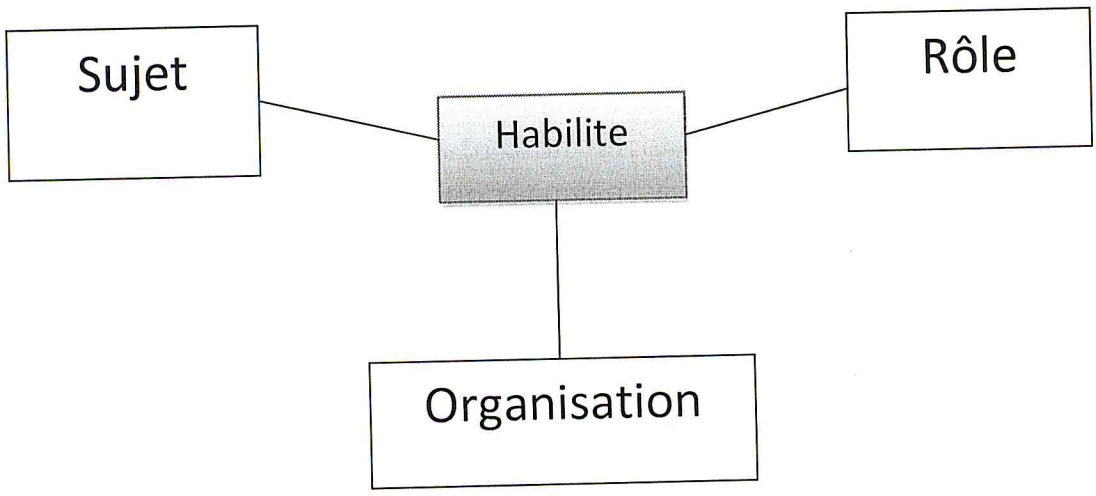


Figure 1.2: La relation Habilite[9].

Habilite (Org, S, R) : signifie que l'organisation Org habilite le sujet S à jouer le rôle R.

3.2. La relation « Utilise » :

Nous introduisons la relation « *Utilise* » entre les entités Objet, Vue et Organisation Comme présenté dans la figure 1.3

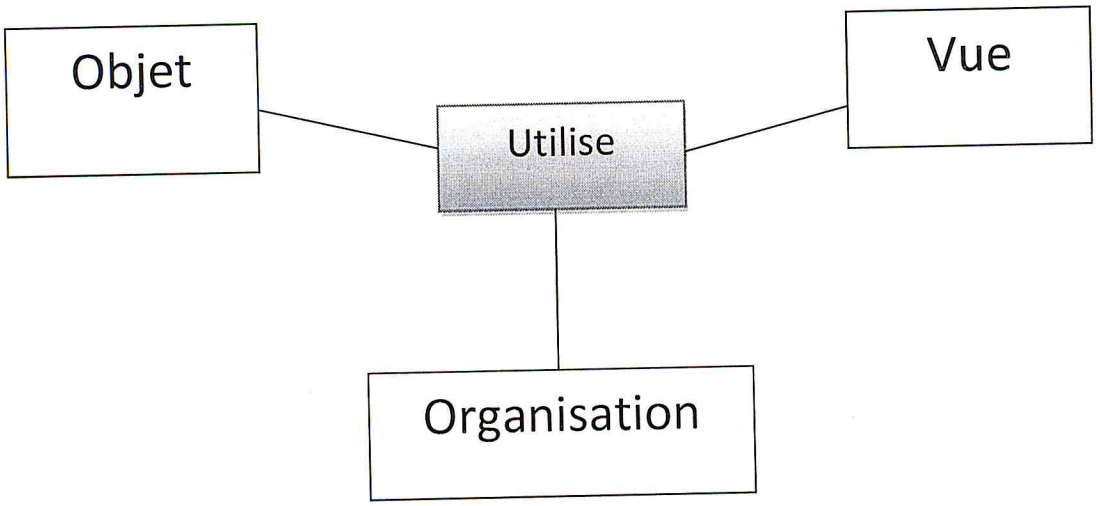


Figure 1.3 : La relation Utilise[9].

Utilise (Org, O, V) : signifie que l'organisation Org utilise l'objet O dans la vue V.

3.3. La relation « Considère » :

Nous introduisons la relation « *Considère* » entre les entités action, activité et Organisation Comme présenté dans la figure 1.4

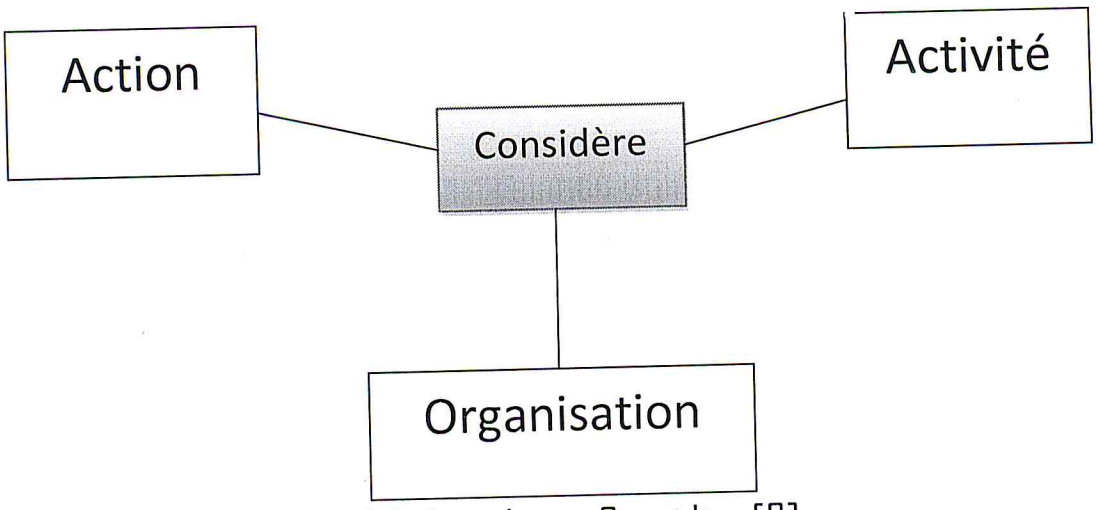


Figure 1.4 : La relation Considère[9].

Considère (Org, A, Ac) : signifie que l'organisation Org considère l'action A comme faisant partie de l'activité Ac.

3.4. La relation « Définit » :

Nous introduisons la relation « *Définit* » entre les entités Organisation, Sujet, Objet, Action et Contexte Comme présenté dans la figure 1.5

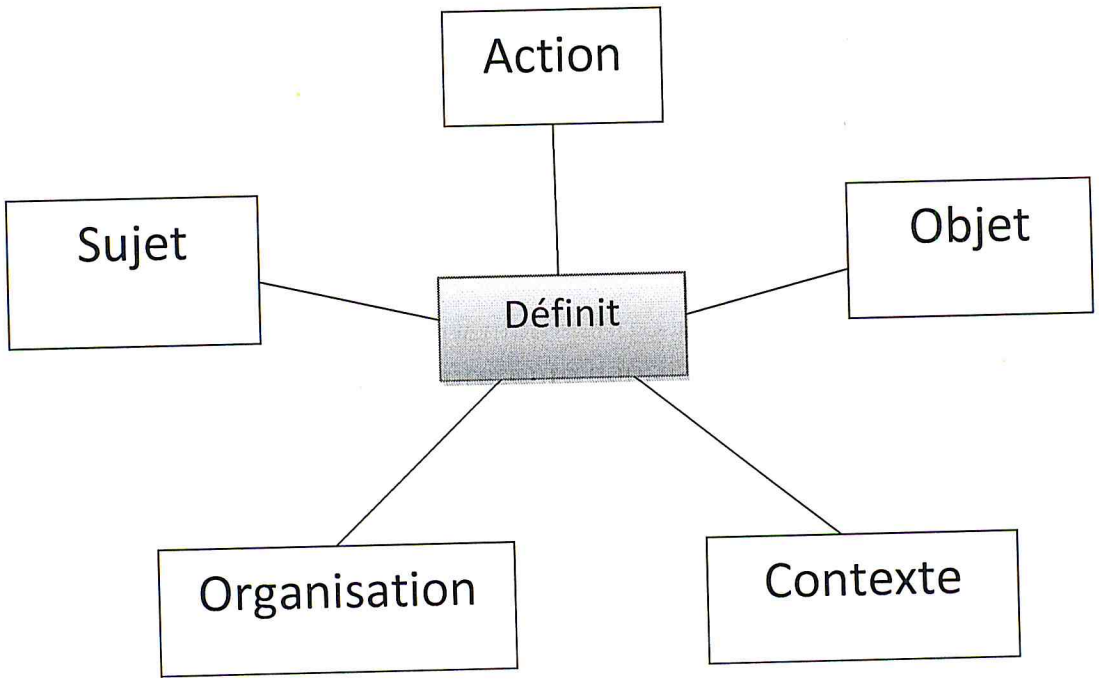


Figure 1.5 : La relation Définit[9].

Définit (Org, S, A, O, C) : signifie qu'au sein de l'organisation Org, le contexte C est vrai entre le sujet S, l'objet O et l'action A.

4. La politique de sécurité :

En utilisant les entités et les relations introduites dans les sections précédentes, nous pouvons à présent définir des politiques de sécurité appliquées au sein d'une organisation.
Une politique de sécurité réglemente les accès au système à travers des permissions, des interdictions, des obligations et des recommandations. Nous ne traitons que les permissions, en considérant que les mêmes raisonnements s'appliquent aux interdictions, aux obligations et aux recommandations.
L'objectif ici est d'ajouter une nouvelle entité *Permission* afin de relier les organisations, les rôles, les vues, les activités et les contextes. Plus précisément, si **Org** est une organisation, **R** est un rôle, **A** est une activité, **V** est une vue et **C** et un contexte, **Permission (Org, R, A, V, C)** signifie que l'organisation **Org** accorde au rôle **R** la permission de réaliser l'activité **A** sur la vue **V** dans un contexte **C**.

Exemple :

Permission (Mustapha-Bacha, Médecin, Consulter, Dossier-Médical, Urgence) : signifie que l'hôpital Mustapha Bacha (l'organisation) accorde aux médecins (rôle) la permission de consulter (l'activité) les dossiers médicaux (la vue) en cas d'urgence (contexte).

5. L'hiérarchie dans le modèle Or-BAC :

Dans le modèle Or-BAC, L'hiérarchie définit respectivement une relation d'ordre partiel sur l'ensemble des rôles, des vues et des activités. Nous présentons dans cette section les règles générales d'héritage des permissions qui leur sont associées [4].

5.1. L'hiérarchie de rôles :

Nous nous attachons dans un premier temps à étudier la hiérarchie de rôles. Pour cela nous introduisons le prédicat *Sub_role*.
Sub_role (Org, R1, R2) : signifie que 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.

5.2. L'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*.

Sub_activité (Org, A1, A2) : signifie que dans l'organisation Org, l'activité A1 est une sous-activité d'A2.

5.3. L'hiérarchie de vues :

Comme pour les rôles et les activités, l'ensemble des vues est structuré par des hiérarchies dépendantes de l'organisation. Ces hiérarchisations sont modélisées par le prédicat *sub_vue*.

Sub_vue (Org, V1, *V2) qui signifie : dans l'organisation Org, la vue V1 est une sous-vue de la vue V2.*

6. Conclusion :

Nous avons présenté dans ce chapitre le modèle de contrôle d'accès Or-BAC, ses entités et les différentes relations entre ces entités.

Or-BAC est centré sur le concept d'organisation. En effet, tous les autres concepts que nous avons présentés dépondent d'une organisation donnée.

Nous avons aperçu qu'à partir de ces concepts nous pouvons définir des relations afin d'exprimer une politique de sécurité comme un ensemble de permissions, d'interdictions, d'obligations et de recommandations. Nous n'avons présenté que les permissions en considérant que les mêmes règles s'appliquent au reste.

Dans la partie suivante, nous allons étudier la délégation dans le modèle Or-BAC ainsi que ses notions.

Chapitre 2

Etude de la délégation

La délégation permet de donner à un utilisateur particulier un privilège, sans donner ce privilège à toutes les personnes ayant le même rôle que lui.

La délégation, bien que très utilisée, est très peu modélisée dans les politiques de sécurité car ce concept est très complexe. En effet, grâce à une délégation, une permission peut être donnée par le détenteur d'un droit à un tiers pour agir à sa place ou à la place d'un autre.
On voit déjà ici apparaître qu'une délégation peut faire intervenir plusieurs parties :

- Le sujet qui possède le privilège.
- Le sujet a qui on délègue le privilège.
- Le sujet qui délègue le privilège (pour agir à sa place ou à la place d'un autre).

Il existe trois types de situation dans lesquelles la notion de délégation apparaît [3]:

- La maintenance d'un rôle.
- La décentralisation de l'autorité.
- Le travail de collaboration.

La maintenance d'un rôle correspond au cas où un utilisateur doit déléguer une partie de ses permissions afin qu'on puisse remplir toutes ses obligations pendant son absence.
La décentralisation de l'autorité est surtout utile dans le cas où on modifie une partie de l'organisation. En pratique, ce cas peut correspondre à l'ouverture d'un nouvel hôpital dans lequel on va transférer une partie des médecins exerçant dans les autres hôpitaux de la région.
Le cas du ***travail en collaboration*** est évident, si on souhaite que notre partenaire puisse lire les documents que l'on possède sur un projet donné, il faut lui en donner l'autorisation.

Cependant, la délégation pose de nombreux problèmes. Entre autre, un utilisateur **X** ayant obtenu tous les droits d'un autre utilisateur **Y** peut ôter les droits à **Y** si **X** possède certains droits administratifs. Il peut aussi arriver que l'on oublie de révoquer une délégation faite à **Z** et qui n'a plus d'utilité d'être, ce qui peut laisser la possibilité à **Z** de se faire passer pour quelqu'un d'autre. C'est l'une des raisons pour lesquelles il est important de définir deux types de permission, celles qui sont délégables et celle qui ne peuvent l'être.

1. Notions de délégation :

La délégation est liée à une multitude de notions Comme le montre la figure 2.1 :

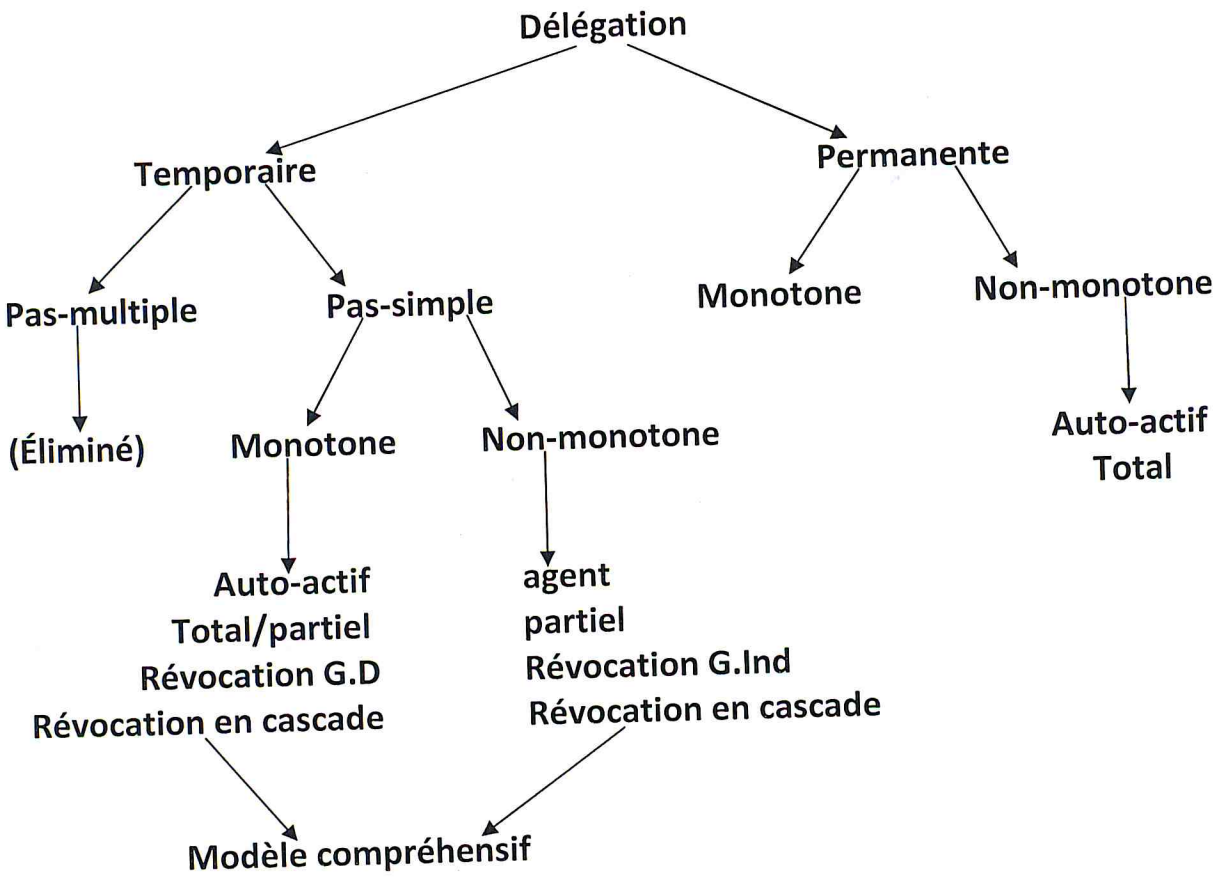


Figure 2.1: Types de délégation

1.1. Délégation permanente/temporaire :

La délégation permanente est qu'un utilisateur X délègue ses droits d'accès à un objet O à un autre utilisateur Y de manière permanente. Par exemple, un directeur donne le droits d'accès à tous les dossiers de l'entreprise qu'il dirige à son conseillé de manière permanente, et donc le conseillé peut accéder à ces dossiers de manière permanente (Il n'y'a pas de limite de temps).

La délégation temporaire est qu'un utilisateur X délègue ses droits d'accès à un objet O à un utilisateur Y de manière temporaire. Autrement dit, une délégation limitée dans le temps. Par exemple, si un médecin doit s'absenter pour une période d'un mois, il donnera la permission d'accès aux dossiers médicaux de ses patients un autre médecin juste pendant un mois, et il récupérera ses droits avec son retour.

1.2. Délégation monotone/non-monotone :

La délégation monotone est qu'un utilisateur X délègue ses droits d'accès à un objet O à un autre utilisateur Y tout en gardant la puissance de son rôle. C'est-à-dire, X ne perd pas ses droits d'accès à O en les délégants à Y. Par exemple, un directeur délègue ses droits à son secrétaire pour effectuer une tâche précise, dans ce cas là, le directeur ne perdra pas ses droits d'accès. Et alors le directeur et le secrétaire auront tous les deux les mêmes permissions jusqu'à ce que la tâche se réalise.

La délégation non-monotone est qu'un utilisateur X délègue ses droits d'accès à un objet O à un autre utilisateur Y, mais dans ce cas là X perd ses droits d'accès et Y sera le seul à pouvoir accéder à O. Par exemple, Le directeur des finances délègue ses droits au comptable à la fin du mois pour le règlement des payes, et dans ce cas là, le directeur ne pourra pas accéder aux dossiers financiers de l'entreprise jusqu'à ce que le comptable termine son travail. Cependant, le directeur des finances pourra toujours surveiller le travail du comptable du moment qu'il est responsable de ce service.

1.3. Délégation « Grant-dependant »/ « Grant-independant » :

La délégation de type « grant-dependant » est qu'un utilisateur X délègue ses droits d'accès à un objet O à un autre utilisateur Y, et Y délègue les mêmes droits à un autre utilisateur Z, et que X soit le seul à pouvoir révoquer ces droits à Y ou à Z. Par exemple, un directeur donne le droit d'accès aux dossiers financiers à sa secrétaire générale, la secrétaire générale donne ce droit à un autre secrétaire, et le directeur est le seul à enlever ce droit au secrétaire générale et/ou du deuxième secrétaire.

La délégation de type « grant-independant » est qu'un utilisateur X délègue ses droits d'accès à un objet O à un autre utilisateur Y, et Y délègue les mêmes droits à un autre utilisateur Z, et on autorise Y à révoquer les droits délégué à Z. Autrement dit, X peut enlever les droits délégués à Y et/ou à Z, et même Y peut enlever les droits qu'il a délégué à Z.

1.4. Délégation totale/partielle :

La délégation totale est qu'un utilisateur X délègue toutes les permissions qui lui sont assignées par son rôle à un autre utilisateur Y. Par exemple, Le directeur des ressources humaines peut remplacer le directeur général pendant son absence, et dans ce cas là, le directeur général doit déléguer toutes ses permissions au directeur des ressources humaines afin qu'il puisse gérer l'entreprise et même prendre des décisions s'il faut.

La délégation partielle est qu'un utilisateur X délègue un sous-ensemble des permissions qui lui sont assignées par son rôle à un autre utilisateur Y. Par exemple, un professeur à une université qui a des aides enseignants, des aides de laboratoire et une secrétaire peut déléguer son rôle d'enseignant à l'un de ses aides enseignants, son rôle de recherche à l'un de ses aides de laboratoires et son rôle d'administration à sa secrétaire afin qu'elle puisse gérer son mail.

1.5. Délégation auto-active/ par agent :

La délégation auto-active est qu'un utilisateur X délègue ses droits d'accès à un objet O à un autre utilisateur Y, et que X soit la personne qui administre la délégation. C'est-à-dire, il n'y'a pas d'intermédiaire entre X et Y. Par exemple, sous la même analogie de professeur utilisée dans l'exemple ci-dessus, le professeur délègue ses droits à ses aides enseignants, ses aides de laboratoire ou à sa secrétaire lui-même.

La délégation par agent est qu'un utilisateur X délègue ses droits d'accès à un objet O à un autre utilisateur Y à l'aide d'un agent, l'agent pouvant être n'importe quelle tierce personne dans l'organisation. Par exemple, le professeur demande à sa secrétaire de déléguer son rôle de recherche à son aide de laboratoire sans que sa secrétaire s'affecte ces droits ou les modifie.

1.6. Délégation à un pas/ à pas multiple :

La délégation à n-pas est qu'un utilisateur X délègue ses droits d'accès à un objet O à un autre utilisateur Y, et le même droit peut être délégué à n-1 personnes .Par exemple, X délègue à Y un droit D à 2-pas et Y délègue D à 1-pas à Z.

1.7. Délégation simple/ multiple :

La délégation simple est qu'un utilisateur X délègue ses droits d'accès à un objet O à une personne unique.

La délégation multiple est qu'un utilisateur X délègue les mêmes droits d'accès à un objet O à une multitude de personnes en même temps.

1.8. Délégation par accord unilatéral/ bilatéral :

La délégation par accord unilatéral est qu'un utilisateur X délègue ses droits d'accès à un objet O à un utilisateur Y, et pour effectuer cette délégation on ne prend en compte que l'accord de X (la personne désirant déléguer son droit).

La délégation par accord bilatéral est qu'un utilisateur X délègue ses droits d'accès à un objet O à un utilisateur Y, et pour effectuer cette délégation on prend en compte l'accord de X et Y.

2. La révocation de la délégation :

Si la délégation est temporaire, il faut pouvoir la révoquer. On a pu voir précédemment deux types de délégation jouant sur la révocation. Lorsque la délégation est "grant-dependant" alors seule la personne à l'origine de la délégation peut ôter ce droit. Quand la délégation est de type "grant-independant" seules les personnes ayant engendré la délégation d'un droit à une personne peuvent lui révoquer ce droit.

Cependant, la personne, dont les droit ont été révoqués, a peut être pu déléguer ce droit auparavant.

Cette situation peut poser des problèmes dans certains cas. Selon le type de délégation, la personne ayant était déchue d'un droit peut récupérer ce droit grâce à une personne à qui elle aurait délégué le droit[16].

Pour anticiper ce problème, on peut créer deux types de révocation. Un premier type permet de ne révoquer le droit qu'à une personne désignée. Le deuxième type permet de révoquer le droit sur une personne, ainsi que sur toutes les personnes ayant reçu ce droit par délégation, c'est une révocation en cascade.

3. Conclusion:

Nous avons présenté dans ce chapitre le principe de délégation, ses différentes notions et la révocation de la délégation.

Nous avons vu que la délégation a de multiples avantages. Cependant, si on l'autorise à mauvais escient, elle peut aller à l'encontre de la politique. Des personnes mal intentionnées pourraient utiliser la délégation afin d'effectuer des révocations qu'ils n'avaient pas le droit de faire. D'où l'importance de bien administrer sa politique de contrôle d'accès, si la délégation est mise en place.

Dans la partie suivante, nous allons présenter les caractéristiques de base d'une logique de description.

Chapitre 3

Etude de la logique de description

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*** [5].

Ce chapitre présente les logiques de description ***(LD)***, une famille de langages de représentation de la connaissance 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.

1. 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.

1.1. 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 problèmes d'inférence en temps souvent polynomial, par le biais d'une catégorie d'algorithmes de vérification de subsomption 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 [6].

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

Dans les années 1990, une nouvelle classe d'algorithmes s'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 [6].

2. 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 [7].

2.1. Objets de la logique de description :

Les objets qui sont définis et manipulés dans une logique de description sont les concepts, les individus et les rôles.

- ***Les concepts*** : peuvent être vus comme des prédicats logiques unaires. Comme pour les réseaux sémantiques, ils peuvent être définis ou primitifs.

- ***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.*, le concept avec lequel le rôle établit une relation, et la cardinalité, *i.e.*, le nombre minimal et maximal que peut prendre un rôle

Exemple : La description du concept suivant décrit toute personne majeure.
Personne-Majeure ≡ Personne ⊓ âge : MIN 18.

- *Personne-Majeure* est un concept défini.
- *Personne* est un concept primitif.
- *âge* est un rôle.
-

2.2. Niveaux de description :

La modélisation des connaissances d'un domaine avec les LD se réalise en deux niveaux comme le montre la figure 3.1. Le premier, *le niveau terminologique (*ou TBox); Une TBox comprend la définition des concepts et des rôles alors que le second, *le niveau factuel (*ou ABox); décrit les individus en les nommant et en spécifiant en termes de concepts et de rôles, des assertions qui portent sur ces individus nommés. 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.

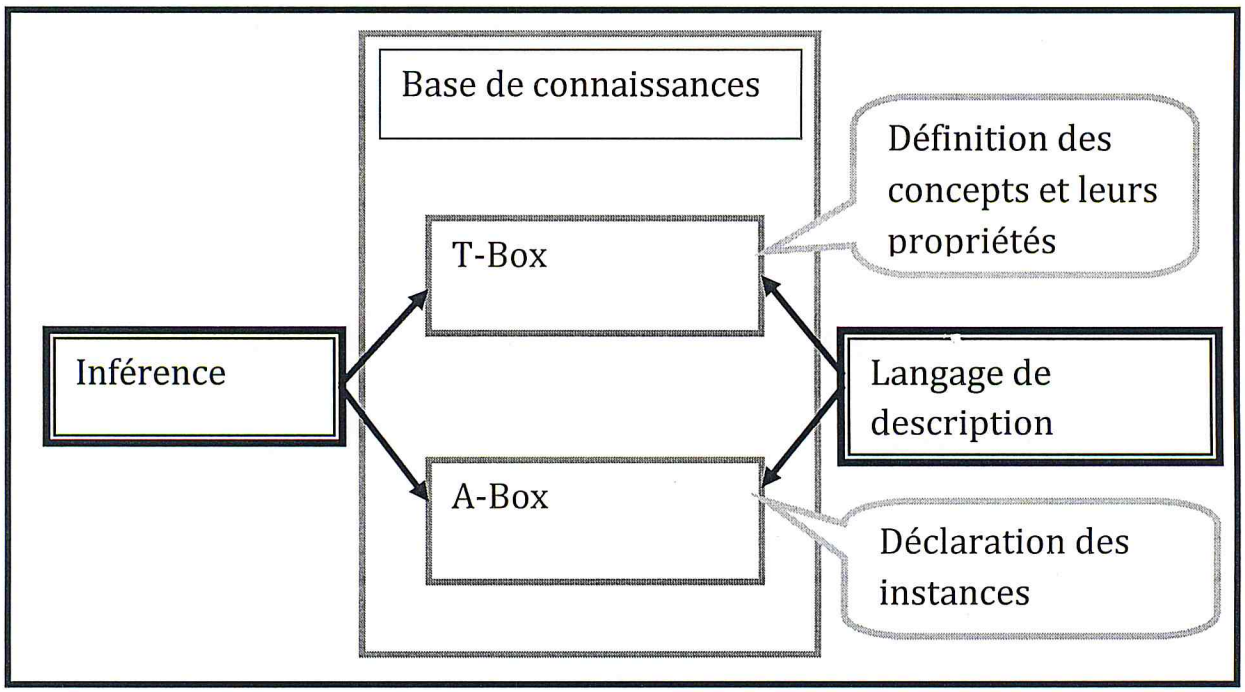


Figure 3.1 : Structure générale des logiques de description

2.2.1. Niveau terminologique (T-Box) :

La T-Box contient toutes les définitions des concepts du domaine ainsi que les 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. Par exemple si on spécifie que *humain* est une sous-classe de *animal*, on peut déduire qu'une mère est un animal. 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.

2.2.2. Niveau factuel (A-Box) :

Nommé aussi le langage assertionnel. Dans la A-Box, on retrouve les assertions sur les individus. En d'autres mots, on y spécifie quelles sont les entités du monde et à quelle classe elles appartiennent. C'est dans la A-Box, par exemple, qu'on spécifiera que MARIE est une mère, c'est à-dire un individu qui est une *instance* de la classe mère. La ABox contient aussi des énoncés spécifiant les relations qui existent entre les individus. Ainsi, comme dans la T-Box il sera spécifié qu'une mère doit avoir au moins un enfant, la A-Box devra contenir au moins un autre individu, et une relation entre celui-ci et Marie indiquant qu'il est un de ses enfants.

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.

2.3. La logique minimale AL:

Les langages de la logique descriptive sont déterminés par la forme des énoncés qui sont permis. La plupart des langages utilisés découlent du langage *AL* (*Attributive Language*), dont l'expressivité est plutôt limitée.

2.3.1. Les constructeurs d'AL :

La figure 3.3 illustre les constructeurs offerts par $\mathcal{AL}$ pour l'édification de concepts composés.

C, D ⟶ A (Concept atomique)

| T (Concept universel)

| $\bot$ (Le concept le plus spécifique)

| ¬A (La négation atomique)

| C ∏ D (L'intersection)

| ∃R.T (Quantification existentielle limitée)

| $\forall$ R.C (Quantification universelle complète)

Figure 3.3 : La grammaire des expressions conceptuelles selon AL

- Le constructeur **C⊔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.T** désigne l'ensemble des individus, membres du domaine d'un rôle R pour une interprétation donnée.
- Le quantificateur universel **$\forall$ 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.

2.3.2. La sémantique formelle d'AL:

La sémantique du langage *AL présentée dans la figure 3.4* 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.
Nous avons une fonction d'interprétation I qui associe à chaque description un sous-ensemble de Δ.

$I(T) = \Delta$

$I(\bot) = \{\}$

$I(\neg A) = I(\Delta) \setminus I(A)$

$I(C \sqcap D) = I(C) \sqcap I(D)$

Figure 3.4 : La sémantique formelle d'AL

Deux concepts C et D d'une T-Box ***AL*** s'équivalent si et seulement si I (C) = I (D) pour toute interprétation I [6].

2.4. Les extensions d'AL :

Il existe trois façons proéminentes d'étendre AL :

- ajouter des constructeurs de concepts.
- ajouter des constructeurs de rôles.
- énoncer des contraintes sur l'interprétation des rôles (Baader, 2003).

2.4.1. L'extension par ajout de constructeurs de concepts ou de rôles :

Le tableau dans la figure 3.5 illustre des exemples de constructeurs pour augmenter AL. La première colonne contient la lettre qui désigne le constructeur, la deuxième sa syntaxe d'utilisation et la dernière sa sémantique. La nomenclature des LD dicte que pour chaque constructeur ajouté, il faut agglutiner la lettre correspondante au nom de la logique originale. Par exemple, la logique AL, enrichie de l'union (**U**) et de la quantification existentielle complète ($\exists$), se nomme $\mathcal{ALUE}$.

Le constructeur $\mathcal{O}$ permet la description de concepts par l'énumération d'individus nommés, $\mathcal{U}$ désigne l'union de concepts arbitraires, $\mathcal{E}$ la quantification existentielle complète, $\mathcal{C}$ la négation complète, $\mathcal{I}$ les rôles inverses et $\mathcal{H}$ l'inclusion entre rôles. Les constructeurs $\mathcal{F}$, $\mathcal{Q}$ et $\mathcal{N}$ sont trois variantes de la contrainte de cardinalité sur rôle.

2.4.2. L'extension par ajout de contraintes sur l'interprétation des rôles :

La spécification d'un ensemble de rôles transitifs $\mathcal{NR}+$, $\mathcal{R}+$ constitue une extension par ajout de contraintes sur l'interprétation des rôles (désignée par la lettre $\mathcal{AL}$), qui permet l'expression de rôles transitifs tels qu'*ancêtreDe* ou *frèreDe*.

2.4.3. L'extension des types primitifs ($\mathcal{D}$) :

Une dernière extension, symbolisée par la lettre ($\mathcal{D}$), ajoute le support des types primitifs. Cette extension augmente **AL** d'un second domaine d'interprétation $\boldsymbol{\Delta_D^I}$ disjoint avec $\boldsymbol{I(\Delta)}$ et qui représente l'ensemble des valeurs de type primitif. Le domaine $\boldsymbol{\Delta_D^I}$ définit plusieurs sous-domaines tels que les entiers, les chaînes de caractères, les entiers positifs, etc. Les éléments de ces domaines se nomment individus primitifs.
De plus, l'extension ajoute un nouveau type de rôle, défini comme une relation binaire sur $\boldsymbol{I(\Delta)} \times \boldsymbol{\Delta_D^I}$ est appelé rôles à valeurs primitives. La lettre $\mathcal{U}$ représente l'ensemble de ces rôles. Cet ajout autorise la spécification d'assertions de rôle tel que *u (a, 205006007)* et *v (a, "Jean Jacques")*.

2.5. Services d'inférence :

Un système de LD stocke non seulement des terminologies et des assertions, mais offre également les services d'inférence. La charge principale du raisonnement dans une LD est de découvrir des connaissances implicites à partir des connaissances explicites par l'inférence.
Les services d'inférence sont également réalisés sur toutes la T-Box et ainsi que sur l'A-Box. Les deux inférences de base dans les LDs 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 basées sur des calculs de relations de subsomption.

2.5.1. Subsomption :

Cette relation permet d'ordonner les concepts dans une hiérarchie. Woods décrit cinq types de subsomptions et les relations qui existent entre ces différents types. Nous définissons que deus types qui sont la subsomption extensionnelle et structurelle.

2.5.1.1. Subsomption extensionnelle :

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

2.5.1.2. 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. Le concept Enfant possède toutes les propriétés du concept Etre-Humain plus les des propriétés qui lui sont spécifiques (ex : âge < 12). La subsomption structurelle correspond au point de vue algorithmique de la subsomption. Lors de la création d'une hiérarchie ou de l'ajout de nouveaux concepts, on effectue des calculs de *subsomption structurelle*.

2.5.2. 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.

2.5.2.1. 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*).

2.5.2.2. 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.

2.5.2.3. 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.

2.5.2.4. Détection d'incohérence :

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

2.5.3. Inférences assertionnelles :

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

2.5.3.1. 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.
Pour savoir si un objet *o1* est une instance d'un concept *C*, on calcule le concept abstrait *Ao1*, on teste ensuite si *C* subsume *Ao1*. Si c'est le cas, on déduit alors qu'o1 est une instance du concept *C*.

2.5.3.2. 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.

3. Conclusion :

Nous avons présenté dans ce chapitre la logique de description car notre travail repose essentiellement sur cette logique. 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. Les DLs sont responsables de plusieurs notions de base dans la Représentation de la Connaissance et du raisonnement.

Dans ce chapitre nous avons étudié les logiques de description classiques d'une manière générale. Nous avons vu ses objets, ses deux niveaux de description ainsi que la logique minimale avec ses extensions et les services d'inférence offerts par cette derniere.

Nous avons choisi d'appliquer notre modèle à un système d'information médical. Nous présentons dans le chapitre suivant une illustration du fonctionnement de notre modèle et le déroulement des mécanismes traités dans cette étude.

Chapitre 4

Formalisation d'Or-BAC en logique de description

La formalisation du modèle de contrôle d'accès dynamique et contextuel DL-OrBAC en logique de description nous donne un bon niveau d'expressivité. La syntaxe des DLs permet la prise en charge des notions de rôle, activité, vue et ainsi que le contexte [7].

Nous avons présenté dans les chapitres un et trois le modèle Or-BAC et la logique de description.

Nous détaillons dans ce qui suit le modèle DL-OrBAC à travers la construction de la base de connaissance qui se compose d'une TBox et d'une ABox, ainsi que la définition des mécanismes d'inférences.

1. Représentation des connaissances :

La modélisation d'un domaine avec les logiques de description (DL) se réalise en deux niveaux. Le premier est le niveau terminologique ou *TBox* qui décrit les connaissances générales d'un domaine alors que le second est le niveau factuel ou *ABox* qui décrit les individus en les nommant et en spécifiant en termes de concepts et de rôle [5].

Plusieurs *ABox* peuvent être associées à une même *TBox* ; chacune représente une configuration constituée d'individus et utilise les concepts et rôles de la *TBox* pour l'exprimer[8].

La description du modèle OrBAC en taxonomie DL se fait comme suit :

- Les entités du modèle (organisation, rôle, activité, vue, sujet, action, objet) sont traduites en concepts atomiques et deviennent les entités élémentaires de la TBox ;

- Les relations binaires se traduisent en rôles atomiques ;

- Pour les relations n-aires (>2), il n'existe pas d'équivalent au niveau de la logique de description. Ce problème est typique des langages de représentation de la connaissance. Il existe une approche appelée *la réification*, qui consiste à créer un concept pour la relation n-aires et une propriété pour chacun des rôles de la relation.

Le modèle OrBAC a été défini par ces auteurs en logique du 1er ordre [5], nous décrivant dans ce qui suit le modèle avec la logique de description.

1.1. TBox :

La *TBox* contient l'ensemble des concepts primitifs et les axiomes qui permettent de relier les différentes entités du système.
Comme nous l'avons déjà vu dans le chapitre 1 :

- ✓ L'*organisation* est l'entité centrale du modèle. L'organisation regroupe un ensemble de sujets dont chacun joue un rôle particulier.

- ✓ Un *rôle* associe un ensemble de *sujets* qui remplissent les mêmes fonctions et sur lesquels on applique les mêmes règles de sécurité. Nous donnons comme exemple : le sujet "mehdi" qui joue le rôle "Secrétaire" dans l'organisation "Service de cardiologie"[15].

- ✓ Une *vue* regroupe des *objets* qui satisfont une propriété commune. Par exemple, dans le domaine médical, une vue "dossier médical" correspond à l'objet "Dossier médical du patient".

- ✓ Une *activité* regroupe un ensemble d'*actions* qui ont le même principe. Nous donnons comme exemple, l'activité "Gestion" qui englobe l'ensemble des actions "Lire, écrire, modifier, supprimer".

- ✓ Les contextes sont utilisés pour spécifier les circonstances concrètes dans lesquelles les organisations accordent aux sujets des permissions de réaliser des actions sur les objets.

Dans notre système, nous considérons que :
· *Par défaut, le contexte est normal.*
· *Toute action qui n'est pas permise est interdite.*

1.2. Définition des relations :

1.2.1. La relation « habilite » :

Nous avons :

Sujet $\sqsubseteq$ T

Rôle $\sqsubseteq$ T

Organisation $\sqsubseteq$ T

$$Habilite \sqsubseteq (HabiliteS.Sujet) and (HabiliteR.Rôle) and (HabiliteOr.Organisation)$$

Les relations *HabiliteS, HabiliteR, HabiliteOr sont des relations binaires* tel que [8]:

- *HabiliteS :* relie le concept *Habilite* au concept *Sujet.*
- *HabiliteR :* relie le concept *Habilite* au concept *Rôle.*
- *HabiliteOr :* relie le concept *Habilite* au concept *Organisation*.

Exemple :

La connaissance "*mehdi*" est un sujet habilité au rôle "*Medecin*" dans l'organisation "*Beni-Messous*" est représentée par H1 qui est une instance du concept "*Habilite*".

La description est comme suit :

Habilite (H1) $\sqsubseteq$ *HabiliteS.Sujet (Mehdi)* and *HabiliteR.Role (medecin)* and *HabiliteOr.Organisation (beni-Messous).*

1.2.2. La relation « Utilise » :

Nous avons :

Objet $\sqsubseteq$ T

Vue $\sqsubseteq$ T

Organisation $\sqsubseteq$ T

Utilise $\sqsubseteq$ (UtiliseO.Objet) and (UtiliseV.Vue) and (UtiliseOr.Organisation)

Les relations: UtiliseO, UtiliseV, UtiliseOr sont des relations binaires tel que :[8]

- UtiliseO: relie le concept Utilise au concept Objet.
- UtiliseV: relie le concept Utilise au concept Vue.
- UtiliseOr: relie le concept Utilise au concept Organisation.

Exemple :

La connaissance "Fichier1" est un objet utilisé dans la vue "Fichier" dans l'organisation "USTHB" est représentée par U1 qui est une instance du concept "Utilise"

La description est comme suit :

Utilise (U1) $\sqsubseteq$ UtiliseO.Objet (Fichier1) and UtiliseV.Vue (Fichier) and UtiliseOr.Organisation (USTHB).

1.2.3. La relation « Considere » :

Nous avons :

Action $\sqsubseteq$ T

Activité $\sqsubseteq$ T

Organisation $\sqsubseteq$ T

Considère $\sqsubseteq$ *(ConsidèreAc.Action)* and (*ConsidèreAv.Activité)* and (*ConsidèreOr.Organisation)*

Les relations: *ConsidèreAc, ConsidèreAv, ConsidèreOr sont des relations binaires* tel que [8]:

- *ConsidèreAc:* relie le concept *Considère* au concept *Action.*
- *ConsidèreAv:* relie le concept *Considère* au concept *Activité.*
- *ConsidèreOr:* relie le concept *Considère* au concept *Organisation*.

Exemple :

La connaissance "*Ecrire*" est une action utilisé dans l'activité "*Modifier*" dans l'organisation "*USTHB*" et représentée par C1 qui est une instance du concept.

La description est comme suit :

Considère (C1) $\sqsubseteq$ *ConsidèreAc.Action (Ecrire)* and *ConsidèreAv.Activité (Modifier)* and *ConsidèreOr.Organisation (USTHB).*

1.2.4. La relation « Definit» :

Nous avons :

Sujet $\sqsubseteq$ T

Action $\sqsubseteq$ T

Objet $\sqsubseteq$ T

Contexte $\sqsubseteq$ T

Organisation $\sqsubseteq$ T

Definit $\sqsubseteq$ *(DefinitS.Sujet)* and (*DefinitA.Action)* and (*DefinitO.Objet) and (DefinitC.Contexte) and (DefinitOr.Organisation).*

Les relations: DefinitS, DefinitA, DefinitO, DefinitC, DefinitOr *sont des relations binaires* tel que [8]:

- DefinitS*:* relie le concept *Definit* au concept *Sujet.*
- DefinitA*:* relie le concept *Definit* au concept *Action.*
- DefinitO*:* relie le concept *Definit* au concept *Objet.*
- DefinitC*:* relie le concept *Definit* au concept *Contexte.*
- DefinitOr*:* relie le concept *Definit* au concept *Organisation.*

Exemple :
Le sujet « Mehdi », l'action « écrire » et l'objet « fichier1 » sont définis dans l'organisation « USDB » dans le contexte « normal » et représentée par D1 qui est une instance du concept definit.
La description est comme suit :
Definit (C1) $\sqsubseteq$ *DefinitS.Sujet (Mehdi)* and *DefinitA.Action (écrire)* $\sqcap$*DefinitO.Objet(fichier)* and *DefinitC.Contexte(normal)* and *ConsidèreOr.Organisation (USDB).*

1.2.5. L'attribution des permissions abstraites :

Cet axiome définit une relation entre les entités abstraites : organisation, rôle, activité et vue. Une permission par défaut est donné à un rôle **R** dans une organisation **Or** pour effectuer l'activité **A** sur la vue **V**.

Permission $\sqsubseteq$ *(PermissionR.Rôle)* and *(PermissionAv.Activité)* and *(PermissionV.Vue)* and *(PermissionOr.Organisation)* and *(PermissionC.Context)*

Les relations : *PermissionR, PermissionAv, PermissionV* et *PermissionOr* sont des relations binaires tel que :

- *PermissionR :* relie le concept *Permission* au concept *Role.*
- *PermissionAv :* relie le concept *Permission* au concept *Activité.*
- *PermissionV :* relie le concept *Permission* au concept *Vue.*
- *PermissionOr :* relie le concept *Permission* au concept *Organisation.*
- *PermissionC :* relie le concept *permission* au concept *Contexte.*

Exemple :

dans l'organisation "USDB", toute personne qui joue le rôle "Enseignant" à le droit d'effectuer l'activité "Modifier " sur la vue " Bulletin " dans le contexte " Normal" . Ceci est représenté par P1 qui est une instance du concept "Permission" décrit par l'expression suivante :

Permission(P1) $\sqsubseteq$ *PermissionR.Rôle(Enseignant)* and *PermissionAv.Activité(Modifier)* and *PermissionV.Vue(Bulletin)* and *PermissionOr.Organisation(USDB) and PermissionC.Contexte(Normal).*

1.2.6. Les permissions concrètes :

Une permission concrète implémente les entités concrètes de notre système qui sont : les sujets, les actions et les objets.

Est-Permis $\sqsubseteq$ *(Est-PermisS.Sujet)* and *(Est-PermisAc.Action)* and *(Est-PermisO.Objet)*

Où, *Est-PermisS, Est-PermisAc* et *Est-Permis* sont des relations binaires tel que :

_ *Est-PermisS :* relie le concept *Est-Permis* avec le concept *Sujet*.

_ *Est-PermisAc :* relie le concept *Est-Permis* avec le concept *Action.*

_ *Est-PermisO :* relie le concept *Est-Permis* avec le concept *Objet.*

Exemple :

Le sujet "Mehdi" a la permission de faire l'action "Ecrire" sur l'objet "Fichier1". Nous définissons EP1 une instance du concept Est-Permis et nous écrivons :

Est-Permis(EP1) $\sqsubseteq$ *Est-PermisS.Sujet (Mehdi)* $\sqcap$ *Est-PermisAc.Action (Ecrire)* $\sqcap$ *Est-PermisO.Objet (Fichier1).*

1.2.7. La relation sous-role :

Sous_role $\sqsubseteq$ *(Sous_role1.Role)* and *(Sous_role2.Role)* and *(Sous_RoleOr.Organiation)*

Où, *Sous_Role1, Sous_Role2, Sous_RoleOr* sont des relations binaires tel que :

_ *Sous_Role1:* relie le concept *Sous_Role1* avec le concept *Role*.

_ *Sous_Role2:* relie le concept *Sous_Role2* avec le concept *Role.*

_ *Sous_RoleOr:* relie le concept *Sous_RoleOr* avec le concept *Organisation.*

Exemple :

Le role "Medecin" est un sous-Role du role "Chirurgien" dans l'organisation Beni-messous

Sous_role(EP1) $\sqsubseteq$ *sous_Role1 (Medecin)* and *Sous_Role2 (Chirurgien)* and *sous_RoleOr(Beni-Messous).*

.

1.2.8. La relation sous-vue :

Sous_Vue $\sqsubseteq$ *(Sous_Vue1.Vue)* and *(Sous_Vue2.vue)* and *(Sous_VueOr.Organisation)*

Où, *Sous_Vue1, Sous_Vue2, Sous_vueOr* sont des relations binaires tel que :

_ *Sous_Vue1:* relie le concept *Sous_Vue1*avec le concept *vue.*

_ *Sous_Vue2:* relie le concept *Sous_Vue2* avec le concept *Vue.*

_ *Sous_VueOr:* relie le concept *Sous_vueOr* avec le concept *Organisation.*

Exemple :

La vue "Bulletin" est une sous-Vue de la Vue "Dossier administratif" dans l'organisation USDB

Sous_Vue (SV1) $\sqsubseteq$ *Sous_Vue1 (Bulletin)* and *Sous_Vue2 (dossier administratif)* and *sous_VueOr(USDB).*

1.2.9. La relation sous-activite :

Sous_Activite $\sqsubset$ *(Sous_Activite1.Activite)* and *(Sous_Activite2.Activite)* and *(Sous_ActiviteOr.Organisation)*

Où, *Sous_Activite1, Sous_Activite2, Sous_ActiviteOr* sont des relations binaires tel que :

_ *Sous_Activite1:* relie le concept *Sous_Activite1* avec le concept *Activite*.

_ *Sous_Activite2:* relie le concept *Sous_Activite2*avec le concept *Activite.*

_ *Sous_ActiviteOr:* relie le concept *Sous_ActiviteOr* avec le concept *Organisation.*

Exemple :

L'activite "modifier" est une sous-Activite de l'activite"gestion" dans l'organisation USDB

Sous_Activite (SA1) $\subseteq$ *Sous_Acitvite1 (modifier)* and *Sous_Acitvite2 (gestion)* and *sous_ActiviteOr(USDB)*

1.3. La délégation :

Nous définissons dans cette section de notre modèle la délégation. Nous montrons. La délégation a des caractéristiques comme a été montré dans le chapitre 2 ; nous présentons dans cette section la totalité, la permanence et la monotonie.

L'approche que nous suggérons pour gérer la délégation est l'utilisation de la notion de *contexte* et des *vues administratives.*

1.3.1. La délégation partielle :

Nous définissons deux vues administratives: *Licence_delegation et Role_delegation*. Ces vues sont utilisées, respectivement pour déléguer des droits (Délégation partielle) et les rôles (délégation totale) et elles sont définis comme suit:

Logique du 1er rodre :
Ppermission(GR, A, O, C): - Utilise(L, licence_délégation), Benificiaire (L, GR), Privilège (L, A), Cible (L, O), Contexte (L, C).

Logique de description :
Utilise ⊓ Benificiaire ⊓Privilege ⊓Cible ⊓Contexte2 ⟶Ppermission

Tel que :

- *Utilise* ⊑ *(UtiliseL.Licence)* and *(UtiliseC.Cible)* .
- *Benificiaire* ⊏ *(BenificiaireL.Licence)* and *(BenificiaireS.Sujet)* .
- *Privilege*⊏ *(PrivilegeL.Licence)* and *(PrivilegeA.Action)* .
- *Cible* ⊏ *(CibleL.Licence)* and *(CibleO.Objet)* .
- *Contexte2* ⊏ *(contexteL.Licence)* and *(ContexteC.Contexte).*
- *Ppermission* ⊏*(PpermissionS.Sujet)* and *(PpermissionA.Action)* and *(PpermissionO.Objet) and (PpermissionC.Contexte).*

L'écriture précécente explique que pour qu'une délégation soit vérifiée il faut créer les deux vues *Licence_delegation et Role_delegation* et que toutes les autres relations soient vérifiées.

1.3.2. La délégation totale:

La logique du 1er ordre :
Hablite (GR; Rôle): - Utilise (RD, Role_delegation), Assignee(RD, GR), Assignement (RD, Rôle).

La logique de description :
Utilise ⊓ Assignee ⊓ Assignement ⟶ Habilite

Tel que :

- *Utilise* ⊏ *(UtiliseRD.Role_Delegation)* and *(UtiliseR.Role) and (UtiliseOr.Organisation).*
- *Assignee* ⊏ *(AssigneeRA.Role_Assignee)* and *(AssigneeS.Sujet).*
- *Assignement* ⊏ *(AssignementRA.Role_Assignee)* and *(AssignementR.Role) .*
- *Habilite* ⊏ *(HabiliteS.Sujet)* and *(HabiliteR.Role) and (HabiliteOr.Organisation) .*

L'insertion d'un objet dans la vue *délégation_licence* ou dans *Role_Delegation* permettra au sujet de respectivement déléguer l'autorisation et le rôle à un bénéficiaire. Donc pour gérer la politique délégation on doit définir quel sujet (Rôle ou utilisateur) possède l'accès à ces vues et dans lequel contexte. Cela est défini par des faits ayant la forme suivante:

Ppermission (gr, déléguer, Délégation_Licence, Contexte).

Ppermission (gr, délégué, Role_delegation, contexte).

Pour illustrer cette approche, nous considérons une situation où il ya deux utilisateurs : Hamza, un professeur et Hafida, sa secrétaire. Le rôle secrétaire n'est généralement pas permis à accéder aux notes des étudiants. Cependant, Hamza décide de déléguer à Hafida une autorisation de mettre à jour les notes des étudiants. De toute évidence, Hamza doit avoir une autorisation de déléguer ce droit.

Pour cela, l'administrateur doit d'abord créer la vue administrative:

Logique du 1er ordre :
Utilise (L, note_délégation): - Utilise (L, licence_délégation), Privilège (L, mise_à_jour), Cible (L, notes_etud).

Logique de description:
Utilise $\sqcap$ Privilege $\sqcap$ Cible $\longrightarrow$ Utilise

Ou Utilise, Privilege, Cible sont déclarés précédemment (voir 1.3.1), et on déclare mise_a_jour et notes_etud comme étant des instances des classes Action et Objet respectivement.

La vue note_delegation est dérivée de la vue licence_délégation et ne contient que des licences à des mises à jour des notes des étudiants.

Ppermission (professeur, déléguer, note_delegation, nominal).

où nominal représente le contexte par défaut.

Hamza peut déléguer à hafida une autorisation de mettre à jour les notes de ses étudiants.

Ppermission (hafida, mise_a_jour, note_etud_hamza, nominale).

Hafida peut mettre à jour les notes des étudiants de hamza dans le contexte nominale.

1.3.3. La délégation permanente/ temporelle:

La permanence se réfère aux types de délégation en fonction de leur durée de temps. En effet, dans certaines circonstances, la délégation s'applique uniquement à titre temporaire et sera automatiquement révoquée après un délai donné. Ce peut être modélisé dans notre approche en utilisant simplement un contexte temporel. Pour plus de détails sur la définition du contexte, voir [6].

Dans l'exemple précédent, il n'existe aucune spécification temporelle d'une durée de temps dans la délégation, alors la délégation est permanente.

Maintenant, si nous supposons que hamza veut déléguer à Hafida la permission de mettre à jour les notes de ses élèves pendant ses vacances, alors il doit le préciser dans le contextecomme suit :

Ppermission (hafida , mise_à_jour, note_etud_hamza, vacances_hamza).

1.3.4. La délégation monotone/non-monotone:

La délégation Monotone signifie que lors d'une délégation le délégant maintient la permission qu'il a déléguée, tel que décrit dans l'exemple des sections précédentes.

Cependant, avec une délégation non-monotone, le concédant perd cette autorisation pour la durée de la délégation.

Pour modéliser la délégation non-monotone, nous définissons la vue *licence_transfére* comme suit:

Logique du 1er ordre :
Utilise (L, licence_delegation): - utilise (L, licence_transfert).

Interdiction (S, A, O, C, Max): - Utilise (L, licence_transfert), Delegant (L,S), Privilege (L, A), Cible (L, O), Contexte2 (L, C).

Logique de description :
Utilise ⊓ Delegant ⊓ Cible ⊓ Contexte2 ⟶ Interdiction

Ou Utilise, Cible, Contexte2 sont déclarés précédemment (voir 1.3.1). et delegant est déclaré comme suit :

- *Delegant* ⊏ *(DelegantL.Licence)* and (*delegantS.Sujet).*

Licence_transfert est une sous-vue de Licence_delegation. Ainsi, l'insertion d'un objet dans cette vue créera une nouvelle autorisation au bénéficiaire. En outre, il permettra de créer une interdiction à la personne qui a délégué ses permissions. Par conséquent, le délégant perd l'autorisation qu'il a déléguée.

A noter que le délégant perd ses permissions seulement pour le temps de la délégation.

1.4. La révocation :

La révocation est un aspect très important de la délégation, elle est définie comme suit :

Logique du 1er rodre :
Interdiction(GR, A, O, C): - Utilise(L, licence_revocation), Benificiaire (L, GR), Privilège (L, A), Cible (L, O).

Logique de description :
Utilise ⊓ Benificiaire ⊓ Privilege ⊓ Cible ⟶ Interdiction

Ou Utilise, Cible, Benificiaire, Privilege sont déclarés précédemment (voir 1.3.1). et interdiction est déclaré comme suit :

- *interdiction* ⊏*(InterdictionS.Sujet)* and (*InterdictionA.Action)* and (*interdictionO.Objet).*

1.5. Définition des règles de sécurité :

La relation :

(Permission ⊓ Habilite ⊓ Utilise ⊓ Considère) · ⟶ est-permis

Ppermission ⟶ est-permis

Est vérifiée **Si** :

- Une permission par défaut existe pour un rôle **R**, une activité **Av** et une vue **V** dans une organisation **Or**.
- Un sujet **S** est habilité dans ce rôle **R**.
- Un objet **O** est utilisé dans cette vue **V**.
- Une action **Ac** implémente cette activité **Av** (Considère).

Ou :

- Une Délégation faite par le Sujet S1 qui joue le Rôle R1 au Sujet S2 qui joue le Rôle R2

Alors une permission concrète peut être déduite. Autrement dit le sujet **S** a le droit de faire l'action **Ac** sur l'objet **O**

1.6. ABox :

La ABox contient tous les individus de la base de connaissances. Dans notre cas, la ABox va contenir les entités concrètes qui sont l'ensemble des *sujets* de l'organisation, par exemple, Karim, Ali, etc., l'ensemble des *actions* des activités de l'organisation, prenons comme exemple, lire, écrire, supprimer, etc. et tous les *objets* des vues introduites dans l'organisation, par exemple, dossier médicale du patient, ordonnance, etc.

2. Conclusion :

Dans cette partie nous avons établi la modélisation formelle d'un modèle de contrôle d'accès dynamique avec la logique de description où les autorisations sont attribuées aux sujets selon leur rôle dans l'organisation et le contexte dans lequel ils se trouvent.

L'objectif principal de la logique de description est de raisonner efficacement pour minimiser les temps de réponse, ainsi la principale qualité des LD réside dans leurs algorithmes d'inférence dont la complexité est souvent inférieure aux complexités d'autres logiques ainsi sur sa très grande expressivité.

Nous avons défini la TBox, les différents axiomes qui nous permettent la construction de la base de connaissances et les différents éléments constituant la ABox (rôle, activité, vue, sujet, action, objet).

Nous avons présenté aussi dans ce chapitre les différents mécanismes d'inférence qui sont implémentés par notre modèle.

Nous avons choisi d'appliquer notre modèle à un système d'information médical. Nous présentons dans le chapitre suivant une illustration du fonctionnement de notre modèle et le déroulement des mécanismes traités dans cette étude.

Chapitre 5

Implémentation

Dans les parties précédentes nous avons décrit et formaliser les principes de fonctionnement du processus de *Délégation* dans le modèle *Or-BAC* et les notions de base de *la logique de description*.

A fin d'expérimenter une situation de délégation dans ce modèle, nous avons utilisé une interface java et un langage d'ontologie **OWL** (Ontologie Web Langage) pour manipuler la base de connaissance qui est le noyau du système. D'autres langages ont été introduits pour les besoins de l'application que nous allons décrire ultérieurement.

1. Choix du logiciel :

1.1. Le langage OWL :

OWL (*ontologie Web Langage*) est un dialecte XML basé sur une syntaxe RDF (*ressource Descriptif Framework*), il fournit les moyens pour définir les ontologies web structurées. Il a été proposé par W3C *(world Wide web consortium)* et il est devenu une recommandation depuis janvier 2004.

OWL est un langage de description d'ontologie conçu pour la publication et le partage d'ontologies sur le web sémantique, il est basé sur la recherche effectuée dans le domaine de la logique de description. **OWL** peut être vu comme un format de fichier pour certaines logiques de description, il permet de décrire des ontologies, c'est-à-dire définir des terminologies pour décrire des domaines concrets. Une terminologie se constitue de concepts et de propriétés (aussi appelés *rôle* en logiques de description). Un domaine se compose d'instance de concepts

1.1.1. Les espèces d'OWL :

OWL permet, grâce à sa sémantique formelle basée sur une fondation logique largement étudiée, de définir des associations plus complexes des ressources ainsi que les propriétés de leurs classes respectives. **OWL** définit trois sous- langage, du moins expressif au plus expressif : **OWL-LITE, OWL-DL et OWL-FULL.**

- **OWL-LITE :** est le plus simple des sous-langages d'OWL, il a été conçu pour une mise en œuvre aisée, offrant aux utilisateurs un sous-ensemble fonctionnel qui leur permettra de démarrer avec OWL.

- **OWL-DL(Ontologies web langage for Description logic) :** Il a été conçu pour tenir compte du secteur existant de la logique de description et offrir un sous-ensemble du langage avec des propriétés de calcul souhaitable pour les systèmes de raisonnement. Il est plus complexe qu'OWL-LITE, permettant une expressivité bien plus importante.

- **OWL-FULL :** est la version la plus complexe d'OWL, il assouplit certaines contraintes exercées sur OWL-DL. OWL-FULL est destiné aux situations ou il est très important d'avoir un haut niveau de capacité de description.

Il existe entre ces trois sous-langages une dépendance de nature hiérarchique: toute ontologie OWL-LITE valide est également une ontologie OWL-DL valide, et toute ontologie OWL-DL valide est également une ontologie OWL-FULL valide.

En pratique, le langage **OWL** est conçu comme une extension de **RDF** *(Resource Description Framework)* et **RDFS** *(RDF Schéma)*, **OWL** est destine à la description de classes et de types de propriétés. De ce fait, il est plus expressif que RDF et RDFS, auxquels certains reprochent une insuffisance d'expressivité due à la seule définition des relations entre objets par des assertions. **OWL** apporte aussi une meilleure intégration, une évolution, un partage et une inférence plus facile des ontologies.

Dans le cadre de notre projet, nous avons utilise le langage OWL-DL.

1.2. Protégé :

Divers logiciels permettent de créer des ontologies. Nous pouvons citer *OntoEdit* qui crée des ontologies grâce à une interface, le tout est base sur XML. Cependant, pour la réalisation de ce projet, nous avons décidé de travailler sur protégé 2000. Nous avons utilisé la version de *protégé 3.5 beta*.

Avant de présenter notre travail, il semble donc intéressant de découvrir les caractéristiques du logiciel protégé[11].

Protège est un logiciel libre d'utilisation. Il est produit et mis à disposition par *l'Université de Stanford* au sein du laboratoire « *stanford Medical Informatics* » dans le cadre du programme « *National Center for Biomedical Ontology* ». Il est basé sur la technologie *java* avec une interface graphique très agréable et facile a prendre en main, tous les outils pour créer et manipuler facilement des ontologies dans divers formats, pour être plus claire ce n'est pas un outil dédié a **OWL,** mais un éditeur hautement extensible. Protégé dispose aussi de nombreux plugins, comme *OntoViz* (pour visualiser graphiquement les ontologies) et *Sprqltab* (pour interroger les bases de connaissance).

Il est également possible de raisonner sur les ontologies en utilisant un moteur d'inférence général tel que *JESS*. Ou des outils d'inférence spécifiques au web sémantique basés sur des logiques de description (DL), ces deux outils peuvent être facilement intégrés à Protégé. Les logiques de description permettent de définir les bases logiques des différents formalismes de représentation de la connaissance sur le plan de la représentation et sur le raisonnement.

Enfin, protégé permet d'extraire une ontologie créée sous un format **OWL**, il propose plusieurs formats de fichiers pour les ontologies dont OWL. Cette ontologie pourra donc être consultée avec un autre éditeur ou être affichée sur une page internet [11].

Ce qui nous intéresse dans notre application c'est le fragment *OWL-DL*, protégé permet de construire des ontologies *OWL-DL* respectant le formalisme de la description logique, le passage entre le model Or-BAC formalisé en DL a L'OWL-DL se fait automatiquement.

1.3. Langage de programmation :

Pour pouvoir manipuler tous ces outils et afin de concevoir notre application, nous avons choisi le langage le plus répandu et les plus utilisé : *JAVA*, puisque Protégé propose aux développeurs une *API java* pour manipuler et travailler avec les ontologies **OWL** ; il a été lui-même bâtis en utilisant *JENA* (un frame Work proposant un cadre de travail pour la manipulation des ontologies à bas niveau en java), pour cela nous avons choisis ce langage.

1.3.1. JAVA :

Java est un langage de programmation orienté objet, développé par *Sun Microsystems*. Il permet de créer des logiciels compatibles ave nombreux systèmes d'exploitation (Windows, Linux, Macintosh, Solaris). Ce langage peut être utilisé sur internet pour des petites applications intégrées à la page web (applet) ou encore comme langage serveur (jsp).

Le langage reprend en grande partie la syntaxe C++, très utilisé par les informaticiens. Néanmoins, Java a été épurée des concepts les plus subtils du C++ et à la fois les plus déroutants, tels que l'héritage multiple remplacé par l'implémentation des interfaces. Les concepteurs ont privilégie l'approche orienté objet de sorte qu'en Java, tout est objet a l'exception des types primitifs (nombres entiers, nombres a virgule flottante, etc.)

1.3.2. Environnement de développement Java utilisé:

Netbeans est un environnement de développement intègre(**IDE**) pour java, c'est aussi le seul qui supporte *OWLapi* (une api java pour manipuler et faciliter la programmation des ontologies en OWL-FULL et OWL-DL) chose qui a favorise son déploiement.

NetBeans est placé en open source par sun en juin 2000 sous licence **CDDL (Common Development and Distribution License).** En plus de java, NetBeans permet également de supporter différents autres langages, comme python, C, C++, XML et HTML. Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages web).

NetBeans est lui-même développé en java, ce qui peut le rendre assez lent et gourmand en ressources mémoires[17].

2. Opération de base sur la base de connaissances (Module de base de connaissance) :

Cela est basé sur le fonctionnement du protégé en utilisant les API de ce dernier. Notre application permet en premier lieu de :

1. Charger des fichiers OWL.
2. Ajouter, supprimer et afficher des individus appartenant aux différents concepts.
3. La délégation utilisant des concepts de délégation.
4. Résoudre le conflit.

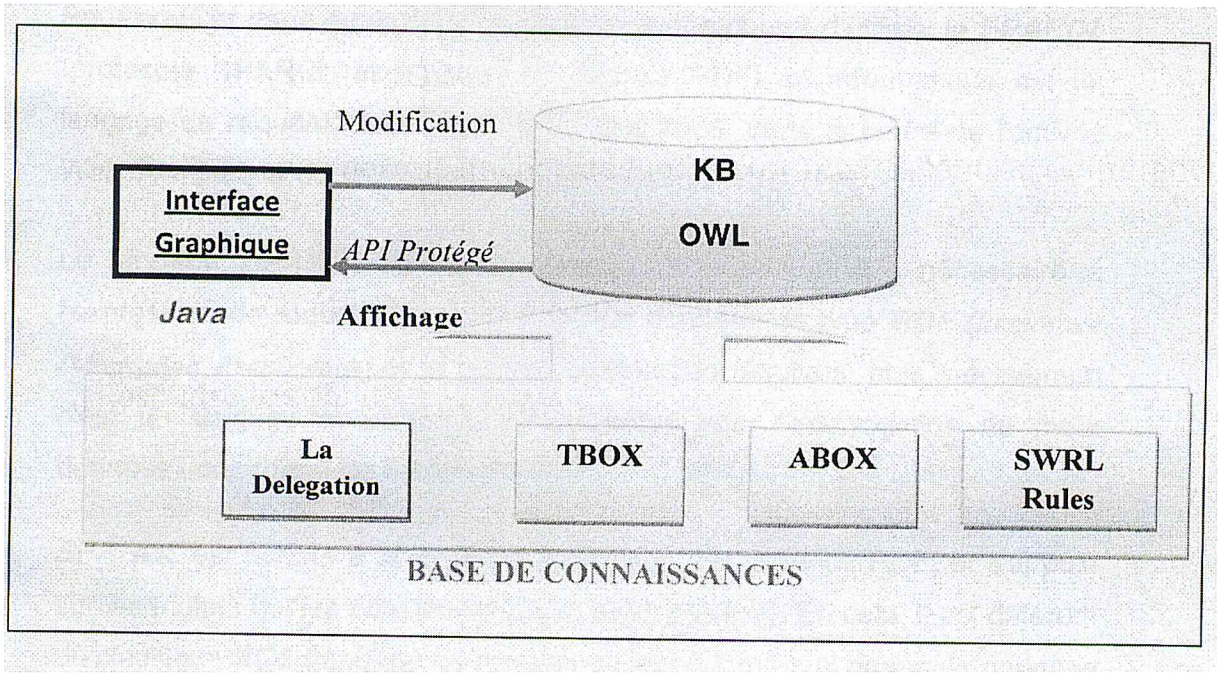


Figure 5.1 : Une base de connaissance

3. langage d'interrogation d'ontologies SPRQL (module d'interrogation) :

Le module d'interrogation présente le mécanisme utilisé dans notre application pour réaliser des interrogations complexes sur la base de connaissance. L'API de protégé s'est révélée limitée concernant l'interrogation des bases de connaissances en OWL (elle permet seulement d'afficher les individus appartenant à un concept donné), pour exploiter le fichier OWL nous avons besoin d'un langage de requêtes et de son moteur *« engine ».*

Il existe plusieurs langages pour interroger les bases de connaissance OWL tel que : SQWRL (Semantic Query-Enhanced Web Rule Language), SPRQL (Semantic Protocol And RDF Query Language), Nrql et le SWRL (Semantic Web Rule Language).

Dans notre application nous avons choisit d'utiliser le SPARQL (protocole SPARQL et langage de requêtes RDF).

Le langage *SPARQL* définit la syntaxe et la sémantique nécessaire à l'expression de requêtes sur des données de type *RDF* (Ressource Description Framework) et la forme possible des résultats, plus précisément c'est un langage et protocole de requêtes pour l'interrogation de métadonnées sous formes de fichiers *RDF*.

SPARQL permet d'exprimer des requêtes interrogatives ou constructives :

- Une requête **SELECT,** de type interrogative, permet d'extraire du graphe *RDF* un sous – graphe correspondant à un ensemble de ressources vérifiant les conditions définies dans une clause **WHERE.**

- Une requête **CONSTRUCT**, de type constructive, engendre un nouveau graphe qui complète le graphe interrogé.

Par exemple sur un graphe *RDF* contenant des informations généalogiques on peut par une requête **SELECT** trouver les parents ou grands-parents d'une personne donnée, et par des requêtes **CONSTRUCT** ajouter des relations frère-sœur, cousin-cousine, oncle-neveu, qui ne seraient pas explicitement déclarées dans le graphe initial.
Dans notre application seul les requêtes interrogatives de type **SELECT** on été utilisées.

3.1. Extrait du graphe RDF :

Les autres concepts et rôle suivent le même raisonnement, et le fichier résultant et le noyau de notre application, c'est un fichier dont l'extension est OWL et qui contient une **TBOX** et une **ABOX** vide.

Le code **RDF/XML** présenté dans la figure Ci-dessous définie le concept *est_prmis*

```
<owl:Class rdf:ID="Est_permis">
    <rdfs:subClassOf
rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:ID="est_permisA"/>
            </owl:onProperty>
            <owl:someValuesFrom rdf:resource="#Action"/>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:ID="est_permisS"/>
            </owl:onProperty>
            <owl:someValuesFrom rdf:resource="#Sujet"/>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:ID="est_permisO"/>
            </owl:onProperty>
              <owl:someValuesFrom rdf:resource="#Objet"/>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
```

Figure 5.2 : le code rdf/xml définissant le concepts est_permis

3.2. Exemple d'une requête SPARQL :

L'exemple présenté dans la figure 5.3 présente une requête SPARQL (en utilisant l'ontologie Orbac.owl)

```
PREFIX ontology-name:< http://www.owl-ontology.com/ontology1200573467.owl>

SELECT ? Habilite ?Organisation from ontology-name

WHERE {? Habilite ontology-name: HabiliteS ?Sujet}
```

Figure 5.3 : une requête SPARQL

On remarque la déclaration des espaces de noms en début, suivi de la requête proprement dite. Le nom des variables est précède d'un point d'interrogation ?.

La ligne **SELECT** permet de sélectionner l'ensemble des tuples, ou lignes de variables (Role1, Role2) correspondant aux contraintes de la clause **WHERE**.

Le module d'interrogation de notre modèle se présente dans la Figure 5.4.

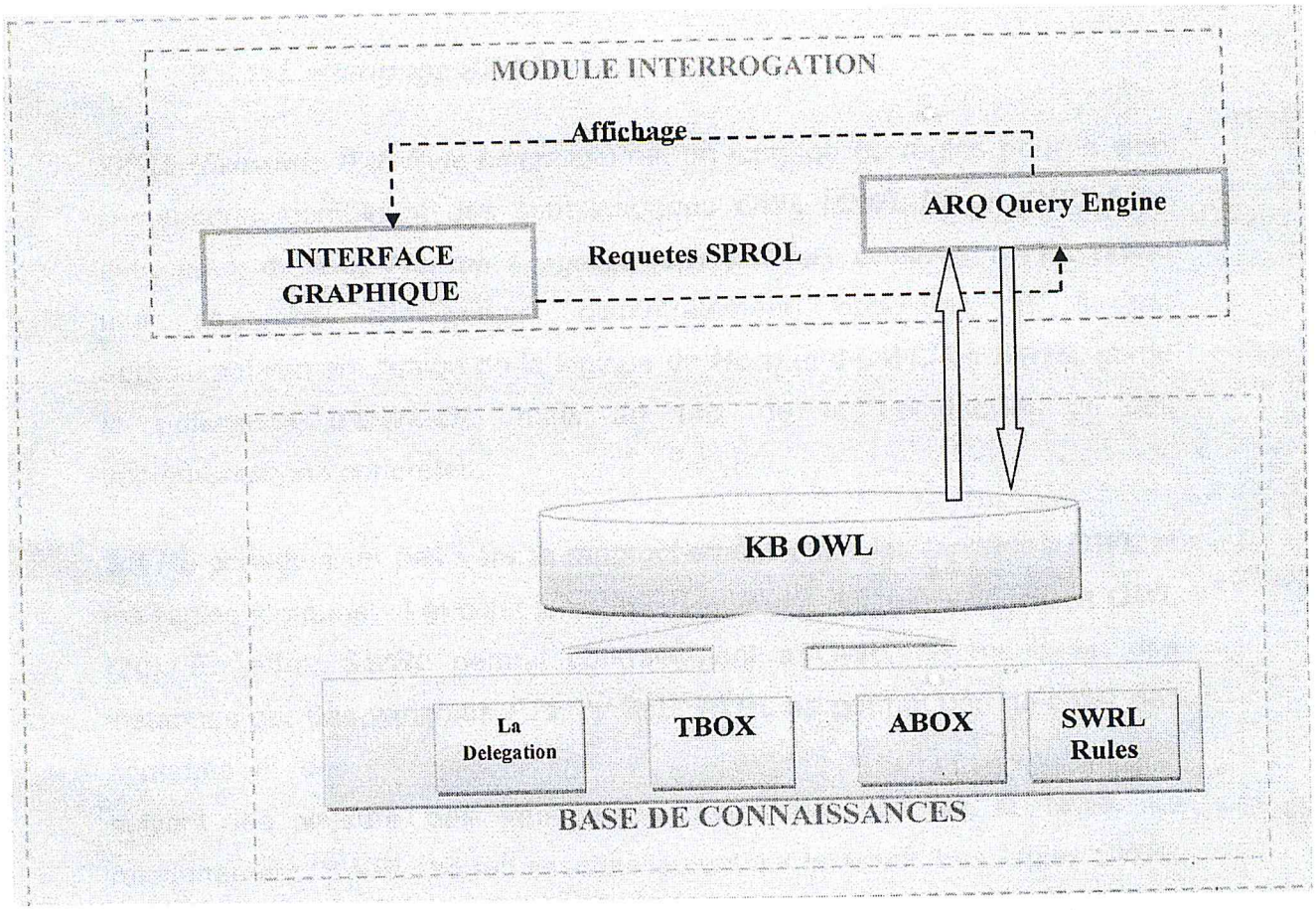


Figure 5.4 : Architecture du module Interrogation

4. Moteur d'inférences :

Un moteur d'inférence permet aux systèmes experts de conduire des raisonnements logiques et de dériver des conclusions a partir d'une base de connaissances.

4.1. Le langage SWRL :

SWRL (Semantic Web Rule Language) est un langage de règles pour le web sémantiques, combinant des sous-langages **OWL (OWL-DL et OWL-LITE)** avec ceux du Rule Markup Language(Unary/Binary Datalog). SWRL prend une approche d'intégration diamétralement opposée, et il est approximativement l'union de la logique de Horn et d'OWL. Le SWRL garde la puissance d'OWL-DL, mais au prix de la décidabilité et des implémentations concrètes.

SWRL constitue un pas vers le rapprochement entre les langages ***OWL*** et les règles logiques, il enrichit la sémantique d'une ontologie définit en ***OWL***. En effet, ***SWRL*** permet contrairement a ***OWL***, de manipuler des instances par des variables (?x ?y ?z). ***SWRL*** ne permet pas de créer des concepts ni des relations, il permet simplement d'ajouter des relations suivant les valeurs des variables et la satisfaction de la règle. Le raisonnement ***SWRL*** se fait essentiellement sur la ***Abox***. Les règles ***SWRL*** sont construites suivant ce schéma :

Antécédent ⟶ conséquent

Le fonctionnement d'une règle est basée sur le principe de satisfiabilité de l'antécédent ou du conséquent. Pour une règle, il existe trois cas de figure :

- L'antécédent et le conséquent sont définis. Si l'antécédent est satisfait alors le conséquent doit l'être.
- L'antécédent est vide, cela équivaut a un antécédent satisfait ce qui permet de définir des faits.
- Le conséquent est vide, cela est équivaut a un conséquent insatisfait, l'antécédent ne doit pas être satisfiable.

Dans l'exemple suivant la relation estOncleDe a été construite en ***OWL***, ***SWRL*** apporte la description de cette relation et relie les instances concernées.

Nous savons qu'une personne est un oncle, mais nous ne savons pas de qui. ***OWL*** ne permet pas de définir une relation qui représente le fait d'être oncle d'une personne. Avec **SWRL** c'est possible :

aEnfant(?x, ?Y) ^estFrereDe(?Z, ?x) ⟶ estOncleDe(?z, ?y).

SWRI est unique en étant une extension de OWL-DL, afin que les utilisateurs de OWL-DL puissent ajouter a leurs règles et a maintenir clairement les ontologies sémantiques.
Mais il est nécessaire de rappeler que le SWRL a des inconvénients, qui résident dans le fait que l'OWL ne supporte que les propretés binaires, et il en est de même pour le SWRL, ceci causera d'énorme problème dans l'implémentation.

4.2. Le moteur SWRL :

le moteur d'inférence que nous allons utiliser devra supporter le raisonnement a base de règle SWRL. Il y a plusieurs moteurs d'inférences disponibles gratuites ou bien payantes, mais notre choix va se faire seulement sur un moteur qui supportera le SWRL
Nous avons choisie le Jess car c'est le seul qui réponds aux 4 critères décisifs de notre choix qui sont : l'api java, le support des règles SWRL, le SWRL bridge et le raisonnement SWRI.

4.3. JESS (java Exper Shell System):

Nous avons choisi **Jess** en tant que premier candidat d'intégration pour le rédacteur de SWRL parce qu'il fonctionne de façon transparent avec Java, a une base étendu d'utilisation, il est bien documente et est très facile a employé et a configurer.

Jess est un moteur de règles écrit en **JAVA** qui est un outil de développement permettant la construction des systèmes experts(a base de règles et/ou d'objets)la vérification et la validation du système. Il peut s'intégrer a protège sous la forme d'un **plug-in**, réalisant ainsi une correspondance (**mapping**) en temps réel entre les instances, classes de la base de connaissance et les faits, en outre, plusieurs projets ont employé **Jess** avec succès dans l'environnement de **Protège** (par exemple **SWRLJessTab, SweetJess, JessTab**). Jess fournit une interface interactive de ligne de commande et un **Javabased Api** a son moteur de règle. Ce moteur peut être enfonce dans des applications java et fournit un temps d'exécution bidirectionnel flexible communicant entre les règles de Jess et le java.

Le système de **Jess** se compose d'une base de règles d'une base de fait, et d'un moteur d'exécution comme le montre la figure 5.5.

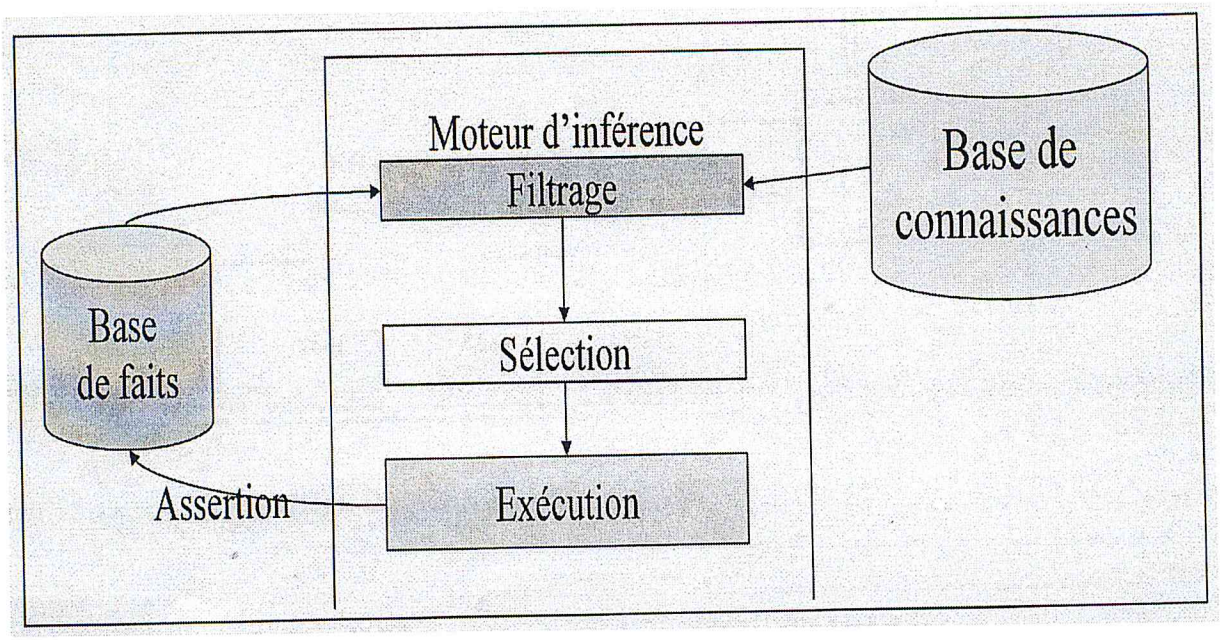


Figure 5.5 : Architecture d'un système a base de règles

5. *l'architecture du module d'inférence :*

Le mécanisme de fonctionnement du monde d'inférence est le suivant :

La base de connaissance (ensemble Tbox, Abox, SWRL rules, la délégation) est employée au Jess a travers le SWRL bridge, ensuite le jess effectue un raisonnement selon les règles SWRL et enrichit la base de connaissance avec les nouveaux faits a travers le SWRLbridge.

La figure 5.6 montre l'architecture du module d'inférence et montre son interaction avec le module de base de connaissance.

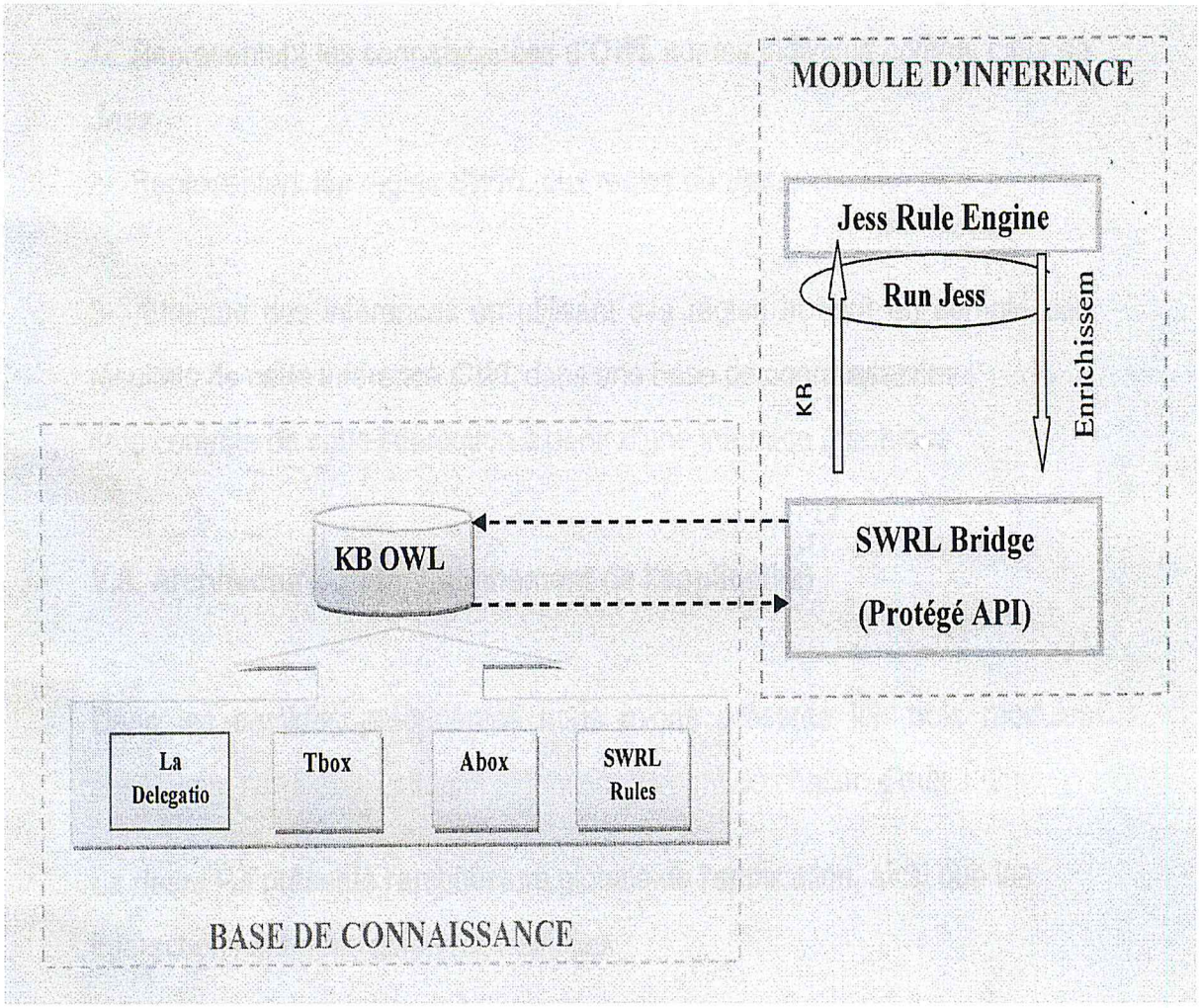


Figure 5.6 : Le moteur d'inference

6. *intégration du rédacteur de SWRL avec le moteur de règles Jess :*

Les règles ***SWRL*** raisonnent sur les individus ***OWL***. Quand une règle SWRL est exécutée elle peut créer de nouvelles classifications pour les individus existant. Par exemple, si une règle qui affirme qu'un individu doit être classe comme un membre d'une classe particulière, cet individu doit être un membre de cette classe[11].

De même, si une règles ***SWRL*** affirme que deux individus sont lies via une propriétés particulière, alors que cette propriété doit être associée a chaque individu qui satisfait la règle.

Ainsi, quatre principales taches doivent être accomplis pour permettre à jess d'interagir avec l'éditeur SWRL :

1- Représentent les connaissances d'OWL sur les individus comme cible de Jess.

2- Représentent les règles SWRL aux règles de Jess.

3- Effectue des inférences en utilisant ces règles et tien en compte des résultats de cette inférence OWL dans une base de connaissance.

4- Le contrôle de cette interaction a partir d'une interface graphique.

7. *Travail réalisé :*

Nous allons à présent donner les étapes par lesquelles nous sommes passé lors de la réalisation de notre travail :

Exemple :

Nous avons choisi d'implémenter la relation Habilite en utilisant protège.

- ***Etape 1 :***

Nous allons définir les classes qui sont : Sujet, Rôle, Organisation.
Puis les Règles en OWL-DL

- ***(HabiliteS some Sujet)and (HabiliteR some Role)and (HabiliteOr some Organisation)***

Comme le montre la figure 5.7

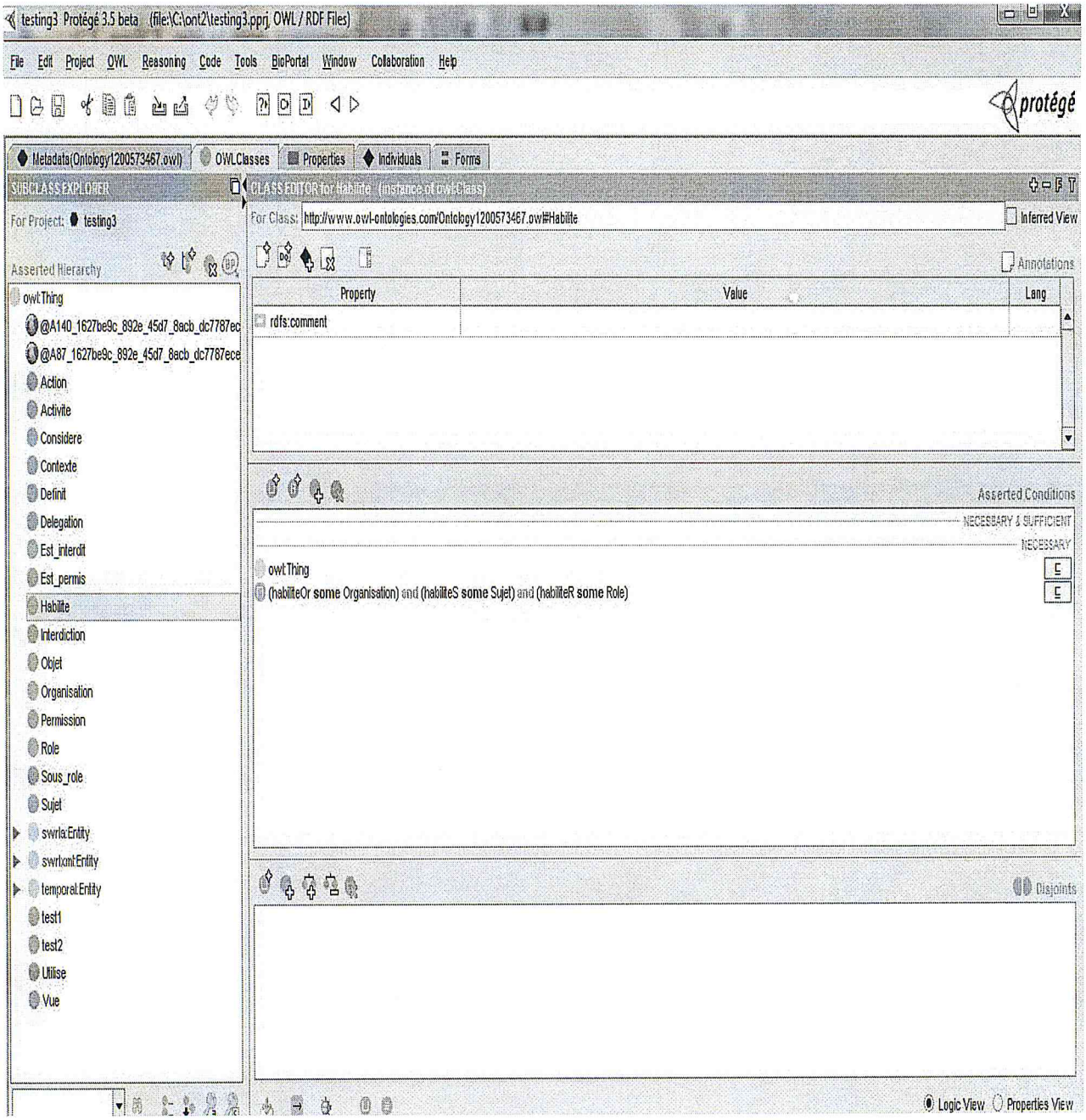


Figure 5.7 : Onglet classes dans OWL

➢ ***Etpae 2 :***

Dans cette étape nous définissons les relations, leurs rangs et leurs domaines. Les relations sont définies comme des objets dans protège, nous prenons par exemple la relation **HaniliteS.**

Cette relation a pour rang : **Habilite** et pour domaine : **Sujet.**

La figure 5.8 représente l'onglet propriété dans protège qui représente la relation HabiliteS et ses propriétés.

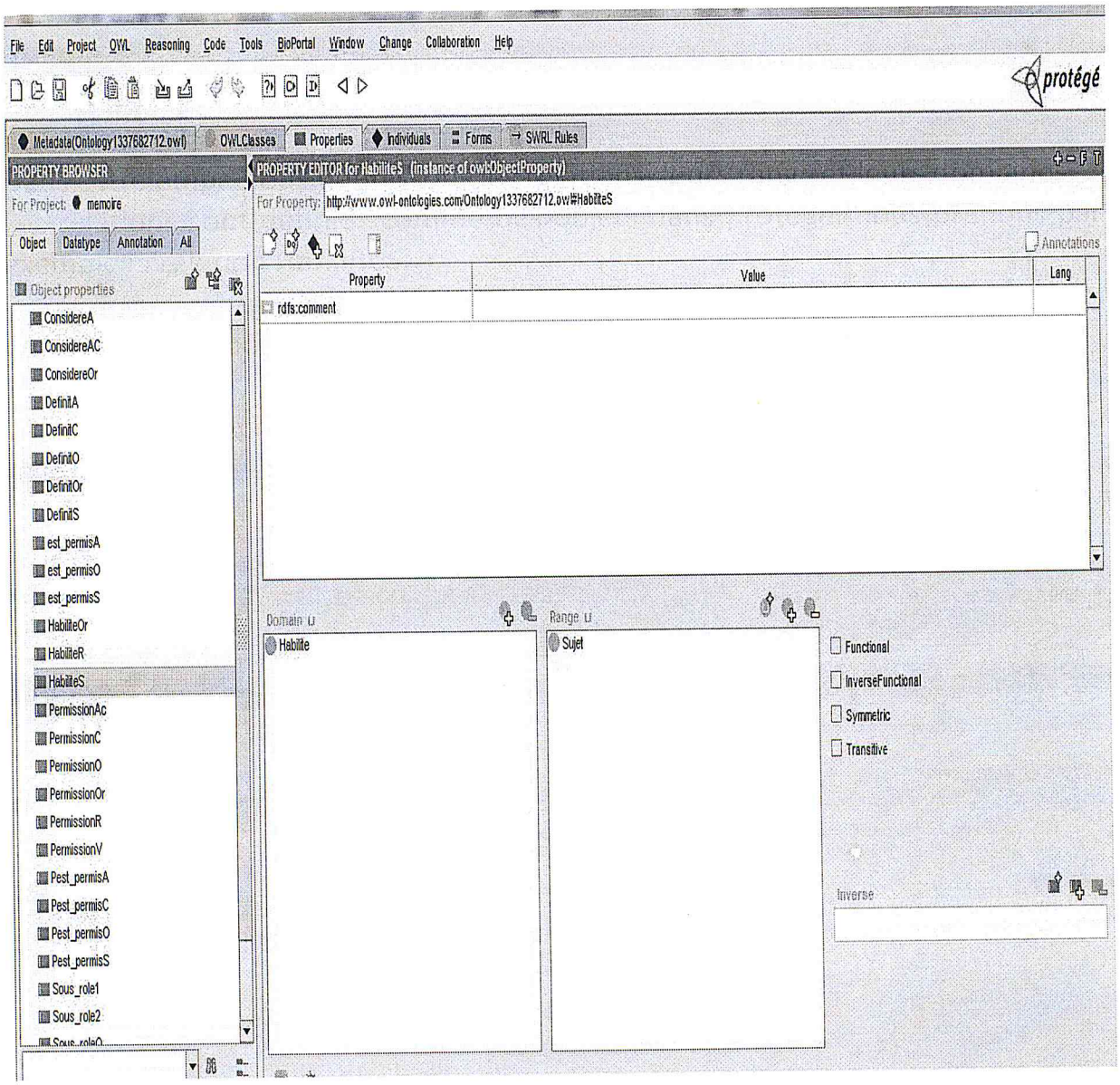


Figure 5.8 : La relation HabiliteS

➢ ***Etape 3:***

Cette étape est la dernière étape concernant la relation Habilite dans laquelle nous allons présenter la restriction su la classe Habilite.

La figure 5.9 représente la restriction dans protège sur le concept Habilite.

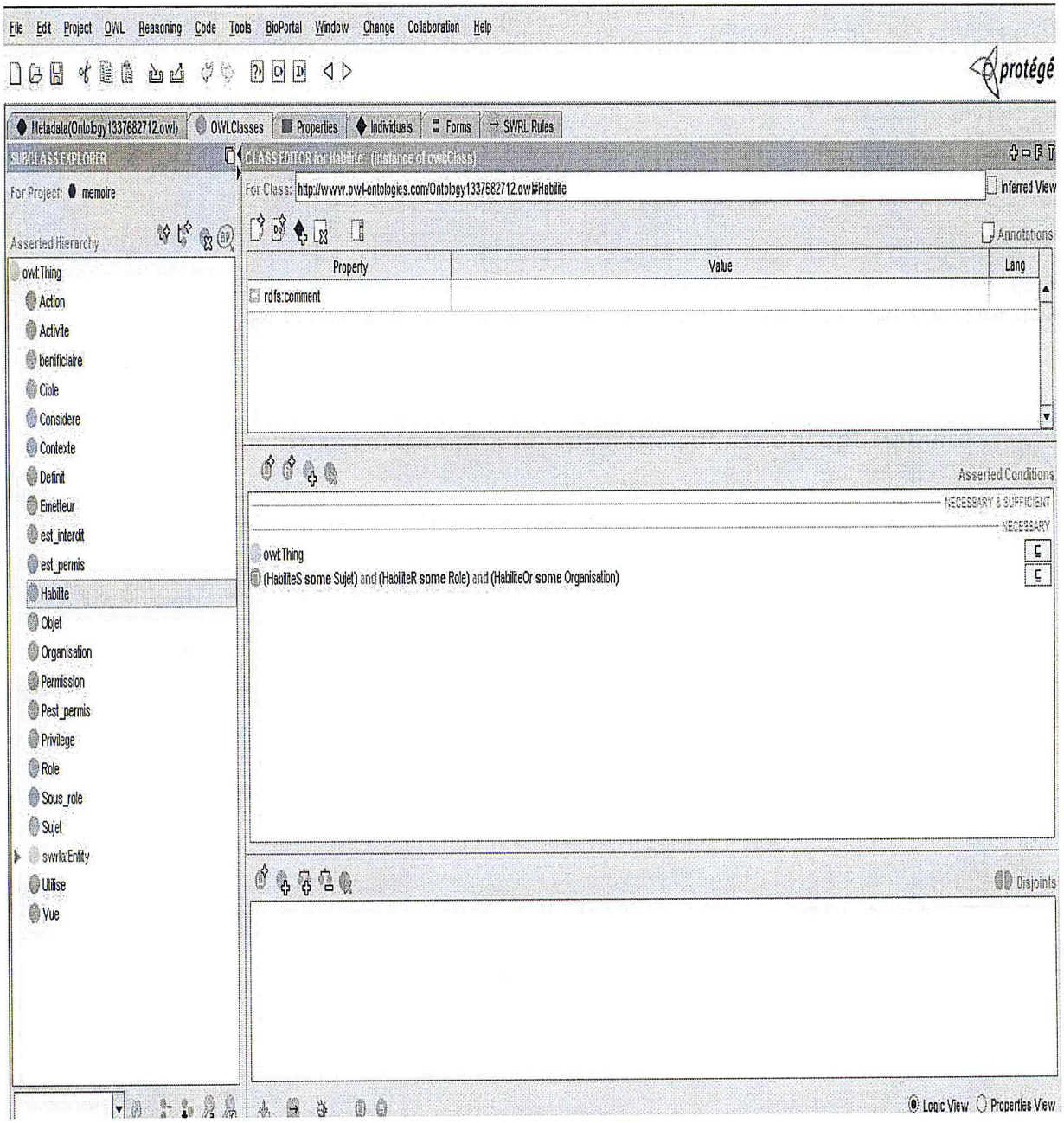


Figure 5.9 : La Restriction sur habilite

8. Présentation de l'application :

Notre programme fournit une interface qui nous permet de spécifier les politiques du modèle Or-BAC et la délégation.

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

8.1. La fenêtre principale :

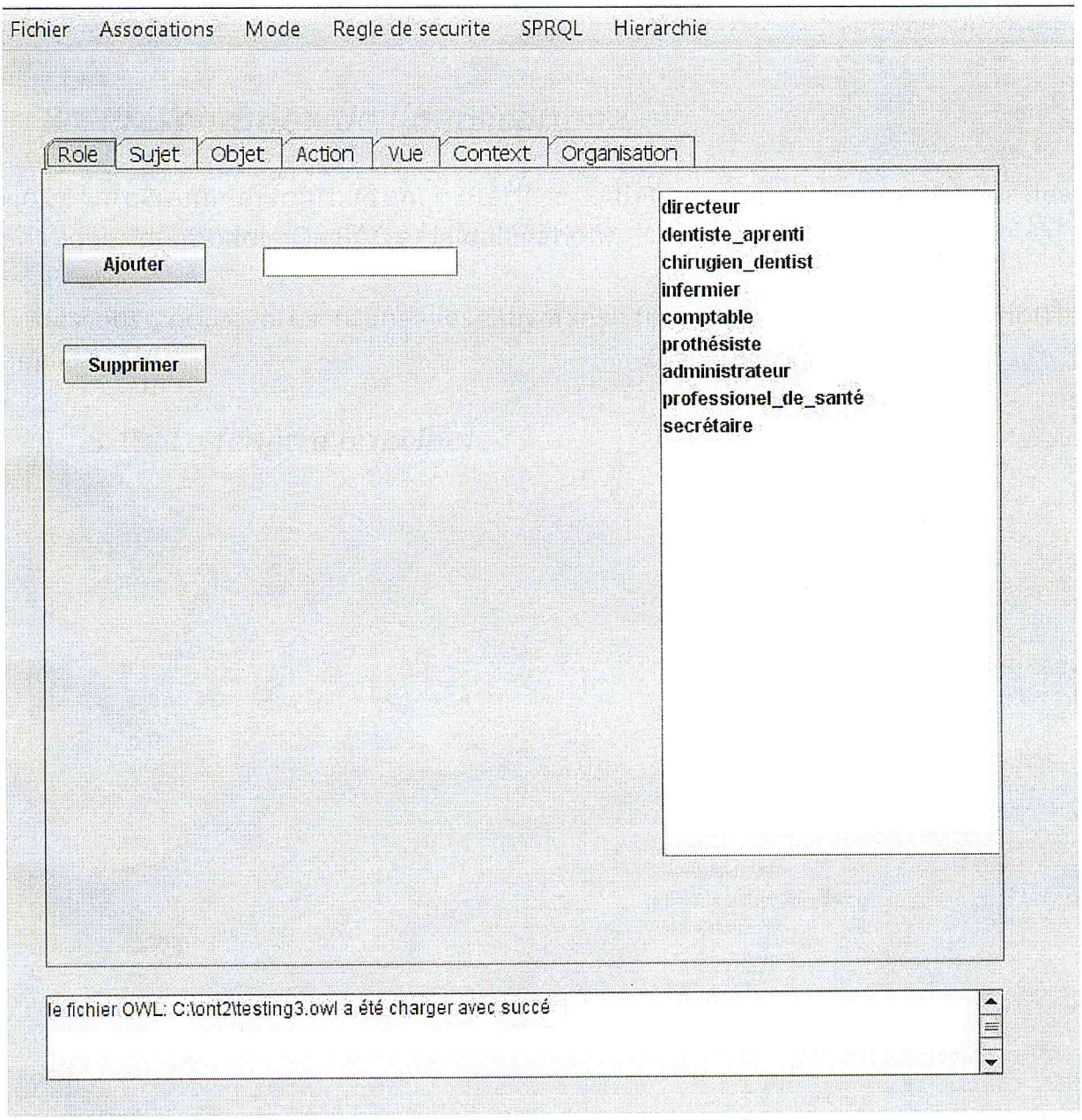


Figure 5.10 : La Fenêtre principale

8.2. La fenêtre Habilite :

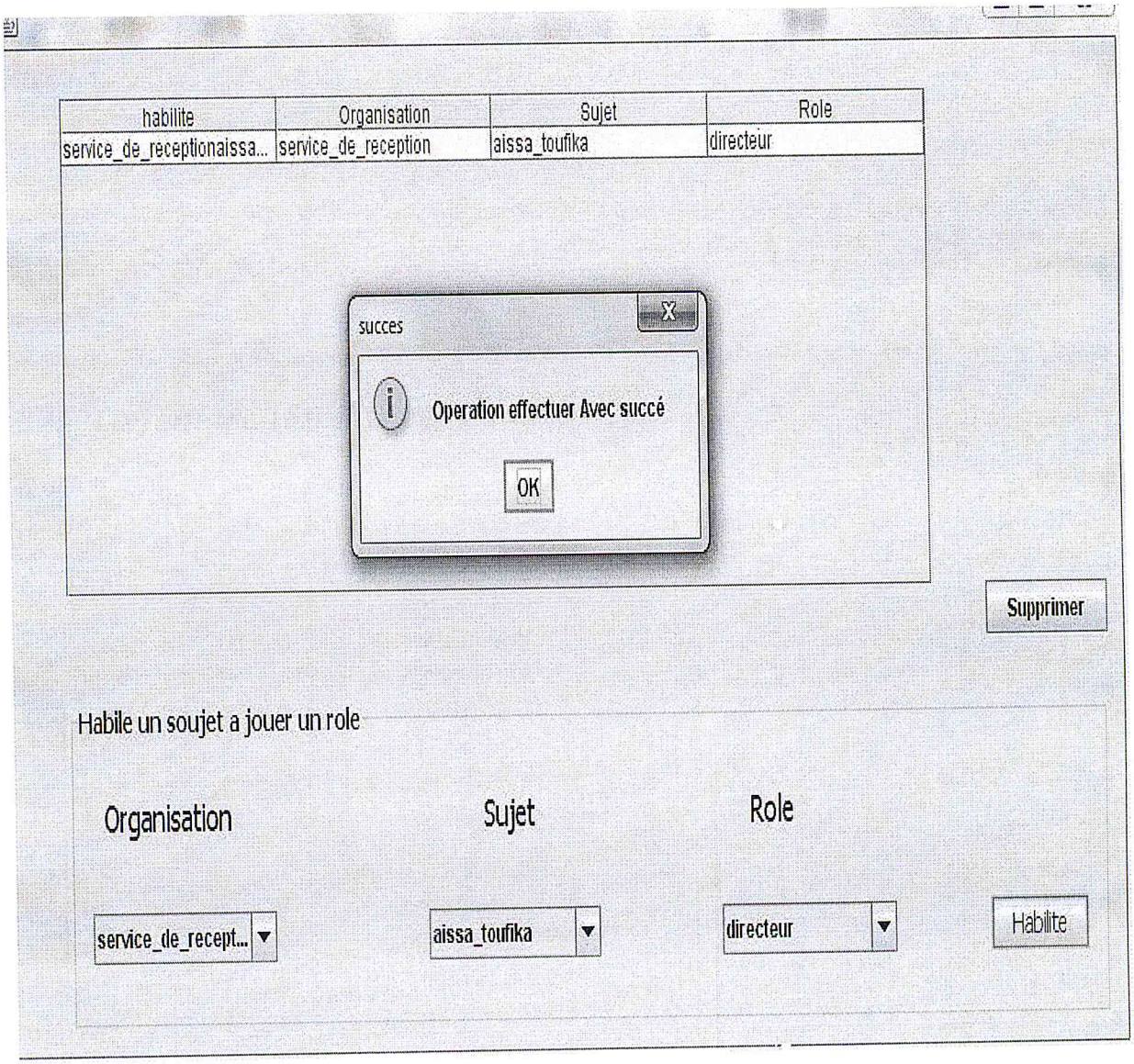


Figure 5.11 : La Fenêtre Habilite

8.3. La fenêtre Déléguer :

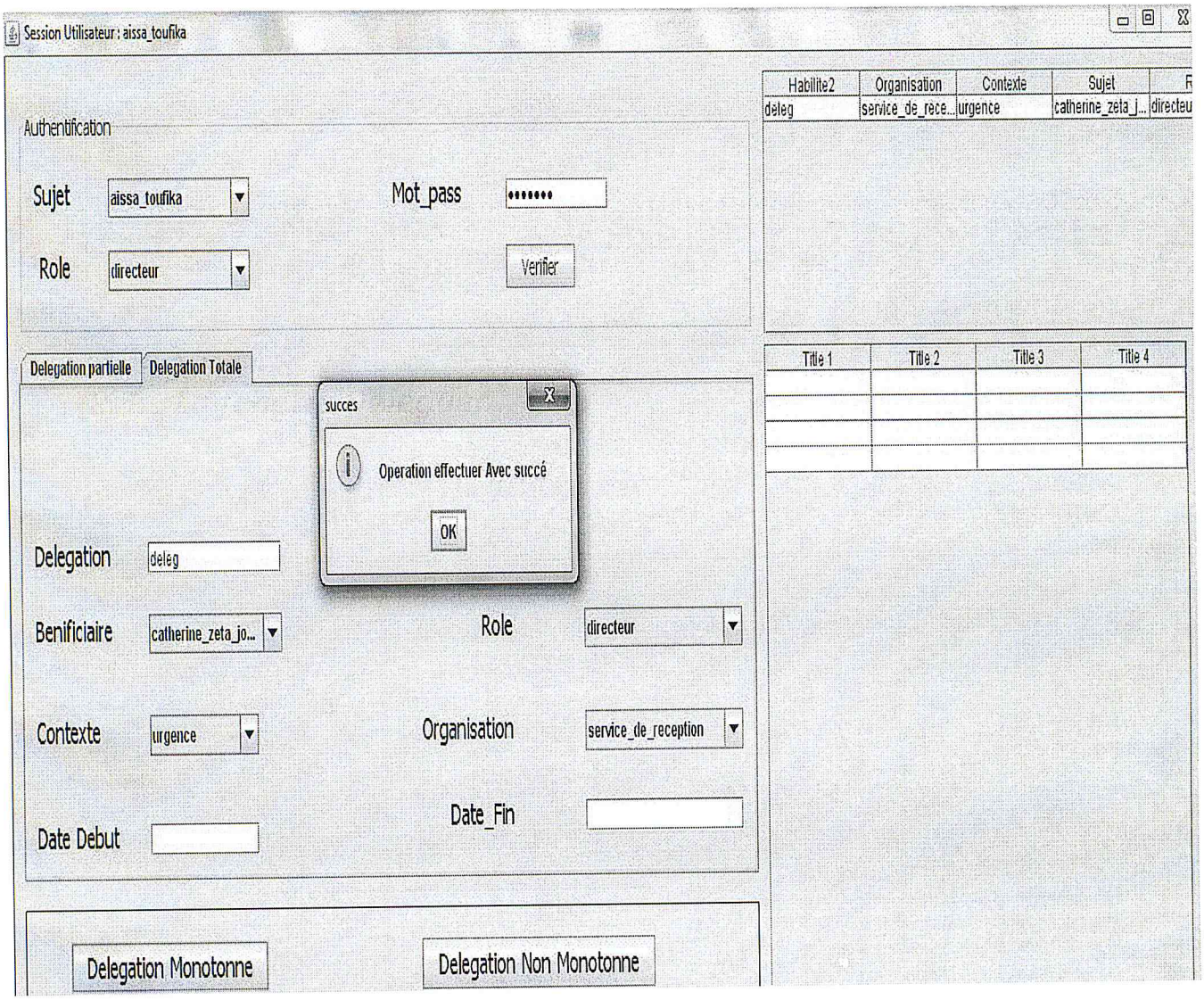


Figure 5.12 : La Fenêtre de délégation

8.4. La fenêtre Révoquer :

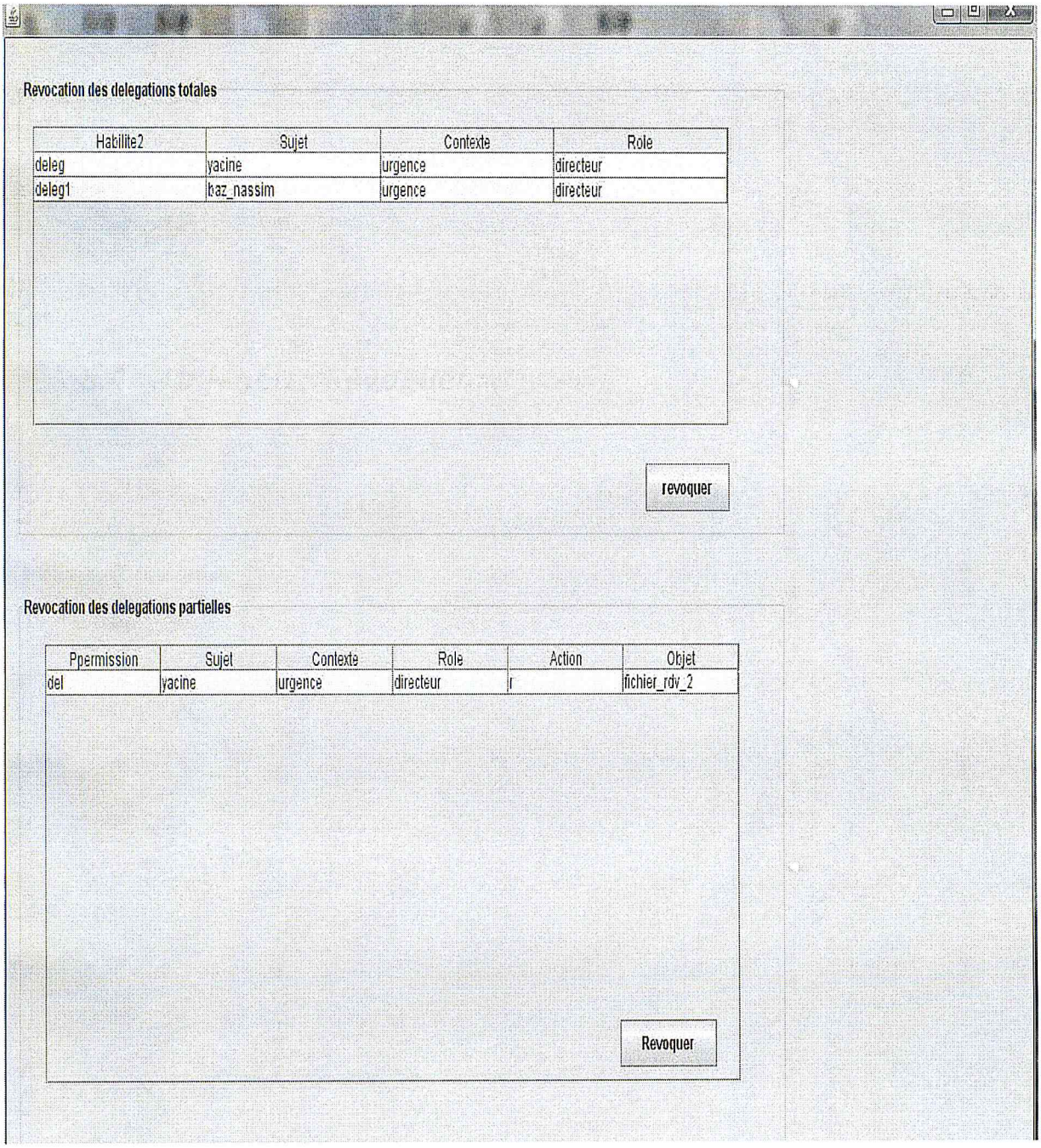


Figure 5.13 : La Fenêtre de révocation

8.5. La fenêtre Vérifier :

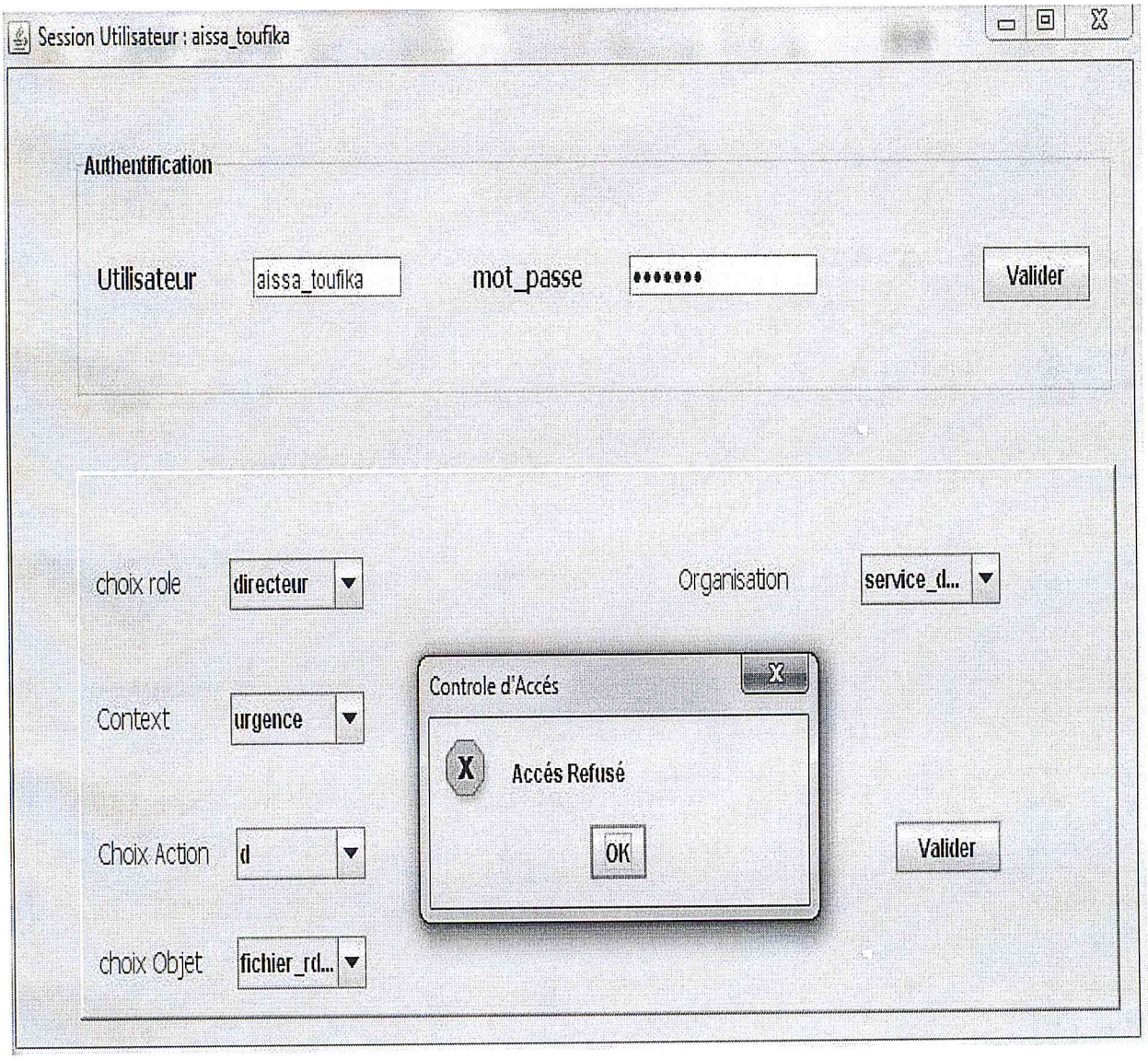


Figure 5.14 : La Fenêtre de Vérification

9. Conclusion :

Dans ce chapitre nous avons appliqué le modèle de contrôle d'accès Or-BAC sur un système d'information médical et nous l'avons présenté à travers quelques interfaces graphiques.

Or-BAC regroupe deux parties :

- La première consiste en la création et la manipulation des différentes entités qui composent notre base de connaissances. Nous avons détaillé les différentes opérations d'ajout et de suppression ainsi que les relations d'habilitation (entre rôle et sujet), considération (entre activité et action) et utilisation (entre vue et objet) ainsi que les relations de délégation.

- La deuxième partie se focalise sur l'introduction des règles de sécurité et l'inférence des entités créées dans la première partie. Elle contient aussi un service d'interrogation qui facilite à l'utilisateur d'avoir une réponse rapide sur une requête de demande d'accès dans différents contextes.

Conclusion générale

Assurer la sécurité est une des plus importantes préoccupations des chercheurs.il est important de pouvoir contrôles 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 sécurités. Sa conception et son développement doivent être menés de manière à garantir sa fiabilité et sa sûreté. En effet, toute faille au sein de cette conception 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 **DL-OrBAC.**

Les trois principales phases qui ont constitué 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- OrBAC*.

- L'expression de la délégation des permissions avec le modèle Or-Bac qui nous a facilité la tâche avec l'utilisation des vues, les permissions contextuelles ...etc. , ce qui nous a permis de spécifier la délégation.

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

Le modèle Or-BAC est centré sur le concept d'organisation, et est structuré en deux niveaux : Concret et abstrait. A partir du triplet <Sujet, Action, Objet> du niveau concret et du triplet <Rôle, Activité, Vue> du niveau abstrait. En effet, toutes ces entités dépendent d'une organisation donnée

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.

La délégation permettra de donner à un utilisateur particulier un privilège, sans donner ce privilège à toutes les personnes ayant le même rôle que lui.

Nous avons choisi la logique de description comme formalisme de représentation.

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.

Notre travail consistait à développer un modèle de contrôle d'accès inspiré d'Or-BAC formalisé avec la logique de description et prenant en compte la notion de délégation.

Cette étude nous a permis de mieux comprendre l'utilité de la logique de description et son application dans le cadre pratique tel qu'un modèle de contrôle d'accès.

Plusieurs enrichissements peuvent être définit pour ce travail, nous proposons en perspectives :

- Prendre en compte d'autres contextes tels que le contexte temporel et spatial.
- Intégrer au modèle DL-OrBAC, en plus des permissions les obligations et les interdictions.

Bibliographie:

1. Héritage non monotone pour une logique de description – Application au contrôle d'accès–, BENKOUSSAS Chahinez,USTHB, 2011.

2. La délégation dans le Modèle Or-BAC, Bellague Youcef, mémoire pour l'obtention d'un diplôme d'ingénieur d'état en informatique. Université SAAD DAHLAB BLIDA. USDB 2009-2010.

3. F. Cuppens, N. Cuppens-Boulahia, et A. Miège Inheritance hierarchies in the Or-BAC Model and application in a network environment .Second Foundations of Computer Security Workshop (FCS'04). Turku, Finlande. July 2004.

4. Managing Delegation in Access Control, Meriam Ben Ghorbel-Talbi*a;b*, Frederic Cuppens*a*, Nora Cuppens-Boulahia*a*, Adel Bouhoula*b* *a*GET/ENST Bretagne,

5. C.Bettini, S .Jajodia, X.S. Wang ET D. Wijesekera. Obligation Monitoring in policy management. International Workshop, Policies for Distributed Systems and Networks (Policy-2002), Monterey CA, 5-7 juin 2002.

6. Michel Gagnon <Logique descriptive et OWL> Michel Gagnon.

7. Or-BAC Organisation based Acces Control. Frederic Cuppens et Alexender Miege. ENST Bretagne, Campus de Rennes.

8. La modélisation du modèle de sécurité Or-BAC en logique de description, Aissa toufik, Baz Nassim, mémoire pour l'obtention d'un diplôme d'ingénieur d'état en informatique. Université SAAD DAHLAB BLIDA. USDB 2007-2008.

9. Or-BAC : un modèle de contrôle d'accès basé sur l'organisation. Anas Abou El Kalam, Rania El Baida et philippe Balbiani. IRIT et LAAS-CNRS.

10. Administration d'une politique de contrôle d'accès. Celine Coma, ENST Bretagne, Campus de Rennes.

11. Writing Rules for semantic Web using SWRL and Jess, Martin O'connor, Holger Knublauch, Samson Tu, Mark Musen Stanford medical Informatics, Stanford University School of medecine Stanford.

12. W.Wilikens, S.Feriti, and M.Masera.A context related authorization acces control method base on RBAC: a case study from the healthcare domain. In SACMAT02, june 3-4 2002.

13. A. Abou Elkalem, Rania El Baida, P.Balbiani, S.Benfarhat, F.Cuppens, Y.deswarte, A. Miege, C. Saurel, and G.Trouessin. Organisation Based Acces Control. Pages 120.131,2003.

14. F.Cuppens and A.miege. modelling contexts in the Or-BAC Model. 19th Applied computer Security Associates Conference (ACSAC 2003). Las vegas, Nevada. USA. Decembre 8-12,2003.

15. Définition d'un environnement formelle d'expression de politiques de sécurité. Modele Or-BAC et extension. Alexender Miege, 2003

16. F. Cuppens, N. Cuppens-Boulahia, and M. Ben-Ghorbel.High Level Conflict Management Strategies in AdvancedAccess Control Models. *Electronic Notes in Theoretical. Computer Science (ENTCS)*, 186:3–26, July 2007.

Webographie:

17. http://www.techno-science.net/?onglet=glossaire&definition=5346, 2012

18. http://orbac.org/index.php?page=orbac&lang=fr,2012

19. http://www.philippefournierviger.com/description_logics/introduction_logiques_de_description.php,2012

20. http://www.inf.unibz.it/~franconi/dl/course/,2012