République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université SAAD DAHLAB-BLIDA
USDB

Faculté des Sciences
Département Informatique

Mémoire pour l'obtention du diplôme de master en informatique.

Option : Ingénierie du Logiciel

Thème :

Un modèle de contrôle d'accès multi-niveaux contextuel

Promotrice :
Dr BOUSTIA Narhimene

Réalisé par :
M^lle Ben M'hamed Amani
M^lle Bensettiti Imene

Organisme d'accueil : Université Saad Dahlab Blida

Soutenue le : / 07 / 2010, devant le jury composé de :

Président :

Examinateur :

Examinateur :

Promotion 2010-2011

Remerciement

Tout d'abord, on remercie Allah pour la patience, la volonté, la force et la santé qu'Il nous a donné afin de réaliser ce travail.

Nos chaleureux remerciements s'adressent à notre promotrice Melle Boustia, on voudrait lui exprimer toute notre reconnaissance pour son encadrement, ses conseils, l'aide et le temps qu'elle a bien voulu nous consacrer et sans qui ce mémoire n'aurait jamais vu le jour.

On remercie également les membres du jury pour avoir accepté d'examiner et d'évaluer ce modeste travail.

Nos remerciements vont droit à tous nos enseignants du département d'informatique qu'on a eu le plaisir de rencontrer durant nos études. Merci pour votre aide, conseils et encouragements.

Enfin, nous remercions toute personne qui, à des degrés divers, contribué sur le plan intellectuel, technique, moral à l'achèvement de ce travail.

Dédicace

Je dédie ce mémoire

À mes chers parents, pour leur soutien moral, leur disponibilité, leur compréhension et surtout leurs encouragements quotidiens : « Grand Merci ».

À mes chers sœurs et mon cher frère que Allah les protèges.

À mon binôme et ma meilleure amie « Imene».

À toutes ma famille, et mes amis.

J'espère qu'ils trouveront dans ce travail toute ma reconnaissance et tout mon amour.

Amani

Je dédie ce modeste travail :

A la meilleur des mamans

A mes très chers frères Tahar et Tewfik

A ma belle sœur Soumia

Au petit Mohamed Ali

A mon binôme et amie Amani

A tous mes amis qui se reconnaitront

A toute ma promotion de master

Imene

Table des matières

Chapitre 4 : **DL-CMLAC$_{\delta\varepsilon}$**

Chapitre 5 : **Implémentation**

Table des figures

Introduction générale

Les modèles de contrôle d'accès se classent en deux grandes catégories, les politiques de contrôle d'accès discrétionnaire (DAC pour Discetionary Access Control) et les politiques de contrôle d'accès obligatoire (MAC pour Mandatory Access Control).

Le contrôle d'accès est dit obligatoire lorsque l'accès aux objets est basé sur le niveau de sensibilité de l'information contenue dans les objets. L'autorisation d'accéder à un objet est accordée à un sujet si le niveau d'autorisation de ce sujet est en accord avec le niveau de sensibilité de l'information.

La politique multi-niveaux de Bell-LaPadula est une politique obligatoire, développée pour le DoD (Department of Defense) des Etats-Unis.

Le but de ce projet est de **construire un modèle de contrôle d'accès dynamique et contextuel inspiré des politiques multi-niveaux, nous permettant d'attribuer les autorisations selon le contexte actuel**.

Pour représenter un modèle de sécurité, il serait intéressant de développer une approche formelle permettant de définir les différents composants de ce modèle.

Nous proposons de formaliser le modèle de sécurité avec La Logique de Description défauts et exceptions développée par Coupey et Fouqueré afin d'exploiter l'aspect de la non monotonie dans la représentation du contexte.

Dans un premier temps, nous établiront un état de l'art sur les différents modèles de contrôle d'accès. On s'intéressera par la suite au modèle multi-niveaux, ainsi qu'aux Logiques de Description en général. Une étude du système JCLASSIC$_{\delta\varepsilon}$ sera faite et nous terminerons par la réalisation de l'outil.

Partie 1: Etat de l'art

Chapitre 1: Les Modèles de contrôle d'accès

Ce chapitre définit les principaux modèles de sécurité existant, leurs politiques, leurs avantages et leurs inconvénients.

Chapitre 2: Modèle multi-niveaux

Dans ce chapitre on s'intéresse à un cas particulier de politique de contrôle d'accès obligatoire à savoir les politiques multi-niveaux.

Chapitre 3: Les Logiques de Description

Ce chapitre présente les LDs classiques en donnant leurs origines, leurs langages (terminologique et assertionnel), leurs mécanismes d'inférence et quelques domaines où elles sont utilisées.

Partie 2: Etude, réalisation et implémentation

Chapitre 4: DL-CMLACδε

Dans ce chapitre on conceptualise le DL-CMLAC$_{\delta\varepsilon}$, on présente les différents axiomes qui permettent de construire une base de connaissances LD capturant ses caractéristiques, avec l'usage de défaut (δ) et exceptions (ε).

Chapitre 5: Implémentation

Ce chapitre a été consacré aux évaluations du modèle DL-CMLAC$_{\delta\varepsilon}$, on va faire une description de toutes les étapes de l'étude en les illustrant à l'aide des interfaces graphiques. Ainsi que la présentation des différents outils et environnements de la réalisation.

Chapitre 1

Les modèles de contrôle d'accès

Les politiques de sécurité, ou plus précisément leurs schémas d'autorisation, se classent en deux grandes catégories : les politiques discrétionnaires (ou DAC pour Discretionary Access Control) et les politiques obligatoires (ou MAC pour Mandatory Access Control).

Il existe également des variantes de ces politiques qui peuvent mieux s'adapter à des organisations particulières, comme les politiques basées sur la notion de rôles (ou RBAC pour Role-Based Access Control) ou encore sur la notion d'équipes (ou TMAC pour TeaM-based Access Control).

Idéalement, il faut construire une politique de sécurité de telle sorte qu'aucune séquence valide d'applications des règles (du schéma d'autorisation) ne puisse amener le système dans un état tel qu'un objectif de sécurité soit violé, en partant d'un état initial sûr (on parle de politique de sécurité cohérente). Ceci suppose l'utilisation d'une méthode formelle de construction des règles du schéma d'autorisation, à partir d'une spécification formelle des objectifs de sécurité.

2.2 Modèle de sécurité :

Les politiques d'autorisation les plus citées dans la littérature sont généralement associées à un modèle de sécurité. D'une manière générale, un modèle peut être défini comme un formalisme (souvent mathématique) qui offre une vue subjective mais pertinente de la réalité. On modélise pour mieux comprendre le système qu'on développe, c'est-à-dire pour visualiser ses propriétés, spécifier sa structure ou son comportement, documenter et guider sa construction, etc. A partir de là, un modèle de sécurité peut être défini comme un formalisme permettant de représenter, de façon claire et non-ambiguë, la politique de sécurité. Il aide à l'abstraire (afin de réduire sa complexité) et à faciliter sa compréhension, comme il peut servir à vérifier que cette politique est complète (tout est protégé) et cohérente, et que la mise en œuvre par le système de protection est conforme aux propriétés attendues du système **[Poinsot]**.

Les modèles de sécurité peuvent être classés en deux grandes familles :

- des modèles généraux, qui sont plutôt des méthodes de description formelle, pouvant s'appliquer à toute sorte de politiques. C'est par exemple le cas de modèles de machines à états, représentant le système comme un ensemble d'états et de transitions qui, à partir d'un état courant et une valeur d'entrée, détermine le nouvel état du système. Il y a également les modèles basés sur les matrices d'accès, manipulant les trois concepts fondamentaux que sont les sujets, les objets et les actions. Les éléments qui se situent au croisement de la ligne L et de la colonne C correspondent aux droits possédés par le sujet correspondant à L sur l'objet C.

- des modèles spécifiques, développés pour représenter une politique d'autorisation particulière. Citons à titre d'exemple les modèles fondés sur les ***treillis***, qui affectent à chaque utilisateur et à chaque objet un niveau de sécurité précis.

3. Différents contrôles d'accès :

3.1 Les contrôles d'accès discrétionnaires (DAC) :

Dans le cas d'une politique discrétionnaire, les droits d'accès à chaque information sont manipulés librement par le responsable de l'information (généralement le propriétaire), qui les affecte. Les droits peuvent être accordés par ce responsable à chaque utilisateur, à des groupes d'utilisateurs, ou bien aux deux. Ceci peut parfois amener le système dans un état d'insécurité (c'est-à-dire contraire aux objectifs de sécurité qui ont été choisis) **[Poinsot]**.

Prenons un exemple simple, reposant sur les mécanismes de protection d'Unix. Dans un tel système, les droits d'accès à un fichier sont définis et modifiables librement par l'utilisateur propriétaire du fichier (et donc par les processus qui s'exécutent pour son compte).

Supposons que le schéma d'autorisation se définisse (informellement) de la manière suivante : un utilisateur peut créer des fichiers dont il devient alors propriétaire ; le propriétaire d'un fichier peut décider quels utilisateurs sont autorisés à lire ses fichiers. D'autre part, supposons que la politique exige de respecter l'objectif suivant : les utilisateurs qui n'ont pas le droit de lire un fichier ne doivent pas pouvoir en connaître le contenu.

Une telle politique n'est pas réalisable par des mécanismes d'autorisation discrétionnaire, parce que :

- Si ***s1*** est un sujet s'exécutant pour le compte de l'utilisateur ***u1*** propriétaire du fichier ***f1***, il peut donner au sujet ***s2*** (s'exécutant pour le compte d'***u2***) le droit de lecture sur ***f1*** : ce que l'on note par *(s1, f1, propriétaire)* ⟶ *(s2, f1, lire).*

- ***s2*** peut créer un fichier ***f2*** (dans lequel il peut donc écrire) sur lequel il peut donner le droit de lecture à ***s3*** (s'exécutant pour le compte d'***u3***) : *(s2, f2, créer)* ⟶ *(s2, f2, écrire) et (s3, f2, lire).*

- ***s2*** peut alors recopier ***f1*** dans ***f2*** pour transmettre les informations de ***f1*** à ***s3*** à l'insu du propriétaire ***s1*** : *(s2, f1, lire) et (s2, f2, écrire) et (s3, f2, lire)* ⟶ *(s3, c(f1), lire) où (c(f1) désigne une copie de f1).*

Une politique discrétionnaire n'est donc applicable que dans la mesure où il est possible de faire totalement confiance aux utilisateurs et aux sujets qui s'exécutent pour leur compte. Une telle politique est par là même vulnérable aux abus de pouvoir provoqués par maladresse ou par malveillance. Ainsi, s'il est possible à un utilisateur d'accéder à certains objets ou d'en modifier les droits d'accès, il est possible qu'un cheval de Troie s'exécutant pour le compte de cet utilisateur (à son insu) en fasse de même. De plus, si un utilisateur a le droit de lire une information, il a (en général) le droit de la transmettre à n'importe qui.

3.2 Les contrôles d'accès obligatoires (MAC) :

Une politique de sécurité d'autorisation obligatoire (ou MAC de l'anglais " Mandatory Access Control ") impose des règles d'autorisation incontournables qui s'ajoutent aux règles discrétionnaires. Une politique obligatoire suppose que les utilisateurs et objets aient été étiquetés **[Poinsot]**.
Classiquement, les objets se voient attribuer une classification, tandis que les utilisateurs possèdent une habilitation. Les règles qui régissent les autorisations d'accès sont basées sur une comparaison de l'habilitation de l'utilisateur et de la classification de l'objet. Ces règles incontournables assurent que le système vérifie des propriétés générales de confidentialité ou d'intégrité par exemple. Ces règles sont souvent utilisées conjointement aux règles d'une politique discrétionnaire car leur pouvoir d'expression est en général relativement faible.

Par exemple, un utilisateur sera autorisé à manipuler une information dans le système si l'utilisateur en question possède le droit de lecture sur l'information (contrôle discrétionnaire) et s'il est habilité à manipuler cette information (contrôle obligatoire).

Dans notre travail, nous nous intéressons à un cas particulier de politique MAC, celle du DoD (Department of Defense des États-Unis) qui suit le modèle de sécurité dit de Bell-La Padula **[Bell et La Padula, 1976]**.

3.3 Le contrôle d'accès basé sur le rôle (RBAC) :

Une politique de contrôle d'accès à base de rôle (RBAC pour Role-Based Access Control) se base sur la description des fonctions qu'un utilisateur a le droit d'accomplir au sein d'une organisation pour établir les règles d'accès aux informations **[Ferraiolo et Kuhn, 1992]**.

Le contrôle d'accès basé sur le rôle (RBAC) est une technologie qui a pris beaucoup d'attention, en particulier, dans le domaine des applications commerciales grâce à son potentiel de réduire la complexité et le coût de la gestion de sécurité sur le réseau. Sous RBAC, la gestion de la sécurité est simplifiée par l'utilisation des rôles, par la hiérarchie et les contraintes pour organiser les privilèges d'accès.

Avec RBAC, le concept central des politiques d'accès est le rôle. Les permissions sont affectées à des rôles et les rôles sont affectés à des utilisateurs. Ainsi, en administrant RBAC, deux types différents d'associations doivent être contrôlés :

- associations entre les utilisateurs et les rôles,
- et associations entre les rôles et les permissions.

Si la position du travail dans une organisation est représentée par un rôle simple, alors, quand la position du travail d'un utilisateur change, il y a seulement deux associations d'utilisateur/rôle à changer : enlever l'association entre l'utilisateur et le rôle courant de l'utilisateur et ajouter une association entre l'utilisateur et le nouveau rôle de l'utilisateur.

Eléments du modèle :

- U : ensemble d'utilisateurs
- R : rôles (fonctionnel, organisationnel)
- P : permissions
- S : sessions (les utilisateurs doivent toujours commencer par activer une session), activées par un seul utilisateur, qui peut activer plusieurs rôles
- AU : affectation des utilisateurs aux rôles (relation plusieurs-à-plusieurs)
- AP : affection des permissions aux rôles (relation plusieurs-à-plusieurs)

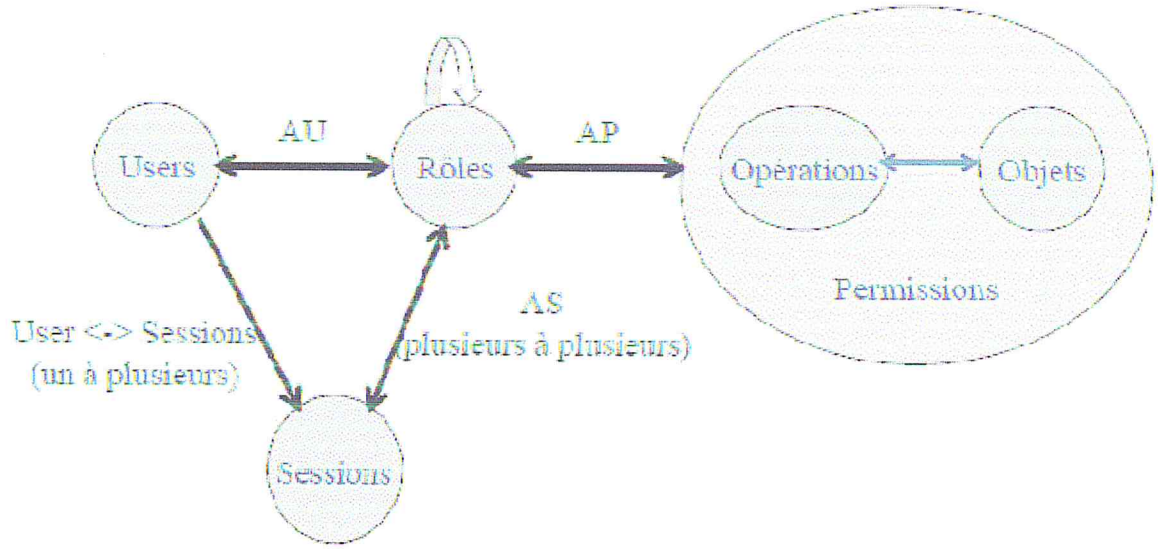


Figure1 : Modèle RBAC

Plusieurs modèles RBAC ont été proposés afin d'inclure de nouveaux concepts tels que la hiérarchie des rôles, les contraintes et les sessions.

3.4 Le contrôle d'accès basé sur l'organisation (ORBAC) :

Pour faire face aux limites de RBAC, un modèle ORBAC (Organisation Based Access Control) a été proposé. Ce modèle étend le modèle RBAC et attribut les permissions, les obligations, les recommandations et les interdictions pour réaliser des activités à un rôle dans une organisation sur un groupe d'objets dans un contexte donné **[Hattak, 2010]**. Dans le modèle ORBAC, huit entités ont été définies :

- Entité Sujet : qui fait référence aux utilisateurs par exemple, « Ali », « Leila », «Mohamed», etc.
- Entité Organisation : comme « département comptable ».
- Entité Rôle : est utilisée pour structurer le lien entre les sujets et les organisations. Dans le domaine médical, le rôle « cardiologue » est joué par des utilisateurs alors que le rôle« service des urgences » est joué par des organisations.

- Entité Objet : représente principalement les entités non actives comme les fichiers, les courriers électroniques, les formulaires imprimés, etc.
- Entité Action : englobe principalement les actions informatiques comme « lire », « écrire », « envoyer », etc.
- Entité Vue : un ensemble d'objets qui satisfont une propriété commune, par exemple: dans un hôpital, la vue « dossiers administratifs » peut désigner l'ensemble des dossiers administratifs du patient. La même vue peut être définie différemment suivant l'organisation. Exemple : la vue « dossier médical » peut être définie dans un hôpital comme un ensemble de documents Word, et comme un ensemble de documents Latex dans un autre hôpital.
- Entité Activité : correspond à des actions qui ont un objectif commun. Exemple : « consulter », « modifier », « transmettre », etc. L'activité « consulter » peut correspondre, dans l'organisation hôpital, à l'action « lire » un fichier, mais peut tout aussi bien correspondre à l'action « select » sur une base de données dans une autre organisation.
- Entité Contexte : permet d'exprimer des circonstances telles que « urgence », « médecin traitant », etc.

4. Conclusion :

Dans ce chapitre, nous avons présenté les modèles de contrôles d'accès et leurs politiques. Une politique de contrôle d'accès multi-niveaux est une politique de contrôle d'accès obligatoire se basant sur des niveaux de sécurité qui sont partiellement ordonnés. C'est l'objet du chapitre suivant.

Chapitre 2

Modèle multi-niveaux

Chapitre 2
Modèle multi-niveaux

1. Introduction :

Le contrôle d'accès est un mécanisme grâce auquel un système autorise ou interdit les ***actions*** demandées par des ***sujets*** sur des ***objets***. C'est un moyen mis en œuvre a priori, pour s'assurer de la légitimité des accès, selon une politique déterminée (i.e. la loi).

Ce chapitre présente les modèles obligatoires ou à niveaux (Mandatory Access Model : MAC) **[MAC, 1975]** où les objets sont classés par niveau (***exemple de hiérarchie de sensibilité*** **:** très secret, secret, confidentiel, non classé), par ensemble ou par catégorie ; les sujets sont définis par un niveau d'habilitation et des autorisations d'accès aux catégories. Ils accèdent en écriture aux objets de leur niveau d'habilitation et en lecture aux objets dont le niveau est inférieur ou égal à leur niveau d'habilitation. Un sujet est donc obligé d'abaisser son niveau d'habilitation pour une session donnée dans le but d'écrire sur des objets dont le niveau de secret est inférieur à son niveau d'habilitation. Le schéma forme ***un treillis*** partiellement ordonné appelé schéma ***multi-niveaux.***

Contrairement aux autres politiques d'accès, les systèmes à sécurité multi-niveaux imposent un contrôle de flux strictement monodirectionnel : un flux d'information n'est valide que s'il provient d'un niveau dominé vers un niveau dominant.

2. Sécurité informatique:

La sécurité informatique recouvre trois types de prévention [Information Technology Security Evaluation Criteria] : **[Common Criteria for Information Technology Security Evaluation, 2006] :**

- ***Confidentialité:*** prévention de la divulgation non autorisée de l'information.
- ***Intégrité :*** protection contre modification ou destruction non autorisées.
- ***Disponibilité :*** fournir les données aux utilisateurs autorisés quand ils en ont besoin.

2.1 Modèle de sécurité:

2.1.1 *Contrôle d'accès obligatoire/par mandats (MAC):*

- **Objectif :**
  - Contrôler la diffusion de l'information même si les applications ne sont pas de confiance
  - Assurer le cloisonnement des informations
- **Principes :**
  - Basé sur le contenu, i.e. la sensibilité de l'information
  - Un niveau de sensibilité (label) est attribué aux objets
  - Un utilisateur doit obtenir un « mandat » (une autorisation) pour accéder à une information sensible.
  - Seule l'organisation a le pouvoir de donner des droits
- **Dans la pratique :**
  - Modèle le plus répandu : multi-niveaux sécurisé dans le domaine de la défense **[Denning, 1982]**.
  - Modèle de Bell LaPadula. **[Bell and LaPadula, 1976]**.

2.1.2 Modèle de Bell & LaPadula:

Le modèle de Bell et LaPadula fut le premier modèle de type MAC proposé. L'objectif de ce modèle est de définir des contraintes pour contrôler l'exécution des applications de manière à empêcher les attaques par Chevaux de Troie contre la confidentialité. Le modèle de Bell et LaPadula considère des politiques de sécurité de type MAC particulière, dite politique de sécurité **multi-niveaux**.

Élaboré en 1975 pour le département de la défense américaine, ce modèle propose des règles pour prévenir la divulgation de l'information dans un système informatique. Les entités de base dans ce modèle sont :

- Entité active : **sujet** (processus, programmes en exécution)
- Entité passive protégée : **objets** (données, fichiers, programmes, sujets)

La classification des sujets et des objets accorde un niveau de sécurité à chaque sujet et chaque objet. Formellement, chaque objet est associé à un niveau de sécurité de la forme (niveau de classification, un ensemble de catégories). Chaque sujet est également associé à un niveau de sécurité maximum et un niveau de sécurité courant, qui peut être changé dynamiquement.

L'ensemble des niveaux de classification est ordonné par la relation « < ». Le niveau de sécurité A domine B si et seulement si le niveau de classification de A est supérieur au niveau de classification de B, et l'ensemble de catégories de B est inclus dans l'ensemble de catégories de A. Par exemple, si on a l'ensemble (top secret, secret, confidentiel, non classifié) et un ensemble de catégories (nucléaire, défense, etc.) où :

Non classifié < confidentiel < secret < top secret

Le niveau de sécurité (top secret, {nucléaire, défense}) domine le niveau de sécurité (secret, {nucléaire}) car top secret > secret et l'ensemble de catégories {nucléaire} est inclus dans l'ensemble {nucléaire, défense}.

Le modèle est une machine à état **(B, M, F, H)** où :

B : représente l'accès courant d'un sujet à un objet. C'est un triplet (sujet, objet, attribut d'accès). Les attributs d'accès permis dans le modèle sont :

E : non observation et non altération
R : observation et non altération
A : non observation et altération
W: observation et altération

M : la matrice(Mso), enregistre les modes d'accès permis pour un sujet sur un objet.

H : la hiérarchie, structures orientées imposées aux objets du système.

F : la fonction de niveau, les sujets et les objets reçoivent deux types de désignations formelles de sécurité : niveau de classification (non classifié, confidentiel, secret, top secret, etc.) et une catégorie (nucléaire, défense, etc.). Donc, la désignation de sécurité est une paire (classification, catégorie) cette paire est appelée niveau de sécurité.

Pour éviter la divulgation de l'information, deux caractéristiques doivent être maintenues :

- ***La propriété simple (no read up) :*** si on a un accès courant (sujet, objet, attribut d'accès) alors le niveau de sécurité du sujet domine celui de l'objet.
- ***Propriété étoile (no write down) :*** si un sujet a accès à un objet ***o1*** et peut altérer un objet ***o2*** alors le niveau de l'objet ***o1*** doit être dominé par le niveau de l'objet ***o2***.

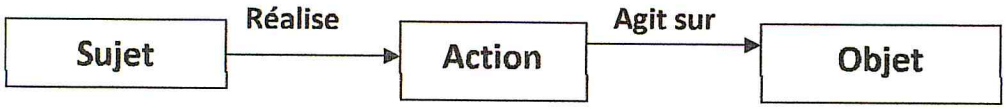
Exemple : admettons que le niveau de sécurité de l'objet **o1** domine celui de l'objet **o2**. Si **Sujet1** lit l'objet **o1** et **Sujet1** écrit les informations de l'objet **o1** dans l'objet **o2**, Alors il y aura une fuite de données.

3.3 Concepts de bases du modèle multi-niveaux :

- **Sujets :** c'est des entités actives du SI
  - Utilisateurs, processus travaillant pour le compte d'utilisateurs.

- **Objets :** c'est des entités passives du SI
  - Contiennent les informations à protéger (fichiers, relations dans une BD relationnelle, ...).
- **Actions (ou Opérations)** permettent aux sujets de manipuler les objets.
  - Lecture d'un fichier, requête dans une BD.

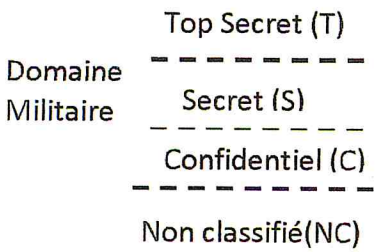
Les sujets ont des *permissions* de réaliser des actions sur des Objets



3.3.1 Structures de contrôle d'accès multi-niveaux :

- Pour certaines politiques de sécurité, il est nécessaire de considérer différents « niveaux » de sécurité.
- ***Niveaux de sécurité :***
  - Niveau : ensemble des niveaux, les objets et les utilisateurs sont classifiés en fonction de leur sensibilité.

  - Exemple :



Niveau : est complètement ordonné, T>S>C>NC

2002], alors que dans de nombreuses organisations (hôpital, entreprise,...) il existe un réel besoin de ne donner des droits que dans des circonstances précises.

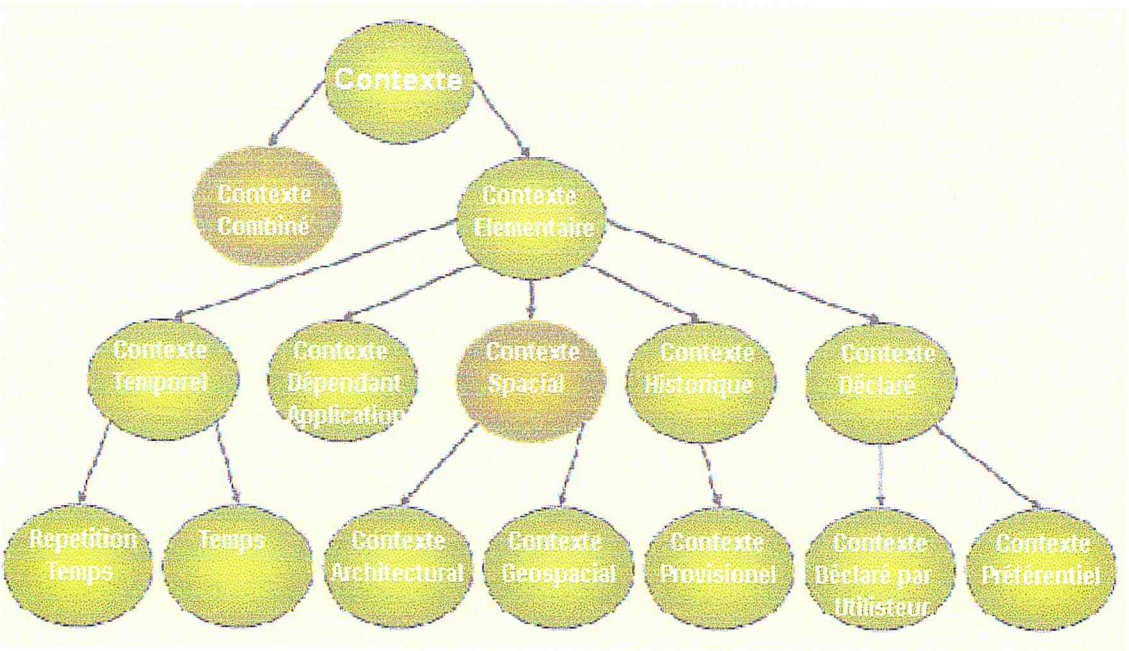


Figure 3: Hiérarchie de contextes dans multi-niveaux

Pour le modèle multi-niveaux, les différents contextes sont regroupés par type (comme sur le schéma ci-dessus):

- Contexte temporel: C'est le contexte qui régit la durée de validité des privilèges.
- Contexte spatial: Il peut être lié à l'appartenance à un réseau, ou à la position géographique, ou à toute autre situation spatiale.
- Contexte déclaré par l'utilisateur: Ce type de contexte est activé, par exemple, par le médecin en cas d'urgence, ou pour signaler que l'on effectue un audit. Dans ces cas exceptionnels, des permissions peuvent être données alors qu'elles seraient interdites dans un cas normal. L'utilisateur qui déclare le contexte est obligé en contrepartie de faire un compte-rendu des opérations effectuées et peut être des raisons qui l'ont motivé à déclarer ce contexte.
- Contexte pré requis: Son utilisation permet de contraindre les sujets concernés par les permissions ou les interdictions dépendant de ce contexte qui vient réduire ou étendre les droits d'accès hérités du rôle associé.
- Contexte provisionnel: Ce contexte permet de donner des privilèges en fonction de l'historique. Par exemple, le contexte "accès limités à 2 fois" regarde si le document a été accédé au moins 2 fois.

A noter que dans une modélisation multi-niveaux, on définit toujours un contexte par défaut.

Chapitre 3

Logique De Description

Chapitre 3
Les Logiques de Description

1. Introduction :

Un système à base de connaissances est un programme capable de raisonner sur un domaine d'application pour résoudre un certain problème, en s'aidant de 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 des systèmes à base de connaissances, 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 appelés **Logiques de Descriptions**, ou encore logiques terminologiques.

Dans le formalisme des Logiques de Descriptions, un concept permet de représenter un ensemble d'individus, tandis qu'un rôle représente une relation binaire entre individus. Un concept correspond à une entité générique d'un domaine d'application et un individu à une entité particulière, instance d'un concept. Concepts, rôles et individus obéissent aux principes suivants :

Un concept et un rôle possèdent une description structurée, élaborée à partir d'un certain nombre de constructeurs. Une sémantique est associée à chaque description de concept et de rôle par l'intermédiaire d'une interprétation. Les manipulations opérées sur les concepts et les rôles sont réalisées en accord avec cette sémantique.

Les connaissances sont prises en compte selon plusieurs niveaux : la représentation et la manipulation des concepts et des rôles relèvent du niveau terminologique ; la description et la manipulation des individus relèvent du niveau factuel ou niveau des assertions. Le niveau terminologique est aussi qualifié de Tbox et le niveau factuel d'Abox.

La relation de subsomption permet d'organiser concepts et rôles par niveau de généralité : intuitivement, un concept C subsume un concept D si C est plus général que D au sens où l'ensemble d'individus représenté par C contient l'ensemble d'individus représenté par D. Une base de connaissances se compose alors d'une hiérarchie de concepts et d'une (éventuelle) hiérarchie de rôles.

Les opérations qui sont à la base du raisonnement terminologique sont la classification et l'instanciation. La classification s'applique aux concepts, le cas échéant aux rôles, et permet de déterminer la position d'un concept et d'un rôle dans leurs hiérarchies respectives ; La construction et l'évolution de ces hiérarchies est ainsi assistée par le processus de classification. L'instanciation permet de retrouver les concepts dont un elle est individu.

Cette introduction est une brève synthèse sur les Logiques de Descriptions qui ont pour but de mettre en valeur les idées générales sur lesquelles reposent ces logiques et de montrer la richesse et la rigueur du formalisme associé, ainsi que les possibilités de représentation qui sont offertes.

2. Origine des Logiques de Description :

Les Logiques de Description sont une famille de langages de la représentation de la connaissance qui peut être utilisée pour représenter la connaissance d'un domaine d'application par un moyen clair, formel et structuré.
Ce sont des formalismes logiques de représentation qui se distinguent des Réseaux Sémantiques et représentations à base de Frames par leur sémantique formelle basée sur la logique.
Le nom « Logiques de Description » est motivé par le fait que, d'une part, les notions importantes du domaine sont décrites par des expressions construites par des concepts atomiques (prédicats unaires) et des rôles atomiques (prédicats binaires) utilisant des constructeurs de rôle et de concept fourni par une Logique de Description particulière. Et d'autre part, les Logiques de Description diffèrent de leurs prédécesseurs, tels que les Réseaux Sémantiques et les Frames, étant donné qu'elles sont équipées d'une logique formelle basée sur des sémantiques.
On distingue trois générations de systèmes qui sont :

- Systèmes de Logique de Pré-Description.
- Systèmes de Logiques de Description.
- Systèmes de Logiques de Description actuels.

2.1 Systèmes de Logique de Pré-Description :

L'ascendant des systèmes de Logiques de Description est le KL-One. Les Logiques de Description sont des formalismes de représentation des connaissances basés essentiellement sur le langage KL-One **[Brachman & al, 1991].** Le langage KL-One est considéré comme racine de la famille entière des langages.
Les réseaux sémantiques qui sont à l'origine du langage KL-One, ont été introduits dans les années 60 comme une représentation des concepts fondamentaux des mots anglais, et sont devenus un type populaire de structures pour représenter une large variété de concepts des applications en Intelligence Artificielle.

Le langage de représentation KL-ONE est un système de représentation de connaissances dont les travaux ont débuté en 1978 sous la direction de **Ronald Brachman [Brachman & Schmolze, 1985].** Il a été complété par un mécanisme de classification et une sémantique formelle en 1983 **[Schmolze & Lipkis, 1983].**

Il introduit la notion de langage hybride. Un langage hybride comporte deux composantes : un langage terminologique qui consiste en une description des connaissances comme les classes d'objets (les concepts) et les relations entre classes (les rôles), et un langage assertionnel qui est une description des connaissances spécifiques (les individus appartenant à ces concepts).

- *Les concepts primitifs*: expriment des conditions nécessaires, mais pas suffisantes pour l'appartenance d'un individu au concept. Par exemple, si un individu est une personne alors il possède une date de naissance, mais l'inverse n'est pas vrai : un animal possède une date de naissance mais n'est pas une personne.
- *les concepts définis*: expriment des conditions d'appartenance nécessaires et suffisantes. Par exemple, si un individu est un homme, c'est-à-dire, c'est une personne dont le sexe est masculin, l'inverse est vrai : chaque personne dont le sexe est masculin est un homme.

KL-ONE propose comme outil de raisonnement la classification de concepts : il s'agit d'un algorithme appelé «classifier», permettant de placer de nouveaux concepts dans la hiérarchie existante.

2.2 Systèmes de Logiques de Description :

Les dernières Pré-Logiques de Description proviennent directement de KL-One qui lui même est un résultat direct d'analyses formelles. Les systèmes de Logiques de Description qui suivront comme future génération proviendront plus de recherches théoriques sur les logiques de terminologie que de conséquences d'examen de KL-One et d'autres derniers systèmes.

Les résultats ultérieurs sur les compromis entre l'expressivité du langage de la Logique de Description et la complexité de raisonner, et plus généralement, l'identification de la source de complexité dans les systèmes de Logiques de Description, ont montré le besoin d'une sélection des concepts du langage et que les services de raisonnement fournis par le système sont profondément influencés par les concepts fournis par l'utilisateur.

On peut distinguer trois approches pour l'implémentation des services de raisonnement :

- La première peut être considérée comme 'limitée et complète' ou encore comme des systèmes qui sont étudiés par restriction de l'ensemble des concepts de telle sorte que la subsomption puisse être calculée efficacement, possible en temps polynomial. Le système *Classic* **[Brachman *et al.*, 1991]** est un exemple de ce genre d'approche.

- La seconde approche désignée comme expressive et incomplète, puisque l'idée est de fournir un langage expressif et un raisonnement efficace. L'inconvénient est que l'algorithme de raisonnement s'avère être incomplet dans ces systèmes. Un exemple de ce genre de système est le système *Loom* **[MacGregor, 1991].**

- Dans la troisième approche nous avons les systèmes caractérisés comme étant expressifs et complets. Ils ne sont pas efficaces comme ceux des approches précédentes.

2.3 Systèmes de Logiques de Description actuels :

Dans la génération actuelle des Systèmes de Représentation de la Connaissance basés sur les LDs, le besoin d'algorithmes complets des langages expressifs est devenu pressant.
L'expressivité du langage des Logiques de Description est nécessaire pour raisonner sur les modèles de données. Les données semi structurées ont contribué à l'identification de la plupart des extensions importantes pour des applications pratiques.

L'étude d'algorithmes complets pour les Logiques de Description expressives a mené à:
- des extensions signifiantes de techniques basées sur les tableaux et,
- l'introduction de plusieurs techniques d'optimisation partiellement empreintes de théorèmes démontrés et partiellement développées pour les Logiques de Description.

Le premier exemple de système développé est le **[Fact++]** **[Tsarkov *et al.,2006*]**. Cette recherche a été également influencée par les relations récemment découvertes entre les Logiques de Description et les autres logiques.

3. Logiques de Description :

Ce que nous entendons par Logique Descriptive est en fait une famille de formalismes pour représenter une base de connaissances d'un domaine d'application. Plus spécifiquement, ces logiques permettent de représenter des concepts (aussi appelés *classes*) d'un domaine et les relations (aussi appelées *rôles*) qui peuvent être établies entre les instances de ces classes. Par exemple, on pourrait les utiliser pour représenter les concepts *humains*, *femme* et *mère*, et spécifier que toute mère est une femme qui est le parent d'au moins un humain. Comme toute logique, des mécanismes d'inférence y sont associés, permettant ainsi de déduire de nouveaux faits à partir d'une base de connaissances.
Supposons par exemple qu'une base de connaissances définit le concept *mère* comme sous-classe du concept *humain*. Si on apprend qu'une entité est une mère, on peut automatiquement déduire qu'elle est aussi un humain. De même, avec la définition de mère ci-dessus, si un fait établit qu'une femme a deux enfants, on peut déduire qu'elle est une mère.

3.1 Eléments basiques de la Logique de Description :

- Les (noms de) **Concept** sont équivalents aux prédicats unaires
  - En général, les concepts sont équivalents aux formules avec une variable libre
- Les (noms de) **Rôle** sont équivalents aux prédicats binaires
  - En général, les rôles sont équivalents aux formules avec deux variables libres
- Les (noms d') **Individus** sont équivalents aux constantes
- Les **Constructeurs** sont restreints pour que :
  - Le langage reste décidable et, si possible, avec une complexité faible.
  - L'utilisation des variables n'est pas explicite

- Forme de restriction sur co-domaine comme ∃r.C et ∀r.C
  - Les caractéristiques telles que calcul peuvent être exprimées succinctement.

3.2 Langages de Description :

Les descriptions élémentaires sont les concepts atomiques et les rôles atomiques à partir desquels des descriptions complexes peuvent être générées via les constructeurs de concepts et les constructeurs de rôles. Dans ce qui suit, on utilisera les lettres A et B pour designer des concepts atomiques, la lettre R pour un rôle atomique et les lettres C et D pour les descriptions de concepts. Les langages de description sont distingués par rapport aux constructeurs qu'ils fournissent.
Dans ce qui suit, nous introduisons d'abord un langage minimal appelé *AL*, qui est enrichi progressivement de nouveaux constructeurs. Le langage *AL* s'appuie sur les langages *FL* et *FL-* présentés ci-dessous, qui sont les langages pour lesquels ont été établis les premiers résultats théoriques sur les LDs.

3.2.1 Les logiques *FL* et *FL-* :

Les logiques *FL* et *FL-* sont les premières Logiques de Descriptions, à travers lesquelles, des résultats théoriques sur la complexité de la subsomption ont été établis **[Brachman & Levesque, 1985] [Brachman & Levesque, 1987] [Nebel, 1988].** *FL-* comporte trois connecteurs : *and*, *all* et *some.* Elle a donné naissance à la logique *FL* en y ajoutant le constructeur *restrict* dont la syntaxe est r|C, qui introduit une contrainte sur le co-domaine d'un rôle. La grammaire sur laquelle repose *FL* est la suivante :

C ⟶ P	Concept primitif
T	Le concept le plus général
⊥	Le concept le plus spécifique
C⊓D	Conjonction de deux concepts
∀ r.C	Quantification universelle
∃ r	Quantification existentielle
R ⟶ r	Rôle
∃ r | C	Restriction sur le rôle

Figure4 : Grammaire de la logique FL

3.2.2 Le Langage de Description basique *AL* :

La logique *AL* introduite par **Schmidt-Schaub** et **Smolka** en 1991 **[Schmidt-Schaub & Smolka, 1991]** a étendu la logique *FL–* en y ajoutant la négation des concepts primitifs.
Cette logique peut-être considérée comme la logique de base des autres logiques de descriptions qui existent.

Les descriptions de concepts dans le langage AL sont générées selon la règle syntaxique suivante:

C ⟶ P	Concept primitif
T	Le concept le plus général
⊥	Le concept le plus spécifique
¬P	La négation d'un concept primitif
C⊓D	Conjonction de deux concepts
∀ r.C	Quantification universelle
∃ r	Quantification existentielle

Figure5 : Grammaire de la logique AL

C et **D** sont des concepts, **P** est un concept primitif et **r** est un rôle primitif.
• Le concept **T** désigne le concept le plus général tandis que le concept ⊥ désigne le concept le plus spécifique (l'ensemble vide).
• Le connecteur ¬ est utilisé pour évoquer la négation d'un concept primitif. Il représente les individus qui n'appartiennent pas au concept primitif **P**.
• Le connecteur **⊓** permet de faire la conjonction de deux concepts composés, il permet de représenter l'ensemble des individus, membres à la fois du concept **C** et du concept **D**.
• Le quantificateur universel **∀ r.C** représente l'ensemble des individus du domaine d'un rôle r qui sont en relation, par le biais de **r**, avec les individus du concept **C**.
• Le quantificateur existentiel **∃ r** désigne l'ensemble des individus qui sont membres du domaine d'un rôle r.

L'exemple suivant : nous donne une idée sur ce qui peut être exprimé dans le langage AL :
Soient **Person** et **Female** deux concepts atomiques. Donc **Person ⊓ Female** et **Person ⊓ ¬Female** sont des concepts décrivant intuitivement les personnes qui sont femelle et les personnes qui ne sont pas femelle. En outre, étant donné un rôle atomique **hasChild**, on peut construire les concepts **Person ⊓ ∃hasChild.T** et **Person ⊓ ∀ hasChild.Female** dénotant les personnes qui ont un enfant et les personnes dont tous les enfants sont une femelle. Enfin, en utilisant le concept bottom, on peut également décrire les personnes qui n'ont pas d'enfants par **Person ⊓ ∃hasChild: ⊥.**

3.2.3 La famille de la logique *AL :*

Les différentes LDs sont formées en ajoutant à la logique ***AL*** **= {T, ⊥, ⊓, ∀, ∃, ¬}** différents connecteurs **[Nebel, 1990]**.
La particularité de cette famille (*AL*-langages) est que chaque logique qui en fait partie comporte la syntaxe de base *AL* plus un certain nombre de connecteurs. Chaque connecteur additionnel est représenté par une lettre qui est ajoutée au nom du langage.
Nous présentons la famille de la logique *AL* dans **la figure 6**, cette famille n'est pas exhaustive.

	or ($\mathcal{U}$)	c-somme (ε)	not ($\mathcal{C}$)	atleast ($\mathcal{N}$)	atmost ($\mathcal{N}$)	and-role ($\mathcal{R}$)	atleast qualifié($\mathcal{Q}$)
$\mathcal{ALU}$	X						
$\mathcal{AL}\varepsilon$		X					
$\mathcal{ALC}$			X				
$\mathcal{ALN}$				X	X		
$\mathcal{ALR}$						X	
$\mathcal{ALQ}$							X
$\mathcal{ALNR}$				X	X	X	
$\mathcal{ALU}\varepsilon$	X	X					
$\mathcal{ALCNR}$			X	X	X	X	

Figure6 : Famille de la logique AL

- Le connecteur or : ou disjonction de concepts (représenté par la lettre *U*), noté (or C D) ou(C U D), permet de représenter les individus qui appartiennent soit au concept C soit au concept D.
- Le connecteur c-some : ou quantification existentielle de rôle (représenté par la lettre ε), noté (c-some r C) ou ∃r.C, spécifie l'existence d'au moins un couple d'individus (x,y) en relation par l'intermédiaire du rôle r, où C est le concept de y .
- Le connecteur not : ou négation sans restriction (représenté par la lettre *C*), noté (not C) ou ¬ C, permet la négation de concept primitif ou défini.
- Les connecteurs atleast et atmost : ou restriction de nombre de rôle (représenté par la lettre *N*), notés (atleast n r) ou ≥ n r et (atmost n r) ou ≤ n r, ces connecteurs fixent la cardinalité (nombre de valeurs élémentaires) minimale et maximale du rôle au quel ils sont associés.
- Le connecteur and-role : ou conjonction de rôle (représenté par la lettre *R*), noté (r1 ⊓ r2), permet la conjonction de deux rôles.
- Le connecteur atleast qualifié : ou restriction de cardinalité qualifiée (représenté par la lettre *Q*), noté (≥ n r.C), permet de spécifier le nombre minimale ou maximale d'entité d'une classe spécifique par l'intermédiaire du rôle r.

Exemple :

-La logique *ALU* comprend les connecteurs de base de la logique *AL*, ainsi que le connecteur d'union représenté par la lettre *U*.

-La logique *ALNR* est composée des connecteurs de base d'*AL*, augmentée des connecteurs atleast et atmost (représenté par *N*) et and-rôle (représenté par *R*).

3.3 Structure générale de système LD :

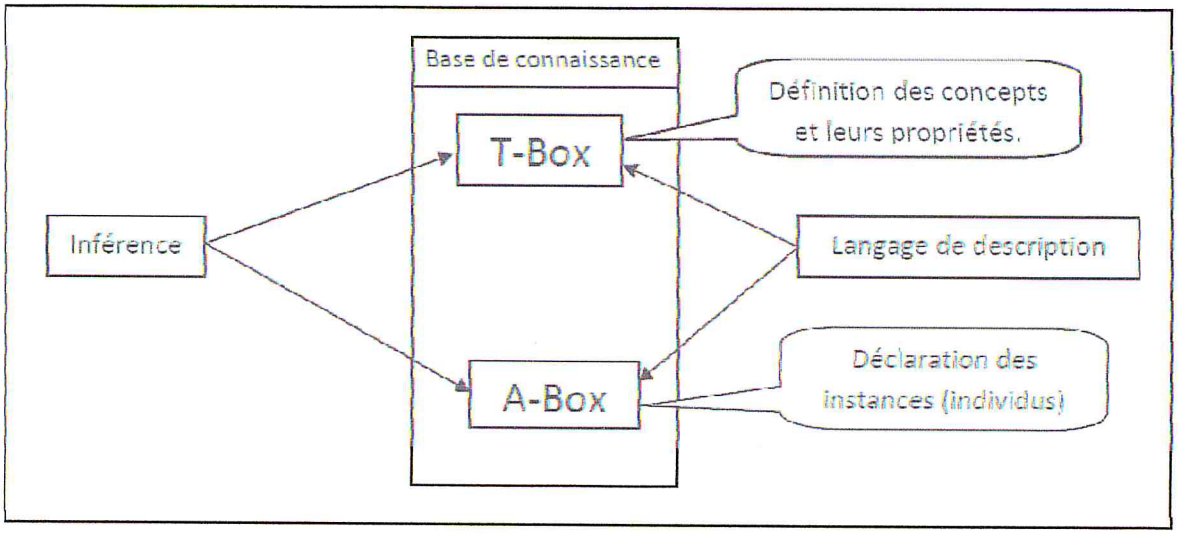


Figure7 : Structure générale des logiques de descriptions

3.4 La ABox et la TBox :

Dans une base de connaissances en Logique Descriptive, on distingue deux composantes : la*TBox* et la *ABox*.

3.4.1 Le langage terminologique T-Box :

Contient tous les axiomes définissant les concepts du domaine.
Dans la TBox, on est généralement intéressé 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 TBox 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 TBox 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 *homme* est une sous-classe de *humain*, on peut déduire qu'un *végétarien* est un *humain*.

3.4.2 Le langage assertionnel A-Box :

Contient des assertions sur des individus, en spécifiant leur classe et leurs attributs.
Le type d'inférence réalisé avec la TBox diffère de celui réalisé avec la ABox.
Dans la ABox, 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 ABox, 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 TBox il sera spécifié qu'une mère doit avoir au moins un enfant, la ABox devra contenir au moins un autre individu, et une relation entre celui-ci et Marie indiquant qu'il est un des ses enfants.

Les inférences avec la ABox 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 TBox déclare qu'un célibataire est une personne non mariée. La ABox sera inconsistant si elle contient un célibataire qui est marié avec une autre personne.

3.5 Les mécanismes d'inférence dans les Logiques de Descriptions :

Des mécanismes d'inférence sont associés aux LDs, ils permettent la déduction des connaissances qui ne sont pas représentées explicitement dans la base.
Les deux inférences principales 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 fondées sur la relation de subsomption.

3.5.1 Notion de subsomption :

La relation de subsomption permet d'organiser les concepts et les rôles par niveaux de généralité. Intuitivement, un concept *A* subsume un concept *B* si *A* (le subsumant) est plus général que *B* (le subsume). Cette réalité peut être exprimée de trois différentes façons:

a) Définition ensembliste en extension : un concept *A* subsume un concept *B* si et seulement si l'ensemble des représentants dénoté par *A* contient nécessairement l'ensemble des représentants dénoté par *B* **[Levesque et Brachman, 1987].** Dans cette définition, chaque concept est représenté par un ensemble et chacune des relations de subsomption est représentée par une inclusion ensembliste.
Par exemple, le concept qui dénote les mammifères subsume celui dénotant les chats.

b) Définition ensembliste en intention : Un concept *A* subsume un concept *B* si toute entité représentée par *B* l'est aussi par *A*. Ainsi, chaque entité dont la description est définie par *B* possède les caractéristiques qui sont spécifiées par A **[Finin, 1986].**
Par exemple, le concept dénotant les chats possède toutes les caractéristiques de celui qui dénote les mammifères en plus de celles qui sont propres aux chats. Un concept se compose donc d'une description propre à lui-même et d'une description « partagée » ou «commune » avec ses subsumants.

c) Définition logique : Un concept *A* subsume un concept *B* si un élément qui est décrit par *B* implique logiquement que cet élément est décrit par *A*. Ainsi, l'implication *(Chat(x) => Mammifère(x))* est vraie pour tout *x* **[McGregor, 1988].**

En utilisant la notion d'interprétation, on peut définir la relation de subsomption de la manière suivante : un concept *B* est subsumé par un concept *A* (respectivement *A* subsume *B),* ce qui se note $B \sqsubseteq A$, si et seulement si $B^I \sqsubseteq A^I$ pour toute interprétation I.

La subsomption est à la base de la classification dans la Logique de Description. C'est une relation d'ordre partiel qui permet d'organiser les concepts en une hiérarchie. Chaque concept possède une description qui permet de le comparer aux autres et ainsi de déterminer les relations de subsomption qu'il entretient avec les autres concepts.

3.5.2 Les inférences terminologiques :

Nous présentons, maintenant, les principales opérations qui se trouvent au niveau de la T-Box, appelées les opérations terminologiques.
Nous commençons par présenter le mécanisme principal (la classification de concepts), puis nous donnerons la définition des autres opérations. La liste de ces opérations n'est pas exhaustive mais présente l'ensemble des opérations les plus couramment rencontrées dans les LDs.

- **Classification de concepts** : est l'opération qui permet de placer un concept 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 la taxonomie.
Le processus de classification se décompose en trois étapes principales :
(i) La recherche des concepts les plus spécifiques (aussi appelés subsumeurs) (SPS),
(ii) La recherche des concepts les plus généraux (aussi appelés subsumés) (SPG),
(iii) La mise en place des nouveaux liens entre le nouveau concept et les concepts présents dans une taxonomie.

Autres opérations terminologiques :

- **Détection d'incohérence** : Dans la T-Box, il est intéressant de savoir si tous les concepts définis sont cohérents, c'est-à-dire, pour chaque concept, s'il peut exister au moins un individu membre de ce concept.

- **Héritage :** un concept hérite des propriétés des concepts qui le subsument.

- **Complétion :** la complétion permet de retrouver des propriétés hérités du concept et aussi des propriétés déduites logiquement.

3.5.3 Les inférences assertionnelles (factuelles) :

Les opérations assertionnelles varient d'une logique de description à une autre. La plupart des A-Box contiennent l'opération de reconnaissance d'instances.

-**Reconnaissance d'instances :** la reconnaissance d'instances consiste à trouver, pour un individu donné, les concepts les plus spécifiques dont il est instance.

Autres opérations assertionnelles :

-**Détection d'inconsistance :** 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.

-**Propagation** : une assertion sur un individu I, peut avoir des conséquences logiques sur des individus en relation avec cet individu. Les LDs qui bénéficient de l'opération de propagation propagent les nouvelles informations déduites sur les individus concernés.

-**Requêtes** : l'utilisateur d'une base de connaissances peut faire des requêtes en posant par exemple des questions de type : 'Quels individus sont en relation par r avec l'individu I ?'

- **Fermeture de rôle :** certaines LDs possèdent une opération de fermeture de rôle.
Si un rôle r est fermé pour un individu I, alors il n'existe pas d'autres individus en relation, par r, avec I que ceux exprimés explicitement.

- **Chaînage avant :** étant donné des règles ayant des concepts en prémisse et en conclusion.
Si on a trouvé un individu qui satisfait un concept se trouvant en prémisse d'une règle, on déduit qu'il satisfait aussi le concept se trouvant en conclusion de cette règle. Cette déduction est effectuée à l'aide d'un mécanisme fonctionnant en chaînage avant.

4. Présentation de la logique de description C-CLASSIC :

La Logique de Description C-CLASSIC est décrite par **W.W.Cohen** et **H.Hirsh** dans **[Cohen & Hirsh, 1994]**. C'est une variante des Logiques de Descriptions classiques, elle comprend un algorithme de calcul de subsomption polynomial, correct et complet.

C ⟶	T	Concept le plus général
	$\bot$	Concept le plus spécifique
	P	Concept primitif
	ONE-OF $\{I_1, I_2, I_3 \ldots I_n\}$	Concept en extension
	MIN u	Ensemble de réels > u
	MAX u	Ensemble de réels ≤ u
	C ⊓ D	Conjonction de concepts
	∀R : C	Précise le co-domaine de R
	R FILLS $\{I_1, I_2, I_3 \ldots I_n\}$	Restriction de valeurs pour R
	R AT-LEAST n	Restriction de nombre pour R (minimum)
	R AT-MOST n	Restriction de nombre pour R (maximum)

Figure8 : Syntaxe de la logique C-CLASSIC

- Le concept T dénote le concept le plus général,
- le concept ⊥ dénote le concept le plus spécifique,
- Le connecteur Π dénote la conjonction de deux concepts,
- Le connecteur ONE-OF permet de définir un domaine de valeurs en donnant la liste exhaustive des instances,
- Le connecteur MIN (respectivement MAX) permet de définir un domaine de valeurs strictement supérieures (respectivement inférieures ou égales) à u,
- La quantification universelle (∀ R: C) précise le co-domaine du rôle R, qui est le concept avec lequel le rôle établit une relation,
- Le connecteur FILLS associe une ou plusieurs valeurs à un rôle,
- AT-LEAST (respectivement AT-MOST) sont des restrictions de cardinalité qui fixent le nombre minimal (respectivement maximal) de valeurs que peut prendre le rôle.

Les expressions formées à partir de cette grammaire, sont appelées des "descriptions de concept" ou, plus simplement, des "termes".

Exemple de description en C-CLASSIC :

Définition du concept C : personnes majeures pratiquant entre deux et quatre sports pris dans l'ensemble {Tennis, Foot, Hand-ball, Volley-ball}.
C ≡ personne Π ∀ age: MIN 18 Π sports AT-LEAST 2 Π sports AT- MOST 4 Π ∀ sports: ONE OF {Tennis, Foot, Hand-ball, Volley-ball}.

4.1 Problèmes liés à la présence d'individus dans la T-Box :

La présence d'individus dans un Langage de Description pose deux sortes de problèmes résolus dans **[Borgida & Patel-Schneider, 1994]**.
Le premier problème concerne le changement de la hiérarchie des concepts. En effet, la définition des individus peut évoluer dans le temps, de nouveaux faits concernant un individu peuvent apparaître à tout moment et remettre en cause la hiérarchie obtenue après classification.

Ex : Supposons l'état de la base de connaissances comme suit :
C ≡ ∀ sports : SPORT-DANGEREUX
D ≡ ∀ sports : ONE-OF {Escalade}
Catch :: C (Catch est une instance du concept C)

L'ensemble des instances du Concept C contient l'individu Catch et l'ensemble des instances du concept D contient l'individu Escalade.
D'un point de vue sémantique extensionnel, il n'y a aucune relation de subsomption entre ces deux concepts (C ne subsume pas D et D ne subsume pas C).
Or, l'ajout de l'information "Escalade :: SPORT-DANGEREUX" , i.e., "Escalade" est instance du concept "SPORT-DANGEREUX", impliquera une nouvelle relation de subsomption entre les deux concepts C et D.
En effet, l'ensemble des instances de C devient {Catch, Escalade}, et celui du concept D ne change pas {Escalade}, de ce fait la classification de C et D a changé (C subsume D: l'ensemble des instances de C contient l'ensemble des instances du concept D).

Le second problème, concerne la complexité du calcul de subsomption. Vu que les individus sont considérés comme des éléments du domaine, cela peut induire à un raisonnement de type disjonctif ce qui augmente la complexité des algorithmes de calcul de subsomption, et cela malgré que les concepteurs de la logique C-CLASSIC ont élagués le connecteur "or" afin d'éviter cela.

Exemple : Supposons la description suivante :
C ≡ ∀ sports : SPORT-DANGEREUX ⊓ ∀ sports : ONE-OF {Tennis, Escalade}

L'ensemble des instances du concept C est pris dans l'ensemble {Tennis, Escalade}, cette description permet de déduire que : Tennis ou Escalade sont des SPORT-DANGEREUX.
Pour résoudre ces deux problèmes **A. Borgida** et **P. F. Schneider**, ont développé une solution qui consiste à distinguer les individus utilisés dans la T-Box (appelés "Individus classic"), et ceux utilisés dans la A-Box. Les individus classic ne possèdent aucune propriété et aucun fait de la A-Box ne les concerne.

4.2 Sémantique de C-CLASSIC :

L'étude théorique du mécanisme de raisonnement (complétude, correction et complexité des algorithmes d'inférence) nécessite de formaliser les descriptions de concepts et cela en dotant le système d'une sémantique formelle.

4.2.1 Sémantique extensionnelle :

Dans une sémantique extensionnelle, le concept est décrit en donnant l'ensemble de ses instances.
Ex : chiffres paires= {0, 2, 4, 6, 8}

- **Subsomption extensionnelle**

Un concept C subsume d'une manière extensionnelle un concept D si et seulement si l'ensemble des instances de C contient l'ensemble des instances de D (notée : D ⊆ e C).

Ex : L'ensemble des instances du concept *enfant* est inclus dans l'ensemble des instances du concept *humain* (enfant ⊆ e humain).

4.2.2 Sémantique descriptive :

La sémantique descriptive, consiste à écrire les descriptions de concepts à partir d'un système équationnel EQ **[Ventos, 1997]** contenant des axiomes, qui souligne les principales propriétés des connecteurs de la logique C-CLASSIC.

- **Subsomption descriptive**

Un concept C subsume descriptivement un concept D (notée : D ⊆ d C), si et seulement si, C ⊓ D $=_{EQ}$ D.

Exemple : Soient C et D deux termes de C-CLASSIC,
D ≡ ∀ sports: ONE-OF {Tennis, Escalade}
C ≡ sports AT-MOST 4
C subsume t-il descriptivement D (i.e. D ⊆ d C) ?

Pour répondre à cette question le système EQ est appliqué :
C ⊓ D ≡ ∀ sports: ONE-OF {Tennis, Escalade} ⊓ sports AT-MOST 4
≡ ∀ sports: ONE-OF {Tennis, Escalade} ⊓ sports AT-MOST 2 ⊓ sports AT-MOST 4
(**axiome 18:** ∀ R ONE-OF { I1...In} = ∀ R ONE-OF {I1...In} ⊓ R AT-MOST n)
≡ ∀ sports: ONE-OF {Tennis, Escalade} ⊓ sports AT-MOST 2
(**axiome 15:** R AT-MOST m ⊓ R AT-MOST n = R AT-MOST mini(n, m))
≡ ∀ sports: ONE-OF {Tennis, Escalade} (axiome 18)
≡ D
D'où, C subsume D puisque C ⊓ D= $_{EQ}$D.

4.2.3 Sémantique intensionnelle (structurelle) :

Dans une sémantique intensionnelle, le concept est décrit par l'ensemble de ses propriétés.
Exemple : chiffre-pair ≡ chiffre ⊓ divisible-par-2

• **Subsomption intensionnelle**
Un concept C subsume structurellement un concept D, si et seulement si, l'ensemble de ses propriétés est inclus dans l'ensemble des propriétés de D.

Exemple : L'ensemble des propriétés du concept enfant contient l'ensemble des propriétés du concept être-humain, plus les propriétés qui lui sont spécifiques (ex : âge <12).

5. Applications développées avec des systèmes de Logiques de description :

On distingue plusieurs domaines d'applications, certains incluant Software Engineering, Configuration, Médecine, bibliothèques numériques et les Systèmes d'Informations basés sur Web.
Comme il existe plusieurs autres domaines d'applications où les LDs jouent un rôle important, comme les domaines qui comprennent le Traitement du Langage Naturel et la Gestion des Bases de Données.
Certaines applications, dont la création a duré des années, sont arrivées au niveau du prototype seulement, mais plusieurs d'entre-elles ont la totalité des systèmes industriels. Plusieurs projets sur le traitement du langage naturel basés sur l'utilisation des LDs ont été entrepris, certains ont atteint le niveau d'applications industrielles.

Le langage naturel :

L'utilisation des LDs dans le traitement du langage naturel pour la représentation de la connaissance peut être utilisée pour communiquer le sens des phrases.
Cette connaissance est typiquement concernée par le sens des mots (le dictionnaire), et par le contexte c'est à dire une représentation de la situation et du domaine de dialogue.

L'expressivité du langage naturel mène aussi à des investigations concernant les extensions des LDs, comme par exemple le raisonnent par défaut. Le travail sur le langage naturel a nécessité la construction d'ontologies.

6. Conclusion :

Dans ce chapitre, nous avons présenté les Logiques de Descriptions, qui constituent un formalisme de représentation de connaissances caractérisé par les points suivants :

- Un langage permet de construire des descriptions conceptuelles qui sont génériques (concepts primitifs et définis) ou individuelles (instances).
- Une sémantique est associée à chaque construction syntaxique par l'intermédiaire d'une interprétation.
- Une relation de subsomption permet d'organiser les descriptions par niveau de généralité, et de procéder à des inférences ; cette relation est à la base des processus de classification et d'instanciation.

Chapitre 4

DL-CMLACDE

Chapitre 4
DL-CMLAC$_{\delta\varepsilon}$

1. Introduction :

La politique de sécurité décrit les conditions qui doivent êtres vérifiés pour qu'une action soit permise ou interdite. Le contrôles d'accès est définit généralement de la manière suivante : «dans un système, et dans un contexte donnée, un utilisateur établit une requête, c'est au système d'autoriser la requête d'accès ou bien la refuser ».

Dans notre travail, nous nous inspirons de la politique de contrôle d'accès multi-niveaux utilisé par les militaires pour assurer la confidentialité des informations afin de construire un modèle de contrôle d'accès contextuel et dynamique.

Dans le modèle de contrôle d'accès multi-niveaux, un sujet S peut accéder à un objet O si et seulement si son niveau d'habilitation est supérieur ou égale au niveau de classification de l'objet.

Pour donner une représentation formelle DL-CMLAC$_{\delta\varepsilon}$(modèle de contrôle d'accès multi-niveaux contextuel avec la Logique de Description défaut et exception), nous utilisons JCLASSIC$_{\delta\varepsilon}$ que nous avons développé dans ce but. C'est un système basé sur la Logique de Description « default (δ) et exception (ε) », inspiré par $AL_{\delta\varepsilon}$ Logique de Description **[Coupey et Fouqueré, 1997]**. Ce type de raisonnement non monotone dans la Logique de Description n'est pas suffisamment développé.

En fait, il n'y a aucun système développé pour ce type de raisonnement dans le web. Les systèmes les plus connus comme Classic, Racer, Fact++, Kaon, et Pellet sont basé sur le raisonnement monotone.

Le connecteur δ représente la notion de défaut, le connecteur ε est utilisé pour représenté la propriété qui n'est pas présente dans la description du concept ou de l'individu, mais qui doit l'être.

Le principal intérêt de cette approche est l'introduction de la notion de connaissance par défaut, cette dernière peut être une partie de la définition d'un concept.

La principale opération est le calcul de la relation de subsumption de JCLASSIC$_{\delta\varepsilon}$, qui est utilisé pour classifier et déduire la connaissance. La relation d'héritage est utilisée pour calculer les propriétés héritées.

L'utilisation de notre système dans le contrôle d'accès dynamique est intéressante pour plusieurs raisons : il permet de représenter des contextes composés, ajouter de nouveaux contextes et de déduire de nouvelle autorisations dépendantes du contexte.

Dans notre travail, nous commençons par présenter notre raisonneur non monotone JCLASSIC$_{\delta\varepsilon}$ et décrire les relations de subsomption et d'héritage qui permettent de raisonner sur une base de connaissance avec défauts et exceptions. Après nous présentons notre modèle de contrôle d'accès multi-niveaux contextuel. Nous décrivons son langage, nous montrons comment les autorisations peuvent être décrites.

2. JCLASSIC$_{\delta\varepsilon}$:

Dans cette partie, nous allons présenter notre raisonneur JCLASSIC$_{\delta\varepsilon}$ qui est une combinaison des AL$_{\delta\varepsilon}$ et du formalisme C-Classic **[Brachman and al. 1991; Brachman et al. 1989].**

Les Logiques de Descriptions sont des formalismes de représentation de connaissances issus du système KL-ONE **[Brachman 1978]**. Plusieurs systèmes ont été construit en se basant sur les LDs tel que Classic **[Brachman and al. 1991; Brachman et al. 1989]**, Fact++ **[Fact++]**, Racer **[Racer].** Les LDs ont été utilisé de manière effective dans des applications.

Les systèmes cités précédemment sont tous basés sur un raisonnement monotone. JCLASSIC$_{\delta\varepsilon}$ est un système de Logique de Description non monotone qui nous permet de traiter les défauts et les exceptions.

Les connecteurs de JCLASSIC$_{\delta\varepsilon}$ sont l'union des connecteurs de C-Classic **[Brachman and al. 1991]** et des connecteurs δ et ε.
Par exemple : "Student ⊓ δ (Publication At -least2) ⊓ ∀ Age:Max25" décrit les étudiants qui ont généralement au moins deux publications et qui ont moins de 25 ans.

Le connecteur δ représente la notion défaut. Par exemple Oiseau ⊑ Animal ⊓ δVole, «δVole » dans la définition du concept Oiseau veut dire que généralement les oiseaux volent.

Le connecteur ε est utilisé pour représenter une propriété qui n'est pas présente dans la description d'un concept ou d'un individu mais qui doit l'être. L'individu Pingouin en JCLASSIC$_{\delta\varepsilon}$ a la description suivante : Pingouin ⊑ Oiseau ⊓ Vole$^{\varepsilon}$. La propriété Vole$^{\varepsilon}$ exprime le fait que voler doit être dans la définition de pingouin puisque c'est un oiseau. La présence de Vole dans la définition de Pingouin rend possible la classification de celui-ci sous le concept Oiseau.

2.1 Syntaxe de JCLASSIC$_{\delta\varepsilon}$:

La syntaxe JCLASSIC$_{\delta\varepsilon}$ de est définie à partir d' :
•Un ensemble ***R*** de rôles primitifs.
•Un ensemble ***P*** de concepts primitifs.

• Des constantes ***T*** et ⊥ qui correspondent respectivement au concept le plus général et au concept le plus spécifique.
•Un ensemble ***I*** d'individus.
•De la règle syntaxique suivante : (***C*** et ***D*** sont des concepts, ***P*** est un concept primitif, ***R*** rôle primitif, ***u*** un nombre réel, ***n*** est un nombre entier et ***Ii*** des individus).

C⟶ T	Concept le plus général
⊥	Concept le plus spécifique
P	Concept primitif
ONE-OF $\{I_1, I_2, I_3...I_N\}$	Concept en extension
MIN $_u$	Ensemble de réel >u
MAX $_u$	Ensemble de réel ≤u
C ⊓ D	Conjonction de deux concepts
∀R:C	Precise de co-domaine de R
R FILLS $\{I_1, I_2, I_3...I_N\}$	Restriction de valeurs pour R
R AT-LEAST n	Restriction de nombre pour R (minimum)
R AT-MOST n	Restriction de nombre pour R (maximum)
δC	Concept C par défaut
C^{ε}	Exception sur le concept

Figure9 :Syntaxe de la logique JCLASSIC$_{\delta\varepsilon}$

2.2 Sémantique de JCLASSIC$_{\delta\varepsilon}$:

Comme pour la Logique de Description C-CLASSIC, JCLASSIC$_{\delta\varepsilon}$ est aussi dotée d'une sémantique intentionnelle, afin de formaliser les différentes descriptions de concepts, et de pouvoir inférer sur ces descriptions.

Les descriptions de concepts sont décrites via un système équationnel appelé EQ, qui souligne de manière formelle les principales propriétés des connecteurs de JCLASSIC$_{\delta\varepsilon}$.

2.2.1 Système équationnel EQ:

∀ A, B, C ∈ JCLASSIC$_{\delta\varepsilon}$, Ii ∈ I:

- 01: (A ⊓ B) ⊓ C = A ⊓ (B ⊓ C)
- 02: A ⊓ B = B ⊓ A
- 03: A ⊓ A = A
- 04: T ⊓ A = A
- 05: ⊥⊓ A = ⊥
- 06: (∀R : A) ⊓ (∀ R : B) = ∀R : (A ⊓ B)
- 07: ∀R : T = T
- 08: ONE-OF E1 ⊓ ONE-OFE2 = ONE-OF(E1 \ E2)
- 09: MIN m ⊓ MIN n = MIN maxi (m, n)
- 10 : MAX m ⊓ MAX n = MAX mini (m,n)

- 11: R FILLS E1 ⊓ R FILLS E2 = R FILLS(E1UE2)
- 12: R FILLS ∅ = T
- 13: R AT-LEAST m ⊓ R AT-LEAST n = RAT-LEAST maxi(m, n)
- 14: R AT-LEAST 0 = T
- 15: R AT-MOST m ⊓ R AT-MOST n = R AT-MOST mini (n, m)
- 16: R AT-MOST 0 = ∀ R: ⊥
- 17: R FILLS $\{I_1, I_2, I_3...I_N\}$ = R FILLS $\{I_1, I_2, I_3...I_N\}$ ⊓ R AT-LEAST N
- 18: ∀R: ONE-OF $\{I_1, I_2, I_3...I_N\}$ = ∀ R: ONE-OF $\{I_1, I_2, I_3...I_N\}$ ⊓ R AT-MOST N
- 19: R AT-LEAST N ⊓ ∀R: ONE-OF $\{I_1, I_2, I_3...I_N\}$ = R AT-LEAST N ⊓ ∀R: ONE- OF $\{I_1, I_2, I_3...I_N\}$ ⊓ R FILLS $\{I_1, I_2, I_3 ...I_N\}$
- 20: R AT-MOST n ⊓ R FILLS $\{I_1, I_2, I_3...I_N\}$ = R AT-MOST n ⊓ R FILLS $\{I_1, I_2, I_3...I_N\}$ ⊓ ∀R:ONE-OF $\{I_1, I_2, I_3...I_N\}$
- 21: ONE-OF ∅
- 22: MIN m ⊓ MAX n (if n ≤ m).
- 23: R AT-LEAST m ⊓ R AT-MOST n (if n ≥ m).
- 24: R FILLS E1 ⊓ ∀R: ONE-OF E2 (if E2 ⋣ E1).
- 25: $\delta(A)^{\varepsilon} = A^{\varepsilon}$
- 26: δ (A ⊓ B) = (δ A)⊓(δ B)
- 27: A ⊓ δ A = A
- 28: A^{ε} ⊓ δ A = A^{ε}
- 29: δδ A = δA

Les axiomes « 1 ⟶24 » sont classique et concerne les propriétés des connecteurs de C-Classic, et les axiomes « 25 ⟶29 » correspondent aux propriétés des connecteurs $AL_{\delta\varepsilon}$ i.e propriétés des connecteurs δ et ε.

- De manière informelle, les axiomes expriment, que la conjonction de concept est :
  - 1 : associative
  - 2 : commutative
  - 3 : idempotence
  - 4 : le concept le plus général de l'hiérarchie Top (T) est l'élément neutre de la conjonction.
  - 5 : le concept le plus spécifique Bottom (⊥) est l'élément absorbant.
  - 6 : le connecteur ∀R: C est distributif sur les conjonctions du concept.
  - 7, 12,14 : représentent une restriction fausse sur les rôles.

- Les propriétés suivantes décrivent la relation de subsomption entre des concepts identique :

  - 08: ONE-OF E1 subsume ONE-OF E2 si E2 ⊑ E1.

Exemple: ONE-OF {Rouge, Bleu, Vert} subsume ONE-OF {Rouge, Vert}.

  - 09: MIN m subsume n if n ≥ m.
  - 10: MIN m subsume MAX n if n ≤m.
  - 11: R FILLS E1 subsume R FILLS E2 if E1 ⊑ E2.

Exemple: couleur FILLS {Rouge, Vert} subsume couleur FILLS {Blanc, Rouge, Vert}.

- 13: R AT-LEAST m subsume R AT-LEAST n if $n \geq m$.
- 15: R AT-MOST m subsume R AT-MOST n if $n \leq m$.

- Les propriétés suivantes décrivent la relation de subsomption entre différents connecteurs :

  - 17: R AT-LEAST n subsume R FILLS $\{I_1, I_2, I_3 \ldots I_N\}$.

  Ex: une personne qui pratique au moins le tennis et le handball pratique au moins deux sports.

  ≡ Sport AT-LEAST 2 subsume sport FILLS {tennis; handball}.

  - 18: R AT-MOST n subsume ∀R: ONE-OF $\{I_1, I_2, I_3 \ldots I_N\}$.

  Ex: un objet qui peut avoir comme couleur rouge ou verte, a au plus deux couleurs.

  ≡ Couleur AT-MOST 2 subsume ∀ couleur: ONE-OF {rouge, vert}.

- les propriétés suivantes définissent l'équivalence entre deux termes. L'équivalence est défini comme une double subsomption : soit A et B deux termes de JCLASSIC$_{\delta\varepsilon}$: A≡B si et seulement si A subsume B et B subsume A.

  - 19 et 20: R AT-LEAST n ⊓ ∀ R: ONE-OF $\{I_1, I_2, I_3 \ldots I_N\}$ est equivalent à R AT-MOST n ⊓ R FILLS $\{I_1, I_2, I_3 \ldots I_N\}$.

Exemple (19): si un objet a au moins deux couleurs (couleur AT-LEAST 2) et si ses couleurs sont le rouge et le vert (∀couleur: ONE-OF {rouge, vert}) alors c'est un objet qui a comme couleur au moins le rouge et le vert (couleur FILLS {rouge, vert}).

Exemple (20):si un objet a au plus deux couleurs (couleur AT-MOST 2) et ses couleurs sont au moins le rouge et vert (couleur FILLS {rouge, vert}) alors c'est un objet qui peut avoir le rouge et le vert comme couleur (∀couleur: ONE-OF {rouge, vert}).

- Il ya une équivalence entre quelque termes du JCLASSIC$_{\delta\varepsilon}$ et Bottom ⊥. Ces termes correspondent à l'inconsistance.
  - 21: ONE-OF ∅.
  - 22: MIN m ⊓ MAX n if $n \leq m$.
  - 23: R AT-LEAST m ⊓ R AT-MOST n if $n \geq m$.
  - 24: R FILLS E1 ⊓ ∀ R: ONE-OF E2 if $E2 \not\supseteq E1$.

Exemple: un objet qui a au moins deux couleurs le rouge et le vert (couleur FILLS {rouge, vert}) et ses couleurs peuvent être seulement le noir et le blanc (∀couleur: ONE-OF {noir, blanc}) n'existe pas.

- 16:il y a une équivalence entre AT-MOST 0 et ∀ : ⊥. Ces deux termes représentent l'absence de fillers du rôle R.

Exemple: Personne ⊓ Enfant AT-MOST 0 et Personne ⊓ ∀ : ⊥ signifient les personnes qui n'ont pas d'enfant.

- Les propriétés suivantes correspondent aux propriétés des connecteurs δ et ε.
  - 25: l'exception d'un concept défaut est égale à l'exception d'un concept.
  - 26: le défaut d'une conjonction de concept est similaire à la conjonction de deux concept défaut.
  - 27 et 28: expriment le fait que A et A^{ε} sont subsumés par δA. Cela veut dire aussi que la priorité est donné au concept ou à l'exception de celui-ci.
  - 29:permet de supprimer les chaines défauts redondantes.

2.2.2 Forme normale :

Nous définissons une structure algébrique $CL_{\delta\varepsilon}$ qui est utilisé pour donné une sémantique intentionnel ou le concept est dénoté par une forme normale de ses ensembles de propriétés.

Pour définir la relation de subsomption entre deux concepts nous devons comparer leur description. Pour cela, les concepts sont caractérisés par une forme normale de leurs propriétés plutôt que leur ensemble d'instances.

La forme normale d'un concept défini avec la description T (note fn(T)) est un couple **<tσ, tδ>** où **tσ** contient les propriétés strictes de T et **tδ** ses propriétés défauts.

tσ et tδ sont des sextuplets de la forme **(dom, min, max, π, r, ε)** avec :

- **dom**: est un ensemble d'individus si la description contient une propriété ONE-OF, sinon le symbole UNIV.
- **min** (respectivement **max**): est un réel si la description contient une propriété MIN (respectivement MAX), sinon le symbole MIN-R (respectivement MAX-R).
- **π:** est un ensemble de concepts primitifs appartenant à la description T.
- **r:** est de la forme <R, fillers, least, Most, c> où :
  - R: est un nom de rôle.
  - fillers: est un ensemble d'individus si la description contient une propriété r FILLS, sinon ϕ.
  - least: (respectivement Most) est un entier si la description contient une propriété AT-LEAST (respectivement AT-MOST), sinon 0 (respectivement NOLIMIT).
  - c: est la forme normale de C si la description contient une propriété ∀R: C.
- **ε:** est un ensemble de sextuplets de la forme (dom, min, max, π, r, ε)

Ex: la forme normale du concept C ≡A Π δB est :

Fn(C)= (<UNIV, MIN-R, MAX-R, {A}, ϕ, ϕ >, <UNIV, MIN-R, MAX-R, {A, B}, ϕ, ϕ>).

$JCLASSIC_{\delta\varepsilon}$	$CL_{\delta\varepsilon}$
T	< (UNIV, MIN-R, MAX-R, ϕ, ϕ, ϕ) , (UNIV, MIN-R, MAX-R, ϕ, ϕ, ϕ) >
P	< (UNIV, MIN-R, MAX-R, {P}, ϕ, ϕ), (UNIV, MIN-R, MAX-R, {P}, ϕ, ϕ) >
ONE-OF E	< (E, MIN-R, MAX-R, ϕ, ϕ, ϕ), (E, MIN-R, MAX-R, ϕ, ϕ, ϕ)>
MIN u	(UNIV, u, MAX-R, ϕ, ϕ, ϕ), (UNIV, u, MAX-R, ϕ, ϕ, ϕ)>
MAX u	< (UNIV, MIN-R, u, ϕ, ϕ, ϕ), (UNIV, MIN-R,u, ϕ, ϕ, ϕ)>
∀ R : C (C ≢ T et C ≢ ⊥)	<(UNIV, MIN-R, MAX-R, ϕ,{<R,ϕ,0,|$c_{\Theta.dom}$|,c>},ϕ), (UNIV, MIN-R, MAX-R, ϕ, {<R,0, |$c_{\Theta.dom}$|, c>},ϕ)>
∀ R : C et C ≡ ⊥	<(UNIV, MIN-R, MAX-R, ϕ,{<r, ϕ,0, 0, bo >},ϕ), (UNIV, MIN-R, MAX-R, ϕ,{<r, ϕ,0, 0, bo >},ϕ)>
∀ R : C et C ≡ T	<(UNIV, MIN-R, MAX-R, ϕ, ϕ , ϕ), (UNIV, MIN-R, MAX-R, ϕ,ϕ,ϕ)>
R FILLS E (E ≠ ϕ)	< (UNIV, MIN-R, MAX-R, ϕ, {<r, E, |E|, NOLIMIT,t>},ϕ), (UNIV, MIN-R, MAX-R, ϕ, {<r, E, |E|, NOLIMIT,t>}, ϕ)>
R FILLS ϕ	<(UNIV, MIN-R, MAX-R, ϕ, ϕ,ϕ), (UNIV, MIN-R, MAX-R, ϕ, ϕ, ϕ)>
R AT-LEAST n (n >0)	(UNIV, MIN-R, MAX-R, ϕ,{<r, ϕ, n,NOLIMIT,t>}, ϕ),(UNIV, MIN-R, MAX-R, ϕ,{<r, ϕ, n,NOLIMIT, t >}, ϕ)>
R AT-LEAST 0	<(UNIV, MIN-R, MAX-R, ϕ, ϕ,ϕ), (UNIV, MIN-R, MAX-R, ϕ, ϕ, ϕ)>
R AT-MOST n (n >0)	< (UNIV, MIN-R, MAX-R, ϕ,{<r, ϕ,0, n, t >} , ϕ),(UNIV, MIN-R, MAX-R, ϕ,{<r, ϕ, 0,n, t >} , ϕ)>
R AT-MOST 0	< (UNIV, MIN-R, MAX-R, ϕ,{<r, ϕ,0, 0, bo >} , ϕ), (UNIV, MIN-R, MAX-R, ϕ,{<r, ϕ,0, 0, bo >} , ϕ)>
C ⊓ D	c⊗d
δC	<(UNIV, MIN-R, MAX-R, ϕ, ϕ , ϕ),c_δ>
C^ε	<(UNIV, MIN-R, MAX-R, ϕ, ϕ, c_δ),($c_{\delta.dom}$, $c_{\delta.max}$,$c_{\delta.min}$,$c_{\delta\pi}$, $c_{\delta r}$, $c_{\delta\varepsilon}$ U c_δ)>
⊥	Bo

2.3 Inférence en $JCLASSIC_{\delta\varepsilon}$:

Le système LD fournit plusieurs services de raisonnement. Dans notre travail, nous nous intéressons à la classification de concept (TBox) et d'instance (ABox). Ces deux services utilisent la relation de subsomption et d'héritage.

2.3.1 Subsomption :

$Sub_{\delta\varepsilon}$ est composé de deux étapes (voir Algorithme $Sub_{\delta\varepsilon}$). La première est la normalisation de la description. La seconde est une comparaison syntaxique entre les formes normales. C et D deux termes de $JCLASSIC_{\delta\varepsilon}$. Pour répondre à la question « C est il subsumé par D » nous appliquons la procédure suivante. Les formes normales de C et « C ⊓ D » sont calculées par la procédure de normalisation.

Il y a deux étapes dans la comparaison. On compare la partie stricte des deux concepts. S'il y a équivalence, alors on compare les parties défauts. Les détails des algorithmes de

normalisation et comparaison ne sont pas donnée ici (Ils sont donné en annexe). Si les deux formes normales sont égales, alors l'algorithme retourne « oui ». Sinon il retourne «non ».

Algorithme $Sub_{\delta\varepsilon}$

En entrée : C et D deux descriptions de concepts de $JCLASSIC_{\delta\varepsilon}$.
En sortie : Réponse Oui ou Non à la question «D subsume t-il C ?»

Début
{*Calcul de formes normales*}
Fn (C) ⟵ Normalisation (C)
Fn (C ⊓ D) ⟵ Normalisation (C ⊓ D)
{*Traitement BOTTOM*}
Si Fn (C) = bo[1] **alors** réponse ⟵ Oui
sinon
{*Traitement de l'héritage de l'exception*}
Si (($_{\delta}$ Fn (D) $_{\varepsilon}$ .taille)%2)
Si (($_{\delta}$ Fn (C) .taille) !%2) **alors** réponse ⟵ Non
Sinon
Si Fn (C ⊓ D) = bo **alors** réponse ⟵ Non
Sinon **{*comparaison des formes normales*}**
Comparer (Fn (C), σ Fn (C ⊓ D) , σ rep1)
si rep1 = ''Oui'' **alors**
Début
Comparer (fn (C), $_{\delta}$ fn (C ⊓ D), $_{\delta}$ rep2)
Réponse ⟵ rep2
Fin
Sinon Réponse ⟵ Non
Fin Si
FinSi
FinSi
FinSi
FinSi
Fin

[1] bo: est une constante qui représente la forme normale du concept le plus spécifique, BOTTOM (⊥).

2.3.2 Héritage :

Dans un cadre strict (dans le cadre d'une LD ne comportant pas de connecteur défaut, exception), le mécanisme d'héritage est similaire au mécanisme de subsomption : un concept hérite de toutes les propriétés des concepts qui le subsument. Or, dans le cadre du $JCLASSIC_{\delta\varepsilon}$, qui comporte des connaissances de type défaut et exception. Un concept ne pourra pas hériter de toutes les propriétés d'un concept qui le subsume, pour cela nous avons ajoutées dans l'algorithme de subsomption, « le traitement d'héritage de l'exception », et ceci en enlevant les propriétés exceptés.

Exemple :
Le concept mammifère hérite des propriétés animal, vivipare et vertébré.
MAMMIFERE ≡ ANIMAL ⊓ VERTEBRE ⊓ δ VIVIPARE

Le concept ornithorynque est subsumé par le concept mammifère, il hérite des propriétés animal et vertébré mais il n'hérite pas la propriété vivipare puisqu'elle est exceptée.
ORNYTHORYNQUE ≡ ANIMAL ⊓ VERTEBRE ⊓ OVIPARE ⊓ $VIVIPARE^{\varepsilon}$.

L'algorithme de subsomption a été prouvé décidable et de complexité polynomiale. **[Boustia and Mokhtari 2009].**

3. Modèle de contrôle d'accès:

Nous entamons cette partie, par la présentation d'une description informelle de notre modèle. Le but principal de notre approche est d'autoriser l'administrateur de définir une politique de sécurité indépendamment de l'implémentation. La méthode choisie pour atteindre se but est l'introduction d'un niveau abstrait **[Boustia and Mokhtari 2010].**

Dans notre modèle de contrôle d'accès nous définissons deux niveaux :

-***Niveau Concret :*** contient les concepts : Subject et Object.
-***Niveau Abstrait:*** contient les concepts : Role et View.

On abstrait le sujet dans un rôle, Un rôle est un ensemble de sujet, sur lesquels on applique la même règle de sécurité, de la même façon une vue est un ensemble d'objets, sur lequel la même règle de sécurité est appliqué. Le schéma de notre modèle est présenté dans la **[figure : 10]**

Un niveau d'habilitation est donné aux rôles, et un niveau de classification est donné aux vue.

On peut définir des privilèges abstrait à partir d'entités abstraites et le niveau, et à partir de ses privilèges on peut déduire des privilèges concrets.

De plus, touts les privilèges dans notre modèle peuvent être exprimés avec un contexte, cela veut dire que cette politique de sécurité est dynamique.

Axiome.1 : (contexte normal). Par défaut, le contexte est normal.

Axiome.2 : *toute action* qui n'est pas permise est interdite. Dans ce cas, il suffit de définir juste la relation de permission.

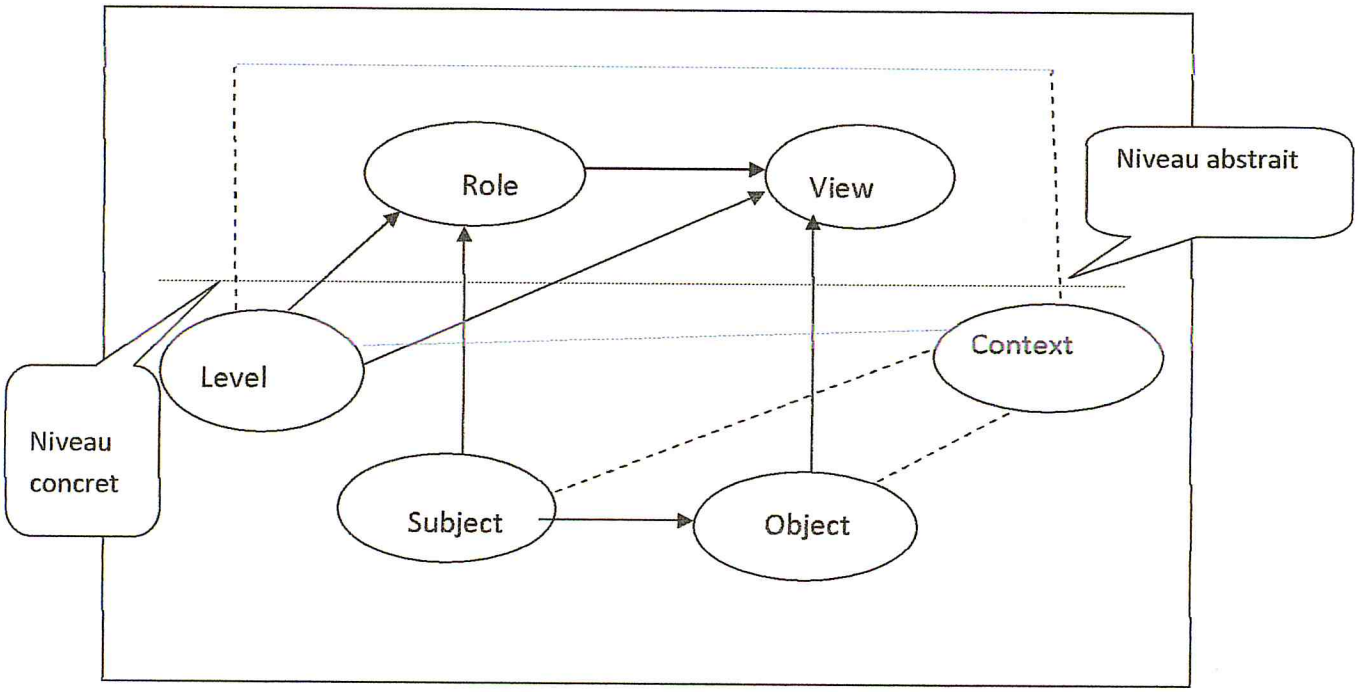


Figure 10 : Contrôle d'accès multi-niveaux basé sur un contexte

Maintenant On conceptualise le modèle de contrôle d'accès, et on construit une base de connaissances d'une LD capturant ses caractéristiques, y compris le contexte avec l'utilisation des connecteurs défauts (δ) et exceptions (ε).

On définit alors les axiomes de la *TBOX* et la *ABOX* avec des exemples, pour illustrer leurs contenus et utilités.

La TBox :

On définit une base de connaissance K en Logique de Description. L'alphabet de K inclut les concepts atomiques suivants :

Axiome d'attribution du rôle

Il définit la relation entre Sujet et Rôle

Subject $\sqsubseteq$ T

Role ⊑ T

Employ ⊑ EmployS.Subject ∏ EmployR.Role

-Les LDs ne permettent pas d'établir des relations de dimension supérieure à 2. Pour cette raison on a recours aux rôles: EmployS, EmployR qui sont des relations binaires entre concepts.

EmployS et EmployR sont des relations binaires tel que :

- EmployS : lie le concept Employ au concept Subject.
- EmployR : lie le concept Employ au concept Role.

Axiome d'attribution de la vue

Il définit la relation entre Objet et Vue.

Object ⊑ T

View ⊑ T

Use ⊑ UseO.Object ∏ UseV.View

UseO et UseV sont des relations binaires tel que :

- UseO : lie le concept Use au concept Object.
- UseV : lie le concept Use au concept View.

Axiome de définition de la classification

Il définit la relation entre la Vue et le Niveau de classification.

LevelV ⊑ T

Attribute⊑ AttributeV.View ∏ AttributeL.LevelV

AttributeV et AttributeL sont des relations binaires tel que :

- AttributeV : lie le concept Attribute au concept View.
- AttributeL : lie le concept Attribute au concept LevelV.

Axiome de définition de l'habilitation

Il définit la relation entre un Rôle et son Niveau d'habilitation, tel que :

LevelR ⊑ T

Assign⊑ AssignR.Role ∏ AssignL.LevelR

AssignR et AssignL sont des relations binaires tel que :

- AssignR : lie le concept Assign au concept Role.
- AssignL : lie le concept Assign au concept LevelR.

Pour définir une permission par défaut, on utilise l'axiome suivant :

Axiome d'attribution de permission

Il définit la relation entre Rôle et la Vue.

Une permission défaut est donnée à un rôle « **R** », sur une vue « **V** » quand son niveau d'habilitation est supérieur ou égale à son niveau de classification.

δPermission ⊑ PermissionR.Role ∏ PermissionV.View ∏ Attribue ∏ Assign ∏ LevelR

Tel que :

PermissionR et PermissionV sont des relations binaires, comme :

- PermissionR: lie le concept Permission au concept Role.
- PermissionV : lie le concept Permission au concept Vue.
- LevelR ⊑ haslevel AT-LEAST LevelV

Une permission concrète est exprimée dans l'axiome suivant:

Axiome de permission concrete

Il définit la relation entre Sujet et Objet.

Is-permitted⊑ Is-permittedS.Subject ∏ Is-permittedO.Object

Une permission concrète est donné à un Sujet "**S**" sur un Objet "**O**" tel que :

Is-permittedS et Is-permittedO sont des relations binaires, comme :

- Is-permittedS: lie le concept Is-permitted au concept Subject.
- Is-permittedO: lie le concept Is-permitted au concept Object.

Définition des règles de sécurité :

δIs-permitted ⊑ Employ ∏ Use ∏ δPermission

Si un Sujet "**S**" est employé dans un rôle "**R**" (Employ), et s'il y a une relation entre un objet O et une vue V (use), et si on a par défaut une " relation permission " entre rôle "**R**" et une vue "**V**" (δpermission), on déduit qu'un Sujet "**S**" a par défaut la permission d'exécuter une action " **Ac** " sur un objet " **O** " **(δIs-permitted).**

Et parce que Is-permitted ⊑ δIs-permitted alors une permission concrète peut être déduite d'une permission défaut. On peut finalement dire qu'un sujet "**S**" a la permission d'accéder à l'objet "**O**".

Is-permitted$^{\varepsilon}$ ⊑ Employ ∏ Use ∏ Permission$^{\varepsilon}$.

Par conséquent, si on a une exception, sur un concept permission, écrit Permission$^{\varepsilon}$, on dit qu'on a une exception sur le concept **Is-permitted**, écrit **Is-permitted$^{\varepsilon}$** , et parce que **Is-permitted ⊈ Is-permitted$^{\varepsilon}$** (une permission concrète ne peut pas être déduite par une permission exceptionnel). Alors on déduit qu'un sujet n'a pas la permission d'accéder à l'objet **O**.

4. Conclusion :

On a présenté, dans ce chapitre, un nouveau modèle de politique de sécurité, qui comble plusieurs lacunes laissé par les modèles précédents, en introduisant la notion d'autorisations dynamiques. Ceci a pu être réalisé en ajoutant un niveau abstrait, qui contient les concepts de rôles et de vues.

Dans cette nouvelle approche, au lieu d'assigner un niveau de classification aux objets et un niveau d'habilitation aux sujets, on les assigne respectivement aux vues et aux rôles.

Ce modèle a été formalisé par une Logique de Description non monotone à savoir DL-CMLAC$_{\delta\varepsilon}$.

Dans le chapitre suivant, on illustre l'utilisation de ce modèle par une application.

Chapitre 5

Implémentation

Chapitre 5
Implémentation

Après avoir achevé la phase théorique qui englobe les notions nécessaires à la réalisation de notre application, nous abordons dans ce chapitre la phase pratique (implémentation).Dans cette partie nous ferons une description de toutes les étapes de l'application en les illustrant avec des interfaces.

1. Outil de développement:

Notre travail a été conçu dans un environnement Windows 7, en utilisant le langage de programmation java pour implémenter l'algorithme d'inférence qui consiste à associer les permissions concrètes aux individus, et java swing pour implémenter les interfaces.

1.1 Windows 7 :

Windows 7 est le dernier en date des systèmes d'exploitations de la société Microsoft et successeur de Windows Vista sorti le 22 octobre 2009.

Cette nouvelle version de Windows reprend l'acquis de Windows Vista tout en apportant de nombreuses modifications, notamment par divers changements au niveau de l'interface, de l'ergonomie générale, un effort particulier pour la gestion transparente des machines mobiles et le souci d'améliorer les performances globales du système (fluidité, rapidité d'exécution même sur des systèmes moins performants, tels les netbooks) par rapport à son prédécesseur **[Windows7].**

1.2 Java :

Le langage java est un langage de programmation informatique orienté objet créé par **James Gosling** et **Patrick Naughton** employés de Sun Microsystems avec le soutien de **Bill Joy** (cofondateur de Sun Microsystems en 1982), présenté officiellement en 1995.
Le langage Java a la particularité principale que les logiciels écrits avec ce dernier sont très facilement portables sur plusieurs systèmes d'exploitation tels qu'UNIX, Microsoft Windows, Mac OS ou GNU/Lunix avec peu ou pas de modifications. C'est la plate-forme qui garantit la portabilité des applications développées en Java **[java].**

Nous avons choisi Eclipse **(Helios 3.6)** comme IDE pour java.

1.3 Eclipse :

Eclipse est un environnement de développement intégré libre extensible, universel et polyvalent, permettant de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM).

La spécificité d'Eclipse IDE vient du fait de son architecture totalement développée autour de la notion de plugin.

La base de cet environnement de développement intégré est l'*Eclipse Platform*, composée de :

- *Platform Runtime* démarrant la plateforme et gérant les plug-ins.
- SWT, la bibliothèque graphique de base de l'EDI.
- *JFace*, une bibliothèque graphique de plus haut niveau basée sur SWT.
- *Eclipse Workbench*, la dernière couche graphique permettant de manipuler des composants, tels que des vues, des éditeurs et des perspectives. **[Eclipse]**

1.4 EASY PHP:

EASY PHP (nous avons utilisé **EASY PHP dans sa version 2.1) [EasyPhp],** est une plateforme de développement Web, et un package comprenant :

- Le serveur de bases de données **MYSQL** version 4.1.9 ;
- Le compilateur du langage à script **PHP** version 4.3.10 ;
- Le serveur web **APACHE** version 1.3.33 ;
- L'interface d'administration **PHP MYADMIN** version 2.6.1 ;

1.5. LE SGBD MYSQL :

MYSQL est un serveur de base de données SQL multi-utilisateur et multi-thread (capacité d'une même application à effectuer plusieurs taches simultanées). Il a été essentiellement conçu pour pouvoir gérer de grandes bases de données très rapidement **[MySQL]**.

2. Implémentation

Comme nous l'avons déjà mentionné dans le chapitre précédent, notre travail consiste à réaliser en premier lieu le raisonneur « J-CLASSIC$_{\delta\varepsilon}$ » composé de deux parties dépendantes. La première partie permet la création des objets de la base (concepts, rôles, individus et règles) et la seconde partie fait appel aux mécanismes d'inférence.
Notons que les deux parties sont dépendantes car les mécanismes d'inférence se déclenchent en même temps que la création.
Après la conception de la base de connaissance, nous avons implémenté les algorithmes de subsomption et d'héritage dans DL-CMAL$_{\delta\varepsilon}$, qui nous permettent de raisonner sur la base de connaissance. Nous illustrons ceci à travers une étude de cas.

Figure 11: Interface d'accueil du système

2.1 Construction de la base de connaissances :

Une base de connaissances est simplement une collection de concepts, rôles, individus, règles et les relations entre eux.

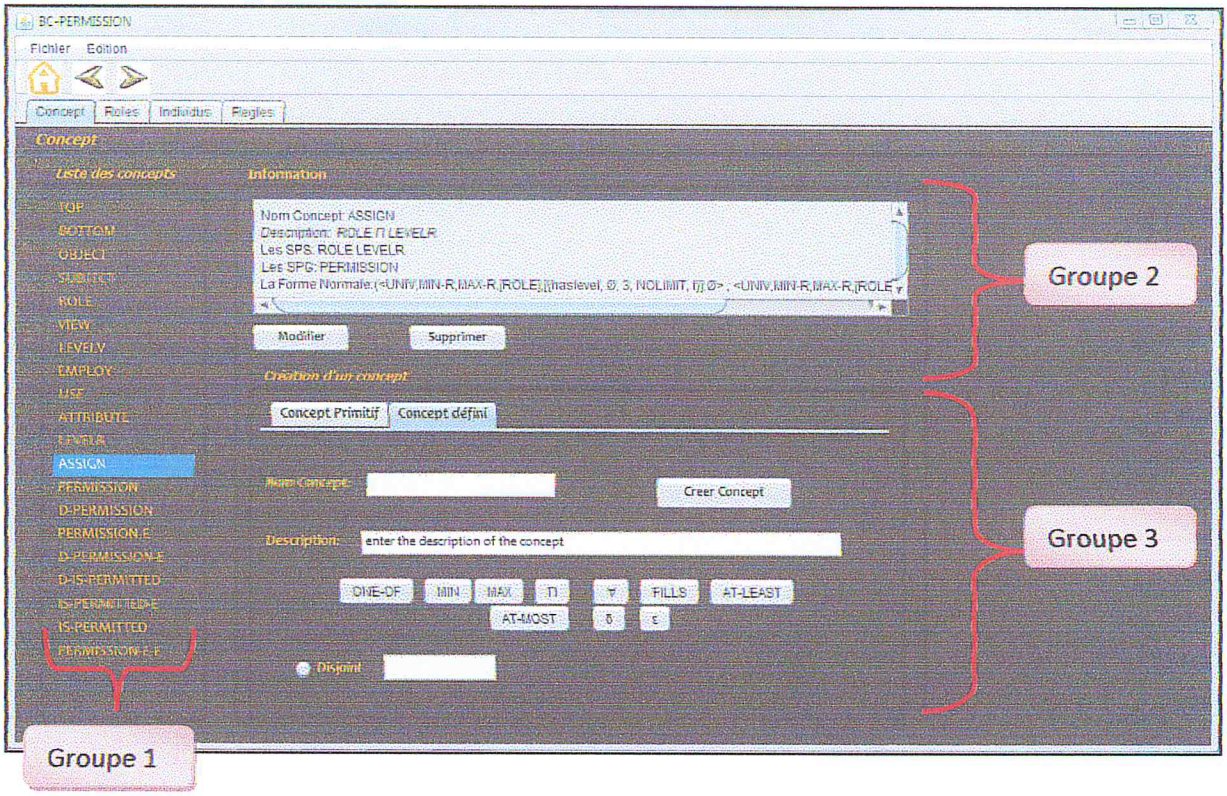


Figure 12: Construction d'une base de connaissances de DL-CMLAC$_{\delta\varepsilon}$

Cette fenêtre permet la création des objets de la base et cela en choisissant l'onglet correspondant à l'objet (ex: concept).

- Le 1er groupe: permet d'afficher toutes les instances des objets qui existent dans la base.
- Le 2eme groupe : permet d'afficher toutes les informations liées à un objet de la base.
- Le 3eme groupe : permet la création d'un nouvel objet.

Cet onglet permet la création des règles de la base de connaissances, la modification ou la suppression.

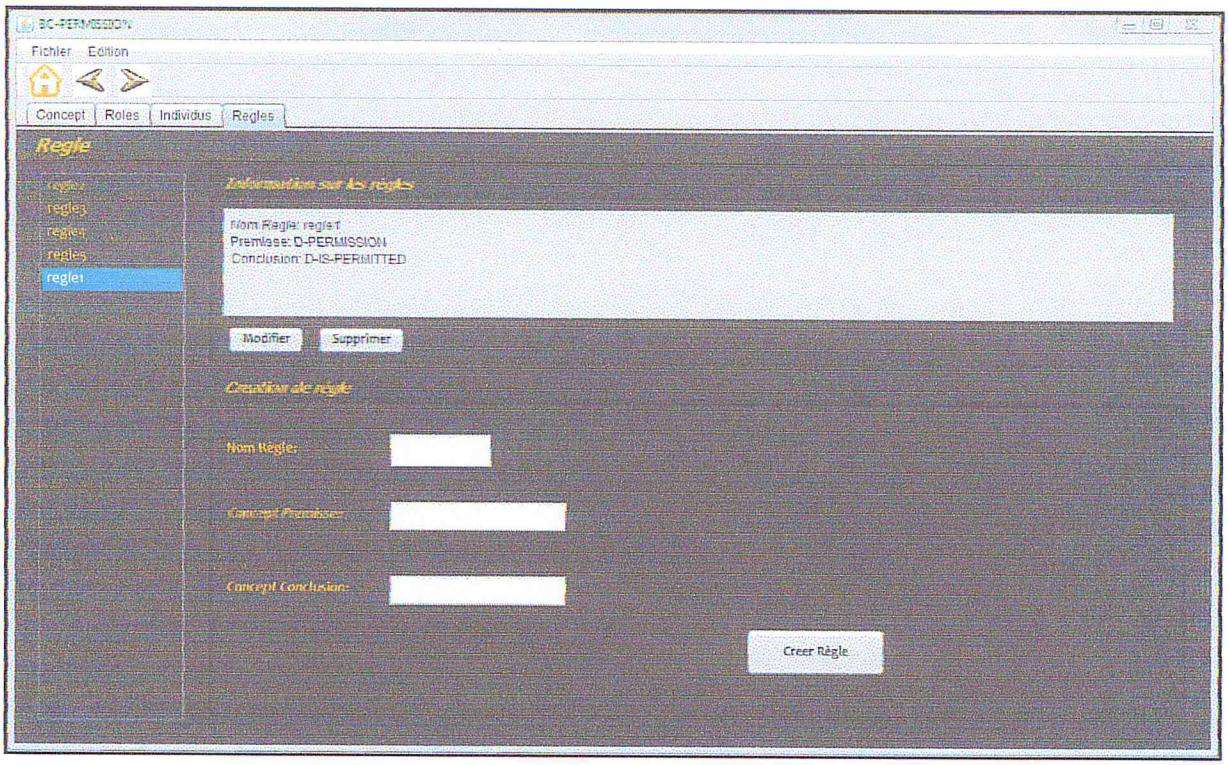


Figure 13: Ajout des règles dans la base de connaissances de DL-CMLAC$_{\delta\varepsilon}$

2.2 Inférence :

Dans cette section on présente un cas étudié dans lequel on démontre comment les autorisations peuvent être déduites dans des contextes différents définit par l'utilisateur de notre raisonneur.

Cas étudié :

Pour illustrer comment les autorisations peuvent être déduites, on peut utiliser les instances suivantes :

A-Box
Subject (Adam)
Role (Student)
View (Course)
Object (Db-course)
LevelR (Public)
LevelV (Public)
Employ (E1)⊑ EmployS.Subject (Adam) ∏EmployR.Role (Student)
Use (U1) Use⊑ UseO.Object (Db-course) ∏ UseV.View (Course)
Attribute (At1)⊑ AttributeV.View(Course) ∏ AttributeL.LevelV(Public)
Assign (As1)⊑ AssignR.Role(Student) ∏ AssignL.LevelR (Public)

Table A-Box

Utilisant les instances de cette ABox, le système infère que chaque personne qui joue le rôle d'étudiant (Student) est par défaut permit de consulter les cours (Course), et ajoute cette instance dans la A-Box :

δPermission (P1)

D'où,

δPermission (P1) ⊑ PermissionR.Role (Student) ∏ PermissionV.View (Course) ∏ Attribue(At1) ∏ Assign(As1) ∏ LevelR

En utilisant la précédente A-Box, on montre comment la déduction peut être faite dans différent contextes.

2.2.1 Contrôle d'accès dans un contexte défaut :

On suppose qu'Adam (User) veut lire Db-course, peut –il obtenir ce privilège ?

On sait que :

- Adam joue le rôle d'étudiant (Student) : Employ (E1) ;
- Et, Db-course est un Objet, utilisé dans la vue (Course) : Use(U1);
- Et, Student a un niveau d'habilitation égale à public : Assign (At1) ;
- Et, Course a un niveau de classification égale à public : Attribute(At1) ;

Et finalement, par défaut chaque personne qui joue le rôle d'étudiant (Student) a la permission de consulter Course, quand l'étudiant a un niveau d'habilitation public, et course est classé public et quand le contexte Normal est à vrai : δPermission (P1)

Formellement, on écrit :

Employ (E1) ∏ Use (U1) ∏ δPermission (P1)

En utilisant les règles de sécurités, on peut déduire que la proposition précédente subsume **δIs-permitted(I1)**, tel que :

Is –permitted (I1) ⊑Is –permittedS.Subject (Adam) ∏ Is –permittedO.Object (Db-course).

Et parce que,

Is –permitted (I1) ⊑ δIs –permitted (I1), alors on peut déduire qu'Adam à la permission de lire Db-course.

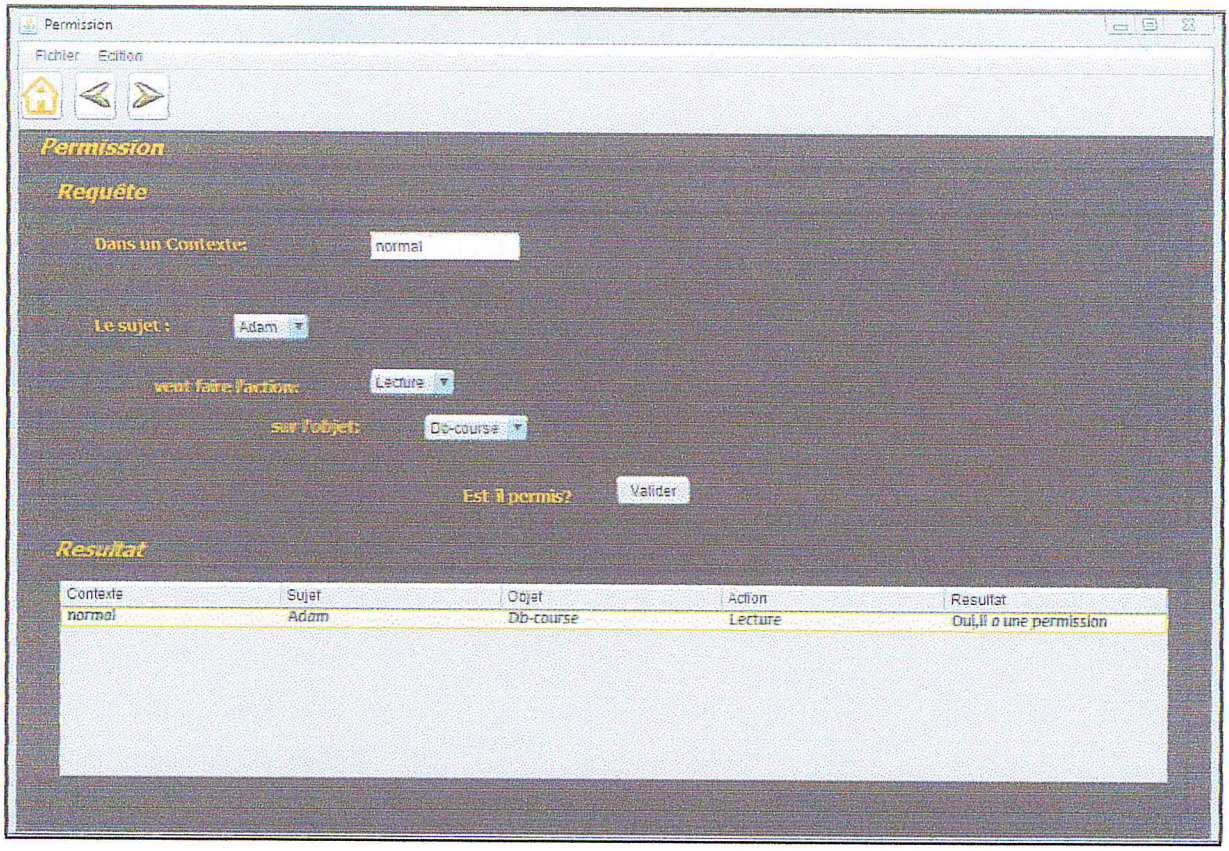


Figure 14: Déduction d'une permission par défaut

2.2.2 Contrôle d'accès, si le contexte « Absence de l'étudiant » est à vrai :

On suppose que l'étudiant Adam, n'a pas assisté à ses cours, et il veut lire Db-course, peut il obtenir ce privilège ?

Dans le contexte **« absence de l'étudiant »**, le système déduit une nouvelle instance P2, laquelle est définie comme suit :

δPermission (P2) ⊑ PermissionR.Role (Student) ∏ PermissionV.View (Course) ∏ Attribue(At1) ∏ Assign(As1) ∏ LevelR

Et on ajoute dans l'A-Box, la règle suivante:

Permission (P1) $^{\varepsilon}$ **⊑ δPermission (P2)**

On sait que :

Adam joue le rôle d'étudiant (Student) : Employ (E1) ;

- Et, Db-course est un Objet, utilisé dans la vue (Course) : Use(U1);
- Et, Student a un niveau d'habilitation égale à public : Assign (At1) ;
- Et, Course a un niveau de classification égale à public : Attribute(At1) ;

Finalement, par défaut chaque personne qui joue le rôle d'étudiant (Student) a la permission de consulter Course, quand l'étudiant a un niveau d'habilitation public, et Course est classé public et quand le contexte est « absence de l'étudiant » est à vrai :

δPermission (P2)

On obtient :

Employ (E1) Π Use (U1) Π δPermission (P2)

≡Employ (Emp1) Π Use (Use1) Π δ Permission (P1) $^{\varepsilon}$

On sait que : $A^{\varepsilon} \equiv \delta A^{\varepsilon}$

On obtient

≡Employ (Emp1) Π Use (Use1) Π Permission (P1) $^{\varepsilon}$

En utilisant les règles de sécurités, on peut déduire que la proposition précédente subsume **Is-permitted(I1)** $^{\varepsilon}$.

Et parce que **Is-permitted (I1)** $\not\subseteq$ **Is-permitted (I1)** $^{\varepsilon}$, on ne peut pas déduire Is-permitted (I1), donc Adam n'as pas le droit de lire Db-course, quand il n'a pas assisté à ses cours.

Notre politique, nous permet d'avoir plus d'une exception dans un contexte, par exemple : absence d'un étudiant accompagné d'une justification.

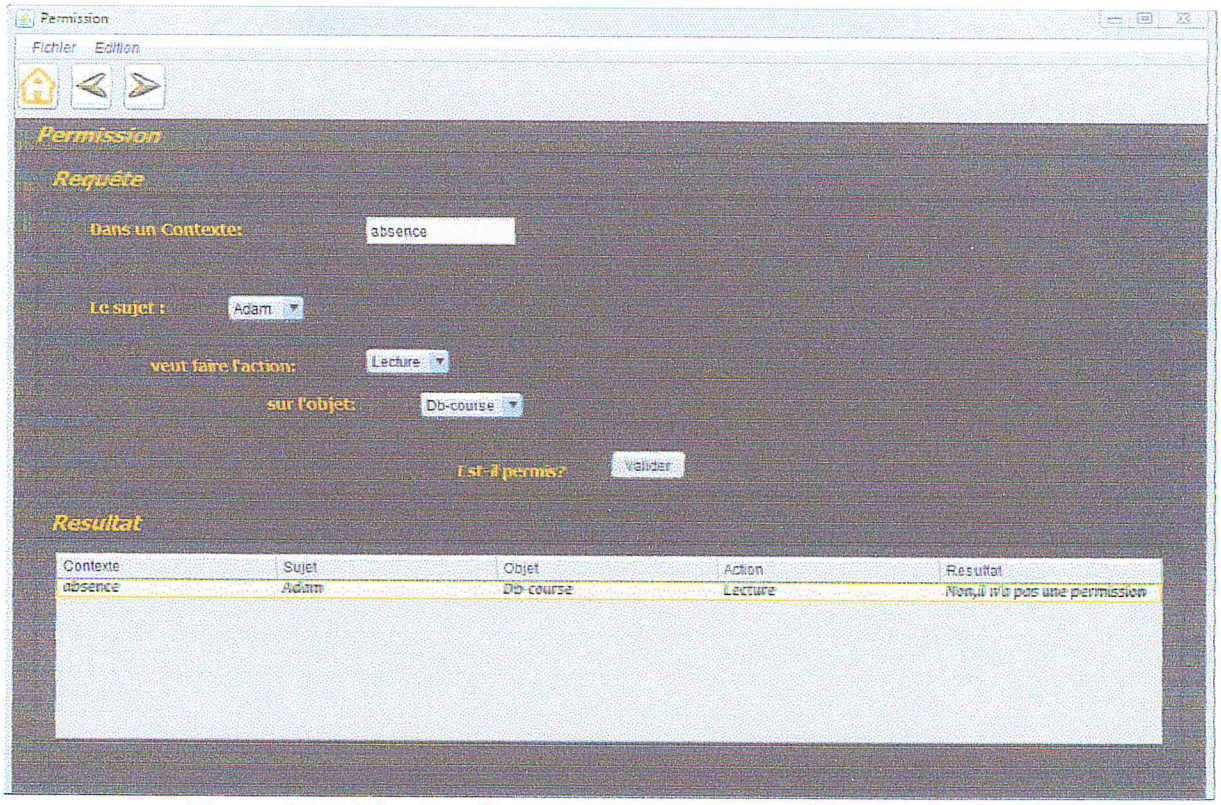


Figure 15: Déduction d'une permission en cas d'exception

2.2.3 Contrôle d'accès, si le contexte «Justification de l'absence d'un étudiant » est à vrai :

Supposons que l'étudiant « Adam » présente une justification pour son absence. Il veut lire «Db-course », peut il obtenir ce privilège ?

Dans le contexte « justification de l'absence d'un étudiant », le système déduit une nouvelle instance, qui est définie comme suit :

δPermission(P3) $\sqsubseteq$ PermissionR.Role(Student) Π PermissionV.View(Course) Π Attribue(At1) Π Assign(As1) Π LevelR

Et, on ajoute à la A-Box, la règle suivante :

$(\text{Permission(P2)}^{\varepsilon})^{\varepsilon} \sqsubseteq \delta$Permission(P3)

On sait que :

- Adam joue le rôle d'étudiant (Student) : Employ (E1) ;
- Et, Db-course est un Objet, utilisé dans la vue (Course) : Use(U1);
- Et, Student a un niveau d'habilitation égale à public : Assign (At1) ;
- Et, Course a un niveau de classification égale à public : Attribute(At1) ;

Finalement, par défaut chaque personne qui joue le rôle d'étudiant (Student) a la permission de consulter Course, quand l'étudiant a un niveau d'habilitation public, et Course a une classification public et quand le contexte est « justification de l'absence d'un étudiant» est à vrai :

δPermission (P3)

On obtient :

Employ (E1) Π Use (U1) Π δPermission (P3)

$\equiv$Employ (Emp1) Π Use (Use1) Π δ Permission $(\text{P2})^{\varepsilon}$

$\equiv$Employ (Emp1) Π Use (Use1) Π δ $(\text{Permission (P1)}^{\varepsilon})^{\varepsilon}$

On sait que : $(\delta\ A^{\varepsilon})^{\varepsilon} \equiv \delta\ A$

On obtient

$\equiv$Employ (E1) Π Use (U1) Π δ Permission (P1)

En utilisant les règles de sécurités, on peut déduire que la proposition précédente subsume **δIs-permitted(I1)**.

Et parce que **Is-permitted (I1) ⊑ δIs-permitted (I1),** on peut déduire Is-permitted (I1), donc Adam a le droit de lire Db-course, quand l'étudiant présente une justification pour son absence.

Figure 16: Déduction d'une permission en cas d'exception d'exception

3. Conclusion :

Dans ce chapitre, nous avons illustré notre modèle de contrôle d'accès à travers un exemple. Nous avons montré que notre politique est dynamique, vu que les autorisations sont recalculées à chaque changement de contexte.

Conclusion générale

Les modèles classiques de sécurité existant ne sont pas suffisamment expressifs pour une politique de contrôle d'usage. Un modèle plus expressif devra, d'une part, permettre de spécifier aussi bien les permissions, les interdictions et les obligations générales du SI. D'autre part, il devra permettre également de définir des règles de sécurité dépendant d'un contexte d'application. Ce qui est développé par le modèle ***multi-niveaux***.

Pour cela, nous avons développé un modèle de contrôle d'accès **multi-niveaux** basé sur le concept« contexte » pour générer une autorisation dynamique.

Pour formaliser notre modèle nous avons utilisé la Logique de Description, qui nous permet de représenter et de manipuler les différentes notions de notre domaine, comme les concepts, les rôles, les individus...

Les Logiques de Description classiques ne sont pas suffisantes. Pour les compléter, nous avons choisi une LD étendue non monotone, en utilisant les opérateurs défaut et exception, qui nous a donné la possibilité de réaliser un outil de raisonnement non monotone «J-Classic$_{\delta\varepsilon}$».

Les algorithmes que nous avons générer pour la classification des concepts sous les conditions strictes et défauts n'ont pas augmenté la complexité de l'expressivité du raisonneur.

Dans le futur, nous comptons étendre le travail à d'autres types d'application. Nous voulons aussi introduire d'autres types de contextes, tel que le contexte spatio-temporelle, et le changement de niveau de sensibilité des données et des sujets.

ANNEXES

Procédure de calcul de la forme normale : Normalisation

L'algorithme qui suit est un algorithme de calcul de la forme normale (procédure de Normalisation), qui distingue les termes de la forme ∀ r : C (avec C ≠ T), et ceux comportant le connecteur Π (appelés termes composés), et les autres termes appelés termes simples. L'algorithme utilise la procédure calcul-index et la procédure union-fn (notée ⊗).

Procédure de Normalisation

En entrée : D une description étendue d'un concept de J-CLASSIC$_{\delta\varepsilon}$.
En sortie : Fn (D), la forme normale de D.

Début
{*Cas de base: termes simples*}
Si D est T **alors** Fn (D) ← < (UNIV, MIN-R, MAX-R, ϕ, ϕ, ϕ), (UNIV, MIN-R, MAX-R, ϕ, ϕ, ϕ) >
Si D est ⊥ **alors** Fn (D) ← bo[1]
Si D est un concept primitif P
alors Fn (D) ← < (UNIV, MIN-R, MAX-R, {P}, ϕ, ϕ), (UNIV, MIN-R, MAX-R, {P}, ϕ, ϕ), >
Si D est ONE-OF E
alors Fn (D) ← < (E, MIN-R, MAX-R, ϕ, ϕ, ϕ), (E, MIN-R, MAX-R, ϕ, ϕ, ϕ), >
Si D est MIN u
alors Fn (D) ← < (UNIV, u, MAX-R, ϕ, ϕ, ϕ), (UNIV, u, MAX-R, ϕ, ϕ, ϕ), >
Si D est MAX u
alors Fn (D) ← < (UNIV, MIN-R, u, ϕ, ϕ, ϕ), (UNIV, MIN-R,u, ϕ, ϕ, ϕ), >
Si D est r FILLS E (E ≠ ϕ)
alors Fn (D) ← < (UNIV, MIN-R, MAX-R, ϕ, {<r, E, |E|, NOLIMIT,t>}, ϕ),
(UNIV, MIN-R, MAX-R, ϕ, {<r, E, |E|, NOLIMIT,t>}, ϕ)>
Si D est r FILLS ϕ **alors** Fn (D) ← t[2]
Si D est r AT-LEAST n (n >0)
alors Fn (D) ← < (UNIV, MIN-R, MAX-R, ϕ, {<r, ϕ, n,NOLIMIT, t >} ,ϕ),
(UNIV, MIN-R, MAX-R, ϕ, {<r, ϕ, n,NOLIMIT, t >} ,ϕ)>
Si D est r AT-LEAST 0 **alors** Fn (D) ← t
Si D est r AT-MOST n (n >0)
alors Fn (D) ← < (UNIV, MIN-R, MAX-R, ϕ, {<r, ϕ,0, n, t >} ,ϕ),
(UNIV, MIN-R, MAX-R, ϕ, {<r, ϕ, 0,n, t >} ,ϕ)>
Si D est r AT-MOST 0 **alors** Fn (D) ← < (UNIV, MIN-R, MAX-R, ϕ, {<r, ϕ,0, 0, bo >} ,ϕ),
(UNIV, MIN-R, MAX-R, ϕ, {<r, ϕ,0, 0, bo >} ,ϕ)>
Si D est ∀R : T **alors** Fn (D) ← t
Si D est ∀R : ⊥ **alors** Fn (D) ← <(UNIV, MIN-R, MAX-R, ϕ,{<r, ϕ,0, 0, bo >} ,ϕ),
(UNIV, MIN-R, MAX-R, ϕ, {<r, ϕ,0, 0, bo >} ,ϕ)>

[1]bo: est une constante qui représente la forme normale du concept le plus spécifique, BOTTOM (⊥).
[2]t: représente la forme normale du concept le plus général, le concept top. (le premier cas dans l'algorithme)

```
{*Termes composés*}
Si D est ∀ r : C

Alors  Début Fn (C) ← Normalisation (C)
       Si Fn (C) = t alors Fn (D) = t
       Sinon  Si Fn (C) = b0
                    alors fn (D) ← <(UNIV, MIN-R, MAX-R, ϕ,{<r, ϕ,0, 0, bo >} ,ϕ),
                                    (UNIV, MIN-R, MAX-R, ϕ,{<r, ϕ,0, 0, bo >} ,ϕ)>
              Sinon Fn (D) ← <(UNIV, MIN-R, MAX-R, ϕ,{<r, ϕ,0, |Cσ.dom|, C>},ϕ),
                                    (UNIV, MIN-R, MAX-R, ϕ, {<r, ϕ,0, |Cσ.dom|, C >} ,ϕ)>
              FinSi
       FinSi

       Fin

Si D est δC alors   Début
                    Fn (C) ← Normalisation (C)
                    Fn (D) ← < (UNIV, MIN-R, MAX-R, ϕ, ϕ, ϕ), Fn (C )δ >
                    Fin
Si D est C ε alors  Début
                    Fn (D)σdom ← UNIV
                    Fn (D)σmin ← MIN-R
                    Fn (D)σmax ← MAX-R
                    Fn (D)σπ ← ϕ
                    Fn (D)σr ← ϕ
                    Fn (D)σε ← calcul-index (Fn (C) δ)
                    Fn (D)δdom ← Fn (C)δdom
                    Fn (D)δmin ← Fn (C)δmin
                    Fn (D) δmax ← Fn (C)δmax
                    Fn (D)δ π ← Fn (C)δ π
                    Fn (D)δr ← Fn (C)δr
                    Fn (D)δε ← Fn (D)σε U Fn ( C)δε
                    Fin
Si D est A Π B alors Début
                    Fn (A) ← Normalisation (A)
                    Fn (B) ← Normalisation (B)
                    Fn (D) ← Fn (A) ⊗ Fn (B) {*union de deux formes normales*}
                    Fin
FIN
```

Procédure calcul-index :

Le fait qu'une exception doit toujours portée sur un défaut, cela veut dire qu'à chaque fois que l'utilisateur saisie une exception dans sa description, il faut vérifier que cette exception est déjà portée sur un défaut.

Pour un gain d'espace mémoire, nous définissons la procédure calcul-index qui consiste à coder les sextuplets défaut de la description, et cela en construisant une table d'index

comportant des couples (nombre, sextuplet), où nombre correspond au codage du sextuplet défaut qui sera utilisé dans le calcul de la forme normale.
Cette façon de coder est importante dans le cas où la description comprend une série d'ε imbriqués.

Procédure calcul-index (S) (S est un sextuplet)

```
Début
 Si il existe un couple (n, s) dans la table d'index alors calcul-index(s) ← n
   Sinon  on crée un nouveau couple dans la table d'index (m, s)
          où m=nombre de couples dans la table d'index +1
    calcul-index ← m
FIN
```

Procédure d'union de deux formes normales ⊗ :

Elle permet de faire l'union de deux sextuplets. Cette union se fait sur chacun de leurs champs (dom, min, max,π r,ε).

Procédure union-fn

En entrée : deux sextuplets A et B:
(a.dom, a.min, a.max, a.π, a.r, a.ε) et (b.dom, b.min, b.max, b.π, b.r, b.ε)
En sortie : (u.dom, u.min, u.max, u.π, u.r, u.ε) le sextuplet résultant de l'union de A et B

```
Début
  u.dom ← a.dom ∏ b.dom
  u.min ← maxi (a.min, b.min)
  u.max ← mini (a.min, b.min)
  u.π ← a.π U b.π
  u.ε ← a.ε U b.ε
  u.r ← ϕ
  Pour tout <r, fills1, least1, most1, C1> ∈ a.r Faire
    Si ∃< r, fills2, least2, most2, C2 > ∈ b.r
       u.r ← u.r U < r, fills, least, most, C> avec:
       fills ← fills1 U fills2
        C ← C1 ⊗ C2
      least ← maxi (least1, least2, |fills|)
      most ← mini (most1, most2, |Cσ.dom|)
      Si least = |Cσ.dom| alors fills ← Cσ.dom
      Sinon Si most =|fills| alors Cσ.dom ← fills
                Cδ.dom ← Cδ.dom ∏ Cσ.dom
            FinSi
      FinSi
     u.r ← u.r U < r, fills1, least1, most1, C1>
    FinSi
   Pour tout < r, fills, least, most, C> ∈ b.r telque ∄ d'élément de nom r dans a.r
```

```
        u.r ← u.r U < r, fills, least, most, C>
    FinPour
FinPour
FIN
```

Procédure de comparaison de deux formes normales (Comparer) :

La procédure Comparer représente la seconde partie de l'algorithme de $Sub_{\delta\varepsilon}$. Cette procédure permet de tester l'égalité entre deux sextuplets t1 et t2.
t1 (respectivement t2) est de la forme: (ti.dom, ti.min, ti.max, ti.π, ti.r, ti.ε), avec i = 1 (respectivement 2).

Procédure Comparer (T1, T2, Réponse) (equals)

```
Début
  Si (T1. dom ≠[3] T2.dom) ou (T1.min ≠T2.min) ou (T1.max ≠T2.max) ou (T1.π≠[4]T2.π)
      ou  (T1.ε ≠ T2.ε)
  Alors Réponse ← Non

  Sinon {*Comparaison des champs rôles*}
    Début
      Appel Comparer-rôle (T1.r, T2.r, rep)
      Réponse ← rep
    Fin
  Fin Si
FIN
```

3 Il s'agit d'une égalité ensembliste (l'ordre des éléments n'intervient pas).
4 Les champs ε sont des ensembles de nombres qu'il suffit de comparer. Il est inutile d'étendre les nombres par Des sextuplées leur correspondant dans la table index

Procédure Comparer- rôle (A.r, B.r, Réponse -Rôle) (ComparerIR)

La procédure Comparer fait appel à la procédure Comparer-rôle qui permet de tester l'égalité des ensembles dénotant les rôles.

Début
Réponse-Rôle ← Oui
Si |A.r|≠|B.r| **alors** Réponse-Rôle ← Non
Tant que (A.r ≠ ϕ) et (Réponse-Rôle = Oui) **faire**
Soit e∈ A.r (e = <r, fills1, least1, most1, c1>); chercher e' dans B.r de même nom r
Si e'∉ B.r **alors** Réponse-Rôle ← Non
Sinon soit e'∈B.r (e = <r, fills2, least2, most2, c2>)
Si (fills1≠ fills2) ou (least1≠least2) ou (most1≠most2) **alors** Réponse-rôle ← Non
Sinon
Début {*Comparaison des sextuples strict et défaut des formes normales C1 et C2*}
Appel Comparer (C1.σ, C2.σ, reps)
Si reps= Oui **alors**
Début
Appel Comparer (C1.δ, C2.δ, repd)
Réponse-Rôle ← repd
Fin
Sinon Réponse-Rôle ← Non
FinSi
Fin
A.r ← A.r-e
B.r ← B.r- e'
Fin Tant que
FIN

Algorithme de classification :

Cette procédure permet de placer un concept C (défini ou primitif) à la place la plus appropriée dans la hiérarchie, elle fait appel à deux procédures TrouverSPS et TrouverSPG.

Algorithme Classifier

En entrée : le concept C à classifier.
En sortie : les deux tableaux SPS, SPG du concept C et de nouveaux liens sont rétablis.

Début
TrouverSPS ()
TrouverSPG ()
FIN

6 Dans notre raisonneur, la classification ne s'applique que sur des concepts et non pas sur des rôles.

Procédure de calcul des SPS :

Étant donné un concept C à insérer dans la hiérarchie, le raisonneur J-CLASSIC$_{\delta\varepsilon}$ commence son calcul à partir de la racine de cette hiérarchie, c'est-à-dire, du concept TOP.
Dans un premier temps, il fait un test de subsomption entre le concept C et les SPG du concept TOP (fait appel à la procédure TrouverSubsumants), si aucun concept du tableau des SPG du concept TOP ne subsume le concept C, alors les sous- hiérarchies des SPG du concept TOP sont élaguées, et C sera classé juste après le TOP. Sinon, il fait appel à la procédure TrouverSPS (subsumant de C) qui fait le même traitement que TrouverSPS mais, en prenant les éléments du tableau des subsumant de C comme racine au lieu du concept TOP. Cette procédure met à jour le tableau des SPS du concept C.

Procédure TrouverSPS ()

En entrée: le concept C à classer, tableau des SPG du concept TOP
En sortie: le tableau des SPS du concept C

Début
à partir du tableau des SPG du concept TOP trouver tous les subsumants du concept C
{*Appel TrouverSubsumant*}
Si subsumants de C n'existe pas
Alors {*le concept C sera classifié juste après le concept TOP*}
- ajouter le concept TOP au tableau des SPS du concept C
Sinon appel TrouverSPS (subsumant de C)
FinSi
FIN

Procèdure TrouverSubsumant

En entrée : le concept C à classer, tableau de concepts TAB1
En sortie : tableau de concepts TAB2.

Début
Pour tous éléments Ci du tableau TAB1
Faire
Tester si le concept Ci subsume le concept C **{*Appel TestSubsume*}**
Si "oui" **alors** ajouter Ci au tableau des concepts TAB2
Fin
Fait

Procédure TrouverSPS (tableau de subsumant)

En entrée: tous les subsumants du concept C à classer.
En sortie: le tableau des SPS du concept C.

```
Début
Pour tous éléments Ci des subsumants du concept C
  Faire
    À partir des SPG du Ci trouver tous les subsumant du concept C
    {*appel Trouver Subsumant *}
    Si subsumants de C n'existe pas
    Alors ajouter Ci au tableau des SPS du concept C
    Sinon appel TrouverSPS (subsumant de C) {*Appel récursif*}
    Finsi
  Fait
Fin
```

Procédure de calcul des SPG :

Après le calcul des concepts les plus spécifiques du concept C, passons à la deuxième phase du processus de classification, qui consiste à calculer les concepts les plus généraux que le concept C (ses SPG).
Cette procédure permet de trouver les relations de subsomption, qui existent entre le concept C et les descendants de ses concepts les plus spécifiques. Le calcul commence par des tests de subsomption entre le concept C à classer et les SPG des SPS trouvés dans la première phase (Appel trouverSubsumé) et cela en descendant dans la hiérarchie.
Ensuite il parcourt la hiérarchie dans le sens inverse, i.e., à partir du concept BOTTOM à fin de trouver les SPG du concept à classer qui se trouve dans les sous- hiérarchies élaguées dans la première phase (Appel Trouver SPG (Tableau de concept)).
Cette procédure (TrouverSPG) permet de mettre à jour le tableau des SPG du concept C et aussi de rétablir les nouveaux liens qui existent entre le concept C et les concepts déjà existant dans la hiérarchie.

Annexes

Procédure TrouverSPG

En entrée : le concept C à classifier, tableau des SPS du concept C.
En sortie : tableau des SPG du concept C.

```
Début
  Pour tous élément Ci du tableau des SPS du concept C
   Faire ajouter le concept C au tableau des SPG
     {*Sachant que les éléments Ci sont les SPS du concept C alors C est le
        SPG de Ci*}
   Si le concept BOTTOM appartient au tableau des SPG de Ci
   Alors - supprimer le concept BOTTOM du tableau SPG du Ci
         - supprimer le Ci du tableau des SPS du BOTTOM
   Sinon Début - à partir des SPG de Ci trouvé les Cj qui sont subsumés par Ci
         {*appel TrouverSubsumé*}
          Pour tous Cj
            Faire
                - supprimer Ci du tableau des SPS de Cj
                - ajouter C au tableau des SPS de Cj
            Fait
          -ajouter les Cj au tableau des SPG de C
          -supprimer les Cj du tableau des SPG de Ci
         Fin
   FinSi
   Si le tableau des SPG du concept C est vide
    Alors -ajouter le concept BOTTOM au tableau des SPG du C
          -ajouter le concept C au tableau des SPS du BOTTOM
   FinSi
     -à partir des SPS du BOTTOM trouver les subsumés du concept C (nommé Ck)
        {*Appel Trouver Subsumé*}
      -Appel Trouver SPG (Ck)
  Fait
FIN
```

Procédure TrouverSubsumé

En entrée : le concept C à classer, tableau de concepts TAB1
En sortie: tableau de concepts TAB2.

```
Début
  Pour tout élément Ci du tableau TAB1
  Faire
  Tester si le concept Ci subsume le concept C {*Appel TestSubsume*}
  Si oui alors ajouter Ci au tableau des concepts TAB2
  Fait
FIN
```

Procédure TrouverSubsumé (Tableau de concepts TAB)

En entrée: tableau de concept, le concept C à classer.
En sortie : le tableau des SPS et SPG sont mis à jours.

```
Début
    Pour chaque élément Ck du tableau TAB donné en argument
    Faire
        À partir des SPS de Ck trouver les subsumés du concept C
    S'il n'existe pas de subsumé
    Alors
        Ajouter au tableau des SPG de C le concept Ck
        Si le concept BOTTOM appartient au tableau des SPG de Ci
        Alors - supprimer le concept BOTTOM du tableau SPG du Ci
              - supprimer le Ci du tableau des SPS du BOTTOM
        FinSi
        Ajouter au tableau des SPS du Ck le concept C.
    Sinon Appel TrouverSubsumé (subsumé de C) {*Appel récursif*}
    FinSi
Fin
```

Héritage :

Le raisonneur J-CLASSIC$_{\delta\varepsilon}$ est doté d'un mécanisme d'héritage, qui consiste à retrouver toutes les propriétés d'un concept C à partir des propriétés des concepts qui le subsument (de ses SPS).
Lors de la création d'un concept, le concept hérite automatiquement des propriétés strictes des concepts qui le subsument (grâce à leurs descriptions), néanmoins le fait que l'outil permette à l'utilisateur de saisir pour un concept donné un ensemble de groupes disjoints et que cet ensemble ne fait pas parti de sa description, un algorithme d'héritage des groupes disjoints a été implémenté.

Annexes

Algorithme HériterGroupe

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

```
Début
  Pour tous Ci du tableau des SPS du concept C
    Faire
      -ajouter les groupes disjoints et les groupes disjoints hérités du Ci au tableau des
        groupes disjoints hérités du concept C
      Si Ci est incohérent alors C est incohérent
    Fait
  Pour tous GRi du tableau des groupes disjoints de C
    Faire
      -ajouter le concept C dans la table des concepts des groupes disjoints.
      -Appel la procédure EstIncohérent {*Pour vérifier la cohérence de la BC*}
      Si les SPG du C sont différents de BOTTOM
       alors
        Pour chaque Ci du tableau des SPG du C
          Faire appel HériterGroupe pour le concept Ci {*Appel récursif*} Fait
      Fsi
    Fait
FIN
```

Détection d'incohérence :
Le raisonneur J-CLASSIC$_{\delta\varepsilon}$ permet de détecter un seul type d'incohérence qui est liée au groupe disjoint: nous utilisons l'algorithme EstIncohérent pour détecter ces incohérences.

Algorithme EstIncoherent

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

```
Début
    Pour tous élément Ci du tableau des SPS du concept C
    Faire
     Pour tous élément Cj du tableau des SPS du C tel que i≠ j
     Faire
      L'intersection des deux tableaux du groupe disjoint (Groupe Disjoint, Groupe Disjoint
            Hérité) de Ci et Cj
      Si l'intersection est nulle
      Alors return (false) {*le concept est cohérent*}
      Sinon return (true) {*le concept est incohérent*}
      FinSi
     Fait
    Fait
FIN
```

Détection d'équivalence :

L'outil permet aussi de détecter les équivalences entre deux concepts. Deux concepts sont équivalents si leurs formes normales sont équivalentes.

Algorithme EstEquivalent

En entrée: la description du concept C
En sortie: Booléen

```
Début
 pour chaque Ci du tableau des concepts
 Faire
  Si la forme normale du Ci ≡ la forme normale de C
   alors retourne (True)
 Fait
 return (False)
FIN
```

Déclenchement de règles :

Rappelons la forme des règles du raisonneur J-CLASSIC$_{\delta\varepsilon}$: les règles sont sous la forme d'un concept en prémisse et d'un concept en conclusion.
Ces règles permettent de déduire de nouvelles connaissances à partir de celles stockées dans la base. Par exemple, si on trouve un individu qui satisfait un concept se trouvant en prémisse d'une règle, on déduit qu'il satisfait aussi le concept se trouvant en conclusion de cette règle.
Cette déduction est effectuée à l'aide d'un mécanisme fonctionnant en chaînage avant.
Le déclanchement des règles se fait à différentes reprises, lors de la création ou modification d'un nouvel individu et aussi lors de la création, modification ou suppression d'une nouvelle règle, et cela permet de mettre à jour le tableau des instances des individus.

Remarque:
Si le déclanchement d'une règle engendre une inconsistance pour un individu alors cette règle ne sera pas déclenchée pour cet individu.

Algorithme DéclencherRègle

{*se déclenche lors de la création ou modification d'un nouveau individu*}
En entrée: individu Id, tableau des règles.
En sortie: tableau ConceptDéduit est mis à jour.

```
Début
 int taille,
 Faire
  taille ← nombre d'élément du tableau ConceptDéduit
 Pour chaque règle Ri du tableau des règles
  Faire
   Si [ContientPremisse (p, ConceptSPS) ou ContientPremisse (p, ConceptSPS)] et
      [Individu est non inconsistant]
   alors
   ajouter le concept se trouvant en conclusion au tableau ConceptDéduit de l'individu Id
   FinSi
  Fait
 TantQue taille ≠ nombre d'éléments du tableau ConceptDéduit.
 FIN
```

Procédure ContientPrémisse

En entrée: Concept P, tableau de concept TAB
En sortie: Booléen

```
Début
 Si P appartient au tableau de concept TAB
 alors return (True)
  Sinon Pour chaque élément Ci du tableau TAB
  Faire
   Si les SPS du Ci ≠ TOP
    alors Si ConteintPremisse (P, SPS du Ci) = true {*Appel récursif*}
          alors return (True)
          Fsi
   Fsi
  Fait
 return (false)
 Fsi
FIN
```

Algorithme Déclencher {*se déclenche lors de la création d'une nouvelle règle*}

En entrée: individu Id, tableau des règles.
En sortie: tableau des SPS de l'individu Id est met à jour.

```
Début
Pour toutes Idi du tableau des individus
  Faire
    Si ContientPremisse (p, ConceptSPS) ou ContientPremisse (p, ConceptSPS)
      {*Id est instance de la prémisse*}
      alors {*Id est aussi instance de la conclusion*}
      - déclencheRègle pour Idi
    FinSi
  Fait
FIN
```

Détection d'inconsistance :

L'outil détecte l'inconsistance des individus. Un individu est inconsistant si sa description est incohérente.

Algorithme EstInconsistant

En entrée: tableau ConcetSPS, tableau ConceptDéduit
En sortie: Booléen

```
Début
    tableau de Concept TabCon,
    TabCon ← union des deux tableaux: ConcetSPS et ConceptDéduit
    Pour chaque élément Ci du tableau TabCon
  Faire
    Pour chaque élément Cj du tableau TabCon
  Faire
        L'intersection des deux tableaux des groupes disjoints (groupes disjoints, groupes
        disjoints hérités)
    Si l'intersection est nulle
    alors return (false)
    Sinon return (true)
    Fsi
  Fait
Fait
Fin
```

Références

[Bell et La Padula,1976] Bell et La Padula, Secure Computer System : Unified Exposition and Multics Interpretation, 1976.

[Borgida & Patel-Schneider, 1994] A. Borgida, Patel-Schneider. Complete algorithm for subsumption in the CLASSIS description logics. Artificial Intelligence Research, 1994.

[Boustia and Mokhtari, 2009] N. Boustia, A. Mokhtari. JClassic$_{\delta\varepsilon}$: a non monotonic reasoning system. In Internal Report n°21-09, LRIA, USTHB, Algeria, 2009.

[Boustia and Mokhtari, 2010] N. Boustia, A. Mokhtari. A dynamic access control model. In Applied Intelligence Journal, DOI 10.1007/s10489-010-0254-z., 2010.

[Brachman, 1978] R.J. Brachman. A structural paradigm for representing knowledge. In Technical report 3605, BBN Report, 1978.

[Brachman et al., 1989] R.J. Brachman, D.L. McGuinness, L. Alperin Resnick, and A. Borgida. CLASSIC: A Structural Data Model for Objects. In Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, pages 59-67, June 1989.

[Brachman & al, 1991] Living with Classic : When and how to use a KLONE-like language. In J. F. SOWA, Ed., *Principles of semantic networks*, chapter 14, pp. 401-456,1991.

[Brachman & Levesque, 1985] Ronald J. Brachman and Hector J.Levesque, editors, Readings in knowledge representation. Morgan Kauffmann, Los altos, 1985.

[Brachman & Levesque, 1987] Levesque H. et Brachman R.J., Expressiveness an Tractability, in Knowledge Representation and Reasoning, Computational Intelligence, vol.3, 1987, pages 78-93.

[Brachman & Schmolze, 1985] An overview of the KL-ONE knowledge representation

[Common Criteria for Information Technology Security Evaluation, 2006] Security functional components, Part2. Technical Report CCMB-2006-09-002, sept 2006.

[Cohen & Hirsh, 1994] W.W. Cohen and H. Hirsh. Learning the CLASSIC description logic: Theoretical and experimental results. In International Conference of Knowledge Representation and Reasoning, pages 121-133. 1994.

[Coupey et Fouqueré, 1997] F. Coupey and C. Fouqueré. Extending conceptual definitions with default knowledge. In Computational Intelligence, Vol 13, Num 2,1997.

[DAC, 1971]B. Lampson. Protection. *5th PrincetonSymposium on Information Sciences and Systems*,pages 437-443, Mars 1971 .

Références

[Denning, 1982] D. E. Denning. Multilevel secure database systems: Requirements and model. In NAS/AFSB Summer Study on Multilevel Database Management Security, Working Paper, June 1982.

[EasyPhp] http://www.easyphp.org (Access date: 10 June, 2011).

[Eclipse] http://www.eclipse.org (Access date: 10 June, 2011).

[Fact++] http://owl.man.ac.uk/factplusplus/, 2007 (Access date:21 February, 2009).

[Ferraiolo et Kuhn, 1992] D. Ferraiolo et R. Kuhn. Role-based access control. Proceedings of the 15th NIST/NCSC National Computer Security Conference, pages 554–563, 1992.

[Finin, 1986] Finin T.W., Interactive Classification: A technique for acquiring and maintaining Knowledge Bases, in Proc. IEEE, vol. 74,1986, pages 1414-1421.

[Hattak, 2010] I. Hattak. Analyse formelle des politiques de sécurité, université du Québec en Outaouais, 2010.

[Java] http://www.oracle.com/us/technologies/java (Access date: 10 June, 2011).

[Kumar and al. , 2002] A. Kumar, N. Karnik and G. Chae. Context Sensitivity in Role Based Access Control. In ACM SIGOPS Operating Systems Review, pp. 53-66, 2002.

[Levesque et Brachman, 1987] H.J. Levesque and R.J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. Computational Intelligence, 3(2):78-93, 1987.

[MAC, 1975]K. J. Biba. Integrity consideration for securecomputer systems. Technical Report MTR-3153,The MITRE Corporation, Juin 1975.

[McGregor, 1988] McGregor R., A deductive Pattern Matcher, in Proc. 7th AAAI, Saint-Paul, Minnesota, 1988, pages 403-408.

[MacGregor, 1991] R. MacGregor, *Inside the LOOM clasifier*. SIGART bulletin, 2(3), pp 70-76, 1991.

[Mamache] F.Mamache.LES EXTENSIONS DE LA LOGIQUE DE DESCRIPTION, Faculté d'Electronique & Informatique, USTHB.

[MySQL] http://www.webotheque.fr/glossaire-web/sgbd-mysql.php, 2011(Acess date :11, June 2011).

[Nebel, 1988] B. Nebel. Computational complexity of terminological reasoning in Back. Artificial Intelligence, 34(3):371-383, 1988.

Références

[Nebel, 1990] Nebel, B. (1995). Reasoning and Revision in Hybrid Representation Systems. Lecture Notes in Computer Science, Springer-Verlag.

[Poinsot] L.Poinsot. Politiques et modèles de sécurité, Université Paris 13 - Institut Galilée.

[Racer] http://www.racer-systems.com, 2010 (Access date:17 May, 2010).

[Schmolze & Lipkis, 1983] Schmolze J.G. et Lipkis T.A., Classification in the KL-ONE Knowledge Representation System, in Proc. of the 8th IJCAI, Karlsruhe, 1983, pages 330-332.

[Schmidt-Schaub & Smolka, 1991] M. Schmidt-Schauÿ and G. Smolka. Attributive concept descriptions with complements. Artificial Intelligence, 48(1):1-26, 1991.

[Tsarkov *et al., 2006*] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. In Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006), volume 4130 of Lecture Notes in Artificial Intelligence,pages 292-297. Springer, 2006.

[Ventos, 1997] V.Ventos. C-CLASSICδε: une logique de description pour la définition et l'apprentissage de concepts avec défauts et exception. Thèse, université Paris-Nord, France, 1997.

[Windows 7] http:\\www.microsoft.com/en-US/windows7/products/home (Access date: 10 June 2011).

Résumé

Les modèles de contrôle d'accès comme DAC, RBAC, TBAC ou TMAC ne permettent de modéliser que des politiques de sécurité qui se restreignent à des permissions statiques. Ils n'offrent pas la possibilité d'exprimer des règles contextuelles relatives aux permissions, aux interdictions, et aux obligations. Le but de ce travail est de définir un modèle qui permet de spécifier de telles politiques de sécurité contextuelles. Ce modèle appelé A Contextual Multilevel Access Control s'appuie sur un langage formel basé sur la logique de description avec défaut et exception.

MOTS-CLÉS : *Contrôle d'accès, modèles de sécurité, modèle multi-niveaux, Logique de description, contexte.*

Abstract

None of the classical access control models such as DAC, RBAC, TBAC or TMAC is fully satisfactory to model security policies that are not restricted to static permissions but also include contextual rules related to permissions, prohibitions, and obligations. In our work we define a new model that provides solutions to specify such contextual security policies. This model, A Contextual Multilevel Access Control, is presented using a formal language based on description logic with default and exception.

KEYWORDS: *Access control, security model, multi-level model, Description Logic, context.*