N° D'ORDRE 2008-ISAL-0028 ANNÉE 2008

THÈSE

STRUCTURATION RELATIONNELLE DES POLITIQUES DE CONTRÔLE D'ACCÈS

REPRÉSENTATION, RAISONNEMENT ET VÉRIFICATION LOGIQUES

Présentée devant :
L'Institut National des Sciences Appliquées de Lyon

Pour obtenir :
Le grade de docteur

Spécialité :
Informatique

Formation doctorale :
Informatique

École doctorale :
Informatique et Information pour la Société

Par :
Romuald THION

SOUTENUE PUBLIQUEMENT DEVANT LE JURY COMPOSÉ DE :

Stéphane COULONDRE, Maître de Conférences, INSA de Lyon ..... Co-directeur de thèse
André FLORY, Professeur des Universités, INSA de Lyon ..... Co-directeur de thèse
Georges GARDARIN, Professeur des Universités, Université de Versailles ..... Examinateur
Philippe PUCHERAL, Professeur des Universités, Université de Versailles ..... Rapporteur
Florence SÈDES, Professeur des Universités, Université Paul Sabatier Toulouse ..... Rapporteur

LABORATOIRE D'INFORMATIQUE EN IMAGE ET SYSTÈMES D'INFORMATION

À qui de droit

Avant-propos

Remerciements

Étape difficile de la rédaction de thèse s'il en est, mais au combien appréciée : les remerciements. Qui remercier ? Dans quel ordre ? Pourquoi ? Tant de questions qui resteront certainement sans réponse aucune. Alors, évitons prudemment tout écueil en optant pour tout ce qu'il y a de plus classique : une grande liste sans ordre particulier.

Je tiens en premier lieu à remercier Stéphane Coulondre, qui est en grande partie le responsable de mon engagement dans la recherche et l'enseignement. C'est avec plaisir que je t'ai connu et que j'ai travaillé avec toi depuis bientôt 5 ans (mais ça ne rajeunit pas du tout ...). Je remercie également André Flory qui m'a livré quelques ficelles du métier et qui a su me conseiller avisément. Je remercie bien sûr les membres du jury, Georges Gardarin, Florence Sèdes et Philippe Pucheral, d'avoir accepté leurs charges respectives. J'adresse une dédicace particulière aux rapporteurs : un grand merci pour vos critiques. J'espère que vous ne m'en avez pas trop voulu en voyant le nombre de pages de la thèse.

Je remercie bien sûr l'ensemble des doctorants et autres chercheurs avec qui j'ai échangé et partagé (voire plus si affinités) pendant ces dernières années et qui se sont également lancé dans l'aventure d'une thèse. En premier lieu, ceux qui ont le plus souffert en étant mes co-bureaux : Jube, Sandro, Yann, Ny-Hango, les voisins d'en face ou d'à côté : Marian, Jean-Marc (x2) Ieva, Jeremy, Céline, sans oublier les collègues du CITI (bouh, un autre labo !) : Danto, Yvan, Nico. Je pourrai certainement encore étendre cette liste je pense.

Enfin, je remercie ma famille ainsi que tous les proches qui m'ont soutenu et avec qui j'ai passé tant de temps ces dernières années. Chers amis de lycée (Maillet, Angus, Couscous, Jaymz, Loutre, Bozot, Toto et d'autres vieux raisins), de l'INSA et toute la bande d'IF (Gus, Lorentz, Pastis, Seb, Damien et vos compagnes respectives bien sûr !) ou d'ailleurs (attention, certains cumulent des mandats de catégories, n'est ce pas Jube ?), je vous remercie tous. Et une dernière bise pour Virginie, pour la route.

Résumé

LES enjeux de la sécurité des systèmes d'information, avec l'omniprésence de l'informatique, la mise en ligne des systèmes et l'augmentation du nombre d'utilisateurs, sont devenus considérables : sécurité nationale, image de marque, aspects légaux ou encore pertes financières directes et indirectes colossales.

Le *contrôle d'accès*, ou *autorisation* – c'est-à-dire le mécanisme qui définit et impose ce qu'il est permis et interdit de faire – est un outil technique et organisationnel incontournable lorsqu'on envisage de garantir la sécurité d'un système. Initié par la défense américaine dans les années 70, ce domaine de recherche traite de multiples problématiques relatives à la notion de droit dans un système : structuration, formalisation, sémantique, stockage, représentation ou encore vérification et raisonnement.

Face à la diversité et à la taille croissante des systèmes d'information, les modèles historiques de contrôle d'accès ont trouvé leurs limites : trop rigides, insuffisamment sûrs ou difficiles d'administration. Ces limites ont conduit à la proposition du *contrôle d'accès à rôles* (*Role Based Access Control* – RBAC) dont le principe est d'introduire un niveau d'indirection entre utilisateurs et permisions [Ferraiolo03b]. Depuis – initiées par les modèles RBAC – de nombreuses propositions sur le contrôle d'accès ont été faites, dont on peut identifier deux grandes tendances :

- d'une part, une modélisation de plus en plus riche et complexe des organisations, permettant de mieux gérer les droits d'accès en les calquant sur la réalité pour faciliter le travail des administrateurs : multiples niveaux d'indirection et d'abstraction ou introduction d'aspects temporels par exemple. Ces propositions, guidées par les besoins, tendent à manquer de cadres formels et d'outils cohérents, rendant difficile leur intégration homogène dans les systèmes.
- d'une autre part, des propositions fondées sur des aspects théoriques, qui tâchent à partir d'un cadre logique déterminé de proposer des langages et des outils permettant de modéliser et de raisonner sur les droits d'accès : utilisation de la négation pour exprimer des politiques hybrides d'autorisation et d'interdiction ou raisonnements temporels par exemple. Ces propositions, basées sur des cadres logiques novateurs, peuvent être en revanche assez éloignées des préoccupations et des besoins des administrateurs.

La ligne directrice de ce travail se fonde sur la *fertilisation croisée* entre ces deux approches : plutôt que de créer un cadre théorique sur mesure dans lequel exprimer une modélisation complexe, mais difficile d'utilisation, où à l'inverse, de proposer des solutions *ad hoc* pour des problèmes d'administration qui peuvent être généralisés, la thèse propose d'utiliser un cadre *bien fondé*, permettant d'*exprimer et de résoudre* de grandes classes de problèmes d'administration des droits d'accès, avec un souci d'*homogénéité* et d'*utilisabilité* de la proposition. Nous développerons d'une part un cadre logique pour le contrôle d'accès mais aussi un ensemble d'outils formels pour la conception et l'administration du contrôle d'accès.

La thèse montre que les modèles de contrôle d'accès, qui permettent d'organiser les droits dans un système, partagent des concepts communs. Après avoir identifié ces éléments structurants et les relations qui les unissent, nous montrons que ces modèles peuvent être formalisés à l'aide du modèle relationnel, et que les propriétés dont ils jouissent peuvent être modélisées à l'aide de *dépendances de données* : des classes de formes syntaxiques restreintes de la logique du premier ordre permettant de garantir l'*intégrité* de données relationnelles [Abiteboul95, Beeri84].

Les classes de dépendances sont organisées en ensembles de formules *de plus en plus expressives*, ce qui permet, selon les classes, d'exprimer des propriétés des modèles de contrôle d'accès *de plus en riches* : de la matrice de contrôle d'accès la plus simple – formalisée par une relation unique ([Lampson74]) – aux modèles les plus récents de la littérature, qui intègrent hiérarchies et contraintes – exprimés à l'aide des classes de dépendances les plus générales [Maher96]. L'approche permet de mettre à profit des résultats qui dépassent le cadre strict du modèle relationnel et résolvent des problèmes actuels d'expression, de vérification et de raisonnement sur les politiques de contrôle d'accès, tout en assurant une intégration homogène dans les systèmes d'information.

Un des problèmes majeurs du contrôle d'accès que nous traitons est de s'assurer qu'une politique est *intègre*, c'est-à-dire qu'elle satisfait bien aux propriétés qu'elle doit respecter. En effet, une grande partie des failles des systèmes sont dues à des erreurs ou des oublis d'administration : proposer des outils de vérification automatique permet d'éviter et de corriger ces erreurs, améliorant ainsi la sécurité des systèmes.

Par exemple, dans les politiques RBAC, on peut définir des rôles qui sont *mutuellement exclusifs* – aucun utilisateur ne doit pouvoir les endosser simultanément. Ce type de contrainte permet d'interdire qu'un utilisateur ne dispose pas de trop de droits : on ne souhaite pas qu'une personne puisse à la fois faire une demande d'achat et la valider. Les dépendances permettent de formaliser les propriétés algébriques que les contraintes respectent, de vérifier qu'elles sont cohérentes et qu'elles sont effectivement satisfaites par les politiques. L'expressivité des classes de dépendances les plus générales nous permet de traduire et de vérifier des contraintes complexes qui n'ont pas été, jusque-là, abordées dans la littérature.

Ce travail propose également de définir des outils et des méthodes pour prendre en compte le cycle de vie et l'évolution des modèles de contrôle d'accès. Supposons un cas de réorganisation d'une entreprise où le système d'information est restructuré. Les responsables de la sécurité choisissent d'introduire un nouveau concept pour faciliter la gestion des droits. Les outils proposés dans la thèse permettent d'exprimer cette restructuration et de vérifier que les droits exprimés dans le nouveau modèle sont bien équivalents à ceux dont les utilisateurs disposaient avant.

Dans le cadre de cette restructuration, les contraintes que les administrateurs avaient définies indépendamment les uns des autres se retrouvent désormais regroupées dans une seule et même politique. La thèse propose d'exploiter les algorithmes de preuve qui ont été développés pour les dépendances, pour éliminer les contraintes redondantes et ainsi réduire leur nombre, avec toujours l'objectif de faciliter l'administration. Notons que ces procédures de preuve ne passent pas par des réécritures ou des transformations qui compliquent le travail d'analyse des administrateurs.

Le travail proposé met à profit des ponts existants entre les dépendances et des domaines de recherche connexes. Nous proposons une représentation permettant d'exprimer graphiquement les règles et propriétés qui définissent les modèles de contrôle d'accès et de suivre les étapes des raisonnements sur les politiques à l'aide des graphes conceptuels [Salvat96].

Nous proposons également une méthode pour assister la réorganisation des droits nécessaire lors de la restructuration des politiques de contrôle d'accès à l'aide de l'analyse de concepts formels [Arévalo07]. Par exemple, quand une organisation décide d'introduire la notion de rôle dans ses politiques, alors qu'elle ne disposait jusque-là que des droits exprimés sous forme de listes d'autorisations, l'outil proposé assiste l'identification des hiérarchies rôles implicitement présents dans les listes.

Enfin, notre approche propose plusieurs perspectives d'utilisation de travaux issus des bases de données pour les problématiques du contrôle d'accès. L'étude des dépendances et de leurs applications est un domaine de recherche actif, abordant des thèmes variés : la réparation automatisée de base de données [Chomicki05], les requêtes consistantes [Bertossi06], le maintien des dépendances lors des mises à jour [Christiansen06] ou encore la correspondance de schémas [Fagin06].

Ces recherches devraient permettre d'apporter des éléments de réponse aux problématiques les plus actuelles sur le contrôle d'accès : réparation automatique de politiques, mode dégradé, interrogation de politiques distribuées, modélisation des droits des administrateurs ou encore coopération de politiques organisées selon des modèles différents.

Notes

Table des matières

Table des figures

Liste des tableaux

Liste des définitions

Don't rely on network safeguards and firewalls to protect your information. Look to your most vulnerable spot. You'll usually find that vulnerability lies in your people.

Kevin D. Mitnick – "The Art of Deception", p.79

1 Introduction

▷ *En début de chaque chapitre nous proposons un résumé des principales contributions présentées. Sur la page suivant chaque résumé, une table des matières du chapitre est proposée.*

Le présent chapitre est l'introduction de la thèse. Nous découvrons les problématiques de la sécurité et du contrôle d'accès dans les systèmes d'information. Nous présentons également les objectifs, nos motivations, nos principales contributions, le périmètre scientifique ainsi que la structure du manuscrit. ◁

Plan du chapitre

Le XX^{e} siècle fut marqué par la révolution des technologies de l'information et des communications. L'informatique est passée, en l'espace de quelques décennies, du statut de curiosité scientifique et de prototype militaire à celui d'outil incontournable et massivement répandu : désormais, toutes les organisations – de l'ordinateur domestique aux systèmes d'information des multinationales – stockent, traitent et transmettent des quantités massives d'information.

1.1 Sécurité des systèmes d'information

Avec l'ère du numérique, toute une variété de nouveaux défis techniques, scientifiques et sociaux est apparue. L'un d'entre eux est la *sécurité des systèmes d'information*. Il éveille utilisateurs, industriels et scientifiques. Alors que nous incluons tous dans les problèmes de sécurité, les logiciels malveillants ou le courrier indésirable qui polluent nos environnements de travail, les grandes entreprises redoutent la mise hors d'usage de leurs systèmes de production et la fuite d'informations confidentielles. Les enjeux de la sécurité sont devenus tels – image de marque, pertes financières directes ou indirectes massives – que des moyens considérables sont investis pour garantir la sécurité.

Les principes d'*authentification* et d'*autorisation* sont incontournables lorsqu'on envisage de garantir la sécurité d'un système. L'*authentification* concerne la preuve de l'identité, l'*autorisation* – synonyme de *contrôle d'accès* – définit et impose ce qu'il est permis et interdit de faire. La notion de contrôle d'accès est bien antérieure à l'informatique : de tout temps des gardes, des chaînes et des herses ont été dressés pour protéger des accès illégitimes.

Les recherches dans le domaine du contrôle d'accès furent initiées par le *Departement of Defense* américain (DoD) dans les années 70 [Lampson74]. Des modèles théoriques furent élaborés et très largement implantés, pour faire face aux besoins de sécurité, militaires à l'époque. Avec la démocratisation de l'informatique et son usage incontournable dans presque toutes les organisations, tous les acteurs de l'informatique prêtent désormais attention à la sécurité de leurs systèmes d'information.

1.2 Besoins du contrôle d'accès

Les recherches sur le contrôle d'accès s'attachent à de multiples problèmes relatifs à la notion de droit dans un système : organisation, formalisation, sémantique, architecture... Une des rencontres majeures sur le sujet est l'ACM *Symposium on Access Control Models And Technologies*[1]. Le regain d'intérêt sur le contrôle d'accès – motivé par les enjeux et besoins de sécurité – date des années 1990, période où la famille des modèles de

[1] SACMAT, dont l'adresse du site internet est `http://www.sacmat.org`

contrôle d'accès basés sur les rôles, appelés généralement *contrôle d'accès à rôles* pour *Role Based Access Control* (RBAC), ont été proposés pour la première fois.

La famille des modèles RBAC présente une nouvelle organisation des droits centrée sur le concept de *rôle*. Cette notion a été étudiée dans de nombreux domaines (modèles objets, philosophie, gestion des connaissances, modèles agents...) et introduite dans le contrôle d'accès pour simplifier l'administration des droits des grands systèmes. Depuis, les principes fondateurs des modèles RBAC ont largement été répandus et adaptés pour aboutir à la définition de nouveaux modèles raffinés, intégrants des notions de plus en plus complexes. De nombreuses propositions ont été faites, que d'aucuns appelleraient les modèles *x-Based Access Control*, comme les notions de treillis (*Lattice*-BAC), de temps (*Temporal*-RBAC, *Generalized-Temporal*-RBAC), d'équipes (*Team*-BAC), de tâches (*Task*-BAC), d'organisations (*Organization*-BAC) ou encore de contextes (*Context*-BAC) ont été proposées pour structurer les droits.

La volonté sous-jacente à cette profusion de modèles de contrôle d'accès est de proposer une organisation des droits qui permette de traduire au mieux les « politiques de sécurité », c'est-à-dire l'ensemble des règlements qui régissent la façon dont sont protégées les ressources d'un système. Ces politiques sont exprimées en langue naturelle au début du cycle de développement du contrôle d'accès. Il faut ensuite traduire ces règlements dans les systèmes grâce à des modèles de contrôle d'accès qui soient :

- suffisamment expressifs, pour permettre l'expression de règlements complexes et représenter fidèlement la structure et le fonctionnement des organisations,
- décidables, pour évaluer les autorisations à partir des règlements,
- faciles d'administration, les modèles doivent fournir un ensemble de primitives et de fonctionnalités simplifiant les activités des administrateurs.

La mise en place d'une politique de contrôle d'accès dans un système est un chantier important pour une organisation, à la croisée de l'informatique, de la gestion des ressources humaines et de la qualité. Dans cette activité interviennent de multiples problématiques relatives à la notion de droit dans un système, dont nous identifions six axes de recherche majeurs.

La conception de nouveaux modèles de contrôle d'accès. La famille RBAC a ouvert un grand nombre de perspectives d'application et a contribué à fonder une nouvelle structuration du contrôle d'accès, où des concepts et relations intermédiaires entre utilisateurs et permissions sont introduits pour organiser les droits d'accès de façon souple et efficace.

La formalisation des modèles de contrôle d'accès. Un cadre logique dans lequel spécifier formellement les modèles de contrôle d'accès permet une meilleure caractérisation des propriétés qu'ils doivent satisfaire et donne un langage d'expression commun sans ambiguïtés. La formalisation permet également d'utiliser des outils mathématiques pour raisonner sur les modèles.

L'enrichissement des modèles de contrôle d'accès. Les notions de contraintes et de propriétés de sécurité ont été étudiées pour ajouter du sens aux politiques de contrôle d'accès. Le but de ces travaux est de permettre aux organisations d'exprimer les spécificités de leurs métiers par l'intermédiaire de règles et de restrictions que les politiques de contrôle d'accès doivent satisfaire.

La vérification des politiques de contrôle d'accès. S'assurer qu'une politique est sûre, équivalente à une autre, qu'elle instancie bien un modèle ou qu'elle satisfait à des propriétés de sécurité sont des besoins actuels des administrateurs. Avec la taille des politiques, les capacités humaines sont mises en défaut : des outils permettant d'automatiser l'analyse et la vérification des politiques deviennent nécessaires.

La mise en œuvre du contrôle d'accès. Que ce soit dans les systèmes d'exploitation ou les systèmes de gestion de bases de données, il faut définir comment, à partir des modélisations conceptuelles puis des formalisations, mettre en œuvre le contrôle d'accès effectif dans les systèmes.

Les méthodologies de mise en œuvre du contrôle d'accès. Il faut proposer des outils et des méthodes pour que les organisations puissent mettre en place un contrôle d'accès structuré. Cette activité côtoie celles du génie logiciel, de la modélisation conceptuelle et de la représentation de connaissances.

1.3 Motivations

La thèse propose des outils et des méthodes répondant aux différents besoins qui surviennent au long du cycle de vie du contrôle d'accès : de la conception du modèle à l'administration et l'évolution des politiques. Nous voulons apporter une réponse cohérente aux problèmes de formalisation, d'expression, de vérification, de conception et de représentation posés par le contrôle d'accès.

Pour cela nous proposons un socle commun bâti sur la caractérisation logique des modèles de contrôle d'accès qui permette la conception de modèles adaptés aux besoins des acteurs de la sécurité. De plus, la thèse munit le cadre proposé d'outils pour la réalisation des activités d'administration, de conception et de vérification des politiques. Les *critères* qui guident la réalisation de ces objectifs sont :

l'expressivité : nous souhaitons un cadre qui permette de *concevoir* des modèles de contrôle d'accès élaborés et d'exprimer des contraintes complexes sur ces modèles. L'expressivité de la proposition permettra par exemple de développer de nouvelles classes de contraintes et de propriétés de sécurité,

l'homogénéité : nous souhaitons une approche qui permet de répondre aux problématiques qui surviennent tout au long du cycle de vie du contrôle d'accès dans un cadre formel sous-jacent *homogène*. Le choix d'un paradigme logique sur lequel nous pouvons raisonner directement sans passer par des transformations et des réductions est guidé par ce critère,

le bon fondement : nous désirons que la proposition se base sur un cadre logique *bien fondé* et *établi*, sur lequel de nombreux résultats et applications ont été proposés. Nous ne voulons pas composer un paradigme *ad hoc*, mais mettre à profit des résultats existants et reconnus comme sûrs, et nous en servir pour répondre et anticiper les problématiques du contrôle d'accès,

l'utilisabilité : nous désirons que notre proposition soit utilisable par les responsables de la sécurité des systèmes d'information et qu'elle réponde à leurs attentes en étant proche des systèmes existants, sans proposer de surcouche mais plutôt en *intégrant* les notions de contrôle d'accès au cœur des systèmes d'information. Cette motivation nous conduit à éviter les mécanismes boîtes noires, à se rapprocher des bases de données.

Nous proposons d'utiliser le cadre des *dépendances de données* pour caractériser et répondre aux classes de problèmes que nous avons présentées. La notion d'*intégrité* des modèles et des politiques en particulier est au centre de notre approche. Notre proposition innove en intégrant des aspects transversaux du contrôle d'accès qui ont jusqu'ici été traités de façon indépendante, et se démarque en proposant une approche cohérente qui répond aux différents problèmes qui surviennent lors de l'élaboration des politiques de contrôle d'accès.

1.4 Contributions

Nous proposons une *structuration relationnelle* du contrôle d'accès, basée sur l'identification des concepts communs aux différentes propositions de ce domaine. Nous définissons un modèle de contrôle d'accès comme une organisation structurée de données, utilisée pour prendre une décision d'accès, c'est-à-dire accorder ou refuser à un sujet d'effectuer une action sur un objet [Thion07a]. La structuration relationnelle proposée notée $\mathbf{AC} = (sch, P, \Sigma)$ est composée :

- d'un schéma relationnel *sch* qui permet d'organiser les politiques,
- d'un ensemble de principes P qui permettent de dériver de nouvelles informations à partir de celles connues, en vue de prendre les décisions d'accès,
- d'un ensemble de propriétés Σ que les politiques doivent respecter.

Il s'agit d'une structuration fondée sur les fondements logiques des bases de données relationnelles. Les principes P et les propriétés Σ sont des ensembles de formules logiques closes. Les dépendances de données sont des fragments de la logique du premier ordre. Elles permettent l'expression de principes et de propriétés complexes. Différentes restrictions sont imposées aux composants de la structure. On impose notamment qu'il existe une seule et unique plus petite interprétation de P. Les différentes classes de dépendances à l'expressivité et à la décidabilité contrôlées nous permettent de définir et de faire varier ces restrictions.

La généricité de la structure relationnelle permet de concevoir différents modèles de contrôle d'accès. Nous illustrons notre approche avec la famille des modèles à rôles dont nous prenons en compte de nombreuses extensions qui ne font pas partie du standard RBAC : hiérarchies multiples, différentes sémantiques de l'exclusion, aspects contextuels, contraintes de prérequis et surtout contraintes d'intégrité.

À partir de la structuration relationnelle, nous définissons ce qu'est une politique de contrôle d'accès : une instance en extension du schéma. Pour faciliter l'administration, les propriétés définissent comment déduire une politique en intention sur laquelle sont basées les décisions d'accès. Les propriétés Σ permettent de garantir l'intégrité des politiques. S'assurer de la cohérence des droits est pour nous un objecttif central [Thion06c, Thion06a].

À partir de la structuration relationnelle nous définissons les différentes activités qui interviennent sur le contrôle d'accès : la conception de modèle, de politique et l'administration des politiques. Les outils algorithmiques existants pour les dépendances sont utilisés pour répondre aux problèmes de raisonnement et vérification du contrôle d'accès. Ils permettent d'automatiser certaines tâches et de s'assurer de l'intégrité des politiques. Nous exploitons également des connexions entre les dépendances et les graphes conceptuels ou l'analyse de concept formel pour enrichir la palette d'outils proposés [Thion06b, Thion07b].

1.5 Organisation du manuscrit

1.5.1 Principe de Quintilien

Quintilien est un rhéteur latin du I^{er} siècle après J.-C. auquel on doit un principe[2] – *Quis, Quid, Ubi, Quibus Auxiliis, Cur, Quomodo, Quando* (qui, quoi, où, avec quels moyens, pourquoi, comment, quand) – qui résume la démarche des instructions judiciaire. Nous proposons d'appliquer cette démarche afin identifier les aspects essentiels des problèmes auxquels nous nous intéressons dans la thèse :

***quis* – qui** : les responsables de la sécurité, les concepteurs de modèles de contrôle d'accès et les administrateurs de politiques,

***quid* – quoi** : les problèmes qui surviennent au long du cycle de vie d'une politique de contrôle d'accès, de la conception à l'évolution, en essayant de généraliser les approches proposées dans la littérature sur le contrôle d'accès,

***ubi* – où** : dans les systèmes d'information, en particulier *avec* les bases de données relationnelles qui en sont une des pierres angulaires, et qui disposent de nombreux mécanismes techniques robustes,

[2]Ce principe se rapproche d'un des sept outils de la qualité : le Qui, Quoi, Où, Quand, Comment, Pourquoi (QQOQCP) qui sert à identifier un problème à partir de six questions fondamentales.

quibus auxiliis – **avec quels moyens** : en proposant une structuration relationnelle, bien fondée et expressive, qui permet d'aborder les modèles de contrôle d'accès d'un point de vue générique,

cur – **pourquoi** : car la prise en compte du contrôle d'accès fait partie intégrante du système d'information et de la mise en place d'une politique de sécurité, et que cette prise en compte doit se faire de façon homogène et cohérente,

quomodo – **comment** : en étant le plus homogène possible et en s'intégrant au mieux dans les paradigmes existants, sans ajouter de mécanismes *ad hoc*, et en construisant des outils sur la structuration relationnelle,

quando – **quand** : au plus tôt du cycle de conception des systèmes, pour que la sécurité de l'information soit prise en compte dans le cycle de vie du système d'information.

1.5.2 Périmètre de l'étude

Nous avons identifié six axes de recherche majeurs qui vont guider notre proposition. D'autres directions sont envisageables et d'autres perspectives pourraient être prises en compte. La notion de délégation par exemple, qui est couramment utilisée dans les organisations, ou encore les principes de confiance, de partage et de collaboration sécurisés sont d'autres domaines de recherches actifs. Nous évoquerons certains de ces travaux, mais embrasser l'ensemble de ce champs d'étude dépasserait le cadre que nous nous fixons.

Devant l'étendue des directions prises par la recherche en sécurité et le dynamisme du domaine, nous avons effectué des choix pour définir les limites de notre travail. La portée de la thèse couvre la majorité des aspects introduits dans les modèles de contrôle d'accès. Or ces derniers sont inscrits dans le cadre plus large des *architectures* de sécurité. Le contrôle d'accès, qui est un objet d'étude à part entière, reste un *moyen* mis en œuvre pour garantir la sécurité des systèmes, mais seul il est inutile. La métaphore de la portée blindée aux multiples points sécurisant l'accès à une maison dont les fenêtres sont grandes ouvertes représente assez fidèlement cette situation.

Deux domaines en particulier sont étroitement liés au contrôle d'accès : d'une part la cryptographie, et d'autre part les mécanismes d'authentification qui se basent sur des primitives cryptographiques, comme les architectures de clefs publiques. Si nous évoquons ces aspects dans la thèse, ils ne sont pas l'objet principal de notre étude.

1.5.3 Plan de la thèse

Après cette introduction générale de la thèse, le chapitre 2 présente l'état de l'art relatif à nos travaux : après avoir précisé la place du contrôle d'accès au sein de la problématique générale de la sécurité des systèmes d'information, nous critiquons les

différentes modélisations du contrôle d'accès. Cette étude est conduite sur les aspects pratiques, mais aussi théoriques. Comme nous avons souhaité couvrir un champ relativement large des recherches sur le contrôle d'accès, deux essais de tableaux de synthèse concluent cette étude.

Ensuite, le chapitre 3 propose le cadre logique dans lequel nous formalisons le contrôle d'accès. Nous justifions le choix des dépendances de données pour réaliser nos objectifs avec les critères fixés. La description de ce cadre est illustrée par son application aux modèles à rôles.

Une fois le cadre logique déterminé, nous définissons dans le chapitre 4 une modélisation générique des autorisations dans les systèmes en nous appuyant sur les traits communs que partagent les modèles de contrôle d'accès. Nous portons une attention particulière à intégrer les différentes critiques qui leur sont adressées. Nous catégoriserons également les différents aspects intervenants lors de la réalisation effective du contrôle d'accès.

Dans le chapitre 5 nous développons l'utilisation de la modélisation générique proposée. Après avoir défini les principales activités relatives à l'organisation des droits dans les systèmes, nous proposons un ensemble d'outils pour la conception de modèles puis pour l'administration des politiques de contrôle d'accès.

Le chapitre 6 donne les clefs techniques de l'implantation de notre approche et propose des exemples d'applications. Ce chapitre décrit une architecture logicielle du contrôle d'accès reponsant sur les systèmes de gestion de bases de données pour le stockage des politiques. Nous présentons également une bibliothèque pour la manipulation de dépendances que nous avons développée pour le raisonnement et la vérification du contrôle d'accès.

Enfin, le dernier chapitre conclut la thèse en synthétisant et discutant notre approche et les principaux choix que nous avons faits. Le chapitre propose des perspetives de recherche organisées selon cinq axes : la mise en œuvre d'une architecture de contrôle d'accès, l'utilisation d'autres fragments de logique, l'extension du périmètre du contrôle d'accès, l'étude des modèles d'administration et enfin la réutilisation de nos travaux pour le domaine du contrôle d'accès aux réseaux.

Nous proposons de plus trois annexes. La première expose les notions de logique du premier ordre auxquelles nous faisons référence. Cette première annexe décrit également les liens qui unissent la logique aux fondements des bases de données relationnelles. La seconde annexe est technique. Elle propose des listes des principaux symboles utilisés dans la thèse ainsi que des exemples de modèles de contrôle d'accès. La troisième propose des traces de preuves obtenues avec notre prototype.

Exemple 1.1 Un exemple hospitalier filé

La thèse est illustrée par des exemples d'applications. Afin qu'ils soient homogènes et qu'ils représentent des situations de la vie réelle, ils portent souvent sur un établissement hospitalier, voire plusieurs. Pour s'éviter de pesantes tournures comme « supposons l'existence d'une structure hospitalière », nous proposons de prendre la même. Ces exemples sont « flottants », comme les figures ou les tableaux, et peuvent être lus relativement indépendamment du texte qui les encadre.

L'établissement partiellement fictif utilisé dans les exemples est désigné par l'acronyme CHM : le Centre Hospitalier de Marlitémon. Ainsi, quand nous traitons de rôles, nous évoquons des fonctions ou des professions de la santé, lorsqu'il s'agit de données, des dossiers patients sont pris comme exemples.

Le CHM est certes de taille modeste, environ 500 postes et 1.582 utilisateurs, mais les applications se multiplient : déjà plus de 12 reparties sur six serveurs, eux-mêmes sous des systèmes d'exploitation différents. Il va sans dire que ces applications sont variées, allant d'antiques terminaux du bureau de gestion des entrées, au tout nouvel intranet communautaire web 2.0 réparti sur les cinq sites du CHM.

Les membres de l'équipe informatique sont constamment occupés à résoudre des problèmes de droits, que ce soit pour accéder à une imprimante, accéder à un dossier administratif ou encore émettre un bon de commande. Il devient difficile de s'assurer qu'une modification des droits ne va pas à l'encontre de ce que l'on prévoit. Face à l'évolution des techniques et des besoins, les membres de l'équipe se retrouvent encombrés de soucis qu'ils souhaiteraient bien éviter.

(6) It is easier to move a problem around (for example, by moving the problem to a different part of the overall network architecture) than it is to solve it.
(6a) (corollary). It is always possible to add another level of indirection.

Ross Callon – RFC1925, "The Twelve Networking Truths"

2

Contrôle d'accès au système d'information

▷ *Ce chapitre commence par définir les notions de sécurité des systèmes d'information auxquelles s'intéresse la thèse, et notamment la place du contrôle d'accès dans le cadre général de la sécurité. Ensuite, nous étudierons les modèles de contrôle d'accès selon :*

les principes d'organisation des droits *: de nombreux modèles de contrôle d'accès ont été proposés, chacun introduisant de nouveaux principes et concepts pour faciliter la modélisation et l'administration des droits. Nous mettrons en évidence que ces modèles partagent des principes et des concepts communs,*

les enrichissements *: les principes fondateurs des modèles* RBAC *ayant largement été répandus, adaptés et repris dans des propositions de modèles contrôle d'accès, la littérature couvre désormais un très large spectre d'extensions des modèles de base dont nous décrivons les principaux enrichissements,*

les formalismes et modèles logiques proposés *: plusieurs cadres logiques ont été proposés pour formaliser les structures, les principes et les propriétés des modèles de contrôle d'accès. Nous comparerons les cadres existants selon leur expressivité et les applications aux problématiques du contrôle d'accès proposées.*

Le choix d'un modèle logique est guidé d'une part, par le besoin d'expressivité nécessaire pour capturer la richesse des modèles de contrôle d'accès et d'autre part par le besoin de décidabilité des structures théoriques. Ce chapitre se termine par une synthèse présentée à l'aide de deux tableaux croisés, évaluant d'une part les modèles de contrôle d'accès, et d'autre part les cadres logiques proposés pour leur étude.

L'annexe B synthétise les symboles utilisés pour les définitions de ce chapitre. ◁

Plan du chapitre

2.1 Place du contrôle d'accès dans la sécurité

La notion de système d'information couvre l'ensemble des éléments qui participe à la collecte, au stockage, à la gestion, au traitement et à la diffusion de l'information au sein des organisations. Cette définition recouvre à la fois des processus manuels et automatisés. Dans la thèse, nous nous intéressons principalement aux systèmes d'information *informatisés*, c'est-à-dire aux ressources matérielles et logicielles des systèmes.

2.1.1 Introduction à la sécurité des systèmes d'information

Les principes de la qualité incitent les concepteurs, développeurs et autres acteurs à prendre en compte les propriétés non fonctionnelles dans le développement des systèmes d'information [ISO91]. Une de ces propriétés est la *sécurité* : la propriété qui permet aux utilisateurs d'avoir une confiance justifiée dans le système et dans les services qu'il délivre. Cette notion dépasse largement le cadre des systèmes d'information informatisés, et couvre des domaines allant de la sécurité physique à la sécurité logique. La propriété de sécurité peut être analysée selon plusieurs sous-propriétés :

- la *confidentialité*, l'information ne doit être accessible qu'aux ayants droits,
- l'*intégrité*, l'information doit être cohérente et valide,
- la *disponibilité*, le système doit être accessible et prêt à l'emploi,
- la *fiabilité*, la continuité des services rendus doit être assurée,
- la *maintenabilité*, le système doit être corrigible et capable d'évoluer,
- la *sûreté* de fonctionnement[1], le système ne doit pas provoquer de dégâts.

Lorsqu'on traite de *sécurité l'information*, on s'attache plus précisément à considérer la sécurité comme la combinaison des trois propriétés de *confidentialité*, d'*intégrité* et de *disponibilité* [Rihaczek91]. D'autres facettes peuvent être intégrées à cette notion, en particulier pour les domaines d'application où la sécurité revêt un aspect important. C'est le cas du secteur de la santé et des affaires sociales [Abou El Kalam03]. On peut alors introduire des propriétés additionnelles comme l'*auditabilité* (les accès et les opérations doivent être enregistrés), l'*authenticité* (la propriété d'être vrai) ou la *non-répudiation* (l'impossibilité de nier être l'auteur d'un accès ou d'une opération).

De nombreuses pratiques et dispositifs participent à garantir les propriétés de sécurité dans les systèmes d'informations : la *protection des réseaux* (par exemple : pare-feux, détection d'intrusions), le *cryptage* de l'information, la *sauvegarde* des données, la *redondance* matérielle et logicielle ou encore les architectures d'*authentification*. Parmi les moyens mis en œvre pour renforcer la sécurité, nous allons nous intéresser aux politiques de sécurité.

[1] Il est notable que cette définition de la « sûreté de fonctionnement » est celle entendue par « safety » en anglais, c'est-à-dire la propriété de limiter les dégâts que le système puisse occasionner, ou de rendre cette occurrence la moins probable possible [Deswarte03].

Type	Description	*Exemples*
Physique	restrictions d'accès physiques aux ressources	*barrières* *coffres* *passes et clefs*
Administrative	règlements et procédures pour renforcer la sécurité	*enquêtes, audits* *reponsabilisation* *bonnes pratiques*
Logique	restrictions d'accès logiques aux ressources informatisées, mises en œuvre par des logiciels et matériels	*authentification* *identification* *cryptage* *cloisonnement* *organisation des droits*

TAB. 2.1 – Spécialisations des politiques de sécurité

2.1.2 Politique de sécurité

Une *politique de sécurité* est [Rihaczek91] :

> *. . . the set of laws, rules and practices that regulate how sensitive information and other resources are managed, protected and distributed within a specific system. It shall identify the security objectives of the system and the threats to the system.*

La mise en place d'une *politique de sécurité* est un dispositif organisationnel fondamental pour garantir la sécurité de l'information d'un système. Cette politique peut être spécialisée selon les propriétés auxquelles elle s'attache plus spécifiquement. On peut distinguer parmi ces spécialisations [Solomon05] (tableau 2.1) :

- les politiques de *sécurité physique* qui définissent des procédures et des moyens pour protéger les ressources des risques (électriques, incendies. . .) et des accès physiques aux matériels (postes de contrôle, barrières, blindages. . .),
- les politiques de *sécurité administrative* qui régissent la sécurité organisationnelle et les procédures en vigueur dans l'établissement,
- les politiques de *sécurité logique* qui font référence aux aspects informatiques et techniques des systèmes et définissent les actions légitimes qu'un utilisateur peut effectuer. La sécurité logique concerne deux aspects principaux :
  - l'identification et l'authentification : donner et prouver son identité,
  - l'*autorisation*, ou *contrôle d'accès* : la vérification de la légitimité des opérations demandées, selon une *politique de contrôle d'accès*, ou *politique d'autorisation*

Nos travaux s'intéressent en particulier aux politiques de sécurité logique et plus précisément à l'organisation des droits et au contrôle des accès aux systèmes. Nous allons donc définir ces aspects.

2.1.3 Politique de contrôle d'accès

Contrôler les accès, c'est déterminer si un *sujet* peut effectuer une *action* demandée sur un *objet*. Une *permission* est un droit d'action sur un objet, un *privilège* est l'affectation d'une permission à un sujet. Les principaux ensembles impliqués sont représentés par :

- l'ensemble $\mathcal{U}$ des *utilisateurs* connus du système. Les utilisateurs utilisent le système par l'intermédiaire d'un sujet,
- l'ensemble $\mathcal{S}$ des *sujets* connus du système. Les sujets peuvent être des processus, des machines : toute entité informatique qui représente l'utilisateur au sein du système et se comporte selon sa volonté,
- l'ensemble $\mathcal{O}$ des *objets* connus du système. Un objet est une entité du système comme un tuple, une table (cas des systèmes de gestion de bases de données), un objet ou un fichier (cas des systèmes de fichier),
- l'ensemble $\mathcal{A}$ des *actions* connues du système que l'on peut effectuer sur les objets. Les actions comprennent par exemple les opérations de sélection, suppression, modification et création de tuples dans un système de gestion de bases de données,

Définition (Contrôle d'accès [Lampson74])**.** *Le contrôle d'accès est un mécanisme grâce auquel un système autorise ou interdit les* actions *demandées par des* sujets *(entités actives) sur des* objets *(entités passives).*

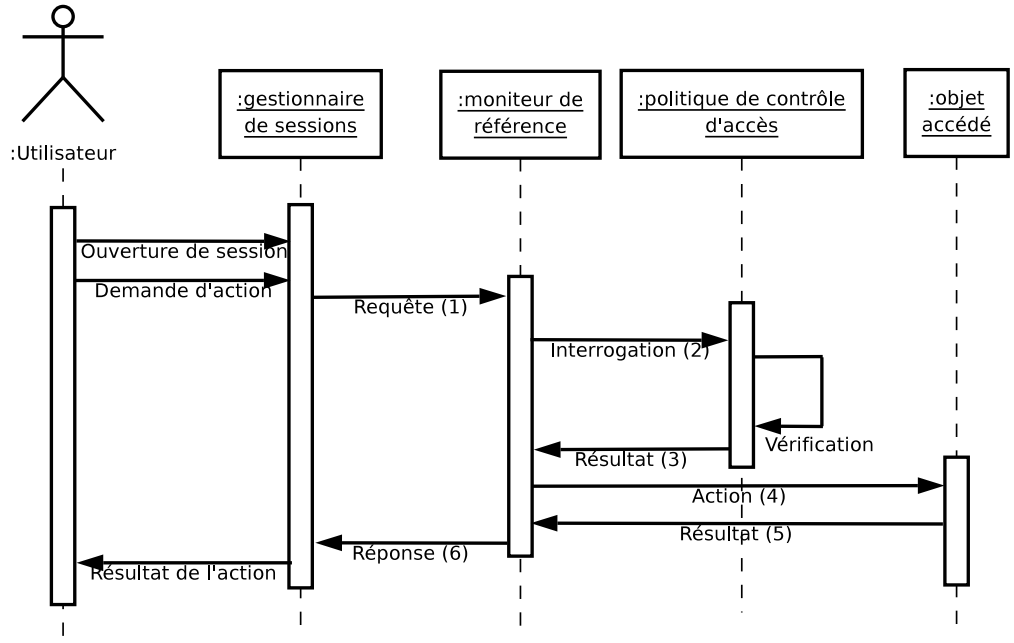


FIG. 2.1 – Mécanisme de moniteur mis en œuvre pour réaliser le contrôle d'accès.

Le contrôle d'accès renforce particulièrement la *confidentialité* et l'*intégrité* de l'information, *a fortiori* sa disponibilité. Le contrôle d'accès est généralement mis en œuvre par un *moniteur*, intermédiaire entre les utilisateurs et les ressources auxquelles ces derniers essaient d'accéder.

La définition d'un tel moniteur est délicate, car ce dernier doit être *incontournable*, *inviolable* et *vérifié*. Lorsqu'un utilisateur demande un accès, le moniteur va décider si cet accès est autorisé ou non d'après *la politique de contrôle d'accès* : une instance spécialisée de la politique de sécurité logique, qui s'attache à définir les droits des utilisateurs des systèmes. Le principe de fonctionnement d'un moniteur est décomposable en six[2] étapes, comme l'illustre la figure 2.1 :

1. envoi de la requête de l'utilisateur au moniteur,
2. interrogation de la politique de contrôle d'accès,
3. réponse de la politique,
4. le moteur accède à la ressource si l'accès est autorisé pour exécuter la requête,
5. retour de l'exécution de la requête,
6. retour de la requête ou exception en cas d'accès non autorisé.

La thèse s'intéresse aux politiques de contrôle d'accès logique, c'est-à-dire à la définition des droits des utilisateurs des systèmes. Nos travaux concernent à la fois l'*expression* des droits et leur *gestion* par les administrateurs de la sécurité. Les politiques de contrôle d'accès logique sont parmi les principaux dispositifs permettant de garantir la sécurité des systèmes d'information. Structurer les droits, c'est-à-dire exprimer pourquoi et comment un utilisateur acquiert les privilèges, est l'objet des *modèles de contrôle d'accès*, qui sont au centre de notre étude.

2.2 Modèles historiques de contrôle d'accès

Cette section présente les premières approches proposées sur le contrôle d'accès. Si elles sont conceptuellement simples, elles ont néanmoins bénéficié d'études approfondies et de nombreuses applications. Cette étude est importante, car :

- les approches historiques sont à la racine des propositions ultérieures,
- ces recherches ont abouti à des résultats majeurs,
- elles permettent d'introduire les principales notions du contrôle d'accès,
- les modèles historiques ont largement été implémentés.

[2] En cas d'accès non autorisé, les étapes 4 et 5 n'ont pas lieu.

2.2.1 Matrice de contrôle d'accès et modèles discrétionnaires

Les premières formalisations du contrôle d'accès sont dues à Lampson, qui parmi les premiers a défini les concepts de *sujet*, d'*objet* et d'*actions* [Lampson74]. Ces termes sont traditionnellement[3] repris dans la littérature sur le contrôle d'accès [Landwehr81]. Nous les utiliserons également tout au long de la thèse.

	Fichier1	Fichier2	Fichier3	Fichier4
Alice	rw	r	r	
Bob	r	rw	r	rwx
Charly	r	r	rw	rwx
Denise			r	r

TAB. 2.2 – Exemple jouet de matrice de contrôle d'accès

2.2.1.1 Principe

Le modèle de Lampson introduit dans ces premières études est la *matrice de contrôle d'accès* qui permet de représenter le *triplet d'autorisation* $\mathcal{ACCESS}$ entre sujets, actions et objets $\mathcal{ACCESS} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{O}$. Cette représentation est à la base de nombreux développements. Le tableau 2.2 est un exemple jouet de matrice de contrôle d'accès[4], impliquant quatre sujets (en lignes), quatre objets (en colonnes) et trois actions (`r` pour *read*, `w` pour *write* et `x` pour *execute*).

Le principal modèle de contrôle d'accès qui peut être exprimé grâce à cette matrice est le contrôle d'accès *discrétionnaire* (*Discretionnary Access Control* – DAC). Ce modèle est mis en œuvre dans de très nombreuses applications, comme les systèmes d'exploitation de type Unix/Linux. Un modèle discrétionnaire est défini comme [TCS85] :

> *...a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control).*

Il existe une ambiguïté sur l'adjectif *discrétionnaire*, qui peut être entendu :

– soit comme le fait que les droits sont organisés selon une *matrice* de contrôle d'accès, où l'on peut lire les permissions des utilisateurs en lignes, ou en colonnes,

[3] Quelquefois *sujet* et *utilisateur* sont confondus, comme *ressource* et *objet* ou *opération* et *action*.
[4] Cette matrice servira d'exemple tout au long de la thèse.

mais sans préciser si c'est une autorité qui définit les droits, où si ce sont les utilisateurs qui peuvent le faire. Pour cette interprétation, certains auteurs préfèrent traiter de contrôle d'accès basé sur *l'identité* (*Identity*-BAC) [Deswarte04],
– soit comme le fait que les utilisateurs peuvent *eux-mêmes* définir les droits d'accès sur les ressources dont ils sont *propriétaires*. Pour cette interprétation, les sujets sont eux-mêmes des objets et il existe une action supplémentaire `owner` dans la matrice de contrôle d'accès. Cette action permet au sujet qui en dispose de définir les opérations autorisées sur toute la colonne correspondante. C'est cette définition de *discrétionnaire* que nous utiliserons dans la suite.

2.2.1.2 Administration des droits

De nombreux systèmes (par exemple, des systèmes d'exploitation ou de gestion de bases de données) ont mis en œuvre le contrôle d'accès basé sur des matrices. Ces systèmes sont partitionables en deux grandes catégories, selon la méthode qu'ils proposent pour administrer les matrices de contrôle d'accès :

– manipuler les permissions par lignes : *Capabilities List* (CL),
– manipuler les permissions par colonnes : *Access Control List* (ACL).

Exemple 2.1 Gestion des matrices de contrôle d'accès par listes

Au CHM, un service d'annuaire *Lightweight Access Directory Protocol* (LDAP) a été mis en place par un valeureux stagiaire. Dans ce type d'annuaire, on peut définir quels sont les droits des utilisateurs sur des nœuds et des feuilles d'un arbre, que nous représentons par des fichiers d'un système de fichiers.

En se basant sur la matrice d'exemple du tableau 2.2, voici deux exemples de listes qui permettent de représenter les droits des usagers sur les fichiers 1,2 et 3 pour Alice et Bob. La syntaxe proposée est un pseudolangage simplifié, inspiré de ceux utilisés dans les annuaires :

Type *Access Control List*

```
GRANT r
        ON : Fichier1, Fichier2,
             Fichier3
        TO : Alice, Bob
GRANT w
        ON : Fichier1
        TO : Alice
        ON : Fichier2
        TO : Bob.
```

Type *Capabilities List*

```
SUBJECT Alice
        IS GRANTED r
        ON : Fichier1, Fichier2, Fichier3
        IS GRANTED w
        ON : Fichier1.
SUBJECT Bob
        IS GRANTED r
        ON : Fichier1, Fichier2, Fichier3
        IS GRANTED w
        ON : Fichier2.
```

La gestion des listes s'avère fastidieuse et sujette à erreurs en présence d'un grand nombre d'utilisateurs et de permissions : les listes peuvent se chevaucher et il n'existe pas de représentation unique d'une politique de contrôle d'accès.

Les travaux de Lampson ont été repris par Harrisson, Ruzzo et Ullman pour étudier l'administration des droits [Harrison76]. Les opérations administratives (par exemple, ajouter des sujets, des ressources, définir les actions autorisées) qui permettent de manipuler les matrices de contrôle d'accès sont étudiées et le *safety problem* est caractérisé.

Ce problème est le suivant : à partir d'une politique de contrôle d'accès reconnue comme sûre, est-il possible d'atteindre une politique non sûre, c'est-à-dire où les droits disposent d'une propriété interdite. Les travaux ont prouvé que ce problème est indécidable dans le cas général, en réduisant ce problème à celui du *problème de l'arrêt* d'une machine de Turing. Ce résultat limite fortement les ambitions de vérification des politiques de contrôle d'accès.

Dans la démonstration du résultat, chaque politique est représentée par une matrice de contrôle d'accès et des opérations administratives permettant de passer d'une politique à une autre sont définies à partir de six opérations primitives (ajouter et supprimer des colonnes, ajouter et supprimer des lignes, ajouter et supprimer des actions dans une cellule de la matrice). Par exemple : l'opération administrative `GrantREAD(S1,S2,O)` accorde au sujet `S2` le droit de lecture sur l'objet `O` si `S1` est propriétaire de `O`.

Les auteurs ont cependant montré que le *safety problem* est décidable si des restrictions fortes (par exemple, si l'on ne peut ajouter ni d'objets ni de sujets à la matrice) sont imposées sur le modèle de contrôle d'accès.

Théorème (Indécidabilité du safety problem [Harrison76])**.** *It is undecidable whether a given configuration of a given protection system is safe for a given generic right.*

2.2.1.3 Problèmes soulevés

Les modèles discrétionnaires ont une faiblesse importante : on ne peut plus contrôler ce qui est fait de l'information une fois que celle-ci a été accédée par un utilisateur légitime. Le problème majeur de la transitivité de la lecture qui en découle peut être une source de diffusion de chevaux de Troie.

Outre le problème de la transitivité du droit de lecture et de l'indécidabilité dans le cas général du *safety problem* pour les modèles DAC, une des critiques adressées aux modèles basés sur les matrices est qu'ils sont difficiles à administrer dans de grandes structures. Quand les organisations comportent de nombreux sujets et objets et lorsque des changements fréquents ont lieu dans les droits, il devient impossible de dégager une vision globale de l'organisation des droits et de garantir la sécurité du système. C'est pour répondre à ces problèmes que les modèles à niveaux sont apparus.

Exemple 2.2 Transitivité de la lecture des systèmes DAC

Au CHM, les mécanismes de partage de données des systèmes d'exploitation sont utilisés par toute l'administration. Il arrive des fois qu'un document confidentiel se retrouve mis en partage. Si ce dernier est lu par un utilisateur, alors il est difficile de contrôler l'usage qu'il en sera fait. Supposons l'état (*sûr*) suivant :

- La directrice des affaires financières, Mme. F., est propriétaire d'un rapport sur le budget prévisionnel de la cantine,
- Mme. F. donne le droit de lecture sur le rapport à M. S., pour qu'il le relise,
- Mme. F. ne donne le droit de lecture sur ce rapport à aucun autre sujet,

Ensuite, définissons un état comme *non sûr* si un autre sujet que M. S ou Mme. F. peut connaitre le contenu du rapport, en particulier le responsable de la restauration. Supposons la succession d'états suivant :

- M. S. fait une copie du rapport en l'enregistrant par mégarde dans le mauvais dossier partagé,
- par défaut, son éditeur de texte donne le droit de lecture à tout le monde.

On a atteind un état *non sûr* dans lequel le redouté responsable de la restauration peut prendre connaissance du contenu du rapport.

2.2.2 Modèles à niveaux et mandataires

2.2.2.1 Principe

Dans les modèles mandataires (*Mandatory Access Control* – MAC) les usagers ne peuvent pas définir les droits d'accès, car ils ne sont pas propriétaires des données : toutes les ressources informatiques sont la propriété *exclusive* de l'organisation.

Ces modèles sont basés sur la notion de *niveau de sécurité* : les objets et les sujets sont classés par confidentialité et un niveau leur est attribué. C'est à partir des niveaux de sécurité attribués au requérant et à la ressource démandée que sont dérivées les actions autorisées. Le tableau 2.3 présente les niveaux *Confidentiel-Défense*, *Secret-Défense*, *Très Secret-Défense* utilisés par le gouvernement français.

Définition (Contrôle d'accès mandataire [Sandhu93])**.** *Le contrôle d'accès mandataire est exprimé en termes de niveaux de sécurité associés aux sujets et aux objets et à partir desquels sont dérivés les actions autorisées.*

2.2.2.2 Flux de l'information dans les systèmes mandataires

À partir des niveaux, on définit des principes régissant la manière dont l'information est transmise dans le système, ce qui va imposer les droits de lecture et d'écriture des sujets. L'application de ces principes est en vigueur dans des systèmes militaires, où plus généralement dans tout système où les sujets ne peuvent (ou ne doivent) pas

Classification	Description
Très Secret-Défense	Le niveau *Très Secret-Défense* est réservé aux informations ou supports protégés dont la divulgation est de nature à nuire très gravement à la défense nationale et qui concernent les priorités gouvernementales en matière de défense.
Secret-Défense	Le niveau *Secret-Défense* est réservé aux informations ou supports protégés dont la divulgation est de nature à nuire gravement à la défense nationale.
Confidentiel-Défense	Le niveau *Confidentiel-Défense* est réservé aux informations ou supports protégés dont la divulgation est de nature à nuire à la défense nationale ou pourrait conduire à la découverte d'un secret de la défense nationale classifié au niveau *Très Secret-Défense* ou *Secret-Défense*.

TAB. 2.3 – Exemple de classification de confidentialité [Journal Officiel98]

se faire confiance. Les principes de contrôle du flux de l'information imposent qu'un sujet doive *recevoir* des informations provenant de sujets de plus *faible* accréditation, et *transmettre* des informations à des sujets de plus *haute* accréditation.

2.2.2.3 Administration

L'administration des modèles mandataires est uniquement basée sur l'*attribution* des niveaux de sécurité et sur les principes de transmission de l'information : c'est le système qui, à partir de deux règles de contrôle du flux de l'information, détermine les opérations autorisées en fonction de l'accréditation du requérant et de la classification de l'objet. Les règles présentées dans le tableau 2.4 sont celles du modèle de Bell et LaPadula [Bell75], orienté confidentialité. Le modèle de Biba et Denning [Denning76] est quant à lui orienté intégrité des données et propose deux règles duales de celles de Bell et LaPadula où écriture et lecture sont inversées.

2.2.2.4 Problèmes soulevés

Les premiers modèles DAC ont proposé d'organiser les niveaux selon un ordre strict [Bell75, Denning76]. Cette contrainte a été relaxée par les modèles où les niveaux sont organisés selon des treillis [Sandhu93]. Cette hiérarchisation des niveaux accorde une plus grande liberté de modélisation aux administrateurs du contrôle d'accès.

Propriété	Nom commun	Description
Simple security rule	*No read up*	Un sujet accrédité d'un niveau donné ne peut pas accéder en *lecture* à des objets d'un niveau plus élevé
⋆-property	*No write down*	Un sujet accrédité d'un niveau donné ne peut pas accéder en *écriture* à des objets d'un niveau moins élevé

TAB. 2.4 – Principes du modèle de Bell et LaPadula [Bell75]

Les modèles DAC sont bien adaptés aux systèmes à haute confidentialité et fortement structurés – les systèmes militaires par exemple – ou plus généralement pour tout *environnement clos et contrôlé* – comme les systèmes bancaires. Cependant, pour de grandes organisations ils s'avèrent trop rigides, car il est difficile de classer objets et sujets dans un nombre prédéfini de niveaux de sécurité.

Les systèmes de gestion de base de données ayant utilisé les modèles mandataires n'ont eu que peu de succès commercial, vraisemblablement à cause de leur rigidité et de la hiérarchisation stricte qu'ils imposent aux utilisateurs et aux objets. Une formalisation logique à l'aide de règles DATALOG des principes des modèles DAC appliqués aux bases de données a été proposée [Cuppens99, Cuppens00].

2.2.3 Modèles basés sur les vues

2.2.3.1 Principes

Dans les systèmes de gestion de bases de données relationnelles, le niveau le plus proche de l'utilisateur est composé de *vues* : des représentations construites sur le modèle logique de données sous-jacent. Une vue dans une base de données est assimilable à une requête conjonctive sur la base à laquelle on a donné un nom : elle peut être soit *virtuelle* – elle est calculée sous forme de réécriture de requêtes – soit *matérialisée* – elle est stockée puis mise à jour. Le concept de vue est utilisé entre autres pour exprimer les politiques d'autorisation dans les systèmes de gestion de base de données en limitant l'accès au modèle logique. Une vue sur une base de données **R** s'exprime en logique par une formule close de la forme [Abiteboul95] :

$$ans(e_1, \ldots, e_m) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_n) \wedge \phi(u)$$

Dans cette expression, les R_i sont des noms de relations de **R**, *ans* est un nom de relation ne faisant pas partie de **R**, les u_i sont des tuples (utilisant soit des variables soit des constantes) et $\phi(u)$ est une conjonction de contraintes arithmétiques (prédicats *built-in*) sur les termes u_i. La relation *ans* est donc une relation définie en *intention*.

Lorsqu'elles sont utilisées à des fins de contrôle d'accès, les vues permettent de masquer certaines données à l'utilisateur soit :

- en interdisant l'accès à des *attributs* d'une relation, pour que les utilisateurs n'aient pas connaissance de tout une colonne d'une table, en utilisant la projection par exemple,
- en interdisant l'accès à des *tuples* d'une relation, pour que les utilisateurs n'aient pas connaissance de certaines lignes d'une table, en utilisant par exemple la sélection.

2.2.3.2 Administration

Les vues sont, du point de vue du contrôle d'accès, des objets abstraits qui masquent les objets concrets sous-jacents. L'administration des vues se base sur les primitives définies dans le système de gestion de base de données. Les droits sur les vues sont ainsi manipulés comme des droits sur des relations classiques. La grammaire du langage d'administration du standard *Structured Query Language* (SQL) est [ISO99] :

```
GRANT <action> ON <objet>
TO <sujet>
[WITH GRANT OPTION]

<action> ::= ALL |      SELECT | DELETE |
        INSERT [<attribut>]| UPDATE [<attribut>]|
        REFERENCE [<attribut>]
<objet> ::= TABLE <relation> * | VIEW <vue> *
<sujet> ::= PUBLIC| <sujet> *
```

2.2.3.3 Problèmes soulevés

Comme l'illustre l'exemple « utilisation du concept de vue pour contrôler les accès », les modèles basés sur les vues posent les problèmes de :

- la *cohérence* des droits. En reprenant l'exemple du personnel externe, supposons qu'un administrateur autorise l'accès en lecture à la table *Personnel* directement à tout utilisateur. Le règlement stipulant que l'adresse des employés ne doit pas être accessible est contourné,
- la *maintenance* des multiples contrôles d'accès implémentés par l'intermédiaire des vues. Dans l'exemple proposé, pour chaque service de l'organisation il devient nécessaire de définir une nouvelle vue, avec le risque d'explosion combinatoire du nombre de vues et de droits accordés sur ces vues.

Exemple 2.3 Utilisation du concept de vue pour contrôler les accès

Au CHM, une application de gestion du personnel est utilisée. Cette application est principalement une interface graphique pour accéder à des données gérées avec un système de gestion de bases de données. Le schéma de base de données manipulé par cette application est composé de trois relations :

- $Personnel(ID, Nom, Prenom, Adresse)$, les employés,
- $Service(ID, Description, Lieu)$, les services médicaux,
- $Affectation(IDPersonne, IDService)$, les affectations des personnes.

La vue $PersonnelExterne(Nom, Prenom, Lieu)$ permet de restreindre les accès au seuls champs nom et prénom des personnes du bureau externe, sans avoir accès à leurs adresses personnelles, elle est exprimable en logique par :

$$Personnel(IDP, N, P, A) \wedge Service(IDS, D, L) \wedge$$
$$Affectation(IDP, IDS) \wedge D = \texttt{bureauExterne}$$
$$\rightarrow PersonnelExterne(N, P, L)$$

2.2.4 Synthèse des modèles historiques

Les organisations des droits historiques présentées dans cette section ont introduit les principes génériques du contrôle d'accès. Le souci d'imposer des règles strictes à conduit à introduire la notion de niveaux d'accès : des intermédiaires organisés entre sujets et permissions qui n'existent pas dans les approches matricielles. Nous verrons que ce principe générique de structuration des droits a donné naissance à de grandes familles de modèles.

2.3 Structuration du contrôle d'accès

Après l'organisation hiérarchique stricte des modèles à niveaux, de nouvelles organisations des droits ont été proposées à partir des années 90 : les modèles à rôles (*Role-Based Access Control* – RBAC). Le principe général de ces modèles est d'introduire un niveau d'indirection entre utilisateurs et permissions, en faisant en sorte que cette nouvelle abstraction permette [Ferraiolo03b] :

- d'organiser les droits de la façon la plus proche possible de la structure des organisations, afin de permettre aux administrateurs de manipuler les droits d'une façon plus intuitive,
- de limiter le nombre d'affectations des permissions, pour éviter les erreurs en réduisant la taille des politiques,
- d'exprimer de nouvelles règles et contraintes, qui permettront de traduire facilement les politiques exprimées en langue naturelle.

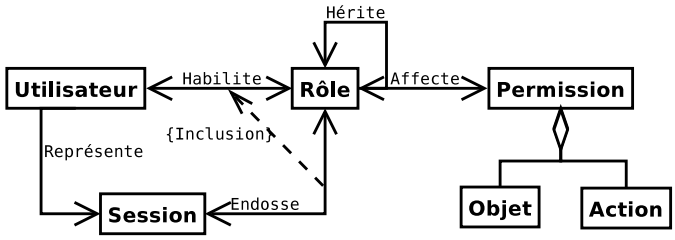


FIG. 2.2 – Représentation du modèle RBAC_1 (UML)

2.3.1 Concept de rôle pour structurer les droits

2.3.1.1 Noyau du modèle

Le concept central de *rôle* dans les modèles RBAC est défini comme [Sandhu96] :

> *. . . a job function or job title within the organization with some associated semantics regarding the authority and responsibility conferred on a member of the role.*

La principale motivation de l'introduction de la notion de rôle dans l'organisation des droits est la constatation qu'une grande partie des décisions de contrôle d'accès est déterminée par l'*autorité hiérarchique* et la *fonction* des utilisateurs.

Utiliser le rôle comme intermédiaire entre les sujets et les permissions facilite et simplifie les tâches d'administration en diminuant le nombre d'affectations à manipuler. Ce modèle est largement adopté par les entreprises et les industriels [Ferraiolo01] et a été appliqué dans de grandes structures [Roeckle00, SAIC04]. Les logiciels commerciaux Trusted Solaris, Windows Authorization Manager, Oracle 9 et Sybase Adaptive Server ont mis en œuvre tout ou partie des principes des modèles RBAC.

La notion de rôle dans les modèles RBAC diffère de la notion de groupe : un groupe est essentiellement une collection d'utilisateurs alors que le rôle est *à la fois* une collection d'utilisateurs *et* de permissions [Sandhu96]. C'est donc un concept qui peut être défini soit par son intention – les permissions dont dispose le rôle – soit par son extension – les utilisateurs qui l'endossent.

Dans les modèles basés sur les rôles, l'affectation directe de permissions aux utilisateurs est *interdite* : tous les privilèges sont attribués par l'intermédiaire des rôles. À la différence des modèles mandataires, il n'y a pas de règles (comme celles du tableau 2.4) régissant les actions autorisées des sujets sur les objets : ce sont les administrateurs qui définissent les permissions.

Définition (Session dans les modèles RBAC [Ferraiolo03b])**.** *Une* session *est l'entité qui représente un utilisateur* actif *au sein du système. Une session donnée est attribuée à* un et un seul *utilisateur. Dans une session, un utilisateur choisit d'endosser tout ou partie des rôles qui lui sont attribués.*

Dans les modèles RBAC, les *sujets* sont les *sessions* qui représentent les utilisateurs actifs dans le système. Cette distinction n'était pas toujours faite dans les modèles de contrôle d'accès basés sur l'identité, ce qui amenait parfois à confondre utilisateur et sujet. Dans la famille des modèles RBAC, on distingue donc deux triplets $\mathcal{U} \times \mathcal{A} \times \mathcal{O}$ pour les autorisations des utilisateurs, selon que l'on s'intérèsse ou non aux sessions :

- $\mathcal{STATIC}$: les autorisations *statiques*, valables dans *tout* état du système et indépendantes des sessions,
- $\mathcal{DYNAMIC}$: les autorisations *dynamiques*, valables dans une état *donné* du système et dépendantes des sessions en cours,

Ainsi, lorsqu'un aspect des modèles RBAC fait référence aux sessions, on traite d'aspect dynamique et statique sinon. Par exemple, dans le tableau 2.5, les relations $\mathcal{SU}$ et $\mathcal{SR}$ sont qualifiées de dynamiques alors que $\mathcal{URA}$ et $\mathcal{PRA}$ sont statiques (tableau 2.5). On peut également construire le triplet de Lampson $\mathcal{ACCESS} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{O}$ des autorisations associées à chaque session.

La différence entre aspects statiques et dynamiques n'est pas représentable formellement : ni dans la formalisation ensembliste proposée dans le standard, ni dans les approches logiques que nous présenterons à la fin de ce chapitre. En revanche, cette distinction est importante pour l'implémentation et la structuration des modèles de contrôle d'accès.

2.3.1.2 Famille des modèles à rôles

Comme plusieurs modèles à rôles ont été proposés, on traite d'une *famille* RBAC composée de quatre modèles [Sandhu96] (figure 2.3) :

- le modèle RBAC_0, qui présente les concepts et relations de base – le *noyau* – du modèle,
- le modèle RBAC_1, qui reprend le modèle RBAC_0 et introduit la notion de *hiérarchie*, détaillée en section 2.3.2,
- le modèle RBAC_2, qui reprend le modèle RBAC_0 et introduit la notion de *contrainte*, détaillée en section 2.5.1,
- le modèle RBAC_3, qui reprend les modèles RBAC_1 et RBAC_2 et prend en compte les intéractions entre contraintes et hiérarchie.

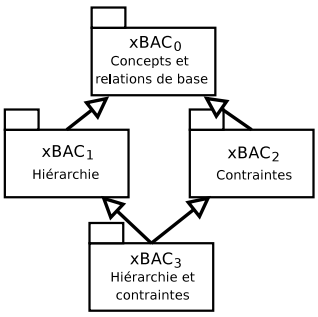


FIG. 2.3 – Famille *x*-BAC (UML)

Ces raffinements successifs illustrent une orientation générale des recherches sur les modèles de contrôle d'accès : à partir d'un noyau, introduisant les concepts et relations principales du modèle, des enrichissements supplémentaires sont proposés. Cette structuration de la famille des modèles RBAC a été reprise par exemple dans les mo-

dèles TBAC [Thomas97b] et GEORBAC [Damiani07], c'est la raison pour laquelle nous avons utilisé le terme *x*-BAC dans la figure 2.3. Le tableau 2.5 synthétise les concepts et relations introduits dans le modèle de contrôle d'accès $RBAC_0$, qui sont à la base de la formalisation ensembliste de la famille des modèles RBAC.

	Notation	Description
Concepts	$\mathcal{U}$	ensemble fini d'utilisateurs
	$\mathcal{R}$	ensemble fini de rôles
	$\mathcal{A}$	ensemble fini d'actions
	$\mathcal{O}$	ensemble fini d'objets
	$\mathcal{S}$	ensemble fini de sujets (sessions)
Relations	$\mathcal{P} \subseteq \mathcal{O} \times \mathcal{A}$	une permission est une action sur un objet
	$\mathcal{URA} \subseteq \mathcal{U} \times \mathcal{R}$	affectation many-to-many de rôles aux utilisateurs
	$\mathcal{PRA} \subseteq \mathcal{R} \times \mathcal{P}$	affectation many-to-many de permissions aux rôles
	$\mathcal{SU} \subseteq \mathcal{S} \times \mathcal{U}$	relation many-to-one entre sessions et utilisateurs
	$\mathcal{SR} \subseteq \mathcal{S} \times \mathcal{R}$	relation many-to-many entre sessions et rôles

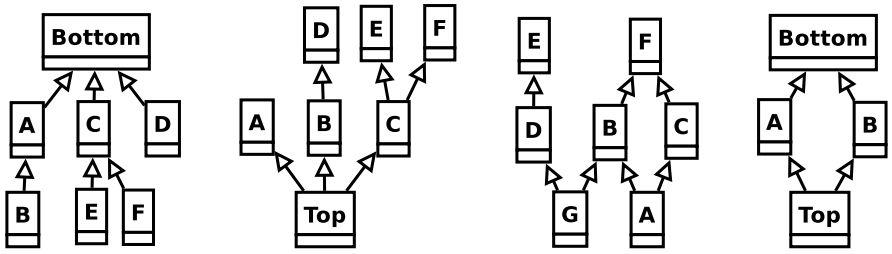
TAB. 2.5 – Concepts et relations du noyau RBAC [Ferraiolo03b]

2.3.2 Hiérarchisation des rôles

La hiérarchie des rôles est une des innovations proposées dans les modèles RBAC. Elle permet de réduire le nombre de rôles et d'affectations (entre rôles et utilisateurs ou entre rôles et permissions) en introduisant une notion d'héritage entre rôles : un rôle hérite des permissions accordées à ses parents. On peut interpréter héritage comme une relation *est un* : un *a est un b* quand le rôle *a* hérite du rôle *b*. La hiérarchisation limite la redondance des affectations simplifiant ainsi son administration.

La notion de hiérarchie de rôles rappelle la hiérarchisation des niveaux de sécurité dans les systèmes MAC : dans les deux approches, on munit les concepts intermédiaires entre sujets et objets d'une relation binaire interne de domination. Selon les propriétés algébriques que la relation de domination respecte, différents types de hiérarchies ont été identifiés, en particulier une distinction est faite entre [Ferraiolo03b] :

- la hiérarchie *limitée* : les rôles munis de la relation de domination forment :
  - un *arbre*, où chaque rôle, sauf le rôle le moins privilégié noté $\perp$, *est dominé* par un seul rôle. Cette organisation implique l'existence d'un rôle racine $\perp$, dont tous les utilisateurs disposent implicitement, figure 2.4 (a),
  - un *arbre inversé*, où chaque rôle, sauf le rôle le plus privilégié noté $\top$, *domine* un rôle. Cette organisation en arbre inversé implique l'existence d'un rôle *root* $\top$, qui domine tous les autres rôles, figure 2.4 (b)
- la hiérarchie *générale*, où la relation de domination est transitive et réflexive, elle forme alors un *préordre* [Gavrila98, Barker03], si la relation de domination est



(a) Arbre (b) Arbre inversé (c) Ordre partiel (d) Treillis

FIG. 2.4 – Exemples de hiérarchies (UML)

aussi antisymétrique, c'est un *ordre partiel (large)* [Kuhn97] : c'est le modèle le généralement admis, figure 2.4 (c),
– le treillis : pour simuler une politique MAC à l'aide de rôles [Osborn00], on peut imposer que la hiérarchie soit un treillis, figure 2.4 (d).

D'aucuns ont proposé une relation de domination irréflexive et transitive [Jansen98] qui forme un *ordre partiel strict*. Cette différence n'est pas gênante car les ordres partiels larges et stricts sont en correspondance un à un : à tout ordre strict on peut faire correspondre un ordre large associé (sa fermeture réflexive) et réciproquement (sa réduction réflexive). Nous reprenons les définitions de [Ferraiolo03b], corrigées dans [Li07] [5].

Définition (Hiérarchie générale de rôles [Ferraiolo03b])**.** *$\mathcal{RH} \subseteq \mathcal{R} \times \mathcal{R}$ est un ordre partiel appelé* relation d'héritage*, noté $\succeq$, où $r_1 \succeq r_2$. Toutes les permissions accordées à r_2 sont accordées à r_1 et tous les utilisateurs membres de r_1 sont membres de r_2 :*

$$r_1 \succeq r_2 \quad \Rightarrow \quad auth\_perms(r_2) \subseteq auth\_perms(r_1) \wedge \\ auth\_users(r_1) \subseteq auth\_users(r_2)$$

Définition (Hiérarchie limitée de rôles [Ferraiolo03b])**.** *Soit $\prec\!\!\prec$ la* relation de domination *entre rôles : $r_1 \prec\!\!\prec r_2$. Si $r_1 \preceq r_2$ et il n'existe pas r_3 tel que $r_1 \preceq r_3 \preceq r_2$ où $r_1 \neq r_3$ et $r_2 \neq r_3$. Une hiérarchie de rôles est dite limitée :*

– *en arbre si :*
$$\forall r, r_1, r_2 \; r \succ\!\!\succ r_1 \wedge r \succ\!\!\succ r_2 \Rightarrow r_1 = r_2$$
– *en arbre inverse si :*
$$\forall r, r_1, r_2 \; r \prec\!\!\prec r_1 \wedge r \prec\!\!\prec r_2 \Rightarrow r_1 = r_2$$

[5] Une réponse a cette critique est publiée dans la même revue. Ce débat conduira vraisemblablement à une révision du standard [Ferraiolo07].

L'introduction de hiérarchies de rôles conduit à des affectations implicites. En effet, quand a hérite de b et qu'un utilisateur est *explicitement* affecté au rôle a, cet utilisateur est *implicitement* affecté au rôle b. Cette nuance entre affectations explicite et implicite est représentée par les applications *assigned_users* et *assigned_perms* (affectés) ainsi que par *auth_users* et *auth_perms* (autorisés) [Ferraiolo03b]. Elles illustrent la différence entre les affectations déterminées par les administrateurs de celles *dérivables* par les propriétés algébriques des hiérarchies de rôles.

Définition (Utilisateurs affectés et autorisés)**.** *L'application assigned_users* : $\mathcal{R} \rightarrow 2^{\mathcal{U}}$, *fait correspondre à un rôle l'ensemble des utilisateurs qui lui sont explicitement affectés. L'application auth_users :* $\mathcal{R} \rightarrow 2^{\mathcal{U}}$ *fait correspondre à un rôle l'ensemble des utilisateurs autorisés à l'endosser :*

$$\begin{aligned} assigned\_users(r) &= \{u \in \mathcal{U} \mid (u,r) \in \mathcal{URA}\} \\ auth\_users(r) &= \{u \in \mathcal{U} \mid r' \succeq r \wedge (u,r') \in \mathcal{URA}\} \\ \forall r \in \mathcal{R}\ & assigned\_users(r) \subseteq auth\_users(r) \end{aligned}$$

Définition (Permissions affectées et autorisées)**.** *L'application assigned_perms :* $\mathcal{R} \rightarrow 2^{\mathcal{P}}$, *fait correspondre à un rôle l'ensemble des permissions qui lui sont explicitement affectées. L'application auth_perms :* $\mathcal{R} \rightarrow 2^{\mathcal{P}}$ *fait correspondre à un rôle l'ensemble des permissions autorisées :*

$$\begin{aligned} assigned\_perms(r) &= \{p \in \mathcal{P} \mid (r,p) \in \mathcal{PRA}\} \\ auth\_perms(r) &= \{p \in \mathcal{P} \mid r' \preceq r \wedge (r',p) \in \mathcal{PRA}\} \\ \forall r \in \mathcal{R}\ & assigned\_perms(r) \subseteq auth\_perms(r) \end{aligned}$$

Après l'introduction de la hiérarchie dans les modèles RBAC, plusieurs propositions ont repris cette notion dans de nouveaux modèles de contrôle d'accès. Par exemple, pour hiérarchiser les objets [Barker03], les contextes [Covington01], les zones géographiques [Damiani07] ou les différents concepts du modèle ORBAC [Miège05]. La section suivante va présenter d'autres concepts intermédiaires que les rôles pour structurer les droits d'accès.

2.3.3 Autres concepts pour la structuration des droits

Dans certaines applications, la seule notion de rôle ne permet pas de refléter fidèlement les structures des organisations. Des auteurs ont alors proposé d'intégrer dans les modèles de contrôle d'accès d'autres concepts, soit en ajoutant des concepts supplémentaires à ceux des modèles RBAC, soit en les remplaçant. Cette section va présenter les principaux concepts intermédiaires introduits entre utilisateurs et permissions.

2.3.3.1 Concept d'organisation

Le modèle *Organization-Based Access Control* (ORBAC) est construit autour de la notion centrale d'*organisation* [Abou El Kalam03, Cuppens04a, Cuppens04b, Miège05]. ORBAC rassemble plusieurs concepts introduits dans différents modèles (rôle, vue, équipe, ...) et les relie par l'intermédiaire de la notion d'organisation. Les principes de hiérarchies [Cuppens04a], de contraintes [Cuppens03, Miège05] et de modèle d'administration [Cuppens04b] ont également été repris dans ORBAC.

La figure 2.5, simplifiée de [Abou El Kalam03], illustre la structuration proposée du modèle ORBAC. Cette illustration est divisée en trois parties :

- la partie *supérieure* de ce schéma est la politique *abstraite*, composée des concepts de rôle, de vue, d'activité et de contexte, communs à l'ensemble des organisations. La relation entre ces concepts génériques permet de définir des permissions *abstraites*,
- la partie *inférieure* est la politique *concrète*, composée des concepts de sujet, action et objet propres à chaque organisation et desquels sont dérivées les permissions *concrètes*.
- la partie *intermédiaire* joue le rôle de médiateur entre les parties supérieures et inférieures, elle va *instancier* la politique abstraite en une politique concrète, en faisant correspondre aux rôles, vues et activités abstraits des utilisateurs, objets et actions concrets du système.

Le concept d'organisation est ainsi un élément pivot entre les politiques abstraites – consensuelles entre les organisations – et les concrètes – spécifiques à chaque organisation. En effet, le modèle ORBAC est structuré grâce à trois relations *ternaires* : *Habilitation*, *Considération*, *Utilisation*, toutes liées au concept d'organisation. Cette particularité permet d'exprimer le fait qu'une personne puisse disposer de rôles différents dans des organisations différentes, propriété qui n'est pas exprimable dans d'autres modèles. Par exemple, le modèle RBAC est uniquement composé de relations *binaires* entre les concepts du modèle.

2.3.3.2 Concept d'équipe

La notion d'*équipe* a été proposée dans le modèle TMAC (*TeaM-based Access Control*), où les permissions sont associées aux rôles ainsi qu'aux équipes. Les privilèges dont disposent les utilisateurs sont l'union des permissions accordées aux rôles qu'endossent les utilisateurs et des permissions accordées aux équipes dont les utilisateurs font partie [Thomas97a].

La notion d'équipe a été introduite pour représenter des aspects transversaux des rôles qui ne sont pas directement exprimables dans les modèles RBAC. Le principe de TMAC est d'accorder à chaque utilisateur membre d'une équipe l'union des permissions accordées aux autres membres de l'équipe *actuellement actifs*, en plus des permis-

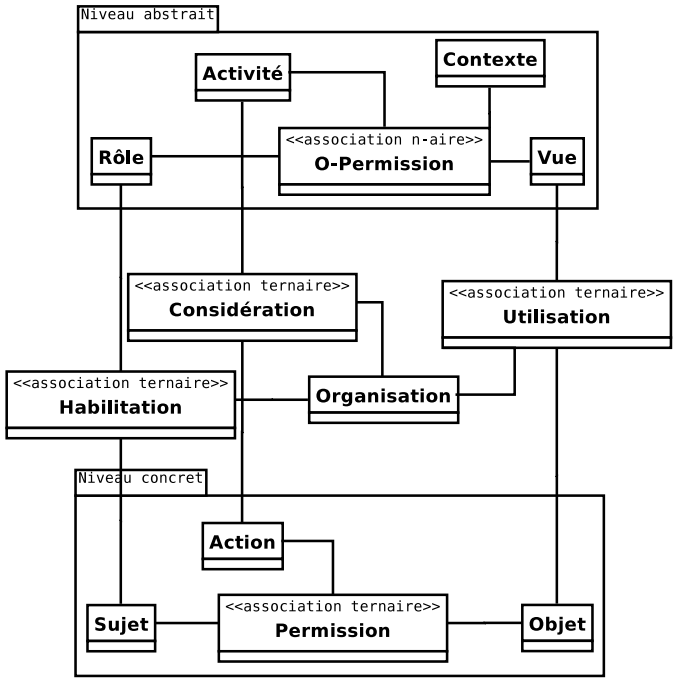


FIG. 2.5 – Représentation du modèle ORBAC (UML)

sions accordées par les rôles qu'il endosse. Les relations de ce modèle sont dynamiques, car les permissions sont accordées par l'intermédiaire des sessions *actives* des membres d'une équipe. Il s'agit d'un modèle par essence dynamique, dans lequel les droits effectifs sont évaluables seulement lors des requêtes d'accès en non une fois pour toutes.

2.3.3.3 Concept de tâche

Les modèles à niveaux sont dits « orientés flux » : l'attribution des droits est basée sur l'autorité des acteurs et les règles régissant la transmission de l'information à des acteurs de niveaux différents. Plusieurs approches ont pris en compte la notion de flux de l'information pour organiser les droits.

La famille des modèles TBAC (*Task-Based Access Control*) [Thomas97b] est organisée en quatre modèles, comme la famille RBAC (figure 2.3). La famille des modèles TBAC propose de structurer les droits selon les *tâches* que les acteurs du système d'information doivent effectuer, mais sans concept de rôle. Les autorisations sont ainsi vérifiées au fil de l'exécution des activités qui sont décomposées en tâches.

La famille des modèles WRBAC (*Workflow Role-Based Access Control*) [Wainer03, Wainer07] se base quant à elle sur la notion de rôle, mais raffine le modèle RBAC en y introduisant les concepts :

Exemple 2.4 L'équipe comme aspect transversal des rôles

Le CHM se retrouve établissement témoin pour mettre en œuvre un nouvel équipement mobile. Certains utilisateurs se retrouvent équipés d'assistants personnels. Le dispositif est censé permettre un meilleur suivi du patient. Le logiciel déployé utilise le modèle RBAC, et trois rôles ont été définis :

- le rôle `Infirmier`,
- le rôle `Médecin`,
- le rôle `Secrétaire`.

Chaque rôle dispose d'un ensemble de permissions permettant de réaliser les tâches dont il a la charge. Le CHM est structuré en équipes de soins, chacune étant composée d'un ou plusieurs infirmiers, médecins et secrétaires. L'une d'entre elles est l'équipe pilote qui évalue la mise en œuvre de TMAC.

Le modèle RBAC ne permet pas à un secrétaire ou un infirmier de saisir des diagnostics. Cependant, dans certaines situations (urgence par exemple), *en présence* du médecin responsable de son équipe, un infirmier devrait pouvoir saisir des diagnostics. Le modèle TMAC permet d'exprimer cette attribution dynamique de permission.

- d'une unité organisationnelle (*organizational unit*), c'est un groupe d'utilisateurs dont un est désigné comme étant le responsable de l'unité.
- de cas d'utilisation (*case*), c'est une relation entre privilèges et utilisateurs qui permet de modéliser les activités métiers. Elle représente les flux dans le système.

Ces familles de modèles ont été proposées pour contrôler l'exécution des activités dans les processus métiers, ces derniers étant découpés en tâches élémentaires qui sont composées pour former des activités complexes. Ainsi, le cas d'utilisation de « la réservation d'un voyage » va être composé de l'achat d'un billet d'aller, la réservation d'un hôtel, la location d'un véhicule sur place, l'achat de billets de spectacle, etc.

Les auteurs de [Bertino99] ont organisé leur proposition de contrôle d'accès autour de la notion de processus métier. Leur motivation est de combler une lacune des modèles RBAC : il est difficile avec des rôles de spécifier que des tâches doivent être effectuées par des utilisateurs différents. Les modèles orientés flux visent principalement à renforcer le principe *séparation des tâches* (S*eparation of Duties* – SOD). Ce principe a été introduit dans [Clark87], approfondi dans [Nash90] puis repris dans différents modèles de contrôle d'accès [Sandhu96, Bertino99, Thomas97b, Abou El Kalam03].

Définition (Principe de séparation des tâches [Nash90])**.** *La* séparation des tâches *est un principe de sécurité qui impose que les acteurs qui interviennent dans la réalisation d'une tâche soient différents.*

Dans le cas général, la séparation des tâches s'exprime par une règle de la forme « il faut qu'au moins n utilisateurs différents interviennent dans ce processus métier », comme à chaque étape du processus métier correspondent des tâches qui nécessitent des permissions, on peut exprimer la séparation des tâches comme une contrainte de la

forme « il ne peut pas y avoir $(n-1)$ utilisateurs qui disposent à eux seuls de l'ensemble des permissions pour effectuer l'ensemble du processus » [Nash90].

Le besoin d'exprimer la séparation des tâches a été pris en compte dans les modèles contrôle d'accès. Il a reçu une attention particulière dans les modèles RBAC. Les notions de *contraintes* introduites dans le modèle RBAC_2, et plus précisément la notion d'*exclusion mutuelle*, ont été développées à cet effet. Ces contraintes permettent de limiter les privilèges des utilisateurs et augmentent le pouvoir d'expression des modèles RBAC.

Exemple 2.5 Exemple de séparation des tâches

Au CHM, les commandes de produits pharmaceutiques sont quotidiennes. Une application permet heureusement de traiter ces commandes, en les groupant automatiquement par fournisseurs, dates, etc. Les étapes identifiées pour exercer l'activité sont :

1. passer une commande de produits et enregistrer le bon de commande,
2. réceptionner la facture et vérifier qu'elle correspond à la commande,
3. réceptionner les produits et vérifier qu'ils correspondent à la facture,
4. autoriser le paiement sur facture.

La politique de sécurité pourrait[a] imposer qu'aucun utilisateur ne puisse commander et autoriser le paiement de sa propre commande afin d'éviter des abus de pouvoir, des détournement d'argent ou encore des contrats de complaisance. Il s'agit d'un exemple de contrainte de *séparation des tâches*.

[a] UNE POLITIQUE SÉVÈRE DEVRAIT MÊME IMPOSER QUE CE SOIT LE CAS.

2.3.4 Synthèse de la structuration des droits d'accès

De nombreux concepts ont été introduits pour structurer les droits d'accès selon l'activité et le métier des utilisateurs du système. Ces notions sont toutes des *intermédiaires* entre les utilisateurs et les permissions du système. Ainsi, selon les spécialités des organisations, il peut être intéressant d'introduire la notion de flux, d'équipe ou de rôle pour représenter le plus fidèlement possible les interactions et les responsabilités des acteurs. Selon les activités, l'utilisation de hiérarchies multiples peut être un choix de modélisation pertinent.

Il est donc difficile de considérer qu'un modèle de contrôle d'accès est meilleur qu'un autre : cela dépend essentiellement du domaine d'application et du type de l'organisation qui le met en œuvre. Par exemple, les modèles à niveaux sont bien adaptés aux structures très organisées, où la confidentialité est une priorité. C'est pourquoi les organisations militaires et banquaires les utilisent encore.

D'un autre côté, certaines structures ont des organisations beaucoup plus souples et complexes. Elles souhaitent alors définir leurs propres modèles de contrôle d'accès adaptés à leur besoins. Pour les établissements de santé par exemple, le Groupement pour la Modernisation du Système d'Information Hospitalier (GMSIH) a proposé un modèle de contrôle d'accès sur mesure, adapté aux métiers de la santé. Ce modèle réutilise les concepts présentés dans cette section (rôle, organisation, hiérarchies) et en introduit de nouveaux [GMSIH03].

Les propositions basées sur les rôles forment la plus grande famille des modèles de contrôle d'accès et sont les plus étudiées du domaine. Un standard des modèles RBAC a été défini [Ferraiolo01, Ferraiolo03b]. Il formalise les principes des modèles RBAC et spécifie les primitives nécessaires à leur implantation. Or d'une part, ce standard souffre encore de lacunes [Li07], et d'autre part la mise en œuvre de RBAC dans les systèmes ne respecte pas toujours les standards. Des extensions, des adaptations ou même des modifications structurelles étant introduites (en particulier dans les grands systèmes comme Solaris ou Oracle).

2.4 Administration des modèles structurés

Dans les modèles structurés, les concepts intermédiaires entre utilisateurs et permissions ont une place centrale dans l'administration des droits :

- ils représentent les *responsabilités* ou les *activités* des utilisateurs,
- leurs hiérarchisations permettent de représenter la *structure* de l'organisation,
- ils sont *obligatoires* : il n'y a pas d'affectation directe de permissions aux utilisateurs, les privilèges sont toujours acquis par l'intermédiaire des concepts.

Cette section sur l'administration des politiques de contrôle d'accès structurées s'intéresse aux mécanismes et principes mis en œuvre pour manipuler les concepts et relations ainsi que pour organiser les droits des administrateurs sur les politiques.

2.4.1 Primitives d'administration

Une fois qu'une organisation a décidé de mettre en place un modèle de contrôle d'accès, il faut développer des outils pour permettre aux administrateurs de définir et faire évoluer les droits du système. Pour les modèles RBAC, un ensemble d'opérations nécessaires est proposé dans le standard qui se compose de deux parties [Ferraiolo01, Ferraiolo03b] :

- d'une part, la spécification ensembliste des modèles de la famille RBAC ($RBAC_0$, $RBAC_1$, $RBAC_2$ et $RBAC_3$) et la formalisation partielle en logique du premier ordre des propriétés qu'une politique doit respecter pour être intègre. Cette formalisation a été reprise et étendue par les autres modèles intégrant également la notion de rôle [Thomas97b, Damiani07, Barker03, Covington01]. Dans l'état de l'art,

nous avons repris la formalisation proposée des modèles RBAC, en prenant en compte les critiques qui lui ont été adressées [Li07]. Cette première partie du standard donne la structure de la politique de contrôle d'accès à rôles,
- d'autre part, la spécification des *fonctionnalités* attendues d'une implantation des modèles RBAC, que nous désignons par *primitives* d'administration :
  - les opérations de *revue*, c'est-à-dire les primitives permettant l'interrogation d'une politique. En termes d'opérations dans une base de données, il s'agit du langage de requêtes de la politique,
  - les opérations de *manipulation*, c'est-à-dire les primitives permettant l'ajout, la suppression et la modification de données dans une politique. En termes d'opérations dans une base de données, il s'agit du langage de manipulation,

Les spécifications du standard RBAC définissent la sémantique des opérations sur la politique ainsi que les préconditions et postconditions devant êtres vérifiées pour que ces opérations ne rendent pas la politique incohérente [Ferraiolo03b, Gavrila98]. Les spécifications des primitives d'administration sont utilisées dans la définition des modèles d'administration [Sandhu99].

Cette partie du standard n'a pas été aussi bien développée dans les modèles de contrôle d'accès autres que RBAC. Comme les opérations administratives et de revue sont des droits sur les politiques, les auteurs ont proposé de les organiser selon des *modèles d'administration*. Il s'agit donc de « métapolitiques » de contrôle d'accès : des organisations des permissions des administrateurs.

2.4.2 Modèle d'administration

Avec l'informatique contemporaine, il peut être difficile d'envisager une administration monolithique centralisée d'une politique de contrôle d'accès de taille considérable. Généralement, plusieurs administrateurs interviennent sur une même politique, avec ou sans partage des responsabilités strictement défini. Dans le cas de grandes organisations, le nombre d'administrateurs peut devenir assez grand pour qu'une structuration de leurs privilèges soit nécessaire.

Une fois un modèle structuré reconnu et accepté, un *modèle d'administration* du modèle de base peut être proposé. Le principe de cette approche, qui a été repris dans plusieurs propositions, est d'ajouter des concepts et relations *administratifs* au modèle de base. Le modèle d'administration organise les privilèges des administrateurs.

Dans plusieurs propositions [Sandhu99, Cuppens04b], le modèle d'administration est un modèle « miroir » du modèle de base qui s'organise selon les mêmes principes : les concepts et relations introduits dans le modèle de base sont utilisés pour structurer les droits des utilisateurs finaux ainsi que ceux de leurs administrateurs. L'objectif de ces modèles est d'avoir une structuration homogène de l'ensemble des droits. Parmi les propositions qui ont suivi cette démarche, on retrouve les modèles d'administration :

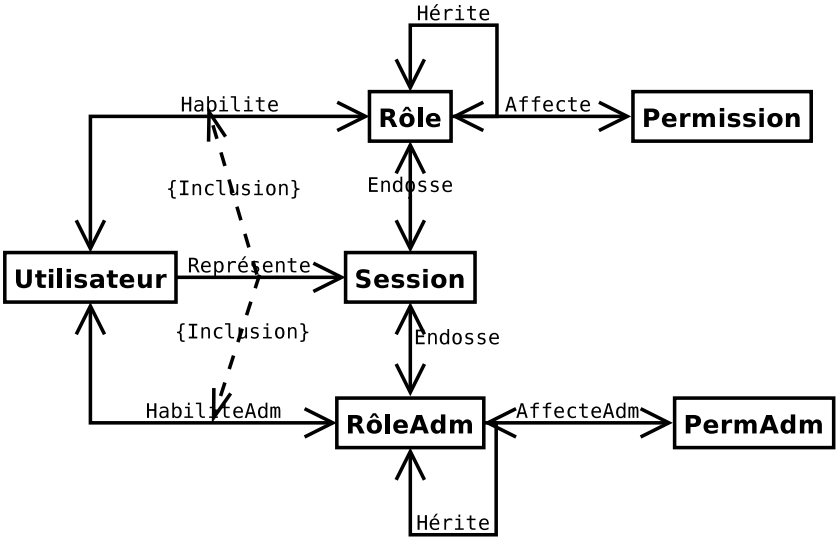


FIG. 2.6 – Modèle d'administration de RBAC : ARBAC (UML)

- *Administrative*-RBAC (ARBAC) pour les modèles RBAC [Sandhu99, Sandhu98],
- *Scoped-Administrative*-RBAC (SARBAC) qui reprend ARBAC en utilisant la notion de portée administrative sur la hiérarchie de rôles [Crampton03b, Crampton05],
- *Administrative*-ORBAC (ADDORBAC) pour ORBAC [Cuppens04b],
- X-GTRBAC-*Admin*, pour le modèle GTRBAC [Bhatti05].

La figure 2.6 est une représentation UML de ARBAC (*Administrative Role-Based Access Control*) le modèle d'administration de RBAC. Dans ARBAC, les concepts de *rôles administratifs* et de *permissions administratives* sont introduits. Les permissions administratives sont les opérations qui permettent d'ajouter, modifier ou supprimer des concepts et relations « classiques » dans les politiques RBAC. Des exemples sont l'affectation et la révocation de rôles aux utilisateurs, l'affectation et la révocation de permissions aux rôles ou encore l'affectation et la révocation de relations d'héritage entre rôles.

La hiérarchisation des concepts dans les modèles de contrôle d'accès revêt une place stratégique dans l'administration des politiques structurées : la hiérarchie est l'élément pivot sur lequel vont être opérées la majeure partie des opérations d'administration. Ainsi, disposer d'une bonne hiérarchie est indispensable pour tirer parti des modèles structurés [Coyne96].

Pour les modèles à rôles, il est reconnu comme bénéfique d'organiser les droits des administrateurs à partir de la hiérarchie des rôles [Crampton05]. Or d'une part, l'identification des rôles est délicate, mais d'autre part, la réorganisation en hiérarchie de rôles définis précédemment sans ordre est une tâche difficile. Si mal entreprise, la hiérarchisation peut être un remède pire que le mal. Il est ainsi important d'assister les administrateurs lors de l'identification et de la hiérarchisation des rôles.

Exemple 2.6 Un méta de trop

Le CHM est très impliqué dans une collaboration régionale de lutte contre le cancer. De nombreux administrateurs collaborent sur ce réseau de soin. Chacun d'entre eux est responsable d'une partie d'un établissement de santé : il faut pouvoir organiser leurs droits, car on projette à terme d'étendre le réseau.

Le modèle d'administration peut être différent du ou des modèles de contrôle d'accès. On peut par exemple organiser les droits des administrateurs d'une politique ORBAC avec un modèle mandataire : chaque administrateur aurait un niveau d'accréditation lui permettant d'accéder à tout ou partie des primitives d'administration.

Dans une perspective visonnaire, les reponsables informatiques du réseau de soin on choisi de mettre en œuvre un modèle RBAC. Les droits d'utilisation des primitives d'administration peuvent alors être régis par le modèle ARBAC qui les structure grâce au concept de rôle administratif. *Quid* des administrateurs qui manipulent les rôles administratifs ? S'agit-il d'une instance encore plus haute que les administrateurs à l'échelle régionale ? On pourrait imaginer un « modèle d'administration du modèle d'administration ». L'expérience montre qu'un seul nouveau niveau d'indirection suffit dans la pratique.

2.4.3 Représentation graphique de modèles et de politiques

Une grande partie des failles dans les mécanismes de contrôle d'accès sont dues à des erreurs humaines d'administration : avec un nombre croissant d'utilisateurs, les politiques deviennent plus grandes, les cas particuliers plus nombreux et la gestion des droits plus complexe. Ceci est d'autant plus vrai lorsque des enrichissements ou des extensions de modèles sont intégrés dans les modèles de contrôle d'accès.

Il est admis que les outils d'aide à l'administration doivent être munis d'interfaces graphiques appropriées, permettant à l'utilisateur de suivre les étapes des processus automatisés qu'ils intègrent [Koch03, Tidswell01, Sandhu04]. Plusieurs formalismes graphiques pour la gestion des politiques de contrôle d'accès ont été proposés : certains spécifiques à RBAC [Ferraiolo03a], d'autres plus génériques [Tidswell01, Koch03]. Ces propositions visent à faire correspondre des symboles et des actions graphiques à des primitives d'administration. Comme à chaque représentation est généralement associé un cadre logique sous-jacent donné (théorie des graphes, graphes conceptuels, UML), nous détaillons les propositions de ce domaine en section 2.6.3.

2.4.4 Synthèse de l'administration des modèles structurés

Le terme d'*administration* recouvre un large spectre des activités de gestion effectuées par les administrateurs : des opérations de création de la politique à la maintenance quotidienne en passant par l'évolution du contenu. Cette section a survolé les notions principales relatives à ce champ d'activité :

- les *primitives* administratives : des ensembles d'opérations permettant d'interroger et de manipuler des politiques,
- les *modèles* d'administration : des modèles de contrôle d'accès aux primitives administratives,
- la *représentation* des modèles et des politiques : des formalismes graphiques en correspondance avec les primitives,

Nous reprendrons ces notions dans la suite de la thèse, plus précisément dans le chapitre 5, où nous proposons une généralisation des notions de primitives et des outils de conception de modèle.

Les propositions de modèles d'administration permettent généralement l'expression de contraintes dans les politiques. La section suivante présente cette notion ainsi que les principaux enrichissements et extensions introduits dans le contrôle d'accès.

2.5 Enrichissements et extensions des modèles structurés

Plusieurs modèles de contrôle d'accès organisent les droits des utilisateurs selon les processus métiers dans lesquels ils interviennent. Répartir les tâches d'un même processus entre plusieurs acteurs permet de garantir le principe de séparation des tâches [Nash90]. Ce principe est généralement introduit dans les modèles structurés par l'intermédiaire de la notion de *contrainte* qui permet de limiter les permissions accordées aux utilisateurs. La littérature distingue deux grandes catégories de contraintes :

- les contraintes *dynamiques*, qui doivent être évaluées lors des demandes d'accès, c'est-à-dire lors de l'exécution du moniteur et de l'ouverture de sessions,
- les contraintes *statiques*, qui sont évaluables une fois pour toutes et qui ne dépendent pas de l'exécution du moniteur.

Une contrainte peut généralement être déclinée dans l'une ou l'autre des catégories : en dynamique, si la contrainte porte sur les *sujets* qui représentent l'activité des utilisateurs du système, ou en statique, si la contrainte porte sur les *utilisateurs* eux-mêmes, qui sont des entités passives du point de vue du contrôle d'accès. La section suivante s'intéresse aux contraintes les plus utilisées dans les modèles de contrôle d'accès : les contraintes d'exclusion mutuelle.

2.5.1 Séparation des tâches et exclusion mutuelle

Le moyen proposé pour garantir le principe de *séparation des tâches* est la notion de *contrainte*, et en particulier les relations d'*exclusion mutuelle* entre concepts d'un modèle de contrôle d'accès (par exemple, entre tâche et utilisateur ou entre rôle et session). La distinction entre *objectif* et *moyen* est attribuable aux auteurs de [Li04a] qui ont précisé ces deux notions, parfois confondues :

- la *séparation des tâches* : un principe de sécurité qui vise à garantir la sécurité d'un processus, en imposant que des acteurs différents interviennent dans sa réalisation, c'est *un objectif* à garantir [Clark87, Nash90, Simon97],
- *l'exclusion mutuelle* entre concepts : une contrainte définie entre les concepts d'un système qui interdit qu'ils aient des utilisateurs en commun, c'est un *moyen* utilisé pour exprimer et garantir la séparation des tâches [Kuhn97].

2.5.1.1 Exclusion mutuelle dans les modèles à rôles

En sus des notions de rôle et de hiérarchie associées, les modèles RBAC ont introduit les contraintes d'exclusion mutuelle avec le modèle RBAC_2. Les contraintes permettent d'imposer de restrictions devant être garanties dans les politiques. Les contraintes sont un aspect important du modèle RBAC, voire même une des principales motivations [Ahn00]. Les contraintes d'exclusion mutuelle (statiques et dynamiques) entre rôles sont d'ailleurs les seules proposées dans le standard [Ferraiolo03b].

Définition (Exclusion mutuelle statique entre rôles [Ferraiolo03b])**.** *L'*exclusion mutuelle statique *entre deux rôles r_1 et r_2, impose qu'aucun utilisateur ne dispose à la fois du rôle r_1 et de r_2. Soit $\mathcal{MER} \subseteq \mathcal{R} \times \mathcal{R}$ la relation d'exclusion mutuelle statique :*

$$\forall r_1, r_2 \ (r_1, r_2) \in \mathcal{MER} \Rightarrow \mathit{assigned\_users}(r_1) \cap \mathit{assigned\_users}(r_2) = \emptyset$$

Définition (Exclusion mutuelle dynamique entre rôles [Ferraiolo03b])**.** *L'*exclusion mutuelle dynamique *entre deux rôles r_1 et r_2, impose qu'aucun utilisateur ne dispose d'une session dans laquelle il endosse simultanément les rôles r_1 et r_2.*

La possibilité pour un utilisateur de disposer de plusieurs sessions est un problème pour l'application des contraintes dynamiques[6]. En effet, si deux rôles r_1 et r_2 sont en exclusion dynamique, cela n'interdit pas à un utilisateur de disposer de deux sessions s_1 et s_2 où il endosse séparément r_1 et r_2 : la séparation des tâches n'est pas garantie. C'est un des arguments justifiant l'intérêt particulier porté à l'exclusion mutuelle statique [Li04a].

L'exclusion mutuelle est donc une relation binaire interne entre rôles. Comme la relation d'héritage, l'exclusion mutuelle doit vérifier des propriétés algébriques, identifiées par les auteurs de [Benantar06, Ferraiolo03b, Gavrila98] :

- elle est symétrique, si r_1 est en exclusion avec r_2, alors r_2 est en exclusion avec r_1 : $\forall r_1, r_2 \ (r_1, r_2) \in \mathcal{MER} \Rightarrow (r_2, r_1) \in \mathcal{MER}$.
- elle est irréflexive, un rôle r ne peut pas être en exclusion mutuelle avec lui même : $\forall r \ (r, r) \notin \mathcal{MER}$

[6] Nous ne proposons pas de formalisation l'exclusion mutuelle dynamique car elle nécessiterait l'introduction de nouvelles définitions techniques.

L'exclusion mutuelle présentée a été étendue aux utilisateurs, aux objets et aux permissions, sous les dénominations de [Crampton03a, Gligor98][7] :

- *User-based separation of duty*, aucun rôle ne doit avoir d'utilisateurs en commun,
- *Permission-based separation of duty*, aucun rôle (ou aucun utilisateur) ne doit avoir de permissions en commun,
- *Object-based separation of duty*, aucun rôle (ou aucun utilisateur) ne doit avoir d'accès communs sur des objets.

2.5.1.2 Interaction entre exclusion et hiérarchie

Certains modèles, comme RBAC_3, autorisent l'expression de contraintes sur des concepts hiérarchisés. Or, les interactions entre les relations d'exclusion mutuelle et les relations d'héritage induisent de nouvelles propriétés que les politiques de contrôle d'accès doivent respecter.

Pour le modèle RBAC_3, on doit redéfinir la relation d'exclusion d'exclusion mutuelle de RBAC_2 pour qu'elle ne porte non pas sur les utilisateurs *affectés explicitement* (application *assigned_users*) aux rôles, mais à tous ceux qui l'*endossent implicitement* (application *auth_users*), c'est-à-dire à tous les utilisateurs qui sont affectés à des rôles qui spécialisent des rôles en exclusion :

$$\forall r_1, r_2 \ (r_1, r_2) \in \mathcal{MER} \Rightarrow auth\_users(r_1) \cap auth\_users(r_2) = \emptyset$$

De plus, la relation d'exclusion et la relation d'héritage portant toutes deux sur les rôles, la définition de contraintes d'exclusion peut conduire à des politiques inconsistantes. Les propositions des auteurs de [Kuhn97, Gavrila98, Benantar06, Ferraiolo03b] ont évalué ce problème avec des formalisations légèrement différentes : la relation d'héritage entre rôles ne dispose pas des mêmes propriétés dans les différentes approches. Nonobstant cette différence, les principales propriétés algébriques qui doivent être vérifiées par la relation d'exclusion ont été identifiées :

- deux rôles en exclusion mutuelle ne peuvent pas hériter l'un de l'autre : $\forall r_1, r_2 \ (r_1, r_2) \in \mathcal{MER} \Rightarrow \neg(r_1 \succeq r_2)$
- il ne doit pas y avoir de rôle qui hérite de deux rôles en exclusion mutuelle : $\forall r_1, r_2, r \ r \succeq r_1 \wedge r \succeq r_2 \Rightarrow (r_1, r_2) \notin \mathcal{MER}$,
- l'exclusion mutuelle est propagée par la relation d'héritage : $\forall r_1, r_2, r \ (r_1, r_2) \in \mathcal{MER} \wedge r \succeq r_1 \Rightarrow (r, r_2) \in \mathcal{MER}$

Ces propriétés n'ont pas été généralisées aux différentes variétés de l'exclusion mutuelle proposées dans [Crampton03a, Gligor98]. Nous pensons que ces propriétés sont applicables à ces formes d'exclusion particulières, mais aussi à d'autres, plus générales encore.

[7] La remarque sur l'ambiguité entre séparation des tâches et exclusion se ressent ici, pour la dénomination que nous avons adopté, il s'agit de *contraintes d'exclusion mutuelle* et non de *séparation*.

Exemple 2.7 Interaction entre exclusion et hiérarchie

Au CHM trois rôles sont définis dans les structures de soin :
- le rôle d'infirmier, qui peut lire des prescriptions,
- le rôle de médecin, qui peut lire et écrire des prescriptions,
- le rôle de responsable des soins, qui hérite à la fois du rôle d'infirmier (pour disposer des mêmes droits que ses subordonnés) et de celui de médecin (en imaginant qu'on impose qu'un responsable des soins soit toujours un médecin).

Afin d'éviter qu'un infirmier puisse un jour modifier des prescriptions, les responsables de la sécurité croient bon de définir comme *exclusivement mutuels* les rôles d'infirmier et de médecin. Un utilisateur qui endosse le rôle de responsable des soins endosse implicitement les rôles d'infirmier et de médecin et viole alors la contrainte d'exclusion. Une telle politique n'est pas intègre.

On pourrait pour ce cas privilégier l'affectation multiple de rôles à l'héritage multiple. Alors, il faudrait affecter à chaque responsable des soins le rôle d'infirmier et celui de médecins. Ceci conduit à multiplier les affectations, augmentant ainsi la taille et la complexité de la politique.

2.5.2 Intégration du contexte

Avec l'évolution des systèmes prenant en compte des caractéristiques par essence dynamiques, comme la position géographique d'un utilisateur, l'état des équipements ou l'horaire d'utilisation, il est devenu nécessaire d'intégrer les aspects contextuels dans les modèles de contrôle d'accès.

Les principes des modèles RBAC ont été repris dans plusieurs propositions d'extensions permettant une définition plus expressive des principes des modèles de contrôle d'accès. La prise en compte la notion de *contexte* dans les politiques à rôles est un domaine d'investigation actif. Cependant, la définition du contexte en informatique est délicate, une définition générale est celle proposée par Dey [Dey01] :

> *Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

Les quatre principales directions des enrichissements contextuels présentées dans cette section concernent :

- la prise en compte *du temps*, pour limiter les actions possibles sur des plages horaires ou pour exprimer l'activation périodique de rôles,
- la prise en compte *de l'espace*, pour exprimer des restrictions géographiques sur les rôles endossables ou les actions possibles par exemple,
- la prise en compte du *contexte des utilisateurs et des objets*,
- les choix de *modélisation* et de *formalisation* des aspects contextuels.

2.5.2.1 Difficultés de modélisation du contexte

À la différence des données « traditionnelles » des politiques de contrôle d'accès, les données *contextuelles* sont souvent numériques. L'évaluation du contexte se base essentiellement sur des comparaisons de valeurs ou des opérations d'analyse numérique. Une particularité commune des données numériques de contexte est qu'elles prennent généralement leurs valeurs sur des domaines *continus*. On peut cependant éviter aux administrateurs de manipuler de tels domaines continus :

- en *discrétisant* les domaines au prix d'approximations. On peut par exemple discrétiser le temps avec une granularité en seconde, ou paver régulièrement le plan avec un motif géométrique,
- en ne s'intéressant non pas aux valeurs elles-mêmes mais aux relations *qualitatives* entre les données.

Plutôt que de manipuler des horaires, on peut par exemple utiliser l'algèbre des intervalles d'Allen [Allen83] (*chevauche*, *finit avec*, *commence avec*, ...). Dans le cas de contexte spatial, on utilisera des relations topologiques [Egenhofer91] (*dans*, *touche*, *croise*, ...). Ces relations qualitatives spatiales sont très intéressantes pour la modélisation des contextes géographiques, car elles sont invariantes par changement de projection ou d'échelle, contrairement aux distances et aux angles.

Le temps ou l'espace sont souvent donnés en exemples comme les principaux représentants du contexte. Mais selon les applications et les domaines d'activités, d'autres valeurs numériques que les coordonnées où les dates peuvent être utilisées. Par exemple, dans les applications orientées réseau, on s'intéresse au temps de latence et au débit des liaisons, dans les applications de surveillance physique, on collectera des valeurs de pression ou de température. Ainsi la notion de contexte est délicate à au moins deux égards :

- c'est une notion qui peut être entendue différemment selon les domaines d'application et les organisations. Pour le géographe, la notion d'espace fait partie du métier, pour l'historien, la notion de chronologie également. La notion de contexte est ainsi relative : alors que pour une organisation donnée, la prise en compte du temps dans le contrôle d'accès est incontournable et fait partie des éléments structurants fondamentaux du modèle, pour une autre ce sera la notion d'espace ou de bande passante,
- la notion de contexte introduit une nouvelle complexité dans l'organisation des droits. La prise en compte du contexte nécessite la définition de nouvelles relations, qui à leurs tours induisent de nouvelles propriétés que les politiques de contrôle d'accès doivent respecter. De plus, certains aspects des contextes sont difficiles à formaliser dans le cas général.

La prise en compte du contexte dans le contrôle d'accès, ou plus généralement dans les systèmes d'information *context-aware*, nécessite la mise en œuvre de mécanismes dédiés à l'acquisition des informations contextuelles. Que ce soit avec un système GPS

pour les coordonnées spatiales, une horloge globale pour les contextes temporels, ou d'autres architectures de type *context-broker* pour collecter de multiples informations de contexte. Dans tous ces cas, nous supposons, comme pour le moniteur de référence, que les mécanismes mis en œuvre sont *incontournables*, *inviolables* et *vérifiés*.

2.5.2.2 Contexte temporel

Pour combler le déficit d'expression d'aspects contextuels des modèles RBAC, les auteurs de [Mossakowski03] proposent d'introduire le temps dans le contrôle d'accès et d'utiliser la logique temporelle du premier ordre (*temporal first-order logic*) comme cadre formel. Les procédures de preuves sur la logique temporelle permettent d'inférer sur des règles du type « un utilisateur peut passer une commande, mais un autre doit *ensuite* la valider » afin de vérifier qu'il n'existera pas d'état inconsistant de la politique.

Avec le modèle nommé *Extended*-RBAC (ERBAC), les auteurs de [Mossakowski03] proposent de manipuler des règles de contrôle d'accès basées sur l'ordre temporel d'exécution des tâches dans les processus métier. Le modèle *Authorization Model for Temporal and derived Data* (AMTD) [Atluri02] poursuit des buts similaires avec le cadre des bases de données temporelles. Ce modèle n'est cependant pas structuré avec le concept de rôle. Enfin, les auteurs de [Joshi05] proposent un modèle nommé *Generalized-Temporal*-RBAC (GTRBAC), qui étend le modèle *Temporal*-RBAC (TRBAC) [Bertino01] en autorisant de nouvelles formes de contraintes temporelles.

Tous ces modèles intégrant le contexte temporel dans la modélisation des droits sont basés sur l'existence d'une horloge globale fiable, grâce à laquelle sont développées des contraintes d'accès basées sur l'horaire où les sujets agissent dans le système :

- l'activation et la désactivation périodique de rôle,
- les affectations de rôles aux utilisateurs et de permissions aux rôles,
- la durée pendant laquelle on peut endosser un rôle.

Comme pour les relations d'exclusion mutuelle qui interagissent avec les hiérarchies, l'introduction du contexte temporel a conduit à proposer des relations d'héritage raffinées en présence de contexte [Joshi02] :

- *Inheritance-only hierarchy* : la hiérarchie classique RBAC,
- *Activation-only hierarchy* : la hiérarchie basée sur le temps,
- *General inheritance hierarchy* : la combinaison des deux précédentes.

Comparées au modèle AMTD [Atluri02], qui est bâti sur un ensemble de règles sans concepts intermédiaires, les propositions basées sur les rôles [Mossakowski03, Atluri02, Joshi05] sont plus fortement structurées et permettent de développer des modèles d'administration [Bhatti05], qui ne sont pas réalisables avec le modèle AMTD.

2.5.2.3 Contexte géographique

Les spécificités des applications mobiles ou basées sur la localisation ont été prises en compte par les auteurs de [Damiani07]. Leur proposition, la famille des modèles *Geographical*-RBAC (GEORBAC), étend les modèles RBAC en définissant de nouveaux concepts spatiaux pour représenter la position des sujets et celles des objets. Ces nouveaux concepts sont utilisés pour limiter géographiquement l'utilisation des rôles. L'intéraction entre aspects géographiques, contraintes et hiérarchie a conduit les auteurs de [Damiani07] à structurer leur famille de modèles comme celle de RBAC (figure 2.3).

Le principe proposé dans GEORBAC est de comparer une position physique, supposée obtenue de façon fiable (par exemple la localisation GPS), à des positions logiques (exemples : route, ville, région) auxquelles sont associés des rôles géographiques. Ce principe a été appliqué à d'autres modèles de contrôle d'accès que RBAC. Les modèles TMAC [Thomas97a, Georgiadis01] et ORBAC [Cuppens03] reprennent le principe de comparaison entre positions physiques et logiques.

2.5.2.4 Contexte d'utilisateur et d'objet

Certaines approches ont choisi d'intégrer la notion de contexte dans le contrôle d'accès en la liant étroitement aux utilisateurs et aux objets, sans introduire de nouveaux concepts spécifiques. Les modèles *Context-Sensitive*-RBAC (CSRBAC) et *Environment*-RBAC (ENVRBAC) regroupent les paramètres dépendants des utilisateurs et des objets dans des *contextes de sujets* et des *contextes d'objets* [Kumar02, Covington01]. Les contextes peuvent comprendre la position géographique ou l'heure de la demande d'accès. Dans CSRBAC et ENVRBAC, un accès est autorisé si et seulement si les contextes de sujets sont inclus dans les contextes d'objets.

2.5.2.5 Formalisation des contextes

Une des difficultés de la prise en compte du contexte dans le contrôle d'accès n'est non pas sa formalisation dans un cadre théorique, mais plutôt les choix de modélisation et leur intégration dans les politiques. La fonction du *moniteur de référence* est modifiée : ce dernier doit désormais assurer des fonctions de *context-broker*. Le moniteur doit collecter les informations contextuelles relatives aux sujets requérants, à l'objet cible et même éventuellement à l'action demandée.

Plusieurs propositions ont modélisé le contexte dans les modèles de contrôle d'accès, nous avons présenté les plus remarquables extensions de RBAC. La formalisation logique des contextes est catégorisable en deux approches : les contextes peuvent être représentés soit par des contraintes dans les *principes* des modèles, soit par des conditions portant sur les *faits* des politiques.

Contraintes dans les principes des modèles

Afin de pouvoir prendre en compte l'expression de relations arithmétiques dans la formalisation logique du contexte dans les modèles de contrôle d'accès, le cadre théorique utilisé est une extension de DATALOG, généralement DATALOG^C qui permet l'expression de contraintes arithmétiques dans les formules logiques [Barker03, Joshi05, Atluri02, Bertino01, Miège05, Damiani07].

Dans le modèle ORBAC (figure 2.5), le contexte est représenté par un concept spécifique *contexte*, relié à tous les concepts concrets (sujet, action et objet) et au concept d'organisation par une relation spécifique. Cette relation exprime qu'un contexte est valide dans une organisation donnée pour un triplet sujet, action et objet donné. Le concept de contexte est utilisé dans la définition des permissions abstraites.

Conditions sur les faits

Dans les modèles RBAC introduisant le temps (TRBAC, GTRBAC, ENVRBAC), les horaires sont introduits dans toutes les relations des modèles, afin de permettre une flexibilité maximum pour l'expression de politiques de contrôle d'accès prenant en compte les horaires.

Avec ce type de modélisation du contexte temporel, pour conditionner les affectations de rôles aux utilisateurs entre les dates constantes $\mathtt{h}_1$ et $\mathtt{h}_2$ on utilisera la formule f_1. Pour limiter les affectations de permissions aux rôles on utilisera f_2.

$$\begin{array}{ll} f_1 : & \mathit{Heure}(H), \mathtt{h}_1 \geq H \geq \mathtt{h}_2 \rightarrow \mathit{Habilite}(\mathtt{user}, \mathtt{role}) \\ f_2 : & \mathit{Heure}(H), \mathtt{h}_1 \geq H \geq \mathtt{h}_2 \rightarrow \mathit{Affecte}(\mathtt{role}, \mathtt{action}, \mathtt{object}) \end{array}$$

Une contrainte d'utilisation engendrée par cette modélisation est qu'elle rend tous les concepts et relations du modèle de contrôle d'accès *dynamiques* en les conditionnant par l'heure. De plus, tout devient des règles : la séparation entre contenus factuel et déduit dans les politiques disparait.

2.5.3 Interdictions et politiques hybrides

Les politiques de contrôle d'accès peuvent être catégorisées en deux grands types généraux, en fonction du principe qui guide la gestion des droits des utilisateurs dans le système :

- les politiques *ouvertes* : tout ce qui n'est pas explicitement interdit est permis,
- les politiques *fermées* : tout ce qui n'est pas explicitement permis est interdit.

Définition (Principe du moindre privilège [Nash90])**.** *Le* moindre privilège *est un principe de sécurité qui impose que les acteurs du systèmes ne disposent que du minimum de privilèges (permissions) nécessaire à la réalisation des tâches qui leurs sont attribuées.*

La modélisation logique d'une politique ouverte ou fermée est identique, seul diffère la mise en œuvre du moniteur. Pour respecter le principe du moindre privilège, les politiques ouvertes sont mieux adaptées. Néanmoins, pour répondre au besoin d'expressivité et de flexibilité de certaines organisations, il est apparu comme intéressant de proposer des politiques *hybrides*, permettant la définition à la fois d'interdictions et d'autorisations [Halpern03, Bertino03, Jajodia01, Abadi07, DeTreville02]

L'introduction de la négation dans la définition des droits d'accès pose plusieurs problèmes d'ordres pratiques, mais aussi théoriques :

- elle conduit à des conflits et demande la définition de règles permettant de les résoudre. Les mécanismes de résolution proposés peuvent être soit des règles de haut niveau, exprimées en logique du second ordre par exemple [Li00b, Al-Kahtani04], soit des règles portant sur des symboles de relations intermédiaires auxquels les administrateurs n'ont pas accès. Ces relations sont introduites uniquement pour leur intérêt technique [Miège05],
- elle rend plus coûteux les algorithmes qui déterminent si un accès est autorisé. En effet, la sémantique des formules exprimées dans des cadres théoriques supportant la négation est plus complexe que celle de ceux qui ne l'autorisent pas [Abiteboul95].
- elle est difficile à représenter graphiquement. Son support n'est pas directement pris en compte ni dans les propositions d'interfaces d'administration des modèles à rôles, ni dans les paradigmes de représentation graphique des modèles de contrôle d'accès [Tidswell01, Koch03, Ferraiolo03a, Sandhu04].

Exemple 2.8 Difficultés engendrées par les politiques hybrides

Supposons que le CHM, qui utilisait une politique fermée basée sur les rôles, réforme sa gestion des droits d'accès avec un modèle hybride. Les administrateurs utilisaient une application graphique, dans laquelle pour associer une permission à un rôle il fallait tracer un lien entre le rôle et l'objet de la permission.

- Pour représenter les interdictions les administrateurs vont devoir manipuler de nouveaux symboles, qui peuvent être multipliés si l'on souhaite proposer des options comme « ajouter toutes ces permissions sauf ... ».
- Que faire si un administrateur interdit une permission à un rôle alors qu'elle était autorisée par un rôle parent ?
- Quelle résolution des conflits choisir lorsqu'un utilisateur dispose de plusieurs rôles, dont un qui interdit un accès et l'autre qui l'autorise ?

Les difficultés engendrées par les politiques hybrides ont été rencontrées dans un autre mécanisme de sécurité : les pare-feux. Alors que le contrôle d'accès logique autorise ou interdit des *actions demandées* par les utilisateurs, les pare-feux autorisent ou interdisent des *paquets à transiter* sur les réseaux. Dans les politiques des pare-feux, il a été fait le choix historique d'organiser le filtrage des paquets avec des politiques hybrides. Ce choix a causé et cause encore de nombreux problèmes d'expression [Gouda04], de

stockage [Liu04], de vérification des règles [Al-Shaer04, Mayer06] et est une source d'erreur de configuration considérable [Wool04].

Cependant, dans les modèles structurés qui permettent l'expression de contraintes, comme l'exclusion mutuelle, il est possible de limiter les permissions des utilisateurs, sans utiliser explicitement d'interdiction. De plus, en pratique, on peut limiter l'impact de l'absence d'interdictions en proposant des fonctionnalités adéquates dans les interfaces graphiques d'administration qui limitent le nombre d'opérations élémentaires à effectuer pour exprimer l'équivalent d'une interdiction.

2.5.4 Vers le contrôle d'accès basé sur des cadres logiques

Les principaux modèles de contrôle d'accès que nous avons présentés sont basés sur des structurations fortes des autorisations. Un nombre limité de concepts et de relations forme le noyau du modèle, à partir duquel sont dérivées les autorisations effectives. Les principes et algorithmes qui régissent ces dérivations sont également en nombres limités. Le but est de proposer des modèles de contrôle d'accès qui répondent au mieux aux besoins de modélisation des droits des organisations. C'est avant tout la nature *opérationnelle* du contrôle d'accès qui prime sur sa formalisation logique.

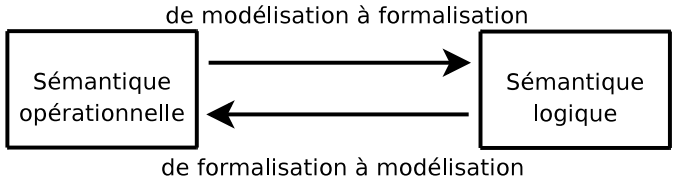


FIG. 2.7 – Modélisation et formalisation du contrôle d'accès

Depuis 2003 environ, nous assistons à la tendance inverse, en rupture avec le développement des modèles à rôles. Cette tendance est désignée sous le terme de « contrôle d'accès basé sur les règles » (*Rule-Based Access Control – Rule-*BAC). Les deux démarches sont illustrées par la figure 2.7.

Comme il est difficile de rassembler en un unique modèle cohérent et homogène les besoins variés de sécurité des organisations, les auteurs proposent désormais des cadres logiques dans lesquels formaliser et développer des modèles de contrôle d'accès. Cette approche formelle est bien moins pragmatique que celle motivée en premier lieu par l'usage et l'administration du contrôle d'accès. De multiples cadres logiques avec lesquels modéliser le contrôle d'accès ont été proposés [Halpern03, Bertino03, Jajodia01, Abadi07, DeTreville02, Al-Kahtani04]. Leur étude est l'objet de la section suivante.

2.6 Cadres logiques pour le contrôle d'accès

Nous avons présenté un ensemble de modèles de contrôle d'accès structurés et de principes qu'ils mettent en œuvre, depuis les modèles historiques les plus simples, jusqu'à des modèles riches, comportant de nombreux concepts permettant de structurer les droits.

Cependant, dans un but d'unification, d'autres approches de modélisation du contrôle d'accès ont été proposées. Elles consistent à identifier des cadres logiques sur lesquels bâtir des politiques de contrôle d'accès. Ces approches s'attachent rarement à un modèle particulier : elles visent moins à structurer les droits qu'à les formaliser pour pouvoir utiliser des outils formels. Disposer de cadres logiques dans lesquels exprimer les règles régissant les droits dans les systèmes est une étape qui facilite :

- la définition de langages sans ambiguïtés (fragments de la logique),
- l'évaluation théorique du coût des algorithmes qui seront exécutés (complexité),
- la définition de procédures d'inférence sur les politiques (décidabilité).

Les approches que nous présentons dans cette section sont basées sur des cadres de logique du premier ordre[8], dans lesquels des constructeurs spécifiques sont développés pour exprimer des règlements de sécurité. La plupart de ces cadres sont basés sur DATALOG et ses dérivés.

Un programme DATALOG est un ensemble de règles, sous forme de clauses de Horn, satisfaisant à certaines restrictions syntaxiques, selon la variante de DATALOG utilisée. Les extensions de DATALOG comprennent principalement le support de la négation ($\text{DATALOG}^{\neg}$) ou des contraintes arithmétiques (DATALOG^{C}). DATALOG constitue un fragment de la logique du premier ordre qui n'autorise pas les fonctions et qui est particulièrement adapté à la modélisation logique des bases de données relationnelles : c'est le paradigme des bases de données déductives[9].

Contrairement aux programmes logiques PROLOG (PROgrammation LOGique), en DATALOG les *faits* (le contenu de la base de données relationnelle) et règles qui les gouvernent sont clairement séparés. Dans les programmes PROLOG, les faits font partie intégrante du programme [Abiteboul95]. Cette séparation permet par exemple d'optimiser l'accès aux données. De plus, les études théoriques des modèles DATALOG portent généralement sur des structures *finies*, comme l'ensemble des faits contenus dans une base de données.

De façon informelle, nous pourrions dire que les programmes DATALOG sont des programmes logiques avec *peu* de règles et *beaucoup* de données, alors que les programmes PROLOG sont composés de *peu* de données et de *beaucoup* de règles.

[8] Il a été proposé d'utiliser des logiques d'ordres supérieurs dans [Appel99] par exemple.

[9] Nous décrivons les liens entre logique des prédicats et base de données dans l'annexe A.

2.6.1 Cadres pour la gestion de la confiance

Dans la thèse, nous avons fait le choix de ne pas traiter d'authentification, nous supposons ces mécanismes sous-jacents existants. Cependant, les études sur l'authentification et l'autorisation ont apportés des réponses à certains problèmes de formalisation logique. En effet, plusieurs cadres logiques pour l'étude des règlements de sécurité ont été proposés dans le domaine du *Trust Management*, que nous traduirons par « infrastructure de gestion de la confiance ».

Certaines propositions de cadres logiques pour le *Trust Management* s'orientent plus sur les infrastructures de gestions de clefs et les échanges sécurisés [Clarke01], d'autres s'intéressent à la gestion des droits et en particulier les langages pour exprimer les modèles et les politiques de contrôle d'accès. Nous portons notre attention sur ces dernières propositions principalement.

2.6.1.1 Logique de délégation et gestion de la confiance

L'approche nommée *Delegation Logic* (DL ou D1LP), vise la définition des autorisations dans les systèmes distribués [Li03a, Li00a, Li00b]. Le cadre propose des constructeurs spécifiques pour formaliser les règles à partir desquelles sont dérivées les autorisations. L'expression `Bob` *says remove*(`file1`) indique que le sujet `Bob` souhaite supprimer le fichier `file1`. Cette expression utilise le constructeur d'assertion *says*.

Le cadre DL propose deux constructeurs pour modéliser la délégation (constructeurs *delagates* et *represents*) et un autre pour l'expression de contraintes de séparation des tâches (constructeur *threshold*). Il n'a pas été proposé de mécanisme d'inférence traitant directement les expressions écrites en DL. En revanche, il est possible de réécrire les expressions de ce langage en DATALOG à l'aide de règles qui permettent de traduire ces constructeurs en logique des prédicats du premier ordre.

La proposition de cadre logique RT_1^C [Li03b] pour *Role-based Trust-management* se base sur *Constraint*-DATALOG (DATALOG^C). DATALOG^C est une extension de DATALOG où des contraintes arithmétiques sont autorisées dans le corps des règles, c'est une forme restreinte de la programmation logique par contraintes. Un exemple d'utilisation de RT_1^C est proposé, il s'agit d'analyser et de vérifier des expressions du langage *KeyNote*, une infrastructure de gestion de la confiance [Blaze99].

2.6.1.2 D3LOG

Le langage *Secure Dynamically Distributed Datalog* (D3LOG), proposé par Jim Trevor ajoute à DATALOG des constructeurs spécifiques reliés aux échanges cryptographiques [Jim01a]. L'objectif de D3LOG est d'exprimer des modèles de contrôle d'accès, mais surtout les échanges sécurisés et les étapes de l'authentification : cela forme le cadre D3LOG proposé dans [Jim01b].

En particulier, on peut exprimer en D3LOG qu'une entité est détentrice de clefs de cryptographie. L'objectif est la formalisation logique de protocoles d'échange cryptographique en vue de l'évaluation de l'authentification des systèmes. Un exemple de modélisation est proposé dans [Jim01a]. D3LOG est utilisé pour spécifier le protocole DNSSEC[10]. L'évaluation du langage est une réduction en DATALOG [Jim01b].

2.6.1.3 BINDER

BINDER est un langage d'expression de règlements de sécurité basé sur la logique du premier ordre, défini par son auteur comme « un DATALOG pour les autorisations » [DeTreville02]. Des constructeurs spécifiques sont introduits pour modéliser les chiffrements utilisés dans les échanges. On peut exprimer qu'un message est crypté avec RSA (*Rivest Shamir Adleman*, un algorithme de cryptographie à clefs publiques très répandu) en spécifiant la clef publique utilisée par exemple. Le langage introduit également le constructeur *says*, similaire à celui proposé dans DL. Le mécanisme utilisé pour l'évaluation de BINDER est une réduction en DATALOG.

Enfin, Martín Abadi a proposé d'utiliser le *Dependency Core Calculus* (DCC), une extension du λ-calcul pour formaliser et inférer sur les modèles de contrôle d'accès [Abadi07]. L'approche permet de raisonner sur des formules du type de BINDER. Cette proposition montre que le DCC est adéquat pour représenter des règles de contrôle d'accès aux systèmes distribués. Cependant, aucune application sur un modèle de contrôle d'accès existant n'est proposée, seule une structuration *ad hoc* est utilisée.

2.6.1.4 Synthèse sur les cadres pour la confiance

Les propositions présentées s'intéressent à un spectre très large de la sécurité, celui de la gestion de la confiance qui comprend identification, authentification et contrôle d'accès. Comme l'authentification se base sur des primitives cryptographiques, plusieurs de ces propositions les modélisent, ce qui les rapproche des travaux sur la vérification de protocoles d'échange sécurisés (exemple avec DNSSEC pour D3LOG). Un autre trait de ces propositions est qu'elles s'intéressent particulièrement aux systèmes hautement distribués, pour lesquels il est difficile de disposer d'une autorité centrale connaissant tous les droits dans le système.

Ces approches proposent des résultats théoriques très généraux, dont la portée dépasse le cadre strict de la gestion de la confiance [Jim01b]. Le revers de la médaille est que ces résultats sont quelquefois difficiles à mettre en œuvre. De plus, pour ce qui est de l'expression des modèles de contrôle d'accès et de leurs principes :

[10] DNSSEC est un service de résolution de noms sécurisé qui étend *Domain Name Service* [Arends05]

– d'une part, ces approches ne basent pas la structuration des droits sur des modèles établis : elles laissent libres de modéliser sous forme de règles les principes et les propriétés de sécurité que les politiques de contrôle d'accès doivent vérifier,
– d'autre part, ces propositions ne traitent ni de la mise en œuvre du principe de séparation des tâches ni des contraintes d'exclusion mutuelle,
– enfin, nonobstant l'héritage affirmé de DATALOG, peu d'éléments sont donnés quant au stockage des politiques et à leur administration.

2.6.2 Cadres pour le contrôle d'accès

Les propositions précédentes de cadre logiques s'intéressent aux architectures de gestion de la confiance. Celles que nous présentons maintenant ne s'intéressent plus à l'authentification, mais seulement au contrôle d'accès. Les aspects étudiés sont principalement l'expressivité des modèles ainsi que l'interrogation et la vérification de politiques afin de prouver qu'elles satisfont bien à des propriétés de sécurité.

2.6.2.1 Spécialisation de programmes logiques

Une première approche est celle Steve Barker [Barker03], qui propose de spécialiser des programmes logiques avec contraintes pour la modélisation des règlements de contrôle d'accès. La programmation logique par contraintes (*Constraint Logic Programming* – CLP) est une extension de PROLOG autorisant la présence de contraintes arithmétiques dans les formules [Jaffar94].

Steve Barker étudie en particulier RBAC_3 et propose des extensions de ce modèle de contrôle d'accès pour exprimer des politiques hybrides. Il est aussi proposé de prendre en compte le temps dans les autorisations pour se rapprocher du modèle TRBAC [Barker03]. L'approche proposée est principalement orientée sur la performance et compare l'efficacité des programmes logiques spécialisés face à un *solver* de contraintes classique. Cependant, la proposition ne prend en compte ni l'intégrité des politiques, ni l'interaction entre hiérarchies et contraintes. De plus, certaines propriétés de RBAC_3 ne sont pas prises en compte [Ferraiolo03b].

L'approche de Halpern et Weissman se base aussi sur l'étude d'un fragment de logique, LITHIUM, construit autour de la notion de bipolarité [Halpern03]. La bipolarité est une restriction syntaxique qui permet de garantir la décidabilité de l'implication en présence de négation, cette notion est proche de la stratification en $\text{DATALOG}^{\neg}$. La proposition de Halpern et Weissman vise la définition de politiques hybrides, prenant en compte des autorisations temporelles simples. Le modèle de contrôle d'accès proposé est une variante hybride du modèle RBAC_1. Malheureusement, la formalisation proposée de RBAC n'est pas celle du standard. La formalisation de l'affectation de rôles aux utilisateurs interdit la quantification des rôles ($\exists\, r \in \mathcal{R} \ldots, \forall r \in \mathcal{R} \ldots$) ce qui limite fortement l'expression des propriétés de sécurité à vérifier.

2.6.2.2 C-Datalog

Les auteurs de [Bertino03] ont proposé d'utiliser le paradigme *Complex*-DATALOG (C-DATALOG), une extension orientée objet de DATALOG [Greco92] qui permet d'exprimer de façon naturelle des hiérarchies entre les concepts des modèles de contrôle d'accès. Cette proposition est utilisée pour représenter les modèles *Identity*-BAC, MAC et RBAC hybrides. L'approche permet également l'expression de contraintes sur les politiques sous forme de règles, mais sans structuration. La méthode proposée pour l'implantation est une traduction en DATALOG des modèles exprimés en C-DATALOG.

2.6.2.3 Authorization Specification Language

Le langage *Authorization Specification Language* (ASL), est proposé dans [Jajodia01] comme cadre logique pour exprimer des politiques de contrôle d'accès hybrides. Ce framework se base sur la notion de *stratification* pour permettre l'expression de négation. Les *strates* sont des restrictions syntaxiques sur les expressions autorisées du langage. Le cadre est illustré par un modèle de contrôle d'accès *ad hoc* qui intègre des notions de rôles et de hiérarchies, mais sans se baser strictement sur le cadre de RBAC (comme le font [Barker03, Bertino03]). L'article s'intéresse particulièrement aux problèmes de gestion des conflits qui surviennent dans les politiques hybrides, mais ne traite pas des contraintes entre concepts.

2.6.2.4 Logiques modales

Les logiques modales étendent la logique des prédicats du premier ordre en ajoutant, en plus des connecteurs booléens, des modificateurs qui peuvent s'interpréter informellement comme des adverbes sur les prédicats. Les modalités permettent d'exprimer la possibilité, la permission, l'obligation, la connaissance, etc. La sémantique des mondes possibles (ou sémantique de Kripke) permet d'évaluer la vérité des propositions de logique modale. Selon les modes autorisés dans la construction des formules, différentes logiques modales existent :

- la logique modale *classique*, exprime la nécessité et la possibilité (et leur négation),
- la logique *déontique*, exprime l'obligation, l'interdiction et la permission, elle permet d'exprimer des règlements moraux,
- les logiques *temporelles*, introduisent des modes temporels sur un temps discret, comme toujours, un jour, jusqu'à ce que, ...

La logique déontique [Jones93] a été proposée pour évaluer les règlements de sécurité [Abou El Kalam03, Cholvy97], en particulier pour résoudre les conflits dans les politiques hybrides. Certaines simplifications ont été apportées, notamment la suppression de la modalité de recommandation. Le cadre de la logique déontique a été utilisé

pour exprimer formellement le modèle ORBAC, mais pour l'évaluation des formules, l'approche se base sur une transformation en logique du premier ordre non modale.

La logique temporelle linéaire (*Linear Temporal Logic* – LTL) a été proposée pour vérifier qu'une politique satisfait bien à des propriétés attendues. L'approche de [Hansen05] est celle du *Model-Checking* de politiques RBAC. Le modèle de contrôle d'accès et la politique sont exprimés dans un langage (PROMELA), les propriétés attendues – c'est-à-dire les contraintes sur les modèles, comme l'exclusion mutuelle – sont exprimées en LTL. Le vérificateur SPIN [Holzmann97] vérifie que les propriétés sont satisfaites en explorant tous les états possibles du système RBAC. Les auteurs de [Mossakowski03] ont également proposé d'utiliser une logique temporelle pour représenter formellement les propriétés des modèles de contrôle d'accès *Extended*-RBAC qui permet l'expression de contraintes temporelles.

2.6.3 Cadre basés sur la représentation graphique

Les auteurs de [Koch03] notent que pour éviter les erreurs dans les politiques de contrôle d'accès, il est utile de disposer d'une modélisation graphique qui permette de représenter les règles de contrôle d'accès et rendre ainsi plus facile les tâches d'administration. [Koch03] fait un tour d'horizon des paradigmes graphiques proposés pour la spécification et la vérification graphique de politiques de contrôle d'accès. Le type de vérifications étudié est la « résolution des conflits », c'est-à-dire la détection de contraintes inconsistantes. Trois approches sont évaluées dans [Koch03], nous adjoignons à cette comparaison une quatrième (la dernière) :

- ALLOY est un langage graphique basé sur la logique du premier ordre muni d'un *model checker* qui permet l'expression de comportement et contraintes complexes [Jackson00]. La méthode proposée de vérification pour les politiques RBAC est un parcours exhaustif de toutes les sessions possibles. Cette approche ne permet pas d'inférer sans définir de politique ni d'exprimer des contraintes.
- *Graph Transformation* (GT) est un modèle de graphe où des règles de transformation sont utilisées pour déduire des connaissances [Rozenberg97]. Ce modèle ne permet pas de représenter toutes les propriétés que les modèles de contrôle d'accès respectent. Il ne permet pas non plus l'expression des politiques.
- UML (*Unified Modelling Language*) muni d'OCL (*Object Constraint Language*) pour ajouter une sémantique logique aux schémas UML [Richters01]. Cette modélisation reste cependant semi-formelle car basée sur UML. De plus, la méthode proposée pour la vérification des propriétés est une transformation en GT.
- la représentation graphique du modèle *Dynamically Types Access Control* (DTAC), basée sur les graphes. Cette approche permet de représenter graphiquement les noyaux des modèles de contrôle d'accès, ainsi que les principes qui les gouvernent [Tidswell01]. Comme GT, ce modèle ne permet pas de représenter les politiques, seulement les modèles.

	Représentation	Vérification
UML	Concepts, principes	Transformation en GT
ALLOY	Concepts	Parcours exhaustif
GT	Concepts, principes, contraintes	Parcours exhaustif, résolution de conflits
DTAC	Concepts, principes, contraintes	Résolution de conflits

TAB. 2.6 – Cadres pour la spécification et la vérification graphique

Seul le modèle DTAC est appliqué à d'autres modèles que RBAC. Le tableau 2.6, en partie extrait de [Koch03], synthétise les apports de ces propositions. La première colonne indique les aspects des modèles de contrôle d'accès que les propositions sont capables de représenter. La seconde colonne indique quelles sont les possibilités de vérifications proposées par les approches.

2.7 Discussion et synthèse

Cette section résume et critique l'état de l'art que nous avons proposé, en croisant les aspects fonctionnels des modèles de contrôle d'accès : structuration, hiérarchisation des concepts, enrichissements et outils d'administration, avec les aspects théoriques des cadres proposés pour leur étude. Nous terminons ce chapitre par une représentation synthétique, sous forme de tableaux croisés (tableaux 2.7 et 2.8).

2.7.1 Structuration des droits

Depuis les modèles historiques, nous avons suivi l'évolution du contrôle d'accès vers la structuration des droits, puis l'enrichissement progressif des modèles avec des contraintes et la prise en compte de contextes et enfin les cadres logiques pour représenter et raisonner sur le contrôle d'accès. Nous sommes en présence de propositions :

- de *modèles* de contrôle d'accès qui introduisent de multiples concepts pour structurer les droits, du rôle à l'organisation en passant par les tâches et les hiérarchies,
- d'*enrichissements* des modèles, avec la notion de contraintes – en particulier l'exclusion mutuelle et la séparation des tâches – ou l'intégration du contexte,
- *composites*, combinant des concepts introduits dans différents modèles, avec le but d'offrir la palette la plus vaste possible d'expression pour structurer les droits,
- de *cadres logiques* qui permettent l'expression de multiples règles de déduction des autorisations, mais qui n'imposent pas de structuration des droits,
- de prise en compte des besoins de l'*administration* et de la *conception* de modèles et de politiques de contrôle d'accès.

La famille des modèles RBAC et les modèles qui les ont étendus ont fait l'objet d'études approfondies. Nous avons assisté à des raffinements successifs du noyau fondateur $RBAC_0$, construit autour de la notion centrale de rôle. Ces enrichissements n'ont pas toujours comblé les lacunes des modèles initiaux [Li07].

D'autres auteurs ont préféré remplacer la notion de rôle des modèles RBAC par une ou plusieurs autres, comme les organisations dans ORBAC ou les tâches dans TBAC. Une démarche de raffinement à partir de ces nouveaux concepts est alors entreprise. Les extensions s'orientent souvent vers la prise en compte du contexte et des contraintes. Cependant ces approches n'intègrent pas certaines des propriétés identifiées dans les modèles RBAC. Le problème majeur de l'*intégrité* des politiques, étudié dans le cadre de RBAC, n'a pas toujours été repris.

Nous pensons qu'il est important d'identifier des concepts centraux pour que les administrateurs puissent structurer efficacement les droits d'accès. Nous considérons même que la structuration du contrôle d'accès est *incontournable* pour une gestion sûre et efficace des droits, et qu'en conséquence, les approches non structurées sont trop délicates à utiliser, en particulier car elles manquent d'outils pratiques associés.

Nous pensons cependant qu'aucune structuration n'est meilleure qu'une autre dans le cas général : concevoir un bon modèle de contrôle d'accès pour une organisation est une tâche spécifique propre à cette dernière. En effet, pour une petite structure, un modèle contextuel simple est un choix satisfaisant : il est envisageable de gérer plusieurs contextes associés aux utilisateurs si leur nombre est réduit. À l'inverse, pour une multinationale, il sera certainement nécessaire d'avoir une notion de rôle générale accompagnée de concepts et de relations spécifiques, capables de prendre en compte de multiples facettes des métiers. Ceci explique en partie les spécificités des implémentations du standard RBAC.

Le modèle de contrôle d'accès pour les métiers de la santé proposé par le GMSIH est un bon exemple de modèle développé pour une catégorie d'organisations, par et pour les acteurs des systèmes d'informations de santé [GMSIH03]. Nous pensons donc qu'il faut permettre aux concepteurs de développer des modèles de contrôle d'accès adéquats aux besoins de sécurité des organisations, en prenant en compte l'évolution des modèles et les besoins de l'administration.

2.7.2 Intégrité des politiques

Dans tous les cas d'utilisation des modèles de contrôle d'accès : il est incontournable de s'assurer de l'*intégrité* de la politique, c'est-à-dire s'assurer que la politique de contrôle d'accès respecte bien le modèle dont elle est une instance. En particulier, des propriétés devant être respectées sont définies dans les modèles, il faut s'assurer que les politiques les satisfassent effectivement.

Plusieurs propositions ont identifié les propriétés que les modèles à rôles doivent respecter. Des outils pour vérifier que les politiques respectent bien ces propriétés ont été proposés en conséquence. Cependant, ces vérifications n'ont pas toujours envisagées pour les modèles autres que RBAC.

Cette constatation est particulièrement notable pour les approches qui permettent de définir de nouveaux modèles *ad hoc* bâtis sur des cadres logiques, sans structuration imposée. La définition de nouveaux concepts et relations dans un modèle implique la définition de contraintes s'assurant que la politique est valide. Par exemple, l'interaction entre la notion de hiérarchies et l'exclusion mutuelle induit des propriétés que les politiques doivent respecter. Or plusieurs modèles proposent d'utiliser l'exclusion et les hiérarchies, mais peu ont identifié ces interactions.

2.7.3 Cadres logiques

Les cadres logiques que nous avons présentés sont majoritairement proposés comme des extensions de DATALOG. Ces modèles permettent d'exprimer des règles de déduction, mais certaines contraintes permettant de s'assurer que les politiques sont valides ne sont pas envisagées. De plus, l'implantation des modèles est généralement une réécriture de règles comportant des constructeurs *ad hoc* en DATALOG, ce qui ajoute une étape de traduction des modèles.

Plusieurs modèles utilisent la notion de rôle, or dans [Ferraiolo03b] il est indiqué que *toute* affectation de privilège *doit* se faire par l'intermédiaire d'un ou plusieurs rôles, aucune affectation directe de permission à un utilisateur n'est autorisée. Cette propriété permet de s'assurer qu'aucun administrateur n'a « court-circuité » la notion de rôle, et évite de tomber dans l'écueil des modèles matriciels où la revue des permissions est difficile. À notre connaissance, ce principe élémentaire des modèles RBAC n'a pas été prise en compte dans d'autres modèles structurés.

Enfin, les propositions de formalisation ne séparent pas toujours manifestement les différents composants du contrôle d'accès, en particulier la distinction entre d'une part, le modèle qui structure les droits, et la politique qui en est l'instance d'autre part. De même, les activités de conception de modèle, de conception de politiques et d'administration sont généralement mélées. Enfin, la séparation entre le contenu factuel de la politique et ce qui peut en être déduit n'est pas toujours séparé non plus.

2.7.4 Notre approche

Notre approche est de structurer les politiques de contrôle d'accès du point de vue des données. Nous allons catégoriser l'organisation des droits d'accès selon :

– les *modèles* de contrôle d'accès : l'ensemble de concepts et de relations qui permettent de *structurer* les droits,

- les *politiques* elles-mêmes : des *instances* des modèles de contrôle d'accès que les administrateurs alimentent et sur lesquelles se basent les moniteurs,
- les *principes* des modèles : des ensembles de règles de déduction qui permettent de *dériver* les autorisations effectives à partir des politiques,
- les *contraintes d'intégrité* des modèles : des ensembles de propriétés qui permettent de s'assurer de la validité des politiques.

Le modèle DATALOG et ses dérivés ont été largement adoptés comme cadre d'études des modèles de contrôle d'accès structurés. Ces modèles peuvent s'avérer insuffisants pour exprimer certaines restrictions que les politiques doivent satisfaire, en particulier celles qui expriment l'existence de fait inconnus à l'avance, mais dont on souhaite imposer la présence. Grâce à l'utilisation des *dépendances de données*, notre approche permet :

- d'exprimer et de concevoir des modèles de contrôle d'accès structurés,
- de prendre en compte des enrichissements apportés sur ces modèles,
- de définir formellement des propriétés que les modèles doivent respecter,
- de définir formellement l'intégrité et l'équivalence des politiques,
- d'utiliser des outils de preuve pour vérifier des propriétés :
  - abstraites, relatives aux modèles eux-mêmes,
  - concrètes, relatives aux politiques instances des modèles,
- de pouvoir contrôler la décidabilité des raisonnements,
- de proposer une représentation graphique des modèles,
- de proposer des outils pour assister la création des politiques.

Notons qu'il existe un débat sur l'intérêt de mettre en œuvre des classes de dépendances de données de plus en plus générales, qui permettent d'exprimer des restrictions de plus en plus expressives sur les données. En effet, d'aucuns pensent que les seules classes de dépendances utiles pour la modélisation du monde réel sont en fait les plus connues, à savoir les dépendances fonctionnelles, les dépendances d'inclusion et éventuellement, les dépendances multivaluées. D'autres pensent néanmoins que des classes plus expressives sont d'intérêt [Fagin86].

La thèse va dans cette direction en présentant des applications aux besoins de modélisation du contrôle d'accès qui font appel aux classes de dépendances les plus générales. Nous exploitons également l'expressivité dans l'autre direction pour proposer de nouvelles propriétés d'intrégité des politiques.

2.7.5 Tableaux de synthèse

Les tableaux 2.7 et 2.8 résument le chapitre d'état de l'art en croisant, d'une part l'étude des modèles de contrôle d'accès (tableau 2.7) et d'autre part les cadres logiques proposés pour leur étude (tableau 2.8). Il s'agit avant tout d'essais de synthèse, dans la mesure où il est délicat de trouver des critères objectifs consensuels pour comparer les propositions de modèles de contrôle d'accès et de cadres logiques, mais aussi d'affirmer qu'une proposition remplit ou non un des critères.

Un rond plein (•) dans une case indique un support explicite d'un critère, un rond vide (○) indique que le critère est peut-être applicable ou qu'il n'a pas été détaillé. Enfin une case vide indique que le critère qu'aucun élément n'est donné et (N/A) qu'il n'est pas applicable du tout.

Les deux tableaux pourraient être regroupés pour former une seule représentation de synthèse qui aurait la forme de la lettre L. La partie grisée de la figure 2.8 est l'intersection entre les deux tableaux de synthèse. Il s'agit du croisement entre les modèles de contrôle d'accès d'une part, avec le tableau 2.7 qui forme la barre verticale du L, et les cadres logiques utilisés pour les formaliser d'autre part, le tableau 2.8 qui forme la barre horizontale du L.

Nous proposons comme dernière ligne de chaque tableau de synthèse la proposition de la thèse. Nous n'avons pas fait figuré dans les critères certains éléments de comparaisons qui n'ont, à notre connaissance, pas été proposés, comme la conception de modèle ou la catégorisation entre aspects statiques et dynamiques sur laquelle nous définissons l'administration des politiques.

La page suivante est volontairement laissée blanche pour que les tableaux et leurs descriptions respectives soient face-à-face.

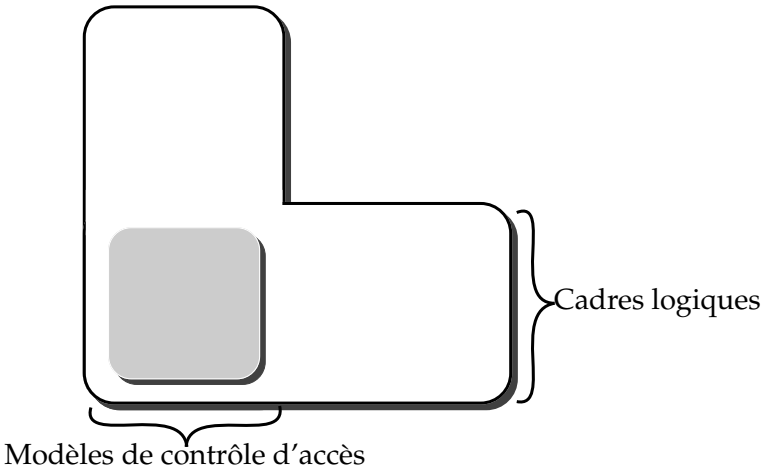


FIG. 2.8 – Combinaison des deux tableaux de synthèse

2.7.5.1 Critères de synthèse des modèles de contrôle d'accès

Afin de faciliter la lecture du tableau de synthèse 2.7, les modèles de contrôle d'accès sont regroupés selon les grandes catégories de concepts présentés :

- les cinq premières colonnes regroupent les modèles historiques,
- les quatre suivantes concernent des modèles orientés tâches et flux,
- la colonne suivante concerne le modèle ORBAC,
- les six colonnes suivantes regroupe les modèles organisés selon la notion de rôle,
- la dernière regroupe sous un terme générique les modèles basés sur les règles, section 2.5.4.

Les critères (en lignes) proposés pour évaluer les différents modèles de contrôle d'accès sont regroupés en quatre blocs :

1. la *structuration* des modèles :
   - quels sont les *concepts* utilisés ou introduits pour structurer les droits,
   - quels types de *hiérarchies* ont été proposés pour organiser les concepts,
   - la possibilité de définir des *interdictions* dans le modèle,
   - les aspects de la structure qui ont été *formalisés*,
2. les *enrichissements* proposés pour ces modèles :
   - les catégories de *contextes* introduites dans l'expression des autorisations,
   - les *contraintes* qui peuvent être définies dans les modèles,
3. les outils d'*administration* qui ont été développés :
   - les *modèles* d'administration, qui organisent les droits des administrateurs,
   - les primitives *d'interrogation* des politiques
   - les outils de *fouille* pour assister la création de politiques,
   - les outils permettant de *vérifier l'intégrité* des politiques
   - les *représentations graphiques* des modèles et des politiques,
4. les cadres logiques proposés pour formaliser et vérifier les modèles de contrôle d'accès. Ces cadres sont comparés dans le tableau 2.8. C'est l'intersection entre les deux tableaux de synthèse représentée par la partie grisée de la figure 2.8.

		Modèle	**Identité**	DAC	MAC	**Treillis**	**Vues**	TMAC	**WRBAC**	TBAC	AMTD	**ORBAC**	RBAC	TRBAC	GTRBAC	CSRBAC	GEORBAC	ENVRBAC	**Rule-BAC**
Structuration	Concept	Rôles						●	●			●	●	●	●	●	●	●	○
		Données	●	●			●				●	●							○
		Flux						●	●	●									○
		Niveaux			●	●													○
		Multiples						○	○		○	●				○	○	○	●
	Hiérarchies	Limitées				●				●			●			●	●	●	○
		Générales							●		●	●	●	●		●	●	●	○
		Multiples										○	○		●		●	●	●
	Interdictions											●	●		○				●
	Formalisation	Principes	●	●	●	●	●		●		●	●	●	●		●	●		
		Structures	○	○	●	●	○		●		●	●	●	●		●	●		
		Intégrité	●	●	●	●	●					○	●						
Enrichissements	Contexte	Temps									●	○		●	●	○		○	●
		Espace										○				○	●	○	●
		Concept						●	●	●						●			●
		Libre										●						●	●
	Contraintes	Exclusions							●	●	●	●	●	●	●	●	●	●	
		Pré-requis							○				●	○	○				
		Libres							●	●	●	●						●	●
Administration	Modèle			○	○	○	○	○	●				●	●		●				
	Interrogation			○	○	○	○	○					○	●						●
	Fouille					○	○							●						
	Intégrité			●	●	●	●						○	●						○
	Graphique					○	○	●					○	●						
Cadre	C-DATALOG			○	○	○	○	○	○	○	○			●						●
	DATALOG$^{\neg}$			○	○	○	○						●	○						
	RT$_1^C$			○		○	○	○						●	●	○	○	○		●
	DATALOG^C			○		○	○	○						●	●	●	●	●	○	●
	DL			○	○	○	○		○	○	○	○		○						●
	D3LOG			○	○	○	○		○	○	○	○		○						●
	BINDER			○	○	○	○		○	○	○	○		○						●
	GT			○	○									●						
	ALLOY			○	○									●						
	DTAC			●	○	●	●		○	○	○	○		●						
	UML/OCL			○	○									●	○					
	LITHIUM													○	○	○				●
	CLP			○		○	○	○						●	●	○			○	○
	LTL			○		○	○	○						●	●					
	L. déontique			○		○	○	○					●	○						
	L. de description			○		○	○	○						●						
	DCC																			●
	ASL			○	○	○	○		○	○	○	○		○	○	○	○			●
Notre proposition				●	○	●	●	○	●	●	●	○	●	●	●	○	○	○	○	●

TAB. 2.7 – Essai de synthèse des modèles de contrôle d'accès

2.7.5.2 Critères de synthèse des cadres logiques

Plusieurs cadres ont été proposés pour formaliser et inférer sur les différents modèles de contrôle d'accès du tableau 2.7. Les tableaux 2.8 regroupent les cadres logiques en quatre en grandes familles :

- les six premières lignes rassemblent les modèles basés sur DATALOG,
- les quatre suivantes sont des paradigmes graphiques,
- les deux lignes suivantes concernent deux cadres de programmation logique,
- les trois lignes suivantes sont des propositions de logiques modales,
- les deux dernières lignes traitent des cadres DCC [Abadi07] et ASL [Jajodia01].

Les critères (en colonnes) pris en compte pour l'évaluation des modèles logiques proposés pour la formalisation du contrôle d'accès sont :

1. l'*expressivité* des langages, basée sur la comparaison des constructions de formules logiques autorisées,
2. les *outils* développés pour ces modèles, en comparant si des algorithmes d'interrogation, d'inférence ou de satisfaction ont été proposés. Notons que les problèmes de la satisfaction et de l'interrogation sont assez similaires : pour le premier on s'intéresse à l'existence d'une valuation alors qu'on les cherche toutes pour le second,
3. les *caractéristiques* des propositions, selon trois sous-critères :
   - que le cadre soit développé comme une réduction d'un autre modèle,
   - que le cadre se base sur les axiomes et règles d'inférences de la logique du premier ordre,
   - que le cadre sépare le modèle de l'instance,
4. les *applications* à la conception et à l'administration du contrôle d'accès proposées.

Cadre	Ref.	Modèles cibles	Fonctions	Contraintes corps	Contraintes tête	Négation	Existentiel	Multiples têtes	Constructeurs	Modes	Interrogation	Implication logique	Satisfaction	Réécriture	Règles inférence	Séparation modèle/instance	Intégrité des politiques	Conception de modèle	Évolution de modèle	Identification automatisée	Représentation
Basés sur DATALOG																					
C-DATALOG	[Bertino03]	Modèles multiples							●		●	●				●	○	○			
DATALOG$^{\neg}$	[Miège05]	ORBAC				●					●	●				●	●				
RT$_1^C$	[Li04b]	Trust management		●					●		●	●		●		●	○	○			
DATALOG^C	[Li03b]	Trust management		●							●	○	○			●	○	○			
DL	[Li03a]	Trust management		●					●		●	○	○	●		●	○	○			
D3LOG	[Jim01a]	Trust management		●					●		●	○	○	●		●	○	○			
BINDER	[DeTreville02]	Trust management		●					●		●	○	○	●		●	○	○			
Graphiques																					
GT	[Koch03]	RBAC				N/A							●				●				●
ALLOY	[Koch03]	RBAC				N/A							●				●				●
DTAC	[Tidswell01]	Modèles multiples				N/A							●				●	●	○		●
UML/OCL	[Sohr05]	RBAC				N/A							●	●		●	●				●
Programmation logique																					
LITHIUM	[Halpern03]	Non-spécifié		●		●					●	●	○			●	○	○			○
CLP	[Barker03]	TRBAC	●	●		●					●	●	○				○				
Basés sur PROLOG																					
LTL	[Hansen05]	RBAC		●	●					●	●		●	●	●						
L. déontique	[Abou El Kalam03]	ORBAC								●	●	●	●	●	●						
L. de description	[Liu05]	RBAC								●	●	●	●		●						
DCC	[Abadi07]	Non-spécifié	●	●	●				●			●	●		●			○			
ASL	[Jajodia01]	Modèles multiples		●							○	○	○				○	○			
Notre proposition		Modèles multiples		●	●		●	●			●	●	●			●	●	●	●	●	●

TAB. 2.8 – Essai de synthèse des cadres logiques

Beware of bugs in the above code ; I have only proved it correct, not tried it.

Donald Knuth – "Notes on the van Emde Boas construction of priority deques : An instructive use of recursion"

3

Choix d'un cadre logique

▷ *L'état de l'art a montré que de nombreux modèles ont été proposés pour organiser les droits des utilisateurs selon les besoins des organisations. La thèse ne vise pas à définir un nouveau modèle de contrôle d'accès mais à proposer une structuration logique qui permette de reprendre les éléments de modélisation existants en les généralisant (chapitre 4). L'objectif est de pouvoir construire des outils permettant la conception de nouveaux modèles et la vérification des politiques (chapitre 5).*

La réalisation de ces objectifs nécessite de choisir un cadre logique approprié homogène qui permet d'exprimer et d'inférer sur le plus grand nombre d'aspects présentés dans l'état de l'art. Le cadre que nous proposons est issu du modèle relationnel : c'est celui des dépendances de données*. Ce chapitre présente le cadre et l'illustre avec son application aux modèles* RBAC *qui est la famille de modèle la plus étudiée.*

Les contributions présentées dans ce chapitre concernent :

- *la classification en composants (noyau, principes et propriétés) des notions introduites dans la littérature sur les modèles de contrôle d'accès,*
- *la proposition d'un cadre logique pour l'étude des modèles et des politiques de contrôle d'accès,*
- *l'introduction de la notion d'intégrité des politiques de contrôle d'accès,*
- *l'analyse et la synthèse des développements successifs introduits dans les modèles* RBAC*,*
- *l'application du cadre logique pour la vérification des modèles* RBAC*.*

La définition des dépendances dans ce chapitre se base sur [Abiteboul95]. Plus précisement les chapitres huit (Functional and Join Dependencies) et dix (A larger Perspective). La notation utilisée et les définitions se réfèrent à l'annexe A. La formalisation logique des modèles RBAC *est donnée en annexe B. Une trace de l'inférence de la section 3.4.5 est proposée en annexe C.* ◁

Plan du chapitre

3.1 Critères pour un cadre logique

Le chapitre d'état de l'art a montré que plusieurs cadres issus de la logique ont été proposés pour formaliser le contrôle d'accès. Chacun mettant à profit l'expressivité offerte pour développer et enrichir les modèles. Les directions prises par ces recherches peuvent être organisées selon trois objectifs :

- répondre au besoin d'*expressivité* des modèles, qui conduit à des propositions théoriques riches, étendant DATALOG notamment,
- répondre au besoin de *vérification* des modèles et des politiques, qui conduit à l'utilisation d'outils de preuve à la décidabilité contrôlée permettant d'inférer sur les droits d'accès,
- répondre au besoin de *réalisation* du contrôle d'accès, qui conduit à privilégier les cadres disposant de mécanismes techniques nécessaires à une implémentation.

3.1.1 Critères fonctionnels pour le cadre logique

On retrouve ainsi une problématique de qualité, où une proposition de conception et d'administration des politiques doit satisfaire au mieux ces trois objectifs. Nous avons dû identifier un cadre théorique permettant de répondre au mieux aux objectifs fonctionnels suivants :

- *l'expression des entités et relations* introduits dans les différents modèles de contrôle d'accès par les administrateurs, pour répondre au mieux aux besoins des organisations,
- *l'expression des enrichissements des modèles* en permettant la modélisation de contraintes et d'aspects complexes,
- *de formalisation des principes et propriétés* que les modèles doivent respecter, dans un langage d'expression commun et sans ambiguïté,
- *la vérification* des politiques, le cadre doit être doté d'algorithmes permettant de vérifier efficacement que la politique est correcte,
- *l'implantation* dans les systèmes d'information, en particulier la possibilité de stocker et interroger efficacement les politiques,
- la conception d'*outils* d'aide à l'administration des politiques, en particulier la représentation graphique et l'aide à la conception.

3.1.2 Critères non fonctionnels pour le cadre logique

Le cadre que nous avons sélectionné répond à ces verrous scientifiques de modélisation et de formalisation du contrôle d'accès logique. Dans le développement de notre proposition, nous avons accordé une importance particulière à :

l'homogénéité : notre approche permet de répondre à de nombreuses problématiques qui surviennent tout au long du cycle de vie du contrôle d'accès, dans un cadre logique sous-jacent homogène,

le bon fondement : la proposition s'appuie sur les fondements logiques des bases de données et utilise des outils qui ont été développés pour répondre à des besoins de modélisation complexe,

l'utilisabilité : nous proposons des outils permettant d'automatiser une partie des tâches d'administration, utilisables par les acteurs de la sécurité des systèmes d'information.

Notre proposition est construite sur le modèle relationnel, et principalement avec les *dépendances de données* qui permettent de s'assurer que les données en extension (la politique elle-même) et en intention (les données déductibles à partir de la politique) sont cohérentes avec les entités du monde réel qu'elles représentent. Notre architecture logique est décomposable en trois niveaux :

- le modèle relationnel, sous-jacent aux dépendances et aux bases de données déductives, forme la couche la plus basse.
- le modèle DATALOG permet l'expression de règles de déduction de nouveaux faits en intention à partir de faits connus en extension.
- les dépendances de données, qui permettent d'exprimer des contraintes sur les données en intention ou en extension.

L'apport scientifique de notre proposition se démarque de celles présentées dans l'état de l'art par la prise en compte de l'*intégrité* des politiques. En effet, nous considérons qu'il est d'une grande importance de s'assurer que les droits des utilisateurs sont consistants et qu'aucune erreur n'est introduite dans les politiques. La notion d'intégrité est centrale dans la thèse.

3.1.3 Application aux modèles à rôles

Comme l'a montré l'état de l'art, Les modèles RBAC sont à la base de nombreux développements et forment un socle commun des notions introduites dans le contrôle d'accès. Ce chapitre présente les principes de notre proposition avec la structuration de la famille RBAC avec le cadre relationnel.

Nous allons successivement enrichir la modélisation basée sur les rôles pour prendre en compte les modèles étendant la famille de base et montrer quels sont les apports des dépendances. Ensuite, nous généraliserons cette approche à d'autres modèles de contrôle d'accès dans le chapitre 4.

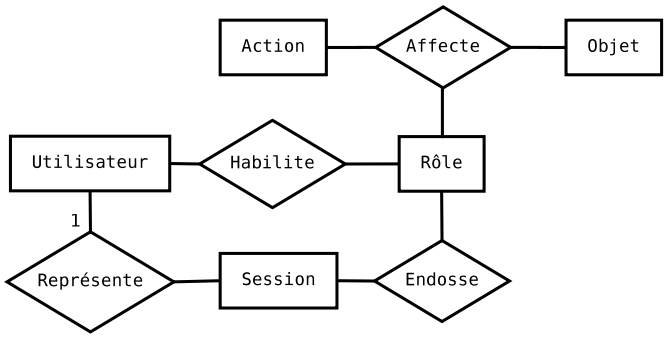


FIG. 3.1 – Modèle Entité-Relation de RBAC_0

3.2 Structuration des modèles

Notre proposition se base sur l'hypothèse qu'un modèle de contrôle d'accès est exprimable en terme de *concepts* et de *relations* entre ces concepts. Nous avons déjà introduit les notions d'utilisateurs, de sujets, d'objets et d'actions. L'étude des modèles de contrôle d'accès a montré que selon les modèles, plusieurs concepts intermédiaires peuvent être introduits, et que des relations supplémentaires permettent d'exprimer les hiérarchies et les contraintes de ces modèles.

3.2.1 Noyau des modèles de contrôle d'accès

Nous proposons d'utiliser le cadre des bases de données relationnelles, étendu par des règles de déductions et des dépendances de données. L'objectif est de représenter, d'intégrer et de vérifier les modèles et les politiques de contrôle d'accès. Le modèle relationnel est un socle sur lequel s'appuient les autres couches de la proposition.

Illustrons l'application du cadre relationnel à la représentation du modèle RBAC_0, dont le tableau 2.5 présente la caractérisation ensembliste. Une représentation du modèle RBAC_0 en Entité-Relation est donnée en figure 3.1. Nous avons choisi de représenter l'affectation de permissions aux rôles par une relation $Affecte$ ternaire. Dans le cadre relationnel nous notons $\mathbf{RBAC_0}$ la modélisation de RBAC_0. La structure de RBAC_0 est représentable par le schéma en extension de base de données $edb = \{Habilite(U,R), Affecte(R,A,O), Représente(S,U), Endosse(S,R)\}$. Ce schéma de base de données relationnelle exprime comment les droits dans le système sont structurés et enregistrés.

Une politique de contrôle d'accès organisée selon le modèle RBAC_0 est ainsi une instance $\mathbf{I}$ du schéma *edb* de $\mathbf{RBAC_0}$. Le tableau 3.1 est un exemple de politique où les droits statiques des utilisateurs correspondent à la matrice de contrôle d'accès du tableau 2.2. On remarque qu'un utilisateur peut être affecté à plusieurs rôles et qu'il peut endosser plusieurs sujets (sessions) [Sandhu96].

Relation *Habilite*

Utilisateur	Rôle
Alice	Infirmier
Alice	Médecin
Bob	Infirmier
Bob	Gastrologue
Charly	Infirmier
Charly	Pédiatre
Denise	Secrétaire

Relation *Endosse*

Sujet	Rôle
S1	Infirmier
S1	Médecin
S2	Infirmier
S3	Gastrologue
S4	Pédiatre
S5	Secrétaire

Relation $Affecte$

Rôle	Action	Objet
Infirmier	r	Fichier1
Infirmier	r	Fichier2
Infirmier	r	Fichier3
Médecin	w	Fichier1
Gastrologue	w	Fichier2
Gastrologue	r	Fichier4
Gastrologue	w	Fichier4
Gastrologue	x	Fichier4
Pédiatre	w	Fichier3
Pédiatre	r	Fichier4
Pédiatre	w	Fichier4
Pédiatre	x	Fichier4
Secrétaire	r	Fichier3
Secrétaire	r	Fichier4

Relation *Représente*

Sujet	Utilisateur
S1	Alice
S2	Bob
S3	Bob
S4	Charly
S5	Denise

TAB. 3.1 – Instance du schéma relationnel de $\mathbf{RBAC_0}$

3.2.2 Revue de politiques

Le langage des requêtes conjonctives formé sur les relations du schéma de $\mathbf{RBAC_0}$ permet ce qui est nommé *la revue des permissions* (*permissions review*), c'est-à-dire la possibilité d'interroger la politique de contrôle d'accès en vue de son administration [Ferraiolo03b]. Notons que pour l'instant, nous n'avons pas introduit la notion de hiérarchie de rôles. Les requêtes suivantes sont des exemples de revues :

1. l'ensemble des permissions attribuées aux utilisateurs par l'intermédiaire des rôles qui leurs sont attribués : $\{u, a, o \mid \exists r\ Habilite(u, r) \wedge Affecte(r, a, o)\}$,
2. l'ensemble des utilisateurs qui disposent dans une session de la permission d'écrire (r) : $\{u \mid \exists s, o, r\ Représente(s, r) \wedge Endosse(s, r) \wedge Affecte(r, \mathtt{r}, o)\}$,
3. l'ensemble des rôles qui disposent de permissions sur l'objet File4 et qui sont actuellement endossés au cours de sessions : $\{r \mid \exists a, s\ Affecte(r, a, \mathtt{Fichier4}) \wedge Endosse(s, r)\}$,
4. l'ensemble des utilisateurs auxquels le rôle Infirmier est attribué, ou simplement l'ensemble des infirmiers du système : $\{u \mid Habilite(u, \mathtt{Infirmier})\}$,
5. l'ensemble des permissions accordées au rôle Infirmier, ou simplement l'ensemble des droits des infirmiers du système : $\{a, o \mid Affecte(\mathtt{Infirmier}, a, o)\}$.

Les requêtes 4 et 5 en particulier sont du plus grand intérêt : elles correspondent aux applications *assigned_users*(`Infirmier`) et *assigned_perms*(`Infirmier`) définies en section 2.3.2. Nous pouvons nommer ces requêtes pour construire des vues *AssignedUsers* et *AssignedPerms* définies en intention. Le pouvoir d'expression des requêtes conjonctives permet de réaliser les fonctionnalités simples, mais incontournables, qu'un système de gestion de contrôle d'accès *doit* proposer.

Or, l'implantation des fonctionnalités de revue peut être délicate. Le standard RBAC indique que parmi l'ensemble des spécifications fonctionnelles qu'une implantation de RBAC doit comporter, seules les revues *assigned_users* et *assigned_perms* sont obligatoires [Ferraiolo03b]. Notre approche résout ce problème, non seulement pour les modèles RBAC, mais aussi pour tout modèle de contrôle d'accès dont la structure est exprimable dans le modèle relationnel. La citation suivante est extraite de [Ferraiolo03b] :

> *When URA and PRA relation instances have been created, it should be possible to view the contents of those relations from both the user and role perspectives[...]*
> *Since not all RBAC implementations provide facilities for viewing role, user, and session permissions or active roles for a session, these functions have been designated as optional or advanced functions in our requirement specification.*

Si le modèle relationnel est utilisable pour représenter les concepts et les relations des modèles de contrôle d'accès, il ne permet pas de modéliser leurs principes. Le modèle relationnel permet de représenter la *structuration* du modèle de contrôle d'accès : les concepts et relations qui le composent et qui permettent d'organiser les droits, mais n'exprime pas comment dériver les autorisations effectives à partir de la politique.

3.3 Modélisation des principes

À partir du schéma de la figure 3.1, comment exprimer la manière dont sont dérivées les autorisations des utilisateurs ? Comment également exprimer ce qu'est la relation d'héritage et les propriétés qu'elle doit respecter ? Comment exprimer des revues de permissions en présence de hiérarchie de rôles ? La modélisation de ces principes est effectuée à l'aide des modèles DATALOG et DATALOG^C présentés dans cette section.

3.3.1 Présentation

La construction des vues et des requêtes conjonctives est le premier pas vers le langage DATALOG. Ce dernier étend les requêtes conjonctives en introduisant la récursion. En effet, certaines requêtes d'intérêt ne sont pas exprimables par des requêtes conjonctives. Notons qu'un paradigme équivalent aux requêtes conjonctives munies de l'union est *nr*-DATALOG, c'est-à-dire DATALOG dépourvu de récursion.

Comme en témoigne le tableau de synthèse 2.8, l'application du modèle DATALOG aux problématiques du contrôle d'accès a été reconnue comme pertinente [Li03b] :

1. c'est un paradigme déclaratif exprimé dans un sous-ensemble de la logique du premier ordre, à la sémantique logique définie et non ambiguë,
2. il a été très étudié et étendu à la fois par les communautés de la programmation logique et des bases de données,
3. l'absence de fonctions permet de s'assurer de la décidabilité du modèle.

3.3.2 Définitions

Nous allons définir deux modèles principaux issus des bases de données : DATALOG et DATALOG^C qui étend le précédent. Nous ne couvrons pas les modèles basés sur DATALOG qui intègre la négation ou la disjonction.

3.3.2.1 Modèle DATALOG

Définition (Règle de déduction DATALOG [Abiteboul95])**.** *Une* règle de déduction DATALOG *est une expression de la forme :*

$$R_1(X_1), \ldots, R_n(X_n) \rightarrow R_0(X_0)$$

exprimée en logique du premier ordre par la clause de Horn équivalente :

$$\forall x_1, \ldots, x_n (R_1(X_1) \wedge \ldots \wedge R_n(X_n) \rightarrow R_0(X_0))$$

où $R_0, \ldots, R_n$ sont des symboles de prédicats, $X_0, \ldots, X_n$ sont des listes de termes d'arité correspondante dont $x_1, \ldots, x_n$ est l'ensemble des variables libres. Chaque variable apparaissant dans X_0 doit apparaître dans au moins un des $X_1, \ldots, X_n$ La tête *de la règle est l'expression $R_0(X_0)$, le* corps *$R_1(X_1), \ldots, R_n(X_n)$.*

Une règle est récursive *si une relation R apparaît dans la tête et dans le corps, non-récursive sinon. Une relation en* extension *n'apparaît* que *dans le corps des règles, une relation en* intention *apparaît dans la tête d'une règle au moins. Un ensemble de règles* DATALOG *est appelé un* programme DATALOG*.*

Nous remarquons que la présence des quantificateurs existentiels sur des variables en tête n'est pas autorisée : toute variable de la tête doit apparaître dans le corps. Le cadre logique DATALOG a été repris dans plusieurs approches d'étude du contrôle d'accès. DATALOG définit le paradigme des bases de données déductives, qui permettent l'expression de relations en intention récursives. Une base de données peut désormais être définie comme une collection de *faits* et de *règles*. Pour un programme DATALOG P, la base de données déductive est composée :

Exemple 3.1 Intérêt de la récursion

Prenons l'exemple de l'accessibilité des nœuds d'un graphe pour illustrer l'intérêt de la récursion dans les requêtes.

Supposons une relation R stockant l'ensemble des arêtes E d'un graphe dirigé $G = (V, E)$. Il est intéressant de savoir si à partir d'un nœud a, on peut atteindre un nœud b, où si depuis a on peut atteindre tous les autres nœuds.Ces requêtes faisant appel à la récursion, elles ne sont pas exprimables en calcul conjonctif : il faut étendre le langage de requêtes, c'est le but de DATALOG.

– d'un schéma en *intention*, noté *idb* pour *Intensional DataBase*,
– d'un schéma en *extension*, noté *edb* pour *Extensional DataBase*,
– le schéma, noté *sch* est l'ensemble $idb \cup edb$.

L'instance de la base de données en extension $\mathbf{I}$ est un ensemble de faits connus comme vrais. La base de données en intention $\mathbf{I}'$ est un ensemble faits *déduits* à partir de ceux connus et des règles de dérivation P. Trois approches *équivalentes* permettent de définir la *sémantique* d'un programme DATALOG P [Abiteboul95] :

1. comme un ensemble de *formules closes* de la logique, et de considérer les données déductibles comme une interprétation $\mathbf{I}'$ qui rende vraies les formules de P pour $\mathbf{I}$ donné. Comme plusieurs (voire une infinité) de modèles peuvent exister, on s'intéresse au *plus petit possible*, c'est-à-dire au modèle minimum de P (s'il existe). C'est la sémantique de la *théorie des modèles*,
2. comme un ensemble de *règles de production* qui produisent les propositions déductibles à partir de celles connues dans $\mathbf{I}$. C'est une approche de *preuve* sur des formules du premier ordre, où P produit des théorèmes en dérivant par application de règles des lemmes intermédiaires. Deux grandes stratégies sont l'évaluation *top-down* où l'on essaie de construire la preuve en résolvant des sous-buts successifs, ou *bottom-up* où l'on dérive successivement des théorèmes depuis les faits en vérifiant si le but appartient à cet ensemble de théorèmes déduits. C'est la sémantique de la *théorie de la preuve*,
3. comme la *solution d'une équation* du plus petit point fixe, cette approche procédurale consiste à appliquer successivement toutes les règles, considérées comme des opérateurs qui ajoutent des faits à la base de données à partir de ceux connus et prouvés. L'application des règles est itérée jusqu'à obtenir un point fixe : quand aucun nouveau fait n'est déductible, et ce point fixe est le plus petit possible. Cette approche est comparable à la stratégie *bottom-up* de preuve.

Techniquement, pour les approches (1) et (2), on fait correspondre à chaque $R(u)$ de l'instance de la base de données une règle de la forme $\rightarrow R(u)$, c'est-à-dire un *fait*, comme dans les programmes logiques PROLOG. Nous disposons de deux théorèmes :

Théorème (Équivalence des sémantiques DATALOG [Abiteboul95])**.** *Les sémantiques de* modèle*, de* preuve *et de* point fixe *des programmes* DATALOG *sont équivalentes.*

Théorème (Plus petit point fixe [Abiteboul95])**.** *Le plus petit point fixe d'un programme* DATALOG *existe, est unique et est calculable en un nombre fini d'étapes.*

En revanche, pour des extensions de DATALOG intégrant la disjonction ou la négation, la définition constructive des programmes DATALOG ne converge pas nécessairement vers un point fixe en un nombre fini d'étapes. C'est le cas général dans la logique du premier ordre autorisant les symboles de fonction. Nous ne nous intéresserons peu à ces modèles dont les procédures de preuve peuvent être coûteuses.

L'expressivité de DATALOG ne suffit pas toujours pour représenter certains aspects des modèles de contrôle d'accès. La prise en compte de certaines contraintes ou du contexte n'est pas possible. Afin de résoudre ce problème, il a été proposé d'étendre le fragment logique de DATALOG avec l'expression de contraintes arithmétiques portants sur des variables de l'hypothèse. L'objectif est de pouvoir modéliser uniformément des relations qualtitaives et surtout quantitatives sur des contextes représentés par des valeurs numériques [Damiani07, Joshi05].

3.3.2.2 Modèle DATALOG^C

Plusieurs extensions de DATALOG ont été proposées, nous avons abordé C-DATALOG, une extension orientée objet, DATALOG$^\neg$ qui permet l'expression de littéraux négatifs dans les règles, et DATALOG^C, qui permet l'expression de contraintes, exprimées sur un domaine de contraintes $(\mathcal{D}, \mathcal{L})$[1].

Définition (Règle de déduction DATALOG^C [Li03b])**.** *Une* règle de déduction DATALOG *avec contraintes, noté* DATALOG^C*, est une expression de la forme :*

$$R_1(X_1), \ldots, R_n(X_n), \psi_C(\tilde{X}) \rightarrow R(X)$$

exprimée en logique du premier ordre par la clause équivalente :

$$\forall x_1, \ldots, x_n \; R_1(X_1) \wedge \ldots \wedge R_n(X_n) \wedge \psi_C(\tilde{X}) \rightarrow R(X))$$

où $R, R_1, \ldots, R_n$ sont des symboles de prédicats, $X, X_1, \ldots, X_n$ sont des listes de termes d'arité correspondante dont $x_1, \ldots, x_n$ est l'ensemble des variables libres. $\psi_C(\tilde{X})$ est une conjonction de contraintes atomiques sur des variables apparaissant dans l'hypothèse. Chaque variable apparaissant dans X doit apparaître dans au moins un des $X_1, \ldots, X_n$ La tête *de la règle est l'expression $R(X)$, le* corps *$R_1(X_1), \ldots, R_n(X_n), \psi_C(\tilde{X})$.*

Ce paradigme généralise DATALOG en enrichissant le langage des règles. Le domaine de contraintes $(\mathcal{D}, \mathcal{L})$ paramètre les règles. La sémantique des programmes DATALOG^C étend celle de DATALOG en imposant que les contraintes des hypothèses soient également satisfaites.

[1] Nous laissons en suspens la structure $(\mathcal{D}, \mathcal{L})$ qui sera détaillée en section 3.4.1.3.

Exemple 3.2 Relations d'Allen avec DATALOG^C

Soit $Periode(ID, T^-, T^+)$ une relation en extension représentant des intervalles entres les instants T^- et T^+, où ID est clef de la relation. Les règles DATALOG^C permettent une définition logique des relations d'Allen :

Expression logique
$Periode(X, T_X^-, T_X^+), Periode(Y, T_Y^-, T_Y^+),$ $T_X^+ < T_Y^- \rightarrow Avant(X, Y)$ La période X est *avant* la période Y
$Periode(X, T_X^-, T_X^+), Periode(Y, T_Y^-, T_Y^+),$ $T_X^+ = T_Y^- \rightarrow Touche(X, Y)$ La période X *termine* quand la période Y *commence*
$Periode(X, T_X^-, T_X^+), Periode(Y, T_Y^-, T_Y^+),$ $T_X^+ > T_Y^- \rightarrow Chevauche(X, Y)$ La période X termine *après* que la période Y ai commencé
$Periode(X, T_X^-, T_X^+), Periode(Y, T_Y^-, T_Y^+),$ $T_X^- = T_Y^- \rightarrow Commence(X, Y)$ La période X *commence en même temps* que la période Y
$Periode(X, T_X^-, T_X^+), Periode(Y, T_Y^-, T_Y^+),$ $T_X^- \geq T_Y^-, T_X^+ \leq T_Y^+ \rightarrow Pendant(X, Y)$ La période X se déroule *pendant* la période Y
$Periode(X, T_X^-, T_X^+), Periode(Y, T_Y^-, T_Y^+),$ $T_X^+ = T_Y^+ \rightarrow Termine(X, Y)$ La période X se *termine en même temps* que la période Y
$Periode(X, T_X^-, T_X^+), Periode(Y, T_Y^-, T_Y^+),$ $T_X^- = T_Y^-, T_X^+ = T_Y^+ \rightarrow Egal(X, Y)$ Les périodes X et Y *commencent et terminent* en même temps

Pour s'assurer qu'une base de données utilisant les relations d'Allen soit intègre, il faut s'assurer que l'attribut ID soit une clef primaire de la relation $Periode(X, T^-, T^+)$, et que l'heure de début d'un intervalle soit bien antérieure à son heure de fin ($T^- \leq T^+$).

Les dépendances de données permettent d'imposer de telles contraintes d'intégrité qui définissent les états valides de la relation $Periode(ID, T^-, T^+)$. À titre d'exemple, nous donnons les deux dépendances génératrices de contraintes suivantes qui expriment ces contraintes :

$$\begin{aligned} Periode(X, T_X^-, T_X^+), Periode(X, T_Y^-, T_Y^+) &\rightarrow T_X^- = T_Y^-, T_X^+ = T_Y^+ \\ Periode(X, T^-, T^+) &\rightarrow T^- \leq T^+ \end{aligned}$$

	Expression logique
τ_1	$\forall S, R, A, O \quad Endosse(S, R), Affecte(R, A, O) \rightarrow Accès(S, A, O)$
τ_2	$\forall U, R, A, O \; Habilite(U, R), Affecte(R, A, O) \rightarrow Statique(U, A, O)$
τ_3	$\forall U, S, A, O \quad Représente(S, U), Accès(S, A, O) \rightarrow Dynamique(U, A, O)$

TAB. 3.2 – Définitions logiques des triplets d'autorisations du modèle RBAC_0

DATALOG^C a été mis en œuvre pour la modélisation du contexte dans les modèles de contrôle d'accès [Li03b, Miège05]. En effet, les contraintes en hypothèses permettent de définir des relations complexes à partir de valeurs numériques. Par exemple, à l'aide de dates on peut définir la notion d'intervalle temporel, puis à l'aide de contraintes sur les dates on peut construire les relations d'Allen sur les intervalles et les utiliser dans la définition de contextes [Bertino01, Joshi03]. De la même façon, les contraintes peuvent être utilisées pour modéliser des relations topologiques entre surfaces en utilisant des coordonnées spatiales [Damiani07].

3.3.3 Déduction dans les modèles à rôles

Tous les modèles de contrôle d'accès ont été développés dans les mêmes buts : organiser les droits des utilisateurs et permettre aux administrateurs de gérer ces droits. Dans la section précédente, nous avons présenté comment des schémas relationnels peuvent exprimer des modèles de contrôle d'accès, avec comme exemple la modélisation de RBAC_0. Cependant, nous n'avons fait que *structurer* les données de sécurité qui forment la politique, mais nous n'avons pas encore exprimé à quoi elles servent ni comment dériver les autorisations à partir des politiques.

3.3.3.1 Triplets d'autorisations

Quand on analyse le rôle du moniteur de référence, c'est-à-dire le programme mis en œuvre entre sujets et objets pour assurer le contrôle d'accès, on remarque qu'il peut être assimilé à une application $\mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow \{vrai, faux\}$, qui à toute demande d'accès faite par un sujet va faire correspondre une décision binaire d'autorisation. De façon équivalente, le moniteur va vérifier qu'un triplet (s, a, o) appartient bien à l'ensemble des autorisations *Accès* qui est un sous-ensemble de $\mathcal{S} \times \mathcal{A} \times \mathcal{O}$.

Nous retrouvons la notion de relation du modèle relationnel, mais l'ensemble des actions autorisées n'est pas défini en *extension* mais en *intention*. En particulier dans le modèle RBAC_0 trois triplets d'autorisations sont d'intérêt Ces trois relations sont introduites ici pour le modèle RBAC_0 mais elles sont génériques : nous les définirons et les utiliserons pour d'autres modèles de contrôle d'accès. Le tableau 3.2 donne les expressions logiques de trois triplets d'autorisation :

1. le triplet fondamental d'autorisation $Accès \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{O}$: l'ensemble des permissions accordées aux *sujets* par l'intermédiaire des rôles. C'est sur ce triplet que le moniteur de contrôle d'accès prend les décisions,
2. le triplet d'autorisation statique orienté utilisateurs $Statique \subseteq \mathcal{U} \times \mathcal{A} \times \mathcal{O}$: l'ensemble des permissions accordées aux *utilisateurs* par l'intermédiaire des rôles attribués. Il s'agit d'autorisations qui sont invariantes quel que soit l'état d'exécution du système,
3. le triplet d'autorisation dynamique orienté utilisateurs $Dynamique \subseteq \mathcal{U} \times \mathcal{A} \times \mathcal{O}$: l'ensemble des permissions accordées aux utilisateurs par l'*intermédiaire* les sujets qui les représentent dans le système.

Exemple 3.3 Les autorisations de `Bob`

Considérons l'utilisateur `Bob` de la politique du tableau 3.1. Deux rôles sont attribués à cet utilisateur `Infirmier` et `Gastrologue`. Dans un modèle RBAC, `Bob` peut choisir d'endosser tout ou partie des rôles qui lui sont attribués, dans une ou plusieurs sessions qui le représentent dans le système

À l'instant présent représenté la politique du tableau 3.1, `Bob` agit dans le système par l'intermédiaire de deux sessions ouvertes en parallèle `S2` et `S3` :
– dans la session `S2`, `Bob` endosse le rôle `Infirmier`,
– dans la session `S3`, `Bob` endosse le rôle `Gastrologue`.

On peut s'intéresser aux permissions attribuées :
– à `Bob` *quelles que soient* les sessions en cours. On peut effectuer une requête sur la relation *Statique* de la forme $\{a, o \mid Statique(\texttt{Bob}, a, o)\}$ pour déterminer cet ensemble de permissions,
– à chacune des deux sessions `S2` et `S3` que `Bob` a ouvert $\{a, o \mid Accès(\texttt{S2}, a, o)\}$ pour déterminer les permissions dont `Bob` dispose dans la session `S2`,
– à `Bob` à *l'instant présent*, c'est-à-dire à l'union des permissions autorisées dans chacune des sessions qu'il endosse. La requête $\{a, o \mid Dynamique(\texttt{Bob}, a, o)\}$ est ainsi exprimable comme $\{a, o \mid Accès(\texttt{S2}, a, o)\} \cup \{a, o \mid Accès(\texttt{S3}, a, o)\}$.

Les autorisations statiques dont `Bob` par l'intermédiaire du rôle `Infirmier` sont grisées dans le tableau 3.3.

À partir d'une politique en extension comme celle du tableau 3.1 et des définitions du tableau 3.2, on peut déterminer les triplets d'autorisations. Les formalisations sont proposées à partir du standard RBAC [Ferraiolo03b]. La relation *Statique* déductible à partir de la politique jouet est celle présentée par le tableau 3.3. Remarquons que les formes syntaxiques du tableau 3.2 peuvent être considérées :

– soit comme des *vues* sur le schéma *edb* de $\mathbf{RBAC_0}$, c'est-à-dire des requêtes conjonctives auxquelles un nom a été donné,
– soit comme des *règles de déduction*, qui expriment comment à partir du schéma *edb* construire les relations *Statique*, *Dynamique* et *Accès* qui forment le schéma en intention *idb*,

– soit comme des *contraintes* sur la politique, qui imposent la présence de tuples dans les relations *Statique*, *Dynamique* et *Accès*. Cette dernière interprétation est la sémantique des dépendances de données, présentée en section 3.4.1.5.

Utilisateur	Action	Objet
Alice	r	Fichier1
Alice	r	Fichier2
Alice	r	Fichier3
Alice	w	Fichier1
Bob	r	Fichier1
Bob	r	Fichier2
Bob	r	Fichier3
Bob	r	Fichier4
Bob	w	Fichier2
Bob	w	Fichier4
Bob	x	Fichier4

Utilisateur	Action	Objet
Charly	r	Fichier1
Charly	r	Fichier2
Charly	r	Fichier3
Charly	r	Fichier4
Charly	w	Fichier3
Charly	w	Fichier4
Charly	x	Fichier4
Denise	r	Fichier3
Denise	r	Fichier4

TAB. 3.3 – Instance de la relation en intention *Statique*

3.3.3.2 Hiérarchisation

Nous avons introduit les bases des modèles RBAC avec la structuration relationnelle **RBAC$_0$** et le schéma *edb*. Nous considérons maintenant le modèle RBAC$_1$, qui ajoute au modèle RBAC$_0$ la hiérarchisation des rôles. Soit $Domine(R_1, R_2)$ la relation de domination directe entre rôles, soit en d'autres termes l'ensemble des relations d'héritage *directes* définies dans la politique, et $Hérite(R_1, R_2)$ sa couverture réflexive et transitive. D'après les définitions données en section 2.3.2, *Domine* correspond à la formalisation logique de la relation $\succ\!\!\succ$ et *Hérite* correspond à celle de $\succeq$.

La relation en extension *Domine* doit être ajoutée au schéma de RBAC$_0$ pour former RBAC$_1$. Les définitions logiques des triplets d'autorisations doivent être reprises en conséquence pour intégrer la hiérarchisation des rôles dans leurs dérivations à l'aide du prédicat *Hérite*.

Si nous choisissons de représenter la hiérarchie par un ordre partiel entre les rôles, les propriétés attendues de la relation d'héritage, présentées par le tableau 3.4, sont :

– elle est *construite* à partir de la relation de domination directe (ρ_0),
– elle est *transitive* : si r_1 domine r_2 et r_2 hérite de r_3 alors r_1 domine r_3 (ρ_1),
– elle est *réflexive* : un rôle r se domine lui-même (ρ_2),
– elle est *antisymétrique* : si r_1 domine r_2 et r_2 domine r_1 alors $r_1 = r_2$ (ρ').

Selon les types de hiérarchies définies dans les modèles, leurs propriétés peuvent varier : pour modéliser une hiérarchie limitée RBAC$_1$ on ajouterait ou modifierait des propriétés au tableau 3.4. Conceptuellement, alors que la transitivité et la réflexivité *génèrent* des relations à partir de celles connues, l'antisymétrie est une *contrainte* sur les valeurs autorisées, qui n'est pas exprimable en DATALOG.

	Expression logique
ρ_0	$Domine(R_1, R_2) \rightarrow Hérite(R_1, R_2)$
ρ_1	$Domine(R_1, R_2), Hérite(R_2, R_3) \rightarrow Hérite(R_1, R_3)$
ρ_2	$Role(R) \rightarrow Domine(R, R)$
ρ'	$Hérite(R_1, R_2), Hérite(R_2, R_1) \rightarrow R_2 = R_1$
ρ''	$Hérite(R_1, R_2), Hérite(R_2, R_1), R_2 \neq R_1 \rightarrow \bot$

TAB. 3.4 – Formalisation logique de l'héritage entre rôles du modèle RBAC_1

Les trois premières expressions du tableau 3.4 ont été reprises de [Barker03] où des modèles à rôles ont été logiquement formalisés dans un fragment de logique proche de DATALOG^C. En revanche, l'expression ρ' du tableau 3.4 n'est pas une règle DATALOG, ni même une règle DATALOG^C. Elle doit être transformée en une expression de la forme ρ'' dans ces modèles. Or cette transformation où les contraintes de la conclusion sont « déplacées » en hypothèse n'est pas possible si des variables sont quantifiées existentiellement dans la conclusion.

De plus, pour respecter les définitions proposées des modèles RBAC, on souhaiterait pouvoir exprimer certaines restrictions sur les relations du schéma *edb* de $\mathbf{RBAC_0}$. En effet, les modèles RBAC imposent qu'à *une session donnée* ne soit associé qu'*un seul et unique* utilisateur. Il s'avère que dans l'instance du tableau 3.1 c'est effectivement le cas, mais rien ne garantit formellement que cette propriété est vérifiée par toute politique.

De même, il est nécessaire d'imposer que les rôles endossés dans une session soient un sous-ensemble des rôles attribués à l'utilisateur que la session représente dans le système. Cette propriété est représentable à l'aide d'une dépendance : un outil qui permet d'imposer des restrictions sur les instances valides d'une base de données qui ne sont pas toutes exprimables en DATALOG ni même en DATALOG^C.

3.4 Intégrité des politiques de contrôle d'accès

3.4.1 Dépendances de données

Les *dépendances de données*, ou *contraintes d'intégrité*, sont un des piliers de la théorie des bases de données relationnelles : les dépendances sont des propriétés devant être satisfaites *par toutes les instances d'une base de données*. Nous préférons utiliser le terme de *dépendance* pour éviter une possible confusion due à la polysémie du terme *contrainte*. Les classes de dépendances les plus étudiées et utilisées en pratique dans les systèmes de gestion de bases de données sont (tableau 3.5) :

- les dépendances *fonctionnelles* qui *généralisent la notion de clef primaire*, elles imposent l'égalité de certaines données en fonction d'autres. Par exemple, la dépendance (1) du tableau 3.5 indique que l'attribut ID de la relation *Employe définit fonctionnellement* les attributs *Nom* et *Prenom* : deux tuples de la relation *Employe* ayant le même identifiant doivent avoir le même nom et le même prénom. Cette dépendance exprime le fait que l'attribut ID est un identifiant unique à partir duquel on peut déduire toutes les informations concernant un employé,
- les dépendances d'*inclusion qui généralisent la notion de clef étrangère*, elles imposent la présence de tuples en fonction d'autres. La dépendance (2) du tableau 3.5 indique que l'attribut ID de la relation *Responsable* est inclus dans l'attribut ID de la relation *Employe* : elle signifie qu'un « responsable » est aussi un « employé ».

(1)	$\forall ID, Nom_1, Prenom_1, Nom_2, Prenom_2$ $Employe(ID, Nom_1, Prenom_1) \wedge Employe(ID, Nom_2, Prenom_2)$ $\rightarrow Nom_1 = Nom_2 \wedge Prenom_1 = Prenom_2$
(2)	$\forall ID, IDEquipe, Fonction$ $Responsable(ID, IDEquipe, Fonction)$ $\rightarrow \exists Nom, Prenom\ employe(ID, Nom, Prenom)$

TAB. 3.5 – Exemple de dépendances fonctionnelle et d'inclusion

Les classes de dépendances les plus étudiées disposent d'une notation propre. Une dépendance fonctionnelle est notée par une expression de la forme $R : X \rightarrow Y$ pour exprimer que dans la relation R, l'ensemble d'attributs X détermine fonctionnellement l'ensemble d'attributs Y. Depuis les dépendances introduites dans l'article fondateur de Codd [Codd70], la littérature a proposé une grande variété de dépendances regroupées en *classes*, répondant au besoin d'expressivité nécessaire pour modéliser des situations complexes.

3.4.1.1 Classes de dépendances

Les classe de dépendances ont successivement été introduites comme des fragments de la logique autorisants des ensembles d'expressions de plus en plus grands. Nous avons tâché, dans la mesure du possible, d'organiser les classes selon cette expressivité. Le tableau 3.6 présente les fragments logiques des principales classes de dépendances :

1. *fonctionnelles*, *Functional Dependencies* (FD), expression (1) du tableau 3.5,
2. *d'inclusion*, *Inclusion Dependencies* (IND), expression (2) du tableau 3.5,
3. *génératrices d'égalités*, *Equality-Generating Dependencies* (EGD) [Beeri84] généralisent les FD en autorisant plusieurs relations en hypothèse,

4. *génératrices de contraintes*, *Constraint-Generating Dependencies* (CGD) [Baudinet95] généralisent les dépendances génératrices d'égalités en autorisant d'autres relations que la seule égalité entre termes,
5. *génératrices de nullité*, *Nullity-Generating Dependencies* (NGD), appelées aussi ou *implication constraints* [Zhang97] ou encore *denial constraints* [Godfrey98] sont une forme particulière des CGD où toutes les contraintes sont présentes en hypothèse. Il s'agit d'une classe très utilisée et générale,
6. *multivaluées*, *Multivalued Dependencies* (MVD) [Abiteboul95] disposent également d'une notation propre, elles généralisent les dépendances fonctionnelles en exprimant « l'orthogonalité » d'attributs d'une relation,
7. *de jointure*, *Join Dependencies* (JD) [Fagin86] généralisent les dépendances multivaluées et expriment qu'un relation soit définissable comme la jointure naturelle d'autres relations,
8. *génératrices de tuples totales*, *Full Tuple-Generating Dependencies* (FTGD) ou *Total Tuple-Generating Dependencies* (TTGD) [Beeri84] ne comportent pas de variables quantifiées existentiellement, elles généralisent les dépendances de jointure,
9. *génératrices de tuples*, *Tuple-Generating Dependencies* (TGD) ou dépendances généralisées *generalized dependencies* [Beeri84] autorisent la quantification existentielle de variables de la conclusion,
10. *génératrices de tuples contraintes*, *Constrained Tuple-Generating Dependencies* (CTGD) [Maher96] étendent les dépendances génératrices de tuples en ajoutant des contraintes en hypothèse et en conclusion,
11. *génératrices de tuples contraintes avec disjonctions*, *Disjunctive Constrained Tuple-Generating Dependencies* (DCTGD) [Wang02] étendent les dépendances génératrices de tuples contraintes en permettant l'expression de disjonctions en conclusion. C'est une des classes les plus générales proposées à ce jour.

Les classes de dépendances sont organisables en catégories non exclusives :

génératrices de tuples ou de contraintes :
- des classes, comme les dépendances d'inclusion, imposent *l'existence* de certaines données en fonction d'autres, elles sont regroupées sous le terme générique de dépendances *génératrices de tuples*,
- d'autres, comme les dépendances fonctionnelles, *limitent* les valeurs possibles des tuples en fonction d'autres, elles sont dites *génératrices de contraintes*,

totales contre partielles :
- les classes dont *toutes* les variables sont quantifiées *universellement* sont *totales*, comme les dépendances multivaluées ou fonctionnelles,
- d'autres autorisent des variables quantifiées *existentiellement* en conclusion, en particulier les dépendances d'inclusion ou génératrices de tuples.

Réf.	Acronyme	Expression en logique
(1)	FD	$\forall \tilde{X}\ R(X_1, X_2), R(X_1, X_2') \rightarrow X_2 = X_2'$
(2)	IND	$\forall \tilde{X}\ R(X) \rightarrow \exists \tilde{Z}\ Q(Y)$
(3)	EGD	$\forall \tilde{X}\ R_1(X_1), \ldots, R_n(X_n) \rightarrow x_i = x_j, \ldots, x_p = x_k$
(4)	CGD	$\forall \tilde{X}\ R_1(X_1), \ldots, R_n(X_n), \phi(\tilde{X}) \rightarrow \psi(\tilde{X})$
(5)	NGD	$\forall \tilde{X}\ R_1(X_1), \ldots, R_n(X_n), \phi(\tilde{X}) \rightarrow \perp$
(6)	MVD	$\forall \tilde{X}\ R(X, X_1, X_1'), R(X, X_2, X_2') \rightarrow R(X, X_2, X_1')$
(7)	JD	$\forall \tilde{X}\ R_1(X_1), \ldots, R_n(X_n) \rightarrow R(Y)$
(8)	TTGD	$\forall \tilde{X}\ R_1(X_1), \ldots, R_n(X_n) \rightarrow Q_1(Y_1), \ldots, Q_m(Y_m)$
(9)	TGD	$\forall \tilde{X}\ R_1(X_1), \ldots, R_n(X_n) \rightarrow \exists \tilde{Z}\ Q_1(Y_1), \ldots, Q_m(Y_m)$
(10)	CTGD	$\forall \tilde{X}\ R_1(X_1), \ldots, R_n(X_n), \phi(\tilde{X}) \rightarrow \exists \tilde{Z}\ Q_1(Y_1), \ldots, Q_m(Y_m), \psi(\tilde{Y})$
(11)	DCTGD	$\forall \tilde{X}\ R_1(X_1), \ldots, R_n(X_n), \phi(X) \rightarrow \bigvee_{i=1}^{p} [\exists \tilde{Z}^i\ Q_1^i(Y_1^i), \ldots, q_m^i(Y_m^i), \psi^i(Y^i)]$

TAB. 3.6 – Formes syntaxiques des principales classes de dépendances

3.4.1.2 Formes syntaxiques

Les classes de dépendances sont exprimables en sous-ensembles restreints de formules de la logique du premier ordre. Ce formalisme permet d'exprimer dans un même langage la grande majorité des dépendances étudiées sous forme de formules à la syntaxe particulière [Abiteboul95]. Le tableau 3.6 donne la forme générale des classes de dépendances avec leurs acronymes anglais respectifs en utilisant les conventions :

- la virgule « , » est utilisée à la place du connecteur logique $\wedge$,
- les R_i, Q_j sont des symboles de relations sur un schéma de base de données $\mathbf{R}$,
- les x_i, y_j sont des termes : des constantes ou des variables,
- les X_i, Y_j sont des listes de termes d'arité correspondante aux prédicats,
- l'ensemble des variables apparaissant dans une liste de termes X est noté $\tilde{X}$,
- $\tilde{Z} = \tilde{X} - \tilde{Y}$ ne désigne non pas toutes les variables de la conclusion de la dépendance, mais *uniquement* celles qui ne sont pas présentes dans l'hypothèse,
- les variables $\tilde{X}$ sont quantifiées *universellement*,
- les variables $\tilde{Z}$ sont quantifiées *existentiellement*,
- $\phi(\tilde{X})$ est une conjonction de contraintes sur des variables de l'hypothèse,
- $\psi(\tilde{Y})$ est une conjonction de contraintes sur des variables de la conclusion.,
- pour $X = x_1, \ldots, x_n$ et $Y = y_1, \ldots, y_n$, $X = Y$ est un abus de notation pour une expression de la forme $x_1 = y_1 \wedge \ldots \wedge x_n = y_n$,
- les constantes sont écrites en fonte à taille de `caractère` fixe,
- les identifiants de termes commençants par une majuscule sont des variables.

Nous remarquons que, quelle que soit la classe considérée, une dépendance est représentable par une formule logique de la forme $\phi \rightarrow \psi$, où ϕ est appelé l'*hypothèse* ou *corps* et ψ la *conclusion* ou *tête*. On remarque ainsi que ces expressions sont très proches des règles DATALOG. Dans les fragments de logique des dépendances :

- les expressions sont toutes *closes*, aucune variable n'est libre,
- les expressions ne comportent *pas de négation*, ni dans la tête, ni dans le corps,
- un prédicat particulier $\perp$ est utilisé : une *antilogie* toujours évaluée à $faux$,
- les fonctions ne sont *pas autorisées* dans la construction des termes,
- il n'y a qu'*une seule* implication par expression,
- seule une classe autorise la présence de *disjonction*, et seulement dans la tête.

3.4.1.3 Domaine de contraintes

Les conjonctions de contraintes $\phi(\tilde{X})$ et $\psi(\tilde{Y})$ sont des formules du premier ordre exprimées dans un langage de contraintes $\mathcal{L}$ n'utilisant que le seul connecteur $\wedge$, pas de fonctions et un nombre fini de symboles relations, disjoints des symboles de relations de la base de données. Le langage $\mathcal{L}$ permet l'expression de relations prédéfinies (*built-in*) entre des termes apparaissant dans les dépendances.

Ces relations particulières sont évaluables indépendamment des données enregistrées dans le système [Maher00]. Les relations arithmétiques utilisées dans les clauses `where` du langage SQL sont de telles relations *built-in* dont la sémantique est implémentée directement dans le moteur d'un système de gestion de bases de données.

La vérité des formules de $\mathcal{L}$ est donnée par une interprétation $\mathcal{D}$, dont le domaine du discours peut être $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$ ou les domaines finis $\mathcal{FD}$. $\mathcal{D}$ est généralement une structure munie des opérations et relations habituelles sur le domaine considéré, comme l'arithmétique sur les entiers $(\mathbb{N}, 0, S, +, \leq, =)$. Les algorithmes qui permettent de déterminer la valider des expressions de $\mathcal{L}$ dans $\mathcal{D}$ sont implémentés sous la forme de *solver* de contraintes.

Le couple $(\mathcal{D}, \mathcal{L})$ est appelé *domaine de contraintes* [Lassez89]. Les dépendances introduisant des contraintes sont paramétrées par ce domaine, comme le sont les bases de contraintes (*Constraint Databases*) [Revesz95] ou les règles DATALOG^{C}. Les programmes logiques par contraintes (CLP) également sont paramétrées par un domaine de contraintes [Jaffar94]. On note CLP($\mathbb{N}$), CLP($\mathbb{Q}$), CLP($\mathbb{R}$) et CLP($\mathcal{FD}$) les programmes logiques par contraintes sur respectivement $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$ ou $\mathcal{FD}$. Les dépendances sont définies quelle que soit la structure $\mathcal{D}$, pourvu que le test de satisfaction et d'implication logique entre des formules de $\mathcal{L}$ soit décidable dans $\mathcal{D}$ [Baudinet95, Maher97].

Les domaines de contraintes les plus courants sont les égalités et inégalités sur les entiers, sur les rationnels, sur les réels ou les contraintes linéaires arithmétiques (L_n proposé en annexe A.1.2). Le besoin d'utiliser de telles structures est apparu avec l'émergence de nouveaux paradigmes de bases de données, comme que les bases de contraintes, spatiales, temporelle, les systèmes d'information géographiques ou multimédias [Wang02].

3.4.1.4 Cadre logique composé

Les classes de dépendances forment un cadre logique composé de fragments de la logique de plus en plus grands. Le tableau 3.6 illustre, d'une part la richesse du domaine mais surtout la tendance générale vers l'expressivité des *classes de dépendances* à laquelle nous allons faire correspondre l'expressivité des *modèles de contrôles d'accès* de plus en plus grande.

Ainsi, selon que l'on souhaite modéliser une propriété d'un modèle de contrôle d'accès « simple », nous utiliserons une dépendance restrictive, ou bien pour une propriété d'un modèle plus riche, nous utiliserons une dépendance d'une classe plus expressive. Le problème peut également être abordé dans la perspective inverse. Disposant d'un fragment de logique défini, il s'agit alors de déterminer quelles sont les propriétés des modèles de contrôle d'accès modélisable. Le cadre « à tiroirs » des dépendances permet ainsi de contrôler l'expressivité, la décidabilité et la complexité de la formalisation logique du contrôle d'accès.

D'après la définition des langages qui permettent l'expression respectivement des vues ($\mathcal{L}_{vues}$), des règles DATALOG ($\mathcal{L}_{DATALOG}$), des règles DATALOG^C ($\mathcal{L}_{DATALOG^C}$) et des dépendances génératrices de tuples contraintes ($\mathcal{L}_{CTGD}$), on a l'inclusion suivante :

$$\mathcal{L}_{vues} \subseteq \mathcal{L}_{DATALOG} \subseteq \mathcal{L}_{DATALOG^C} \subseteq \mathcal{L}_{CTGD}$$

L'interprétation que nous donnons aux formules exprimées dans ces différents langages est la même : les trois considérations ont la même sémantique logique. Ceci nous permet de considérer les règles d'un programme DATALOG ou DATALOG^C comme des dépendances satisfaites par définition par l'interprétation du programme. En revanche l'utilisation qui est faite des règles et des dépendances est différente.

3.4.1.5 Sémantique des dépendances

Les dépendances permettent de *restreindre* les valeurs autorisées des instances d'une base de données : plus la classe est expressive, plus les restrictions exprimables le sont également. La forme $\phi \rightarrow \psi$ commune à toutes les dépendances permet de donner une interprétation proche de celle des règles DATALOG : « si ϕ est vraie dans une instance de la base de données, alors ψ doit l'être aussi ».

Cette sémantique est néanmoins différente, car elle ne considère pas les dépendances comme des règles de *production* mais comme des *restrictions*. Pour les dépendances génératrices de nullité par exemple, on peut interpréter $\phi \rightarrow \perp$ comme « si ϕ est évaluée à vrai, alors la base de données est inconsistante ». La *sémantique* des dépendances est exprimable grâce à la sémantique de la logique du premier ordre présentée en section A.1.2.

Définition (Satisfaction de dépendances)**.** *Une dépendance ϕ de données est* satisfaite *par une instance de base de données* **I** *si et seulement si* **I** *est un modèle de ϕ, noté* $\mathbf{I} \models \phi$.

Un ensemble de dépendances Φ *est* satisfait *par une instance de base de données* **I** *si pour tout* ϕ *appartenant à* Φ, **I** *est un modèle de* ϕ, *noté* $\mathbf{I} \models \Phi$.

Un ensemble de dépendances ϕ implique logiquement *une dépendance* ϕ *si tout modèle de* Φ *est aussi un modèle de* ϕ, *noté* $\Phi \models \phi$.

L'introduction des règles DATALOG puis des dépendances de données a étendu le modèles relationnel pour aboutir à ce que d'aucuns nomment les bases de données *étendues* [Maher00]. Une structure relationnelle (ou base de données) **BD** est désormais composée de [Godfrey98] :

- un ensemble de relations en extension : le schéma *edb* qui donne la structure de stockage et l'instance **I** représente tout les faits connus, c'est le contenu de la base de données. **I** est un modèle de Herbrand du langage de requête de *edb*,
- un ensemble de relations en intention : un ensemble de règles DATALOG P qui définissent le schéma *idb*, en exprimant comment dériver des faits nouveaux à partir de la base en extension. Il existe un modèle $\mathbf{I}'$ contenant **I** qui soit le plus petit modèle de Herbrand de P [Abiteboul95],
- un ensemble de dépendances de données Σ : des restrictions exprimées en logique du premier ordre par un ensemble de formules conformes aux restrictions syntaxiques du tableau 3.6. Les dépendances limitent les instances autorisées, $\mathbf{I}'$ doit être un modèle de Σ pour être intègre.

3.4.2 Preuve de l'implication

Un des aspects théoriques relatifs aux dépendances le plus étudié est celui de l'*implication logique* : étant donnée un ensemble de dépendances Σ et une dépendance σ, σ est-elle satisfaite pour toute instance qui satisfait Σ, noté $\Sigma \models \sigma$? En d'autres termes, tout modèle de Σ est-il aussi un modèle de σ ? Le problème de l'implication s'avère être un problème incontournable lors de l'utilisation des dépendances :

- l'optimisation sémantique de requêtes, il s'agit d'améliorer l'efficacité de l'évaluation des requêtes en éliminant les jointures ou sélections qui sont toujours satisfaites [Godfrey98],
- l'équivalence de requêtes, qui permet de savoir si les résultats des évaluations de deux requêtes sont identiques quelque soit l'instance de base de données [Abiteboul95],
- la maintenance des vues matérialisées lors de la mise à jour des données sources [Gupta95],
- la conception de schéma et en particulier la décomposition de relations et la mise sous forme normale, qui permet d'éviter les anomalies de mise à jour et la redondance de données [Abiteboul95],
- la correspondance de schémas dans le cadre de l'intégration de données de schémas hétérogènes [Fagin06].

Deux approches ont conduit à des algorithmes de décision de l'implication logique. La première est l'*axiomatisation* des classes de dépendances, l'autre est celle de la construction de procédures de *preuve* ou de *décision* de l'implication.

Propriété	**Axiome**
réflexivité	si $Y \subseteq X$ alors $X \rightarrow Y$
augmentation	si $X \rightarrow Y$ alors $XZ \rightarrow YZ$
transitivité	si $X \rightarrow Y$ et $Y \rightarrow Z$ alors $X \rightarrow Z$

TAB. 3.7 – Axiomes d'Armstrong pour les dépendances fonctionnelles

3.4.2.1 Axiomatisation

L'axiomatisation d'une théorie consiste en l'identification d'un ensemble fini et minimal d'expressions supposées vraies et de transformations permettant d'obtenir tous les théorèmes de cette théorie. À partir d'axiomes dont la vérité est supposée, on détermine des règles de déduction permettant de manipuler les axiomes ou toute expression obtenue à partir de ceux-ci. L'enchaînement de ces déductions est une *démonstration* qui conduit à un théorème, c'est donc une notion de preuve syntaxique comme celle présentée en annexe A.1.4. Les enjeux de la recherche d'un système axiomatique sont :

- l'*adéquation*, toutes les preuves construites à partir des axiomes doivent être sémantiquement justes : $\vdash$ implique $\models$,
- la *complétude*, le système d'axiome doit pouvoir dériver tous les théorèmes valides de la théorie : $\models$ implique $\vdash$.

Les dépendances fonctionnelles ont fait l'objet d'études approfondies, une axiomatisation adéquate et complète existante est connue sous le nom de *axiomes d'Armstrong*. Le tableau 3.7 donne les trois axiomes, en utilisant la notation propre aux dépendances fonctionnelles $X \rightarrow Y$.

L'axiomatisation d'Armstrong peut être prolongée pour prendre en compte conjointement les dépendances multivaluées et fonctionnelles. En revanche, il n'existe pas d'axiomatisation finie pour les dépendances de jointures, même considérées seules. Il en est de même pour les dépendances fonctionnelles et de jointures prises conjointement. Informellement, nous pourrions écrire que « plus la classe de dépendances est restrictive, plus elle dispose de bonnes propriétés ».

L'axiomatisation de Armstrong permet de décider de l'implication logique des dépendances fonctionnelles de façon efficace [Abiteboul95]. Si l'existence d'un système axiomatique conduit à un algorithme juste et complet de décision pour l'implication logique, la réciproque n'est pas vraie. Pour la classe des dépendances génératrices de tuples totales par exemple, il n'existe pas d'axiomatisation, cependant un algorithme de décision juste et complet existe. Nous devons faire appel à ces algorithmes pour résoudre des problèmes posés par les modèles et les politiques de contrôle d'accès.

3.4.2.2 Preuve de l'implication en marche avant

L'outil principal pour vérifier l'implication logique de dépendances est appelé le *chase*. Il a été introduit par Beeri et Vardi [Beeri84]. Le *chase* a été développé pour les dépendances génératrices de tuples et les dépendances génératrices d'égalités prises conjointement. Depuis, il a trouvé de nombreuses applications [Graham86, Maher96, Wang02, Christiansen06, Chomicki05, Bertossi06, Fagin06, Chomicki07].

Lorsque la classe est restreinte aux dépendances totales, le *chase* est une procédure de *décision*, c'est-à-dire un algorithme juste et complet qui permet de déterminer en un temps fini si un ensemble de dépendances Σ en implique logiquement une autre σ.

Cependant, lorsqu'on étend la classe de dépendances de Σ aux dépendances partielles, c'est-à-dire aux dépendances génératrices de tuples non totales, le *chase* n'est plus qu'une procédure de *preuve*. Cela signifie que si effectivement $\Sigma \models \sigma$, alors l'algorithme va s'arrêter en proposant une déduction $\Sigma \vdash \sigma$, mais si $\Sigma \not\models \sigma$ alors l'algorithme peut boucler sans fin et ne pas répondre. Par contre, même pour ces classes, si l'algorithme répond, sa réponse sera correcte : l'algorithme est juste.

Pour les applications pratiques, on peut techniquement limiter le nombre d'applications de dépendances pour le calcul de l'implication, en ajoutant par exemple une condition de sortie à la boucle principale de l'algorithme de la figure 3.2. Cette limite est à évaluer selon le nombre de dépendances de Σ pour minimiser le nombre de vrais négatifs[2], dans les cas où l'algorithme est arrêté pour avoir a atteint la limite d'applications alors qu'il aurait terminé sinon.

Du point de vue opérationnel, le *chase* est proche de la construction du point fixe d'un programme DATALOG. En effet, le principe du *chase* est de supposer l'existence de tuples telles que l'hypothèse h de $\sigma \equiv \phi \rightarrow \psi$ soit satisfaite, puis de traiter Σ comme un opérateur de fermeture générant des tuples et des égalités dans une instance symbolique[3] $\mathbf{I}$. Par application du théorème de la déduction ($\Phi \vdash \phi \rightarrow \psi$ est équivalent à $\Phi \cup \{\phi\} \vdash \psi$, annexe A.1.4), à chaque étape du calcul la condition suivante est testée :

- si $\mathbf{I}$ contient une copie de ψ, alors $\Sigma \models \sigma$,
- si $\mathbf{I}$ ne contient pas de copie de ψ et que l'on a atteint un point fixe, alors $\Sigma \not\models \sigma$.

Pour générer les tuples de $\mathbf{I}$, le principe est le suivant : au début, on initialise $\mathbf{I}$ par un ensemble de tuples aux valeurs arbitraires qui satisfont ϕ. Ensuite, pour chaque dépendance $\alpha \in \Sigma$, s'il existe une valuation μ des variables de l'hypothèse de α vers $\mathbf{I}$, alors on applique la conclusion de α valuée par μ. Deux cas se présentent :

- si la conclusion de α génère des *tuples*, alors on les *ajoute* à $\mathbf{I}$,
- si elle génère des *égalités*, alors on *identifie* des symboles de constantes de $\mathbf{I}$.

[2] Les réponses *fausses* qui devraient être *vraies*.

[3] Nous utilisons l'adjectif *symbolique* pour désigner une struture relationnelle chargée en mémoire utilisée par les procédures de preuve mais qui n'a pas vocation à persiter.

De plus, une liste des valuations appliquées à chaque dépendance est maintenue pour éviter qu'une même valuation ne soit appliquée plusieurs fois. L'algorithme de la procédure de preuve de [Beeri84] est donné par la figure 3.2. Il est composé d'une partie d'initialisation, d'une boucle principale et d'une condition d'arrêt. La procédure n'est pas présentée sous cette forme dans [Beeri84]. Dans l'article, la procédure est définie en termes de *tableaux*, un outil adapté à la représentation de dépendances, en particulier quand la base de données est représentable à l'aide d'une unique relation. La version que nous proposons ici a été adaptée et correspond à celle que nous avons implémentée dans notre prototype LIBDEPENDENCIES décrit en section 6.2. Elle est conceptuellement plus proche de [Maher96] que de [Beeri84].

3.4.2.3 Fonctionnement du *chase*

Illustrons le fonctionnement du *chase* avec les dépendances totales du tableau 3.8, où $\Sigma = \{\sigma_1, \sigma_2\}$ et où l'on souhaite prouver que $\Sigma \models \sigma'$. La base de données n'étant composée que d'une unique relation. Nous omettons les quantificateurs universels portant sur chaque variable présente en hypothèse. Les étapes du *chase* sont les suivantes :

1. supposons que l'hypothèse de $\sigma' \equiv \phi \rightarrow \psi$ soit satisfaite, c'est-à-dire ajoutons, à l'instance symbolique **I** les tuples suivants par la valuation $\nu(\phi)$:
   – $R(\mathtt{a}_0, \mathtt{b}_0, \mathtt{c}_1, \mathtt{d}_1)$
   – $R(\mathtt{a}_0, \mathtt{b}_1, \mathtt{c}_0, \mathtt{d}_2)$
   – $R(\mathtt{a}_1, \mathtt{b}_0, \mathtt{c}_0, \mathtt{d}_0)$
2. avec $\sigma_1 \equiv a \rightarrow b$, la valuation $\mu_1 = \{(A_0/\mathtt{a}_0), (B_0/\mathtt{b}_0)\ , (C_1/\mathtt{c}_1), (D_0/\mathtt{d}_1), (B_1/\mathtt{b}_1), (C_0/\mathtt{c}_0), (D_1/\mathtt{d}_2)\}$ est telle que l'ensemble des atomes de $\mu_1(a)$ appartiennent à **I**. Il faut ajouter à **I** la conclusion valuée $\mu_1(b)$:
   – $R(\mathtt{a}_0, \mathtt{b}_0, \mathtt{c}_0, \mathtt{d}_1)$
3. avec $\sigma_1 \equiv a \rightarrow b$, la valuation $\mu_2 = \{(A_0/\mathtt{a}_0), (B_0/\mathtt{b}_1), (C_1/\mathtt{c}_1), (D_0/\mathtt{d}_2), (B_1/\mathtt{b}_0), (C_0/\mathtt{c}_1), (D_1/\mathtt{d}_1)\}$ est telle que l'ensemble des atomes de $\mu_2(a)$ appartiennent à **I**. Il faut ajouter à **I** la conclusion valuée $\mu_2(b)$:
   – $R(\mathtt{a}_0, \mathtt{b}_1, \mathtt{c}_1, \mathtt{d}_2)$
4. avec $\sigma_2 \equiv a \rightarrow b$, la valuation $\xi = \{(A_0/\mathtt{a}_1), (A_1/\mathtt{a}_0), (B_0/\mathtt{b}_0), (C_0/\mathtt{c}_0), (D_0/\mathtt{d}_0), (D_1/\mathtt{d}_1)\}$, $\xi(a)$ appartiennent à **I**, comme σ_2 est une EGD, il faut remplacer $\mathtt{a}_1$ par $\mathtt{a}_0$ (ou $\mathtt{a}_0$ par $\mathtt{a}_1$) dans **I** :
   – le tuple $R(\mathtt{a}_1, \mathtt{b}_0, \mathtt{c}_0, \mathtt{d}_0)$
   – devient alors $R(\mathtt{a}_0, \mathtt{b}_0, \mathtt{c}_0, \mathtt{d}_0)$
5. comme $\nu(\psi)$, où ν est la valuation arbitraire choisie au début, est égal à $R(\mathtt{a}_0, \mathtt{b}_0, \mathtt{c}_0, \mathtt{d}_0)$, $\Sigma \vdash \sigma'$ et l'algorithme à prouvé que $\Sigma \models \sigma'$ par adéquation.

L'algorithme de la figure 3.2 est *bottom-up* : on dérive tous les théorèmes possibles puis on vérifie si celui que l'on souhaite prouver a été dérivé. Le résultat du *chase* ne dépend pas de l'ordre des dépendances appliquées pour construire **I** : quelque soit la

Entrée : Σ : un ensemble de TGD et d'EGD
Entrée : $\sigma \equiv \phi \rightarrow \psi$: une TGD ou une EGD
Sortie : *vrai* si $\Sigma \models \sigma$ ou $faux$ si $\Sigma \not\models \sigma$
begin
  {initialisation}
  soit **I** une instance symbolique vide de **R**
  soit ν une valuation arbitraire des variables de ϕ
  soit Γ l'ensemble de substitutions appliquées pour chaque dépendance
  for *atome* $R_i(c_1, \ldots, c_n)$ *de* $\nu(\phi)$ **do**
    $\mathbf{I} \leftarrow \mathbf{I} \cup R_i(c_1, \ldots, c_n)$
  {boucle principale}
  while $\mathbf{I}' \neq \mathbf{I}$ **do**
    $\mathbf{I}' \leftarrow \mathbf{I}$
    for $\alpha \in \Sigma$, α *de la forme* $a \rightarrow b$ **do**
      for *valuation* μ *telle que* $\mu(a)$ *appartiennent à* **I** **do**
        if $(\alpha, \mu) \notin \Gamma$ **then**
          if α *est une* TGD **then**
            for *atome* $R_i(c_1, \ldots, c_n)$ *de* $\mu(b)$ **do**
              if $R_i(c_1, \ldots, c_n) \notin \mathbf{I}$ **then**
                $\mathbf{I} \leftarrow \mathbf{I} \cup R_i(c_1, \ldots, c_n)$
          else
            for *égalité* $c_i = c_j$ *de* $\mu(b)$ **do**
              remplacer les occurrences de c_i dans **I** par c_j
          $\Gamma \leftarrow \Gamma \cup (\alpha, \mu)$
      {condition d'arrêt}
      if σ *est une* TGD **then**
        for *atome* $R_i(c_1, \ldots, c_n)$ *de* $\nu(\psi)$ **do**
          if $R_i(c_1, \ldots, c_n)$ *appartient à* **I** **then**
            retourner *vrai* et sortir
      else
        for *égalité* $c_i = c_j$ *de* $\nu(\psi)$ **do**
          if $c_i = c_j$ *ou si* (c_i *ou* c_j) *n'est pas présent dans* **I** **then**
            retourner *vrai* et sortir
end
retourner $faux$ et sortir (point fixe)

FIG. 3.2 – Procédure de preuve pour les dépendances

σ_1 MVD	$: R(A_0, B_0, C_1, D_0), R(A_0, B_1, C_0, D_1) \rightarrow R(A_0, B_0, C_0, D_0)$
σ_2 FD	$: R(A_0, B_0, C_0, D_0), R(A_1, B_0, C_0, D_1) \rightarrow A_0 = A_1$
σ' TTGD	$: R(A_0, B_0, C_1, D_1), R(A_0, B_1, C_0, D_2), R(A_1, B_0, C_0, D_0) \rightarrow R(A_0, B_0, C_0, D_0)$

TAB. 3.8 – Exemples de dépendances totales

stratégie de recherche sélectionnée le résultat est le même, ce processus est dit « Church-Rosser » [Abiteboul95].

3.4.2.4 Prise en compte de la quantification existentielle

Pour l'extension du *chase* aux dépendances génératrices de tuples non totales, la différence réside dans l'application d'une dépendance α qui comporte des variables quantifiées existentiellement $\tilde{Z}$. Dans ce cas, il faut étendre la valuation μ aux variables $\tilde{Z}$ en leur affectant des symboles de constantes arbitrairement choisis ne faisant pas partie de **I**. Intuitivement, on imagine que ce peut être une source de bouclage.

Exemple 3.4 Semi-décidabilité du *chase* pour les dépendances non totales

Soit la dépendance σ, $\forall A, B\ S(A, B) \rightarrow \exists A'\ S(B, A')$, et demandons au chase de vérifier si $\sigma \models \forall A, B\ S(A, B) \rightarrow R(A, B)$. Évidemment ce n'est pas le cas, mais regardons le comportement de l'algorithme.

Au début, on suppose qu'il existe un tuple $S(\mathtt{a}_0, \mathtt{b}_0)$. Par application de σ, avec $\mu_0 = \{(A/\mathtt{a}_0), (B/\mathtt{b}_0)\}$, on peut déduire qu'il existe un autre tuple $S(\mathtt{b}_0, \mathtt{a}_1)$ où $\mathtt{a}_1$ est un nouveau symbole de constante : μ_0 a été étendue à $\mu'_0 = \{(A/\mathtt{a}_0), (B/\mathtt{b}_0), (A'/\mathtt{a}_1)\}$. À l'étape suivante, on peut prendre $\mu_1 = \{(A/\mathtt{b}_0), (B/\mathtt{a}_1)\}$ et déduire l'existence d'un tuple $S(\mathtt{a}_1, \mathtt{b}_1)$ et ainsi de suite.

Comme les dépendances génératrices de tuples contraintes sont parmi les plus générales, nous allons présenter deux procédures de preuves qui leurs sont dédiées, développées dans [Maher96]. C'est principalement une propriété du domaine de contraintes utilisé, *Independence of Negative Constraints*, qui fait la différence entre ces deux procédures.

Définition (Independence of Negative Constraints [Lassez89])**.** *Pour un langage de contraintes $\mathcal{L}$ est une interprétation $\mathcal{D}$ de ce langage, le couple $(\mathcal{D}, \mathcal{L})$ dispose de la propriété* Independence of Negative Constraints *(*INC*) si, pour toutes contraintes $c, c_1, \ldots, c_n \in \mathcal{L}$:*

$$\begin{array}{rcl} \mathcal{D} & \models & \tilde{\exists}\ c \wedge \neg c_1 \wedge \ldots \wedge \neg c_n \\ & & \textit{si et seulement si} \\ \mathcal{D} & \models & \tilde{\exists} c\ \wedge \neg c_i \textit{ pour } i \in [1..n] \end{array}$$

D'un point de vue général, les deux algorithmes dédiés aux dépendances génératrices de tuples contraintes reprennent le *chase* [Maher96]. Le principe est d'ajouter à un magasin Env, les contraintes existantes entre les termes des tuples de $\mathbf{I}$. Les contraintes du magasin sont interprétées sur le domaine $\mathcal{D}$. L'égalité en particulier est interprétée dans ce domaine, et non traitée comme l'identification de variables.

Quand $(\mathcal{D}, \mathcal{L})$ dispose de la propriété INC, la vérification de l'implication est simplifiée, car il suffit de considérer chaque « morceau » séparément, sans se soucier de l'interaction des contraintes entre elles. Dans le cas où $(\mathcal{D}, \mathcal{L})$ ne dispose *pas* de la propriété INC, la procédure est plus complexe car $\mathbf{I}$ et Env doivent garder en mémoire quelles sont les contraintes supposées vraies utilisées pour produire des faits.

Quand on vérifie si $\Sigma \models (\phi, c \rightarrow \psi, c')$, où c et c' sont des conjonctions de contraintes, à chaque passe de la boucle principe du *chase*, la condition d'arrêt est testée. Un nouveau cas de terminaison doit être introduit, dans les cas où les contraintes générées (c'est-à-dire produites par les conclusions des dépendances) se contredisent. Trois cas de terminaison sont alors possibles :

1. si $\mathbf{I}$ contient une copie de ψ et que $\mathcal{D} \models Env \rightarrow c'$, alors $\Sigma \models \sigma$,
2. si le magasin est insatisfaisable , alors $\Sigma \vDash \sigma$ par vacuité,
3. si $\mathbf{I}$ ne contient pas de copie de ψ et que l'on a atteint un point fixe, alors $\Sigma \not\models \sigma$.

Exemple 3.5 Contraintes inconsistantes

Soient les dépendances :
– σ_1, $R(A,B) \rightarrow A \leq B$ et σ_2, $R(A,B) \rightarrow A \geq B$.
– σ'_1, $R(A,B) \rightarrow A < B$ et σ'_2, $R(A,B) \rightarrow A > B$.
– σ, $R(A,B) \rightarrow A = B$.

On s'intéresse aux preuves de $\{\sigma_1, \sigma_2\} \models \sigma$ et de $\{\sigma'_1, \sigma'_2\} \models \sigma$. Dans les deux cas, les procédures de preuves commencent par supposer l'existence d'un tuple $R(\mathtt{a}, \mathtt{b})$ qui satisfait l'hypothèse de σ. Pour la preuve de $\{\sigma_1, \sigma_2\} \models \sigma$:

1. par application de σ_1 on ajoute $(\mathtt{a} \leq \mathtt{b})$ à Env,
2. par application de σ_2 on ajoute $(\mathtt{a} \geq \mathtt{b})$ à Env,
3. $\mathcal{D} \models (\mathtt{a} \geq \mathtt{b}) \wedge (\mathtt{a} \leq \mathtt{b}) \rightarrow (\mathtt{a} = \mathtt{b})$.

La procédure termine par le cas de terminaison **1**, on peut déduire que $\{\sigma_1, \sigma_2\} \models \sigma$. Pour la preuve de $\{\sigma'_1, \sigma'_2\} \models \sigma$:

1. par application de σ'_1 on ajoute $(\mathtt{a} < \mathtt{b})$ à Env,
2. par application de σ'_2 on ajoute $(\mathtt{a} > \mathtt{b})$ à Env,
3. $(\mathtt{a} < \mathtt{b}) \wedge (\mathtt{a} > \mathtt{b})$ est *insatisfaisable*.

On peut déduire *par vacuité* que $\{\sigma'_1, \sigma'_2\} \models \sigma$, c'est le cas de terminaison **2**.

3.4.2.5 Preuve de l'implication en marche arrière

Les procédures de preuves que nous avons présentées pour les dépendances génératrices de tuples [Beeri84] et leur extensions aux domaines de contraintes [Maher96, Wang02] permettent d'inférer que $\Sigma \models \sigma$ en *marche avant*. À partir de l'hypothèse de σ, les procédures essaient de prouver sa conclusion.

Une procédure de preuve en marche arrière exploitant la notion de *pièce* à été mise au point pour les dépendances génératrices des tuples [Coulondre03], il s'agit donc d'une alternative au *chase*. La notion de pièce est issue du domaine des *graphes conceptuels* [Salvat96]. Informellement, nous dirons qu'une pièce est un ensemble d'atomes d'une formule liés par la présence de variables quantifiées existentiellement.

De par leur nature, les procédures de preuve de l'implication en marche arrière peuvent faciliter la compréhension et l'analyse des politiques de contrôle d'accès par des administrateurs non spécialistes de la logique.

3.4.3 Propriétés des modèles de contrôle d'accès

Notre hypothèse de travail peut-être exprimée de façon informelle par « les autorisations d'un système sont des données structurées selon un modèle de contrôle d'accès : utilisons les dépendances de données pour s'assurer que les autorisations sont valides ». Les dépendances permettent de modéliser des restrictions sur les instances possibles d'un modèle de contrôle d'accès qui ne sont pas exprimables en DATALOG.

Les garanties de décidabilité du fragment considéré de DATALOG ou DATALOG^C, nous permettent de nous assurer qu'à partir de la politique $\mathbf{I}$ en extension, on peut construire une politique $\mathbf{I}'$ la plus petite possible comportant les relations en intention. Cette instance est avant tout conceptuelle, il n'est pas vraiment nécessaire de la construire physiquement : $\mathbf{I}'$ peut être considérée un ensemble de vues virtuelles définies récursivement.

L'objectif est désormais de savoir si l'instance $\mathbf{I}'$ respecte bien les restrictions que le modèle de contrôle d'accès impose, représentées par un ensemble de dépendances Σ, c'est-à-dire vérifier que $\mathbf{I}' \models \Sigma$. Grâce aux algorithmes qui permettent d'effectuer cette vérification, nous résolvons une partie des problèmes de conception et de vérification des modèles et des politiques de contrôle d'accès.

3.4.4 Propriétés des modèles à rôles

Comme il a été indiqué précédemment que, comme l'illustre la représentation UML de la figure 2.2, pour garantir qu'une politique de contrôle d'accès RBAC$_0$ soit correcte, il faut ajouter l'expression des propriétés fondamentales des modèles RBAC à la structuration relationnelle du contrôle d'accès :

	Type	Expression logique
σ_0	TGD	$Endosse(S,R) \rightarrow \exists U\ Représente(S,U)$
σ_1	FD	$Représente(S,U_1), Représente(S,U_2) \rightarrow U_1 = U_2$
σ_2	TTGD	$Représente(S,U), Endosse(S,R) \rightarrow Habilite(U,R)$
σ_3	TGD	$Statique(U,A,O) \rightarrow \exists R\ Habilite(U,R), Affecte(R,A,O)$
σ_4	TGD	$Accès(S,A,O) \rightarrow \exists R\ Endosse(S,R), Affecte(R,A,O)$
σ_5	TGD	$Dynamique(U,A,O) \rightarrow \exists S,R\ Endosse(S,R), Représente(S,U), Affecte(R,A,O)$
σ	TGD	$Représente(S,U), Accès(S,A,O) \rightarrow \exists R\ Affecte(R,A,O), Habilite(U,R)$
σ'	TTGD	$Dynamique(U,A,O) \rightarrow Statique(U,A,O)$

TAB. 3.9 – Formalisation logique des propriétés du modèle RBAC_0

- l'existence et l'unicité de l'utilisateur associé à une session donnée (σ_0, σ_1),
- l'inclusion des rôles endossés dans une session dans ceux affectés à l'utilisateur (σ_2),
- l'absence d'affectation directe de permissions : le rôle doit être un intermédiaire incontournable pour acquérir les privilèges ($\sigma_3, \sigma_4, \sigma_5$).

Toute politique **I**, qui est une instance du modèle de contrôle d'accès RBAC_0 doit satisfaire les dépendances σ_0, σ_1 et σ_2 du tableau 3.9, c'est-à-dire que ces propriétés doivent être vérifiées par *toute* politique. Les dépendances de ce tableau portent sur le schéma *edb* de $\mathbf{RBAC_0}$. La politique jouet du tableau 3.1 satisfait σ_0, σ_1 et σ_2.

De plus, si on ajoute à la base de données les règles de déductions des triplets d'autorisation du tableau 3.2, alors l'instance $\mathbf{I}'$ doit également respecter σ_3, σ_4 et σ_5 qui expriment qu'on ne peut pas définir des triplets autorisations sans passer par des affectations de rôles aux utilisateurs et de permissions aux rôles.

3.4.5 Utilisation des procédures

Quand on analyse les dépendances présentées par le tableau 3.9, on peut se demander si d'autres propriétés sont satisfaites par les politiques RBAC_0. Soit $\mathbf{I}'$ une instance qui satisfait les dépendances σ_0 à σ_5 du tableau 3.9. Alors $\mathbf{I}'$ doit également vérifier les propriétés suivantes qui sont logiquement impliquées par $\{\sigma_0 \dots \sigma_5\}$:

- si un utilisateur dispose d'une permission par l'intermédiaire d'une session, alors l'utilisateur est affecté à au moins un rôle qu'il endosse dans la session en question, et le rôle auquel il est affecté dispose de la permission. La dépendance σ du tableau 3.9 formalise cette propriété exprimée en langue naturelle,
- les triplets d'autorisations dynamiques sont inclus dans les triplets d'autorisations statiques. Cette propriété est représentée par la dépendance σ' du tableau 3.9.

Ce type d'inférence peut être réalisé à l'aide des procédures de preuve. Ces procédures permettent de prouver l'implication logique d'une dépendance quelle que soit l'instance considérée. Quand on modélise les modèles de contrôle d'accès avec les dépendances, on peut donc raisonner sur les modèles quelle que soient les politiques qui en seront instances. On peut ainsi identifier des propriétés déduites des modèles de contrôle d'accès pour la simplification ou la vérification de modèles, sections 5.2.4 et 5.2.5.

Les procédure de preuves pour les dépendances sont particulièrement appréciables pour ces applications, car quand on prouve que $\Sigma \models \phi \rightarrow \psi$ on obtient une trace de la forme $\phi, \sigma_1 \dots \sigma_n, \psi$ ou chaque transition d'une formule à l'autre résulte de l'application d'une et une seule dépendance de Σ. On dispose ainsi d'une trace de preuve relativement lisible, qui peut être traduite en langue naturelle en « lisant » les applications succésives des dépendances. Ceci permet de faciliter d'analyse d'un preuve pas un utilisateur non spécialiste de la logique.

Exemple 3.6 Inclusion des autorisations dynamiques

Considérons les dépendances des tableaux 3.2 (τ_i) et 3.9 (σ_i). Nous souhaitons vérifier que dans *toute* politique RBAC_0, $Dynamique \subseteq Statique$, c'est-à-dire la dépendance σ' du tableau 3.9. Une trace d'exécution du *chase* est la suivante :

1. supposons l'existence d'un tuple $Dynamique(\mathtt{u}_0, \mathtt{a}_0, \mathtt{o}_0)$,
2. par application de σ_5, il existe également un rôle $\mathtt{r}_1$ et une session $\mathtt{s}_1$ tels que $Endosse(\mathtt{s}_1, \mathtt{r}_1)$, $Représente(\mathtt{s}_1, \mathtt{u}_0)$ et $Affecte(\mathtt{r}_1, \mathtt{a}_0, \mathtt{o}_0)$,
3. comme il existe $Endosse(\mathtt{s}_1, \mathtt{r}_1)$ et $Représente(\mathtt{s}_1, \mathtt{u}_0)$, alors par application de σ_2 il existe $Habilite(\mathtt{u}_0, \mathtt{r}_1)$,
4. comme il existe $Affecte(\mathtt{r}_1, \mathtt{a}_0, \mathtt{o}_0)$ et $Habilite(\mathtt{u}_0, \mathtt{r}_1)$, alors par application de τ_1, il existe $Statique(\mathtt{u}_0, \mathtt{a}_0, \mathtt{o}_0)$,
5. donc si un tuple $Dynamique(\mathtt{u}_0, \mathtt{a}_0, \mathtt{o}_0)$, quel qu'il soit, existe dans la politique alors le tuple $Statique(\mathtt{u}_0, \mathtt{a}_0, \mathtt{o}_0)$ existe aussi, on a prouvé que $\{\sigma_5, \sigma_2, \tau_1\} \models \sigma$.

Ainsi, toute politique RBAC_0 qui respecte les dépendances des tableaux 3.2 et 3.9 respecte la propriété exprimée par la dépendance σ' du tableau 3.9. Cet exemple de preuve effectuée par le *chase* illustre une des applications que nous proposons dans le chapitre 5 : la simplification de modèle de contrôle d'accès. Une trace détaillée et commentée de cette preuve est proposée en annexe C.

	Type	Expression logique
γ_0	TTGD	$Séparation(R_1, R_2) \rightarrow Exclusion(R_1, R_2)$
γ_s	TTGD	$Exclusion(R_1, R_2) \rightarrow Exclusion(R_2, R_1)$
γ_r	NGD	$Exclusion(R, R) \rightarrow \bot$
γ_1	NGD	$Exclusion(R_1, R_2), Habilite(U, R_1), Habilite(U, R_2) \rightarrow \bot$
γ_2	NGD	$Exclusion(R_1, R_2), Endosse(S, R_1), Endosse(S, R_2) \rightarrow \bot$
γ_3	NGD	$Exclusion(R_1, R_2), Affecte(R_1, A, O), Affecte(R_2, A, O) \rightarrow \bot$
γ_4	NGD	$Exclusion(R_1, R_2), Affecte(R_1, A_1, O), Affecte(R_2, A_2, O) \rightarrow \bot$
γ_5	NGD	$Exclusion(R_1, R_2), Endosse(S, R_1), Endosse(S, R_2),$ $Représente(S, U), Représente(S, U) \rightarrow \bot$

TAB. 3.10 – Formalisation logique des propriétés de l'exclusion mutuelle

3.4.6 Expression de l'exclusion mutuelle

En section 2.5.1, nous avons présenté une notion importante des modèles RBAC : les relations d'exclusion mutuelle. Le travail de modélisation de la notion de contrainte dans un modèle de contrôle d'accès nécessite l'analyse des *propriétés algébriques* que la relation d'exclusion respecte et des restrictions qu'elle impose sur les politiques pour qu'elles soient *intègres*.

Comme lors de l'ajout de la hiérarchie de rôles au modèle RBAC_0, nous allons raffiner le modèle RBAC_0 pour prendre en compte l'exclusion mutuelle entre rôles. La formalisation que nous proposons se base sur deux relations et sur les dépendances du tableau 3.10 :

- $Séparation \subseteq \mathcal{R} \times \mathcal{R}$, la relation d'exclusion en *extension* (γ_0),
- $Exclusion \subseteq \mathcal{R} \times \mathcal{R}$, la fermeture symétrique de *Séparation* (γ_s).
- l'irréflexivité de la relation *Exclusion* (γ_r).

Nous avons remarqué que les relations d'exclusion mutuelle ont été interprétées de plusieurs façons dans les modèles RBAC [Gligor98, Crampton03a]. Les dépendances γ_1, γ_2, γ_3, γ_4 et γ_5 du tableau 3.10 formalisent les différentes significations de l'exclusion mutuelle entre rôles. On remarque que les expressions des dépendances γ_1 à γ_5 sont très proches : pour chacune d'entre elles, une des relations du modèle RBAC est présente deux fois dans l'hypothèse. Ainsi, on ajoutera l'une de ces cinq dépendances γ_1 à γ_5 à l'ensemble Σ pour formaliser que la relation $Exclusion(R_1, R_2)$ exprime l'impossibilité d'avoir :

- un *utilisateur* commun à deux rôles : γ_1,
- une *session* commune à deux rôles : γ_2,
- une *permission* commune à deux rôles : γ_3,
- un *objet* commun sur lequel deux rôles auraient droit d'action : γ_4,
- un *utilisateur* commun à deux rôles par l'intermédiaire de *sessions* : γ_5.

Les cinq significations de l'exclusion mutuelle ne sont pas indépendantes. En effet, interdire à deux rôles d'avoir un utilisateur en commun interdit également d'avoir une session en commun, car chaque session ne correspond qu'à un et un seul utilisateur. Les procédures de preuves permettent d'évaluer les interprétations de l'exclusion :

- $\{\gamma_1, \sigma_0, \sigma_2\} \models \gamma_2$,
- $\{\gamma_4\} \models \gamma_3$,
- $\{\gamma_2\} \models \gamma_5$,
- $\{\gamma_5, \sigma_0\} \models \gamma_2$,
- si $\sigma_0 \in \Sigma$, alors γ_5 et γ_2 sont équivalentes.

Grâce à l'identification automatisée des relations entre les différentes significations de l'exclusion mutuelle, un concepteur peut décider de mettre en œuvre deux formes d'exclusion mutuelle avec la garantie qu'elles sont indépendantes. On peut ainsi concevoir un modèle de contrôle d'accès qui comporte deux relations d'exclusion : une première qui interdit d'avoir des utilisateurs communs à des rôles en exclusion (dépendance γ_3 du tableau 3.10), et une seconde qui interdit d'avoir des permissions communes (dépendance γ_1 du tableau 3.10).

Ces vérifications de propriétés abstraites d'un modèle de contrôle d'accès n'ont, à notre connaissance, pas été proposées. Elles sont cependant d'un grand intérêt pratique lors de la conception d'un modèle de contrôle d'accès. De plus, grâce aux outils de preuves dédiées aux dépendances de données que nous avons présentés, ces vérifications sont automatisées : nous montrerons que les outils de preuves permettent de prouver des propriétés non triviales qui ont été démontrées manuellement.

3.5 Discussion et synthèse

3.5.1 De la négation

Les dépendances et le cadre DATALOG que nous avons présenté en section 3.3.3 ne permettent pas l'expression de négation dans les formules. Des extensions de DATALOG ont été proposées. DATALOG$^\neg$ étend les règles DATALOG en autorisant la présence de littéraux négatifs dans le corps des règles. La sémantique des programmes DATALOG$^\neg$ est plus complexe que celle des programmes DATALOG [Abiteboul95] :

- un programme DATALOG$^\neg$ peut ne pas avoir de point fixe du tout,
- plusieurs points fixes minimaux peuvent exister,
- l'application de règles peut ne pas converger vers le point fixe, même s'il existe,
- l'application de règles peut converger vers un point fixe, mais ce dernier n'est pas nécessairement minimal.

Plusieurs solutions ont été apportées à ce problème, en restreignant l'utilisation de la négation avec la notion de strate, ou en dotant les programmes DATALOG$^\neg$ d'une

sémantique plus complète. En revanche, il n'y a pas à notre de connaissance de classes de dépendances autorisant l'expression de littéraux négatifs.

L'absence de négation n'est cependant pas un verrou au développement de modèles de contrôle d'accès. Les dépendances génératrices de nullité ou de contraintes permettent de définir des conditions pour lesquelles une instance n'est *pas* valide, permettant ainsi de modéliser des restrictions dans un modèle de contrôle d'accès sans utiliser explicitement de négation, car $\phi \rightarrow \bot$ équivaut à $\neg(\phi)$.

En sélectionnant le cadre des dépendances de données, nous avons fait le choix de prendre en compte la possibilité d'exprimer des contraintes d'intégrité complexes faisant intervenir des quantificateurs existentiels plutôt que d'utiliser la négation. Aucune proposition de contrôle d'accès n'a encore pris ce parti.

3.5.2 Structuration du cadre logique

Depuis le modèle relationnel, nous avons successivement introduit le modèle DATALOG puis les dépendances de données, pour aboutir à la définition d'un paradigme de bases de données relationnelles étendues. Notre approche « orientée données » nous conduit à définir le contrôle d'accès comme une struture relationnelle $\mathbf{AC} = (edb \cup idb, P, \Sigma)$ composée :

- d'un ensemble de concepts et de relations en extension qui modélisent la *structure* du contrôle d'accès : *edb*
- une *politique* en extension instance du modèle de contrôle d'accès : $\mathbf{I}$,
- un ensemble de relations en intention et de règles de déduction qui modélise les *principes* en vigueur dans le modèle de contrôle d'accès : *idb* et un ensemble de règles P qui les définit,
- une politique en *intention* qui est déterminée à partir des règles de déduction : une instance la plus petite possible $\mathbf{I}'$ telle que $\mathbf{I} \subseteq \mathbf{I}'$ et $\mathbf{I}' \models P$,
- un ensemble de dépendances de données qui limitent les politiques possibles à celles respectant les propriétés des modèles Σ, avec $\mathbf{I}' \models \Sigma$.

Le cadre proposé nous permet d'exprimer des propriétés complexes des modèles de contrôle d'accès, manipulables directement sans réécriture par les algorithmes de preuves. Cette possibilité n'est pas prise en compte dans les propositions étendant sur DATALOG ou la programmation logique [Li03b, Barker03, Barker02]. Nous avons illustré l'utilisation des dépendances aux modèles de contrôle d'accès RBAC en montrant qu'elles :

- permettent la formalisation logique des propriétés des modèles RBAC,
- permettent de donner une sémantique logique à ces modèles,
- sont suffisamment expressives pour modéliser des propriétés qui n'ont pas été prises en compte dans les autres approches,

– permettent de développer la notion d'intégrité d'une politique de contrôle d'accès.

Nous pensons que l'*intégrité* des politiques revêt une place centrale dans la spécification des modèles de contrôle d'accès. En effet, comme la majeure partie des failles sont dues à des erreurs ou à de mauvaises configurations, s'assurer de la correction des politiques instances des modèles de contrôle d'accès est incontournable. Les procédures de preuve, dont nous avons présenté le *chase*, permettent de raisonner sur les structurations relationnelles du contrôle d'accès : elles permettent de vérifier les modèles et les politiques.

3.5.3 Vers une spécification générique du contrôle d'accès

Il n'est pas possible de limiter notre proposition à la seule famille des modèles à rôles telle que définie dans le standard. En effet dans les systèmes d'information :

– d'une part il est rare qu'une implémentation des modèles RBAC respecte à la lettre les spécifications du standard : modéliser ces implémentations nécessite un cadre capable d'exprimer les nuances qui existent entre elles,
– d'autre part, la tendance actuelle est de développer des modèles de contrôle d'accès *ad hoc* qui permettent de répondre au mieux aux besoins de sécurité des organisations. De tels modèles sont souvent composites, intégrant des concepts et des principes pris dans différents modèles de contrôle d'accès,
– enfin, il est courant d'avoir plusieurs modèles de contrôle d'accès cohabitants au sein d'une même organisation. Le maintien des systèmes historiques (*legacy systems*) ainsi que la multiplication des applications conduisent généralement les administrateurs à intervenir sur des politiques structurées selon des modèles différents. Une formalisation homogène facilite l'unification des mécanismes contrôle d'accès à l'échelle globale du système d'information.

Les dépendances de données permettent d'établir les propriétés fondamentales qu'un modèle de contrôle doit respecter et permettent de définir formellement la notion d'intégrité des politiques dotée d'une sémantique logique. L'expressivité du cadre retenu permet également l'expression de propriétés novatrices, qui n'ont pas été proposées dans la littérature.

Le chapitre suivant généralise la formalisation des modèles à rôles de ce chapitre à des familles plus générales de modèles de contrôle d'accès. Il présentera des définitions génériques du contrôle d'accès permettant de concevoir et formaliser logiquement des systèmes de gestion des autorisations complexes.

When in doubt, use brute force.

Butler Lampson, 1984 - "Hints for Computer System Design"

4

Structuration du contrôle d'accès

▷ *Le chapitre précédent a présenté le cadre des dépendances de données et l'a appliqué pour représenter, raisonner et vérifier les modèles de contrôle d'accès à rôles. Ce chapitre reprend cette approche pour des familles de modèles plus générales.*

Nous présentons une modélisation générique des droits fondée sur les concepts communs aux différents modèles de contrôle d'accès que nous avons identifiés. Cette modélisation générique s'appuie sur le modèle relationnel. L'objectif est de représenter l'organisation des droits dans les systèmes à partir de définitions réduites pour ensuite ajouter successivement des couches qui représentent les principes et les propriétés de ces modèles.

C'est un pas vers une métamodélisation du contrôle d'accès dont les instances seraient des modèles présentés dans l'état de l'art. Comme les dépendances de données n'autorisent pas certaines expressions logiques, le modèle générique que nous proposons ne permet pas de prendre en compte certaines des spécificités introduites dans la littérature. Nos principales contributions de ce chapitre concernent :

- *les définitions génériques des composants présentés dans les modèles de contrôle d'accès,*
- *la classification abstraite de ces composants, en particulier les catégorisations :*
  - *en aspects factuels (effectivement stockés) et déduits,*
  - *en aspects statiques (entièrement déterminés par les administrateurs) et dynamiques,*
- *la prise en compte des critiques formulées à l'encontre du standard* RBAC,
- *la généralisation du concept d'intégrité des politiques de contrôle d'accès.*

L'ensemble des définitions que nous proposons dans ce chapitre forment le cœur de notre travail. À partir de cette base de formalisation nous développons des outils pour la vérification des modèles et des politiques de contrôle d'accès dans le chapitre 5. Ces outils utilisent les procédures de preuve pour les dépendances dont la réalisation est décrite dans le chapitre 6.

L'annexe B synthétise la notation et les symboles de prédicats de ce chapitre. Une formalisation générale des modèles RBAC *organisée selon ce chapitre y est proposée.* ◁

Plan du chapitre

Le chapitre précédent a proposé un cadre logique fondé sur le modèle relationnel pour représenter la structuration, les principes et les propriétés des modèles de contrôle d'accès. Nous avons illustré l'application de ce cadre avec la famille des modèles RBAC. Désormais, nous entreprenons de généraliser la modélisation des droits. Ce chapitre propose à notre initiative des définitions novatrices basées sur le modèle relationnel. Elle reprennent et étendent celles présentées dans l'état de l'art.

4.1 Structuration générale du contrôle d'accès

Cette section propose des définitions génériques des ensembles et des relations entre ces ensembles qui sont mis en œuvre dans les modèles de contrôle d'accès. Il s'agit d'une première étape d'identification et de formalisation des principales notions communes aux modèles de contrôle d'accès. Dans les sections suivantes, nous réutiliserons ces définitions pour les faire correspondre au modèle relationnel.

4.1.1 Modèle de contrôle d'accès

Définition (Modèle de contrôle d'accès)**.** *Un* modèle *de contrôle d'accès $\mathcal{AC}$ est exprimable en terme de* concepts*, c'est-à-dire des entités définies et utilisées pour structurer les droits, et de* relations *entre ces concepts, c'est-à-dire des associations n-aires entres les entités. Concepts et relations sont recouvrés par le terme générique d'*aspect*. Les concepts de* sujet *($\mathcal{S}$), d'*utilisateur *($\mathcal{U}$), d'*action *($\mathcal{A}$) et d'*objet *($\mathcal{O}$) sont présents dans tout modèle de contrôle d'accès.*

*À chaque sujet donné n'est associé qu'*un et un seul *utilisateur : il existe une injection de $\mathcal{S}$ dans $\mathcal{U}$. À un modèle $\mathcal{AC}$ est associé un ensemble de principes qui régissent son fonctionnement et de propriétés qui définissent son intégrité.*

Cette définition généralise les principes des modèles structurés. Elle réduit à des composants élémentaires l'organisation des droits. Le tableau 2.5 a présenté les ensembles et relations du modèle RBAC_1. Par rapport à la définition minimaliste proposée au dessus, le modèle RBAC_1 introduit la notion de rôle. D'une même façon, la figure 2.5 a présenté les principaux concepts et relations du modèle ORBAC.

Sujets et utilisateurs peuvent être éventuellement confondus, comme dans les modèles basés sur l'identité. De plus, dans de nombreux modèles, on s'intéresse moins aux ensembles $\mathcal{A}$ et $\mathcal{O}$ qu'aux permissions qui sont des paires d'actions sur des objets.

Définition (Triplet fondamental du contrôle d'accès)**.** *Tout modèle de contrôle d'accès doit permettre de dériver le* triplet fondamental *d'autorisation, $\mathcal{ACCESS}$, qui représente l'ensemble des actions autorisées aux sujets sur les objets du système :*

$$\mathcal{ACCESS} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{O}$$

La dérivation de la relation $\mathcal{ACCESS}$ à partir des concepts et des relations d'un modèle de contrôle d'accès fait partie de ses principes.

Nous exprimons ainsi que l'objectif central des modèles de contrôle d'accès est de permettre de dériver une relation $\mathcal{ACCESS}$ représentable par une matrice de Lampson comme celle du tableau 2.2. Un modèle de contrôle d'accès permet notamment de définir des intermédiaires entre sujets ou utilisateurs et permissions. Selon les domaines d'application d'un modèle de contrôle d'accès, les permissions sont :

- des opérations SQL sur des tables : dans le contrôle d'accès aux systèmes de gestion de bases de données, la politique va définir quelles opérations[1] sont autorisées sur les tables,
- des primitives sur des fichiers : dans les systèmes de gestion de fichier, la politique va définir les primitives autorisées[2] sur les fichiers. Notons que dans ce cas, les objets sont généralement organisés sous forme d'arbres,
- des méthodes sur des services : dans les architectures de services web, la politique va définir quelles méthodes sont autorisées d'invocation par les autres services.

4.1.2 Moniteur

À partir de la définition d'un modèle de contrôle d'accès, nous définissons le *moniteur de référence*. Son mécanisme consiste à évaluer une requête sur la relation $\mathcal{ACCESS}$. Nous n'avons pas encore défini *comment* dériver le triplet $\mathcal{ACCESS}$, en effet ceci est dépendant du modèle considéré. Par exemple, dans les modèles MAC, les actions autorisées sont déduites des labels associés aux utilisateurs et aux objets, alors que dans RBAC, les actions autorisées sont explicitement attribuées aux rôles.

Définition (Moniteur de contrôle d'accès)**.** *Un* moniteur de contrôle d'accès *est modélisable par une fonction $\mathcal{F}$ qui à une action demandée par un sujet sur un objet – c'est-à-dire une requête d'accès – fait correspondre une décision binaire d'autorisation indiquant si l'accès demandé est autorisé ou non.*

$$\mathcal{F} : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow \{vrai, faux\}$$
$$\mathcal{F}(s,a,o) \quad = \quad \begin{cases} vrai & si\ (s,a,o) \in \mathcal{ACCESS} \\ faux & si\ (s,a,o) \notin \mathcal{ACCESS} \end{cases}$$

Nous souhaitons la généralisation des différents principes introduits au fil de l'évolution de la structuration des droits. L'objectif de fournir un cadre dans lequel développer de nouveaux modèles de contrôle d'accès adaptés aux besoins des organisations. Nous réalisation cet objectif en réduisant les politiques de contrôle d'accès à une expression simple : la matrice introduite par les auteurs de [Lampson74].

[1] Par exemple, les opérations `insert`, `delete`, `select` et `update`.

[2] Par exemple, les droits `read`, `write` et `execute`.

4.1.3 Structuration relationnelle

Nous avons choisi de construire la modélisation du contrôle d'accès sur le cadre relationnel. Dans les sections suivantes de ce chapitre nous définirons successivement des couches s'appuyant sur ce fondement en faisant correspondre aux composants du contrôle d'accès des notions issues du modèle relationnel. Ces différentes notions forment un paradgime assimilable aux bases de données déductives. Ainsi, un modèle de contrôle d'accès est représenté par une structure relationnelle $\mathbf{AC} = (sch, P, \Sigma)$ avec :

- $sch = edb \cup idb$: le schéma du modèle de contrôle d'accès,
- edb : le schéma en extension de **AC**, l'ensemble des relations en *extension* du modèle. Pour notre application, cet ensemble représente le noyau du modèle de contrôle d'accès : les concepts et relations introduits pour organiser les droits,
- idb : le schéma en intention de **AC**, l'ensemble de relations en *intention* du modèle. Il s'agit de concepts et de relations déductibles à partir de ceux connus en extension,
- P : l'ensemble de règles de déduction qui définit les relations en intention. Cet ensemble permet de modéliser les principes qui régissent la dérivation des relations en intention à partir de celles en extension,
- $\Sigma = \Sigma_{edb} \cup \Sigma_{idb}$: l'ensemble des dépendances de données composé de Σ_{edb} – l'ensemble des dépendances sur edb – et de Σ_{idb} l'ensemble des dépendances sur idb. Σ permet de modéliser des propriétés que les politiques de contrôle d'accès doivent respecter.

Dans l'expression des formules de logique des règles requêtes conjonctives, des règles DATALOG et des dépendances, nous omettrons de faire figurer les quantificateurs universels sous entendus. Les quantificateurs existentiels seront quant à eux notés explicitement. Pour une politique de contrôle d'accès, instance d'un modèle **AC** nous noterons :

- $\mathbf{I}$: l'instance de edb qui satisfait Σ_{edb}. Il s'agit de la politique de contrôle d'accès définie explicitement par les administrateurs,
- $\mathbf{I}'$: le plus petit point fixe du programme P, c'est-à-dire l'instance de $edb \cup idb$ qui satisfait P et $\Sigma = \Sigma_{edb} \cup \Sigma_{idb}$. Il s'agit de la politique définie implicitement, qui comprend les faits enregistrés par les administrateurs, mais aussi ceux déduits à partir des principes du modèle.

Une règle de déduction DATALOG ou DATALOG^C satisfait les restrictions syntaxiques imposées aux dépendances génératrices de tuples contraintes. Ainsi, nous pouvons assimiler les règles à des dépendances particulières, qui sont satisfaites par $\mathbf{I}'$. Ainsi, l'instance $\mathbf{I}'$ *satisfait* les règles P vues comme des dépendances. La réciproque de cette proposition n'est en revanche pas valide : une dépendance de données peut ne pas satisfaire les restrictions syntaxiques imposées aux règles DATALOG^C. La dépendance $R(A, B) \rightarrow \exists A'\ S(A', A), A' \geq B + A$ n'est pas exprimable en DATALOG^C.

Dans un premier temps, nous définissons ce qu'est un modèle de contrôle d'accès en termes de *concepts* et de *relations* formant son *noyau*. Dans un second temps, nous caractériserons et développerons les différentes notions introduites dans les modèles de contrôle d'accès.

4.2 Concepts et relations des modèles de contrôle d'accès

Cette section définit les schémas des modèles de contrôle d'accès. Nous réutilisons les définitions de la section précédente et explicitons les notions de concepts et de relations dans le modèle relationnel. Cette section définit les composantes *edb*, Σ_{edb} et **I** de la structuration relationnelle proposée.

4.2.1 Représentation des modèles de contrôle d'accès

La proposition suivante forme la brique de base sur laquelle nous développons la suite de la thèse. Dans la généralisation proposée, nous supposons que le noyau d'un modèle de contrôle d'accès est représentable par un *schéma* relationnel. Grâce à cette proposition, nous définissons une politique structurée selon un modèle de contrôle d'accès comme une instance du schéma associé.

Proposition. *Représentation des modèles de contrôle d'accès*

Un modèle de contrôle d'accès structuré $\mathcal{AC}$, composé de concepts et de relations entre ces concepts, est représenté par une structure relationnelle **AC**.

Le noyau *de $\mathcal{AC}$ est représenté par un* schéma relationnel en extension *de contrôle d'accès edb.*

Définition (Politique de contrôle d'accès)**.** *Une* politique de contrôle d'accès **I** *est une instance d'un schéma en extension de contrôle d'accès.*

Ces deux proposition sont fondamentales pour la thèse. Nous supposons la représentablité des noyaux par un schéma relationnel maintenant, pour pouvoir ensuite utiliser les règles de déduction et les dépendances de données qui seront construites audessus du schéma. Elles permettent de définir les principes et les propriétés des modèles.

Afin de différencier les concepts des modèles de contrôle d'accès présentés dans le chapitre 2 et leur représentation dans la structuration relationnelle, nous notons :

– en police calligraphiée *les concepts et relations* d'un modèle de contrôle d'accès $\mathcal{AC}$, par exemple : $\mathcal{S}$ pour le concept de sujet, $\mathcal{A}$ pour celui d'action et $\mathcal{URA}$ pour la relation d'attribution de rôles aux utilisateurs définis dans les modèles RBAC,

– en police droite, la modélisation, grâce à des *formes prédicatives*[3] de la logique du premier ordre, des concepts et relations de de la *structure relationnelle* **AC**, par exemple : $Sujet$, $Action$ d'arité 1 et $Habilite$ d'arité 2.

Définition (Représentation des concepts et des relations d'un modèle)**.** *Un* concept $\mathcal{C}$ *dans un modèle de contrôle d'accès* $\mathcal{AC}$ *est représenté par une relation unaire* $C(ID)$*, où* ID *est appellé* identifiant *du concept. La relation en* extension $C(ID)$ *appartient à edb.*

Une relation *n-aire* $\mathcal{R}$ *entre les concepts* $\mathcal{C}_1, \ldots, \mathcal{C}_n$ *est exprimée par une relation en* extension $R \in edb : R(ID_1, \ldots, ID_n)$ *où les* ID_i *sont respectivement les identifiants des concepts* $C_1, \ldots, C_n$*.*

D'après la définition d'un modèle de contrôle d'accès que nous avons proposée, nous savons que dans toute expression relationnelle d'un modèle, il existera en extension au moins quatre relations unaires $User(U)$, $Sujet(S)$, $Action(A)$ et $Objet(O)$ modélisant respectivement dans la base de données les concepts d'utilisateur ($\mathcal{U}$), de sujet ($\mathcal{S}$), d'action ($\mathcal{A}$) et d'objet ($\mathcal{O}$).

Exemple 4.1 Schéma du modèle RBAC$_0$

Le noyau du modèle RBAC$_0$ peut être défini à l'aide d'un schéma en extension *edb* qui traduit dans le modèle relationnel les définitions ensemblistes proposées dans le standard RBAC [Ferraiolo03b] (tableau 2.5). Dans les modèles RBAC le concept de sujet est appelé session.

Le schéma relationnel *edb* est donné par la figure 3.1, il comporte :
– d'une part, les concepts des modèles RBAC, les relations unaires $User(U)$, $Sujet(S)$, $Role(R)$, $Action(A)$, et $Objet(O)$.
– d'autre part, les relations entre les concepts $Habilite(U, R)$, $Représente(S, U)$, $Endosse(S, R)$ et $Affecte(R, A, O)$.

Une politique jouet **I** instance de ce schéma est donnée par le tableau 3.1.

Les modèles de contrôle d'accès peuvent être définis à partir d'ensembles. Les relations unaires permettent de traduire l'appartenance des concepts à ces ensembles, et les relation n-aires exprimant des relations entre ces ensembles. Le tableau 4.1 synthétise l'application Ψ de représentation. Il résume nos choix fondamentaux de modélisation relationnelle du contrôle d'accès.

4.2.2 Extension de la notion de concept et de relation

Nous pouvons autoriser la définition d'attributs afin de permettre aux administrateurs de la sécurité d'enregistrer des informations qui ne sont pas prises en compte pour les décisions de contrôle d'accès, mais qui peuvent être utiles à l'administration.

[3] C'est-à-dire les relations entre des ensembles du modèle relationnel.

Définition ensembliste	$\xrightarrow{\Psi}$	Structuration relationnelle
$\mathcal{AC}$	$\rightarrow$	**AC**
concept $\mathcal{C}$	$\rightarrow$	forme prédicative unaire $C(ID_C)$
relation n-aire $\mathcal{R}$	$\rightarrow$	forme prédicative n-aire $R(ID_1, \ldots, ID_n)$

TAB. 4.1 – Application de représentation Ψ

Par exemple, au concept d'utilisateur, on aurait certainement besoin d'associer un nom, un prénom, une adresse, un numéro de bureau, etc.

Pour permettre l'expression d'attributs qui ne participent pas à la prise de décision d'accès, nous étendons les notions de concepts et relations d'un modèle en définissant des dépendances fonctionnelles entre les identifiants de concepts et les attributs :

Définition (Concepts et relations *étendus* d'un modèle)**.** *Un concept $\mathcal{C}$ auquel est associé n attributs $A_1, \ldots, A_n$ est exprimé par une relation $(n+1)$-aire en* extension *de edb : $C(ID, A_1, \ldots, A_n)$ où ID est l'identifiant du concept.*

On associe à tout concept $\mathcal{C}$, une dépendance fonctionnelle de Σ_{edb} exprimant que ID est clef de la relation C : $C(ID, A_1, \ldots, A_n), C(ID, A'_1, \ldots, A'_n) \rightarrow A_1 = A'_1, \ldots, A_n = A'_n$.

La prise en compte d'attributs supplémentaires n'est pas complexe, mais est fastidieuse. En effet, nous imposons que ces attributs soient entièrement déductibles à l'aide de dépendances fonctionnelles. Afin de ne pas surcharger les expressions logiques de la thèse, nous ne prendrons pas en compte les attributs dans la suite du développement.

4.2.3 Intégrité des schémas relationnels

Nous avons proposé une représentation des notions de concept et de relation des modèles de contrôle d'accès. Maintenant, nous allons nous attacher aux liens entre ces concepts. Nous allons imposer que les relations construites dans le modèle de contrôle d'accès soient bien toutes reliées à des concepts du modèle. Cette propriété qui doit être garantie dans la politique est représentée par une dépendance génératrice de tuples.

Les dépendances de données ont été définies comme des contraintes d'intégrité sur les schémas de base de données. Nous mettons à profit cette caractéristique pour définir les propriétés structurelles que doivent respecter les noyaux des modèles de contrôle d'accès. Il s'agit du premier pas vers la définition générique de l'intégrité des modèles et des politiques de contrôle d'accès. Certaines des classes de dépendances peuvent être directement prises en compte dans des systèmes de gestion de base de données grâce aux mécanismes de clefs primaires et étrangères.

Définition (Dépendances structurelles des modèles)**.** *Pour chaque relation $\mathcal{R}$ n-aire entre les concepts $\mathcal{C}_1, \ldots, \mathcal{C}_n$, chacun exprimé par une relation $C_n(ID_n)$, une dépendance génératrice*

de tuples totale de Σ_{edb} exprime que chaque identifiant de concept est clef étrangère :

$$R(ID_1, \ldots, ID_n) \rightarrow C_1(ID_1), \ldots, C_n(ID_n)$$

Exemple 4.2 Dépendances structurelles du modèle RBAC$_0$

Le schéma relationnel de RBAC$_0$ est *edb* de $\mathbf{RBAC_0}$. Les dépendances génératrices de tuples de Σ_{edb} permettent de s'assurer que les relations entre les concepts sont bien fondées :
- $Habilite(U, R) \rightarrow User(U), Role(R)$,
- $Représente(S, U) \rightarrow Sujet(S), User(U)$,
- $Endosse(S, R) \rightarrow Sujet(S), Role(R)$,
- $Affecte(R, A, O) \rightarrow Role(R), Action(A), Objet(O)$.

Les dépendances structurelles permettent de s'assurer qu'une politique est bien structurée selon le modèle dont elle est une instance. Cette approche permet de rassembler dans un même cadre deux notions qui sont parfois traitées indépendamment :

- la *confidentialité* des données : l'objectif principal du contrôle d'accès est de n'autoriser que les ayants droit à exécuter des actions sur les objets du système selon une politique. Une partie de notre travail est de structurer cette politique,
- l'*intégrité* des données : les dépendances de données ont été proposées initialement pour s'assurer que les données enregistrées dans une base de données sont correctes. Nous transposons cette approche au contrôle d'accès : nous définissions quels sont les états légitimes des politiques à l'aide des dépendances.

Nous avons proposé une définition des propriétés que les schémas relationnels de contrôle d'accès doivent respecter. On peut cependant en ajouter d'autres spécifiques à chaque modèle. Il a été remarqué qu'il peut être profitable de proposer une notion de session dans les modèles RBAC plus restreinte que celle existante, dans laquelle un utilisateur ne peut disposer que d'un seul sujet [Li07] :

> ***Suggestion 2*** *The standard should accommodate* RBAC *systems that allow only one role to be activated in a session.*

Il s'agit d'un des cas où un modèle de contrôle d'accès peut donner trop de liberté, ce qui rend difficile son implémentation. Dans notre cadre, cette restriction est exprimable par une dépendance génératrice d'égalité, que l'on ajoute à la modélisation existante des modèles RBAC :

$$Endosse(User, Role_1), Endosse(User, Role_2) \rightarrow Role_1 = Role_2$$

4.3 Principes des modèles de contrôle d'accès

Cette section définit les principes des modèles de contrôle d'accès. Il s'agit d'exprimer comment dériver les relations en intention, comme le triplet fondamental ou la hiérarchisation de concepts, à partir de celles en extension de la section précédente. Cette section définit les composantes idb, Σ_{idb}, P et $\mathbf{I}'$ de la structuration relationnelle **AC** proposée. Le schéma en intention idb est défini par un ensemble P de règles de déduction. On peut utiliser le modèle DATALOG ou DATALOG^C pour exprimer ces règles, mais l'on pourrait également utiliser tout fragment de la logique dont une plus petite interprétation unique $\mathbf{I}'$ calculable en un nombre fini d'étapes existe.

4.3.1 Dérivation du triplet fondamental d'autorisation

Après avoir structuré les droits des utilisateurs dans le système, nous allons définir comment dériver, à partir de la politique, le triplet d'autorisation utilisé par le moniteur.

Définition (Dérivation du triplet fondamental d'autorisation)**.** *Le* triplet fondamental *d'autorisation $\mathcal{ACCESS} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{O}$, est dérivé par un ensemble $\Delta \subseteq P$ de règles de déduction, appelées* core-rules*. Pour chaque règle $\delta_i \in \Delta$, la relation en intention Accès apparaît en tête et des relations du modèle de contrôle d'accès apparaissent dans le corps :*

$$\psi_i \rightarrow Accès(Sujet, Action, Objet)$$

Ces règles sont également l'expression de jointures naturelles entre les relations apparaissant dans ψ_i. Ces jointures sont satisfaites par $\mathbf{I}'$. Cette définition généralise les règles proposées pour définir les principes des modèles de contrôle d'accès. Pour les modèles basés sur l'identité, la définition reste valide, mais le corps est vide. En effet, ces modèles sont peu structurés et on enregistre directement les triplets d'autorisation sans l'aide de concepts intermédiaires.

Pour plusieurs modèles de contrôle d'accès, l'ensemble Δ est réduit à une seule *core-rule*. Dans le cas général plusieurs de ces règles peuvent exister. La tableau 3.2 a donné la *core-rule* du modèle RBAC$_0$, qui définit la relation $Accès(S, A, O)$ comme la jointure des relations $Endosse(S, R)$ et $Affecte(R, A, O)$. L'ensemble Δ est dans ce cas composé d'une seule règle. Nous proposons comme autre exemple le modèle ORBAC présenté en section 2.3.3.1. Si l'on ne tient pas encore compte de la notion de contexte, la *core-rule* du modèle ORBAC est définie par la règle suivante [Miège05] :

$$Habilitation(Org, Role, Sujet), Consideration(Org, Activite, Action),$$
$$Utilisation(Org, Vue, Objet), OPermission(Role, Activite, Vue) \rightarrow$$
$$Accès(Sujet, Objet, Action)$$

4.3.2 Dérivation des triplets orientés utilisateurs

En plus du triplet fondamental d'autorisation, il est intéressant de dériver les deux autres triplets orientés utilisateurs $\mathcal{DYNAMIC}$ et $\mathcal{STATIC}$. Le second est nécessaire pour la comparaison *statique* de politiques.

Définition (Dérivation des triplets d'autorisation orientés utilisateurs)**.** *Le* triplet d'autorisation orienté utilisateurs statique $\mathcal{STATIC} \subseteq \mathcal{U} \times \mathcal{A} \times \mathcal{O}$ *(resp.* dynamique $\mathcal{DYNAMIC} \subseteq \mathcal{U} \times \mathcal{A} \times \mathcal{O}$*) sont dérivés par un ensemble de règles de déduction* $\Delta^S \subseteq P$ *(resp.* $\Delta^D \subseteq P$*). Pour chaque règle* $\delta_i^S \in \Delta^S$ *(resp.* $\delta_i^D \in \Delta^D$*) la relation en intention Statique (resp. Dynamique) apparaît en tête et des relations du modèle de contrôle d'accès apparaissent dans le corps :*

$$\begin{array}{rcl} \psi_i^S & \rightarrow & Statique(Utilisateur, Action, Objet) \\ \psi_i^D & \rightarrow & Dynamique(Utilisateur, Action, Objet) \end{array}$$

La tableau 3.2 a donné les règles qui permettent de dériver les triplets orientés utilisateurs à partir des relations du modèle RBAC_0. Les triplets orientés utilisateurs sont utiles pour comparer les permissions accordées aux usagers des systèmes et non aux entités logicielles qui les représentent. Comme nous avons imposé que chaque sujet ne soit associé qu'à un et un seul utilisateur, les permissions autorisées d'un utilisateur à un instant donné par l'intermédiaire des sujets qu'il endosse doivent être incluses dans l'ensemble des permissions dont l'utilisateur dispose, soit $\mathcal{DYNAMIC} \subseteq \mathcal{STATIC}$. La définition suivante exprime ce fait.

Définition (Inclusion des autorisations orientées utilisateurs)**.** *À partir des core-rules* Δ *et des dépendances satisfaites dans le modèle, la dépendance suivante* τ*, appelée* inclusion des autorisations orientées utilisateurs *doit être satisfaite par* $\mathbf{I}'$ *:*

$$Dynamique(U, A, O) \rightarrow Statique(U, A, O)$$

Les procédures de preuves de l'implication logique des dépendances peuvent être mises à profit pour la définition de τ. En effet cette dépendance *doit* être satisfaite par $\mathbf{I}'$, deux cas peuvent se présenter. Soit τ est définie *explicitement* et fait partie de Σ, soit τ est déjà satisfaite *implicitement*, si $\Sigma \cup P \models \tau$. Dans ce second cas, nul besoin de l'ajouter car cela reviendrait à ajouter une expression redondante à Σ. Pour le modèle RBAC_0, nous avons montré que τ est implicitement satisfaite par la définition logique des principes et des dépendances structurelle du modèle. Nous montrons comment le *chase* permet d'effectuer cette vérification en annexe C.

4.3.3 Principe fondamental des modèles structurés

Nous introduisons maintenant une restriction sur les politiques de contrôle d'accès interdisant l'*affectation directe* de permissions. En effet, l'intérêt des modèles structurés est d'introduire des concepts intermédiaires entre sujets et permissions pour faciliter l'administration des politiques. Permettre la définition directe des triplets d'autorisation, non seulement fait perdre l'intérêt de la structuration, mais surtout complique l'analyse des politiques par les administrateurs et est une source considérable d'erreurs que l'on souhaite éviter.

Définition (Principe fondamental des modèles structurés)**.** *Dans tout modèle de contrôle d'accès où des concepts et relations intermédiaires sont définis entre sujets et utilisateurs d'une part, et permissions d'autre part, il est interdit d'affecter directement des permissions.*

Les modèles de contrôle d'accès structurés doivent satisfaire les dépendances du non-court-circuit*, où les variables apparaissant seulement en tête sont quantifiées existentiellement ($\tilde{\exists}$). Pour chaque dépendance δ_i, δ_i^S et δ_i^D appartenant respectivement aux ensembles Δ, Δ^S et Δ^D, il faut ajouter à Σ les dépendances suivantes :*

$$\begin{aligned} Accès(Sujet, Action, Objet) &\rightarrow \tilde{\exists}\, \psi_i \\ Statique(Utilisateur, Action, Objet) &\rightarrow \tilde{\exists}\, \psi_i^S \\ Dynamique(Utilisateur, Action, Objet) &\rightarrow \tilde{\exists}\, \psi_i^D \end{aligned}$$

Les dépendances σ_3, σ_4 et σ_5 du tableau 3.9 sont des exemples de dépendances du non-court-circuit. Elles expriment le fait que l'on ne puisse pas définir d'autorisations dans les modèles RBAC sans utiliser de rôles. Pour le cas limite des modèles basés sur l'identité, les dépendances du non-court-circuit sont des tautologies triviales.

4.3.4 Hiérarchisation des concepts

La hiérarchisation des concepts est un outil qui permet de limiter la redondance des données de contrôle d'accès et de faciliter l'administration des politiques en calquant leurs structures sur celle des organisations. L'objectif de cette section est de donner une caractérisation générale, qui peut être utilisée pour définir des hiérarchies de concepts multiples dans les modèles de contrôle d'accès. Dans les modèles RBAC, ce sont les rôles qui sont hiérarchisés, dans les modèles MAC, ce sont les labels, dans les modèles TBAC, ce sont les tâches. À l'aide de hiérarchies multiples, nous pouvons hiérarchiser les objets en arbre pour représenter la structure d'un système de fichier.

Définition (Concept hiérarchisé)**.** *Un concept $\mathcal{C}$, exprimé par une relation unaire $C(ID)$, d'un modèle de contrôle d'accès est* hiérarchisé *si une relation binaire transitive, réflexive et antisymétrique est définie sur $\mathcal{C} \times \mathcal{C}$. La relation hiérarchique est définie en* intention *comme la fermeture transitive et réflexive d'une relation de* domination *directe en* extension*.*

Soit $Domine_C$ la relation de domination en extension. Soit $Hérite_C$ la relation d'héritage en intention correspondante définie par les règles de déduction :

$$\begin{array}{lrcl} \lambda_R & C(ID) & \rightarrow & Domine_C(ID, ID) \\ \lambda_H & Domine_C(ID_1, ID_2) & \rightarrow & Hérite_C(ID_1, ID_2) \\ \lambda_T & Domine_C(ID_1, ID_2), Hérite_C(ID_2, ID_3) & \rightarrow & Hérite_C(ID_1, ID_3) \end{array}$$

et la dépendance génératrice d'égalité :

$$\lambda_A \quad Hérite_C(ID_1, ID_2), Hérite_C(ID_2, ID_1) \rightarrow ID_1 = ID_2$$

Exemple 4.3 Les multiples hiérarchies du GMSIH

Comme nombre d'établissements de santé, le CHM est membre du Groupement pour la Modernisation du Système d'Information Hospitalier (GMSIH). Dans le modèle de contrôle d'accès que ce groupement propose, deux hiérarchies sont présentes : celle des rôles et celle des structures.

L'objectif de cette modélisation est de permettre d'organiser les droits selon la décomposition en structures et sous-structures des d'établissements de santé. Cette modélisation permet d'associer à un même utilisateur des rôles différents dans des structures différentes. Cependant, alors que la hiérarchie des rôles est un ordre partiel général, la hiérarchie des structures doit former un arbre, ce qui peut être garanti par des dépendances.

À l'aide des dépendances, nous pouvons restreindre les hiérarchies de concepts. En effet, selon les variantes des modèles, ou selon les restrictions imposées par les implémentations des mécanismes de contrôle d'accès, il peut être nécessaire de limiter la hiérarchie à un arbre, un arbre inverse ou un treillis (section 2.3.2). Nous définissons les dépendances qui permettent d'exprimer ces contraintes.

Définition (Hiérarchie limitée)**.** *Soit $Domine_C \subseteq \mathcal{C} \times \mathcal{C}$ une relation de domination en extension sur un concept $\mathcal{C}$ et $Hérite_C$ la relation d'héritage en intention associée, elle est :*

– *limitée en arbre si on ajoute la dépendance fonctionnelle :*

$$\lambda_a : Domine_C(ID, ID_1), Domine_C(ID, ID_2) \rightarrow ID_1 = ID2$$

– *limitée en arbre inverse si on ajoute la dépendance fonctionnelle :*

$$\lambda_i : Domine_C(ID_1, ID), Domine_C(ID_2, ID) \rightarrow ID_1 = ID2$$

– *limitée en treillis si on ajoute les dépendances génératrices de tuples :*

$$\begin{array}{l} \lambda_t : C(ID_1), C(ID_2) \rightarrow \exists ID_\perp \; Hérite_C(ID_1, ID_\perp), Hérite_C(ID_2, ID_\perp) \\ \lambda_b : C(ID_1), C(ID_2) \rightarrow \exists ID_\top \; Hérite_C(ID_\top, ID_1), Hérite_C(ID_\top, ID_2) \end{array}$$

Un modèle MAC est régi par deux *core-rules* présentées par le tableau 2.4 (section 2.2.2) : « un sujet peut `lire` des informations dont la classification est inférieure ou égale à son accréditation » (*No read up*) et « un sujet peut `écrire` des informations dont la classification est supérieure ou égale à son accréditation » (*No write down*). À la différence des modèles à rôles où les *core-rules* dérivent des permissions, les règles des modèles MAC déterminent qu'elles sont les *actions* autorisées.

C'est donc à partir de la hiérarchisation des labels que l'on peut déterminer les droits de lecture et d'écriture des sujets. L'annexe B propose une formalisation des modèles MAC. On définit la relation de domination en extension $Domine_L$, la relation en intention $Hérite_L$ associée puis formaliser les principes des modèles MAC avec les deux règles suivantes :

$$\begin{array}{lcl} Accred(S, L_1), Hérite_L(L_1, L_2), Classif(O, L_2) & \rightarrow & Accès(S, \texttt{read}, O) \\ Accred(S, L_1), Hérite_L(L_2, L_1), Classif(O, L_2) & \rightarrow & Accès(S, \texttt{write}, O) \end{array}$$

Pour des raisons de lisibilité et de maintenabilité des principes d'un modèle, il peut être judicieux de ne pas faire intervenir directement la hiérarchisation des rôles dans l'expression des *core-rules*. On peut alors introduire des relations en intention intermédiaires utiles pour structurer la théorie logique $P \cup \Sigma$.

4.4 Catégorisation des aspects des modèles

4.4.1 Aspects statiques et dynamiques

Les définitions proposées n'ont font pas de distinctions explicites entre les aspects *statiques* et *dynamiques* des modèles de contrôle d'accès. Cette différenciation, faite dans les modèles RBAC, n'est pas toujours proposée dans les autres modèles. Dans la structuration relationnelle que nous proposons, les différences principales entre les aspects statiques et dynamiques sont :

- la *fréquence* de mise à jour des relations : les aspects statiques stockent les droits des utilisateurs de façon structurée. Dans les modèles RBAC ils comprennent par exemple les définitions des utilisateurs, des actions, des objets. Il s'agit donc de données stables vis-à-vis de l'exécution du système, qui ne sont pas modifiées chaque fois qu'un utilisateur ouvre une session. Le contenu des concepts et relations dynamiques quant à lui va être modifié à chaque fois qu'un utilisateur décide de s'identifier dans le système,
- les *droits* sur les relations : les utilisateurs finaux n'ont pas de droits d'administration sur la politique de sécurité et ne peuvent pas modifier le contenu de la politique. En revanche, par l'intermédiaire du gestionnaire de sessions, ils vont ajouter (à l'ouverture), modifier (en choisissant) et supprimer (à la fermeture) des tuples dans les concepts et relations dynamiques.

	Statique	**Dynamique**	
Extention	Factuel statique, $\mathbf{I_s}$	Factuel dynamique, $\mathbf{I_d}$	$\mathbf{I = I_s \cup I_d}$
Intension	Déduit statique, $\mathbf{I'_s}$	Déduits dynamique, $\mathbf{I'_d}$	$\mathbf{I' = I'_s \cup I'_d}$

TAB. 4.2 – Catégorisation des aspects des politiques de contrôle d'accès

Définition (Aspects statiques et dynamiques)**.** *Un aspect (concept, ou relation) d'un modèle de contrôle d'accès est dit* statique *si et seulement si* seuls *les administrateurs peuvent définir et modifier les instances ou s'il est déduit* uniquement *d'autres aspects statiques.*

Un aspect d'un modèle de contrôle d'accès est dit dynamique *si les instances peuvent être définies et modifiées par d'autres acteurs que les administrateurs ou s'il est déduit à partir d'autres aspects dynamiques.*

Nous entendons par autres *acteurs* les utilisateurs du système ou tout mécanisme mis en œuvre dans les moniteurs. Cette distinciton permet de séparer les aspects qui sont *stables* dans une politique de contrôle d'accès de ceux qui vont être modifiés fréquemment hors du contrôle des administrateurs. Les principaux concepts qui permettent de structurer les droits sont généralement statiques. Les rôles dans les modèles RBAC, les tâches dans TBAC, les zones géographiques dans GEORBAC et les organisations dans ORBAC sont tous statiques.

La différenciation en aspects statiques et dynamiques n'influe pas sur la formalisation des modèles de contrôle d'accès. En revanche, la différenciation doit être prise en compte dans les méthodologies de conception de politiques et surtout lors de l'implémentation des modèles dans les systèmes. La séparation entre ce qui varie lors de l'exécution de ce qui est stable est un principe général de modélisation en informatique [Meyers05].

Tout modèle de contrôle d'accès devrait faire l'analyse de la variabilité. Selon les utilisations d'un modèle, il peut être intéressant de n'intégrer que ses aspects statiques. La distinction entre statique et dynamique conduit à mieux structurer les noyaux des modèles. Pour les modèles RBAC, dont le seul concept dynamique est celui de session, cette distinction n'a pas toujours été faite. Dans la critique du standard, on peut lire la suggestion suivante [Li07] :

> ***Suggestion 1*** *The notion of sessions should be removed from Core* RBAC *and introduced in a separate component.*

Nous intégrons cette suggestion en divisant les modèles en composants aux objectifs clairement définis et séparés. Nous allons également introduire une nouvelle division des composants entre factuels et déduits qui correspond également à une des critiques formulées à l'encontre des modèles RBAC.

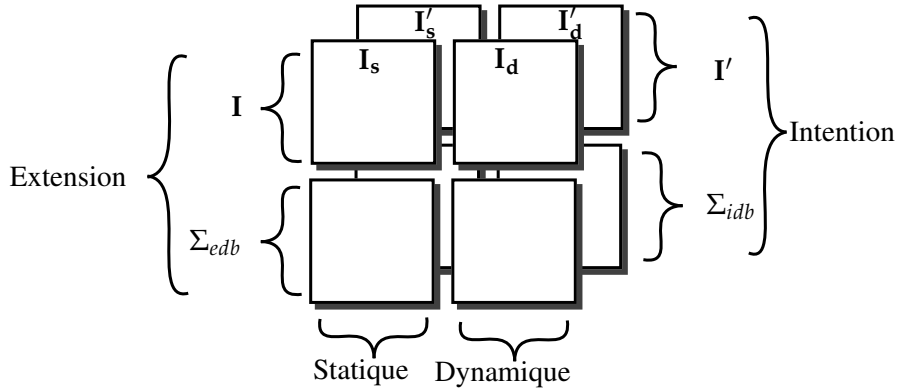


FIG. 4.1 – Catégorisation des composants de la structuration relationnelle

4.4.2 Aspects factuels et déduits

La catégorisation en aspects statiques et dynamiques nous permet de diviser les propriétés que respectent les instances $\mathbf{I}$ et $\mathbf{I}'$ entre celles qui doivent être vérifiées lors des opérations administratives de celles qui doivent l'être lors des accès au moniteur de référence. Cette catégorisation est orthogonale à différenciation entre extension (factuel $\mathbf{I}$) et intention (déduit $\mathbf{I}'$) comme l'illustre le tableau 4.2. Cette séparation est :

- *naturelle*, car elle suit l'évolution des recherches sur le contrôle d'accès, où l'on a commencé par structurer les droits , exprimer des relations déduites puis imposer des restrictions garantissant la validité des politiques [Halpern03],
- *utile*, car cette séparation permet de développer une méthodologie de conception et d'évolution des modèles de contrôle d'accès. De plus, les aspects déduits sont dérivables de façon automatisée et les administrateurs ne doivent pas y avoir d'accès direct hormis pour l'interrogation.

La séparation entre factuel et déduit n'a pas été clairement identifiée en compte dans les modèles RBAC, ce qui conduit à des difficultés de modélisation et d'implémentation. Nous citons deux des critiques adressées au standard à ce sujet [Li07] :

> ***Suggestion 3*** *The standard should make a clear distinction between base relations and derived relations.*
> ***Suggestion 4*** *The Reference Model should maintain a relation that contains the role dominance relationships that have been explicitly added, and update this relation when the role hierarchy changes.*

Cette distinction a été faite dès la présentation du cadre que nous proposons. La notion de *role dominance relationship* est ce que nous avons dénommé une relation de domination $Domine_C$, à partir de laquelle est dérivée une relation d'héritage $Domine_C \subseteq Hérite_C$. Comme il est indiqué par la suggestion quatre, cette distinction permet de définir ce qui doit être recalculé lors des opérations administratives de ce qui n'a pas besoin de l'être.

Les vérifications qui portent sur les aspects statiques permettent de valider une politique et assurer son intégrité quelles que seront les utilisations faites du système. On peut donc diviser les instances $\mathbf{I}$ et $\mathbf{I}'$ en deux sous-ensembles : un premier comprenant les données qui concernent les concepts et relations statiques, un second qui concerne les dynamiques. À chacune des quatre composantes du tableau 4.2 nous associons un sous-ensemble de dépendances de Σ qui définit les états intègres de $\mathbf{I_s}$, $\mathbf{I_d}$, $\mathbf{I'_s}$ et $\mathbf{I'_d}$. Cette correspondance est représentée par la figure 4.1. On divise donc la notion générale de politique en quatre composants :

- $\mathbf{I_s}$ est la politique en extension statique que seuls les administrateurs ont le droit de manipuler. Pour le modèle RBAC_1, il s'agit des relations *Habilite* et $Affecte$ de la politique du tableau 3.1,
- $\mathbf{I_d}$ est la politique en extension dynamique qui comporte les relations que les utilisateurs peuvent manipuler. Ce composant comporte le concept de sujet ainsi que les relations qui le lient avec d'autres concepts. Pour le modèle RBAC_1, il s'agit des relations *Endosse* et *Représente* de la politique du tableau 3.1,
- $\mathbf{I'_s}$ est la politique en intention statique dérivée à partir de $\mathbf{I_s}$ uniquement. Pour le modèle RBAC_1, la relation *Statique* du tableau 3.3 est un exemple d'instance appartenant à $\mathbf{I'_s}$. Les administrateurs n'ont pas d'accès à $\mathbf{I'_s} \setminus \mathbf{I_s}$,
- $\mathbf{I'_d}$ est la politique en intention dynamique dérivée à partir de $\mathbf{I_s}$ et de $\mathbf{I_d}$. Les triplet d'autorisations *Accès* et *Dynamique* font partie de cet ensemble. Les administrateurs n'ont pas d'accès à $\mathbf{I'_d} \setminus \mathbf{I}$.

4.5 Contraintes dans les modèles de contrôle d'accès

Plusieurs propositions ont exprimé des contraintes sur les politiques de contrôle d'accès, dans le but de limiter les privilèges des utilisateurs. Nous avons remarqué que selon les domaines d'application, l'exclusion mutuelle peut être définie différemment. Nous proposons une modélisation générique de la notion de contrainte dans les modèles de contrôle d'accès. Cette définition permet de représenter les différentes notions d'exclusions mutuelles entre concepts et d'expliciter la différence entre ce type de contraintes et les contraintes dites de prérequis.

4.5.1 Modélisation avec les dépendances

Définition (Contrainte dans un modèle de contrôle d'accès)**.** *Une* contrainte *dans un modèle de contrôle d'accès est une dépendance génératrice de contraintes, où $\psi(X)$ est une conjonction d'atomes et $c(\tilde{X})$ et $c'(\tilde{X})$ des conjonctions de contraintes*[4] *:*

$$\psi(X), c(\tilde{X}) \rightarrow c'(\tilde{X})$$

[4] $c'(\tilde{X})$ peut éventuellement être une antilogie $\perp$ dans le cas d'une NGD.

Plusieurs modélisations logiques du contrôle d'accès ont représenté l'antilogie $\perp$ par un prédicat 0-aire $Error()$ et des règles DATALOG de la forme $\psi(X), c(\tilde{X}) \rightarrow Error()$. La vérification de l'intégrité d'une politique consiste alors à vérifier que le prédicat $Error()$ n'appartient pas à $\mathbf{I}'$. Il devient alors difficile d'identifier dans l'ensemble de formules logiques qui composent un modèle de contrôle d'accès, celles qui expriment la déduction de faits, de celles qui permettent de garantir qu'un modèle soit valide. Notre approche quant à elle sépare :

- la *structure* du modèle de contrôle d'accès : son noyau,
- son *contenu* : les politiques instances du modèle,
- ses *principes* : les règles qui permettent de dériver les autorisations,
- les *propriétés* du modèle : les dépendances qui définissent es politiques valides.

4.5.2 Contraintes d'exclusion mutuelle

Définition (Exclusion mutuelle entre concepts)**.** *Un concept $\mathcal{C}$, exprimé par une relation unaire $C(ID)$, d'un modèle de contrôle d'accès est muni d'une* relation d'exclusion mutuelle *si une relation binaire irréflexive et symétrique est définie sur $\mathcal{C} \times \mathcal{C}$. La relation d'exclusion mutuelle est définie en* intention *comme la fermeture symétrique d'une relation de* d'exclusion directe *en* extension*.*

Soit $Séparation_C$ la relation d'exclusion directe en extension, alors la relation d'exclusion mutuelle en intention $Séparation_C \subseteq Exclusion_C$ est définie par les règles de déduction :

$$\begin{array}{rcl} Séparation_C(ID_1, ID_2) & \rightarrow & Exclusion_C(ID_1, ID_2) \\ Exclusion_C(ID_1, ID_2) & \rightarrow & Exclusion_C(ID_2, ID_1) \end{array}$$

et la dépendance génératrice de nullité :

$$Exclusion_C(ID, ID) \rightarrow \perp$$

Nous définissons ainsi les propriétés algébriques qu'une relation d'exclusion mutuelle doit respecter. La définition suivante se base sur la précédente et définit comment une relation d'exclusion mutuelle permet de limiter les instances valides d'une politique de contrôle d'accès.

Définition (Sémantique de l'exclusion mutuelle)**.** *La* sémantique *associée à une relation d'exclusion mutuelle $Exclusion_C$ est exprimable par une dépendance génératrice de nullité, où $\psi(X_1)$ est une conjonction de relations, R une relation entre $\mathcal{C}$ et d'autres concepts, $c(\tilde{X})$ une conjonction contraintes sur les termes de l'hypothèse :*

$$\psi(X_1), R(\ldots ID_1 \ldots), R(\ldots ID_2 \ldots), Exclusion_C(ID_1, ID_2), c(\tilde{X}) \rightarrow \perp$$

Les dépendances γ_1 à γ_5 du tableau 3.10 définissent différentes sémantiques possible de l'exclusion mutuelle dans les modèles RBAC_2. On peut ainsi exprimer que la contrainte est statique, dynamique, orientée permission, objet, ou toute autre signification donnée à la relation d'exclusion dans le modèle de contrôle d'accès (section 3.4.6).

4.5.3 Exclusion en présence de hiérarchies

Il a été remarqué que les contraintes d'exclusion et les hiérarchies peuvent interagir lorsque l'exclusion et la hiérarchie portent sur un même concept, celui de rôle en particulier [Gavrila98, Gligor98]. Si les interactions avaient été identifiées dans les modèles RBAC, elles n'ont pas été prises en compte dans les autres modèles de contrôle d'accès, alors que plusieurs d'entre eux proposent des contraintes d'exclusion et des hiérarchies portant sur les mêmes concepts [Miège05, Barker03]. La généralisation que nous proposons permet de se prémunir contre d'éventuels écueils d'administration.

Définition (Transmission de l'exclusion par hiérarchie)**.** *Soit une* $Hérite_C$ *relation hiérarchique sur un concept* C*, exprimé par une relation unaire* $C(ID)$*, et une relation d'exclusion mutuelle sur ce même concept* $Exclusion_C$*. Alors, la dépendance génératrice suivante, appellée* transmission de l'exclusion par hiérarchie *doit être satisfaite par* $\mathbf{I}'$ *en l'ajoutant à* P *:*

$$Exclusion_C(ID_1, ID_2), Hérite_C(ID, ID_1) \rightarrow Exclusion_C(ID, ID_2)$$

Dans les modèles $RBAC_3$, les utilisateurs membres d'un rôle r sont implicitement affectés aux rôles parents de r. Par exemple, si $r \succ\succ r_1$ et $r \succ\succ r_2$, alors $auth\_users(r_1) \cup auth\_users(r_2) \subseteq auth\_users(r)$. La dépendance précédente permet de transmettre l'exclusion par l'intermédiaire de la relation d'héritage. Si r_1 et r_2 sont en exclusion mutuelle, alors la politique devient invalide [Gavrila98].

4.5.4 Relation de prérequis

Notre approche permet de définir les différentes formes de la relation d'exclusion mutuelle selon les relations sur lesquelles elles portent, mais elle permet également de définir la notion de *prérequis*. Cette notion a été introduite dans les premières propositions de RBAC [Sandhu96], mais ne font pas partie du standard. Certains auteurs les ont cependant pris en compte dans leur modélisation [Hansen05]. Nous proposons dans cette section des définitions de la notion de prérequis modélisée par des dépendances.

Définition (Relation de prérequis)**.** *Une* relation de prérequis *dans un modèle de contrôle d'accès est exprimable par une dépendance génératrice de tuples contraintes de la forme suivante, exprimant que si l'hypothèse (non vide) est vraie dans la politique, alors la conclusion (non vide également) doit l'être aussi :*

$$\forall \tilde{X}\ R_1(X_1), \ldots, R_n(X_n), c(\tilde{X}) \rightarrow \exists \tilde{Z}\ Q_1(Y_1), \ldots, Q_m(Y_m), c'(\tilde{Y})$$

Les relations de prérequis sont utilisées dans les modèles de contrôle d'accès pour l'expression de *conditions* à respecter avant l'exécution d'opérations administratives sur les hiérarchies. Dans les modèles RBAC les conditions de prérequis sont définies sur les relations du noyau : l'affectation de rôles aux utilisateurs (*Habilite*) et la relation d'héritage des rôles (*Hérite*) [Sandhu99, Ferraiolo03b].

Définition (Relation de prérequis intra-relation)**.** *Une relation de prérequis* intra-relation*, porte sur une unique relation d'un schéma de contrôle d'accès. Elle est exprimable par une dépendance de la forme suivante, où le quantificateur existentiel porte sur les variables de la conclusion qui n'apparaissent pas en hypothèse :*

$$\forall \tilde{X}\ R(X_1), \ldots, R(X_n), c(\tilde{X}) \rightarrow \exists \tilde{Z}\ R(Y), c'(\tilde{Y})$$

Le pouvoir d'expression des classes de dépendances comme les dépendances génératrices de tuples contraintes est nécessaire pour la modélisation des relations de prérequis. Cette classe permet en effet d'exprimer des prérequis qui ne sont pas modélisables à l'aide de règles DATALOG^C. Le tableau 3.6 résume les formes syntaxiques logiques auxquelles se réfèrent les dépendances, on remarque que les plus générales d'entre elles autorisent la présence de variables quantifiées *existentiellement* en conclusion et de *contraintes* sur les termes de la conclusion, y compris ceux quantifiés existentiellement. Cette expressivité nous permet de formaliser un aspect des extensions de RBAC : le typage des rôles en sous-catégories à l'aide de prérequis intra-relation.

4.5.5 Combinaison de prérequis et de hiérarchies

Utilisés conjointement avec les relations de domination, les prérequis permettent la formalisation de définitions alternatives des relations d'héritage entre concepts. Dans les modèles basés sur les rôles, il a été proposé par exemple de spécialiser la relation d'héritage de rôles en sous-relations qui imposent de nouvelles propriétés. On peut différencier l'héritage selon que la hiérarchie de rôles représente [Crook03, Joshi02] :

- une hiérarchie *fonctionnelle*. C'est la hiérarchie *est-un* des professionnels de santé que nous utilisons dans les exemples : médecin, généraliste, spécialiste, ...
- une hiérarchie d'*autorité*. Un exemple de hiérarchie d'autorité est l'ordre total des grades dans certaines administrations publiques : commissaire, inspecteur, préposé puis agent. Cet ordre est modélisable par une hiérarchie, mais conceptuellement elle ne représente pas une relation *est-un*,
- une hiérarchie *structurelle*. Certains modèles proposent un organigramme des services pour structurer les droits par exemple. Conceptuellement, il s'agit souvent d'une relation d'inclusion. Une relation d'inclusion respecte les mêmes propriétés algébriques qu'une relation hiérarchique.

Grâce aux classes de dépendances générales, il est possible d'exprimer les différentes propriétés sous-entendues par les spécialisations du concept de rôles. Plusieurs hiérarchies aux sémantiques différentes peuvent être développées sur un même concept. Les différences entre elles peuvent être exprimées comme des propriétés supplémentaires que les relations de domination directes respectent.

Définition (Relation d'héritage spécialisée)**.** *Soient $Domine_C$ une relation de domination sur un concept C et $Hérite_C$ sa relation d'héritage dérivée. La relation $Hérite_C$ est dite* relation

d'héritage spécialisée *si une contrainte de prérequis intra-relation de la forme suivante est respectée, où $R(\ldots ID_1 \ldots)$ est une relation entre $\mathcal{C}$ et d'autres concepts :*

$$Domine_C(ID_1, ID_2), R(\ldots ID_1 \ldots), c(\tilde{X}) \rightarrow \exists ID\ R(\ldots ID \ldots), c'(\tilde{Y})$$

Selon la relation R utilisée dans la définition d'une relation d'héritage spécialisée, on peut exprimer des sémantiques complexes des relations d'héritage. Dans les modèles à rôles, une hiérarchie d'autorité définit une contrainte sur l'héritage à l'aide de la relation d'affectation des rôles aux utilisateurs. On peut ainsi exprimer « qu'il n'y a jamais de subordonné sans chef ». Pour les hiérarchies fonctionnelles ou structurelles, on peut à l'aide des relations d'héritage contraintes exprimer des dépendances sur la composition des équipes ou des services.

Exemple 4.4 Il n'y a jamais de subordonné sans chef

Au CHM, le directeur d'établissement peut être assisté par un ou plusieurs directeurs adjoints auxquels il délègue une partie de ses attributions. Imaginons que le CHM mette en œuvre une extension du modèle de contrôle d'accès RBAC_3 qui intègrerait hiérarchie de rôles, exclusion mutuelle et prérequis.

Les administrateurs souhaitent spécifier une contrainte exprimée en langue naturelle par « s'il existe un directeur adjoint dans l'organisation, alors il existe aussi un *autre* utilisateur directeur ». On peut ainsi poser une contrainte *Nécessite* sur la relation de domination entre rôles *Domine* qui permet de la spécialiser :

$$Nécessite(Sub, Sup), Domine(Sup, Sub), Habilite(U, Sub) \rightarrow \exists U' Habilite(U', Sub)$$

Pour la définition de *Nécessite* on s'appuie sur une relation en extension *Dépend*. On définit alors la relation *Nécessite* comme transitive avec

$$Dépend(R_1, R_2), Nécessite(R_2, R_3) \rightarrow Nécessite(R_1, R_3)$$

L'expression de $Dépend(\texttt{adjoint}, \texttt{directeur})$ dans la politique permet d'imposer un directeur en présence d'un adjoint. Ce type de contrainte est exprimable en utilisant une dépendance génératrice de tuples contraintes, ce qui n'est pas faisable en DATALOG^C ou dans les cadres théoriques qui se réduisent à ce dernier.

On pourrait également définir « qu'il n'existe pas d'anesthésiste sans chirurgien » ,$Dépend(\texttt{anesthésiste}, \texttt{chirurgien})$, ou toute autre propriété qui nécessite l'utilisation de la quantification existentielle dans sa modélisation logique.

4.6 Modélisation du contexte

Avec l'évolution des besoins et des technologies, les systèmes contemporains intègrent désormais espace, temps et d'autres données contextuelles. Ces informations permettent de caractériser les situations et les interactions des utilisateurs avec les systèmes. Dans certains domaines d'application, comme les systèmes d'informations mobiles par exemple, la prise en compte de ces aspects fait partie intégrante de l'architecture des systèmes. Dans l'état de l'art, nous avons présenté deux formalisations des aspects contextuels en DATALOG^C :

- une première où des concepts et relations de contexte sont utilisés dans l'expression des *core-rules*,
- une seconde basée sur le conditionnement des données factuelles de **I**.

Nous avons montré que ces deux approches ne sont pas pleinement satisfaisantes. La première brise la séparation entre aspects statiques et dynamiques : tous les triplets d'autorisation fondamentaux deviennent dynamiques. La seconde rompt la catégorisation entre aspects factuels et déduits. Or ces deux séparations (tableau 4.2) ont été identifiées comme fondamentales dans les modèles de contrôle d'accès [Li07].

4.6.1 Approche proposée

Nous proposons de modéliser le contexte comme des *restrictions* d'accès. Plutôt que de modifier les *core-rules* (première approche) ce qui alourdit la modélisation, ou de conditionner l'existence de tuples par des données contextuelles (seconde approche) ce qui impose des recalculs fréquents de $\mathbf{I}'$, nous proposons de représenter les contextes comme des *restrictions* sur les états intègres des politiques à vérifier à l'exécution seulement.

Dans la modélisation proposée, nous nous basons sur une politique *intègre* $\mathbf{I}'$, qui satisfait les dépendances du modèle que nous avons définies jusque ici $\mathbf{I}' \models \Sigma \cup P$. Cette politique définit l'ensemble des droits dont les utilisateurs disposent. Nous allons ajouter à Σ des dépendances pour *restreindre* l'ensemble des permissions *utilisables* selon des critères contextuels. En d'autres termes, une fois dérivées les matrices de contrôle d'accès, c'est-à-dire les triplets fondamentaux d'autorisations, nous allons limiter les accès des sujets hors contextes en interdisant l'accès à certaines colonnes de la matrice.

Définition (Contraintes d'accès contextuelles)**.** *Une* contrainte d'accès contextuelle *est modélisable par une dépendance génératrice de nullité, où* $\psi_{contexte}(X_1)$ *est une conjonction d'aspects* dynamiques *représentant les informations de contexte,* $\phi(X_2)$ *une conjonction de concepts et relations* dynamiques *du modèle de contrôle d'accès et* $c(\tilde{X})$ *une conjonction de contraintes sur des variables de* $\psi_{contexte}(X_1)$ *et* $\phi(X_2)$*. Les valeurs des variables de* $\psi_{contexte}(X_1)$ *à un instant donné forment un* état de contexte *:*

$$\psi_{contexte}(X_1), \phi(X_2), c(\tilde{X}) \rightarrow \bot$$

Utilisateur	Rôle
Alice	Infirmier
Alice	Médecin
Bob	Infirmier
Bob	Gastrologue
Charly	Infirmier
Charly	Pédiatre
Denise	Secrétaire

$Heure(H), H < 8,$	$Endosse(S, \texttt{Médecin}) \rightarrow \bot$
$Heure(H), H > 20,$	$Endosse(S, \texttt{Médecin}) \rightarrow \bot$
$Heure(H), H < 6,$	$Endosse(S, \texttt{Gastrologue}) \rightarrow \bot$
$Heure(H), H > 22,$	$Endosse(S, \texttt{Gastrologue}) \rightarrow \bot$
$Heure(H), H < 12,$	$Endosse(S, \texttt{Pédiatre}) \rightarrow \bot$
$Heure(H), H > 18,$	$Endosse(S, \texttt{Pédiatre}) \rightarrow \bot$

TAB. 4.3 – Instance de la relation *Habilite* et contraintes contextuelles

4.6.2 États de contexte

Illustrons la définition des états de contexte par une modélisation du temps reprise du modèle TRBAC. Nous supposons la relation *Habilite* d'affectation de rôles aux utilisateurs du tableau 4.3. Les lignes grisées de ce tableau indiquent des rôles qui peuvent être endossés uniquement pendant les horaires d'ouverture des services de soin concernés.

Nous exprimons ces restrictions par des dépendances portant sur la relation dynamique *Endosse* modélisant les rôles endossés dans les sessions. Nous supposons l'existence d'un concept de contexte $Heure(H)$ dont la valeur est l'heure actuelle. Les dépendances du tableau 4.3 modélisent ces restrictions, où $\psi_{contexte}$ est $Heure(H)$ et $c(\tilde{X})$ est composé d'inégalité sur les heures.

Avec cette approche, la vérification de contexte se fait donc à l'ouverture des sessions. Quand un utilisateur `u` ouvre une session `s`, dans laquelle il désire endosser le rôle `r`, il faut vérifier que l'ajout du tuple $Endosse(\texttt{s}, \texttt{r})$ ne viole pas les contraintes d'accès contextuelles[5]. Si tel était le cas, la demande de session initiée par l'utilisateur serait refusée.

Le tableau 4.4 est un extrait du tableau 3.3 limité aux utilisateurs `Alice` et `Bob`, dans lequel nous faisons figurer en gris les permissions qui sont accessibles qu'entre 8h et 20h. Nous y ajoutons également une colonne qui indique par quel rôle `Alice` ou `Bob` acquiert le privilège. Selon l'état de contexte, la représentation en tableau du triplet $\mathcal{U} \times \mathcal{A} \times \mathcal{O} \times \mathcal{R}$ sera composée de plus ou moins de lignes. Le principe consiste à limiter les instances autorisées des aspects dynamiques.

On peut imaginer que, afin de faciliter la gestion des contraintes contextuelles, on pourrait définir des règles DATALOG^C comme $Heure(H), H \leq 20, H \geq 8 \rightarrow HoraireOuverture()$, or cette modélisation impose de recalculer *à chaque changement* d'heure l'instance $\mathbf{I}'$. Notre proposition évite ce problème en modélisant les contraintes d'accès uniquement comme des instances invalides, sans passer par des règles de dé-

[5]Il faut, bien sûr, vérifier que le rôle `r` fait bien partie de ceux attribués à l'utilisateur.

Utilisateur	Action	Objet	Rôle
Alice	r	Fichier1	Infirmier
Alice	r	Fichier2	Infirmier
Alice	r	Fichier3	Infirmier
Alice	w	Fichier1	Médecin
Bob	r	Fichier1	Infirmier
Bob	r	Fichier2	Infirmier
Bob	r	Fichier3	Infirmier
Bob	r	Fichier4	Gastrologue
Bob	w	Fichier2	Gastrologue
Bob	w	Fichier4	Gastrologue
Bob	x	Fichier4	Gastrologue

TAB. 4.4 – Restrictions de permissions hors-contexte

duction. On pourrait éventuellement raffiner la catégorisation en aspects statiques et dynamiques pour séparer les aspects contextuels.

4.7 Discussion et synthèse

4.7.1 Limitations de la notion de contrainte

Un type de contrainte régulièrement proposé dans la littérature porte sur la *cardinalité*. Ces contraintes permettent de limiter le nombre d'instances de relations autorisées dans la politique de contrôle d'accès. Dans les implémentations des modèles RBAC, il est courant de pouvoir associer à chaque rôle une cardinalité, définissant le nombre maximum d'utilisateurs qui peuvent être affectés aux rôles. Les dépendances fonctionnelles permettent d'exprimer le cas dégénéré des contraintes d'unicité, cependant, dans le cas général, il est difficile de les modéliser [Calvanese94]. Une solution à ce problème serait des dépendances génératrices de la forme suivante, exprimant que le nombre de concepts reliés à c par la relation R est limité à n :

$$R(C_1, c), \ldots, R(C_{n+1}, c), C_1 \neq C_2, \ldots, C_1 \neq C_{n+1}, \ldots, C_n \neq C_{n+1} \rightarrow \bot$$

Or cette modélisation n'est pas satisfaisante. D'une part, le nombre d'inégalités dans le corps (égal à $C^2_{n+1} = n(n+1)/2$) rend la solution impraticable dès que n est grand. D'autre part nous perdons l'aspect déclaratif de la proposition : pour chaque contrainte de cardinalité il va falloir créer une dépendance spécifique. Ceci conduit à augmenter le nombre de dépendances et à compliquer le travail d'analyse des administrateurs de la politique, ce qui s'oppose aux objectifs que nous nous sommes fixés.

De plus, nous ne pouvons pas exprimer de relations entre les cardinalités, comme dans les modèles RBAC « tout rôle a une contrainte de cardinalité plus restrictive que

ceux dont il hérite ». En conclusion, nous reléguons à des mécanismes techniques la vérification des contraintes de cardinalité, et nous ne pourrons pas vérifier l'implication logique de propriétés faisant intervenir des contraintes de cardinalité.

4.7.2 Synthèse

Ce chapitre propose une généralisation des différentes notions développées dans les modèles de contrôle d'accès. À partir de définitions communes aux modèles de contrôle d'accès, issues de la littérature ou proposées à notre initiative, nous avons successivement définis :

1. la séparation entre les *modèles* et les *politiques* de contrôle d'accès,
2. la catégorisation en aspects *factuels* et *déduits* d'une part, et *statiques* et *dynamiques* d'autre part, qui permet de diviser les modèles de contrôle d'accès en composants,
3. les notions de *triplets d'autorisation*, qui sont à la base de tout modèle,
4. la notion de *schéma de contrôle d'accès* , la structure principale qui représente le noyau d'un modèle de contrôle d'accès,
5. les notions de *principes fondamentaux* et des *règles de dérivations*, qui expriment comment à partir d'un schéma dériver les triplets d'autorisation,
6. les notions de *hiérarchisation de concepts*, qui permettent de réduire le nombre de concepts et de faire correspondre la structuration des droits aux structures des organisations,
7. les notions de *contraintes* dans les modèles de contrôle d'accès comme l'exclusion mutuelle et les prérequis : des propriétés que les politiques doivent satisfaire,
8. la notion de *contexte* et d'*état de contexte*, en proposant une modélisation qui ne rompt pas avec la catégorisation des aspects des modèles.

Notre proposition est basée sur le cadre des dépendances. Ce cadre expressif permet de représenter de façon homogène les notions sus-citées. Informellement, nous dirions que nous avons fait correspondre des composants des modèles de contrôle d'accès aux briques élémentaires des modèles relationnels. Le tableau 4.5 synthétise cette correspondance.

Si nous avons jusqu'à présent défini le *quoi* du contrôle d'accès, nous n'avons pas proposé de *comment*. Notre modélisation a séparé les modèles des politiques, le chapitre suivant va renforcer cette séparation. Il propose des outils relatifs à :

- la conception des modèles de contrôle d'accès, c'est-à-dire la manipulation de la structure et des principes de l'organisation des droits d'un point de vue *générique*, quelles que soient les politiques qui en sont instances,
- la conception des politiques, c'est-à-dire la définition progressive des données qui composent la politique. Cette conception peut se faire à partir de droits existants dans le cas d'évolution du contrôle d'accès,

Modèle de contrôle d'accès	Structuration relationnelle
noyau	schéma
principes	règles de déduction
propriétés	dépendances
politique factuelle	instance
politique déduite	point fixe de la déduction

TAB. 4.5 – Correspondance entre contrôle d'accès et structuration relationnelle

– l'administration des politiques, c'est-à-dire la manipulation des données instances d'un modèle de contrôle d'accès donné.

When the only tool you have is a hammer, everything begins to look like a nail.
Anonymous

5
Mise en œuvre de la structuration

▷ *Dans le chapitre précédent, nous avons développé une vision abstraite de l'organisation des droits dans les systèmes. Dans ce chapitre, nous proposons une vision pragmatique, orientée selon l'utilisation qui est faite du cadre logique proposé pour le contrôle d'accès. Les contributions que nous présentons relèvent de trois ordres :*

- *nous proposons une structuration en niveaux d'abstraction pour situer et définir les activités de conception et d'administration du contrôle d'accès,*
- *nous proposons des outils permettant d'assister ces deux activités. Il s'agit d'exploiter les possibilités offertes par les procédures de preuve et de satisfaction pour les dépendances pour :*
  - *la vérification et la simplification de modèles de contrôle d'accès dans notre cadre logique,*
  - *la vérification que les politiques satisfont bien aux propriétés des modèles,*
- *nous abordons la problématique du cycle de vie et de l'évolution des droits avec :*
  - *une représentation graphique des principes et des propriétés des modèles de contrôle d'accès, ainsi que la visualisation l'exécution des procédures de preuve,*
  - *une automatisation de la découverte de concepts intermédiaires hiérarchisés présents implicitement dans des politiques existantes.*

Ce chapitre va justifier a posteriori les choix des termes de concepts et de relations que nous avons utilisés pour la définition générique des modèles de contrôle d'accès. De plus, nous mettons en lumière et exploitons les liens qui existent entre plusieurs domaines de recherche : les dépendances de données, les graphes conceptuels et l'analyse de concepts formels.

L'implémentation des procédures que nous utilisons dans ce chapitre a été réalisée et fait l'objet de la section 6.2 du chapitre suivant. En définissant les activités de conception et d'administration de contrôle d'accès nous voyons apparaître le problème de l'organisation des droits des administrateurs qui fait l'objet d'une des perspectives présentées dans le chapitre 7.

Une trace d'inférence de la simplification exposée en section 5.2.5 est proposée en annexe C. ◁

Plan du chapitre

Note sur les dépendances

POUR les vérifications de propriétés que nous proposons dans ce chapitre, nous considérons les règles de déduction DATALOG comme des dépendances génératrices de tuples totales. Les règles DATALOG^C sont des restrictions des dépendances génératrices de tuples contraintes, sans variables quantifiées existentiellement et sans contraintes en conclusion.

La structuration relationnelle d'un modèle de contrôle d'accès $\mathbf{AC} = (sch, P, \Sigma)$ est composée d'un schéma sch, d'un ensemble de règles de déduction P qui permet de dériver les relations en intention et d'un ensemble de dépendances Σ qui permettent de définir l'intégrité des relations.

L'ensemble de formules closes P est considéré comme un ensemble de dépendances génératrices de tuples totales. L'instance $\mathbf{I}'$ est définie par P, elle satisfait par définition les règles de P prises comme des dépendances. Nous avons donné les éléments qui nous autorisent à faire ces considérations en section 4.1.3.

Pour les vérifications de *l'implication logique* d'un expression σ, c'est-à-dire des preuves de $P \cup \Sigma \models \sigma$ ou de $P \cup \Sigma \not\models \sigma$, nous utiliserons uniformément les procédures de preuves des dépendances, sans faire de distinction entre les principes et les propriétés des modèles. Pour les vérifications de la *satisfaction* d'une propriété par une instance, c'est-à-dire des preuves d'implication ou de non implication $\mathbf{I} \models \sigma$ ou de $\mathbf{I}' \models \sigma$, nous considérons également les règles DATALOG et DATALOG^C comme des dépendances.

Selon le fragment logique utilisé pour la modélisation du contrôle d'accès, différentes procédures de preuves peuvent être utilisées. Les procédures de [Maher96] ou de [Wang02] peuvent terminer pas un cas spécifique à la présence de contraintes en conclusion des formules. Cette terminaison par vacuité est le cas où une formule insatisfaisable $\perp$ est contenue dans le magasin de contraintes. Pour les applications que nous proposons, ce cas de terminaison indique la présence d'une erreur manifeste dans la théorie logique.

La définition d'un modèle de contrôle d'accès que nous proposons est un ensemble de *formules closes* de la logique du premier ordre, c'est-à-dire une théorie logique du premier ordre. Le vocabulaire sur lequel est définie cette théorie est composé des noms de relations en intention et en extension, définies par les concepteurs du modèle. L'objectif de la théorie et de formaliser les notions utiles à la définition d'un modèle de contrôle d'accès. La différence d'utilisation entre principes et propriétés d'intégrité des modèles de contrôle d'accès est importante. En revanche, du point de vue de la sémantique logique, elle est moindre. La théorie que nous considérons $T = P \cup \Sigma$ est composée de deux sous-ensembles de formules : les principes P et les propriétés Σ.

5.1 Structuration en couches

Depuis l'identification d'un cadre logique d'intérêt pour la modélisation du contrôle d'accès, nous avons étendu les propositions existantes de modélisation basées principalement sur DATALOG en intégrant des dépendances de plus en plus expressives. Dans notre structuration du contrôle d'accès, nous avons séparé :

- les relations de *base* des modèles,
- les relations *dérivées* des relations de base,
- les *propriétés* que les relations doivent respecter.

Cette séparation est un fil conducteur la thèse. Elle est encouragée par plusieurs propositions du domaine [Halpern03, Li07]. Nous avons utilisé le cadre logique relationnel des dépendances pour définir une modélisation générique, qui intègre les propriétés définies comme fondamentales des modèles de contrôle d'accès et les suggestions d'amélioration proposées pour les modèles RBAC.

5.1.1 Objectifs de la structuration

Jusqu'à présent nous avons principalement spécifié formellement ce que *sont* les modèles de contrôle d'accès, en généralisant les notions et les éléments de modélisation logique existants. Nous allons désormais définir comment mettre en œuvre le contrôle d'accès et exploiter les outils développés pour les dépendances. Il nous faut donc situer les différentes activités qui surviennent lors de l'utilisation des modèles et des politiques de contrôle d'accès.

5.1.2 Décomposition en couches

Afin de situer la conception de modèles de contrôle d'accès, nous mettons en parallèle le cycle de développement d'applications orientées objet et celui du développement du contrôle d'accès. L'organisation des logiciels est divisée en couches allant de l'abstraction la plus générale à la réalisation concrète. Pour les applications orientées objet, ces niveaux vont du métamodèle d'UML jusqu'au code compilé (tableau 5.1).

Cette stratification encourage la séparation entre les différents niveaux d'abstraction qui interviennent dans la conception d'applications. Cette section transpose cette stratification en niveaux au contrôle d'accès. L'objectif est de pouvoir identifier et structurer les différents activités qui surviennent lors de la mise en œuvre de notre modèle logique générique d'organisation des droits. Le tableau 5.1 met en parallèle le développement d'applications orientées objet et la stratification en niveaux d'abstraction du contrôle d'accès :

	Développement logiciel	Contrôle d'accès	
3	meta-modèle UML, qui définit la syntaxe graphique	définition générique des modèles de contrôle d'accès	**3**
2	conception d'application avec des diagrammes UML	conception de modèle de contrôle d'accès	**2**
1	développement de l'application	définition de la politique	**1**
0	déploiement de l'application	déploiement du contrôle d'accès	**0**

TAB. 5.1 – Abstraction en couches pour le contrôle d'accès

- le niveau le plus haut (numéroté 3 du tableau 5.1) est le modèle générique présenté dans le chapitre 4. Passer de ce niveau au niveau inférieur est l'étape que nous désignons par la *conception de modèle*,
- le niveau inférieur (2 du tableau 5.1) est une instance du modèle générique, c'est-à-dire un modèle de contrôle d'accès donné, exprimé dans le cadre que nous proposons. Un tel modèle peut être RBAC, ORBAC, celui proposé par le GMSIH pour les systèmes d'information hospitaliers ou tout modèle conçu sur mesure. Le passage au niveau inférieur est l'étape de *conception de politique*,
- le niveau suivant (1 du tableau 5.1) est la politique de contrôle d'accès, c'est-à-dire un ensemble de faits déterminé par les administrateurs, organisé selon les concepts et les relations définis dans le modèle de contrôle d'accès,
- le niveau le plus bas est celui de la réalisation effective du contrôle d'accès, c'est-à-dire la mise en production du modèle et de la politique. C'est pendant et après le déploiement du contrôle d'accès qu'a lieu l'*administration des politiques*.

5.2 Conception de modèles

5.2.1 Définition

Nous avons défini le niveau le plus haut de la structuration abstraite du contrôle d'accès , c'est le modèle générique du chapitre précédent. Nous définissons l'opération d'instanciation du modèle générique en un modèle donné, c'est-à-dire la conception.

Définition (Conception de modèle de contrôle d'accès)**.** *La* conception d'un modèle de contrôle d'accès *est l'activité qui consiste en l'identification et la définition :*

- *des* concepts et relations *utilisés pour structurer les droits qui forment le noyau,*
- *des* principes *qui régissent la dérivation des relations déduites à partir du noyau,*
- *des* propriétés *que les politiques doivent respecter.*

Cette définition identifie trois sous-activités principales de la conception de modèles de contrôle d'accès L'étape de conception de modèle est fondamentale pour que

le modèle construit réponde bien au cahier des charges de l'organisation. Comme il est difficile dans le cas général de proposer un modèle consensuel d'organisation des droits, nous proposons un modèle qui permette de concevoir des modèles de contrôle d'accès sur mesure.

La démarche d'analyse, de spécification et de conception menée par Anas Abou El Kalam dans sa thèse a conduit à la définition du modèle ORBAC [Abou El Kalam03]. L'objectif était de concevoir une politique de sécurité pour les systèmes d'information et de communication en santé et social.

Dans cette perspective, l'état de l'art et le modèle générique du chapitre 4 peuvent être abordés comme une bibliothèque référençant les principaux modèles de contrôle d'accès existants et les notions qu'ils intègrent. Grâce à cette bibliothèque, les concepteurs disposent des principaux concepts et relations pour construire de nouveaux modèles de contrôle d'accès, en utilisant les dépendances comme cadre logique sous-jacent.

5.2.2 Méthodologie de conception

La conception d'un modèle de contrôle d'accès couvre un domaine très large, qui va de la catégorisation des ressources du système en classes de confidentialité à la formalisation d'aspects sémantiques. La figure 5.1 identifie quatre étapes qui interviennent lors de la conception de contrôle d'accès :

1. l'analyse des besoins de sécurité de l'organisation. Plusieurs méthodologies ont été proposées pour aider les responsables de la sécurité à identifier les points stratégiques des organisations, à évaluer les risques ou à classer les ressources selon leur valeur et leur confidentialité [Ferraiolo03b, Solomon05, Benantar06, Deswarte03]. Cette étape aboutit à l'établissement d'une politique générale de sécurité qui définit les objectifs et les moyens à mettre en œuvre pour garantir les objectifs de sécurité,
2. l'analyse de l'existant et l'identification des concepts et relations à intégrer dans le contrôle d'accès. Cette étape définit selon quels principes les droits des utilisateurs vont être organisés, quelles règles vont régir comment un utilisateur va acquérir des privilèges, et surtout quelles sont les propriétés attendues que les droits vont devoir respecter,
3. la définition formelle du modèle de contrôle d'accès. Cette étape définit de manière univoque la structure, les principes et les propriétés exprimées en langue naturelle lors de l'étape précédente. Le précédent et le présent chapitre interviennent à ce niveau. Nous proposons un cadre formel et un panel d'outils pour les concepteurs grâce auxquels ils peuvent définir et vérifier leur modèle de contrôle d'accès,

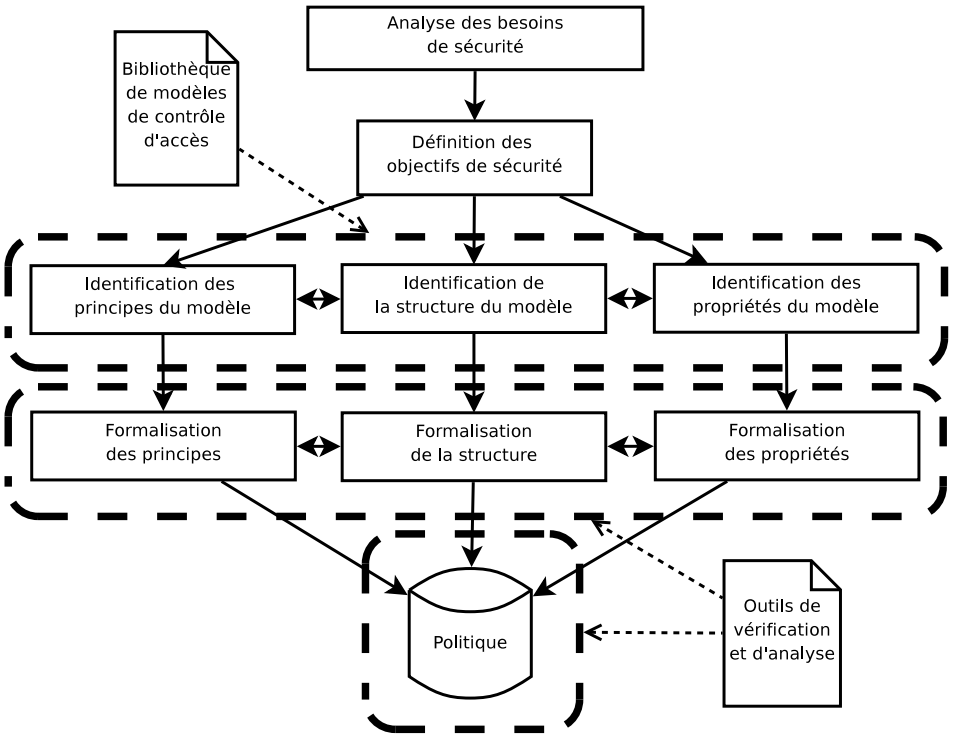


FIG. 5.1 – Étapes de la conception d'un modèle

4. la définition de la politique. Dans cette dernière étape interviennent les administrateurs qui vont établir les droits d'accès par l'intermédiaire des concepts et relations définis par les concepteurs.

Lors de la conception d'un modèle de contrôle d'accès, les concepteurs peuvent définir de nombreux concepts, relations, principes et propriétés. De plus, ces définitions peuvent éventuellement se contredire ou être redondantes. Des outils de vérification et d'analyse des modèles et des politiques permettent de répondre à ces problèmes.

5.2.3 Primitives de conception

L'étape de conception de modèle de contrôle d'accès consiste en la définition logique du schéma, des principes et des propriétés du contrôle d'accès. Il s'agit donc d'une activité où l'on utilise les notions présentées dans le chapitre précédent pour modéliser l'organisation des droits dans un système.

Définition (Primitives de conception)**.** *Une* primitive de conception *est toute opération qui permette la création, la modification ou la suppression des concepts, relations, principes et propriétés d'un modèle de contrôle d'accès.*

		Objectif de modélisation
Principes	dérivation	triplets d'autorisation triplets d'autorisation orientés utilisateurs
	fermeture	symétrie d'une relation transitivité d'une relation réflexivité d'une relation
		relations en intention techniques
Propriétés	propriété	antisymétrie d'une relation irréflexivité d'une relation structurelle du noyau
	limitation	hiérarchie en arbre hiérarchie en arbre inverse hiérarchie en treillis
	contrainte	prérequis exclusion mutuelle interaction hiérarchie/exclusion spécialisation de hiérarchie
		dépendances techniques

TAB. 5.2 – Classification de la théorie du premier ordre

L'aide d'outils graphiques permet de manipuler plus facilement la formalisation logique que nous proposons dans la thèse. Dans ce sens, nous proposons en section 5.4 une représentation des formules logiques qui permet d'exprimer graphiquement les modèles de contrôle d'accès.

5.2.4 Intégrité des modèles

Le tableau 5.2 présente des objectifs de modélisation respectifs aux deux sous-ensembles de la théorie. Des principes et des propriétés techniques peuvent être introduits, pour servir d'intermédiaires dans la modélisation logique. Dans les modèles RBAC on définit généralement deux relations en intention intermédiaires pour représenter logiquement les applications *auth_perms* et *auth_users* définies à partir des relations hiérarchiques entre rôles (section 2.3.2). Il faut s'assurer que les différentes expressions de P et Σ soient cohérentes.

5.2.4.1 Définition

En présence d'une théorie logique de taille conséquente,les règles et les dépendances peuvent s'avérer contradictoires. Ceci est particulièrement vraisemblable quand plusieurs concepteurs interviennent. L'étape préalable à la vérification des modèles est la définition de l'intégrité d'un modèle de contrôle d'accès.

Définition (Modèle non intègre de contrôle d'accès)**.** *Un modèle de contrôle d'accès* $\mathbf{AC} = (edb \cup idb, P, \Sigma)$ *de théorie* $T = P \cup \Sigma$ *est dit* non intègre *s'il n'existe pas de politique* $\mathbf{I}'$ *modèle de T où chaque relation de edb est instanciée.*

S'assurer de l'intégrité d'un modèle de contrôle d'accès est une vérification de l'existence d'un modèle particulier de la théorie logique. Cette vérification doit être effectuée quelle que soit la politique déterminée par les administrateurs. C'est donc une preuve qui fait abstraction des politiques et qui ne concerne que le modèle de contrôle d'accès lui-même. La définition présentée est une spécialisation de la notion de satisfaisabilité d'une théorie logique (annexe A), à laquelle nous avons imposé une contrainte supplémentaire, instanciation de chaque relation en extension.

5.2.4.2 Principe de la vérification

Nous souhaitons utiliser les procédures de preuves comme le *chase* pour automatiser une partie de la vérification de l'intégrité d'un modèle. La vérification est basée sur la construction d'une dépendance σ puis sur la vérification à l'aide d'une procédure de preuve qu'il est impossible de dériver σ à partir de la théorie, soit $P \cup \Sigma \not\models \sigma$.

L'hypothèse ϕ de σ est construite à partir du schéma en extension *edb* composé des concepts $\{C_1 \dots C_n\}$ et des relations $\{R_1, \dots, R_m\}$. ϕ exprime que chaque relation de *edb* est instanciée. La conjonction ϕ est définie comme une conjonction où toute relation et tout concept de *edb* apparaisse, soit une expression de la forme :

$$C_1(ID_1), \dots, C_n(ID_n), R_1(\dots ID_i \dots) \dots, R_m(\dots ID_j \dots)$$

La conclusion de σ est une antilogie $\perp$. Il s'agit donc d'une forme de preuve par l'absurde : si l'on arrive à dériver σ, on dérive une antilogie, c'est donc que la théorie logique n'est pas correcte pour une instance $\mathbf{I}$ quelconque.

Certaines procédures de preuves n'étant que semi-décidables, une preuve négative peut ne pas être fournie. Dans la pratique, si une procédure n'est pas parvenue à une réponse en un nombre déterminé d'étapes, nous considérons par principe de précaution que le modèle n'est pas intègre.

Nous avons défini la *non-intégrité* d'un modèle de manière générale. Des éléments pour une définition plus précise de l'*intégrité* et d'une dépendance σ qui la modélise sont proposés comme discussion et perspective en section 7.3.4. Il s'agit d'un problème difficile qui dépasse nos compétences en informatique théorique.

La définition générale de l'intégrité et la construction d'une dépendance σ qui la représenterait nécessitent d'être poursuivies. Cependant, la définition actuelle nous permet d'effectuer une vérification intéressante : la vérification de l'intégrité d'un modèle RBAC$_3$ comportant la définition d'un rôle particulier `root`.

5.2.4.3 Application de la vérification

Supposons la conception d'un modèle de type RBAC$_3$, comportant le concept de rôle $Role(R)$, une relation d'héritage entre rôles $Hérite(R_1, R_2)$ et une exclusion mutuelle entre rôles $Exclusion(R_1, R_2)$. De multiples concepteurs ont défini successivement dans la même théorie $P \cup \Sigma$ trois expressions logiques :

1. σ_1 la dépendance $Exclusion(R_1, R_2), Hérite(R, R_1), Hérite(R, R_2) \rightarrow \bot$ imposée par la présence d'une relation d'exclusion et d'une hiérarchie sur les rôles,
2. σ_2 la dependance structurelle $Exclusion(R_1, R_2) \rightarrow Role(R_1), Role(R_2)$,
3. τ la règle de déduction $Role(R) \rightarrow Hérite(\texttt{root}, R)$ qui modélise l'existence d'un rôle particulier `root` qui hérite de tous les autres.

Considérons une dépendance qui permette de vérifier l'intégrité de la modélisation. Cette dépendance σ est de la forme $Role(R), \ldots, Exclusion(R_1, R_2) \rightarrow \bot$: nous allons vérifier la non-contradiction de la théorie. La présence de deux rôles exclusivement mutuels dans la politique va rendre la politique inconsistante. Un tel modèle de contrôle d'accès n'est pas intègre à cause de la définition du rôle `root`. Cette vérification est automatisable avec une procédure de preuve. Les étapes de la dérivation sont successivement :

1. on suppose l'existence de deux tuples $Role(\texttt{r})$ et $Exclusion(\texttt{r}_1, \texttt{r}_2)$,
2. par application de σ_2 il existe deux rôles $Role(\texttt{r}_1)$ et $Role(\texttt{r}_2)$,
3. en appliquant deux fois τ on déduit $Hérite(\texttt{root}, \texttt{r}_1)$ et $Hérite(\texttt{root}, \texttt{r}_2)$,
4. on peut donc appliquer σ_1 et dériver $\bot$.

On a ainsi démontré que $P \cup \Sigma \models \sigma$ et la non intégrité de la modélisation. Plusieurs corrections sont envisageables. On peut modifier la règle τ pour définir le rôle `root` autrement ou supprimer cette règle. On peut également modifier les propriétés de l'exclusion mutuelle ou éventuellement y renoncer. L'analyse de la trace de la preuve peut donner des informations aux concepteurs pour choisir la meilleure correction. On remarque que c'est la définition conjointe de σ_1 et de τ qui est une des sources du problème.

Cette démonstration automatisée complète les résultats [Gavrila98]. Cette vérification de l'intégrité d'un modèle RBAC$_3$ a été proposée dans [Benantar06] comme discussion sur l'impact de l'exclusion d'où est extraite la citation suivante :

Exemple 5.1 Un modèle expoitable pour le CHM

Dans la quête d'un modèle de contrôle d'accès qui leur conviennent, les informaticiens du CHM ont interrogé plusieurs des responsables administratifs pour comprendre leurs besoins de modélisation des droits.

Il a été admis que les services de l'établissement sont hiérarchisés, mais aussi que certains rôles ne peuvent pas être endossés dans des services différents. Il est clairement spécifié que l'on ne peut pas cumuler les fonctions de responsable dans plusieurs services. Les concepteurs du contrôle d'accès tâchent ainsi d'exprimer une forme d'exclusion mutuelle.

La vérification de l'intégrité d'un modèle permet aux concepteurs de s'assurer que leur modélisation de l'exclusion de rôles entre services est correcte, et qu'une fois en production, ils pourront effectivement exploiter la relation d'exclusion qu'ils ont définie.

The implication of the above property is that a role graph can have a "root" role (i.e., a role that inherits from every other role) only when no pair of roles in the entire role hierarchy is in any separation of duty relation.

5.2.5 Simplification de modèles

Quand plusieurs concepteurs interviennent lors de la définition d'un modèle de contrôle d'accès, on peut aboutir à une théorie logique de taille conséquente. Nous avons vu que les expressions logiques peuvent concerner des relations communes à un même concept et interagir entre elles, jusqu'à rendre le modèle non intègre. Cependant, les formules de la théorie peuvent être seulement redondantes et donc inutiles. Réduire le nombre de règles de la théorie logique d'un modèle de contrôle d'accès permet de le simplifier. Ceci facilite la compréhension du modèle et les tâches ultérieures d'administration des politiques.

Nous avons montré pour les modèles RBAC_3 que certaines définitions de l'exclusion mutuelles en impliquent d'autre en section 3.4.6, et que le triplet d'autorisation orienté utilisateurs dynamique est inclus dans le triplet statique. Les procédures de preuves pour les dépendances permettent de simplifier les théories en éliminant les dépendances ou les règles redondantes.

Définition (Redondance dans un modèle de contrôle d'accès)**.** *Soit T la théorie logique d'un modèle de contrôle d'accès. Soit $\sigma \in T$, si $T\backslash\{\sigma\} \models \sigma$ alors la formule logique σ est dite* redondante*. De plus, $T\backslash\{\sigma\}$ et T ont les mêmes modèles.*

La sémantique logique d'une théorie dont ont été éliminées les règles redondantes reste inchangée par le procédé d'élimination. C'est-à-dire que l'ensemble des politiques autorisées, instances valides du modèle, reste le même. L'approche que nous proposons permet une simplification intéressante des propriétés d'intégrité des politiques RBAC_3. Ce résultat a été prouvé manuellement par les auteurs de [Gavrila98].

Les propriétés identifiées concernent seulement les modèles à rôles du type RBAC_3. De plus, elles ne prennent pas en compte certaines extensions qui ont depuis été largement adoptées, comme la hiérarchisation des objets. La thèse vise un objectif plus large de conception de modèles de contrôle d'accès, pour lequel il est profitable de disposer de résultats similaires à ceux obtenus dans [Gavrila98]. Dans la mesure du possible nous essayons de dériver de tels résultats automatiquement à l'aide de procédures de preuve.

Exemple 5.2 Automatisation de la simplification de modèle

Les auteurs de [Gavrila98] ont établi un ensemble de propriétés qui doivent être vérifiées par toute politique RBAC_3. Cet ensemble inclut une exclusion mutuelle statique orientée utilisateur (*Static Separation of Duties* – SSD – prédicat $Exclusion$) et la hiérarchisation des rôles (prédicat $Hérite$). Un extrait de ces propriétés formulées en logique et traduites dans notre formalisation est donné par le tableau suivant :

	Description
σ_1	*any two roles assigned for a same user are not in separation of duties* $Habilite(User, Role_1), Habilite(User, Role_2), Exclusion(Role_1, Role_2) \rightarrow \bot$
σ_2	*no role is mutually exclusive with itself* $Exclusion(Role, Role) \rightarrow \bot$
σ_3	*mutual exclusion is symetric* $Exclusion(Role_1, Role_2) \rightarrow Exclusion(Role_2, Role_1)$
σ_4	*any two roles in ssd do not inherits one another* $Hérite(Role_1, Role_2), Exclusion(Role_1, Role_2) \rightarrow \bot$
σ_5	*there is no role inheriting two roles in ssd* $Exclusion(Role_1, Role_2), Hérite(Senior, Role_1), Hérite(Senior, Role_2) \rightarrow \bot$
σ_6	*If a role inherits another role and that role is in SSD with a third one, then the inheriting role is in SSD with the third one.* $Hérite(Senior, Role_1), Exclusion(Role_1, Role_2) \rightarrow Exclusion(Senior, Role_2)$.

Soit $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6\}$, les exécutions des procédures comme le *chase* permettent de prouver que les propriétés σ_4 et σ_5 sont redondantes et qu'on peut donc réduire la taille de la théorie :

– $\Sigma \backslash \{\sigma_4\} \models \sigma_4$, par application de σ_3, σ_6 et σ_2,
– $\Sigma \backslash \{\sigma_5\} \models \sigma_5$, par application de σ_3, σ_6, σ_3, σ_6 et σ_2
– et même $\Sigma \backslash \{\sigma_4, \sigma_5\} \models \sigma_4, \sigma_5$.

Une trace commentée de la seconde preuve obtenue avec notre prototype LIBDEPENDENCIES est proposée en annexe C.

5.3 Administration des politiques

Nous avons fait figurer dans l'illustration 5.1 des « outils de vérification et d'analyse » des politiques et des modèles. Dans la section précédente, nous avons répondu à des problèmes de vérification et de simplification survenant lors de la *conception* de modèles de contrôle d'accès, nous allons désormais nous attacher aux problèmes qui surviennent lors de l'*administration* des politiques, une fois le modèle conçu. Vérifier que les politiques sont intègres est une fonctionnalité importante d'un système de contrôle d'accès, qui permet d'éliminer des erreurs d'administration.

5.3.1 Conception des politiques

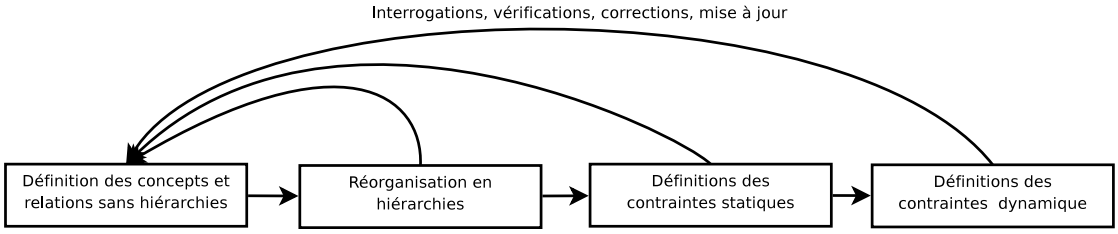


FIG. 5.2 – Etapes de la conception d'une politique de contrôle d'accès

Nous avons décrit les étapes de la conception d'un modèle de contrôle d'accès, mais une fois ce modèle déterminé et validé, il reste à définir la politique. En présence de hiérarchies multiples, de plusieurs types de contraintes ou encore de concepts intermédiaires multiples entre utilisateurs et permissions, l'établissement d'une politique doit suivre une méthodologie pour être mené à bien. La méthodologie de conception proposée par [Ferraiolo03b] est illustrée par la figure 5.2 :

1. la définition des concepts principaux du modèle, sans hiérarchisation,
2. la réorganisation des concepts en hiérarchie,
3. la définition des contraintes statiques,
4. la définition des contraintes dynamiques.

Lors de la conception d'une politique instance d'un modèle de contrôle d'accès, les administrateurs définissent des concepts et des relations en extension du modèle. Dans les quatre principales étapes de la conception, les administrateurs ont besoin d'outils permettant :

- de *manipuler* la politique : saisir, modifier et supprimer des données,
- d'*interroger* la politique définie jusqu'à présent,
- de *vérifier* que la politique respecte bien les propriétés du modèle.

5.3.2 Primitives d'administration

Dans un premier temps, nous définissons ce que sont les primitives d'administration : des opérations sur les politiques qui sont nécessaires pour l'administration du contrôle d'accès. Des primitives ont été définies pour les modèles à rôles dans le standard RBAC [Ferraiolo03b]. Elles définissent les fonctionnalités – minimales – nécessaires aux tâches d'administration.

En considérant les noyaux des modèles de contrôle d'accès comme des schémas relationnels et les politiques comme des instances de ces schémas, les langages de manipulation de données d'interrogation des systèmes de gestion de bases de données fournissent certaines de ces primitives.

Définition (Primitive d'administration)**.** *Une* primitive d'administration *est toute opération sur une polique instance d'un modèle de contrôle d'accès qui permet soit :*

- *la* modification *de la politique en extension, c'est une primitive de* manipulation*.*
- *l'*interrogation *de la politique en intention, c'est une primitive de* revue*.*

5.3.2.1 Manipulation de politiques

Cette première catégorie de primitives recouvre celles offertes par les langages de manipulation de données dans les systèmes de gestion de base de données. C'est grâce à ces fonctions que les administrateurs peuvent manipuler la politique en extension **I**.

Définition (Primitive de manipulation)**.** *Une* opération *(ou* primitive*)* administrative *sur une politique de contrôle d'accès est une opération permettant l'ajout, la modification ou la suppression de tuples dans une instance d'un modèle de contrôle d'accès.*

Dans le cadre que nous proposons, les opérations administratives recouvrent les opérations `insert` (ajout), `update` (modification) et `delete` (suppression) classiques des systèmes de gestion de bases de données. Elles permettent de manipuler **I**.

Ces opérations permettent de manipuler *uniquement* **I** et pas la partie déduite $\mathbf{I}'\backslash\mathbf{I}$. Les dépendances du non-court-circuit proposées en section 4.1.3, permettent d'interdire explicitement des manipulations de $\mathbf{I}'\backslash\mathbf{I}$ qui ajouteraient des droits. Ces dépendances contribuent à garantir le principe d'administration des modèles structurés.

Définition (Principe d'administration des modèles structurés)**.** *Les administrateurs ne peuvent définir que des relations en extension du schéma edb d'un modèle de contrôle d'accès* **AC***. Il ne peuvent pas manipuler l'ensemble en intention* $\mathbf{I}'\backslash\mathbf{I}$*.*

5.3.2.2 Revue de politiques

Une des fonctionnalités attendues d'un système de gestion de politiques de contrôle d'accès est de permettre l'interrogation des politiques. Cette fonctionnalité est appelée la *revue de politique*. Il s'agit d'une des primitives fonctionnelles définies dans les standards d'implantation des modèles RBAC. Dans le cadre proposé, nous définissons la notion de revue en étendant celle de [Barker03] :

Définition (Revue d'une relation de politique de contrôle d'accès)**.** *Une* revue *d'une relation R de politique de contrôle d'accès est l'évaluation d'une requête conjonctive, éventuellement récursive, sur* $\mathbf{I}'$ *faisant intervenir la relation R dans son corps* ψ *:*

$$\{e_1, \ldots, e_n \mid \psi\}$$

Pour l'évaluation des requêtes récursives, nous pouvons utiliser les algorithmes proposés pour l'évaluation des programmes DATALOG [Abiteboul95]. La formulation proposée permet de définir sans ambiguïté la notion de revue. Par exemple, quand d'aucuns traitent de revue des *affectations* de rôles aux utilisateurs dans les modèles RBAC, nous définissons qu'ils entendent l'ensemble des requêtes exprimables sur la relation *Habilite*.

De plus, nous définissons la revue de façon générique, sans avoir à spécifier concept par concept et relation par relation l'ensemble des opérations définies sur le modèle, comme cela a été fait dans le standard RBAC [Ferraiolo03b, Ferraiolo01]. En effet, si l'approche exhaustive du standard est applicable sur le modèle RBAC_3 qui comporte cinq concepts, cinq relations principales, une relation de domination et deux relations d'exclusion, elle l'est déjà moins sur des modèles comme ORBAC.

5.3.3 Intégrité des politiques

Des primitives nécessaires à la conduite des activités d'administration ont été définies pour les modèles RBAC. Nous proposons maintenant de vérifier que les opérations effectuées par les administrateurs ne violent pas les propriétés que les politiques doivent respecter. Ces propriétés sont définies dans le modèle de contrôle d'accès par Σ. Nous définissons la condition à laquelle une politique est intègre vis-à-vis de ces propriétés.

Définition (Politique intègre de contrôle d'accès)**.** *Soit* $\mathbf{I}$ *une politique instance du schéma edb d'un modèle de contrôle d'accès* **AC***. Soit P un ensemble de règles définissant les relations en intention idb,* $\mathbf{I}'$ *le plus petit modèle de P contenant* $\mathbf{I}$ *et* Σ *un ensemble de propriétés du modèle de contrôle d'accès. La politique de contrôle d'accès* $\mathbf{I}$ *est* intègre *si et seulement si* $\mathbf{I}'$ *est un modèle de* Σ*.*

La définition de l'intégrité d'une politique que nous proposons se base sur la notion de modèle de la théorie du premier ordre qui définit un modèle de contrôle d'accès et la satisfaction de l'ensemble de dépendances Σ par ce modèle $\mathbf{I}' \models \Sigma$.

Dans le chapitre 3, nous avons donné le schéma *edb* du modèle RBAC_0 avec la figure 3.1 ainsi qu'une politique jouet $\mathbf{I}$ définie sur ce schéma avec le tableau 3.1. Nous avons également donné les règles de déductions P des triplets d'autorisations avec le tableau 3.2 ainsi que l'ensemble des dépendances Σ que les politiques doivent respecter avec le tableau 3.9.

Nous disposons ainsi d'une formalisation relativement complète du modèle RBAC_0. On peut calculer $\mathbf{I}'$ connaissant P et $\mathbf{I}$. On peut utiliser pour cela un algorithme de calcul du point fixe d'un programme DATALOG. Ensuite, on peut mettre en œuvre le *chase* pour vérifier que $\mathbf{I}' \models \Sigma$.

On peut, comme cela est fait dans les programmes PROLOG, construire un ensemble de dépendances Γ où l'hypothèse de chaque dépendance est une tautologie et la conlusion un tuple de $\mathbf{I}'$. Ensuite, pour chaque $\sigma \in \Sigma$, on lance le *chase* pour vérifier que $\Gamma \models \sigma$. L'étape de construction de $\mathbf{I}'$ et le *chase* peuvent être combinés dans une seule et même procédure de vérification de la satisfaction. C'est l'exploitation des fonctionnalités logicielles communes entre les différentes procédures proposées sur le modèle relationnel qui nous a conduit à implémenter la LIBDEPENDENCIES que nous décrivons en section 6.2.

Selon la dépendance non satisfaite, les administrateurs devront corriger la politique, ou éventuellement demander aux concepteurs du modèle de revoir leurs définitions des propriétés ou des principes. Selon le type de dépendance non satisfaite, on peut apporter différentes corrections.

5.3.4 Correction des politiques

Le tableau 5.3 propose un ensemble de corrections types, réalisables grâce aux primitives de manipulation, selon le type de propriétés du modèle de contrôle d'accès que la politique ne respecte pas. Selon le type de propriété, différentes classes de dépendances peuvent être utilisées pour la représenter. La sémantique des dépendances génératrices de tuples impose la *présence* de tuples, alors que les autres *limitent* les valeurs possibles. L'intégrité d'une politique peut donc être divisée en deux sous-propriétés selon qu'on s'intéresse à la satisfaction des dépendances génératrices [Graham86] :

- de tuples, c'est la *complétude*[1] d'une politique,
- de nullité ou de contraintes, c'est la *consistance*[2] d'une politique.

Pour réparer une politique incomplète ou inconsistante, les administrateurs doivent analyser la source de non intégrité. Pour cela, ils doivent suivre les étapes de l'algo-

[1] Le nom *complétude* ne désigne pas ici l'abscence de propositions indécidables.
[2] Le nom *consistance* ne désigne pas ici l'ascence de propositions contradictoires.

rithme qui a déterminé que la politique n'est pas intègre, puis modifier la politique en conséquence. Ce travail d'analyse et de débogage peut être compliqué par la présence de réécritures des formules de la théorie et l'absence de trace de l'exécution.

En utilisant des algorithmes qui manipulent les dépendances sans les modifier, chaque étape de l'inférence possède une signification exprimable en termes de concepts et de relations du modèle de contrôle d'accès. Chaque nouvelle dérivation dans l'inférence correspond à l'application d'une formule de la théorie du modèle de contrôle d'accès. Les procédures dédiées aux dépendances ne les transforment pas. On évite ainsi des transformations techniques de la théorie par les algorithmes d'inférence :

- l'introduction de fonctions de Skolem lors de la suppression des quantificateurs existentiels, qui introduisent des symboles arbitraires supplémentaires,
- la décomposition des dépendances à têtes multiples en clauses de Horn, qui ne comportent qu'un seul atome en conclusion. Des sous-formules techniques ne sont pas introduites et le nombre de formules de la théorie n'est pas augmenté,
- la production de formules intermédiaires dans l'inférence, comme les résolvantes avec la SLD-Resolution (*Selective Linear Definite clause resolution*) de PROLOG.

Deux grandes classes de vérification de l'intégrité sont envisagées, selon l'impact en termes de coût de calcul lors de l'exécution du moniteur de contrôle d'accès :

- la vérification *statique* des politiques, qui concerne la partie de la politique invariante par exécution du moniteur. Cette vérification est à effectuer *avant* la validation d'une opération administrative,
- la vérification *dynamique* des politiques, qui concerne la partie de la politique variante lors de l'exécution du moniteur. Cette vérification est à effectuer *avant* la validation d'une opération sur la politique demandée à l'initiative d'un acteur autre qu'un administrateur.

	Dépendance	**Type**	**Correction possible**
propriété	antisymétrie	EGD	*Suppression* de cycles
	irréflexivité	NGD	*Suppression* d'une arête
limitation	hiérarchie en arbre	EGD	*Suppression* de pères
	hiérarchie en arbre inverse	EGD	*Suppression* de fils
	hiérarchie en treillis	TTGD	*Ajout* de relations
contraintes	prérequis	CTGD	*Ajout* du prérequis
	exclusion	NGD	*Suppression* d'affectations
	hiérarchie/exclusion	NGD	*Suppression* d'affectations
	spécialisation de hiérarchie	CTGD	*Suppression* d'affectations

TAB. 5.3 – Corrections possibles d'une politique non intègre

5.3.5 Comparaison statique de politiques

Dans le chapitre 4, nous avons montré que les modèles de contrôle d'accès partagent de nombreux traits communs. Une des caractéristiques partagées est que les modèles doivent permettre de dériver les triplets d'autorisation *Accès*, *Statique* et *Dynamique*.

Dans le cas général, le caractère variable des triplets dynamiques interdit de pouvoir raisonnablement envisager l'exécution d'algorithmes coûteux lors de l'exécution du moniteur. Cette évaluation est en revanche tout à fait réalisable pour des modèles de contrôle d'accès au périmètre limité. Les principaux goulots d'étranglement que nous avons constatés expérimentalement sont le nombre de contraintes arithmétiques et les interactions entre hiérarchies multiples.

Ainsi, hormis pour les modèles basés sur l'identité dans lesquels sujets et utilisateurs sont confondus, dans le cas général on va principalement comparer des politiques du point de vue statique, c'est-à-dire comparer des triplets *Statique* obtenus avec différentes politiques, voire différents modèles.

Définition (Comparaison statique de politiques)**.** *Soient deux politiques de contrôle d'accès* $\mathbf{I_1}$ *et* $\mathbf{I_2}$ *intègres, exprimées respectivement sur les modèles de contrôle d'accès* $\mathbf{AC_1}$ *et* $\mathbf{AC_2}$*. Soient* $Statique_1$ *les autorisations statiques de la première politique et* $Statique_2$ *celles de la deuxième.* $Statique_1 \in \mathbf{I'_1} \supseteq \mathbf{I_1}$ *et* $Statique_2 \in \mathbf{I'_2} \supseteq \mathbf{I_2}$*.*

La politique de contrôle d'accès $\mathbf{I_1}$ *est dite plus* restrictive *que* $\mathbf{I_2}$ si et seulement si *:*

$$Statique_1 \subseteq Statique_2$$

La politique de contrôle d'accès $\mathbf{I_1}$ *est dite plus* permissive *que* $\mathbf{I_2}$ si et seulement si *:*

$$Statique_1 \supseteq Statique_2$$

Les politiques de contrôle d'accès $\mathbf{I_1}$ *et* $\mathbf{I_2}$ *sont dites* équivalentes si et seulement si *:*

$$Statique_1 \subseteq Statique_2 \text{ et } Statique_1 \supseteq Statique_2$$

Lors de l'évolution des modèles de contrôle d'accès – lorsque de nouveaux concepts intermédiaires sont ajoutés par exemple – la comparaison permet de s'assurer que les droits qui existaient *avant* modification du noyaux sont bien les mêmes que ceux *après* évolution. Pour la réalisation de la comparaison de politique, on peut s'appuyer sur la comparaison de l'évaluation d'une requête de sélection sur $Statique_1$ et $Statique_2$.

La section 5.5 propose une méthode permettant l'identification de nouveaux concepts intermédiaires hiérarchisés, implicitement présents dans des relations existantes qui garantit que la politique obtenue après l'introduction du nouveau concept est statiquement équivalente à l'ancienne. Le cas de l'introduction d'une nouvelle relation intermédiaire hiérarchisée dans un modèle de contrôle d'accès nécessite de manipuler le schéma du modèle de contrôle d'accès et de modifier la théorie associée. Nous avons défini les primitives qui permettent cette manipulation.

Si la manipulation d'un schéma est une opération relativement simple pour un non-logicien, la définition d'une théorie logique ne l'est pas. La section suivante propose une représentation graphique des théories logiques qui permet aux concepteurs de modèles et aux administrateurs de visualiser schémas, principes et propriétés des modèles de contrôle d'accès.

5.4 Représentation graphique de modèles

Le processus de conception de modèle est complexe, à la croisée de plusieurs disciplines comme l'ingénierie, la qualité, la conception logicielle et la représentation des connaissances. Pour concevoir un modèle de contrôle d'accès efficace, il est nécessaire de formaliser certains aspects des organisations.

Afin de pouvoir mener à bien la conception de modèles sur mesure comme celui du GMSIH, il est nécessaire de proposer une représentation des modèles, qui permette une visualisation et une manipulation graphique de la théorie logique $P \cup \Sigma$.

Les auteurs de [Ferraiolo03a] ont proposé un *Role Control Center* spécifique pour l'administration de politiques de contrôle d'accès à rôles. Cette approche *ad hoc* ne propose pas d'outils pour la vérification des modèles et des politiques. Contrairement à cette proposition, nous proposons une représentation de modèles de contrôle d'accès générique ainsi que des outils pour inférer graphiquement. Nous pensons qu'une interface de conception de modèle doit :

- avoir une interface graphique appropriée,
- être capable d'exprimer une grande variété de concepts, relations, principes et propriétés des modèles de contrôle d'accès,
- disposer de mécanismes permettant de suivre graphiquement les raisonnements automatisés effectués sur les politiques,

Nous avons presenté, en section 2.6.3 de l'état de l'art, les principales propositions de modélisation graphiques génériques du contrôle d'accès. Nous allons dans cette section présenter une représentation visuelle des modèles de contrôle d'accès en correspondance directe avec le cadre logique que nous avons utilisé. Nous allons pour cela nous appuyer sur les *graphes conceptuels*.

5.4.1 Graphes conceptuels

Les graphes conceptuels sont un modèle de représentation et d'inférence graphique. Ils ont été introduits au début des années 80 par John Sowa pour la représentation des connaissances [Sowa84]. Depuis, ils ont été considérablement développés et enrichis. Il s'agit d'un modèle formellement bien fondé doté d'une sémantique logique [Chein92]. La figure 5.3 est un exemple de graphe conceptuel. Formellement, les graphes conceptuels sont définis comme des graphes bipartis non orientés :

- deux types de sommets existent : les *concepts* (rectangles) et les *relations* (ellipses),
- deux concepts ou deux relations ne peuvent pas être relié(e)s directement,
- un *vocabulaire* des types de concepts et de relations forme le support. Types de concepts et de relations sont organisés en treillis,
- à chaque sommet du graphe sont associés un *type* de concept ou de relation ainsi qu'un *marqueur* qui est soit :
  - le *marqueur générique* $*$ permet d'exprimer que le sommet est d'un type, mais sans l'identifier spécifiquement,
  - un *marqueur individuel* qui permet d'identifier le sommet.

De plus, on peut définir dans le cadre des graphes conceptuels des règles de graphes avec le connecteur $\rightarrow$. Les règles de graphes permettent de dériver de nouvelles connaissances à partir de celles connues. Une règle est composée de deux graphes : un qui forme l'hypothèse et l'autre la conclusion de la règle. Si plusieurs marqueurs génériques sont définis dans une règle, on les différencie par un identifiant apposé au marqueur générique, comme $*a$ ou $*b$. Les figures 5.4 et 5.5 sont des règles de graphes conceptuels.

Nous ne détaillerons pas plus le cadre théorique des graphes conceptuels. Nous laisserons de côté la formalisation issue de la théorie des graphes, car nous mettrons principalement à profit les liens entre règles de graphes conceptuels et logique.

5.4.2 Correspondance avec les dépendances

Les graphes conceptuels sont en correspondance directe avec un fragment de la logique du premier ordre. Les opérations classiques de la logique trouvent leurs équivalents en graphes conceptuels [Coulondre98, Wermelinger95]. La notion centrale d'*unification* logique correspond à la notion d'*isomorphisme* de graphes. Les règles de graphes sont de même à mettre en correspondance avec des formules de la logique du premier ordre ne comportant qu'un seul symbole d'implication.

5.4.2.1 Fragment de la logique

Le fragment considéré est en fait celui des *dépendances génératrices de tuples* sans contraintes. Ceci a permis d'exploiter certains résultats développés pour les graphes aux dépendances de données, notamment la notion de *pièce de graphe*. Une procédure de preuve en marche arrière pour les dépendances génératrices de tuples a été développée ainsi [Coulondre03]. Nous allons mettre à profit la correspondance entre dépendances et graphes pour la représentation des principes et des propriétés des modèles de contrôle d'accès.

Nous sommes cependant limités au fragment logique des dépendances génératrices de tuples sans contraintes. Cette classe de dépendances permet cependant d'exprimer les principes et les propriétés des modèles RBAC. La principale limitation pour disposer

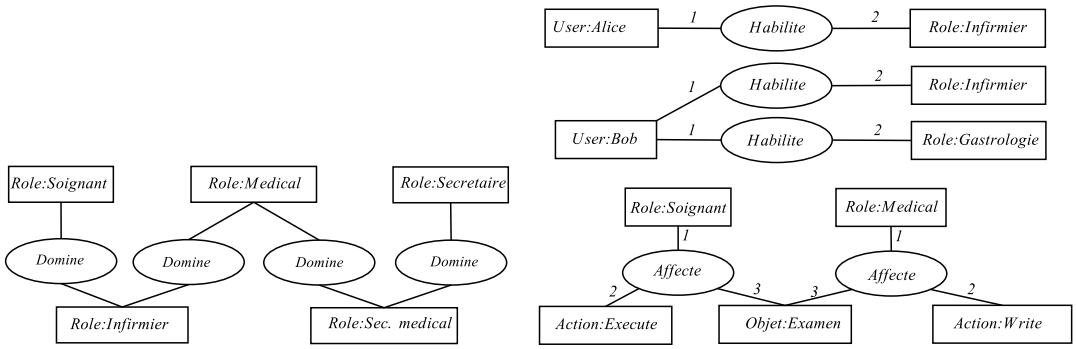
logique du premier ordre	$\overset{\Phi}{\underset{\Phi^{-1}}{\leftrightarrows}}$	**graphes conceptuels**
relation unaire		type de concept
relation n-naire		type de relation
symbole de prédicat		type de concept ou de relation
symbole de constante		marqueur individuel
variable		marqueur générique $*$
faits (sans variable)		graphe faits (sans générique)
base de données		base de connaissance
procédure de preuve		chaînage
dépendance		règle de graphes

TAB. 5.4 – Applications de correspondance Φ et Φ^{-1}

d'un cadre graphique plus expressif qui comprendrait en son sein les graphes conceptuels est l'impossibilité d'exprimer des contraintes arithmétiques dans les graphes. Malgré la restriction au fragment des dépendances génératrices de tuples, les graphes disposent d'un pouvoir suffisant pour représenter les notions du chapitre précédent, à l'exception des aspects contextuels et des prérequis. Nos arguments en faveur de l'utilisation des graphes conceptuels sont :

- les graphes conceptuels sont munis d'une sémantique logique. Une fonction Φ permettant de passer des graphes conceptuels et des règles de graphes à leurs pendants logiques. Sa réciproque Φ^{-1} étant définie également [Chein92, Wermelinger95]. Le tableau 5.4 illustre ces applications,
- le marqueur générique est équivalent aux quantificateurs existentiels et universels, selon sa présence en hypothèse ou conclusion des règles de graphes. Ce marqueur permet de représenter les principes et les propriétés des modèles,
- les graphes ont un pouvoir d'expression supérieur aux règles DATALOG,
- les mécanismes d'inférence dédiés, le *chaînage de graphes*, permettent une représentation des raisonnements fondés sur des opérations graphiques élémentaires [Chein92, Salvat96].

Nous découvrons que le choix fait *a priori* du terme de *concept* et de celui de *relation* n'était pas anodin. En effet, la signification que nous leur avons donnée – c'est-à-dire respectivement des relations unaires et n-aires – est celle des graphes conceptuels. Nous avons fait ce choix, car il conforte l'intuition qu'un grand nombre de connaissances, dont les modèles de contrôle d'accès font partie, peuvent être exprimées en termes de concepts et de relations. La possibilité de représentation est un des postulats fondamentaux des graphes conceptuels [Sowa84].



(a) Hiérarchie de rôles (b) Affectations

FIG. 5.3 – Représentation de faits avec les graphes conceptuels

5.4.2.2 Support des graphes conceptuels

Le support des graphes conceptuels permet de modéliser des relations d'héritage entre *types* de concepts. Nous avons choisi de ne pas utiliser cette fonctionnalité pour représenter les concepts hiérarchisés des modèles de contrôle d'accès, définis en section 4.3.4. Deux arguments nous ont conduits à effectuer ce choix :

- d'une part on évite le risque une explosion combinatoire du nombre de concepts à cause de la structure de treillis et à cause des affectations multiples possibles de rôles aux utilisateurs. En modélisant les rôles d'un modèle RBAC avec le treillis de concept, pour chaque affectation multiple de rôles à un utilisateur donné, comme d'utilisateurs multiples à un même rôle, il faut créer un sous-rôle dans le support,
- d'autre part, modéliser la hiérarchisation d'un concept avec le treillis de concepts rompt les liens exploités avec l'application de correspondance Φ. En effet, si chaque rôle devient un type, il serait alors un symbole de prédicat dans l'équivalent logique du graphe. Cette approche interdit de quantifier des variables représentant des rôles. C'est une critique que nous avons formulée à l'encontre de la modélisation de [Halpern03].

En revanche, le support des graphes conceptuels permet de pour sous-typer les concepts et les relations d'un modèle de contrôle d'accès. En section 3.4.6 nous avons montré que la notion d'exclusion mutuelle peut être sous-typée selon les restrictions qu'elle impose. De même, en section 4.5.5, nous avons proposé de raffiner la notion de domination de rôles avec des contraintes supplémentaires.

Les treillis de concepts ou de relations permettent ainsi d'exprimer des dépendances entre des aspects des modèles de contrôle d'accès. On peut grâce à eux, exprimer des règles de la forme $R_1(X_1) \rightarrow R_2(X_2)$ qui expriment qu'une relation R_1 « est une » relation R_2. Cette approche a été proposée dans [Bertino03] pour organiser entre eux des concepts et relations d'un modèle de contrôle d'accès.

Exemple 5.3 Professionnels de santé et patients

Un nouveau projet du CHM a pour but de réorganiser les droits d'une application de gestion de dossiers patients. Depuis la loi du 30 mars 2001, il doit être permis aux patients d'accéder à leurs dossiers. Le projet a pour but de rendre cet accès possible. Dans le modèle de contrôle d'accès, on définit ainsi deux sous-catégories du concept d'utilisateur : les professionnels de santé et les patients.

Ce sous-typage du concept d'utilisateur (symbole de prédicat $User$) est représentable par deux dépendances qui ont leur correspondance dans le treillis des concepts des graphes conceptuels :

- $Patient(U) \rightarrow User(U)$
- $Professionnel(U) \rightarrow User(U)$

5.4.3 Représentation graphique de dépendances

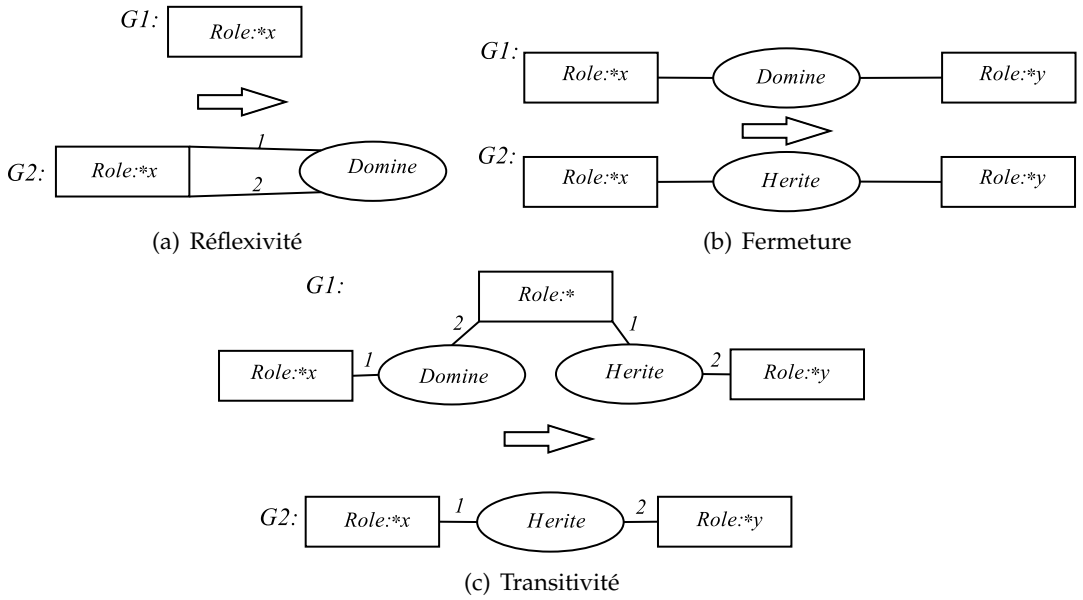


FIG. 5.4 – Hiérarchisation de rôles avec les graphes

Les fragments de la logique du premier ordre des différentes classes de dépendances de données sont au centre de notre formalisation du contrôle d'accès. Les procédures de preuve de l'implication constituent quant à elles le cœur des mécanismes de vérification que nous proposons. Représenter graphiquement ces deux composants théoriques est un premier pas vers la conception d'un atelier logiciel pour la conception et l'administration de politiques. Disposer d'une représentation graphique associée au cadre logique que nous utilisons améliore l'utilisabilité de notre proposition.

Comme dans le chapitre 3 et le chapitre 4, nous avons à plusieurs reprises illustré notre proposition avec son application aux modèles RBAC. Nous allons reprendre ces éléments de modélisation avec les graphes conceptuels. Nous allons faire correspondre aux dépendances que nous avons construites des règles de graphes conceptuels. Notons que pour le cas des dépendances génératrices de nullité, il faut faire correspondre à l'antilogie $\perp$ un type de concept spécifique.

Hiérarchisation de concepts En section 4.3.4 nous avions proposé une généralisation de la notion de hiérarchie des concepts. La figure 5.4 illustre l'utilisation des graphes conceptuels pour représenter graphiquement les règles qui permettent de définir qu'une relation d'héritage soit la fermeture transitive et réflexive d'une relation de domination directe. Ils sont en relation, par l'intermédiaire de la fonction Φ, avec les dépendances suivantes :

- figure 5.4 (a) : $Role(X) \rightarrow Role(X), Hérite(X, X)$,
- figure 5.4 (b) : $Domine(X, Y) \rightarrow Hérite(X, Y)$,
- figure 5.4 (c) : $Domine(X, Z), Hérite(Z, Y) \rightarrow Hérite(X, Y)$.

Principe fondamental La *core-rule* des modèles RBAC définit comment dériver les autorisations statiques des utilisateurs. Ce principe fondamental peut être exprimé par la règle de graphes de la figure 5.5 ou l'expression logique associée :

$$\forall U, R_u, R_p, A, O\ User(U), Role(R_u), Role(R_p), Action(A), Objet(O)$$
$$Habilite(U, R_u), Hérite(R_u, R_p), Affecte(R_p, A, O) \rightarrow Statique(U, A, O)$$

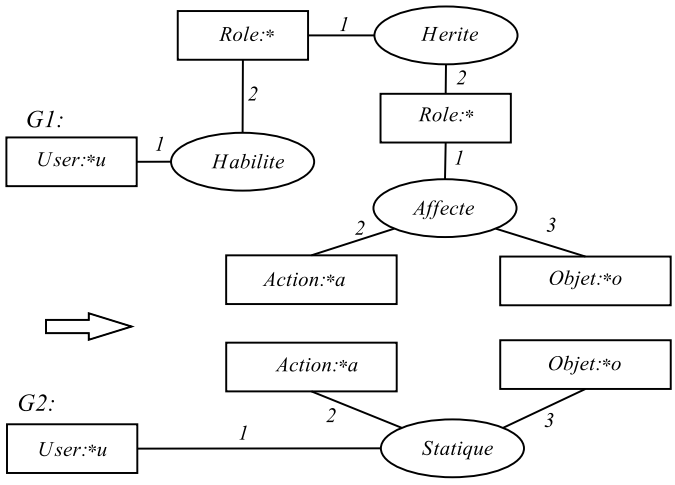


FIG. 5.5 – *Core-rule* des modèles RBAC

Représentation des politiques Les graphes conceptuels sont des outils de représentation et de manipulation de connaissance. L'ensemble des faits connus dans la base est représenté par des graphes simples, qui ne comportent pas de marqueur générique. Grâce aux règles de graphes, de nouvelles connaissances, c'est-à-dire de nouveaux graphes, peuvent être déduites. La figure 5.3 présente des graphes de faits, c'est-à-dire la représentation visuelle d'une politique **I**. Deux sous graphes sont présentés, le premier est un extrait de hiérarchie de rôle (a), le second est un exemple d'affectation de rôles à des utilisateurs et de permissions à ces rôles.

5.4.4 Raisonnement avec les graphes

Les procédures de preuves dédiées aux graphes, les chaînages, sont définies en termes d'opérations élémentaires sur les graphes comme la fusion et la scission de sommets. On peut donc visualiser les différentes étapes effectuées par les algorithmes de chaînage. Ceci facilite l'analyse des étapes des raisonnements effectués sur les graphes conceptuels.

Supposons que l'on souhaite vérifier s'il existe un utilisateur disposant du droit d'`exécution` et de celui d'`écriture` sur les `examens` dans une politique. Ce raisonnement fait intervenir les règles de graphes ainsi que la politique jouet précédemment présentée. Nous allons utiliser une preuve en marche arrière qui répond positivement à cette vérification. Ce chaînage tâche d'identifier successivement des isomorphismes de graphes entre les hypothèses des règles et la base de connaissance.

Le processus qui conduit à prouver qu'effectivement, il existe bien un utilisateur qui dispose de deux droits est décomposable en dix étapes. À chaque étape, une seule règle de graphe est appliquée. Les sous-buts successifs que la procédure essaie de prouver sont entourés en pointillés dans les figures 5.6 et 5.7. Les étapes sont les suivantes :

1. pour prouver qu'un utilisateur dispose des droits d'`exécution` et d'`écriture`, il faut chercher des rôles qui disposent de ces permissions (étapes 1 à 3),
2. le rôle de `soignant` dispose d'une de ces permissions (étapes 3 à 4),
3. le rôle de `médical` dispose également de ces permissions (étapes 3 à 4),
4. les rôles héritent des permissions de leurs parents. Le rôle `Infirmier` hérite à la fois de `soignant` et de `médical`, (étapes 4 à 8),
5. `Alice` et `Bob` disposent de ce rôle (étapes 8 à 10).

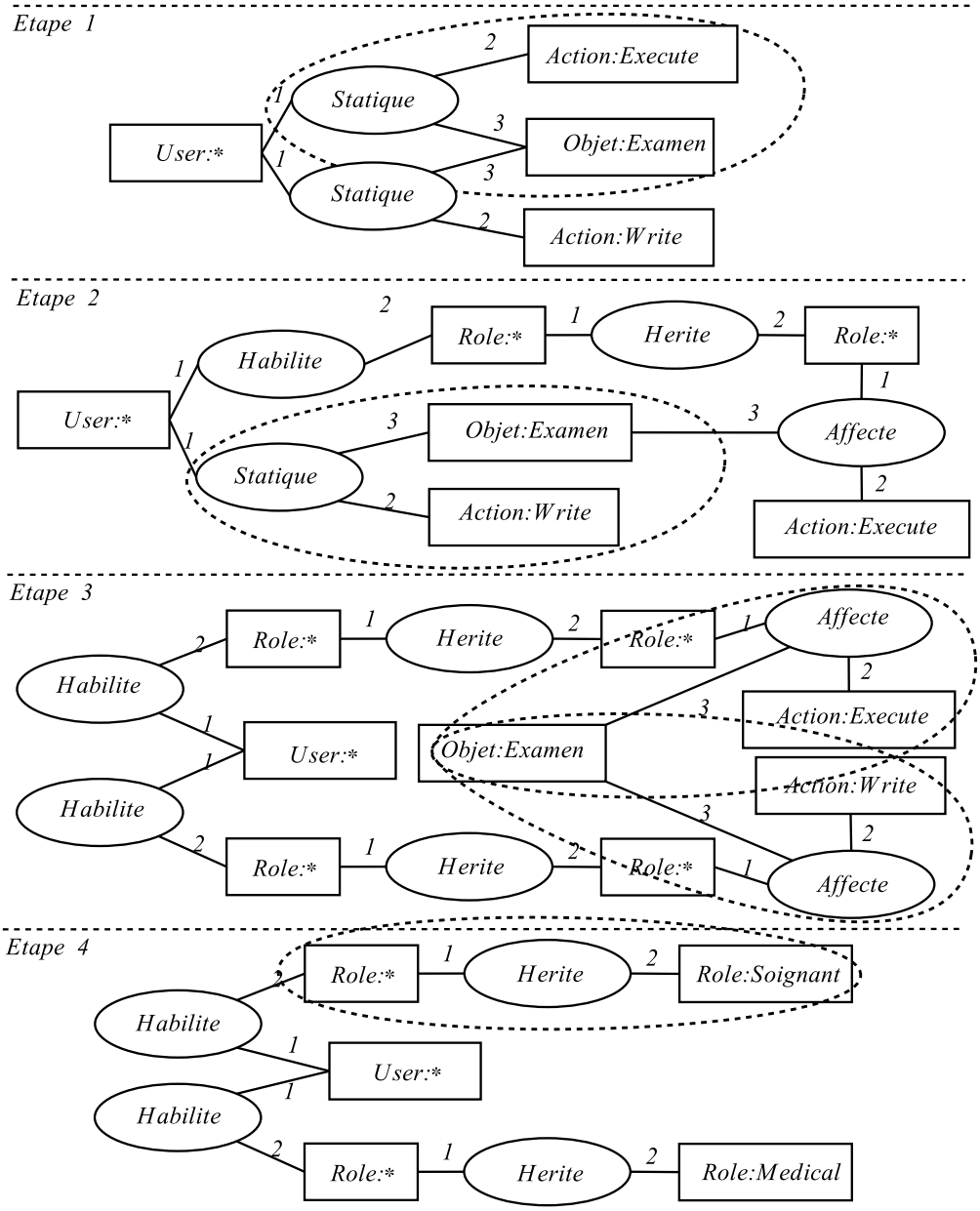


FIG. 5.6 – Procédure de preuve en marche arrière sur les graphes (1/2)

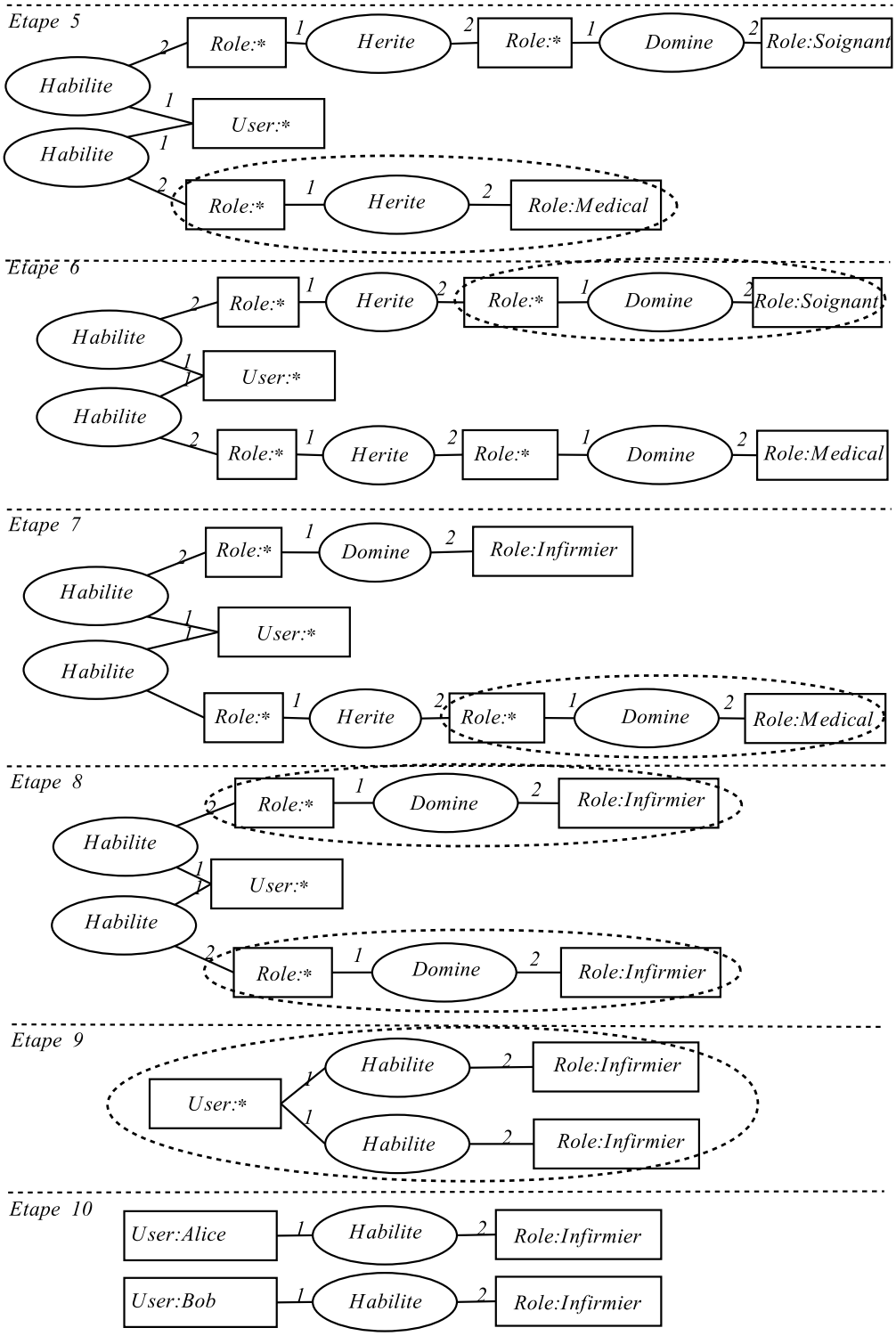


FIG. 5.7 – Procédure de preuve en marche arrière sur les graphes (2/2)

Ainsi, il existe bien au moins un utilisateur, en l'occurrence identifié par le marqueur individuel `Alice` ou `Bob`, qui dispose des deux droits. Techniquement, la procédure à réussi à dériver le but vide et les étapes de l'inférence forment une trace linéaire. La linéarité de la trace est un argument en faveur de l'utilisation du chaînage par des utilisateurs qui ne sont pas spécialistes de la logique.

Une des rencontres principales sur les graphes conceptuels est l'*International Conference on Conceptual Structures* (ICCS). C'est en participant à cette rencontre que nous avons découvert le cadre de l'analyse de concepts formels, qui nous a permis de développer une méthode de découverte de concepts hiérarchisés implicitement présents dans les politiques de contrôle d'accès.

5.5 Ingénierie de rôles hiérarchisés

Quand les organisations évoluent, les politiques de contrôle d'accès doivent refléter les changements qui surviennent. Les évolutions mineures peuvent être prises en compte en mettant à jour les politiques sans changer les modèles. Cette approche est acceptable pour les restructurations où les fonctions des salariés changent au sein de l'organisation. Dans le cas d'évolutions majeures cependant, il faut modifier le modèle de contrôle d'accès lui-même.

Exemple 5.4 Fusion d'organisations

Supposons, dans le cadre d'un remaniement ministériel majeur, que le CHM jusque-là autonome soit regroupé avec un ou plusieurs autres établissements de santé pour former une nouvelle entité régionale. Cette fusion d'organisations impose de complètement réorganiser les systèmes d'information de santé existants pour en créer un nouveau à l'échelle de la région.

Chaque établissement disposait de son propre modèle de contrôle d'accès. Lors de la fusion, il va falloir réorganiser les droits, pour disposer d'un modèle de contrôle d'accès commun et homogène au nouvel établissement issu de la fusion, par exemple le Centre Hospitaler Commun Drômardèchois (CHCD).

La définition d'un nouveau modèle global, en adéquation avec la nouvelle organisation administrative du CHDC, va fortement impacter les politiques déjà existantes. Dans le modèle de contrôle d'accès commun, on va introduire de nouveaux concepts intermédiaires pour représenter la nouvelle structure administrative. On peut légitimement introduire tout ou partie des concepts et relations des modèles ORBAC ou du GMSIH.

Les tâches d'administrations qui résultent de changements dans la structuration des droits sont lourdes : reprise des droits existants, vérification de non-régression ou vérification de l'intégrité de la nouvelle politique. Ces activités sont représentées par les boucles de rétroaction de la figure 5.2.

Dans les modèles de contrôle d'accès structurés, la hiérarchisation des concepts joue un rôle prépondérant dans l'administration des politiques : c'est un élément pivot qui doit refléter la structure de l'organisation qui met en œuvre le contrôle d'accès.

L'introduction de nouveaux concepts intermédiaires hiérarchisés dans un modèle (ou la hiérarchisation de concepts organisés à plat) est un problème commun aux modèles de contrôle d'accès hiérarchisés. Il est important de proposer des outils permettant d'assister cette tâche. Dans le cadre des modèles à rôles, cette activité est appelée l'*ingénierie des rôles*. Cette section propose une automatisation – partielle – de cette activité, généralisée au cadre relationnel de contrôle d'accès que nous proposons.

5.5.1 Ingénierie des rôles

Un des arguments principaux en faveur d'une gestion des droits basée sur des rôles hiérarchisés est que les coûts de l'administration des politiques sont réduits [Crampton03a, Ferraiolo03b]. La hiérarchisation permet en effet de réduire la taille des politiques **I** en proposant des affectations implicites déduites à partir des propriétés algébriques de la relation d'héritage entre rôles. Malheureusement, la définition de rôles hiérarchisés dans l'organisation des droits est une activité coûteuse en termes :

- de conception de modèle. L'introduction d'un nouveau concept hiérarchisé dans un modèle existant impose l'identification et la définition de nouveaux principes et propriétés du modèle,
- de conception de politique, c'est-à-dire le coût de l'identification et de définition des relations entre les concepts par les administrateurs.

5.5.1.1 Probématique

Le travail d'Edward Coyne est à l'origine du terme *ingénierie des rôles* [Coyne96]. Sa proposition centre l'activité d'ingénierie comme la définition des affectations de permissions aux rôles, sous-entendant aussi celles de rôles aux utilisateurs :

> *The concept of role engineering (RE) is an approach to defining roles and assigning permissions to the roles. RE must capture the organization's business rules, as these relate to access control, ...*

L'ingénierie des rôles est entreprise lors de la conception d'une politique de contrôle d'accès RBAC. Si l'ingénierie des rôles est une activité essentielle pour que l'établissement d'une politique RBAC soit fructueux, c'est également l'activité la plus coûteuse et la plus longue. Un rapport du NIST indique que le coût d'identification de chaque rôle est évalué à environ 5.000 dollars [Gallaher02].

La thèse de Pete Epstein a considérablement étendu Coyne [Epstein02]. L'activité d'ingénierie des rôles définie par Epstein peut être résumée par « l'identification d'un ensemble de rôles qui soit *complet*, *correct* et *efficace* ».

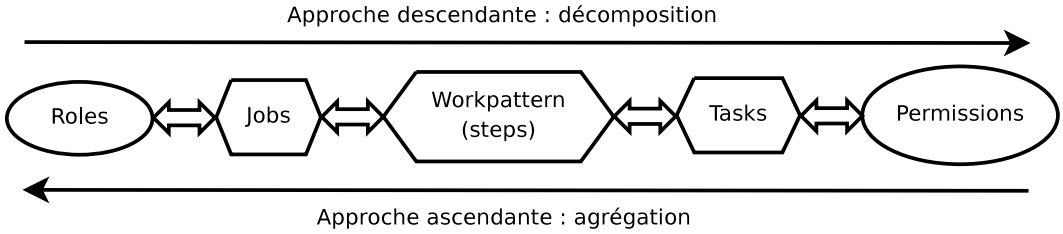


FIG. 5.8 – Approches d'ingénierie des rôles [Epstein02]

Pour ses besoins méthodologiques, Epstein a étendu le modèle RBAC_1 en introduisant trois nouveaux concepts intermédiaires entre rôles et permissions : *Jobs*, *WorkPatterns* et *Tasks*. Ce nouveau modèle a été baptisé *Role Permission Assignment Model 2001* (RPAM1). Ce modèle a permis à Epstein d'identifier et de développer deux principales stratégies d'identification des rôles, illustrées par la figure 5.8 :

- l'approche *descendante* est basée sur la décomposition des rôles en concepts intermédiaires. Cette approche se base sur la structure de l'organisation, mais pas sur les privilèges existants. Elle a été appliquée avec succès dans de grandes structures [Roeckle00], et étendue par des méthodes à base de scénarios et de cas d'utilisation [Neumann02]. Cette approche, basée sur une modélisation rigoureuse des processus métiers, n'a pas à notre connaissance été automatisée. Elle est adaptée aux organisations qui déploient un modèle RBAC *ex nihilo*,
- l'approche *ascendante*, dite *role-mining* [Kuhlmann03] est une identification des rôles implicitement présents dans le système. Cette approche est basée sur l'agrégation des permissions existantes, sans tenir compte cependant de la structure de l'organisation. Cette approche se base sur des droits existants.

5.5.1.2 Objectifs

Il est rare qu'une organisation mette en place un modèle de contrôle d'accès RBAC sans se baser sur des droits existants. Un des objectifs des travaux en ingénierie des rôles est d'automatiser l'approche ascendante d'identification des rôles et de leurs hiérarchies. Le but est de faciliter et de réduire la durée et le coût de l'ingénierie des rôles. La proposition de cette section va dans ce sens en proposant une méthode ascendante d'identification automatisée de rôles.

Nous considérerons les modèles de contrôle d'accès comme des moyens pour structurer des politiques. L'objectif de tout modèle de contrôle d'accès est de permettre la dérivation du triplet fondamental $\mathcal{ACCESS} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{O}$. Nous avons également défini un triplet statique *Statique* entre utilisateurs, actions et objets. Nous envisageons ansi l'activité d'ingénierie des rôles comme une activité de rétroconception qui tâche d'identifier des relations intermédiaires dans une relation *Statique* existante.

La notion de rôle dans les modèles RBAC diffère de la notion de groupe : un groupe est essentiellement une collection d'utilisateurs alors que le rôle est *à la fois* une collection d'utilisateurs *et* de permissions [Sandhu96]. C'est donc un concept qui peut être défini soit par son intention – les permissions dont dispose le rôle – soit par son extension – les utilisateurs qui l'endossent. Cette définition nous sert de base pour notre démarche. Nous allons assimiler la notion de *rôle* dans les modèles RBAC à celle de *concept formel* de l'analyse de concepts formels (*Formal Concept Analysis* – FCA).

5.5.1.3 Méthode

Nous mettons en correspondance la formalisation des modèles de contrôle d'accès RBAC et celle de l'analyse de concepts formels. Basés sur la correspondance entre rôle et concept formel, nous développons le parallèle entre ces deux domaines jusqu'à parvenir à définir l'ingénierie des rôles en terme d'extraction de concepts formels. Les concepts formels sont présentés comme des rôles potentiels calculés. La hiérarchie de rôles obtenue est une base de travail pour les concepteurs de politique. Dans le cas général les concepts formels sont des intermédiaires à identifier dans une relation binaire existante.

Cette section décrit successivement les contextes fomels et les treillis de concepts. Nous exprimerons ensuite l'ingénierie des rôles en termes de concepts formels que nous mettons en parallèle avec la décomposition d'une relation binaire entre utilisateurs et permissions. Nous décrivons enfin l'algorithme utilisé pour identifier les rôles. Comme nous l'avons fait en section 5.3.5 et avec les mêmes arguments, nous nous limiterons à l'identification de concepts hiérarchisés statiques.

5.5.2 Matrice de contrôle d'accès et contexte formel

Le tableau 2.2 est un exemple jouet de matrice de contrôle d'accès, impliquant quatre sujets (en lignes), quatre objets (en colonnes) et trois actions (`r` pour *read*, `w` pour *write* et `x` pour *execute*). Pour l'application présentée, on identifie les sujets de cette matrice à des utilisateurs.

5.5.2.1 Restriction à une relation binaire

Tout d'abord, nous décomposons la relation ternaire *Statique* en deux relations à l'aide du concept de permission. Le concept de permission $\mathcal{P}$ est défini par les paires d'actions et d'objets utilisées dans la politique de contrôle d'accès. On définit ainsi une relation $Permet \subseteq \mathcal{U} \times \mathcal{P}$ qui est une restriction de la relation *Statique*. En effet, dans les modèles basés sur RBAC, on n'affecte jamais directement d'action ou d'objet à un rôle, on affecte des couples d'action et d'objet, c'est-à-dire des permissions.

$$Statique(U, A, O) \leftrightarrow \exists P\ Perm(P, A, O), Permet(U, P)$$

Nous définissons également les deux dépendances fonctionnelles suivantes qui imposent l'équivalence entre un identifiant de permission et un couple d'action et d'objet :

$$\begin{array}{rcl} Perm(P,A,O), Perm(P,A',O') & \rightarrow & A = A', O = O' \\ Perm(P,A,O), Perm(P',A,O) & \rightarrow & P = P' \end{array}$$

L'intérêt de cette transformation est de pouvoir ensuite manipuler une matrice booléenne, car de nombreux problèmes de fouille de données ont été caractérisés sur de telles structures et de nombreux algorithmes sont disponibles. De plus, dans plusieurs cas d'utilisation on a seulement accès aux identifiants de permissions et pas à leur décomposition en action et objet.

Le tableau 5.5 présente graphiquement la relation *Permet* construite à partir de la relation *Statique* du tableau 2.2. Nous avons pris comme identifiant de chaque permission la concaténation des identifiants d'action et d'objet. Ainsi, `r1` indique le droit de lecture sur le fichier `Fichier1`. Cette relation *Permet* est interprétée dans notre approche, soit comme une restriction de la matrice de Lampson, soit comme un contexte formel.

5.5.2.2 Contexte formel

La formalisation de l'analyse de concepts formels part d'un *contexte formel* : une relation *binaire* entre objets et attributs. L'opération d'analyse consiste, à partir d'un contexte formel, à identifier une hiérarchie de concepts aux propriétés particulières [Wille97].

Définition (Contexte formel et application α et ω)**.** *Un* contexte formel *est un triplet* $\mathbb{K} = (O, A, J)$ *où* O *est l'ensemble des objets,* A *l'ensemble des attributs et* $J \subseteq O \times A$ *la relation d'incidence. La relation d'incidence* J *induit deux applications* α *et* ω *qui relient* 2^O *et* 2^A*.*

$$\begin{array}{l} \forall X \in 2^O, \alpha(X) = \{y \in A \mid \forall x \in X, (x,y) \in J\} \\ \forall Y \in 2^A, \omega(Y) = \{x \in O \mid \forall y \in Y, (x,y) \in J\} \end{array}$$

Les applications α *et* ω *définissent une* correspondance de Galois *entre* 2^O *et* 2^A*. Les applications* $\alpha \circ \omega$ *et* $\omega \circ \alpha$ *définissent deux opérateurs de fermeture sur* 2^O *et* 2^A *respectivement.*

Pour l'application à la découverte de rôles, les *objets* de la FCA sont des *utilisateurs* et les *attributs* des permissions. La relation d'incidence indique quels sont les droits dont les utilisateurs disposent. Nous pouvons ainsi décrire le contexte formel $\mathbb{K}$: ce serait la matrice des droits d'accès du système, la relation *Permet* du tableau 5.5.

Les applications α (resp. ω) s'interprètent alors comme l'application qui à un ensemble d'utilisateurs (resp. de permissions) fait correspondre l'ensemble des permissions (resp. d'utilisateurs) avec lesquelles *tous* les utilisateurs (resp. permissions) de l'ensemble sont en relation.

	r1	w1	r2	w2	r3	w3	r4	w4	x4
Alice	×	×	×		×				
Bob	×		×	×	×		×	×	×
Charly	×		×		×	×	×	×	×
Denise					×		×		

TAB. 5.5 – Contexte formel jouet

5.5.3 Rôles hiérarchisés et treillis des concepts formels

Les sous-ensembles fermés de O et de A, notés C^o et C^a, présentent deux propriétés fondamentales de la FCA : α et ω sont des bijections entre ces familles, appelées *correspondances de Galois*. De plus, ces familles munies de l'inclusion ensembliste forment deux treillis complets isomorphes, appelés *treillis de Galois*. Pour notre application cela signifie que grâce à α et ω, on va pouvoir identifier des couples d'utilisateurs et de permissions (u, p) qui sont organisés selon un treillis. Avec (u, p) et la définition que nous avons donnée de P nous pouvons retrouver le triplet (u, a, o).

La hiérarchie des rôles est une des innovations proposées dans les modèles RBAC. Elle permet de réduire le nombre de rôles et d'affectations (entre rôles et utilisateurs, ou entre rôles et permissions) en introduisant une relation binaire d'héritage entre rôles.

La formalisation ensembliste de RBAC est résumée par le tableau 2.5. Les applications *assigned_users* et *assigned_perms* (affectés) ainsi que par *auth_users* et *auth_perms* (autorisés) fondées sur ces ensembles et la relation d'ordre partiel sur les rôles sont données en section 2.3.2. Nous allons mettre ces applications en correspondance avec celles de l'analyse de concepts formels.

5.5.3.1 Treillis des concepts formels

Définition (Concept formel et treillis)**.** *Une paire d'ensembles en correspondance $c = (X, Y)$ tels que $Y = \alpha(X)$ (ou $X = \omega(Y)$) est appelée* concept formel*.*

L'ensemble des concepts C muni de la relation d'ordre partiel $\leq_C$: $(X_1, Y_1) \leq_C (X_2, Y_2) \Leftrightarrow X_1 \subseteq X_2$ (ou $Y_2 \subseteq Y_1$) forme un treillis de Galois appelé treillis de concepts*.*

L'analyse de concepts formels définit les concepts comme des ensembles *maximaux* d'objets partageant les mêmes attributs. Chaque concept $c = (X, Y)$ définit ainsi deux relations : une première entre objets et concept et une seconde entre concept et attributs. L'ensemble X des objets d'un concept est appelé *extension* (complète), l'ensemble Y des attributs est appelé *intention* (complète).

Théorème (Théorème fondamental de la FCA)**.** *L'ensemble C de tous les concepts qui peuvent être construits sur un contexte formel $\mathbb{K} = (O, A, J)$ donné forme un* treillis complet *quand il est ordonné par l'inclusion ensembliste.*

Analyse de concepts	Contrôle d'accès
concept	concept intérmédiaire
objet	utilisateur
attribut	permission
relation d'ordre partiel	relation d'héritage
extension réduite	utilisateurs *explicitement* affectés
intention réduite	permissions *explicitement* affectées
extension complète	utilisateurs *implicitement* affectés
intention complète	permissions *implicitement* affectées
concepts ancêtres	rôles dont hérite un rôle
concepts descendants	rôles héritant d'un rôle

TAB. 5.6 – Identification des termes de FCA et de RBAC

Appliqué au contrôle d'accès, le treillis[3] des concepts définit deux relations entre utilisateurs et concepts d'une part, et concepts et permissions d'autre part. Ces deux relations correspondent respectivement aux relations *Habilite* et $Affecte$, qui relient rôles et utilisateurs et rôles et permissions. L'extension d'un rôle est l'ensemble de ses utilisateurs *autorisés*, son intention ses permissions *autorisées*. La relation d'ordre $\leq_C$ est considérée comme un ordre partiel entre les rôles : la relation d'héritage *Hérite*.

5.5.3.2 Mise en correspondance des applications

Pour l'application que nous proposons, nous utilisons indifféremment *utilisateur* pour *objet* ou *permission* pour *attribut*, selon que l'on s'exprime en termes *d'ingénierie des rôles* ou *d'analyse de concepts formels*. En prolongeant la correspondance, on peut exprimer les quatre applications définies avec la hiérarchie RBAC avec les termes de la FCA. Pour un rôle r donné :

- $assigned\_users : \mathcal{R} \rightarrow 2^{\mathcal{U}}$, fait correspondre au rôle r son l'*extension réduite*,
- $auth\_users : \mathcal{R} \rightarrow 2^{\mathcal{U}}$, fait correspondre à r son *extension complète*,
- $assigned\_perms : \mathcal{R} \rightarrow 2^{\mathcal{P}}$, fait correspondre à r son son *intention réduite*,
- $auth\_perms : \mathcal{R} \rightarrow 2^{\mathcal{P}}$, fait correspondre à r son son *intention complète*.

Nous arrivons ainsi à la définition d'un *rôle formel* : un couple d'utilisateurs et d'attributs en correspondance par d'une part, l'application $AuthUsers$ et $AuthPerms$ d'autre part. Le tableau 5.6 résume les principales correspondances que nous définissons et mettons en œuvre entre RBAC et FCA.

[3] Un treillis est un ensemble partiellement ordonné tel que chaque couple d'éléments admette une borne supérieure et une borne inférieure.

5.5.4 Analyse de concepts formels et ingénierie des rôles

L'intérêt des concepts intermédiaires entre utilisateurs et permissions est de permettre le regroupement de privilèges. Le tableau 5.5 présente en grisé le regroupement des permissions `r1`, `r2` et `r3`, dont les utilisateurs `Alice`, `Bob` et `Charly` disposent tous les trois. Ces groupements d'utilisateurs $X = \{\texttt{Alice}, \texttt{Bob}, \texttt{Charly}\}$ et de permissions $Y = \{\texttt{r1}, \texttt{r2}, \texttt{r3}\}$ sont tels que $\alpha(X) = Y$ et $\alpha(Y) = X$. Le couple (X, Y) est par définition un *concept formel*.

Dans un modèle de contrôle d'accès RBAC, ce regroupement est un *rôle*, disons `Infirmier`, attribué aux trois utilisateurs. C'est ce qu'illustre le tableau 5.7. On voit donc l'intérêt d'identifier des rôles : si un nouvel infirmier intègre l'organisation, il suffira de lui attribuer le rôle correspondant. Les administrateurs vont ainsi ajouter un seul tuple à la relation *Habilite* entre utilisateurs et rôles. Dans la gestion classique basée sur les identités, on aurait dû définir explicitement trois tuples dans la relation *Statique* : une pour chaque permission attribuée au rôle `Infirmier`.

5.5.4.1 Définition de l'ingénierie des rôles

Lors de l'introduction de la notion de rôle dans le contrôle d'accès on doit s'assurer que les utilisateurs disposent bien de tous les droits qu'ils avaient avant et *seulement* ceux-ci. L'objet de l'ingénierie de rôle est donc de décomposer une relation *Statique* existante en trois sous-relations.

Définition (Ingénierie des rôles hiérarchisés)**.** *Soit Statique une matrice de contrôle d'accès statique existante. L'*ingénierie de rôles hiérarchisés *est l'activité qui consiste à identifier*[4] *:*

- *un ensemble de rôles* $\mathcal{R}$*,*
- *une relation Habilite* $\subseteq \mathcal{U} \times \mathcal{R}$ *et une relation* $Affecte \subseteq \mathcal{R} \times \mathcal{A} \times \mathcal{O}$
- *une relation d'héritage Hérite* $\subseteq \mathcal{R} \times \mathcal{R}$ *réflexive, transitive et antisymétrique,*
- *avec Habilite* $\bowtie$ *Hérite* $\bowtie$ $Affecte = Statique$

Si l'on ne prend pas en compte la hiérarchie, c'est-à-dire si l'on s'intéresse seulement à trouver *Habilite* et $Affecte$ tels que *Habilite* $\bowtie$ $Affecte = Statique$, l'ingénierie de rôles a pour but de représenter la relation *Permet* du tableau 5.5 sous forme de deux relations binaires présentées dans le tableau 5.7. Ces deux dernières relations forment un exemple de résultat que l'on souhaite obtenir automatiquement. Dans cet exemple, cinq rôles (`Infirmier` (`I`), `Médecin` (`M`), `Gastrologue` (`G`), `Pédiatre` (`P`) et `Secrétaire` (`S`)) ont été identifiés pour structurer les droits du tableau 5.5.

La structure du treillis des concepts est beaucoup trop restrictive pour notre application. C'est une restriction importante portant sur la relation $Herite_R$ que l'on cherche à identifier. La structure de treillis impose que pour chaque couple de rôles $(r_1, r_2), \in \mathcal{R}^2$ il existe deux rôles $r_\top$ et $r_\bot$ tels que $r_\top \succeq r_1 \wedge r_\top \succeq r_2$ et $r_1 \succeq r_\bot \wedge r_2 \succeq r_\bot$.

[4] L'opérateur $\bowtie$ désigne la jointure naturelle de deux relations.

	I	M	G	P	S
Alice	×	×			
Bob	×		×		
Charly	×			×	
Denise					×

	r1	w1	r2	w2	r3	w3	r4	w4	x4
Infirmier	×		×		×				
Médecin		×							
Gastrologue				×			×	×	×
Pédiatre						×	×	×	×
Secrétaire					×		×		

TAB. 5.7 – Affectations de rôles aux utilisateurs et de permissions aux rôles

5.5.4.2 Sous-hiérarchie de Galois

Avec une structure de treillis, on risque de multiplier les rôles dans la politiques. Or ces rôles « imposés » par la structure de treillis et n'ont pas nécessairement d'utilisateurs ou de permissions affectés. La *sous-hiérarchie de Galois* (SHG) a été introduite par Godin [Godin95] dans le cadre de la restructuration de hiérarchies de classes, pour réduire le nombre de concepts du treillis des concepts formels. En effet, dans le pire des cas on peut obtenir jusqu'à $2^{min(|O|,|A|)}$ concepts formels. Pour l'application au contrôle d'accès, une hiérarchie d'une telle taille n'est pas concevable.

Définition (Sous-hiérachie de Galois)**.** *Un concept C' est dit ancêtre (resp. descendant) d'un concept C ssi $C' <_C C$ (resp $C' >_C C$). Un objet (resp. un attribut) x est dit être* introduit *par un concept si x fait partie de l'extension (resp. de l'intention) de ce concept et qu'aucun ancêtre (resp. descendant) de ce concept ne contient x dans son extension (resp. intention). C'est le concept-objet (resp. concept-attribut) de x.*

La SHG *d'un treillis de Galois $(C, \leq_C)$ est le plus petit sous-ordre qui ne* contient que *les concepts-objets et les concepts-attributs du treillis de Galois.*

Les objets introduits par un concept forment son *extension réduite*, ses attributs son *intention réduite*. La sous-hiérarchie de Galois supprime donc les concepts qui ne sont *ni concepts objets ni concepts attributs*. C'est concepts peuvent être supprimés, car ils ne véhiculent pas d'information. Ils sont seulement imposés par la structure particulière de treillis. Pour un concept donné, l'extension réduite est l'ensemble des objets qu'il introduit explicitement, soit pour le contrôle d'accès, ses utilisateurs *affectés*.

Pour le contrôle d'accès, cette propriété est particulièrement intéressante, car la SHG va supprimer de $(C, \leq_C)$ tous les rôles qui n'ont ni d'utilisateurs ni de permissions affectés. En d'autres termes, des rôles artificiels que très peu d'administrateurs auraient volontairement définis.

Avec la sous-hiérarchie, on ne garde ainsi que les *rôles-utilisateurs* (auxquels des utilisateurs sont affectés explicitement) et les *rôles-permissions* (auxquels des permissions sont affectées explicitement). Le choix de la sous-hiérarchie de Galois par rapport à la structure du treillis de concept permet d'assurer les critères de correction, complétude et d'efficacité qui contribuent à former une « bonne hiérarchie » de rôles.

5.5.4.3 Dérivation des autorisations

Nous avons défini qu'un modèle de contrôle d'accès doit permettre de dériver le triplet *Statique* qui est toujours un surensemble des permissions actives des utilisateurs *Dynamique*. Dans les modèles RBAC, il est interdit d'attribuer directement des permissions aux utilisateurs. En d'autres termes, on ne peut pas manipuler la matrice 2.2 directement. On doit dériver *Statique* à partir des relations *Habilite*, *Hérite* et $Affecte$.

On assimile l'application *auth_users* à une relation qui contient chaque paire (r, u) telle que $u \in auth\_users(r)$. La définition relationnelle de cette relation en intention est expressible par la règle de déduction

$$Habilite(U, R), Hérite(R, R') \rightarrow AuthUsers(R', U)$$

On peut effectuer le même procédé pour définir la relation $AuthPerms(R, A, O)$. Ces règles sont également l'expression de jointures naturelles entre une relation en extension et la relation en intention *Hérite*.

Définition (Dérivation des autorisations dans les modèles RBAC)**.** *Les autorisations des utilisateurs sont dérivées à partir des affectations et de la hiérarchie de rôles :*

$$Statique(U, A, O) \leftrightarrow$$
$$\exists R\ AuthUsers(R, U) \wedge AuthPerms(R, A, O)$$

Cette définition proposée est déduite à partir de [Ferraiolo03b] où elle ne figure pas explicitement, du moins formellement. Nous proposons une *équivalence* entre le triplet *Statique* et la présence d'un rôle dans la politique RBAC. L'implication de droite à gauche exprime comment dériver les autorisations. En revanche, l'implication de gauche à droite signifie que l'affectation à un rôle est *obligatoire* pour obtenir une permission dans une politique RBAC [Ferraiolo03b].

La question que nous nous posons est « qu'elle est la relation entre la sous-hiéarchie et le principe de dérivation des autorisations ? ». Nous remarquons que chaque utilisateur ou chaque permission n'apparait que dans un seul concept de la sous-hiérarchie. L'extraction des concepts-objets et concepts-attributs de la matrice de contrôle d'accès peut donc s'exprimer comme la recherche d'un ensemble partiellement ordonné tel que chaque utilisateur ne soit associé qu'à un seul et unique rôle, et chaque permission à un seul et unique rôle également.

$$Statique(U, A, O) \leftrightarrow$$
$$!\exists R\ AuthUsers(R, U) \wedge AuthPerms(R, A, O)$$

Nous allons donc proposer une heuristique pour supprimer cette contrainte d'unicité du rôle qui n'a pas lieu d'être dans une politique RBAC. En effet, les affectations multiples (de rôles aux utilisateurs comme de permissions aux rôles) sont autorisées. Cette redondance de l'information répond à des besoins pratiques d'administration du contrôle d'accès [Crampton03a].

5.5.5 Automatisation de l'ingénierie des rôles

L'analyse des relations binaires pour la recherche des ensembles fermés, dont font partie les concepts formels, est un domaine de recherche actif, en France notamment [Pasquier99, Nourine02]. Maintenant que nous avons établi le cadre pour l'ingénierie de rôle basé sur les concepts formels, nous allons pouvoir mettre à profit les résultats sur l'analyse des relations binaires.

5.5.5.1 Choix d'un algorithme

Les auteurs de [Arévalo07] ont proposé un algorithme PLUTON qui permet de calculer la sous-hiérarchie d'un contexte formel. Cet algorithme fait suite à [Berry05] qui propose une technique issue des graphes pour calculer une extension linéaire de la sous-hiérarchie de Galois. PLUTON est composé de trois sous-algorithmes TOMTHUMB, TOLINEXT et TOGSH :

- l'algorithme TOMTHUMB prend en entrée un contexte formel et retourne une liste ordonnée des labels[5] présents dans la SHG. Cette extension linéaire est un ordre total compatible avec l'ordre partiel $(C, \leq_C)$,
- l'algorithme TOLINEXT prend en entrée la liste résultat de TOMTHUMB et fusionne les paires extension et intention consécutives de cette liste qui appartiennent à un même concept. On obtient ainsi un tri topologogique de $(C, \leq_C)$,
- l'algorithme TOGSH calcule la SHG à partir du tri topologogique de TOLINEXT. Pour notre application, nous avons modifié cet algorithme pour qu'il calcule aussi les extensions et intentions complètes. Le surcoût engendré par ce calcul est négligeable face à la complexité de TOGSH [Arévalo07].

La complexité en temps théorique de TOMTHUMB est évaluée en $\mathcal{O}(|J|)$, l'implémentation brutale de TOLINEXT est décrite dans [Berry05] avec une complexité de $\mathcal{O}((|O| + |A|)^3)$, enfin la complexité de TOGSH a été évaluée en $\mathcal{O}((|O| + |A|)^2 * max(|O|, |A|)^2)$. D'autres algorithmes ont été proposés pour calculer la sous-hiérarchie de Galois, cependant PLUTON offre de bonnes performances vis-à-vis de ses concurrents [Arévalo07]. La figure 5.9 (a) est la sous-hiérarchie obtenue par TOGSH à partir du contexte formel du tableau 5.5.

5.5.5.2 Étapes de la méthode proposée

Nous proposons à l'expert du domaine chargé du chantier de d'ingénierie des rôles de mettre en œuvre notre approche semi-automatisée pour les assister dans leur travail. En effet, comme plusieurs hiérarchies de concepts potentielles permettent de couvrir entièrement une relation binaire, le choix final et les éventuelles modifications à

[5] Les labels sont des identifiants de lignes (ou de colonnes) identiques

Description du critère	
FCA	RBAC
cardinalité de l'extension complète	utilisateurs autorisés
cardinalité de l'extension réduite	utilisateurs affectés
cardinalité de l'intention complète	permissions autorisés
cardinalité de l'intention réduite	permissions affectées
surface complète	utilisateurs $\times$ permissions autorisés
surface réduite	utilisateurs $\times$ permissions affectées
concepts immédiatement supérieurs	nombre de pères
concepts immédiatement inférieurs	nombre de fils

TAB. 5.8 – Critères principaux proposés pour le classement des concepts

apporter au contexte formel si besoin, dépendent des besoins des organisations. Notre approche est décomposable en cinq étapes :

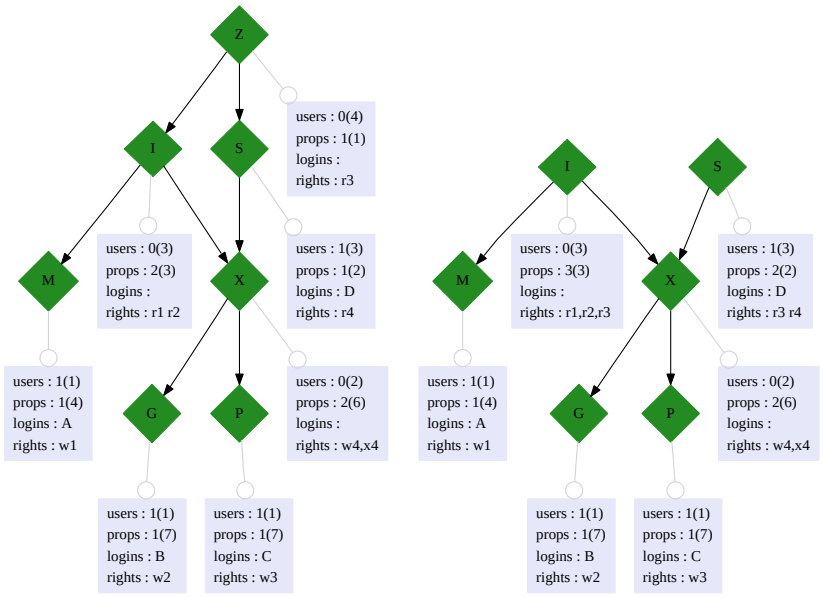
1. obtenir la *matrice des droits* d'accès du système sous forme de contexte formel,
2. calculer la *sous-hiérarchie de Galois* associée à l'aide de PLUTON, qui comporte les concepts potentiels du système.

Ces deux premières étapes forment le corps de l'automatisation. Les trois étapes suivantes sont itérées à partir du tri topologique de la sous-hiérarchie. L'intérêt est de proposer le processus d'élagage suivant comme un « calque » sur le tri topologogique des concepts formels. Cela permet à l'expert de faire varier ce calque en sélectionnant une mesure de pertinence. Nous pouvons ainsi garder en mémoire le tri topologique qui comporte toutes les unions et intersections pertinentes préalablement calculées.

3. *classer les concepts* obtenus selon une mesure de pertinence choisie. Le tableau 5.8 propose huit critères principaux selon lesquels classer les concepts,
4. *élaguer* la sous-hiérarchie afin de réduire sa taille en vue d'obtenir un diagramme de Hasse[6] représentable en relachant les restrictions de la sous-hiérarchie,
5. proposer cette hiérarchie de concepts à l'expert pour évaluation et analyse et itérer le processus pour raffiner la hiérarchie de rôles par élagages successifs.

La figure 5.9 (a) est la sous-hiérarchie obtenue à partir de la matrice du tableau 5.5. La figure 5.9 (b) est la hiérarchie obtenue après élagage en se limitant à six concepts. Le tri de pertinence est la taille de l'extension réduite. Le second tri pris en compte pour des concepts ayant même extension réduite est leur intention réduite.

[6]Un diagramme de Hasse est une représentation visuelle d'un ordre fini qui en facilite la compréhension. Hormis les annotations, les illustrations de la figure 5.9 sont de tels diagrammes.



(a) Sous-hiérarchie de Galois (b) Hiérarchie élaguée

FIG. 5.9 – Hiérarchies issues du contexte du tableau 5.5

Dans ces deux diagrammes, les six rôles potentiels sont représentés par des losanges et identifiés par les rôles du tableau 5.7. Les rôles identifiés par `X` et `Z` sont des concepts qui ne faisaient pas partie des cinq rôles initialement définis. Une note est également associée au dessous de chaque rôle potentiel indiquant :

- le nombre d'utilisateurs affectés explicitement et implicitement,
- le nombre de permissions affectées explicitement et implicitement,
- la liste des utilisateurs affectés explicitement,
- la liste des permissions affectées explicitement,

5.5.5.3 Obtention d'une relation binaire

La première étape que nous avons identifiée dans la méthode est celle de l'obtention de la relation binaire à décomposer. Pour des raisons pratiques évidentes, les droits des utilisateurs ne sont pas directement gérés avec une telle matrice : il faut dériver et croiser des informations existantes sur les privilèges pour l'obtenir.

Le cas des systèmes qui utilisent des politiques sous forme de listes, comme les *Access Control Lists* ou les *Capability Lists* est facile à traiter : chaque liste représente soit une colonne soit une ligne de la matrice des droits. Cependant, il peut exister des dépendances *implicites* entre les permissions du système. Par exemple, il existe des relations implicites dans certains systèmes de gestion de bases de données Oracle. Donner

Entrée : L : un tri topologique de la SHG
Sortie : $G' = (V', E')$: la SHG élaguée
begin
  {initialisation}
  soit M de taille $|O| \times |A|$: une matrice d'élagage
  for $c = (O_c, A_c) \in L$ **do**
    for $o \in O_c$ **do**
      for $a \in A_c$ **do**
        $M[o, a] \leftarrow M[o, a] + 1$
  {boucle principale}
  for $c = (O_c, A_c) \in L$ *du moins au plus pertinent* **do**
    for $o \in O_c$ **do**
      for $a \in A_c$ **do**
        if $M[o, a] > 1$ *(si c est supprimable)* **then**
          $M[o, a] \leftarrow M[o, a] - 1$
          $L \leftarrow L - c$
end
{tous les concepts supprimables de L ont été supprimés}
utiliser TOGSH sur L pour reconstruire le graphe $G' = (V', E')$

FIG. 5.10 – Élagage de la SHG à partir d'une liste ordonnée

le droit de *création* de table à un utilisateur lui attribue implicitement les autres droits de *manipulation de données* sur cette table. L'obtention d'une matrice nécessite donc une étape de traitement préalable dans le cas général.

5.5.5.4 Classement des concepts

Le nombre de concepts de la sous-hiérarchie peut être bien supérieur au nombre d'utilisateurs du système. Il faut donc réduire ce nombre en supprimant les concepts les moins pertinents comme rôles potentiels. Selon les organisations, l'expert peut privilégier de minimiser le *nombre de rôles* dans la hiérarchie, le nombre *d'affectations de permissions* aux rôles ou encore le nombre *d'affectations de rôles* aux utilisateurs.

Le tableau 5.8 présente huit critères utilisables pour classer les concepts selon leur pertinence. La notion de surface d'un concept, c'est-à-dire le nombre de croix qu'il couvre dans le contexte formel, est un indicateur pertinent. Une telle mesure est composée : c'est le produit de l'extension complète par l'intention complète. D'autres mesures peuvent être composées à partir de la liste de critères du tableau 5.8.

5.5.5.5 Élagage de la sous-hiérarchie de Galois

Une fois les concepts ordonnés, nous supprimons du tri topologogique de la sous-hiérarchie les concepts considérés comme les moins pertinents. Comme nous voulons que notre approche soit correcte, il faut que la hiérarchie élaguée comporte *toute* l'information contenue dans le contexte formel d'origine. Le processus d'élagage va donc introduire des affectations *multiples* dans la hiérarchie afin de réduire le nombre de rôles. Le principe de l'algorithme de la figure 5.10 est le suivant :

1. calculer une matrice outil M de taille $|O| \times |A|$, qui stocke pour chaque couple *objet* $\times$ *attribut* le nombre de concepts où il est présent,
2. pour chaque concept $c = (O_c, A_c)$, du moins pertinent au plus pertinent, le supprimer s'il n'entraîne pas de perte d'information et mettre à jour M,
3. une fois la liste des concepts réduite, utiliser l'algorithme TOGSH pour reconstruire le graphe simplifié $G' = (V', E')$,

5.5.5.6 Généricité de l'approche

Les modèles RBAC ont été étudiés et approfondis sur de nombreux aspects. C'est à notre connaissance le seul modèle où l'on a essayé d'automatisé un processus de retro-conception de politique à partir de droits existants. Pour l'application à la découverte d'une hiérarchie de rôles potentielle, la structure de sous-hiérarchie de Galois extraite avec l'algorithme PLUTON est adéquate, car :

- la notion de *concept* peut être une caractérisation formelle de la notion de rôle : une abstraction définissable par ses affectations d'utilisateurs et de permissions,
- les concepts sont des *regroupements maximaux* d'objets et d'attributs, ce qui répond à un besoin en terme d'ingénierie : les rôles doivent comporter le maximum d'utilisateurs et de permissions,
- l'analyse de concepts bénéficie d'une *caractérisation formelle*. De plus, la sous-hiérarchie lève la restriction de treillis et forme ainsi une structure utilisable sur laquelle on effectue des traitements *a posteriori*,
- PLUTON offre des performances suffisantes pour le mettre en œuvre sur des politiques considérables : dizaines de milliers d'utilisateurs et de permissions.

Notons que dans le cas général, la relation à décomposer peut être une autre relation que celle entre utilisateurs et permissions. Dans le modèle de contrôle d'accès introduit par Epstein représenté par la figure 5.8 trois utilisateurs intermédiaires ont été introduits entre rôles et permissions. En utilisant les outils de l'analyse de concepts formels, nous généralisons l'ingénierie de rôles à celle de l'identification de concept dans le cadre relationnel proposé dans la thèse.

Définition (Ingénierie de concepts hiérarchisés)**.** *Soient deux concepts $\mathcal{C}_1$ et $\mathcal{C}_2$, et R une relation existante entre ces concepts $R \subseteq \mathcal{C}_1 \times \mathcal{C}_2$. L'*ingénierie de concepts hiérarchisés *est l'activité qui consiste à identifier $R_1 \bowtie Herite_C \bowtie R_2 = R$ avec :*

– *un concept* $\mathcal{C}$,
– *une relation* $R_1 \subseteq \mathcal{C}_1 \times \mathcal{C}$,
– *une relation réflexive, transitive et antisymétrique* $Herite_C \subseteq \mathcal{C} \times \mathcal{C}$,
– *une relation* $R_2 \subseteq \mathcal{C} \times \mathcal{C}_2$.

Le problème général consiste à identifier l'ensemble $\mathcal{C}$ et les relations R_1 et R_2 mais aussi une relation hiérarchique $Herite_C$ binaire interne sur $\mathcal{C}$ qui représente la hiérarchie des concepts. La relation $Herite_C$ doit respecter les propriétés imposées aux hiérarchies.

5.6 Discussion et synthèse

Dans ce chapitre, nous avons utilisé la structuration relationnelle définie au chapitre 4 pour aborder les problèmes de conception et d'administration du contrôle d'accès. Nous divisons cette synthèse en deux parties : le résumé de notre contribution en termes d'outils pour le contrôle d'accès, puis le positionnement de ces outils dans la perspective générale de l'évolution des modèles et des politiques.

5.6.1 Outils pour le contrôle d'accès

Nous regroupons sous le terme d'*outils* l'ensemble des propositions d'aide à la conception et à l'administration faites dans ce chapitre. En effet, si le contrôle d'accès était un objet, les procédures de vérification et d'identification de concepts, les primitives de manipulation et d'interrogation seraient des outils qui permettraient de créer, modifier, utiliser et réparer cet objet.

Le tableau 5.9 synthétise les différents besoins rencontrés lors de la conception et de l'administration du contrôle d'accès et résume les réponses que nous apportons à ces problèmes. Nous avons catégorisé les besoins en deux parties : ceux qui concernent les modèles de contrôle d'accès et ceux qui concernent les politiques qui en sont instances. Nous proposons un ensemble d'outils et de méthodes qui permettent aux concepteurs et aux administrateurs d'organiser et de réaliser efficacement leurs activités. Nous différencions ainsi le travail de ces deux acteurs du contrôle d'accès :

– le *concepteur* de modèle a pour tâche de définir une organisation des droits qui soit correcte, expressive, adaptée à son client et dont les conditions d'intégrité sont définies formellement. Les outils que nous proposons au concepteur utilisent les procédures de preuve de l'*implication logique*,
– l'*administrateur* de politiques définit et modifie les droits des utilisateurs dans le modèle, et vérifie qu'ils sont corrects. De plus, si le concepteur est amené à modifier le modèle, l'administrateur doit répercuter ces changements en restructurant la politique pour la faire correspondre au nouveau modèle. Les outils que nous proposons à l'administrateur utilisent les procédures de vérification de la *satisfaction*.

	Besoin	Outil et méthode
Modèle	définition	primitive de définition de données pour le noyau et définition logique des dépendances
	vérification de l'intégrité	construire σ et prouver que $P \cup \Sigma \not\models \sigma$
	simplification	prouver que $P \cup \Sigma \backslash \{\lambda\} \models \lambda$
	représentation graphique	transformation avec Φ et Φ^{-1}
	raisonnement graphique	utiliser le chaînage de graphes pour $\models$
Politique	interrogation	primitive d'interrogation de données sur $\mathbf{I}$ et $\mathbf{I}'$
	administration	primitive de manipulation de données sur $\mathbf{I}$
	hiérarchisation de concepts	PLUTON, classement et élagage
	vérification de l'intégrité	prouver que $\mathbf{I}' \not\models \bot$
	correction	modifier $\mathbf{I}$ pour que $\mathbf{I}' \models \Sigma$
	comparaison statique	comparaison des triplets $Statique(U, A, O)$

TAB. 5.9 – Outillage pour le contrôle d'accès

On peut également imaginer une activité de retroconception de modèle, dont l'objectif serait de formaliser des mécanismes de contrôle d'accès existants pour pouvoir utiliser le panel d'outils que nous proposons. On pourrait ainsi vérifier des politiques définies dans des systèmes de gestion de base de données, des systèmes d'exploitation ou des systèmes de gestion de fichiers.

Notons que le rôle du *concepteur de modèle* est une fonction hypothétique que nous imaginons. En effet pour l'instant, son activité serait réduite au choix d'un modèle existant. Cependant la tendance vers la composition sur mesure de modèles de contrôle d'accès, comme en témoigne le modèle proposé par le GMSIH, laisse à penser que nous ne sommes pas dans la pure spéculation : déjà des consultants et des experts doivent définir et adapter des modèles de contrôle d'accès.

5.6.2 Vers l'évolution du contrôle d'accès

La diversité des activités humaines multiplie les structures et les organisations du travail, chacune s'adaptant à différents besoins pour faciliter la gestion des biens ou améliorer l'efficacité et la productivité des acteurs. Avec l'évolution des technologies comme celle des marchés et des enjeux politiques, les organisations sont amenées à être modifiées, restructurées. Afin d'être gérés efficacement, les droits d'accès doivent refléter ces évolutions des structures.

Un cas d'utilisation qui amène de profonds changements dans les organisations est celui des plans d'acquisition ou de fusion d'entreprises. Des entités indépendantes

qui avaient développé leurs propres systèmes d'information se retrouvent à coopérer étroitement. Un nouveau modèle de contrôle d'accès commun doit alors être conçu.

Ces problématiques font intervenir le concepteur de modèle qui doit développer une nouvelle organisation des droits, mais aussi l'administrateur qui va restructurer les droits existants. Lors du cas d'utilisation où l'on doit ajouter le concept de structure dans un modèle qui ne l'intègre pas, les outils de comparaison et de hiérarchisation automatisée assistent les activités de rétroconception de politiques.

La figure 5.11 présente les besoins qui surviennent lors de l'évolution des modèles. Les outils formels que nous avons proposés sont ainsi mis dans la perspective plus générale de l'évolution du contrôle d'accès. L'axe vertical de l'illustration résume les principales activités de conception et d'administration *d'une* organisation des droits. L'axe horizontal représente les changements et leurs impacts sur les droits existants. On remarque que plus un changement survient tôt dans le contrôle d'accès, plus il impacte d'activités.

Le recul porté sur la place des modèles de contrôle d'accès lors de l'évolution des structures fait surgir de nouveaux problèmes : comment découvrir des schémas globaux qui permettent d'unifier plusieurs modèles de contrôle d'accès utilisés indépendamment ? Si un schéma global est défini, comment passer d'un schéma local à un autre ? Comment vérifier des propriétés qui portent sur différents schémas locaux ? Ou encore, comment organiser les droits des administrateurs locaux dans le schéma global ?

La thèse a défini comment structurer logiquement l'organisation des droits dans un système. Les problématiques d'évolution et de coopération que nous venons d'introduire font intervenir quant à elles *plusieurs* modèles de contrôle d'accès. Si nous transposons cette problématique en termes de bases de données, on voit apparaître la notion de correspondance de schéma et d'intégration de données. Ces problèmes complexes ont été abordés dans la littérature sur les bases de données relationnelles. Nous espérons que ces résultats apporteront des éléments de réponses applicables aux problématiques d'évolution et de coopération du contrôle d'accès [Calì04, Fagin06, Fagin03, Lenzerini02]. Ces perspectives sont développées dans le chapitre 7.

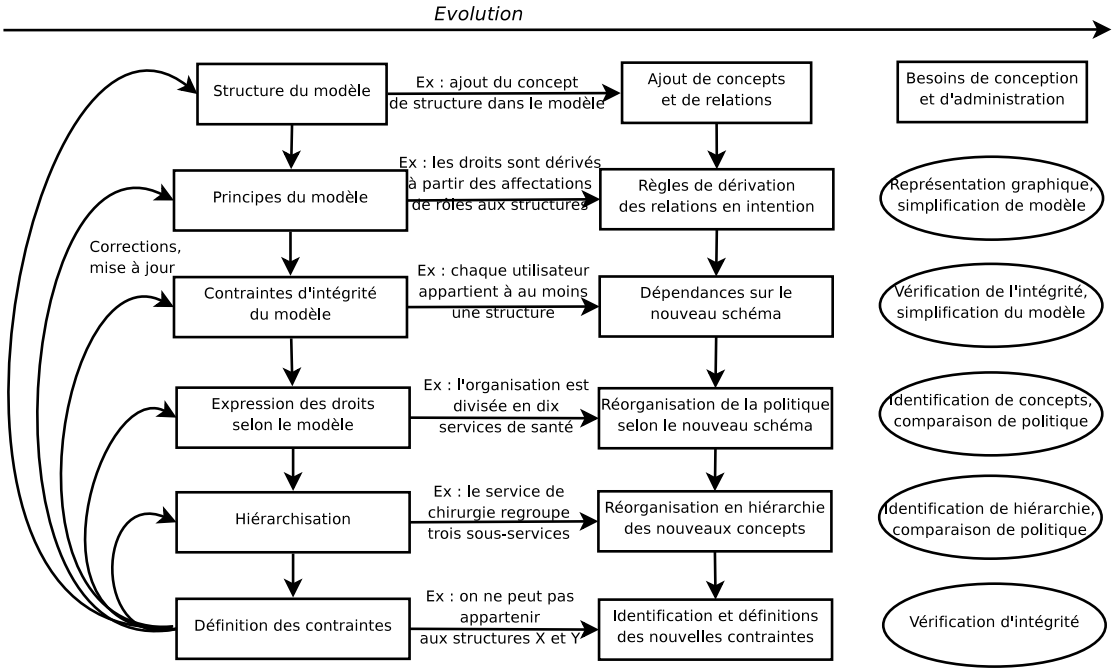


FIG. 5.11 – Évolution des modèles et des politiques de contrôle d'accès

Object-oriented programming is an exceptionally bad idea which could only have originated in California.

Edsger Dijkstra

6

Aspects techniques et réalisations

▷ *Dans le chapitre précédent, nous avons défini les problématiques qui relèvent de l'utilisation de la structuration relationnelle du contrôle d'accès de la thèse. Après avoir identifié les besoins des concepteurs et des administrateurs, nous avons proposé d'utiliser des outils algorithmiques pour l'automatisation d'une partie des activités conduites sur les modèles et les politiques.*

Dans ce chapitre, nous abordons l'implémentation de ces outils et la plaçons dans une architecture du contrôle d'accès organisée. Un des composants logiciels que nous avons identifié pour la réalisation est une bibliothèque d'outils algorithmiques pour les dépendances, elle fait l'objet de la section 6.2.

La section 6.3 présente un cas d'étude de conception du contrôle d'accès du GMSIH. *La suivante détaille l'implémentation de l'ingénierie des rôles et évalue ses performances.* ◁

Plan du chapitre

La thèse a défini une organisation générique des droits dans les systèmes ainsi qu'un ensemble d'outils de conception et d'administration du contrôle d'accès. Nos propositions ont essayé d'analyser puis de synthétiser le contrôle d'accès. Depuis le choix du cadre des dépendances au chapitre 3, jusqu'aux outils du chapitre 5, en passant par les définitions génériques du chapitre 4, nous avons fait correspondre le contrôle d'accès à une structuration relationnelle.

6.1 Architecture du contrôle d'accès

Cette section propose une architecture générique du contrôle dans les systèmes. Cette architecture est fondée sur la correspondance entre modèles de contrôle d'accès et schéma relationnel d'une part, et politiques de contrôle d'accès et instances de schéma d'autre part. Nous décrivons une architecture du contrôle d'accès en deux temps. En premier lieu nous décrivons comment utiliser les systèmes de gestion de base de données relationnelles pour stocker les politiques $\mathbf{I}$. Ensuite, nous abordons le problème du calcul de $\mathbf{I}'$ et de l'implémentation des outils algorithmiques utilisés au chapitre 5

Il ne s'agit pas de concevoir une architecture de contrôle d'accès *pour* les systèmes de gestion de base de données relationnelles, mais plutôt de concevoir une architecture *avec* ces modèles et ces outils. L'objectif est de s'affranchir des problèmes techniques relatifs au stockage des données, pour ensuite raisonner sur le contrôle d'accès. Cette approche, guidée par le critère d'*utilisabilité*, est un premier pas vers une industrialisation de la thèse.

6.1.1 Composants de l'architecture

Actuellement, les dimensions des systèmes d'information des grandes organisations deviennent gigantesques : des dizaines de milliers d'utilisateurs, d'applications et de permissions peuvent y être impliqués. Une architecture du contrôle d'accès doit être en mesure de prendre en compte des politiques de contrôle d'accès de grandes dimensions. Une telle architecture doit également s'intégrer le plus harmonieusement possible dans les systèmes d'information existants.

6.1.1.1 Objectifs

Depuis plusieurs dizaines d'années, les systèmes de gestion de base de données sont une des pierres angulaires des systèmes d'information. Grâce aux bases de données, les organisations sont capables de collecter, stocker, gérer, traiter et diffuser de très grandes quantités d'information.

S'il fallait développer une architecture de contrôle d'accès *ex nihilo*, nous devrions prendre en compte de nombreuses fonctionnalités pour parvenir à une implémentation

satisfaisante : stocker des données en s'assurant de leur validité, gérer la concurrence d'accès, garantir la disponibilité du système et offrir de bonnes performances.

Ces différents objectifs sont pris en compte dans les systèmes de gestion de bases de données. Développer une architecture de contrôle d'accès *avec* les bases de données permet de s'affranchir d'une partie de ces contraintes techniques, en les reléguant aux systèmes de gestion de bases de données eux-mêmes.

6.1.1.2 Catégorisation des aspects des modèles

Le cadre logique que nous avons proposé a défini les aspects des modèles de contrôle d'accès comme des relations mathématiques[1] entre des données. En section 4.4, nous avons catégorisé les aspects des modèles selon deux axes transversaux : la dynamicité des données et leurs définitions logiques. Une donnée d'un modèle de contrôle d'accès, c'est-à-dire une instance d'une relation, peut être :

- *statique* ou *dynamique*, selon la fréquence de mise à jour de la relation et les droits que les administrateurs ont sur cette relation,
- *factuelle* ou *déduite*, selon que l'on puisse déduire la donnée ou au contraire qu'on la définisse en extension.

En opérant ces deux divisions sur les relations des modèles, on aboutit à définir quatre fragments des politiques de contrôle d'accès : $\mathbf{I_s}$, $\mathbf{I_d}$, $\mathbf{I'_s}$ et $\mathbf{I'_d}$. L'intégrité de chacun de ces fragments est assurée par un ensemble associé de dépendances.

Les systèmes de gestion de base de données permettent de gérer les données factuelles et leur structuration. Cette catégorie comprend les sous-politiques $\mathbf{I_s}$ et $\mathbf{I_d}$ mais aussi le schéma $edb(\mathbf{AC})$ qui permet de les définir. Les systèmes de gestion permettent de s'assurer de contraintes d'intégrité sur ces données. Il sont en revanche limités à des sous-classes des dépendances fonctionnelles et d'inclusion : les clefs primaires et étrangères.

6.1.2 Systèmes de gestion de base de données

Il est important de disposer d'un cadre logique pour formaliser puis inférer sur le contrôle d'accès. Cependant, pour être utilisable, une architecture de contrôle d'accès doit disposer de mécanismes qui sortent du cadre strict de la logique. Dans le chapitre 5 nous avons défini plusieurs notions qui n'interviennent pas directement dans la modélisation logique, comme les langages de définition et de manipulation de données.

[1] Dans ce chapitre la notion de *relation* est entendue au sens mathématique du terme – c'est-à-dire comme dans l'annexe A – et non plus comme un lien entre des concepts.

6.1.2.1 Fonctionnalités offertes

C'est lors de l'utilisation des données que de nouveaux problèmes relatifs à l'exécution surviennent. La concurrence, la rapidité, la distribution ou la performance des accès n'ont pas été prises en compte dans la thèse. Deux raisons ont guidé ce choix. D'une part, ces critères auraient trop étendu le périmètre de nos travaux. D'autre part, ces problèmes ont déjà été traités dans le domaine des bases de données. Des propositions qui les résolvent – du moins partiellement – sont implémentées couramment dans les systèmes de gestion de bases de données actuels. Ainsi, le choix d'une architecture fondée sur les bases de données permet de s'appuyer sur les mécanismes offerts de :

persistance des données. C'est bien sûr une des fonctions essentielles d'un système de gestion de bases de données. Les systèmes de gestion de bases de données actuels se conforment dans leur grande majorité au standard SQL [ISO99]. Ce support généralisé permet de s'appuyer sur les nombreuses bibliothèques existantes d'accès aux données,

accès distants aux données. L'architecture client-serveur des systèmes de gestion offre des primitives d'accès distantes transparentes aux utilisateurs. Des interfaces d'administration sont développées sur ces primitives,

gestion d'accès concurrents aux données. Des mécanismes de gestion transactionnelle des accès sont offerts par les systèmes. On dispose grâce à eux de moyens d'intervenir sur les éventuels blocages qui peuvent survenir en présence d'accès multiples aux données,

manipulation des données. Le standard SQL comprend un langage de manipulation de données qui offre de très nombreuses fonctionnalités. Ce langage est nécessaire pour la réalisation des primitives d'administration des politiques de contrôle d'accès,

interrogation des données. Le standard SQL propose un langage d'interrogation de données. Non seulement ce langage est connu de beaucoup d'acteurs de l'informatique, mais en plus les implémentations sont robustes et efficaces,

maintien des données. Les mécanismes existants permettent de vérifier la satisfaction de dépendances fonctionnelles par l'intermédiaire des clefs primaires. Selon les implémentations, des restrictions plus ou moins sévères de ces classes peuvent être utilisées. De même, certaines sous-classes des dépendances d'inclusion peuvent être exprimées comme des clefs étrangères,

performance des accès aux données. Les systèmes de gestion de bases de données doivent permettre un accès efficace aux données, mais aussi aux fonctionnalités qu'ils offrent. Les mécanismes d'indexation et les implémentations optimisées des principaux algorithmes d'interrogation offrent des performances qui permettent la gestion de politiques en extension de grand volume,

disponibilité des données. Comme les systèmes d'information reposent sur les systèmes de gestion de bases de données, ces derniers doivent offrir des mécanismes

pour garantir la sécurité des données qu'ils stockent. La possibilité de dupliquer et de répartir les données ainsi que les mécanismes de redondances et de reprise sur panne permettent de renforcer l'intégrité et la disponibilité des données.

Enfin, les systèmes de gestion de bases de données proposent une fonctionnalité majeure que nous n'avons pas abordée : la possibilité de contrôler les accès aux données qu'ils stockent. En effet, les systèmes de gestion de base de données mettent en œuvre des modèles de contrôle d'accès et proposent des langages de contrôle de données. Ces langages permettent de gérer les droits des utilisateurs.

6.1.2.2 Mécanismes de contrôle d'accès existants

Dans notre approche où les politiques en extension sont des données relationnelles, ces utilisateurs sont les administrateurs. Dans l'architecture que nous proposons, les modèles de contrôle d'accès proposés par les systèmes de gestion de bases de données sont en fait les *modèles d'administration* qui gouvernent les droits des administrateurs sur les politiques.

Les systèmes de gestion de base de données divisent les permissions qu'ils proposent en deux catégories :

- les privilèges *systèmes* concernant la création, la modification et la suppression d'objets. Il s'agit des actions autorisées sur les structures de données : les éléments logiques des bases de données, mais non sur les instances elle-mêmes. C'est donc l'accès aux primitives de définition de données qui est contrôlé avec ces privilèges : droits de création, de modification et de suppression d'index, de tables, de vues, etc. Dans notre modélisation du contrôle d'accès, il s'agit du contrôle des accès aux *primitives de conception* des modèles,
- les privilèges *objets* concernant la manipulation d'objets. C'est-à-dire les actions autorisées sur les données, une fois le modèle de données défini. C'est donc l'accès aux primitives d'interrogation et de manipulation de données qui est contrôlé avec ces privilèges : droits de sélection, d'insertion, de mise à jour et de suppression de tuples. Dans notre modélisation du contrôle d'accès, il s'agit du contrôle des *primitives d'administration* des politiques.

Selon les systèmes industriels, différents modèles de contrôle d'accès sont implémentés pour gérer les privilèges systèmes et objets. Les modèles basés sur l'identité et discrétionnaires sont historiquement présents. Depuis quelques années les modèles à rôles sont mis en œuvre. Pour la plupart, ce sont des modèles basés sur RBAC_1 ou RBAC_2 sans hiérarchisation des concepts qui sont utilisés [Bertino05]. Certains systèmes aux contraintes de sécurité fortes proposent une gestion mandataire des droits comme les mécanismes *Oracle Label Security* d'Oracle, *Label-Based Access Control* de DB2, ou *Label Security* de Microsoft SQL-Server.

Les mécanismes de vues permettent de restreindre l'accès aux relations, mais sont trop statiques et rigides. Par contre, certaines extensions relativement récentes permettent un contrôle d'accès fin et dynamique sur les relations. Des mécanismes de réécriture de requêtes sont implémentés pour contrôler les accès aux colonnes et aux lignes des relations. Les *Virtual Private Databases* (VPD) introduites depuis Oracle 8i permettent ces restrictions. Ces fonctionnalités ont été étendues avec la version 10g.

Ces différents mécanismes permettent de contrôler les accès des administrateurs aux politiques, mais aussi ceux des concepteurs aux modèles.

6.1.3 Gestion des politiques

Nous allons nous intéresser à la réalisation de la base de données de contrôle d'accès que nous proposons. Les systèmes de gestion de bases de données permettent de répondre aux besoins techniques par leur performance et le nombre de fonctionnalités qu'ils supportent. En revanche, les cadres logiques qu'ils implémentent sont très réduits comparés à l'expressivité des classes comme les dépendances génératrices de tuples contraintes. Nous nous intéressons à comment définir et accéder aux instances $\mathbf{I_s}$, $\mathbf{I_d}$, $\mathbf{I'_s}$ et $\mathbf{I'_d}$ définies dans le modèle générique du contrôle d'accès.

6.1.3.1 Aspects factuels

Les systèmes de gestion de bases de données permettent de gérer les politiques en extension. La partie statique des politiques ($\mathbf{I_s}$) est complètement prise en charge par les systèmes de gestion. Les concepteurs de modèles disposent du langage de définition de données pour créer la structure des droits et les relations en extension. Les administrateurs peuvent quant à eux concevoir et administrer la politique en extension statique avec les primitives de manipulation et d'interrogation de données.

Les aspects dynamiques sont ceux sur lesquels des changements fréquents surviennent, ou sur lesquels les administrateurs de la politique ne sont pas seuls à disposer de droits de manipulation. C'est le cas des relations qui permettent de stocker des états de contexte, comme l'heure globale du système.

Les systèmes de gestion actuels supportent des accès fréquents et multiples aux données. Il est donc possible de mettre à jour fréquemment $\mathbf{I_d}$ au regard de la durée de vie des politiques. Ce sont principalement les relations sur lesquelles les utilisateurs interviennent lors de l'exécution du moniteur de références qui sont modifiées fréquemment. C'est le cas des relations portant sur le concept de sessions dans les modèles RBAC.

Certains des aspects contextuels qui nécessitent l'interrogation très fréquente de données, comme la position des usagers ou des informations sur les états d'exécution

des systèmes. Avec la prise en compte des requêtes continues dans Oracle par exemple, il peut être envisageable d'interroger de telles données [Witkowski07].

6.1.3.2 Aspects déduits

La prise en compte des données déduites ($\mathbf{I}'$) à l'aide des systèmes de gestion de base de données est plus difficile. La politique en intention est définie à l'aide du fragment de logique DATALOG ou DATALOG^C. Or, les systèmes industriels se limitent généralement au standard SQL de 1999 qui n'offre pas les possibilités de déduction de DATALOG. Il n'est pas possible d'utiliser un système SQL pour calculer $\mathbf{I}'$ tel que nous l'avons défini dans le cas général, avec récursion.

Le cadre logique utilisé dans notre proposition est plus expressif que DATALOG. Cependant, les dépendances sont divisées en classes à l'expressivité de plus en plus grande. Au fil de notre proposition, nous avons indiqué à quelle classe appartient chaque formule. On peut donc choisir de restreindre l'étendue du cadre logique en se limitant à des restrictions des dépendances. En imposant des restrictions fortes sur les formes syntaxiques autorisées qui modélisent les principes et les propriétés du contrôle d'accès, on limite l'expressivité de notre proposition, mais on s'assure de disposer de mécanismes efficaces et déjà implémentés.

Nous pouvons identifier quatre approches principales pour la réalisation de l'évaluation de requêtes sur les politiques en intention $\mathbf{I}'$:

Utiliser les requêtes et vues récursives Les implémentations récentes de SQL permettent la définition de requêtes récursives. Le mécanisme *Common Table Expression* (CTE) fourni par Microsoft SQL-Server offre cette fonctionnalité. En revanche, les requêtes récursives proposées ne sont pas aussi expressives DATALOG, car restreintes en récursions et en opérations ensemblistes sur les relations. On doit donc limiter l'expressivité des politiques au fragment offert par les CTE. Le mécanisme d'évaluation implémenté reste relativement coûteux.

Utiliser les procédures stockées Les systèmes de gestion de base de données permettent de définir des procédures qui sont exécutées avant ou après la manipulation de données. On peut grâce à eux programmer des traitements lors de la modification des données en extension pour déduire de nouveaux faits. Cette approche est délicate, car les langages de programmation de procédures stockées sont souvent spécifiques aux systèmes et, car ils peuvent manquer de fonctionnalités. De plus, nous diminuons l'intérêt de l'utilisation d'un cadre logique : il faudrait représenter les programmes P par des procédures spécifiques.

Utiliser un moteur DATALOG Pour la réalisation effective des mécanismes d'évaluation des requêtes, on peut se baser sur un moteur de bases de données déductives DATALOG ou DATALOG^C. Ces systèmes permettent de calculer le plus petit modèle $\mathbf{I}'$ d'un programme DATALOG. Ils proposent un langage de requêtes récur-

sives pour interroger la politique et des contraintes d'intégrité expressives comme les FTGD avec contraintes. Des implémentations abouties de ces moteurs existent comme DLV[2] ou XSB [3].

Développer un moteur externe Le principe est de développer un programme externe au système de gestion qui calcule le point fixe $\mathbf{I}'$ à partir de $\mathbf{I}$ et du programme P qui définit les principes du modèle. Plus le fragment de logique pour l'expression des principes est expressif, plus le développement et l'exécution du programme de calcul du point fixe sont coûteux. Une fois vérifiée, on peut éventuellement enregistrer $\mathbf{I}'_\mathbf{s}$ dans le système afin de pouvoir disposer d'un accès performant à la politique grâce à un système de cache. On définirait alors tout ou partie du schéma en intention $idb(\mathbf{AC})$ dans la base de données de contrôle d'accès. Cette approche n'est pas acceptable pour la partie dynamique des politiques déduites $\mathbf{I}'_\mathbf{d}$, car elle imposerait de les recalculer trop souvent.

Les deux premières approches n'ont pas été réalisées. La troisième a été implémentée dans un prototype basé sur un moteur existant. En revanche, la dernière a reçu toute notre attention : c'est celle que nous avons développée.

6.1.3.3 Outils algorithmiques

Avec l'utilisation des systèmes de gestion de base de données existants, nous nous sommes intéressés au stockage de $\mathbf{I}$ ainsi qu'à la construction et à l'interrogation de $\mathbf{I}'$. Si ces fonctionnalités sont incontournables pour réaliser l'architecture du contrôle d'accès, nous n'avons pas abordé l'implémentation de deux outils algorithmiques au centre de la thèse : les procédures de vérification de la satisfaction et celles de preuve de l'implication pour les classes de dépendances générales.

Procédures de satisfaction

Le premier outil algorithmique concerne la vérification des *politiques* de contrôle d'accès. Les procédures de vérification de la satisfaction d'un ensemble de dépendance par une politique $\mathbf{I}$ ou $\mathbf{I}'$ permettent de s'assurer de l'intégrité des données de contrôle d'accès. Or, seules des classes de dépendances dégénérées sont vérifiées par les moteurs de gestion des données traditionnels.

Chacune des quatre méthodes proposées pour le calcul de $\mathbf{I}'$ permet de réaliser la vérification de la satisfaction pour des classes dont l'expressivité est équivalente à DATALOG ou DATALOG^C, c'est-à-dire les dépendances génératrices de tuples totales à simple tête, éventuellement agrémentées de contraintes en hypothèse. En revanche, dans le cas général pour les TGD, les CTGD et les DCTGD sans restrictions, il faut programmer la vérification de la satisfaction. Notons que quelle que soit la classe, une telle

[2] `http://www.dbai.tuwien.ac.at/proj/dlv/`
[3] `http://xsb.sourceforge.net/`

vérification est décidable. En effet, cette vérification est réalisable naïvement en parcourant l'ensemble des données sans les modifier et sans en générer de nouvelles.

Procédures de preuve de l'implication

Le second outil algorithmique concerne la vérification des *modèles* de contrôle d'accès. C'est ce que d'aucuns appelleraient des vérifications abstraites, indépendantes des politiques. Les procédures de preuve de l'implication des dépendances ont été mises à profit pour ces vérifications. Certaines de ces procédures, ou des algorithmes qui en sont proches sont implémentés par les ateliers de conception de base de données. Ils permettent aux développeurs qui conçoivent des schémas de s'assurer de certaines propriétés de ces schémas, comme les formes normales. Or, dans la conception de base de données, c'est principalement l'inférence sur les dépendances fonctionnelles qui est mise en œuvre : un fragment qui ne permet que l'expression de modèles de contrôle d'accès simples comme RBAC_0.

Nous avons implémenté différentes procédures de preuves sur les dépendances. En réalisant ces procédures, nous nous sommes aperçus qu'elles partagent de nombreuses fonctionnalités logicielles communes, que ce soient les structures de données ou les algorithmes élémentaires. Nous nous sommes ainsi orientés vers le développement d'une bibliothèque pour les dépendances de données.

6.2 Bibliothèque pour les dépendances

Nous avons choisi de développer une bibliothèque pour la manipulation et l'inférence sur les dépendances de données. Nous nous sommes limités au fragment logique des dépendances génératrices de tuples contraintes. Cette expérience nous a cependant permis d'effectuer des vérifications non triviales, comme la vérification d'intégrité de modèle de [Benantar06], proposée en section 5.2.4. La réduction du nombre de contraintes d'intégrité de [Gavrila98], proposée en section 5.2.5 a également été effectuée à l'aide de cette bibliothèque.

La bibliothèque, nommée LIBDEPENDENCIES, consiste principalement en l'implémentation de quatre procédures de preuves pour les dépendances. Les algorithmes et les structures de données que nous avons réalisés pour ces procédures permettent également d'effectuer la vérification de la satisfaction des dépendances. Les autres réalisations que comprend la LIBDEPENDENCIES ont réutilisé les primitives que nous avons développées pour les procédures de preuves. La bibliothèque LIBDEPENDENCIES a été successivement raffinée en trois versions :

1. la première version a été implémentée en C à partir d'un prototype écrit en Python par Stéphane Coulondre. Ce prototype implémentait la première procédure de preuve de [Maher96]. Sa réimplémentation a amélioré l'efficacité globale du prototype et l'interfaçage avec les bibliothèques utilisées,

2. la seconde version a été celle du portage de la LIBDEPENDENCIES en C++ et de la compatibilité entre Linux et Windows. Le passage en programmation orientée objet a permis une meilleure réutilisation des sous-algorithmes. La *Standard Template Library* (STL) à été utilisée pour les structures de données,
3. la troisième version est celle sur laquelle ont été réalisées les expérimentations et les principaux développements de recherche. Une refonte du moteur et une nouvelle organisation du code ont été effectuées. Trois procédures de preuves supplémentaires ont été implémentées sur cette version.

Le projet de la LIBDEPENDENCIES est suffisamment conséquent, en termes de volume, du nombre d'interfaces et d'objectifs fonctionnels, pour que sa réalisation nécessite la mise en place d'un environnement de développement relativement complet allant de l'infrastructure de projet, aux choix des logiciels de développement et des bibliothèques.

6.2.1 Environnement de développement

Nous avons fait le choix technique du C++ pour réaliser la LIBDEPENDENCIES [Stroustrup97]. Cette décision conditionne l'ensemble des mesures techniques que nous entreprenons dans la réalisation de la bibliothèque. Plusieurs critères ont guidé ce choix :

- de nombreux outils de qualité sont disponibles pour ce langage ou pour le C. Une suite complète de compilation (*toolchain*) est disponible avec `gcc` et ses projets périphériques,
- la standardisation internationale et le caractère non-propriétaire du langage C++. La suite `gcc` ainsi que la grande majorité des outils que nous avons utilisés pendant la thèse sont sous licence GNU *General Public License* (GPL) ou une autre licence libre,
- la portabilité du langage entre de nombreuses architectures matérielles et logicielles offertes par la suite `gcc` et les autres outils utilisés dans le prototype,
- la vitesse d'exécution, le compilateur `gcc` pour le langage C++ offre de très bonnes performances. Le prix à payer pour l'efficacité et la versatilité du langage est une réelle difficulté à le maîtriser,
- la qualité générale et la disponibilité de bibliothèques comme la STL ou BOOST. De plus, le grand nombre d'interfaces en C/C++ nous a permis de réutiliser de nombreux composants logiciels,
- c'est un langage multiparadigme. Le mécanisme d'instanciation des *templates* permet la programmation générique ou la métaprogrammation. Ce mécanisme a été découvert comme complet au sens de Turing [Alexandrescu01] .

6.2.1.1 Environnement de programmation

Nous avons utilisé la suite logicielle `gcc`[4] pour la compilation des programmes. Cette suite propose, en plus du compilateur, des outils pour le débogage, le profilage et la couverture des tests fonctionnels. Comme interface graphique nous avons utilisé l'environnement de développement intégré KDevelop[5] qui permet la gestion de grands projets. Le générateur de documentation Doxygen[6] a été utilisé pour les manuels.

6.2.1.2 Bibliothèques utilisées

Nous avons utilisé trois grandes familles de bibliothèques pour la réalisation de la LIBDEPENDENCIES : pour l'analyse syntaxique des dépendances, pour les structures de données et les algorithmes bas niveau et pour la résolution des contraintes.

Analyse des formules logiques. Nous avons utilisé le couple Flex[7] pour l'analyse lexicale et Bison[8] pour l'analyse syntaxique des ensembles de dépendances. Tous deux permettent de générer des analyseurs C++. Marginalement, l'analyseur Xerces[9] a été utilisé pour l'analyse syntaxique de fichiers XML.

Algorithmes et structures de données. Nous avons exploité la STL qui propose des structures de données et des algorithmes génériques performants pour le C++. Cette bibliothèque a été complétée par Boost pour certaines fonctionnalités outils : interface ligne de commande, manipulation de chaînes de caractères, structure de tuple. Cette bibliothèque exhaustive propose une excellente bibliothèque de graphes Boost Graph Library (BGL) que nous avons utilisée également [Sieka01].

Résolution des contraintes. Le moteur SWI-PROLOG a été interfacé à la LIBDEPENDENCIES pour son solver de contraintes, mais pas pour ses mécanismes d'unification. Le solver est développé comme une bibliothèque à ce moteur [Christian95] Nous avons également utilisé le moteur DATALOG XSB pour des prototypes intermédiaires et des validations expérimentales mais par pour la LIBDEPENDENCIES.

6.2.2 Procédures de preuve

Les procédures de preuves forment le cœur de la LIBDEPENDENCIES. Actuellement les données commes le ensembles de formules logiques sont simplement enregistrées dans des fichiers texte. Le couple Flex/Bison analyse ces entrées et permet de les charger en mémoire dans nos structures. Les procédures de preuve et de satisfaction ex-

[4] `http://gcc.gnu.org/`
[5] `http://www.kdevelop.org`
[6] `http://www.stack.nl/~dimitri/doxygen/`
[7] `http://flex.sourceforge.net/`
[8] `http://www.gnu.org/software/bison/`
[9] `http://xerces.apache.org/`

ploitent alors ces données et produisent des traces d'exécution qui sont utilisées pour la vérification et l'analyse des inférences.

6.2.2.1 Procédures implémentées

Quatre procédures de preuves issues de la littérature sur les dépendances ont été implémentées dans la LIBDEPENDENCIES. L'intérêt est d'une part de disposer d'éléments de comparaison entre ces procédures, en termes de performance, d'expressivité et de traces d'exécution, et d'autre part d'identifier et réutiliser des algorithmes et structures communes à ces différentes procédures. Nous avons implémenté :

- le *chase* de [Beeri84] pour les TGD, une procédure de décision pour les TTGD,
- la première procédure de [Maher96] pour les CTGD quand le domaine de contraintes dispose de la propriété INC,
- la seconde procédure de [Maher96] qui est strictement plus puissante que la précédente quand domaine de contraintes ne dispose pas de la propriété INC,
- la procédure en marche arrière pour les TGD de [Coulondre03].

Nous n'avons pas implémenté la procédure proposée par [Wang02] qui prend en compte la disjonction dans la conclusion des formules logiques. Grâce à la LIBDEPENDENCIES, il est aisé d'implémenter des algorithmes comme la fermeture d'un ensemble d'attributs par des dépendances fonctionnelles, qui est une application directe de l'axiomatisation de Armstrong pour les dépendances fonctionnelles.

6.2.2.2 Structuration de la LIBDEPENDENCIES

Les procédures de preuve en marche avant [Beeri84, Maher96] comme l'évaluation des requêtes DATALOG se basent sur le calcul d'un point fixe obtenu en itérant l'application des formules, section 3.4.2.2.

Dans l'implémentation, nous avons utilisé une forme de base de données symbolique chargée entièrement en mémoire que l'on interroge et à laquelle on ajoute des tuples. Dans cette instance de données symbolique, on ne supprime jamais de tuples. Ceci est dû au fait que la négation n'est autorisée dans aucun des fragments de logique que nous avons considérés. La LIBDEPENDENCIES est composée des classes principales encapsulant respectivement :

- les structures de données qui permettent de représenter les formules logiques. Des classes permettent de représenter les contraintes dans les formules, les termes et les conjonctions de termes en hypothèse et en conclusion de dépendances (`CTgd`) et enfin des ensembles de dépendances (`CBaseTgd`). Toutes ces structures sont chargées par l'analyseur (`TGDParser`),
- les structures de données représentant la base de données symbolique et les tuples qui la composent (`CDatabase` et `CTuple`),

- les algorithmes d'unification, de calcul de point fixe et de recherche de pièces. Des classes d'environnements (`CChaseEnvironment`) permettent de regrouper les ensembles utilisés lors des inférences. Les procédures qui les utilisent sont invoquées à partir de l'interface principale de la bibliothèque,
- les structures pour mémoriser les valuations utilisées lors des inférences et restituer des traces (`CActivation`).

Cette bibliothèque a été utilisée dans le cadre de nos travaux pour la modélisation et la vérification de modèles et de politiques de contrôle d'accès. Mais les algorithmes étant génériques, la LIBDEPENDENCIES pourrait être utilisée dans une perspective plus large de conception de base de données. La mise sous forme normale faisant intervenir les dépendances multivaluées ou d'inclusion pourrait s'appuyer sur notre bibliothèque. Nous n'avons par encore poursuivi cette direction en l'état actuel, la LIBDEPENDENCIES n'est pas optimisée et son interface est à revoir.

6.3 Cas d'étude

Cette section propose deux cas d'étude qui illustrent la conception d'un modèle de contrôle d'accès puis la vérification de politique avec l'exemple de la détermination des propriétés vérifiées par une hiérarchie de rôles existante.

6.3.1 Conception du modèle du GMSIH

Le Groupement pour la Modernisation du Système d'Information Hospitalier (GMSIH) a proposé trois modèles de contrôle d'accès pour les établissements de santé [GMSIH03]. L'objectif est de faciliter la mise en place de politiques de sécurité pour la santé. En effet, dans ce domaine, les conséquences de la non-sécurité peuvent par exemple conduire à [Deswarte04] :

- une prise de décision médicale pouvant porter préjudice au patient,
- une information médicale amoindrie ou inutilisable,
- l'impossibilité d'exhiber un dossier informatisé comme preuve,
- l'utilisation illégitime de données confidentielles,
- une rupture du suivi médical.

6.3.1.1 Description du modèle

Un modèle principal est proposé, décliné en trois versions : *simplifié*, *intermédiaire* et *complet*. Il s'agit d'une aide pragmatique à l'élaboration de politique de sécurité pour la santé. Les modèles proposés par le GMSIH sont des extensions du modèle RBAC_3 adaptées aux contraintes spécifiques des milieux hospitaliers, les principaux aspects supplémentaires introduits sont :

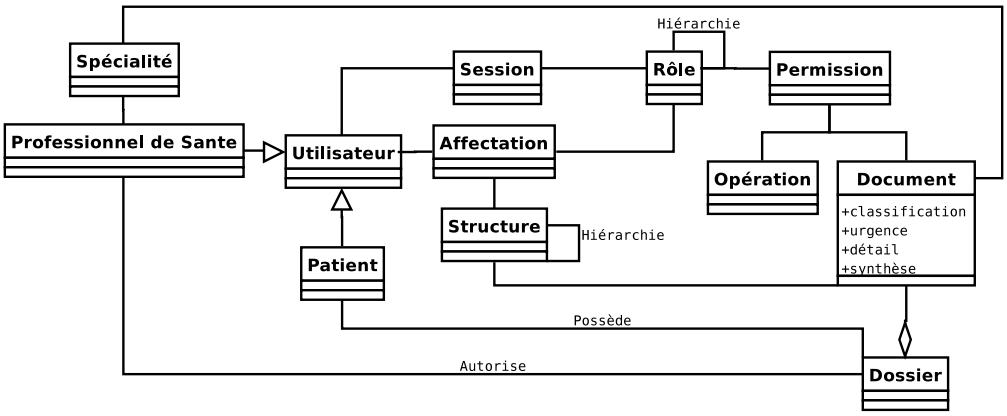


FIG. 6.1 – Modèle de contrôle d'accès du GMSIH (UML) [GMSIH03]

- le concept de *structure* et de sa hiérarchie associée qui modélise les établissements et leurs services,
- la relation d'*affectation* de rôles, qui devient une association ternaire entre *utilisateurs*, *rôles* et *structures*, ainsi un utilisateur peut disposer de plusieurs rôles dans des structures différentes,
- la spécialisation du concept d'utilisateur en *patient* et *professionnel de santé*, spécifique au domaine,
- l'ajout de concepts spécifiques au dossier médical qui sont les ressources contrôlées dans ce modèle.

La figure 6.1 illustre le modèle intermédiaire proposé par le GMSIH. Il s'agit d'un exemple de conception de modèle de contrôle d'accès, où une organisation a demandé une organisation des droits qui permette de représenter le plus fidèlement possible comment sont gérées les autorisations dans les établissements hospitaliers.

6.3.1.2 Application de l'architecture de contrôle d'accès

La région Rhône-Alpes a mis en place une plateforme d'information médicale, connectant la plupart des établissements de santé de la région. Cette plateforme est composée de trois projets principaux : le Serveur Télématique d'Identité Communautaire (STIC), la Plateforme d'Echange entre les Professionnels de Santé (PEPS) et le Dossier Patient Partagé et Réparti (DPPR).

Le dossier patient est un projet dont l'objectif est de donner l'accès à des professionnels de la santé aux informations médicales relatives à un patient et localisées dans le système d'information des établissements partenaires. Chaque système d'information hospitalier connecté envoie à la plateforme centrale une référence des informations médicales qu'il souhaite partager. Le DPPR se base sur l'identifiant régional du patient.

La structuration relationnelle du contrôle d'accès que nous avons proposée apporte une réponse cohérente aux problèmes de représentation, stockage et d'interrogation des politiques de contrôle d'accès, permettant d'évaluer les décisions de contrôle d'accès, d'exprimer des contraintes métiers complexes et de s'assurer de l'intégrité des politiques.

Exemple 6.1 Le CHM adopte le modèle du GMSIH

Avec de nombreux autres établissements de santé, le CHM est très impliqué au sein du GMSIH, il a été décidé que cet établissemt serait pilote pour la mise en œuvre du modèle intermédiaire de contrôle d'accès proposé par le groupement. Il s'agit d'un chantier qui doit être mené avec méthode dont les activités sont :

1. l'identificaton du ou des modèles de contrôle d'accès déjà utilisés,
2. la collecte et l'analyse des politiques déjà utilisées,
3. la définition de la base de données de contrôle d'accès GMSIH,
4. la définition des principes et propriétés des modèles,
5. la réalisation des mécanismes de déduction de faits en intention,
6. la définition des droits dans le nouveau modèle, à partir des droits existants.

Une fois ces étapes menées à bien on aboutit à une politique organisée selon le modèle du GMSIH. Les droits seront ensuite administrés par le service informatique une fois l'architecture mise en production. Cela nécessite l'interface avec les utilisateurs, ressources et accès offerts par les différentes application du CHM sur les ressources.

6.3.1.3 Réalisation spécifique de moteur

L'objectif du contrôle d'accès est de pouvoir déterminer les actions des sujets autorisées sur les objets du système. La dérivation du triplet *Accès* a été exprimée en logique. Cependant, pour une prise de décision efficace, et non pour la vérification, il est envisageable de développer sur mesure un algorithme capable de calculer $\mathbf{I}'$.

Les définitions logiques et l'outillage que nous proposons permettent de concevoir et de réaliser ce moteur. Il est possible qu'une organisation se limite à un fragment de logique restreint aux TTGD pour formaliser le modèle du contrôle d'accès du GMSIH. On peut même se limiter à des fragments plus restrictifs encore où les seules dépendances récursives mises en œuvre soient celles qui expriment la transitivité et la symétrie d'une relation.

Le modèle de contrôle d'accès du GMSIH comporte principalement deux hiérarchies et peu de contraintes. On peut représenter ces deux hiérarchies par des graphes. Avec l'aide d'un algorithme qui calcule la couverture transitive et réflexive de ces graphes, on peut dériver le triplet fondamental d'autorisation et prendre les décisions d'accès.

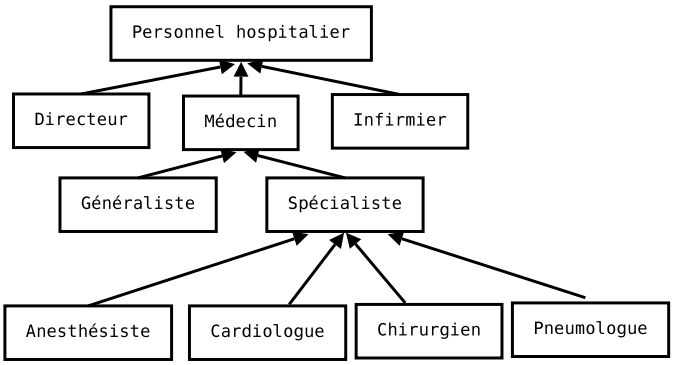


FIG. 6.2 – Exemple de hiérarchie en arbre (UML)

En représentant les graphes de domination par leurs matrices d'adjacence, la construction de la fermeture transitive avec l'algorithme de Warshall a une complexité en $O(n^3)$. Hiérarchies de structures et de rôles sont statiques. Il est ainsi bénéfique de calculer cette couverture puis de la garder en mémoire durant l'exécution du moteur. On peut ainsi savoir en temps constant si deux concepts sont reliés hiérarchiquement.

Les outils de preuve et de vérification que nous avons proposés ne sont pas tous disponibles. Ils pourront éventuellement être développés *a posteriori*, une fois l'architecture mise en œuvre. Une bibliothèque comme la LIBDEPENDENCIES que nous avons développée permet de réaliser ces vérifications en s'interfaçant avec la base de données du contrôle d'accès mise en place.

6.3.2 Détermination du type de hiérarchie

Nous allons illustrer le principe de la vérification de politiques avec la hiérarchie en arbre de la figure 6.2. Le problème à résoudre est alors « Quelle structure respecte une hiérarchie ? Un arbre, un arbre inverse, un treillis ou encore une hiérarchie générale ? ». Le tableau 6.1 donne la hiérarchie de la figure 6.2 sous forme logique :

- la colonne de gauche est un extrait de la politique **I**,
- la colonne de droite est un extrait de la politique déduite $\mathbf{I}'$.

Dans la hiérarchie de rôle d'exemple, chaque arête est une relation de domination *Domine* entre des rôles. L'objectif est de calculer la relation *Hérite* et de savoir de quel type de hiérarchie limitée il s'agit. On remarque qu'il y a 8 arêtes dans le graphe de domination et 10 concepts, l'extension de la relation *Domine* comporte donc 8 tuples. Cette relation fait partie de *edb*. La relation *Hérite* de *idb* comporte quant à elle 24 tuples :

- 8 en extension, par application de λ_H,
- 10 par réflexivité, par application de λ_R,
- 6 = 4 + 2 par transitivité, par application de λ_T,

En section 4.3.4 nous avons défini un ensemble de dépendances qui permettent d'imposer un type de hiérarchie spécifique. La satisfaction des dépendances λ_a, λ_i, ou λ_t et λ_b permet de déterminer si cette hiérarchie est limitée en arbre (λ_a), arbre inverse (λ_i) , treillis (λ_t et λ_b) ou si elle est au contraire générale. Pour cela il faut vérifier si $\mathbf{I}'$ est un modèle de l'une ou d'aucune ces dépendances. Nous notons $\mathbf{I}' \models \sigma$ pour exprimer que $\mathbf{I}'$ est un modèle de σ.

Pour cet exemple d'utilisation, les procédures de preuve comme le *chase* permettent de vérifier la satisfaction des dépendances par $\mathbf{I}'$. On exécuterait successivement l'algorithme de preuve pour déterminer que :

$\mathbf{I}' \not\models \lambda_i$ comme un spécialiste est un médecin, $Domine(\texttt{Spécialiste}, \texttt{Médecin})$, que le rôle généraliste hérite aussi de médecin, $Domine(\texttt{Généraliste}, \texttt{Médecin})$ et que ces deux rôles sont différents, il ne s'agit pas d'un arbre inverse : λ_i n'est pas satisfaite,

$\mathbf{I}' \not\models \lambda_t$ comme un spécialiste est un médecin, qu'un généraliste en est un aussi et qu'il n'existe pas de rôle $r_\top$ qui hérite à la fois de ces deux rôles, c'est-à-dire tel que $Hérite(r_\top, \texttt{Spécialiste})$ et $Hérite(r_\top, \texttt{Généraliste})$, la hiérarchie n'est pas un treillis : λ_t n'est pas satisfaite,

$\mathbf{I}' \models \lambda_a$ en revanche, $\mathbf{I}'$ satisfait bien la dépendance qui modélise que la hiérarchie est un arbre : λ_a est satisfaite.

Supposons désormais qu'un administrateur tente d'ajouter à la hiérarchie le tuple $Domine(\texttt{Spécialiste}, \texttt{Infirmier})$. Alors, la dépendance qui limite la hiérarchie à un arbre $Domine(ID, ID_1)Domine(ID, ID_2) \rightarrow ID_1 = ID_2$ est violée. En effet, on a $Domine(\texttt{Spécialiste}, \texttt{Infirmier})$ et $Domine(\texttt{Spécialiste}, \texttt{Médecin})$ avec $\texttt{Infirmier} \neq \texttt{Médecin}$.

Dans la pratique, la transaction initiée par l'administrateur demandant l'ajout du tuple sera avortée car elle viole la dépendance. Supposons maintenant qu'un autre administrateur essaie d'ajouter le tuple $Domine(\texttt{Infirmier}, \texttt{Médecin})$. Une telle opération sur la politique sera refusée, car la dépendance modélisant l'antisymétrie serait violée : $Hérite(ID_1, ID_2), Hérite(ID_2, ID_1) \rightarrow ID_1 = ID_2$.

Relation $Domine \subseteq \mathbf{I}$:

```
Domine(medecin,personnel)
Domine(directeur,personnel)
Domine(infirmier,personnel)
Domine(specialiste,medecin)
Domine(generaliste,medecin)
Domine(chirurgien,specialiste)
Domine(pneumologue,specialiste)
Domine(anesthesiste,specialiste)
Domine(cardiologue,specialiste)
```

Relation $Hérite \subseteq \mathbf{I}'$:

```
Herite(medecin,personnel)
Herite(specialiste,medecin)
Herite(generaliste,medecin)
Herite(chirurgien,specialiste)
...
Herite(specialiste,personnel)
Herite(generaliste,personnel)
Herite(chirurgien,medecin)
Herite(chirurgien,personnel)
```

TAB. 6.1 – Expression relationnelle de la hiérarchie de la figure 6.2

6.4 Réalisation de l'identification de concepts

Nous avons proposé en section 5.5 une méthode d'identification de concepts hiérarchisés basée sur l'algorithme PLUTON [Arévalo07]. Nous avons implémenté cet algorithme avec l'environnement de développement que nous avons décrit pour la LIBDEPENDENCIES. Il n'est par contre pas intégré à la bibliothèque et fonctionne indépendamment de cette dernière.

Alors que la LIBDEPENDENCIES a une portée très large et un coût de développement conséquent, l'implémentation de notre approche d'identification de rôle est bien plus ciblée et plus modeste. Par ailleurs, l'algorithme PLUTON relève plus de la fouille de données que de la preuve logique. C'est la raison pour laquelle nous l'avons développé comme un logiciel autonome.

6.4.1 Travaux connexes

Les approches ascendantes permettent d'identifier des hiérarchies de rôles implicitement présents dans des permissions existantes. La hiérarchie obtenue peut être un arbre, une forêt d'arbre ou en ordre partiel général selon la proposition. Nous décrivons ici les principales approches d'identification automatisées de hiérarchies de rôles.

6.4.1.1 ROLEMINER

Une première proposition nommée ROLEMINER est basée sur la suite d'outils IBM INTELLIGENT MINER [Kuhlmann03]. La technique utilisée est celle du *demographic clustering*, hybride entre recherche de règles d'associations et classification non supervisée (*clustering*) [Grabmeier02]. Les temps de calcul évoqués dans cette proposition sont de l'ordre de quelques heures pour environ 18.000 utilisateurs. Malheureusement, peu de détails sont donnés dans l'article et les algorithmes ne sont pas décrits, ce qui rend difficile la comparaison avec cette approche.

6.4.1.2 ORCA

La proposition des auteurs de [Schlegelmilch05] est l'algorithme de clustering hiérarchique ORCA. La technique retenue est fondée sur les forêts d'arbres comme représentation des hiérarchies de rôles. Dans le cas général cependant, la hiérarchie est un ordre partiel sur les rôles, ce n'est pas nécessairement un arbre. Ceci conduit à une limitation importante d'ORCA : une permission ne peut être associée qu'à des rôles d'un même arbre. Un logiciel commercial *getRole* a suivi ces travaux. Les auteurs n'abordent pas les performances de leur approche ni la borne maximum du nombre de rôles que l'on peut obtenir.

6.4.1.3 COMPLETEMINER et FASTMINER

Les auteurs de [Vaidya06] proposent d'utiliser l'énumération de sous-ensembles de permissions pour identifier des rôles. Les auteurs proposent deux algorithmes : une énumération complète, COMPLETEMINER, qui calcule tous les sous-ensembles de permissions possibles, et une énumération partielle FASTMINER, qui se limite au calcul d'intersections d'ensembles pris deux à deux. Les jeux d'essais proposés pour l'évaluation de leur approche sont des cas favorables à FASTMINER. L'algorithme COMPLETEMINER n'est pas évalué.

6.4.1.4 Définition de l'ingénierie des rôles

Les auteurs cette dernière proposition ont fait évolué leur approche [Vaidya07]. Plutôt que de composer un algorithme d'extraction des rôles sur mesure, comme avec FASTMINER, ils mettent en relation le problème de l'ingénierie des rôles avec celui du *pavage d'une relation binaire* [Geerts04]. La proposition ne présente pas de résultats expérimentaux. Nous pensons cependant qu'il existe des liens étroits entre leur cadre et le nôtre car ils sont fondés tout deux sur l'analyse d'une relation binaire.

6.4.2 Évaluation de la proposition

Notre implémentation semble donner des résultats pertinents et utilisables. Malheureusement, nous n'avons pas eu la possibilité de faire évaluer les résultats – c'est-à-dire des hiérarchies concepts potentiels – de notre approche par des experts. Les auteurs de [Vaidya06] ont proposé une méthodologie pour évaluer la pertinence et l'efficacité des approches ascendantes d'ingénierie de rôles. Nous l'avons utilisée pour évaluer la nôtre. Elle est plus performante que FASTMINER.

6.4.2.1 Méthodologie

Les auteurs de [Vaidya06] ont proposé une méthodologie pour évaluer la pertinence et l'efficacité des approches ascendantes d'ingénierie de rôles. Nous l'avons utilisée pour évaluer la nôtre. Cette méthodologie est basée sur les principes du *reverse-engineering*. Elle consiste à générer aléatoirement une politique RBAC, à calculer la matrice des droits d'accès à partir de cette politique, puis à vérifier si les rôles calculés à partir du contexte formel sont bien ceux initialement générés :

1. générer une politique RBAC avec ou sans hiérarchie des rôles,
2. construire la matrice des droits d'accès à partir de cette politique,
3. exécuter l'algorithme de découverte des rôles sur la matrice des droits d'accès,
4. classer les concepts calculés selon une mesure de pertinence sélectionnée,

5. parmi les n premiers concepts du classement, compter ceux qui sont effectivement des rôles générés dans la première étape. Le nombre de rôles tirés aléatoirement présents dans les n premiers identifiés donne une mesure de précision en %.

Jeu	Description	`Paramètres`
a	Nombre d'utilisateurs par rôle constant, nombre de permissions variable	`MRU=3` `NR=100, NU=2000`
b	Nombre de permissions par rôle constant, nombre de d'utilisateurs variable	`MRU=3, MPR=150` `NR=200, NP=1500`
c	Nombre de permissions constant, nombre d'utilisateurs par rôle variable	`MRU=3, MPR=150` `NP=1500`
d	Nombre d'utilisateurs, de permissions et de rôles constants, nombre de rôles par utilisateur variable	`NU=2500, NR=100` `MPR=50, NP=1500`

TAB. 6.2 – Paramètres utilisés pour les expérimentations

6.4.2.2 Résultats expérimentaux

Nous avons réalisé quatre jeux d'essai synthétiques, **a**, **b**, **c** et **d** similaires à ceux réalisés dans [Vaidya06]. Pour chaque jeu, quatre courbes sont proposées. Deux concernent la précision en % (sans hiérarchie : `P(f)`, avec : `P(h)`), les deux autres le temps d'exécution en secondes (sans hiérarchie : `T(f)`, avec : `T(h)`).

La figure 6.3 présente les résultats obtenus par notre proposition. Les expérimentations sont décrites dans le tableau 6.2. Le générateur de politique a comme paramètres le nombre de rôles `NR`, d'utilisateurs `NU`, de permissions `NP`, le nombre maximum de rôles `MRU` par utilisateurs, ainsi que le nombre maximum de permissions par rôle `MPR`. Le critère utilisé pour le classement des concepts est la taille de l'extension réduite. C'est le plus proche de celui choisi pour FASTMINER appelé *prioritization*.

Nous obtenons des mesures de précision légèrement supérieures à FASTMINER pour le jeu **a**, supérieures pour le jeu **b** et nettement supérieures pour le jeu **c**. Pour le jeu d'essai **d**, la précision décroît avec l'augmentation du nombre de rôles attribués à chaque utilisateur. Les rôles générés ne sont pas les premiers du classement des concepts, mais sont cependant majoritairement présents dans la SHG : la mesure de pertinence par taille de l'extension réduite n'est donc pas adaptée à ce jeu d'essai.

Notre approche est entre 50 et 100 fois plus rapide que les performances de FASTMINER présentées dans [Vaidya06]. Même si le choix du langage et la programmation influent sur les performances, la différence est suffisamment significative. Les résultats avec un générateur de politique hiérarchisé ou non sont assez similaires, le calcul est sensiblement plus coûteux dans le cas des hiérarchies, car le contexte formel est plus dense.

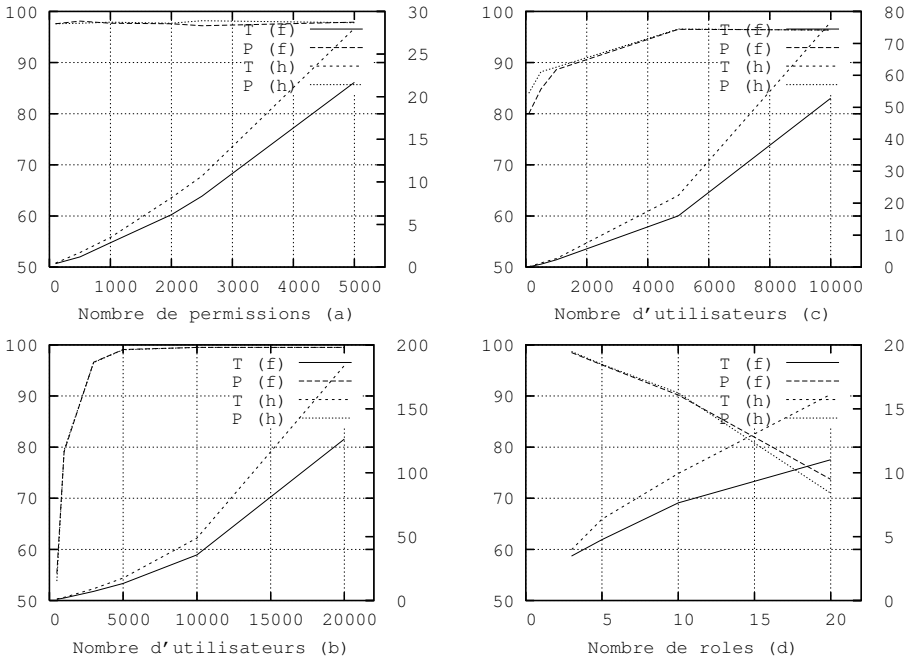


FIG. 6.3 – Résultats expérimentaux, moyenne sur dix exécutions

Nous avons également évalué les performances sur les jeux d'essai proposés par la FIMI (*Frequent Itemset Mining Implementations*), qui met à disposition plusieurs jeux d'essai volumineux pour comparer les algorithmes de data-mining. Par exemple, pour le jeu `connect` de taille 67.557 objets et 129 attributs, la sous-hiérarchie de Galois est calculée en un peu moins de 30 minutes.

6.4.2.3 Jeux d'essais réels

Nous avons eu à notre disposition deux matrices des droits issues d'une même organisation : un établissement hospitalier. L'évaluation de la pertinence des résultats obtenus sur ces jeux d'essais est relativement difficile. En effet, nous ne disposons ni de la hiérarchie de rôles *a priori* comme pour les jeux d'essais synthétiques, ni d'un expert qui puisse évaluer nos résultats.

Contrôle d'accès à une application spécifique

Ce jeu d'essai est une matrice des droits d'accès d'une des applications de l'établissement. Il concerne 37 utilisateurs et les 184 permissions spécifiques à cette application. La SHG calculée à partir de ce jeu d'essai comporte 33 concepts et 117 relations hiérarchiques. Le cas limite pour cet exemple est une politique de 8 rôles avec 2 relations d'héritage. Le temps de calcul pour ce jeu d'essai est inférieur à 0.05 seconde.

Contrôle d'accès à des applications

L'autre jeu d'essai est la matrice d'accès aux applications de l'établissement, qui indique parmi les 12 applications existantes qu'elles sont celles autorisées pour chacun des 1.582 utilisateurs de l'organisation. La SHG calculée à partir de ce jeu d'essai comporte 48 concepts et 93 relations hiérarchiques. L'élagage maximum conduit à une politique de 12 concepts sans hiérarchie : il s'agit du cas limite où chaque rôle donne exactement une permission. Le temps de calcul pour ce jeu d'essai est inférieur à 0.1 seconde.

La SHG est difficile à représenter graphiquement. Le choix d'une mesure de pertinence et du nombre de rôles minimum que l'on souhaite permet de calculer des politiques de tailles variables, comprises entre 12 et 8 rôles. Parmi les 12 applications de la matrice, deux d'entre elles ne sont utilisées que par exactement *une seule* personne chacune. Cette attribution de droit crée ainsi deux concepts propres à ces utilisateurs, qui *ne peuvent pas être supprimés*.

Pour ce jeu d'essai, un expert serait peut-être en mesure de faire une analyse croisée des concepts de la SHG entre deux aspects des rôles :
- les services de santé qui composent l'établissement,
- les fonctions des utilisateurs au sein de l'établissement,

Catégorisation des rôles obtenus

D'après les définitions de la SHG, on peut obtenir trois grandes catégories de rôles, selon qu'un concept introduise des objets, des attributs ou les deux. On peut interpréter ces trois catégories comme :

- les rôles *pertinents* : il s'agit des concepts à la fois concepts-attributs et concepts-objets. Ces rôles sont significatifs : des permissions et des utilisateurs leur sont explicitement affectés.
- les rôles *abstraits* : il s'agit des concepts-attributs : des rôles auxquels *aucun utilisateur n'est affecté directement*, les utilisateurs étant affectés soit à leur ancêtres, soit à leurs descendants dans la hiérarchie. Il est intéressant d'en garder certains pour leur *pouvoir structurant*,
- les rôles *spécifiques* à un groupe d'utilisateurs : il s'agit des concepts-objets, ces rôles sont à éviter. Il vaut mieux privilégier l'affectation multiple plutôt que d'avoir à maintenir *un rôle spécifique* à un groupe d'utilisateur qui n'apporte *aucune permission supplémentaire* que celles de ses ancêtres.

6.5 Synthèse

La thèse propose une structuration relationnelle du contrôle d'accès construite sur les fondements des bases de données. Ce chapitre a proposé une architecture de contrôle d'accès qui exploite la correspondance entre bases de données relationnelles et contrôle d'accès. En utilisant les primitives existantes dans les systèmes de gestion de base de données relationnelles, nous nous affranchissons de contraintes techniques et nous rapprochons les politiques de contrôle d'accès des données métiers. Grâce à cette proximité, on pourra utiliser de façon homogène les primitives de gestion des données.

Un cas d'étude orienté vers les établissements de santé a été proposé pour illustrer le cœur de la thèse sur un cas relativement concret. Le modèle du GMSIH est une réalité, mais les difficultés de vérification et d'administration d'un tel modèle le sont aussi. Le cadre logique pour le contrôle d'accès que la thèse propose permettra peut-être d'y répondre.

Nous avons également décrit la principale réalisation logicielle de la thèse, une bibliothèque LIBDEPENDENCIES d'outils algorithmiques pour les dépendances. C'est un outil logiciel qui nous permet de développer les outils proposés au chapitre 5. Cette réalisation nous a permis de valider expérimentalement notre approche. Notons que les procédures de preuves pour les dépendances génératrices de tuples contraintes n'avaient pas été implémentées jusqu'à présent.

Enfin, nous avons donné les principaux résultats de notre approche d'identification de concepts hiérarchisés présentée en section 5.5. À notre avis, il s'agit de l'outil le plus rapidement industrialisable proposé dans la thèse. Non pas que la LIBDEPENDENCIES soit une gageure, mais la réalisation efficace et utilisable en production d'un tel outil dépasse largement les moyens techniques et humains dont nous disposons.

Nous avons passé sous silence la réalisation de la représentation graphique de modèles présentée en section 5.4. Nous avons réalisé un prototype de cette approche basée sur la bibliothèque *Conceptual Graphs Integrated Tools allowing Nested Typed graphs* (COGITANT[10]). Nous avons ainsi effectué des vérifications basées sur la modélisation en graphes conceptuels des modèles et des politiques de contrôle d'accès. Malheureusement, comme la bibliothèque ne propose pas d'interface graphique, c'est une trace écrite qui résulte des inférences. Nous avons ainsi privilégié la LIBDEPENDENCIES pour nos réalisations.

Que ce soit sur les aspects techniques de ce chapitre, sur d'autres, plus fondamentaux, ou encore sur la coopération de politiques structurées avec des modèles différents, il reste de très nombreuses activités à entreprendre. Le dernier chapitre qui suit résume la thèse et ouvre ses principales perspectives.

[10] `http://cogitant.sourceforge.net/`

The question of whether machines can think... is about as relevant as the question of whether submarines can swim.

Edsger Dijkstra – "The threats to computing science"

7

Conclusion et perspectives

▷ *Ce dernier chapitre conclut la thèse. Il synthétise notre approche ainsi que nos principales contributions. Nous présentons cette synthèse comme une organisation du contrôle d'accès bâtie sur deux propositions fondamentales :*

1. *les modèles de contrôle d'accès sont composés d'une structure, de principes et de propriétés,*
2. *la structure d'un modèle est définissable en termes de concepts et de relations entre ces concepts.*

Ensuite, nous discutons les principaux choix de modélisation que nous avons effectués. Cette discussion conduit à des perspectives organisées selon cinq axes principaux :

- *les extensions et les restrictions du cadre logique,*
- *l'extension du périmètre de la thèse,*
- *la prise en compte de problématiques propres à l'administration,*
- *l'application du contrôle d'accès et son industrialisation,*
- *l'application de la proposition aux politiques de sécurité du réseau.*

Certaines de ces perspectives font déjà l'objet de nos travaux actuels. ◁

Plan du chapitre

7.1 Synthèse générale

DEPUIS ses débuts il y a plus de trente ans, le contrôle d'accès a évolué, il s'est enrichi. En tant que moyen à mettre en œuvre pour garantir la sécurité, le contrôle d'accès a été, dès son plus jeune âge, analysé, formalisé, vérifié. Ce n'est qu'une fois adopté, que le contrôle d'accès a tâché de représenter, le plus fidèlement possible, la réalité de l'organisation des systèmes qui l'utilisent.

Si depuis plus d'une dizaine d'années, nous avons assisté à une profusion de modèles de contrôles d'accès pratiques et théoriques, il n'en reste pas moins que les modèles de contrôle d'accès partagent des fondamentaux communs. Ce que nous avons nommé le *triplet fondamental d'autorisation* en est un. L'objectif de tout contrôle d'accès est de permettre de décider si une action demandée est légitime ou non. Cette décision est prise au regard d'un ensemble d'assertions structurées : une politique organisée selon un modèle.

La notion de *modèle* de contrôle d'accès n'est d'ailleurs apparue que tardivement, avec la multiplication des systèmes informatisés et des enjeux de leur sécurité. Les critères désirés pour les modèles ont ainsi évolué. Loin des organisations élémentaires et militaires, le contrôle d'accès est désormais composé de structures complexes.

Nous ne pensons, ni ne désirons, détenir une quelconque vérité sur le contrôle d'accès. En revanche, nous avons essayé de bâtir une approche cohérente, logique et utilisable de l'organisation des droits. Ce travail a été guidé par une vision personnelle de ce qu'est une politique de contrôle d'accès : un ensemble de données structurées. La thèse propose ainsi une construction du contrôle d'accès, fondée sur deux assertions :

1. les modèles de contrôle d'accès sont composés :
   - d'une structure qui permet d'organiser des informations,
   - de principes qui permettent de dériver de nouvelles informations à partir de celles connues, en vue de prendre les décisions d'accès,
   - de propriétés que la structure et les informations doivent respecter,
2. la structure d'un modèle de contrôle d'accès est définissable en termes de concepts et de relations entre ces concepts.

Ces deux assertions traduisent la volonté de composer une organisation des droits qui soit la plus générale possible, mais aussi qui permette d'assurer sa vérifiabilité. Nous les avons formulées avec l'objectif de s'appuyer sur un existant solide : les fondements des bases de données. Ainsi, une formulation équivalente de la seconde affirmation est qu'un schéma relationnel permet de représenter la structure d'un modèle de contrôle d'accès. À l'aide du cadre logique des bases de données, nous avons successivement défini à partir des travaux existants :

- politique factuelle : une instance du schéma relationnel,
- principes : un ensemble de formules qui définit une politique déduite unique,
- propriétés : un ensemble de formules qui garantit l'intégrité des politiques.

En faisant correspondre au long de la thèse le contrôle d'accès au modèle relationnel, nous avons pu transposer et résoudre les problématiques auxquelles nous nous sommes intéressés. Nous avons ainsi mis à profit les outils issus des bases de données pour définir :

- la conception de modèle,
- la vérification de la non-intégrité de modèle,
- la simplification de modèle,
- la représentation graphique de modèle,
- la conception et la rétroconception de politique,
- l'administration de politique,
- la catégorisation statique et dynamique dans les modèles,
- la vérification de l'intégrité de politique,
- la comparaison statique de politiques.

Ces différentes définitions ont pour but de positionner les différentes activités du contrôle d'accès, mais aussi de permettre l'utilisation d'outils génériques. Nous avons mis en œuvre deux notions principales pour répondre à ces problématiques : la satisfaction et l'implication. Les deux principaux problèmes théoriques sous-jacents sont :

- le problème de la satisfaction, ou de la résolution de requêtes. Il s'agit d'un problème relatif à une instance donnée. Il s'agit de définir un algorithme qui est capable de décider si oui ou non une théorie est satisfaite par cette instance ou de construire une nouvelle instance qui la satisfait.
- le problème de l'implication logique, ou de l'équivalence de requêtes. Il s'agit d'un problème posé sans disposer d'une instance donnée. Il s'agit de définir un algorithme qui est capable de décider si oui ou non toute instance modèle d'une théorie est aussi modèle d'une formule close.

La décidabilité de ces deux problèmes a guidé le choix des fragments de la logique du premier ordre utilisés dans notre proposition. Avec le cadre modulaire des dépendances, qui propose des classes à l'expressivité et à la décidabilité contrôlée, nous pouvons choisir soit d'adapter le cadre logique au contrôle d'accès, ou au contraire d'adapter le contrôle d'accès à l'expressivité fournie par un fragment donné. Nous pouvons ainsi composer un contrôle d'accès sur mesure, en faisant varier les restrictions sur les formules autorisées pour la modélisation.

Informellement, si l'on disposait d'un hypothétique curseur qui représente le choix du fragment logique, le positionner à l'extrémité « contrôle d'accès » produirait un modèle d'organisation des droits infiniment souple, mais sur lequel il serait impossible de prendre une décision. Cet extrême pourrait être une logique d'ordre supérieur.

À l'inverse, positionner l'hypothétique curseur sur l'extrémité « propriétés théoriques » conduirait à un modèle de contrôle d'accès parfaitement décidable, mais d'une rigidité telle qu'il serait impossible de représenter quelque aspect structurel des organisations. On imagine un fragment restrictif comme les dépendances fonctionnelles.

Cet hypothétique curseur caricature grossièrement les deux axes de recherches que nous avons identifiés dans l'état de l'art. Un objectif ambitieux est de trouver la position limite de ce curseur, où l'expressivité est suffisante pour répondre à de nombreux besoins de modélisation, mais où les propriétés de décidabilité sont garanties.

7.2 Discussion générale

Cette section discute le choix du cadre logique, mais aussi les fondements de notre proposition. Nous avons essayé de donner des définitions cohérentes du contrôle d'accès avec le cadre relationnel et les dépendances. Nous avons également tâché de rendre la proposition la plus utilisable possible. Pour ces deux critères, nous avons dû faire des choix de modélisation. Cette discussion pose le cadre général des perspectives que nous détaillons ensuite.

7.2.1 Restrictions sur les fragments de logique

Nous avons bâti la thèse sur deux propositions fondamentales, qui résument les traits communs que nous avons identifiés dans les modèles de contrôle d'accès. Nous avons essayé d'embrasser des notions générales avec le cadre relationnel. Nos propositions imposent cependant des restrictions sévères sur la modélisation et l'utilisation du contrôle d'accès :

- l'impossibilité de pouvoir utiliser la négation dans l'expression des principes et des propriétés. Comme nous souhaitons utiliser les résultats des fragments de la logique du premier ordre des dépendances de données, nous n'avons pas mis en œuvre des modèles DATALOG ni même des langages de requêtes non récursifs autorisant la négation,
- nous nous sommes basés sur l'existence d'une unique interprétation calculable $\mathbf{I}'$ qui soit un modèle des principes. Cependant, cette restriction n'a pas été imposée aux fragments utilisés pour les propriétés.

Ces restrictions peuvent être relâchées, mais cela impacterait fortement la thèse. En effet, des fragments de logique autorisant la négation dont il existe une interprétation unique calculable ont été proposés. La stratification des règles de déduction ou la notion de bipolarité par exemple permettent ces garanties [Abiteboul95, Halpern03].

En introduisant la négation, on ne pourrait plus utiliser les procédures pour les dépendances de façon générique, en considérant les règles de déduction comme des dépendances. Il serait alors possible d'utiliser les procédures de preuves pour un fragment avec négation de façon générique, mais en réduisant les classes de dépendances utilisables à celles autorisées par ces procédures.

Plutôt que la négation, nous avons fait le choix de privilégier la possibilité de quantifier existentiellement des variables de la conclusion et d'exprimer des contraintes sur

ces variables. Cela nous a permis une formulation homogène de plusieurs notions et l'expression de contraintes d'intégrité sur les politiques. Informellement, nous pourrions dire que nous avons choisi de privilégier l'expressivité des propriétés des modèles, plutôt que l'expressivité des principes et la possibilité d'exprimer des politiques hybrides.

Dans le cas général, il peut être difficile de formaliser en logique du premier ordre un modèle complet. Une sévère restriction est que le modèle soit formalisable sans symboles de fonctions, ce qui peut rendre difficile, voire impossible, l'expression de certaines propriétés. De plus, nous n'avons pas introduit la notion de temps dans la modélisation, sauf sous forme de contraintes sur des relations dynamiques évaluées lors de l'exécution du moniteur.

7.2.2 Définitions du contrôle d'accès

Nous avons défini une modélisation de ce qu'est le contrôle d'accès à partir des travaux initiaux de Lampson pour les définitions fondamentales et sur les modèles RBAC pour les principaux développements relatifs aux modèles structurés. D'aucuns peuvent remettre en question les éléments que nous avons considérés comme fondamentaux ou comme communs aux modèles de contrôle d'accès.

Nous avons en particulier passé sous silence certaines des propositions du contrôle d'accès. Il s'agissait principalement de travaux fondés sur des principes généraux, comme la notion de conflit d'intérêts. Ce principe consiste à partitionner les états du système de contrôle d'accès et de définir des règles qui interdisent le passage d'un état à un autre. Considérons un exemple du domaine bancaire.

On pose les banques d'une part, et les institutions de vérification des transactions bancaires et financières d'autre part comme étant en conflit d'intérêts. Un contrôle d'accès comme le modèle « Chinese Wall Policy » permet d'imposer qu'une fois un agent entré dans une ou l'autre de ces parties, il soit impossible d'en changer [Brewer89].

Cela permet de prévenir d'éventuelles actions délictueuses où un ancien agent des institutions de vérifications deviendrait employé d'une banque et utiliserait ses anciens contacts, accès ou tout simplement sa connaissance du système pour outrepasser ses privilèges. Ou éventuellement dans l'autre sens, un employé d'une banque qui devienne chargé de vérifier son ancien employeur. Nous n'avons pas introduit la notion de conflit d'intérêts, car elle relève souvent des politiques de sécurité administratives plus que des politiques de contrôle d'accès logique.

On peut également critiquer la définition de l'administration que nous avons donnée. Nous avons restreint l'activité des administrateurs à la manipulation d'une politique en extension. Il s'agissait pour nous de diviser les activités relevant des modèles de celles relevant des politiques. Or on pourrait autoriser dans l'administration la possibilité de définir des principes ou des propriétés, plutôt que de figer une structuration.

On peut imaginer, dans le cas d'une politique où il n'y a que peu d'exclusion mutuelle, qu'on supprime la relation d'exclusion du modèle et qu'on ajoute, au cas par cas, des dépendances spécifiques sans mettre en œuvre une relation d'exclusion munie de propriétés. Ces dépendances utiliseraient des constantes dans les termes.

Nous avons choisi de ne pas autoriser cette possibilité et d'imposer aux administrateurs de n'agir que sur des faits sans modifier la théorie logique du modèle. Nous souhaitons clairement séparer ce qui est valable pour toute politique de ce qui est spécifique. Nous avons opéré cette distinction par la définition de modèle et de celle de politique, ce qui nous a permis, ensuite, de définir l'administration.

7.2.3 Organisation des perspectives

Les discussions de cette section vont être reprises plus spécifiquement comme des perspectives dont nous donnons les principales directions de recherche. Les trois premiers axes de perspectives présentés concernent directement cette discussion et abordent le choix d'un cadre logique, l'extension du périmètre des modèles de contrôle d'accès et l'organisation des droits des administrateurs sur les politiques.

Nous abordons ensuite des perspectives plus techniques relatives aux domaines d'application et à la LIBDEPENDENCIES. Enfin, la dernière perspective propose une réutilisation de la structuration relationnelle proposée dans thèse aux problèmes des pare-feux. Il s'agit de directions assez générales de recherches qui font l'objet de plusieurs sous-perspectives que nous décrivons dans la section suivante.

Les travaux que nous avons présentés en section 5.4 pour la représentation de modèle et l'ingénierie de rôles en section 5.5 sont des « perspectives partiellement réalisées » qui servent de base pour de futurs développements.

7.3 Aspects logiques du contrôle d'accès

Les quatre perspectives suivantes concernent le choix d'un cadre logique pour l'étude du contrôle d'accès et son application pour une redéfinition logique de certaines notions qui souffrent de lacunes de caractérisation formelle.

7.3.1 Restrictions décidables

Il pourrait être intéressant du point de vue théorique, mais aussi pratique, de définir des restrictions décidables de certaines classes de dépendances. L'introduction du quantificateur existentiel dans les expressions est en particulier une source d'indécidabilité. Empiriquement, nous n'avons pas rencontré de modélisations naïves qui font boucler sans fin les procédures de preuves pour les dépendances.

Ceci peut être dû au fait que nous n'avons pas utilisé de nombreuses variables quantifiées existentiellement, et que les dépendances qui en comportent sont généralement assez restrictives. Cette constatation expérimentale est peut-être formalisable et pourrait conduire à la définition d'une classe de dépendance intermédiaire entre TTGD (sans quantificateur existentiel) et TGD (avec quantificateur existentiel) dont la décidabilité serait garantie.

Les liens existants entre les règles de graphes conceptuels et les dépendances pourraient être utiles dans ce sens. En effet, en s'intéressant aux relations récursivement définies, il a été prouvé que l'on peut identifier des ilots de décidabilité en analysant les règles de graphes [Baget06]. Ce résultat pourrait être transposable sous forme de restriction sur les dépendances génératrices de tuples, où l'on limiterait l'utilisation du quantificateur existentiel à des strates, comme cela a été fait pour la négation.

7.3.2 Dépendances disjonctives

En section 3.4.1.2 nous avons présenté les principales classes de dépendances existantes. Nous avons introduit les dépendances génératrices de tuples contraintes disjonctives (DCTGD). Elles ont été proposées par Junhu Wang dans sa thèse sous la direction de Michael Maher [Wang02]. Plusieurs procédures relatives à cette classe de dépendances ont été développées. Il a été montré qu'elles sont strictement plus puissantes que celle de [Maher96].

La thèse de Wang présente de nombreuses applications des dépendances génératrices de tuples contraintes avec ou sans disjonctions [Wang05]. Nous n'avons pas utilisé le fragment disjonctif dans notre thèse. Il faut évaluer l'intérêt de les mettre en œuvre pour le contrôle d'accès. Un des problèmes qui surviennent est que, si on considère un fragment disjonctif de DATALOG sans contraintes en conclusion, on peut rencontrer plusieurs points fixes. En revanche, on pourrait seulement utiliser la disjonction pour l'expression des propriétés et non des principes. On pourrait par exemple définir des prérequis plus complexes que ceux présentés, comme « s'il existe un utilisateur qui dispose du rôle $\mathtt{r}$, alors il existe un utilisateur qui dispose soit du rôle $\mathtt{r}_1$, soit du rôle $\mathtt{r}_2$ ».

7.3.3 Sémantique formelle de concept

Nous avons repris la formalisation de RBAC de [Ferraiolo03b]. Or il a été remarqué qu'elle est ambiguë, voire erronée [Li07]. La relation d'héritage peut servir à regrouper des utilisateurs, des permissions ou les deux.

> ***Suggestion 5*** *The semantics of role inheritance should be clearly specified and discussed.*

L'étude et la mise en perspective entre contrôle d'accès et analyse de concepts formels en section 5.5 nous a conduits à identifier des liens entre les utilisateurs autorisés d'un rôle et ses permissions autorisées. Nous avons principalement considéré les rôles comme des regroupements d'utilisateurs *et* de permissions. Les résultats de l'analyse de concepts formels nous permettraient de redéfinir la notion de hiérarchie de rôles comme la suivante :

Définition (Sémantique de la relation d'héritage (tentative))**.** *$r_1 \succeq r_2$ si et seulement si toutes les permissions autorisées à r_2 sont autorisées à r_1 et tous les utilisateurs autorisés de r_1 sont autorisés de r_2 :*

$$r_1 \succeq r_2 \Leftrightarrow auth\_perms(r_2) \subseteq auth\_perms(r_1) \wedge \\ auth\_users(r_1) \subseteq auth\_users(r_2)$$

Une telle définition, qui exploite la correspondance de Galois entre les applications α et ω s'inscrit dans le cadre plus large de la formalisation des modèles de contrôle d'accès, pour définir la sémantique d'un concept d'un modèle. On pourrait également envisager de définir formellement la notion de rôle proposée dans le langage XACML utilisé comme véhicule de politiques de contrôle d'accès.

7.3.4 Intégrité d'un modèle de contrôle d'accès

Nous avons proposé une définition générale de la *non-intégrité* d'un modèle de contrôle d'accès en section 5.2.4. Elle nous a servi de base pour des vérifications élémentaires, mais elle ne garantit pas qu'un modèle soit intègre pour autant.

Nous avons modélisé le contrôle d'accès par une structure relationnelle $\mathbf{AC} = (sch, P, \Sigma)$. Mais nous avons fait une distinction entre $\mathbf{I}$ et $\mathbf{I}'$ vis-à-vis des droits des administrateurs et imposé que l'on puisse dériver les triplets d'autorisation $Accès(S, A, O)$, $Statique(U, A, O)$ et $Dynamique(U, A, O)$. Nous avons également imposé que les relations de *sch* soient bien fondées à l'aide de dépendances structurelles. Une définition formelle de l'intégrité d'un modèle devrait donc se baser sur ces hypothèses. Nous pourrions proposer la définition suivante :

Définition (Modèle intègre de contrôle d'accès (tentative))**.** *Un modèle de contrôle d'accès $\mathbf{AC} = (edb \cup idb, P, \Sigma)$ de théorie $T = P \cup \Sigma$ est dit* intègre *si et seulement si T est satisfaisable, c'est-à-dire s'il existe une politique $\mathbf{I}'$ qui est un modèlde de T. Avec de plus $\mathbf{I}'$ « minimale » telle que chaque relation et chaque concept de edb contient au moins un tuple et que chaque relation de $\{Accès, Statique, Dynamique\} \subseteq idb$ contient aussi un tuple.*

L'objectif de cette définition est d'indiquer qu'un modèle n'est pas intègre si les concepteurs ont défini des relations qui ne peuvent pas être utilisées par les administrateurs pour dériver ou contraindre des autorisations.

Il serait intéressant de construire une expression σ qui modélise cette définition de l'intégrité pour ensuite vérifier si $T \models \sigma$ ou $T \not\models \sigma$. Cette construction pourrait s'exprimer au second ordre en quantifiant les relations de *edb*. Il faudrait par exemple définir que l'hypothèse de σ soit *la plus petite possible* pour représenter un critère de « minimalité » de $\mathbf{I}'$. Il s'agirait ainsi de définir une formule de la forme :

$$\forall \mathbf{R} \in edb \ldots \phi(\mathbf{R}) \rightarrow Statique(U, A, O), Dynamique(U, A, O), Accès(S, A, O)$$

Dans le cas général, une ou plusieurs expressions du premier ordre pourraient être vérifiées à partir de cette construction. Or une telle vérification n'est pas décidable pour certaines classes de dépendances. De plus, il n'est pas garanti que σ soit réellement constructible ou même exprimable au premier ordre.

Nous pourrions nous appuyer sur les dépendances du second ordre proposées dans [Fagin06] et sur les résultats de décidabilité des théories logiques [Abiteboul95, Dehornoy07] pour parvenir à une définition de l'intégrité d'un modèle de contrôle d'accès qui exprimerait notre intuition. Cet objectif est difficile du point de vue théorique.

7.4 Extension du périmètre du contrôle d'accès

Cette section présente des perspectives d'application des résultats théoriques issus des bases de données et plus précisément des dépendances. En effet, depuis quelques années plusieurs résultats exploitant le *chase* de Beeri et Vardi [Beeri84] ont été proposés, pour la correction et l'intégration de bases de données notamment.

7.4.1 Automatisation de correction et mode dégradé

Lorsqu'une politique n'est pas consistante, il faut déterminer quelles sont les données à ajouter, modifier ou supprimer pour la rendre intègre. Dans le cas des dépendances génératrices de tuples ou de contraintes, la correction principale est la suppression de données sources d'erreurs, c'est-à-dire qu'il faut trouver un sous-ensemble de $\mathbf{I}'$, qui soit un modèle de Σ.

Ce problème a été étudié dans le cadre général des bases de données,des résultats de complexité ayant été établis [Chomicki05, Bertossi06]. Un autre problème pris en compte est celui du *Consistent Query Answering*, c'est-à-dire, sachant que la base de données ne satisfait pas un ensemble de dépendances Σ, quelles sont les résultats des requêtes qui sont sûrs, c'est-à-dire l'ensemble des tuples de $\mathbf{I}'$ qui respectent Σ et qui font partie des réponses des requêtes. Ces résultats permettraient peut-être de compléter la correction de politique non intègre présentée en section 5.3.4.

7.4.2 Coopération de modèles hétérogènes

Les dépendances de données ont été étudiées dans le cadre de la correspondance de schémas [Calì04, Fagin05, Fagin06]. L'objectif de ces recherches est de définir et d'identifier les propriétés de dépendances génératrices de tuples permettant de faire correspondre un schéma à un autre.

Plusieurs problématiques sont relatives à cette réflexio, comme part l'existence et la définition d'un schéma d'intégration. Ce schéma peut être défini comme une vue globale sur des sources de données (*Global-As-View*). À l'inverse, on peut définir les sources comme des vues sur le schéma global (*Local-As-View*) selon la stratégie d'intégration et de coopération [Calì04, Lenzerini02].

Un problème connexe est celui de l'inversion de correspondance. Connaissant les correspondances de deux sources vers un même schéma global, est-il possible d'identifier des correspondances directes entre les sources ? Ce problème est délicat et a été partiellement résolu à l'aide de dépendances du second ordre [Fagin06].

Cet ensemble de travaux pourrait être utilisé sur la structure relationnelle du contrôle d'accès que nous proposons dans la thèse. En effet, la coopération de politiques hétérogènes est un problème de correspondance de schémas de contrôle d'accès. Les travaux existants et les deux paradigmes *Global-As-View* et *Local-As-View* apporteraient peut-être des réponses aux problèmes de la définition d'un modèle de contrôle d'accès commun.

Sur cette base, les travaux de Ronald Fagin permettraient alors de définir des liens de collaborations directs entre politiques hétérogènes, pourvu que l'on connaisse un schéma commun et des dépendances exprimant les correspondances entre ce schéma et les sources. On pourrait imaginer que le schéma global commun soit la matrice de Lampson exprimée comme une vue sur des modèles structurés, ou bien l'inverse. En revanche, il faut évaluer la possibilité de définir des correspondances sur des relations définies en intention dans chaque source et sélectionner une classe de dépendance réduite sur ces schémas.

7.5 Administration du contrôle d'accès

Dans la thèse, nous avons défini l'administration du contrôle d'accès vis-à-vis de la structuration relationnelle proposée. Cette section propose des perspectives de nouveaux outils pour les administrateurs, mais aussi sur l'organisation de leurs droits dans notre approche.

7.5.1 Modèle d'administration

Selon le point de vue que nous adoptons dans la thèse, les administrateurs sont des utilisateurs du système de contrôle d'accès : ils manipulent et interrogent les politiques, qui peuvent être stockées dans un système de gestion de bases de données relationnelles. Organiser les droits des administrateurs est un enjeu considérable en présence d'un très grand nombre d'utilisateurs. Il faut donc structurer les droits d'utilisation des primitives d'administration, c'est le rôle des *modèles d'administration* dont nous proposons une définition.

Définition (Modèle d'administration)**.** *Un* modèle d'administration *d'un modèle de contrôle d'accès – dit modèle de base – est une organisation des droits d'utilisation des* primitives d'administration *du modèle de base. Le modèle d'administration et le modèle de base peuvent être structurés différemment.*

Cette définition introduit une forme de récursion dans la notion d'administration. En effet, l'ensemble des notions, de la formalisation, des problèmes et des réponses que nous proposons dans cette thèse se retrouve dans les modèles d'administration... Il ne s'agit pas d'un écueil de notre approche, mais plutôt de sa mise en abyme. On pourrait envisager naïvement de « réitérer » la proposition de la thèse pour les modèles d'administration. Mais de nouveaux problèmes inédits surviennent lors de la définition des modèles d'administration.

Dans le cas des modèles à rôles, il est recommandé de limiter les permissions des administrateurs sur la hiérarchie de rôles à des sous-ensembles de la hiérarchie. On définit ainsi des zones d'administratives afin d'éviter que les privilèges des administrateurs ne se recouvrent [Crampton03a]. Pour appliquer ce principe à notre structuration, nous pourrions créer des connexions entre les opérations administratives et une définition logique de ces sous-ordres. On définirait ainsi des dépendances entre le modèle de base et le modèle d'administration. Les travaux sur la correspondance de schémas pourraient nous apporter des éléments de réponse.

Nous avons proposé une catégorisation des aspects des modèles de contrôle d'accès en statique et dynamique relative aux droits des administrateurs sur les relations des modèles. Or, le principe des modèles d'administration comme SARBAC est de s'attacher à la hiérarchie de rôles qui est plus stable que les affectations de rôles aux utilisateurs [Crampton03a]. Pour faciliter la définition de modèles d'administration dans notre structuration relationnelle, nous pourrions envisager d'identifier plus généralement ces sous-ensembles des aspects statiques.

7.5.2 Indécidabilité de la sureté du contrôle d'accès

Il faut remarquer que la définition d'un modèle d'administration d'un modèle de contrôle d'accès fait resurgir le théorème d'indécidabilité du *safety problem*, section 2.2.1.2 [Harrison76]. En effet, ce résultat a été établi dans la cadre des modèles

DAC où les utilisateurs peuvent définir des permissions sur les objets qu'ils détiennent. Dans le cas des modèles structurés, on pourrait penser ce problème décidable, car les utilisateurs ne peuvent pas définir de droits, mais les administrateurs le peuvent. C'est donc moins la sureté du modèle de base que celle du modèle d'administration qui est indécidable.

Nous pourrions envisager de limiter la portée de ce résultat en définissant un modèle de contrôle d'accès quelconque sur lequel les utilisateurs ne peuvent pas définir de permissions. Ensuite, les politiques de ce modèle seraient gérées selon un modèle d'administration qui devrait être complètement décidable, comme les modèles MAC. Il faut évaluer la portée et les implications d'une telle approche. On imagine déjà que le problème est repoussé plus loin, hors des frontières du système d'information, au niveau des procédures administratives qui régissent l'attribution de labels dans les modèles MAC.

7.5.3 Outils pour l'administration

Le chapitre 5 s'est intéressé à l'utilisation de la structuration relationnelle proposée dans la thèse. Nous avons défini les problèmes de conception et d'administration du contrôle d'accès, puis nous avons proposé des outils pour assister ces activités. L'axe de perspectives que nous présentons ici concerne ces outils. Il s'agit de capitaliser l'existant sur la conception de schémas relationnels et d'exploiter des liens entre les dépendances, les graphes conceptuels et l'analyse de relations binaires.

7.5.3.1 Forme normale des noyaux

Les dépendances de jointure sont intéressantes, car elles interviennent dans la décomposition des relations : elles permettent de garantir qu'une décomposition est sans perte. Prenons comme exemple une relation $Statique(U, R, A, O)$, qui stocke les autorisations statiques des utilisateurs et le rôle qui donne la permission. La dépendance de jointure $\bowtie [\{U, R\}, \{R, A, O\}]$ indique que la relation $Statique(U, R, A, O)$ est décomposable en deux sous relations $Habilite(U, R)$ et $Affecte(R, A, O)$, et que ces relations permettent de reconstruire $Statique(U, R, A, O)$ sans pertes.

Une des applications principales des dépendances de données est la conception de base de données. Cette discipline consiste à concevoir de « bons » schémas. À partir de la sémantique des données qu'il organise, le concepteur doit structurer un ensemble d'attributs en relations de telle façon :

- qu'aucune donnée ne soit perdue, ou que de « fausses » données soit générées,
- que le schéma dispose de bonnes propriétés, comme la non-redondance de l'information,
- que le schéma préserve bien la sémantique des données, en particulier les dépendances connues.

Les « bonnes propriétés » désirées pour un schéma sont regroupées en *formes normales*. Des algorithmes de mise sous forme normale existent [Abiteboul95]. Il pourrait être bénéfique de les utiliser pour vérifier le fondement des structurations des droits. Nous n'avons pas posé de conditions particulières sur les noyaux des modèles. On peut imaginer d'indiquer au concepteur du modèle dans quelle forme se trouve le schéma, et même de l'aider à le mettre sous une meilleure forme.

De tels outils existent dans les ateliers de conception de bases de données. Les composants logiciels que nous avons développés dans la LIBDEPENDENCIES pourraient être utilisés pour de tels moteurs, pour mettre des schémas sous forme normale complexe, autorisant la jointure sur des relations définies récursivement par exemple. Cette application pourrait être conduite pour le contrôle d'accès mais aussi pour des problèmes de conception de base de données plus généralement.

Les travaux de Henning Christiansen permettraient même d'étendre la notion de validation d'un modèle [Christiansen06]. Ils recoupent la notion de simplification de modèle présentée en section 5.2.5. L'objectif de ces travaux est de simplifier un ensemble de contraintes d'intégrité. Leur utilisation permettrait peut-être de simplifier les propriétés d'un modèle de contrôle d'accès.

7.5.3.2 Analyse relationnelle de concepts

Nous avons présenté une méthode d'ingénierie des rôles hiérarchisés en section 5.5. Nous l'avons construite à partir d'une structure nommée sous-hiérarchie de Galois qui nous sert de base de rôles potentiels pour la création d'une hiérarchie pertinente.

La structure de la sous-hiérarchie de Galois a été reprise pour la définition de l'analyse relationnelle de concepts. Il ne s'agit non plus de trouver *un* concept intermédiaire hiérarchisé, mais *plusieurs* [Hacene07, Huchard07]. Ceci nous permettrait d'identifier avec un seul algorithme plusieurs concepts entre rôles et permissions, comme ceux définis par Epstein dans sa thèse, section 5.5.1 [Epstein02].

De plus, la notion d'élagage que nous utilisons pourrait être intégrée à un algorithme comme PLUTON, pour aboutir à la définition d'un algorithme de construction de la sous-hiérarchie de Galois sous contraintes, comme il en existe pour la construction du treillis des concepts [Besson06]. Cette approche permettrait de mieux définir le processus d'élagage que nous avons proposé comme un post-traitement de la sous-hiérarchie de Galois.

7.5.3.3 Pavage d'une relation binaire

La recherche de rectangles dans une relation binaire est un socle commun sur lequel ont été définis plusieurs problèmes différents de fouille de données [Geerts04]. Les auteurs de [Vaidya07] ont défini le problème de l'ingénierie des rôles comme celui

du pavage d'une relation binaire. Leur méthode d'analyse et d'interprétation consiste à identifier un ensemble de rôles le plus petit possible, puis d'éventuellement modifier itérativement cet ensemble pour obtenir une hiérarchie satisfaisante.

Notre définition de l'ingénierie des rôles est différente. Elle est exprimable comme la recherche d'un ensemble partiellement ordonné dans une relation binaire, qui comporte toutes les intersections non vides d'utilisateurs et de permissions autorisés. Nous nous basons sur la sous-hiérarchie de Galois pour itérer le processus de raffinement.

Comme notre approche et celle de [Vaidya07] partent toutes deux d'une relation binaire, des ponts peuvent exister entre les deux définitions. Le pavage pourrait être une caractérisation d'un élagage maximal de la sous-hiérarchie de Galois. Le rôle de l'expert serait alors de trouver la hiérarchie la plus satisfaisante entre ces deux extrêmes.

7.5.4 Représentation graphique de modèles

Nous décrivons ici les perspectives relatives aux représentations graphiques des modèles et des politiques, et plus généralement l'interaction entre le concepteur ou l'administrateur et une organisation du contrôle d'accès.

7.5.4.1 Aspects contextuels

Comme nous avons développé un cadre logique pour le contrôle d'accès sur les dépendances de données, nous pouvons choisir de nous limiter à un fragment de logique en choisissant de n'utiliser qu'une classe donnée de dépendances. Malheureusement, les graphes conceptuels sont une forme d'équivalent graphique des dépendances génératrices de tuples totales *sans contraintes*. Hormis la relation d'égalité qui bénéficie d'un statut particulier, les contraintes ne sont pas exprimables dans les graphes conceptuels.

On pourrait envisager d'étendre le formalisme des graphes conceptuels avec des pictogrammes ou de nouveaux symboles graphiques qui permettraient de représenter des aspects qui nécessitent l'utilisation de contraintes, notamment les aspects contextuels comme l'espace ou le temps. Cette solution *ad hoc* n'aurait cependant pas le pouvoir d'expression des dépendances avec contraintes. On pourrait également envisager d'ajouter les contraintes aux graphes sous forme d'annotations formelles, comme *Object Constraint Language*, OCL, qui enrichit UML avec des formules logiques.

Cette solution paraît envisageable, mais il faudrait alors redéfinir de nouvelles procédures de chaînage capables de manipuler ces annotations formelles. De plus, les propositions attenantes aux graphes conceptuels s'attachent à définir toute opération logique graphiquement. On sortirait ainsi du cadre strict des graphes conceptuels pour se rapprocher d'une représentation plus spécifique comme DTAC [Tidswell01], dotée d'une sémantique logique correspondant à la structuration relationnelle que nous proposons.

7.5.4.2 Labélisation

Parmi les concepts présents dans le treillis des concepts formels, certains sont *sup irréductibles* ou *inf irréductibles* [Ganter97] : ils ne peuvent pas être exprimés comme l'union ou l'intersection d'autres concepts. Nous pensons qu'il serait intéressant d'introduire cette notion dans notre proposition : à partir d'une caractérisation formelle, nous apposerions des *labels* sur les concepts de la sous-hiérarchie de Galois. On pourrait faire intervenir ces labels accompagnés d'autres indicateurs sur les notes que nous avons faites figurer sur les hiérarchies de la figure 5.5. L'objectif serait de faciliter la sélection des concepts pertinents et l'interprétation de la hiérarchie de rôles obtenue par l'expert. On pourrait également comparer les irréductibles aux rôles identifiés avec la méthode de [Vaidya07].

7.5.4.3 Intégration des outils

Les graphes conceptuels et l'analyse de concepts formels sont liés [Wille97]. La mise à profit des résultats entre ces deux disciplines permettrait d'homogénéiser encore plus notre proposition. Comme les graphes conceptuels sont liés au fragment des dépendances et que l'analyse de concepts se fonde sur des relations binaires, un pont supplémentaire entre ces deux domaines permettrait d'enrichir encore les deux aspects de la conception de modèle et celui de la conception de hiérarchie que nous avons présentés relativement indépendamment.

7.6 Industrialisation de la proposition

Cette section propose des directions pour améliorer notre proposition en vue d'une éventuelle industrialisation. Il s'agit de verrous techniques, mais aussi scientifiques.

7.6.1 Bibliothèque pour les dépendances

La version sur laquelle nous travaillons actuellement vise à rendre la LIBDEPENDENCIES plus générique, plus efficace et plus utilisable. Ces objectifs passent par l'indépendance des algorithmes vis-à-vis des structures de données, la mise à disposition de meilleures interfaces mieux encapsulées et isolées. Les principaux chantiers logiciels concernant la bibliothèque sont :

- une meilleure organisation des classes d'algorithmes, en particulier la classe outil utilisée par les procédures `CEnvironnement` et ses spécialisations,
- une meilleure utilisation de la STL et des algorithmes génériques qu'elle propose,
- l'intégration de certains patrons de conception pour la structuration de la bibliothèque. Notamment Facade pour le solver, Patron de Méthode pour articuler les

différentes procédures à partir d'une même base, Composite et Visitor pour les structures de données récursives,
- l'utilisation de plus de composants logiciels offerts par BOOST et les projets périphériques, comme la gestion de l'interface ligne de commande, l'utilisation de l'analyseur syntaxique Spirit et l'interfaçage avec les systèmes de gestion de bases de données
- la définition générique, à l'aide de templates, des outils de manipulation de type et de métaprogrammation. Il s'agit là d'une révision majeure qui impacterait profondément le code avec l'adoption du paradigme *Modern C++* [Abrahams04, Alexandrescu01].

Il faut de plus s'assurer de la sécurité de la LIBDEPENDENCIES. En effet, si l'on imagine qu'elle soit utilisée dans un logiciel déployé et mise en production, la LIBDEPENDENCIES serait utilisée pour les outils de raisonnement et de vérification sur le contrôle d'accès, mais également dans le moniteur. Or le moniteur doit être incontournable, inviolable et vérifié. Il faut donc s'assurer de la correction de l'implémentation. La mise en place de tests fonctionnels rigoureux, de bonnes pratiques et l'intégration de cryptographie seraient des directions envisageables.

7.6.2 Domaines d'application de la proposition

Le contrôle d'accès est un des mécanismes les plus transparents des systèmes d'information : on le retrouve dans tout type de logiciel et d'architecture. Notre proposition pourrait donc s'appliquer pour contrôler les accès à un système d'exploitation.

Les modèles RBAC ont été mis en œuvre dans les systèmes Microsoft, Sun et récemment dans Linux, avec le projet de sécurisation orienté noyau `grsecurity`[1]. Le cadre relationnel que nous avons construit permettrait de rétroconcevoir ces modèles et les formaliser logiquement. On pourrait ainsi utiliser les différents outils présentés au chapitre 5. L'expressivité du cadre des dépendances permettrait de modéliser les spécificités des implémentations de RBAC par exemple.

Comme nous avons proposé une structuration relationnelle du contrôle d'accès, mais aussi des outils pour le concevoir et l'administrer, une perspective d'application qui permettrait de réutiliser le plus grand nombre de composants que nous avons définis est un « centre de sécurité » pour les systèmes d'information.

Ces applications sont le centre nerveux des architectures de contrôle d'accès et de sécurité plus généralement. Elles intègrent dans une interface et avec un cadre logiciel commun l'ensemble des applications nécessaires à la conception et à l'administration de sécurité. L'objectif serait d'utiliser la structure relationnelle de la thèse comme base pour le moteur d'un tel centre de sécurité. Ainsi, nous proposerions un cadre sur lequel construire les fonctionnalités attendues d'un tel environnement d'administration. Le

[1] `http://www.grsecurity.net/`

centre de sécurité serait comparable à un atelier de « génie du contrôle d'accès » avec la possibilité de vérifier et de simplifier des modèles et des politiques.

Un langage graphique visuel inspiré des graphes conceptuels permettrait au concepteur de sélectionner les notions qu'il veut intégrer à partir d'une bibliothèque de concepts, relations, principes et propriétés des modèles de contrôle d'accès existants. On pourrait ainsi définir sous forme d'options à activer les propriétés que les relations respectent : choisir que c'est une exclusion, une hiérarchie en arbre, générale, etc. Tout serait déclaratif : la manipulation d'objet graphique serait traduite en terme de dépendances et de relations logiques.

Un autre outil serait un atelier d'administration des politiques de contrôle d'accès. L'atelier de conception de politique utiliserait l'outil d'ingénierie des rôles en proposant plusieurs stratégies de sélection des concepts pertinents. Ensuite, une fois la politique développée, on utiliserait les outils de vérification de politique pour l'administration.

7.6.3 Systèmes de gestion de base de données

Notre proposition a structuré le contrôle d'accès du point de vue des données relationnelles. Dans l'architecture que nous proposons, les politiques de contrôle d'accès sont stockées dans une base de données dédiée. Cette base de données peut être gérée par le même système que celui qui gère déjà les données de production existante de l'organisation. On rapproche ainsi les données métiers des données de contrôle d'accès qui gouvernent les droits que les utilisateurs finaux ont sur les données de production.

Si les dépendances peuvent être utilisées pour les applications à la sécurité que nous avons proposée dans la thèse, elles peuvent également être utilisées pour garantir l'intégrité des données métiers stockées dans le système de gestion de bases de données. Lorsqu'on envisage l'application de notre proposition pour contrôler l'accès à des données on doit séparer :

- d'une part, les *données métiers*, ou *données de production*, c'est-à-dire l'ensemble des informations relatives aux activités de l'organisation et dont nous souhaitons contrôle d'accès. Ces données sont stockées dans des bases et des tables « classiques » du système,
- d'autre part, les données de sécurité, c'est-à-dire l'ensemble des informations relatives aux droits des usagers du système, stockées dans une base de données spécifique, typiquement dans le *catalogue*, apellée quelquefois *métabase* ou *base de données système*.

La prise en compte conjointe de la modélisation des données de production et de celle du contrôle d'accès permettrait d'exprimer des politiques bien plus complexes et plus proches des organisations. L'objectif serait d'organiser le contrôle d'accès aux bases de données avec les outils de bases de données.

7.7 Contrôle d'accès aux réseaux

Cette perspective envisage de réutiliser le cadre relationnel proposé dans la thèse pour une autre application que le contrôle d'accès logique au système d'information : le contrôle d'accès aux réseaux. Il est même envisageable de combiner les deux applications dans une perspective d'intégration globale de la sécurité des accès : sur les couches basses du réseau, mais aussi au niveau des applications.

7.7.1 Problématique

Les pare-feux sont désormais des outils incontournables pour garantir la sécurité des réseaux [CERT/CC05]. La fonction d'un pare-feu est d'inspecter les trames réseau qui le traversent pour déterminer si elles peuvent passer ou non. La prise de décision est basée sur un ensemble de règles, sa politique, qui définit à quelles conditions une trame doit être rejetée ou autorisée à passer. On redéfinit ainsi la problématique centrale du contrôle d'accès.

Avec l'apparition d'utilisations de plus en plus sophistiquées et d'augmentation de la taille des réseaux, les pare-feux ont évolué du simple routeur filtrant à des technologies élaborées telles que l'inspection stateful ou le filtrage applicatif, capables de gérer de très gros débits de données. Ceci rend les politiques de plus en plus complexes et nombreuses, à tel point que la politique d'un pare-feu de taille réelle peut devenir inintelligible pour un humain.

L'administration des pare-feux devient ainsi de plus en plus pénible et nécessite l'utilisation rigoureuse de bonnes pratiques [CERT/CC02] sans quoi le nombre d'erreurs de configuration rendrait le dispositif de filtrage tout à fait inutile. La tâche devient encore plus complexe lorsque plusieurs firewalls sont déployés dans un réseau et que différents administrateurs interviennent.

Afin de réduire au minimum le nombre d'erreurs de configuration des règles de pare-feux, qui sont une source majeure de failles, il faut proposer aux administrateurs des outils d'interrogation et de vérification automatique des pare-feux. La structuration et les outils proposés dans la thèse permettent peut-être d'apporter des éléments de réponse à ce problème.

7.7.2 Objectif

La principale approche utilisée en pratique est celle du test de pénétration (*pentesting*) où l'on teste effectivement le pare-feu en envoyant des paquets et en vérifiant si les trames sont passées ou non, comme le ferait un éventuel pirate. Cette approche n'est pas satisfaisante, car elle est lente, elle encombre les réseaux et elle n'est pas fiable. Comment savoir si un paquet s'est perdu ou s'il a été bloqué ?

La transposition des travaux de la thèse serait donc une approche formelle, basée sur une modélisation de ce que sont les politiques des pare-feux, les principes qui les gouvernent, les paquets et enfin la topologie du réseau. En utilisant un cadre relationnel pour cette modélisation on disposerait d'éléments

- de représentation des principes complexes introduits dans les firewall. Par exemple, des modules sont intégrés dans les produits existants comme `iptables` pour mémoriser les connexions ou gérer l'ouverture dynamique de ports,
- d'expression d'ensemble de propriétés à vérifier sur les jeux de règles et permettant de savoir si ces propriétés sont satisfaites ou pas, comme nous avons vérifié et simplifié des politiques,
- d'interrogation des pare-feux prenant en compte la topologie du réseau. On pourrait par exemple déterminer si un ensemble de trames peut atteindre une destination ou pas, ou identifier quel ensemble de trames peut atteindre une cible donnée.

7.7.3 Choix du cadre logique

Les problématiques des pare-feux sont pour certains aspects semblables à celles du contrôle d'accès, mais l'utilisation faite et les contraintes de ces deux mécanismes de protection sont différentes. Le domaine sur lequel exprimer les politiques est différent. Alors que pour le contrôle d'accès la modélisation est composée de formules logiques avec peu de contraintes et beaucoup de symboles de prédicats, pour les pare-feux il s'agit plutôt d'intervalles sur des valeurs et de peu de symboles de prédicats. De plus, les règles permettent d'exprimer soit l'autorisation soit l'interdiction de transiter. On est dans un cas de politiques hybrides.

De plus, les ensembles principaux concernés par ces deux mécanismes sont différents : pour le contrôle d'accès, on peut manipuler des milliers d'utilisateurs, de rôles ou de permissions, pour les pare-feux, il s'agit d'ensemble de paquets, définis par les protocoles standards qui représentent des nombres de paquets potentiels bien plus grands[2]. Certaines techniques algorithmiques coûteuses ne sont donc pas envisageables.

On pourrait privilégier les travaux issus des bases de contraintes dont l'objectif est de représenter de manière finie de très grands ensembles exprimés par des contraintes arithmétiques [Revesz95]. Dans le paradigme des bases de contraintes, les données, c'est-à-dire les politiques de pare-feux sont représentées en intention. On ne spécifie pas que tel ou tel paquet va être rejeté ou va passer, mais qu'un ensemble de paquets satisfaisants à des contraintes va être rejeté ou va passer. Les règles des politiques pourraient être représentées par des polyèdres définis sur un espace à n-dimensions, chacune représentant un champ des protocoles réseau.

[2] En considérant seulement quatre champs des paquets IPv4 comme l'adresse source (32 bits), de destination (32 bits), le port source (16 bits) et de destination (16 bits) on obtient déjà 2^{96} paquets potentiels...

Les propositions existantes sur l'interrogation de telles données (sélection, projection, opérations ensemblistes) permettent de donner des opérations primitives sur lesquelles s'appuyer. Les dépendances permettraient par exemple d'imposer des contraintes sur les dates et sur le fonctionnement des protocoles. On pourrait modéliser des règles de la forme « si un *ping*[3] est autorisé à sortir, alors le *ping* entrant correspondant sera autorisé à rentrer ».

Plusieurs travaux se sont attachés à la formalisation des règles de pare-feux ou à leur vérification, mais aucun à notre connaissance n'est capable de représenter les règles de pare-feux dynamiques. Plusieurs éléments bibliographiques donnent des pistes pour cette perspective :

- [Al-Shaer04, Bartal04] proposent des outils de vérifications des firewalls à partir de cinq anomalies types,
- [Adi03] propose un état de l'art sur le sujet,
- [Eronen01] propose un approche à base de programmation par contraintes qui est à rapprocher des bases de contraintes,
- [Wool04] donne une liste des erreurs courantes à vérifier dans les pare-feux qui permettrait de créer un liste de propriétés types à vérifier,
- [Liu04] propose un langage d'interrogation de firewall type SQL mais adapté aux domaines de contraintes sans prédicats,
- [Gouda04] propose une structure de données efficace pour la décorrélation et le stockage des règles, c'est une première étape qui permet de supprimer la négation des politiques,
- [Frantzen03] propose un état de l'art des théories mathématiques utilisables pour la vérification de pare-feux,
- [Vigna03] propose une modélisation mathématique de la topologie du réseau, utile pour la vérification de politiques impliquant plusieurs pare-feux dans un même réseau.

Avec la reprise de la LIBDEPENDENCIES et les investigations sur les techniques de fouille dans les relations binaires, l'application aux pare-feux est une perspective que nous comptons réaliser à court terme et dans laquelle nous nous sommes engagés.

[3] Un *ping* est une requête d'écho attendant une réponse telle que définie dans le protocole de contrôle Internet Control Message Protocol (ICMP) [Postel81].

The grand aim of all science is to cover the greatest number of empirical facts by logical deduction from the smallest number of hypotheses or axioms.

Albert Einstein

A

Préliminaires au cadre logique

▷ *La thèse s'appuie sur les* dépendances de données *comme outils de modélisation logique des modèles de contrôle d'accès. Nous avons utilisé et fait référence à la logique du premier ordre dans toute la thèse pour définir une structure relationnelle du contrôle d'accès. Cette annexe reprend des définitions mathématiques existantes qui n'ont pas été présentées sur ce fondement. Dans ce chapitre nous décrivons :*

- *la logique du premier ordre,*
- *le modèle relationnel,*
- *la logique comme fondement des bases de données relationnelles.*

La notation utilisée et les résultats cités sont principalement issus de Foundations of Databases *pour le modèle relationnel et sa formalisation logique (sections A.3 et A.2) [Abiteboul95]. Nous reprenons des éléments des chapitres deux (*Theoretical Background*), trois (*The Relational Model*) et quatre (*Conjunctive Queries*) dans cette annexe.*

*Nous nous sommes appuyés sur le chapitre sept (*Logique du premier ordre*) du cours de logique et de théorie des ensembles de Patrick Dehornoy*[1] *pour la logique du premier ordre [Dehornoy07].* ◁

[1] `http://www.math.unicaen.fr/~dehornoy/surveys.html`

Plan du chapitre

A.1 Logique du premier ordre

L'objectif de la logique mathématique est d'étudier le raisonnement et en particulier la vérité des propositions exprimées dans le langage des mathématiques. C'est un langage formel qui permet la représentation de relations entre objets et la déduction de nouvelles relations à partir de relations connues comme vraies. La logique est utile, car c'est un cadre commun pour exprimer et définir formellement les fondements des bases de données.

A.1.1 Syntaxe de la logique du premier ordre

Définition (Langage)**.** *Un* vocabulaire*, ou* signature*, V d'un langage L du premier ordre est composé :*

- *de* variables*, appartenant à l'ensemble des variables Var,*
- *de symboles de* constantes*, appartenant à l'ensemble des constantes Const,*
- *de symboles de* prédicats *avec leur arité associée, appartenant à l'ensemble Pred*[2]*,*
- *de symboles de* fonctions *avec leur arité associée, appartenant à l'ensemble Fonc*[3]*,*
- *des* connecteurs *standards de la logique appartenant à l'ensemble* $\{\neg, \wedge, \vee, \leftrightarrow, \rightarrow\}$*,*
- *des* quantificateurs $\forall$ *signifiant « pour tout » et* $\exists$ *signifiant « il existe »,*
- *ainsi que des caractères de ponctuation virgule « , » et parenthèses « (» et «) ».*

Les ensembles Fonc, Const, Var et Pred sont disjoints.

Maintenant que nous disposons du vocabulaire de base : le lexique d'un langage du premier ordre, nous allons définir comment construire des phrases : les règles de syntaxes qui définissent si une phrase est bien formée.

Définition (Terme)**.** *L'ensemble Terme des termes d'un langage L sur un vocabulaire V est défini récursivement comme :*

- *une variable est un terme,*
- *une constante est un terme,*
- *si* $f \in$ *Fonc est un symbole de fonction d'arité n, et* $t_1, \ldots, t_n$ *des termes alors une expression de la forme* $f(t_1, \ldots, t_n)$*, appelée* forme fonctionnelle*, est un terme,*
- *rien d'autre n'est un terme*

Une *forme prédicative*, ou *proposition atomique*, ou simplement *atome*, est une expression de la forme $R(t_1, \ldots, t_n)$, où R est un symbole de prédicat d'arité n appartenant à *Pred* et $t_1, \ldots, t_n$ une liste de termes d'arité correspondante.

Définition (Formule)**.** *Une formule F d'un langage L est définie récursivement au moyen des règles suivantes :*

[2] On peut ajouter le symbole de relation binaire particulier « = ».

[3] Les constantes peuvent être considérées comme des fonction d'arité nulle.

- *un atome est une formule,*
- *si ϕ est une formule, alors $\neg\phi$ est une formule,*
- *si ϕ et ψ sont des formules, alors $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \leftrightarrow \psi$ et $\phi \rightarrow \psi$ sont des formules,*
- *si ϕ est une formule et x une variable, alors $\exists x\phi$ et $\forall x\phi$ sont des formules,*
- *rien d'autre n'est une formule.*

Un atome ou la négation d'un atome est appelé un *littéral*. L'ensemble des connecteurs est celui de la logique des propositions. Notons que l'ensemble proposé : négation ($\neg$), conjonction ($\wedge$), disjonction ($\vee$), équivalence ($\leftrightarrow$) et implication ($\rightarrow$) n'est pas minimal, mais que nous l'utilisons pour rendre plus lisible la notation. L'implication $\phi \rightarrow \psi$ en particulier, est un raccourci syntaxique pour $\psi \vee \neg\phi$ de même que $\phi \leftrightarrow \psi$ est équivalent à $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$. De plus, nous notons $\forall x_1, \ldots, x_n$ et $\exists y_1, \ldots, y_n$ pour respectivement $\forall x_1, \ldots, \forall x_n$ et $\exists y_1, \ldots, \exists y_n$

Lorsqu'une variable x appartient à une formule précédée d'un quantificateur, $\forall x$ ou $\exists x$, elle est dite *liée* par ce quantificateur. Si une variable n'est liée par aucun quantificateur, elle est *libre*. La distinction entre variable libre et variable liée est importante. Une variable liée ne possède pas d'identité propre et peut être remplacée par n'importe quel autre nom de variable qui n'apparaît pas dans la formule. Ainsi, $\exists x\ a(x, y)$ est identique à $\exists z\ a(z, y)$ mais pas à $\exists x\ a(x, z)$ et encore moins à $\exists y\ a(y, y)$. Une formule dont *toutes* les variables sont *liées* est appelée une formule *close*, ou *fermée* (*sentence* en anglais), nous verrons que ces formules sont du plus grand intérêt. Une *théorie* est un ensemble de formules closes.

Définition (Clause)**.** *Une* clause *est une formule fermée de la forme :*

$$\forall x_1, \ldots, x_s(A_1 \vee \ldots \vee A_k \vee \neg B_1 \vee \ldots \vee \neg B_n)$$

ou de façon équivalente

$$\forall x_1, \ldots, x_s(B_1 \wedge \ldots \wedge B_n \rightarrow A_1 \vee \ldots \vee A_k)$$

où chaque A_i et B_i est un atome et $x_1, \ldots, x_s$ sont des variables apparaissant dans ces atomes. Une clause telle que $k \leq 1$ est appelée clause de Horn*. Une clause telle que $k = 1$ et $n = 0$ est appelée un* fait*.*

Les clauses de Horn constituent une base de la programmation logique (PROLOG) et des bases de données logiques (DATALOG). Plusieurs propositions de cadres logiques pour le contrôle d'accès se limitent à des clauses. Cependant, leur pouvoir d'expression est grand, mais il ne permet pas toujours d'exprimer certaines propriétés attendues des modèles de contrôle d'accès, du moins sans passer par des réécritures (par exemple la Skolémisation).

Exemple A.1 Socrate est (toujours) mortel

Prenons comme illustration de l'expression des formules de la logique un syllogisme célèbre. Considérons le raisonnement suivant :

1. Tout homme est mortel ;
2. Socrate est un homme ;
3. *donc* Socrate est mortel

Ces expressions exprimées en langue naturelle peuvent s'écrire en logique dans le vocabulaire $Const = \{socrate\}$, $Var = \{H\}$, $Pred = \{Homme, Mortel\}$ et $Fonc = \emptyset$.

1. $\forall H\ (Homme(H) \rightarrow Mortel(H))$
2. $Homme(socrate)$
3. $Mortel(socrate)$

Nous verrons que l'étude du raisonnement « *si* les propositions (1) et (2) sont vraies *alors* la proposition (3) est vraie aussi » est l'objet de l'étude de la *sémantique*.

A.1.2 Sémantique de la logique du premier ordre

Jusqu'ici, nous n'avons pas donné de sens aux formules de la logique, nous n'avons fait que *spécifier* leur syntaxe. Une formule peut être interprétée comme une phrase sur un ensemble d'objets : il est possible de lui donner une *signification* vis-à-vis de cet ensemble d'objets.

L'ensemble des objets considéré est appelé l'*univers du discours*, ou *domaine du discours*, il est noté U. Un prédicat représente une *relation* particulière entre les objets de U, qui peut être vraie ou fausse. Définir cette relation revient à définir les n-uplets d'objets qui satisfont le prédicat. L'univers du discours est donc un ensemble d'objets mathématiques sur lequel une formule logique prend une valeur par *interprétation* :

- des constantes comme des objets particuliers,
- des variables comme des objets quelconques,
- des fonctions comme des fonctions particulières entre objets
- des prédicats comme des relations entre les objets.

Définition (Interprétation)**.** *Une* interprétation[4] *d'un langage L de vocabulaire V est un quadruplet $\mathcal{I} = (\mathcal{U}, \mathcal{C}, \mathcal{F}, \mathcal{P})$ où*

- *$\mathcal{U}$ est un ensemble dénombrable appelé* univers du discours *ou* domaine d'interprétation,
- *$\mathcal{C}$ est une fonction $Const \rightarrow \mathcal{U}$ qui à chaque constante faire correspondre un objet de $\mathcal{U}$,*
- *$\mathcal{F}$ est une fonction $\mathcal{U}^n \rightarrow \mathcal{U}$ qui a chaque symbole de fonction n-aire f fait correspondre son interprétation $f^{\mathcal{I}}$, un objet de $\mathcal{U}$,*

[4] En mathématique l'usage serait plutôt de traiter de structure de type V ou de réalisation de L.

– *$\mathcal{P}$ une fonction U^n dans $\{vrai, faux\}$, qui à chaque symbole de prédicat n-aire p fait correspondre son interprétation $p^{\mathcal{I}}$. $p^{\mathcal{I}}$ est une relation n-aire entre des objets de U, c'est-à-dire un sous-ensemble de U^n.*

Une interprétation $\mathcal{I}$ est finie *si son univers du discours est fini.*

Donnons un exemple de ce que peut être une interprétation. Soit le langage L_N composé à partir du vocabulaire :

– du symbole de constante **0**,
– des symboles de prédicat binaires $\leq, =$,
– des symboles de fonction unaire **S**, et binaires $+, \times$.

Une interprétation de L_N est $\mathcal{I}_N = (\mathbb{N}, 0, S, +, \times, \leq, =)$, où l'univers du discours est $\mathbb{N}$, **0** est associé à l'entier naturel 0, **S** est associée à la fonction successeur, $+$ et $\times$ sont associés à l'addition et la multiplication, enfin, $\leq$ et $=$ sont associés aux relations inférieur ou égal et égal.

Par exemple, les formules suivantes sont bien formées dans le langage L_N :

– $\forall x(\neg(= (\mathbf{S}(x), \mathbf{0})))$,
– $\forall x \exists y(\neg(= (x, \mathbf{0})) \rightarrow = (\mathbf{S}(y), x))$,
– $\forall x(\mathbf{0} \leq x)$,
– $\neg\exists x(\forall y(\leq (y, x)))$.

L'interprétation du terme $\mathbf{S}(\mathbf{S}(\mathbf{0}) + \mathbf{0})$ est l'entier naturel 2 et les formules closes précédentes sont évaluées à *vrai* dans $\mathcal{I}_N$. On peut par exemple associer à la dernière formule la phrase de la langue naturelle « il n'existe pas d'entier naturel plus grand que tous les autres ».

Définition (Valuation)**.** *Une* valuation *μ (de variables) est une fonction $\mu : Vars \rightarrow U$. Pour un terme t, $t^{\mathcal{I},\mu}$ est la sémantique conférée à t par $\mathcal{I}$ en utilisant la valuation μ pour interpréter les variables. La valuation est récursivement étendue aux termes dans une interprétation $\mathcal{I} = (U, \mathcal{C}, \mathcal{F}, \mathcal{P})$ de la façon suivante :*

– *$x^{\mathcal{I},\mu} = \mu(x)$ si $x \in Var$*
– *$c^{\mathcal{I},\mu} = d$ où c est un symbole de constante de Const et $\mathcal{C}$ fait correspondre $d \in U$ à c,*
– *$(f(t_1, \ldots, t_n))^{\mathcal{I},\mu} = f^{\mathcal{I},\mu}(t_1^{\mathcal{I},\mu}, \ldots, t_n^{\mathcal{I},\mu})$ où f est un symbole de fonction de Fonc d'arité n, $\mathcal{F}$ fait correspondre $f^{\mathcal{I},\mu} \in U$ à f et que les $t_i^{\mathcal{I},\mu} \in U$ sont les éléments de U assignés récursivement aux termes t_i.*

La valeur de vérité *d'un atome $p(t_1, \ldots, t_n)$ dans une interprétation $\mathcal{I}$ et une valuation μ est la valeur booléenne obtenue en appliquant la fonction $p^{\mathcal{I}} : U^n \rightarrow \{vrai, faux\}$ assignée à p par $\mathcal{P}$. Les $t_1^{\mathcal{I},\mu}, \ldots, t_n^{\mathcal{I},\mu}$ sont assignés aux termes $t_1, \ldots, t_n$ par $\mathcal{I}$ et μ. $p^{\mathcal{I}}$ fait correspondre aux éléments $t_1^{\mathcal{I},\mu}, \ldots, t_n^{\mathcal{I},\mu}$ de U une valeur booléenne. De façon équivalente, un atome est évalué à vrai si $t_1^{\mathcal{I},\mu}, \ldots, t_n^{\mathcal{I},\mu} \in p^{\mathcal{I}}$. On écrit $\mathcal{I} \models p(t_1, \ldots, t_n)[\mu]$ pour indiquer que $p(t_1, \ldots, t_n)$ est évalué à vrai dans $\mathcal{I}$ pour une valuation μ.*

Dans le cas où le symbole de prédicat binaire « = » est ajouté à l'ensemble des symboles de prédicats, l'interprétation de $=(t_1, t_2)$ est *vrai* si $t_1^{\mathcal{I},\mu} = t_2^{\mathcal{I},\mu}$ et *faux* sinon.

Définition (Satisfaction de formule)**.** *On note $\mu[x/u]$ la valuation identique à μ, sauf pour la variable x auquel elle fait correspondre u. Une formule ϕ est dite* satisfaite *dans une interprétation $\mathcal{I}$ et une valuation μ, noté $\mathcal{I} \models \phi[\mu]$ ssi sa valeur de vérité est vrai.*

La valeur de vérité *d'une formule du premier ordre ϕ dans une interprétation $\mathcal{I}$ et une valuation μ est déterminée en fonction des valeurs de vérité des atomes en appliquant les tables de vérité usuelles aux connecteurs logiques booléens $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ et les règles suivantes aux quantificateurs :*

- *$\exists x\phi$ est vraie dans $\mathcal{I}$ s'il* existe *$u \in U$ tel que $\mathcal{I} \models \phi[\mu[x/u]]$*
- *$\forall x\phi$ est vraie dans $\mathcal{I}$ si* pour tout *$u \in U$, $\mathcal{I} \models \phi[\mu[x/u]]$*

Toute formule *close* ϕ a donc une unique *valeur de vérité* pour une interprétation donnée sur un univers du discours U, et donc quelque soit μ. On note alors $\mathcal{I} \models \phi$ pour indiquer que ϕ est *satisfaite* dans $\mathcal{I}$, ou que $\mathcal{I}$ *satisfait* ϕ, ou encore que $\mathcal{I}$ est un *modèle* de ϕ. Une formule ϕ est dite *valide*, noté $\models \phi$ si toute interprétation la satisfait.

Une formule close ϕ *implique logiquement* une autre formule close ψ, noté $\phi \models \psi$, si tout modèle de ϕ est aussi un modèle de ψ. On peut dire que ψ est une conséquence logique de ϕ.

Une interprétation $\mathcal{I}$ est un *modèle* d'un ensemble Φ de formules closes si $\mathcal{I}$ satisfait chaque formule de Φ. Un ensemble Φ est *satisfaisable* s'il admet un modèle. Un ensemble de formules Φ est *valide* si toute interprétation est un modèle de Φ.

La logique que nous venons de décrire dispose d'un très grand pouvoir d'expression, qui est utilisé pour exprimer formellement et raisonner dans les bases de données. Les requêtes, les vues, les règles DATALOG ainsi que les dépendances de données peuvent être exprimées à l'aide de formules logiques, chapitre 3, dont le sens est défini grâce à la *sémantique logique*.

Théorème. *Soient ϕ et ψ des formules closes de la logique du premier ordre. $\phi \models \psi$ si et seulement si $\models \phi \rightarrow \psi$*

A.1.3 Interprétation de Herbrand

Les formules de la logique du premier ordre s'interprètent dans des structures quelconques formées d'un domaine (l'univers du discours), d'opérateurs (les interprétations des fonctions) et de relations entre les éléments du domaine (les interprétations des prédicats). Une formule est *valide* si elle est vraie dans *toutes* les interprétations sur toutes les structures imaginables…

Nous allons maintenant nous intéresser à des interprétations particulières, qui sont utiles dans le cadre des bases de données : les *interprétations de Herbrand*. Toutes ces

Exemple A.2 Satisfaction d'une formule de l'arithmétique

La satisfaction d'une formule dépend de l'interprétation qu'on en fait, c'est-à-dire au sens que l'on donne aux symboles du langage.

Considérons la formule close $\phi : \forall x_1, \exists x_2 (x_1 = x_2 + 1)$ du langage de l'arithmétique du premier ordre, utilisant la notation infixe.

On peut définir deux interprétations $\mathcal{I}_{\mathbb{N}} = (\mathbb{N}, 1, +, =)$ et $\mathcal{I}_{\mathbb{Z}} = (\mathbb{Z}, 1, +, =)$ pour ce langage, pour lesquelles ϕ sera évaluée différemment.

En effet, la formule ϕ est satisfaite dans $\mathcal{I}_{\mathbb{Z}}$: tout nombre relatif a un prédécesseur, mais pas dans $\mathcal{I}_{\mathbb{N}}$, où 0 n'en a pas. ϕ est donc une formule *satisfaisable* (dans $\mathcal{I}_{\mathbb{Z}}$) mais pas *valide* (elle n'est pas satisfaite dans $\mathcal{I}_{\mathbb{N}}$).

interprétations partagent le *même* univers du discours et les mêmes interprétations des constantes et des symboles de fonctions, appelées *préinterprétation* de Herbrand.

Définition (Univers de Herbrand)**.** *L'*univers de Herbrand *d'un langage L du premier ordre est l'ensemble des termes* clos *(sans variables) formés sur les symboles de fonctions et les constantes du langage. On peut définir l'univers de Herbrand récursivement par strates :*

- *soit H_0 l'ensemble des constantes Const de L. Si Const est vide, alors on pose $H_0 = \{a\}$ où a est une constante arbitraire,*
- *soit H_{i+1} l'union de H_i et de l'ensemble des termes de la forme $f(t_1, \ldots, t_n)$, pour tout symbole de fonction f d'arité n de L et où les t_i sont des éléments de H_i,*

Alors $lim_{i \to \infty} H_i = H_\infty$ est l'univers de Herbrand du langage L.

Par exemple, pour un langage comportant comme symboles de constante a et b et une fonction unaire f, l'univers de Herbrand est ainsi construit :

$$
\begin{aligned}
H_0 &= \{a, b\} \\
H_1 &= \{a, b, f(a), f(b)\} \\
H_2 &= \{a, b, f(a), f(b), f(f(a)), f(f(b))\} \\
&\ldots \\
H_\infty &= \{a, b, f(a), f(b), f(f(a)), f(f(b)), f(f(f(a))), f(f(f(b))), f(f(f(f(a)))), \ldots\}
\end{aligned}
$$

Définition (Base et interprétation de Herbrand)**.** *Une* préinterprétation de Herbrand *d'un langage L est un triplet $\mathcal{H} = (H_\infty, \mathcal{C}, \mathcal{F})$ où*

- *H_∞ est l'univers du discours,*
- *$\mathcal{C}$ est la fonction identité qui à chaque constante de L fait correspondre elle même,*
- *$\mathcal{F}$ interprète chaque fonction f d'arité n comme une fonction $H_\infty^n \to H_\infty$ qui fait correspondre aux termes $t_1, \ldots, t_n$ le terme $f(t_1, \ldots, t_n)$ de H_∞.*

Une base de Herbrand *$\mathcal{B}_H$ est l'ensemble des* atomes clos *(positifs) formés sur les termes de l'univers de Herbrand à l'aide des symboles de prédicats de L.*

Une interprétation de Herbrand $\mathcal{I} = (H_\infty, \mathcal{C}, \mathcal{F}, \mathcal{P})$ *étend une préinterprétation de Herbrand en associant à tout élément de* $\mathcal{B}_H$ *une valeur de vérité, où de façon équivalente, en définissant un sous-ensemble de* $\mathcal{B}_H$ *qui représente les formes prédicatives vraies.*

Ainsi, dans les interprétations de Herbrand, qui ne diffèrent que par $\mathcal{P}$, les termes ont un double rôles : la construction syntaxique abordée jusqu'à présent *et* des éléments de l'univers du discours. La préinterprétation fait correspondre au symbole de constante a, l'élément du discours a et à la construction syntaxique $f(a)$ l'élément du discours $f(a)$.

Théorème (Théorème de Herbrand)**.** *Soit S un ensemble de clauses. S est insatisfaisable si et seulement si S n'a pas de modèle de Herbrand, c'est-à-dire d'interprétation de Herbrand qui le satisfasse.*

L'étude de la satisfaction de formules peut donc être restreinte aux seules interprétations « syntaxiques » que sont les interprétations de Herbrand. Cette propriété n'est cependant pas vraie pour les formules logiques plus générales que les clauses, mais nous savons qu'il est possible de réécrire toute formule ϕ en un ensemble de clauses ϕ^s selon le processus suivant :

1. élimination des symboles d'implication $\rightarrow$ et d'équivalence $\leftrightarrow$ par utilisation des équivalences $\phi \rightarrow \psi \equiv \psi \vee \neg\phi$ et $\phi \leftrightarrow \psi \equiv (\psi \wedge \phi) \vee (\neg\psi \wedge \neg\phi)$,
2. mise sous forme prénexe, c'est-à-dire sous forme d'une formule où les quantificateurs apparaissent au début et dont la portée s'étend sur tous les atomes,
3. élimination des quantificateurs existentiels par *Skolémisation*,
4. mise sous forme conjonctive (conjonction de disjonctions de littéraux), par utilisation des lois de De Morgan : $\neg(\phi \vee \psi) \equiv (\neg\phi \wedge \neg\psi)$ et $\neg(\phi \wedge \psi) \equiv (\neg\phi \vee \neg\psi)$,
5. chaque disjonction de littéraux de cette formule est alors une clause.

L'élimination des variables quantifiées existentiellement se fait par *Skolémisation*, en remplaçant chaque variable par un terme de la forme $f(x_1, \ldots, x_n)$ où f est un nouveau symbole de fonction et où les $x_1, \ldots, x_n$ sont les variables quantifiées existentiellement qui la précédent. Par exemple, la forme normale de Skolem de la formule $\forall x \exists y \forall z\ p(x, y, z)$ est la formule $\forall x \forall z\ p(x, f(x), z)$. Notons que ϕ^s n'est pas équivalent à ϕ mais que ϕ^s est satisfaisable *si et seulement si* ϕ est satisfaisable. Ce principe est à la base de la programmation logique.

A.1.4 Déduction

La logique des *propositions*, couramment utilisée en électronique, peut être vue comme une restriction de la logique des *prédicats* où seuls sont autorisés des symboles de prédicats d'arité nulle. Dans le cas propositionnel, on peut toujours évaluer *trivialement* la validité (ou la satisfaisabilité) d'une formule en testant toutes les affectations

possibles de valeurs de vérité aux variables, c'est-à-dire en utilisant la méthode des tables de vérité. Pour la logique du premier ordre, ce n'est malheureusement pas le cas.

La sémantique donnée aux formules n'a pas de caractère *constructif* et ne conduit pas, par sa définition, à un algorithme. Un des résultats fondamentaux de la logique mathématique est le développement de d'algorithmes pour déterminer l'implication logique : c'est la notion de *preuve*, ou *déduction*, qui permet de bâtir des démonstrations de formules logiques.

Définition (Déduction)**.** *La* déduction*, ou* preuve*, d'une formule ϕ d'un langage L du premier ordre à partir d'un ensemble de formules Φ appartenant également à L, noté $\Phi \vdash \phi$ est une suite de formules $\psi_1, \ldots, \psi_n$ telle que d'une part $\psi_n = \phi$ et d'autre part, $\forall i \in [1..n-1]$, ψ_i est soit :*

- *une formule appartenant à Φ,*
- *soit un* axiome *du calcul de la logique des prédicats,*
- *soit la formule obtenue par application des règles de déduction à partir de formules ψ_j, $j < i$*

Les axiomes de la logique sont obtenus à partir des axiomes du calcul propositionnel, c'est-à-dire que ce sont des formules obtenues en substituant dans *toute tautologie* de la logique des propositions des formules de L aux variables propositionnelles. Soit A, B, C des formules, x une variable, D une formule n'ayant pas x pour variable libre et t un terme, les formules suivantes sont axiomes de la logique du premier ordre :

1. $\neg A \vee A$,
2. $(A \to (B \to A))$,
3. $((A \to (B \to C)) \to (A \to B) \to (A \to C))$,
4. $(\neg B \to \neg A) \to (A \to B)$,

De plus, on ajoute les axiomes spécifiques aux quantificateurs, $(\forall x A(x) \to A(t))$ qui est appelé l'*axiome de substitution*, $((D \to B) \to (D \to \forall x B))$ ainsi que $\exists x \neg F \leftrightarrow \neg \forall x F$. Les deux règles de déductions étant la *coupure*, ou *modus ponens*, et la *généralisation* :

$$\text{modus ponens} \quad \frac{\vdash A, \vdash A \to B}{\vdash B}$$

$$\text{généralisation} \quad \frac{\vdash A,}{\vdash \forall x A}$$

Théorème (Théorème d'adéquation, de cohérence, ou de validité)**.** *Soit Φ un ensemble de formules closes d'un langage L, et ϕ une formule.*

Si $\Phi \vdash \phi$ alors toute interprétation qui satisfait Φ satisfait aussi ϕ : $\Phi \vdash \phi$ implique $\Phi \models \phi$. En particulier, si Φ est valide, il en est de même pour ϕ. $\vdash \Phi$ implique $\models \phi$: toute formule prouvable *est* valide*.*

Théorème (Théorème de déduction)**.** *Soient Φ un ensemble de formules closes d'un langage L, et ϕ et ψ deux formules, on suppose ϕ close. $\Phi \vdash \phi \rightarrow \psi$ est équivalent à $\Phi \cup \{\phi\} \vdash \psi$.*

On appelle *consistant* tout ensemble de formules closes (une théorie) qui une prouve jamais à la fois une formule et sa négation.

Théorème (Théorème de déduction, corollaire)**.** *Soient Φ un ensemble de formules closes d'un langage L, et ϕ une formule.*

- *Φ prouve ψ si et seulement si l'extension $\Phi \cup \{\phi\}$ est inconsistante.*
- *si Φ est consistante, l'une a moins des extensions $\Phi \cup \{\phi\}$, $\Phi \cup \{\neg\phi\}$ est consistante.*

Théorème (Théorème de complétude, Gödel)**.** *Soit Φ un ensemble de formules closes d'un langage L, et ϕ une formule.*

Si $\Phi \models \phi$ alors il existe une preuve ϕ utilisant Φ comme hypothèses : $\Phi \models \phi$ implique $\Phi \vdash \phi$.

Pour des raisons techniques (la description d'algorithmes de preuve), nous allons également introduire la définition d'une *substitution*.

Définition (Substitution)**.** *Une substitution σ est un ensemble fini de substitutions élémentaires (x_i/t_i) notée $\sigma = \{(x_1/t_1), \ldots, (x_n/t_n)\}$ où les x_i sont des variables deux à deux distinctes et où les t_i sont des termes.*

Soit F une formule, l'application de la substitution σ à la formule F notée $\sigma(F)$ est le résultat de la substitution simultanée dans F de chaque occurrence de x_i par le terme t_i et telle que chaque occurrence de x_i qui était libre *dans F ne devienne pas liée.*

Une substitution σ telle que pour deux atomes t_1 et t_2, $\sigma(t_1) = \sigma(t_2)$ est appelée un unificateur *de t_1 et t_2.*

A.2 Modèle relationnel

Le concept mathématique sous-jacent au modèle relationnel est celui de relation dans la théorie des ensembles, c'est-à-dire un sous-ensemble du produit cartésien de *domaines*.

Définition (Attribut)**.** *Un* attribut *est un symbole d'un ensemble fini* $U = \{A_1, \ldots, A_n\}$ *appelé* univers*. Tous les ensembles d'attributs sont des sous-ensembles de* U *pour lesquels nous omettons les accolades, ainsi* ABC *désigne* $\{A, B, C\}$*.*

Soit **dom** *un ensemble dénombrable de constantes. À chaque attribut* A *est associé un ensemble dénombrable de valeurs atomiques appelé* domaine *noté* $Dom(A) \subseteq$ **dom**.

Définition (Produit cartésien)**.** *Le produit cartésien des domaines* $D_1, \ldots, D_k$ *noté* $D_1 \times \ldots \times D_k$ *est l'ensemble* P *des k-uplets* $(v_1, \ldots, v_n)$ *tels que* $\forall v_i \in D_i : (v_1, \ldots, v_k) \in P$*.*

Définition (Relation)**.** *Soit* **relname** *un ensemble dénombrable de noms de relations. Un* schéma de relation $R \in$ **relname** *est un ensemble fini d'attributs* $A_1 \ldots A_n$*. L'*arité *d'un schéma de relation est la cardinalité de son ensemble d'attributs.*

Une instance de relation *de schéma* $R = A_1 \ldots A_n$ *de domaines respectifs* $D_i = Dom(A_i)$ *est un sous-ensemble du produit cartésien* $D_1 \times \ldots \times D_n$*. Une instance de relation est donc un ensemble de n-uplets appelés* tuples $(v_1, \ldots, v_n)$ *avec* $\forall i\ v_i \in D_i$*.*

Pour un tuple t *sur un schéma* R *et un ensemble* $X \subseteq R$*, la notation* $t[X]$ *est la restriction de* t *à* X*. Un tuple* t *sur un schéma de relation* R *peut, de façon équivalente, être défini comme une application* $R :\rightarrow \cup_{A \in R} Dom(A)$ *telle que* $\forall A \in R, t[A] \in Dom(A)$*.*

Définition (Base de donnée)**.** *Un* schéma de base de donnée*, ou simplement* schéma, **R** *est un ensemble fini non vide de schémas de relations* $\{R_1, \ldots, R_n\}$*.*

Une instance *de base de données, ou simplement une* base de données, **I** *est un ensemble d'instances de relations sur le schéma de base de données. Pour distinguer les schémas et instances de* base de données *des schémas et instances de* relation*, nous utilisons une police différente. Par exemple :* **R** *est un schéma de base de données,* R *un schéma de relation,* **I** *une instance de base de données,* I *une instance de relation.*

La représentation habituelle des bases de données relationnelles est celle de *tables*. À chaque table est associé un nom de *relation*, les colonnes de la tables sont les *attributs*. Chaque ligne de la table est un *tuple* ou *enregistrement*. Les valeurs des tuples sont prises dans un ensemble de constantes, le *domaine*, qui en général contient les entiers, les chaînes de caractères, les booléens, mais aussi les dates ou des structures plus complexes comme les images dans les systèmes de gestion de base de données contemporains. Le schéma spécifie la *structure* de la base de données, l'instance donne sa *valeur* actuelle.

A.3 Base de données et logique

Dans le cadre de la logique, la fonction de correspondance $\mathcal{C}$ d'une interprétation peut, à deux symboles de constantes différents faire correspondre un seul et même élément de l'univers du discours U. Cependant, nous nous intéressons à la logique pour son application aux bases de données, ce qui conduit aux spécialisations suivantes [Abiteboul95] :

- les symboles de fonctions ne sont pas autorisés dans la construction des formules,
- les bases de données étant par essence finies, les interprétations le sont aussi,
- les constantes ont un statut particulier, elles sont toutes distinctes.

A.3.1 Syntaxe

Le fait que les fonctions ne soient *pas* autorisées dans la construction des formules est une restriction importante, mais qui permet de simplifier et de garantir la finitude des ensembles manipulés. Pour mettre en parallèle les définitions proposées en section 3.2.1 et le cadre de la logique, nous nous donnons le langage $L_{\mathcal{R}}$ associé au schéma d'une base de données **R**. Le vocabulaire du langage $L_{\mathbf{R}}$ se compose :

- d'un ensemble de variables,
- d'un ensemble *fini* de constantes : les valeurs possibles **dom**,
- d'*aucun* symbole de fonction,
- d'un ensemble *fini* de symboles de prédicats n-aires, auxquels correspondent les symboles de relation n-aires de **relname**,
- à chaque forme prédicative de $L_{\mathbf{R}}$ correspond un schéma de relation de **R**.

Le procédé de formation des formules est le même que celui décrit pour la logique du premier ordre. On note que l'univers de Herbrand est d'une part fini, et d'autre part limité à sa première strate $H_\infty = H_0$. Ainsi, lorsqu'on interprète les formules de $L_{\mathbf{R}}$:

- un prédicat représente une relation particulière entres les objets du discours : définir cette relation revient à définir les tuples qui satisfont le prédicat : l'*instance* de relation,
- à chaque *place* de terme dans un atome, correspond un ensemble de constantes, le domaine,
- la notion d'attribut du modèle relationnel disparait et est remplacée par la *position* des termes dans les atomes,
- une interprétation consistant à définir les relations vraies entre les éléments du domaine, une base de données peut être définie comme l'interprétation (finie) de $L_{\mathbf{R}}$. En particulier, une instance de base de données peut être assimilée à une interprétation de Herbrand finie, ce qui permet de confondre les notations $\mathcal{I} \models \phi$ et $\mathbf{I} \models \phi$ où $\mathcal{I}$ est une interprétation et **I** une instance de base de données.

Une autre hypothèse faite dans les bases de données est l'*hypothèse du monde clos*, qui signifie que seuls sont considérés comme *vrais* les faits stockés dans la base de données. Les faits non enregistrés dans une extension de prédicats (c'est-à-dire une relation) sont supposés faux, à la différence de l'*hypothèse du monde ouvert* où un fait non enregistré est considéré comme inconnu. L'hypothèse du monde clos suppose donc une connaissance complète du monde réel que la base de données décrit, mais dont on n'enregistre que les faits positifs, pour minimiser la taille des informations stockées.

Enfin, pour permettre l'expression de requêtes incluant des comparaisons sur les éléments du domaine d'un attribut, le langage logique est étendu pour inclure le prédicat binaire « $=$ » d'égalité entre termes. On peut étendre les comparaisons avec d'autres prédicats particuliers, dont la valeur de vérité est calculable par la machine et indépendamment des instances, appelés prédicats *built-in*, comme les relations de comparaisons usuelles sur les entiers ($\leq, <, \geq, >, =, \neq$).

Selon que l'on restreigne l'expression des formules de $L_{\mathbf{R}}$, la logique du premier ordre permet de représenter et de définir la sémantique de différents composants des bases de données relationnelles :

- des *requêtes* sur la base de données,
- des *vues* sur la base de données,
- des *règles* de déductions de nouvelles données à partir des connues,
- des *dépendances* entre les données.

A.3.2 Requêtes conjonctives

La syntaxe et la sémantique des *requêtes conjonctives* peuvent être définies sans utiliser la logique, avec l'algèbre relationnelle par exemple. Il s'agit d'ailleurs du point de vue adopté au départ de l'étude du modèle relationnel. Ce n'est que plus tard que le pont entre logique et modèle relationnel a été fait. Il s'avère que les requêtes exprimées en algèbre relationnelle et en logique sont équivalentes : elles ont le même pouvoir d'expression [Abiteboul95].

Définition (Requête conjonctive)**.** *Une formule du calcul conjonctif sur un schéma de base de données de* **R** *est une expression de la forme :*

- *un atome est une formule de requête conjonctive,*
- *si ϕ et ψ sont des formules alors $\phi \wedge \psi$ sont des formules,*
- *si ϕ est une formule et x une variable, alors $\exists x \phi$ est une formule,*

Une requête conjonctive sur un schéma de base de données de **R** *est une expression :*

$$\{e_1, \ldots, e_m \mid \phi\}$$

où ϕ est une formule du calcul conjonctif, $e_1, \ldots, e_m$ un tuple libre et l'ensemble des variables apparaissant dans $e_1, \ldots, e_m$ est exactement l'ensemble des variables libres de ϕ.

Une instance de base de données $\mathbf{I}$ *sur le schéma* $\mathbf{R}$ *satisfait une formule* ϕ *du calcul conjonctif pour une valuation* ν*, noté* $\mathbf{I} \models \phi[\nu]$*, si*

- *soit* $\phi = R(u)$ *est un atome et* $\nu(u) \in \mathbf{I}(R)$,
- *soit* $\phi = (\psi \wedge \xi)$ *est une conjonction de formules et* $\mathbf{I} \models \psi[\nu]$ *et* $\mathbf{I} \models \xi[\nu]$,
- *soit* $\phi = \exists x \psi$ *et pour une constante* $c \in \mathbf{dom}$, $\mathbf{I} \models \psi[\nu[x/c]]$

Enfin, $q = \{e_1, \ldots, e_m \mid \phi\}$ *une requête conjonctive sur un schéma* $\mathbf{R}$*. Pour une instance* $\mathbf{I}$*, l'*image *de* $\mathbf{I}$ *par* q *est :*

$$q(\mathbf{I}) = \{\nu(< e_1, \ldots, e_m >) \mid \mathbf{I} \models \phi[\nu] \text{et } \nu \text{ est une valuation sur les variables libres de } \phi\}$$

Le langage des formules du calcul conjonctif est donc un sous-ensemble de $L_{\mathbf{R}}$. Les réponses d'une requête forment une relation en *intention*, c'est-à-dire une relation dont on n'a pas défini explicitement l'instance, mais dont on a défini la propriété ϕ qu'elle doit respecter.

Ceci conduit au concept de *vue* utilisé dans les systèmes de gestion de base de données et évoqué en section 2.2.3. Une vue est simplement une requête conjonctive à laquelle on fait correspondre un nom de relation, ne faisant pas partie de $\mathbf{R}$, en d'autre termes un nouveau symbole de prédicat.

If I had eight hours to chop down a tree, I would spend six hours sharpening an axe.

Anonymous

B

Théories logiques

▷ *Cette annexe propose des compléments techniques : des listes des symboles et deux exemples de théories logiques pour représenter des modèles de contrôle d'accès.*

En premier lieu, nous synthétisons les principaux symboles utilisés dans la thèse. Ils sont classés selon qu'ils sont utilisés pour les définitions ensemblistes, pour la définition du cadre relationnel ou pour la représentation logique de modèles de contrôle d'accès.

Ensuite, nous présentons une théorie logique pour les modèles MAC *et nous rassemblons les expressions logiques définies pour les modèles* RBAC *au long des différents chapitres. Cette théorie sert de base pour les traces proposées dans l'annexe suivante.* ◁

Plan du chapitre

B.1 Symboles utilisés

LA thèse a repris et défini plusieurs symboles, nous les rassemblons dans cette section. Les symboles sont classés en trois catégories selon qu'ils sont utilisés pour :

- la définition ensembliste du contrôle d'accès,
- la définition de la structuration relationnelle proposée,
- la représentation des modèles avec la structuration relationnelle.

B.1.1 Définitions ensemblistes du contrôle d'accès

Les symboles que nous regroupons dans le tableau B.1 sont ceux issus des propositions existantes étudiées dans l'état de l'art du chapitre 2 que nous reprennons par la suite. Le tableau B.1 synthétise les symboles pour lesquels nous avons utilisé une police calligraphiée ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{O}$ ...). Ces symboles concernent principalement les modèles RBAC.

Symbole	Description	Fact.[1]	Stat.[2]	Réf.
$\mathcal{U}$	utilisateurs	×	×	2.1.3
$\mathcal{S}$	sujets (ou session)	×		
$\mathcal{O}$	objets	×	×	
$\mathcal{A}$	actions	×	×	
$\mathcal{R}$	rôles	×	×	2.3.1
$\mathcal{P} \subseteq \mathcal{O} \times \mathcal{A}$	permission	×	×	
$\mathcal{URA} \subseteq \mathcal{U} \times \mathcal{R}$	affectation	×	×	
$\mathcal{PRA} \subseteq \mathcal{R} \times \mathcal{P}$	affectation	×	×	
$\mathcal{SU} \subseteq \mathcal{S} \times \mathcal{U}$	association	×		
$\mathcal{SR} \subseteq \mathcal{S} \times \mathcal{R}$	association	×		
$\mathcal{RH} \subseteq \mathcal{R} \times \mathcal{R}$	hiérarchie ($\succeq$, $\preceq$)		×	
$auth\_users : \mathcal{R} \rightarrow 2^{\mathcal{U}}$	utilisateurs autorisés		×	2.3.2
$auth\_perms : \mathcal{R} \rightarrow 2^{\mathcal{P}}$	permissions autorisés		×	
$assigned\_users : \mathcal{R} \rightarrow 2^{\mathcal{U}}$	utilisateurs assignés		×	
$assigned\_perms : \mathcal{R} \rightarrow 2^{\mathcal{P}}$	permissions assignées		×	
$\mathcal{MER} \subseteq \mathcal{R} \times \mathcal{R}$	exclusion entre rôles		×	2.5.1
$\mathcal{ACCESS} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{O}$	triplet d'autorisation			4.1.1
$\mathcal{F} : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow \{vrai, faux\}$	moniteur			4.1.2

TAB. B.1 – Symboles ensemblistes du contrôle d'accès

B.1.2 Structuration relationnelle

Nous synthétisons ici les différents symboles utilisés pour la définition de la struture relationnelle sur laquelle nous développons la thèse. Les symboles sont présentés en section 4.1.3. Pour un symbole donné, nous proposons son inteprétation comme composant d'un modèle de contrôle d'accès (en *italique*) et son nom consacré dans [Abiteboul95].

Symbole	Description
$\mathbf{AC} = (sch, P, \Sigma)$	*modèle de contrôle d'accès* base de données déductive
$sch = edb \cup idb$	schéma
edb	schéma en extension (*noyau*)
idb	schéma en intention
P	*principes du modèle* règles de déduction
$\Sigma = \Sigma_{edb} \cup \Sigma_{idb}$	*propriétés du modèle* dépendances de données
Σ_{edb}	*propriétés de edb*
Σ_{idb}	*propriétés de idb*
$\mathbf{I} = \mathbf{I_s} \cup \mathbf{I_d}$	*politique factuelle* (en extension) instance de *edb*
$\mathbf{I_s}$	*politique factuelle statique*
$\mathbf{I_d}$	*politique factuelle dynamique*
$\mathbf{I'} = \mathbf{I'_s} \cup \mathbf{I'_d}$	*politique déduite* (en intention) instance de *sch* plus petit point fixe unique de P
$\mathbf{I'_s}$	*politique déduite statique*
$\mathbf{I'_d}$	*politique déduite dynamique*

TAB. B.2 – Structuration relationnelle

B.1.3 Prédicats du contrôle d'accès

Une fois la structuration relationnelle définie, nous avons modélisé le contrôle d'accès sous forme logique dans le chapitre 4. Nous rassemblons ici les principaux symboles de prédicats utilisés dans les schémas. Nous notons Nom/n pour indiquer que le symbole de prédicat Nom est d'arité n.

Symbole	Description	Fact.	Stat.
$User/1$	utilisateur ($\mathcal{U}$)	×	×
$Sujet/1$	sujet ou session ($\mathcal{S}$)	×	
$Action/1$	action ($\mathcal{A}$)	×	×
$Objet/1$	objet ($\mathcal{O}$)	×	×
$Role/1$	rôle RBAC ($\mathcal{R}$)	×	×
$Label/1$	label MAC	×	×
$Habilite/2$	rôles-utilisateurs ($\mathcal{URA}$)	×	×
$Affecte/3$	rôles-permissions ($\mathcal{PRA}$)	×	×
$Représente/2$	sessions-utilisateur ($\mathcal{SU}$)	×	
$Endosse/2$	sessions-rôles ($\mathcal{SR}$)	×	
$Accred/2$	label-utilisateur	×	×
$Classif/2$	label-objet	×	×
$Perm/3$	permission, actions-objets		×
$AuthUsers/2$	rôles-utilisateurs		×
$AssignedUsers/2$	rôles-utilisateurs		×
$AuthPerms/2$	rôles-permissions		×
$AssignedPerms/2$	rôles-permissions		×
$Accès/3$	triplet fondamental ($\mathcal{ACCESS}$)		
$Statique/3$	orienté utilisateurs statique		×
$Dynamique/3$	orienté utilisateurs dynamique		
$Domine_C$	héritage en extension sur C	×	×
$Hérite_C$	héritage en intention sur C ($\mathcal{RH}$)		×
$Séparation_C$	exclusion en extension sur C	×	×
$Exclusion_C$	exclusion en intention sur C ($\mathcal{MER}$)		×

TAB. B.3 – Symboles de prédicats

B.2 Modèles de contrôle d'accès

B.2.1 MAC

Les modèles MAC sont des modèles structurés simples et efficaces basés sur deux règles de déduction permettant de dériver le triplet *Accès*, section 2.2.2. Les modèles MAC sont fondés sur des labels hiérarchisés en ordre total ou en treillis. Le tableau B.4 présente les expressions logiques représentant successivement :

- utilisateurs et sujets sont mis en bijection (ρ),
- les labels MACsont hiérarchisés (α),
- la hiérarchie des labels est un treillis (β),
- chaque sujet et objet est associé à un unique label (γ),
- on dérive les actions autorisées à partir de la hiérarchie. On note le statut particulier des deux actions `read` et `write` dans ces dépendances (δ),
- on ne peut pas obtenir de permissions sans label (λ).

	Type	Expression logique
ρ_1	TTGD	$User(U) \rightarrow Sujet(S)$
ρ_2	TTGD	$Sujet(S) \rightarrow User(U)$
α_1	TTGD	$Label(ID) \rightarrow Domine_L(ID, ID)$
α_2	TTGD	$Domine_L(ID_1, ID_2) \rightarrow Hérite_L(ID_1, ID_2)$
α_3	TTGD	$Domine_L(ID_1, ID_2), Hérite_L(ID_2, ID_3) \rightarrow Hérite_L(ID_1, ID_3)$
α_4	EGD	$Hérite_L(ID_1, ID_2), Hérite_L(ID_2, ID_1) \rightarrow ID_1 = ID_2$
β_1	TGD	$Label(ID_1), Label(ID_2) \rightarrow \exists ID_\bot Hérite_L(ID_1, ID_\bot), Hérite_L(ID_2, ID_\bot)$
β_1	TGD	$Label(ID_1), Label(ID_2) \rightarrow \exists ID_\top\ Hérite_L(ID_\top, ID_1), Hérite_L(ID_\top, ID_2)$
γ_1	EGD	$Accred(S, L_1), Accred(S, L_2) \rightarrow L_1 = L_2$
γ_2	EGD	$Classif(O, L_1), Classif(O, L_2) \rightarrow L_1 = L_2$
δ_1	TTGD	$Accred(S, L_1), Hérite_L(L_1, L_2), Classif(O, L_2) \rightarrow Accès(S, \texttt{read}, O)$
δ_2	TTGD	$Accred(S, L_1), Hérite_L(L_2, L_1), Classif(O, L_2) \rightarrow Accès(S, \texttt{write}, O)$
λ_1	TGD	$Accès(S, \texttt{read}, O) \rightarrow Accred(S, L_1), Hérite_L(L_1, L_2), Classif(O, L_2)$
λ_2	TGD	$Accès(S, \texttt{write}, O) \rightarrow Accred(S, L_1), Hérite_L(L_2, L_1), Classif(O, L_2)$

TAB. B.4 – Théorie logique des modèles MAC en treillis

Soit T la théorie logique du tableau B.4. À l'aide d'un *chase*, on peut dériver le théorème $T \models Accès(S, \texttt{read}, O), Accès(S, \texttt{write}, O) \rightarrow \exists L\ Accred(S, L), Classif(O, L)$. Les étapes d'une telle dérivation sont :

1. supposons un sujet `s` et un objet `o` tels que $Accès(\texttt{s}, \texttt{read}, \texttt{o})$, $Accès(\texttt{s}, \texttt{write}, \texttt{o})$,
2. par application de λ_1 on dérive $Accred(\texttt{s}, \texttt{l}_1)$, $Hérite_L(\texttt{l}_1, \texttt{l}_2)$ et $Classif(\texttt{o}, \texttt{l}_2)$,
3. par application de λ_2 on dérive $Accred(\texttt{s}, \texttt{l}'_1)$, $Hérite_L(\texttt{l}'_2, \texttt{l}'_1)$ et $Classif(\texttt{o}, \texttt{l}'_2)$,

4. par application de γ_1 on dérive $\mathtt{l}_1 = \mathtt{l}'_1$ (on identifie les symboles),
5. par application de γ_2 on dérive $\mathtt{l}_2 = \mathtt{l}'_2$ (on identifie les symboles),
6. $\exists L\ Accred(S, L), Classif(O, L)$ est vérifiée avec $Accred(\mathtt{s}, \mathtt{l}_1)$ et $Classif(\mathtt{o}, \mathtt{l}_2)$.

B.2.2 RBAC

La famille des modèles RBAC a reçu une attention particulière, c'est en effet une des principales sources qui a conduit à l'élaboration des multiples modèles que nous pouvons désormais rencontrer. De plus, de nombreuses extensions et formalisations indépendantes de ces modèles ont été proposées, chapitre 2.

Au long de la thèse, nous avons très souvent fait référence aux modèles « officiels » du standard RBAC, c'est-à-dire les modèles de référence RBAC_0, RBAC_1, RBAC_2 et RBAC_3 [Ferraiolo03b]. Les tableaux B.5 et B.6 synthétisent les expressions logiques que nous avons développées dans la thèse. Ces formules sont regroupées selon les sections où elles apparaissent.

Une légende verticale indique à quel modèle le plus bas dans la famille ces formules s'appliquent. Une formule d'un modèle est valide pour les modèles qui s'appuient dessus, figure2.3. Le premier tableau s'applique aux modèles RBAC_0 (noyau, principes fondamentaux) et RBAC_1 (hiérarchie). Le second tableau concerne les modèles RBAC_2 (contraintes) et RBAC_3 (contraintes, hiérarchie et contraintes sur hiérarchie).

		Type	Expression logique
			section 3.4.4 et section 4.2.3
$\mathbf{RBAC}_0$	σ_0	TGD	$Endosse(S,R) \rightarrow \exists U\ Représente(S,U)$
	σ_1	FD	$Représente(S,U_1), Représente(S,U_2) \rightarrow U_1 = U_2$
	σ_2	TTGD	$Représente(S,U), Endosse(S,R) \rightarrow Habilite(U,R)$
			section 3.4.4 et section 4.3.3
$\mathbf{RBAC}_0$	σ_3	TGD	$Statique(U,A,O) \rightarrow \exists R\ Habilite(U,R), Affecte(R,A,O)$
	σ_4	TGD	$Accès(S,A,O) \rightarrow \exists R\ Endosse(S,R), Affecte(R,A,O)$
	σ_5	TGD	$Dynamique(U,A,O) \rightarrow \exists S\ Représente(S,U), Statique(U,A,O)$
			section 3.3.3 et section 4.3
$\mathbf{RBAC}_0$	τ_1	TTGD	$Endosse(S,R), Affecte(R,A,O) \rightarrow Accès(S,A,O)$
	τ_2	TTGD	$Habilite(U,R), Affecte(R,A,O) \rightarrow Statique(U,A,O)$
	τ_3	TTGD	$Représente(S,U), Accès(S,A,O) \rightarrow Dynamique(U,A,O)$
			section 3.3.3.2 et section 4.3.4
$\mathbf{RBAC}_1$	ρ_0	TTGD	$Domine(R_1,R_2) \rightarrow Hérite(R_1,R_2)$
	ρ_1	TTGD	$Domine(R_1,R_2), Hérite(R_2,R_3) \rightarrow Hérite(R_1,R_3)$
	ρ_2	TTGD	$Role(R) \rightarrow Domine(R,R)$
	ρ'	EGD	$Hérite(R_1,R_2), Hérite(R_2,R_1) \rightarrow R_2 = R_1$
			section 4.3, section 4.3.4 et section 5.5.4.3
$\mathbf{RBAC}_1$	α_1	TTGD	$Habilite(U,R_1), Domine(R_1,R_2) \rightarrow AuthUsers(R_2,U)$
	α_2	TTGD	$Affecte(R_2,A,O), Domine(R_1,R_2) \rightarrow AuthPerms(R_1,A,O)$
	α_3	TTGD	$Endosse(S,R), AuthPerms(R,A,O) \rightarrow Accès(S,A,O)$
	α_4	TTGD	$Habilite(U,R), AuthPerms(R,A,O) \rightarrow Statique(U,A,O)$
	α_5	TTGD	$Représente(S,U), Accès(S,A,O) \rightarrow Dynamique(U,A,O)$
			section 4.3.3
$\mathbf{RBAC}_1$	β_1	TGD	$AuthUsers(R_2,U) \rightarrow Habilite(U,R_1), Domine(R_1,R_2)$
	β_2	TGD	$AuthPerms(R_1,A,O) \rightarrow Affecte(R_2,A,O), Domine(R_1,R_2)$
	β_3	TGD	$Accès(S,A,O) \rightarrow Endosse(S,R), AuthPerms(R,A,O)$
	β_4	TGD	$Statique(U,A,O) \rightarrow Habilite(U,R), AuthPerms(R,A,O)$
	β_5	TGD	$Dynamique(U,A,O) \rightarrow Représente(S,U),$ $Endosse(S,R), AuthPerms(R,A,O)$

TAB. B.5 – Théorie logique des modèles RBAC_1 et RBAC_2

		Type	Expression logique
			section 3.4.6 et section 4.5.2
$\mathbf{RBAC}_2$	γ_0	TTGD	$Séparation(R_1, R_2) \rightarrow Exclusion(R_1, R_2)$
	γ_s	TTGD	$Exclusion(R_1, R_2) \rightarrow Exclusion(R_2, R_1)$
	γ_r	NGD	$Exclusion(R, R) \rightarrow \bot$
			section 3.4.6 et section 4.5.2, choisir une des cinq sémantiques
$\mathbf{RBAC}_2$	γ_1	NGD	$Exclusion(R_1, R_2), Habilite(U, R_1), Habilite(U, R_2) \rightarrow \bot$
	γ_2	NGD	$Exclusion(R_1, R_2), Endosse(S, R_1), Endosse(S, R_2) \rightarrow \bot$
	γ_3	NGD	$Exclusion(R_1, R_2), Affecte(R_1, A, O), Affecte(R_2, A, O) \rightarrow \bot$
	γ_4	NGD	$Exclusion(R_1, R_2), Affecte(R_1, A_1, O), Affecte(R_2, A_2, O) \rightarrow \bot$
	γ_5	NGD	$Exclusion(R_1, R_2), Endosse(S, R_1), Endosse(S, R_2),$ $Représente(S, U), Représente(S, U) \rightarrow \bot$
			section 4.5.3 et section 4.5.5
$\mathbf{RBAC}_3$	λ_0	TTGD	$Exclusion(R_1, R_2), Hérite(R, R_1) \rightarrow Exclusion(R, R_2)$
	λ_t	TTGD	$Dépend(R_1, R_2), Nécessite(R_2, R_3) \rightarrow Nécessite(R_1, R_3)$
	λ_s	TGD	$Nécessite(Sub, Sup), Domine(Sup, Sub),$ $Habilite(U, Sub) \rightarrow \exists U' Habilite(U', Sub)$
			formules redondantes, section 5.2.5
$\mathbf{RBAC}_3$	λ_1	NGD	$Exclusion(R_1, R_2), Hérite(R, R_1), Hérite(R, R_2) \rightarrow \bot$
	λ_2	NGD	$Hérite(R_1, R_2), Exclusion(R_1, R_2) \rightarrow \bot$
	λ_d	TTGD	$Dynamique(U, A, O) \rightarrow Statique(U, A, O)$

TAB. B.6 – Théorie logique des modèles RBAC_2 et RBAC_3

Let us change our traditional attitude to the construction of programs.
Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

Donald Knuth

C

Traces d'inférence

▷ *Cette annexe propose deux traces d'inférence commentées obtenues avec la* LIBDEPENDENCIES. *Il s'agit d'illustrer la preuve syntaxique de propriétés des modèles de contrôle d'accès. Nous décrivons les résultats obtenus avec notre prototype à l'aide de la théorie logique des modèles* RBAC *présentée dans l'annexe précédente.* ◁

Plan du chapitre

AFIN d'illustrer l'intérêt de l'utilisaiton des procédures de preuves pour la vérification de modèles de contrôle d'accès, nous proposons deux traces, qui reprennent deux exemples de preuve présentés dans la thèse. Les dépendances auxquelles nous faisont référence dans les traces sont celles du tableau B.6. L'algorithme utilisé est l'extension par Maher du *chase* de Beeri et Vardi, section 3.4.2.2, [Beeri84, Maher96]. Cet algorithme est décrit par la figure 3.2.

C.1 Inclusion des triplets d'autorisation

Cette première trace correspond à la preuve fournie par la LIBDEPENDENCIES décrite en section 3.6 : l'inclusion des autorisations dynamiques dans les statiques pour le modèle RBAC$_1$. Cette propriété est modélisée par la dépendance σ de la forme $Dynamique(U, A, O) \rightarrow Statique(U, A, O)$. Par rapport à l'exemple déjà présenté, cette trace prend en compte la hiérarchie de rôle.

```
--------------------------------Tgds in base : 19--------------------------------
[0] (for all)[R,S] endosse(S,R)->(exist)[U] represente(S,U).
[1] (for all)[S,U1,U2] represente(S,U1),represente(S,U2)-> {(U1=U2)}.
[2] (for all)[R,S,U] represente(S,U),endosse(S,R)->habilite(U,R).
[3] (for all)[R1,R2] domine(R1,R2)->herite(R1,R2).
[4] (for all)[R1,R2,R3] domine(R1,R2),herite(R2,R3)->herite(R1,R3).
[5] (for all)[R] role(R)->domine(R,R).
[6] (for all)[R1,R2] herite(R1,R2),herite(R2,R1)-> {(R2=R1)}.
[7] (for all)[R1,R2,U] habilite(U,R1),herite(R1,R2)->authUsers(R2,U).
[8] (for all)[A,O,R,U] authPerms(R,A,O),habilite(U,R)->statique(U,A,O).
[9] (for all)[A,O,U] statique(U,A,O)->(exist)[R] authPerms(R,A,O),habilite(U,R).
[10] (for all)[A,O,R,S] authPerms(R,A,O),endosse(S,R)->acces(S,A,O).
[11] (for all)[A,O,S] acces(S,A,O)->(exist)[R] authPerms(R,A,O),endosse(S,R).
[12] (for all)[A,O,R,S,U] represente(S,U),endosse(S,R),authPerms(R,A,O)
       ->dynamique(U,A,O).
[13] (for all)[A,O,U] dynamique(U,A,O)->(exist)[R,S] represente(S,U),
       endosse(S,R),authPerms(R,A,O).
[14] (for all)[R1,R2] separation(R1,R2)->exclusion(R1,R2).
[15] (for all)[R1,R2] exclusion(R1,R2)->exclusion(R2,R1).
[16] (for all)[R] exclusion(R,R)->error(reflex) {(1=\=1)}.
[17] (for all)[R1,R2,U] exclusion(R1,R2),habilite(U,R1),habilite(U,R2)
       ->error(ssd) {(1=\=1)}.
[18] (for all)[R,R1,R2] exclusion(R1,R2),herite(R,R1)->exclusion(R,R2).
-------------------------------------------------------------------------------

Tuples :
[0] dynamique(_u_0,_a_0,_o_0); -1;

tgdGoal : (for all)[A,O,U] dynamique(U,A,O)->statique(U,A,O).
```

Supposons l'existence d'un tuple $Dynamique(\mathtt{u}_0, \mathtt{a}_0, \mathtt{o}_0)$. Soit en langage naturel, *supposons qu'un utilisateur dispose d'une autorisation dynamique*.

```
stepNumber : 1
++++++++++++++++++++++++++++++++++++++++
```

```
Treating [13]... '(for all)[A,O,U] dynamique(U,A,O)->(exist)[R,S]
        represente(S,U),endosse(S,R),authPerms(R,A,O).'
        Added to tuples: represente(_s_0,_u_0); 13;
        Added to tuples: endosse(_s_0,_r_0); 13;
        Added to tuples: authPerms(_r_0,_a_0,_o_0); 13;
        Added to activations: {A:_a_0,O:_o_0,U:_u_0}; 0;
...[13] treated
```

Par application de β_5 (dépendance `[13]`), il existe également un rôle $\mathtt{r}_0$ et une session $\mathtt{s}_0$ tels que *Endosse*$(\mathtt{s}_0, \mathtt{r}_0)$, *Représente*$(\mathtt{s}_0, \mathtt{u}_0)$ et *AuthPerms*$(\mathtt{r}_0, \mathtt{a}_0, \mathtt{o}_0)$. Soit en langage naturel, *on ne peut pas disposer d'une permission sans passer par un rôle*.

```
stepNumber : 2
++++++++++++++++++++++++++++++++++++++++
Treating [8]... '(for all)[A,O,R,U] authPerms(R,A,O),habilite(U,R)->statique(U,A,O).'
        Added to tuples: statique(_u_0,_a_0,_o_0); 8;
        Added to activations: {A:_a_0,O:_o_0,R:_r_0,U:_u_0}; 3,4;
...[8] treated

Treating [10]... '(for all)[A,O,R,S] authPerms(R,A,O),endosse(S,R)->acces(S,A,O).'
        Added to tuples: acces(_s_0,_a_0,_o_0); 10;
        Added to activations: {A:_a_0,O:_o_0,R:_r_0,S:_s_0}; 3,2;
...[10] treated
```

Comme il existe *AuthPerms*$(\mathtt{r}_0, \mathtt{a}_0, \mathtt{o}_0)$ et *Habilite*$(\mathtt{u}_0, \mathtt{r}_0)$, alors par application de α_4 (dépendance `[8]`), il existe *Statique*$(\mathtt{u}_0, \mathtt{a}_0, \mathtt{o}_0)$. Soit en langage naturel, *on dispose d'une autorisation statique par l'intermédiaire des rôles qu'on endosse*. Notons que par application de α_5 (dépendance `[10]`) on a également dérivé un théorème plus général : $\mathit{Dynamique}(U, A, O) \rightarrow \exists S\ \mathit{Statique}(U, A, O), \mathit{Accès}(S, A, O)$.

```
++++++++++++++++++++++++++++++++++++++++
No more tuples can be produced (FIX POINT), testing termination case:

Possible mappings :
        {A:_a_0,O:_o_0,U:_u_0}; 5;
        statique(_u_0,_a_0,_o_0); 8;

Termination case : the chase is finished and prove F|=g (PROOF)
--------------------------------------------------------------------------------
the chase is finished and prove F|=g (PROOF)
number of rules applied for closure F(l):4
this chase was :0.096241 seconds long
number of tuples generated:7

goal goalTgd: (for all)[A,O,U] dynamique(U,A,O)->statique(U,A,O).
number of answers: 1
final mapping:
[1] {A:_a_0,O:_o_0,U:_u_0}; 5;
```

Ainsi, si un tuple *Dynamique*$(\mathtt{u}_0, \mathtt{a}_0, \mathtt{o}_0)$, quel qu'il soit, existe dans la politique alors le tuple *Statique*$(\mathtt{u}_0, \mathtt{a}_0, \mathtt{o}_0)$ existe aussi, on a prouvé que $\{\beta_5, \alpha_4\} \models \sigma$.

Chaque valuation intermédiaire et la dépendance qui a été utilisée pour générer un tuple est enregistré par une `CActivation`. La trace pourrait être simplifiée en ne présentant que ses activations, c'est-à-dire l'arbre d'inférence partant de l'hypothèse de σ et aboutissant à sa conclusion.

C.2 Simplification des propriétés de l'exclusion mutuelle

En section 5.2.5 nous avons montré comment une procédure de preuve peut permettre de démontrer automatiquement certains résultats pour la simplification d'une théorie logique. La trace d'exécution du *chase* que nous proposons dans cette section est une preuve de $\{\gamma_r, \gamma_s, \lambda_0\} \models \lambda_1$. Ces expressions logiques sont la représentation des propriétés de l'exclusion mutuelle et de l'interaction de l'exclusion avec une hiérarchie définie sur le même concept. La preuve automtisée que nous proposons est inspirée d'un résultat de [Gavrila98].

```
--------------------------------Tgds in base : 4--------------------------------
[0] (for all)[R1,R2] separation(R1,R2)->exclusion(R1,R2).
[1] (for all)[R1,R2] exclusion(R1,R2)->exclusion(R2,R1).
[2] (for all)[R] exclusion(R,R)->error(reflex) {(1=\=1)}.
[3] (for all)[R,R1,R2] exclusion(R1,R2),herite(R,R1)->exclusion(R,R2).
--------------------------------------------------------------------------------
Tuples :
[0] exclusion(_r1_0,_r2_0); -1;
[1] herite(_r_0,_r1_0); -1;
[2] herite(_r_0,_r2_0); -1;

seed : exclusion(_r1_0,_r2_0),herite(_r_0,_r1_0),herite(_r_0,_r2_0)
goal :  {(1=\=1) }
tgdGoal : (for all)[R,R1,R2] exclusion(R1,R2),herite(R,R1),herite(R,R2)-> {(1=\=1)}.
```

Supposons l'existence des tuples $Exclusion(\mathtt{r_1}, \mathtt{r_2})$, $Hérite(\mathtt{r_0}, \mathtt{r_1})$ et $Hérite(\mathtt{r_0}, \mathtt{r_2})$.

```
stepNumber : 1
++++++++++++++++++++++++++++++++++++++++
Treating [1]... '(for all)[R1,R2] exclusion(R1,R2)->exclusion(R2,R1).'
        Added to tuples: exclusion(_r2_0,_r1_0); 1;
        Added to activations: {R1:_r1_0,R2:_r2_0}; 0;
...[1] treated

Treating [3]... '(for all)[R,R1,R2] exclusion(R1,R2),herite(R,R1)->exclusion(R,R2).'
        Added to tuples: exclusion(_r_0,_r2_0); 3;
        Added to activations: {R:_r_0,R1:_r1_0,R2:_r2_0}; 0,1;

        Added to tuples: exclusion(_r_0,_r1_0); 3;
        Added to activations: {R:_r_0,R1:_r2_0,R2:_r1_0}; 3,2;
...[3] treated
```

Par application de γ_s (dépendance `[1]`), on déduit $Exclusion(\mathtt{r_2}, \mathtt{r_1})$. Soit en langage naturel, *l'exclusion est symétrique*. Ensuite, par deux applications de λ_0 (dépen-

dance [3]), on déduit que $Exclusion(\mathtt{r}_0, \mathtt{r}_2)$ et $Exclusion(\mathtt{r}_0, \mathtt{r}_1)$. Soit en langage naturel, *l'exclusion se transmet par la relation d'héritage*.

```
stepNumber : 2
++++++++++++++++++++++++++++++++++++++
Treating [1]... '(for all)[R1,R2] exclusion(R1,R2)->exclusion(R2,R1).'
        Added to tuples: exclusion(_r2_0,_r_0); 1;
        Added to activations: {R1:_r_0,R2:_r2_0}; 4;

        Added to tuples: exclusion(_r1_0,_r_0); 1;
        Added to activations: {R1:_r_0,R2:_r1_0}; 5;
...[1] treated

Treating [3]... '(for all)[R,R1,R2] exclusion(R1,R2),herite(R,R1)->exclusion(R,R2).'
        Added to tuples: exclusion(_r_0,_r_0); 3;
        Added to activations: {R:_r_0,R1:_r2_0,R2:_r_0}; 6,2;
...[3] treated
```

Par application de γ_s (dépendance [1]), on déduit $Exclusion(\mathtt{r}_2, \mathtt{r}_0)$ et que $Exclusion(\mathtt{r}_1, \mathtt{r}_0)$. Ensuite, par application de λ_0 (dépendance [3]), on déduit que $Exclusion(\mathtt{r}_0, \mathtt{r}_0)$.

```
stepNumber : 3
++++++++++++++++++++++++++++++++++++++
Treating [2]... '(for all)[R] exclusion(R,R)->error(reflex) {(1=\=1)}.'
        Added to tuples: error(reflex); 2;
        Added to store: (1=\=1)
        Added to activations: {R:_r_0}; 8;
...[2] treated
```

Par application de γ_r (dépendance [2]), soit en langage naturel, *l'exclusion est irréflexive* , on déduit $\perp$ représenté ici par l'antilogie $1 \neq 1$ stockée dans le magasin de contraintes.

```
-------------------------------------------------------------------------------
there is an inconsistency in the constraint store F|=g vacuously (VACUOUSLY)
number of rules applied for closure F(l):7
this chase was :0.088197 seconds long
number of tuples generated:10
```

Le *chase* pour les dépendances génératrices de tuples contraintes se termine en ayant dérivé la conclusion à prouver : une antilogie. On a prouvé $\{\gamma_r, \gamma_s, \lambda_0\} \models \lambda_1$. Techniquement pour cette preuve, il s'agit du cas de terminaison numéro deux, provoqué par des contraintes incohérentes [Maher96]. Notons que nous avons également utilisé un prédicat *Error*/1. Il pourrait remplacer le *syntatic sugar* utilisé pour représenter $\perp$.

Bibliographie

[Abadi07] Martín Abadi. *Access Control in a Core Calculus of Dependency*. Electronic Notes in Theoretical Computer Science, vol. 172, pages 5–31, Elsevier, Amsterdam, Avril 2007. 46, 47, 50, 62, 63

[Abiteboul95] Serge Abiteboul, Richard Hull & Victor Vianu. Foundations of databases. Addison-Wesley, Boston, 1995. iii, 22, 46, 48, 65, 72, 73, 74, 81, 82, 85, 86, 88, 96, 139, 201, 206, 209, 219, 231, 232, 238

[Abou El Kalam03] Anas Abou El Kalam. *Politiques et modèles de sécurité pour les domaines de la santéet des affaires sociales*. Doctorat, Institut National Polytechnique de Toulouse, Toulouse, 2003. 13, 30, 32, 52, 63, 130

[Abrahams04] David Abrahams & Aleksey Gurtovoy. C++ template metaprogramming : Concepts, tools, and techniques from boost and beyond. Addison-Wesley Professional, Boston, 2004. 213

[Adi03] Kamel Adi, Amy Felty, Luigi Logrippo & Bernard Stepien. *Evaluation of Current Research in Firewall Analysis*. Rapport technique, The UofO LOTOS Research Group, 2003. 217

[Ahn00] Gail-Joon Ahn & Ravi S. Sandhu. *Role-based authorization constraints specification.* ACM Transactions on Information & System Security, vol. 3, no. 4, pages 207–226, ACM Press, New York, 2000. 39

[Al-Kahtani04] Mohammad A. Al-Kahtani & Ravi S. Sandhu. *Rule-Based RBAC with Negative Authorization*. In ACSAC'04 : 20th Annual Computer Security Applications Conference, pages 405–415, Washington, 2004. IEEE Computer Society. 46, 47

[Al-Shaer04] Ehab S. Al-Shaer & Hazem H. Hamed. *Discovery of Policy Anomalies in Distributed Firewalls.* In INFOCOM'04 : 23rd AnnualJoint Conference of the IEEE Computer and Communications Societies, volume 4, pages 2605–2616, Washington, 2004. IEEE Computer Society. 46, 217

[Alexandrescu01] Andrei Alexandrescu. Modern C++ Design : Generic Programming and Design Patterns Applied. Addison-Wesley, Boston, 2001. 183, 213

[Allen83] James F. Allen. *Maintaining Knowledge about Temporal Intervals.* Communications of the ACM, vol. 26, no. 11, pages 832–843, ACM Press, New York, 1983. 42

[Appel99] Andrew W. Appel & Edward W. Felten. *Proof-Carrying Authentication.* In CCS'99 : ACM Conference on Computer and Communications Security, Singapore, pages 52–62, New York, 1999. ACM Press. 48

[Arends05] R. Arends, R. Austein, M. Larson, D. Massey & S. Rose. *Protocol Modifications for the DNS Security Extensions*. RFC 4035, mar 2005. 50

[Arévalo07] Gabriela Arévalo, Anne Berry, Marianne Huchard, Guillaume Perrot & Alain Sigayret. *Performances of Galois Subhierarchy-building Algorithms.* In ICFCA'07 : 5th International Conference on Formal Concept Analysis, Clermont-Ferrand, volume 4390 of *LNCS*. Springer-Verlag, 2007. iv, 163, 191

[Atluri02] Vijayalakshmi Atluri & Avigdor Gal. *An authorization model for temporal and derived data : securing information portals*. ACM Transactions on Information & System Security, vol. 5, no. 1, pages 62–94, ACM Press, New York, 2002. 43, 45

[Baget06] Jean-François Baget & Eric Salvat. *Rules Dependencies in Backward Chaining of Conceptual Graphs Rules*. In Henrik Schärfe, Pascal Hitzler & Peter Øhrstrøm, editeurs, ICCS'06 : 14th International-Conference on Conceptual Structures, Aalborg, Denmark, volume 4068 of *Lecture Notes in Computer Science*, pages 102–116. Springer-Verlag, 2006. 204

[Barker02] Steve Barker. *Access Control for Deductive Databases by Logic Programming.* In Peter J. Stuckey, editeur, ICLP'02 : 18th International Conference on Logic Programming, Copenhagen,Denmark, volume 2401 of *Lecture Notes in Computer Science*, pages 54–69. Springer-Verlag, 2002. 97

[Barker03] Steve Barker & Peter J. Stuckey. *Flexible access control policy specification with constraint logic programming*. ACM Transactions on Information & System Security, vol. 6, no. 4, pages 501–546, ACM Press, New York, 2003. 27, 29, 34, 45, 51, 52, 63, 79, 97, 117, 139

[Bartal04] Yair Bartal, Alain J. Mayer, Kobbi Nissim & Avishai Wool. *Firmato : A novel firewall management toolkit*. ACM Transactions on Computing Systems, vol. 22, no. 4, pages 381–420, ACM Press, New York, 2004. 217

[Baudinet95] Marianne Baudinet, Jan Chomicki & Pierre Wolper. *Constraint-Generating Dependencies.* In Georg Gottlob & Moshe Y. Vardi, editeurs, ICDT'95 : 5th International Conference on Database Theory,

Prague, Czech Republic, volume 893 of *Lecture Notes in Computer Science*, pages 322–337. Springer-Verlag, 1995. 81, 83

[Beeri84] Catriel Beeri & Moshe Y. Vardi. *A Proof Procedure for Data Dependencies.* Journal of the ACM, vol. 31, no. 4, pages 718–741, ACM Press, New York, 1984. iii, 80, 81, 87, 88, 91, 185, 206, 247

[Bell75] D. Bell & L. LaPadula. *Secure Computer Systems : Unified Exposition and Multics Interpretation*. Rapport technique, 1975. 21, 22

[Benantar06] Messaoud Benantar, editeur. Access control systems - security, identity management and trust models. Springer-Verlag, 2006. 39, 40, 130, 134, 182

[Berry05] Anne Berry, Marianne Huchard, Ross M. McConnell, Alain Sigayret & Jeremy Spinrad. *Efficiently Computing a Linear Extension of the Sub-hierarchy of a Concept Lattice.* In ICFCA'05 : 3rd International Conference on Formal Concept Analysis, volume 3403 of *LNCS*, pages 208–222, 2005. 163

[Bertino99] Elisa Bertino, Elena Ferrari & Vijay Atluri. *The specification and enforcement of authorization constraints in workflow management systems*. ACM Transactions on Information & System Security, vol. 2, no. 1, pages 65–104, ACM Press, New York, 1999. 32

[Bertino01] Elisa Bertino, Piero A. Bonatti & Elena Ferrari. *TRBAC : A temporal role-based access control model.* ACM Transactions on Information & System Security, vol. 4, no. 3, pages 191–233, ACM Press, New York, 2001. 43, 45, 76

[Bertino03] Elisa Bertino, Barbara Catania, Elena Ferrari & Paolo Perlasca. *A logical framework for reasoning about access control models*. ACM Transactions on Information & System Security, vol. 6, no. 1, pages 71–127, ACM Press, New York, 2003. 46, 47, 51, 52, 63, 147

[Bertino05] Elisa Bertino & Ravi S. Sandhu. *Database Security-Concepts, Approaches, and Challenges*. IEEE Transactions on Dependable & Secure Computing, vol. 2, no. 1, pages 2–19, IEEE Computer Society, Washington, 2005. 178

[Bertossi06] Leopoldo E. Bertossi. *Consistent query answering in databases*. SIGMOD Record, vol. 35, no. 2, pages 68–76, ACM Press, New York, 2006. iv, 87, 206

[Besson06] Jérémy Besson, Céline Robardet & Jean-François Boulicaut. *Mining a New Fault-Tolerant Pattern Type as an Alternative to Formal Concept Discovery.* In ICCS'06 : 14th International Conference on Conceptual Structures, volume 4068 of *LNCS*, pages 144–157, 2006. 210

[Bhatti05] Rafae Bhatti, Basit Shafiq, Elisa Bertino, Arif Ghafoor & James Joshi. *X-GTRBAC Admin : A decentralized administration model for*

enterprise-wide access control. ACM Transactions on Information & System Security, vol. 8, no. 4, pages 388–423, ACM Press, New York, 2005. 35, 43

[Blaze99] M. Blaze, J. Feigenbaum, J. Ioannidis & A. Keromytis. *The KeyNote Trust-Management System Version 2*. RFC 2704, Internet Engineering Task Force, September 1999. 49

[Brewer89] David F.C. Brewer & Michael J. Nash. *The chinese wall security policy*. In IEEE Symposium on Security and Privacy, pages 206–214. IEEE Computer Society, 1989. 202

[Calì04] Andrea Calì. *Reasoning in Data Integration Systems : why LAV and GAV are Siblings.* In Maristella Agosti, Nicoletta Dessì & Fabio A. Schreiber, editeurs, SEBD'04 : 12th Italian Symposium on Advanced DatabaseSystems, Cagliari, Italy, pages 282–289, 2004. 170, 206, 207

[Calvanese94] Diego Calvanese & Maurizio Lenzerini. *On the Interaction Between ISA and Cardinality Constraints.* In 10th International Conference on Data Engineering, Houston, Texas, pages 204–213. IEEE Computer Society, 1994. 122

[CERT/CC02] CERT/CC. *Test the firewall system*. Security improvement module, 2002. 215

[CERT/CC05] CERT/CC, US Secret Service & CSO magazine. *E-CrimeWatch Survey*. Rapport technique, 2005. 215

[Chein92] Michel Chein & Marie-Laure Mugnier. *Conceptual Graphs : Fundamental Notions*. Revue d'Intelligence Artificielle, vol. 6, no. 4, pages 365–406, Hermes, Cachan, 1992. 144, 145

[Cholvy97] Laurence Cholvy & Frédéric Cuppens. *Analyzing consistency of security policies*. In SP'97 : IEEE Symposium on Security and Privacy, page 103, Washington, DC, USA, 1997. IEEE Computer Society. 52

[Chomicki05] Jan Chomicki & Jerzy Marcinkowski. *Minimal-change integrity maintenance using tuple deletions*. Information & Computation, vol. 197, no. 1-2, pages 90–121, Academic Press, Duluth, Minnesota, 2005. iv, 87, 206

[Chomicki07] Jan Chomicki. *Consistent Query Answering : Five Easy Pieces.* In Thomas Schwentick & Dan Suciu, editeurs, ICDT'07 : 11th International Conference on Database Theory, Barcelona, Spain, volume 4353 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, 2007. 87

[Christian95] Holzbaur Christian. *clp(q,r) Manual Rev. 1.3.2*. OEFAI, Austrian Research Institute for Artificial Intelligence, Vienna, 1995. 184

[Christiansen06] Henning Christiansen & Davide Martinenghi. *On Simplification of Database Integrity Constraints*. Fundamenta Informaticae, vol. 71, no. 4, pages 371–417, Polish Mathematical Society, Warszawa, 2006. iv, 87, 210

[Clark87] D. D. Clark & D. R. Wilson. *A Comparison of Commercial and Military Computer Security Policies*. In IEEE Symposium on Security and Privacy, pages 184–195, 1987. 32, 39

[Clarke01] Dwaine E. Clarke, Jean-Emile Elien, Carl M. Ellison, Matt Fredette, Alexander Morcos & Ronald L. Rivest. *Certificate Chain Discovery in SPKI/SDSI*. Journal of Computer Security, vol. 9, no. 4, pages 285–322, IOS Press, Amsterdam, 2001. 49

[Codd70] E. F. Codd. *A relational model of data for large shared data banks*. Communications of the ACM, vol. 13, no. 6, pages 377–387, ACM Press, New York, 1970. 80

[Coulondre98] Stéphane Coulondre & Eric Salvat. *Piece Resolution : Towards Larger Perspectives*. In Marie-Laure Mugnier & Michel Chein, editeurs, ICCS'98 : 6th International Conference on Conceptual Structures, Montpellier, France, volume 1453 of *Lecture Notes in Computer Science*, pages 179–193. Springer-Verlag, 1998. 144

[Coulondre03] Stéphane Coulondre. *A top-down proof procedure for generalized data dependencies*. Acta Informatica, vol. 39, no. 1, pages 1–29, Springer-Verlag, Berlin, 2003. 92, 144, 185

[Covington01] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dey, Mustaque Ahamad & Gregory D. Abowd. *Securing context-aware applications using environment roles.* In SACMAT'01 : 6th ACM Symposium on Access Control Models and Technologies, pages 10–20, 2001. 29, 34, 44

[Coyne96] Edward J. Coyne. *Role engineering*. In RBAC'95 : 1st ACM Workshop on Role-based access control, page 4, New York, 1996. ACM Press. 36, 154

[Crampton03a] Jason Crampton. *Specifying and enforcing constraints in role-based access control*. In SACMAT'03 : 8th ACM Symposium on Access Control Models and Technologies, pages 43–50. ACM Press, 2003. 40, 95, 154, 162, 208

[Crampton03b] Jason Crampton & George Loizou. *Administrative scope : A foundation for role-based administrative models*. ACM Transactions on Information & System Security, vol. 6, no. 2, pages 201–231, ACM Press, New York, 2003. 35

[Crampton05] Jason Crampton. *Understanding and developing role-based administrative models.* In Vijay Atluri, Catherine Meadows & Ari Juels, edi-

teurs, CCS'05 : 12th ACM Conference on Computer and CommunicationsSecurity, Alexandria, Virginia, pages 158–167. ACM Press, 2005. 35, 36

[Crook03] Robert Crook, Darrel C. Ince & Bashar Nuseibeh. *Modelling access policies using roles in requirements engineering*. Information & Software Technology, vol. 45, no. 14, pages 979–991, Elsevier, Amsterdam, 2003. 118

[Cuppens99] Frédéric Cuppens & Alban Gabillon. *Logical Foundations of Multilevel Databases*. Data & Knowledge Engineering, vol. 29, no. 3, pages 259–291, Elsevier, Amsterdam, 1999. 22

[Cuppens00] Frédéric Cuppens. *Modélisation formelle de la sécurité des systèmes d'informations, Habilitation à Diriger les Recherches*. Doctorat, Université Paul Sabatier, Toulouse, 2000. 22

[Cuppens03] Frédéric Cuppens & Alexandre Miège. *Modelling Contexts in the Or-BAC Model.* In ACSAC'03 : 19th Annual Computer Security Applications Conference, Las Vegas, Nevada, pages 416–427. IEEE Computer Society, 2003. 30, 44

[Cuppens04a] Frédéric Cuppens, Nora Cuppens-Boulahia & Alexandre Miège. *Inheritance hierachies in the Or-BAC Model and Application in a network environnement*. In 2nd Foundation of Computer Security Workshop, July 2004. 30

[Cuppens04b] Frédéric Cuppens & Alexandre Miège. *AdOrBAC : an administration model for Or-BAC.* Computer Systems : Science and Engineering, vol. 19, no. 3, CRL Publishing, Leicester, 2004. 30, 35

[Damiani07] Maria Luisa Damiani, Elisa Bertino, Barbara Catania & Paolo Perlasca. *GEO-RBAC : A spatially aware RBAC.* ACM Transactions on Information & System Security, vol. 10, no. 1, ACM Press, New York, 2007. 27, 29, 34, 44, 45, 74, 76

[Dehornoy07] Patrick Dehornoy. Logique et théorie des ensembles. Notes de cours FIMFA ENS, 2007. 206, 219

[Denning76] Dorothy E. Denning. *A lattice model of secure information flow*. Communications of the ACM, vol. 19, no. 5, pages 236–243, ACM Press, New York, 1976. 21

[Deswarte03] Yves Deswarte & Ludovic Mé, editeurs. Sécurité des réseaux et systèmes répartis. Hermes, Cachan, Oct 2003. 13, 130

[Deswarte04] Yves Deswarte & Anas Abou El Kalam. *Modèle de sécurité pour le secteur de la santé*. Technique et Science Informatiques, vol. 23, no. 3, pages 291–321, Hermes, Cachan, 2004. 18, 186

[DeTreville02] John DeTreville. *Binder, a Logic-Based Security Language*. In SP'02 : IEEE Symposium on Security and Privacy, page 105, Washington, 2002. IEEE Computer Society. 46, 47, 50, 63

[Dey01] Anind K. Dey. *Understanding and Using Context*. Personal Ubiquitous Comput., vol. 5, no. 1, pages 4–7, Springer-Verlag, London, 2001. 41

[Egenhofer91] Max J. Egenhofer. *Reasoning about Binary Topological Relations.* In Oliver Günther & Hans-Jörg Schek, editeurs, SSD'91 : 2nd International Symposium Advances in Spatial Databases, Zürich, Switzerland, volume 525 of *Lecture Notes in Computer Science*, pages 143–160. Springer-Verlag, 1991. 42

[Epstein02] Pete E. Epstein. *Engineering of Role/Permissions assignments*. Doctorat, Fairfax University, VA, USA, 2002. 154, 155, 210

[Eronen01] Pasi Eronen & Jukka Zitting. *An expert system for analyzing firewall rules*. In NordSec'01 : 6th Nordic Workshop on Secure IT Systems, pages 100–107, Copenhagen, 2001. Technical University of Denmark. Technical Report IMM-TR-2001-14. 217

[Fagin86] Ronald Fagin & Moshe Y. Vardi. *The theory of data dependencies : a survey*. In American Mathematical Society, editeur, Symposia in Applied Mathematics, volume 34 of *Mathematics of Information Processing*, pages 19–72, 1986. 57, 81

[Fagin03] Ronald Fagin, Phokion G. Kolaitis & Lucian Popa. *Data exchange : getting to the core.* In PODS'03 : 22nd ACM SIGACT-SIGMOD-SIGART Symposiumon Principles of Database Systems, San Diego, California, pages 90–101. ACM Press, 2003. 170

[Fagin05] Ronald Fagin, Phokion G. Kolaitis & Lucian Popa. *Data exchange : getting to the core.* ACM Transaction on Database Systems, vol. 30, no. 1, pages 174–210, ACM Press, New York, 2005. 206

[Fagin06] Ronald Fagin. *Inverting schema mappings.* In Stijn Vansummeren, editeur, PODS'06 : 25th ACM SIGACT-SIGMOD-SIGART Symposiumon Principles of Database Systems, Chicago, Illinois, pages 50–59. ACM Press, 2006. iv, 85, 87, 170, 206, 207

[Ferraiolo01] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, Richard D. Kuhn & Ramaswamy Chandramouli. *Proposed NIST standard for Role-Based Access Control*. ACM Transactions on Information & System Security, vol. 4, no. 3, pages 224–274, ACM Press, New York, 2001. 25, 34, 139

[Ferraiolo03a] David F. Ferraiolo, R. Chandramouli, Gail-Joon Ahn & Serban I. Gavrila. *The role control center : features and case studies*. In SACMAT'03 : 8th ACM Symposium on Access Control Models and Technologies, pages 12–20, New York, NY, USA, 2003. ACM Press. 37, 46, 143

[Ferraiolo03b] David F. Ferraiolo, Richard D. Kuhn & Ramaswamy Chandramouli. Role-based access control. Artech House Publishers, 2003. ii, 24, 25, 27, 28, 29, 34, 35, 39, 40, 51, 56, 70, 71, 77, 105, 117, 130, 137, 138, 139, 154, 162, 204, 241

[Ferraiolo07] David F. Ferraiolo, Richard D. Kuhn & Ravi S. Sandhu. *RBAC Standard Rationale : Comments on "A Critique of the ANSI Standard on Role-Based Access Control"*. IEEE Security and Privacy, vol. 5, no. 6, pages 51–53, IEEE Computer Society, Los Alamitos, California, 2007. 28

[Frantzen03] Lars Frantzen. Approaches for analysing and comparing packet filtering in firewalls. Master, Technical University of Berlin, 2003. 217

[Gallaher02] Michael P. Gallaher, Alan C. O'Connor & Brian Kropp. *The Economic Impact of Role-Based Access Control*. Rapport technique, Planning report 02-1, National Institute of Standards and Technology, 2002. 154

[Ganter97] Bernhard Ganter & Rudolf Wille. Formal concept analysis : Mathematical foundations. Springer-Verlag, Secaucus, New-Jersey, 1997. Translator-C. Franzke. 211

[Gavrila98] Serban I. Gavrila & John F. Barkley. *Formal Specification for Role Based Access Control User/Role and Role/Role Relationship Management*. In ACM Workshop on RBAC, pages 81–90, 1998. 27, 35, 39, 40, 117, 134, 135, 136, 182, 249

[Geerts04] Floris Geerts, Bart Goethals & Taneli Mielikäinen. *Tiling Databases*. In Einoshin Suzuki & Setsuo Arikawa, editeurs, DS'04 : 7th International Conference on Discovery Science, Padova, Italy, volume 3245 of *Lecture Notes in Computer Science*, pages 278–289. Springer-Verlag, 2004. 192, 210

[Georgiadis01] Christos K. Georgiadis, Ioannis Mavridis, George Pangalos & Roshan K. Thomas. *Flexible team-based access control using contexts*. In SACMAT '01 : Proceedings of the sixth ACM symposium on Access control models and technologies, pages 21–27, New York, NY, USA, 2001. ACM Press. 44

[Gligor98] Virgil D. Gligor, Serban I. Gavrila & David F. Ferraiolo. *On the Formal Definition of Separation-of-Duty Policies and their Composition*. In 1998 Symposium on Security and Privacy, pages 172–183, Oakland, CA, 1998. IEEE Computer Society Press. 40, 95, 117

[GMSIH03] GMSIH. *Politique de Sécurité des Systèmes d'Information des Etablissements de Santé - Politique d'autorisation*. Rapport technique, Groupement pour la Modernisation du Systèmes d'Information Hospitalier, Mars 2003. 34, 55, 186, 187

[Godfrey98] Parke Godfrey, John Grant, Jarek Gryz & Jack Minker. Integrity constraints : Semantics and applications., chapitre 9, pages 265–306. Kluwer, 1998. 81, 85

[Godin95] Robert Godin, Guy Mineau, Rokia Missaoui & Hafedh Mili. *Méthodes de Classification Conceptuelle Basées sur les Treillis de Galois et Applications*. Revue d'Intelligence Artificielle, vol. 9, no. 2, pages 105–137, Hermes, Cachan, 1995. 161

[Gouda04] Mohamed G. Gouda & Alex X. Liu. *Firewall design : consistency, completeness and compactness*. In ICDCS'04 : 24th International Conference On Distributed Computing Systems, pages 320–327, 2004. 46, 217

[Grabmeier02] Johannes Grabmeier & Andreas Rudolph. *Techniques of Cluster Algorithms in Data Mining*. Data Mining & Knowledge Discovery, vol. 6, no. 4, pages 303–360, Kluwer, Hingham, Massachusetts, 2002. 191

[Graham86] Marc H. Graham, Alberto O. Mendelzon & Moshe Y. Vardi. *Notions of dependency satisfaction*. Journal of the ACM, vol. 33, no. 1, pages 105–129, ACM Press, New York, 1986. 87, 140

[Greco92] Sergio Greco, Nicola Leone & Pasquale Rullo. *COMPLEX : An Object-Oriented Logic Programming System.* IEEE Transactions on Knowledge & Data Engineering, vol. 4, no. 4, pages 344–359, IEEE Computer Society, Los Alamitos, California, 1992. 51

[Gupta95] Ashish Gupta & Inderpal Singh Mumick. *Maintenance of Materialized Views : Problems, Techniques and Applications*. IEEE Quarterly Bulletin on Data Engineering ; Special Issue on Materialized Views and Data Warehousing, vol. 18, no. 2, pages 3–18, IEEE Computer Society, Los Alamitos, California, 1995. 85

[Hacene07] Mohamed Rouane Hacene, Michel Dao, Marianne Huchard & Petko Valtchev. *Analyse formelle de données relationnelles pour la réingénierie des modèles UML*. In Isabelle Borne, Xavier Crégut, Sophie Ebersold & Frédéric Migeon, editeurs, LMO'07 : Journées Langages et Modèles à Objets, Toulouse, pages 151–166. Hermès Lavoisier, 2007. 210

[Halpern03] Joseph Y. Halpern & Vicky Weissman. *Using First-Order Logic to Reason about Policies.* In CSFW'03 : 16th IEEE Computer Security Foundations Workshop, Pacific Grove, California, pages 187–201. IEEE Computer Society, 2003. 46, 47, 51, 63, 114, 128, 146, 201

[Hansen05] Frode Hansen & Vladimir Oleshchuk. *Conformance Checking of RBAC Policy and its Implementation*. In ISPEC'05 : 1st Information Security Practice and Experience Conference, volume 3439 of *LNCS*, pages 144–155. Springer-Verlag, 2005. 52, 63, 117

[Harrison76] Michael A. Harrison, Walter L. Ruzzo & Jeffrey D. Ullman. *Protection in operating systems*. Communications of the ACM, vol. 19, no. 8, pages 461–471, ACM Press, New York, 1976. 19, 208

[Holzmann97] Gerard J. Holzmann. *The Model Checker SPIN*. IEEE Transactions on Software Engineering, vol. 23, no. 5, pages 279–295, IEEE Computer Society, Los Alamitos, California, 1997. 53

[Huchard07] Marianne Huchard, Mohamed Rouane Hacene, Cyril Roume & Petko Valtchev. *Relational concept discovery in structured datasets*. Annals of Mathematics and Artificial Intelligence, vol. 49, no. 1-4, pages 39–76, Springer-Verlag, Berlin, 2007. 210

[ISO91] ISO. *Information Technology - Software Product evaluation - Quality Characteristics and Guidelines for their use*. Rapport technique, 1991. Int. Standard ISO/IEC 9126. 13

[ISO99] ISO. International Standardization Organization (ISO) and International Electrotechnical Commission (IEC) ISO/IEC 9075 :1999, database languages – SQL. 1999. 23, 177

[Jackson00] Daniel Jackson, Ian Schechter & Ilya Shlyakhter. *Alcoa : the Alloy Constraint Analyzer.* In International Conference on Software Engineering. IEEE CS Press, Los Alamitos, 2000. 53

[Jaffar94] Joxan Jaffar & Michael J. Maher. *Constraint Logic Programming : A Survey.* Journal of Logic Programming, vol. 19/20, pages 503–581, Elsevier, Amsterdam, 1994. 51, 83

[Jajodia01] Sushil Jajodia, Pierangela Samarati, Maria Luisa Sapino & V. S. Subrahmanian. *Flexible support for multiple access control policies.* ACM Transaction on Database Systems, vol. 26, no. 2, pages 214–260, ACM Press, New York, 2001. 46, 47, 52, 62, 63

[Jansen98] W. A. Jansen. *Inheritance Properties of Role Hierarchies*. In 21st NIST-NCSC National Information Systems Security Conference, pages 476–485, 1998. 28

[Jim01a] Trevor Jim. *SD3 : A Trust Management System with Certified Evaluation.* In IEEE Symposium on Security and Privacy, pages 106–115, 2001. 49, 50, 63

[Jim01b] Trevor Jim & Dan Suciu. *Dynamically distributed query evaluation*. In PODS'01 : 20th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 28–39, New York, NY, USA, 2001. ACM Press. 49, 50

[Jones93] A. J. I. Jones & M. Sergot. *On the Characterization of Law and Computer Systems : The Normative Systems Perspective*. In J.-J. C. Meyer & R. J. Wieringa, editeurs, Deontic Logic in Computer Science :

Normative System Specification, pages 275–307. Wiley, New York, 1993. 52

[Joshi02] James Joshi, Elisa Bertino & Arif Ghafoor. *Temporal hierarchies and inheritance semantics for GTRBAC*. In SACMAT'02 : 7th ACM symposium on Access control models and technologies, pages 74–83, New York, NY, USA, 2002. ACM Press. 43, 118

[Joshi03] James Joshi, Basit Shafiq, Arif Ghafoor & Elisa Bertino. *Dependencies and separation of duty constraints in GTRBAC*. In SACMAT'03 : 8th ACM Symposium on Access Control Models and Technologies, pages 51–64. ACM Press, 2003. 76

[Joshi05] James Joshi, Elisa Bertino, Usman Latif & Arif Ghafoor. *A Generalized Temporal Role-Based Access Control Model.* IEEE Transactions on Knowledge & Data Engineering, vol. 17, no. 1, pages 4–23, IEEE Computer Society, Los Alamitos, California, 2005. 43, 45, 74

[Journal Officiel98] Journal Officiel. *Décret n^o 98-608 du 17 juillet 1998*, 1998. Décret relatif à la protection des secrets de la défense nationale. 21

[Koch03] Manuel Koch & Francesco Parisi-Presicce. *Visual Specifications of Policies and Their Verification.* In Michel Wermelinger & Tiziana Margaria, editeurs, FASE'04 : 7th International Conference on Fundamental Approaches to Software Engineering, Barcelona, Spain, volume 2621 of *LNCS*, pages 278–293. Springer-Verlag, 2003. 37, 46, 53, 63

[Kuhlmann03] Martin Kuhlmann, Dalia Shohat & Gerhard Schimpf. *Role mining - revealing business roles for security administration using data mining technology*. In SACMAT'03 : 8th ACM Symposium on Access Control Models and Technologies, pages 179–186. ACM Press, 2003. 155, 191

[Kuhn97] Richard D. Kuhn. *Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems*. In RBAC '97 : 2ndACM workshop on Role-based access control, pages 23–30, New York, NY, USA, 1997. ACM Press. 28, 39, 40

[Kumar02] Arun Kumar, Neeran M. Karnik & Girish Chafle. *Context sensitivity in role-based access control.* ACM SIGOPS Operating Systems Review, vol. 36, no. 3, pages 53–66, ACM Press, New York, 2002. 44

[Lampson74] Butler W. Lampson. *Protection*. ACM SIGOPS Operating Systems Review, vol. 8, no. 1, pages 18–24, ACM Press, New York, 1974. iii, 3, 15, 17, 102

[Landwehr81] Carl E. Landwehr. *Formal Models for Computer Security*. ACM Computing Surveys, vol. 13, no. 3, pages 247–278, ACM Press, New York, 1981. 17

[Lassez89] Jean-Louis Lassez & Ken McAloon. *Independence of Negative Constraints.* In Josep Díaz & Fernando Orejas, editeurs, TAPSOFT'89 : Proceedings of the International Joint Conference on Theory and Practice of Software Development, Barcelona, Spain, volume 351 of *Lecture Notes in Computer Science*, pages 19–27. Springer-Verlag, 1989. 83, 90

[Lenzerini02] Maurizio Lenzerini. *Data Integration : A Theoretical Perspective.* In Lucian Popa, editeur, PODS'02 : 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Madison, Wisconsin, pages 233–246. ACM Press, 2002. 170, 207

[Li00a] Ninghui Li. *Delegation Logic : A Logic-based Approach to Distributed Authorization.* Doctorat, New York University, 2000. 49

[Li00b] Ninghui Li, Benjamin N. Grosof & Joan Feigenbaum. *A Practically Implementable and Tractable Delegation Logic.* In IEEE Symposium on Security and Privacy, pages 27–42, 2000. 46, 49

[Li03a] Ninghui Li, Benjamin N. Grosof & Joan Feigenbaum. *Delegation logic : A logic-based approach to distributed authorization.* ACM Transactions on Information & System Security, vol. 6, no. 1, pages 128–171, ACM Press, New York, 2003. 49, 63

[Li03b] Ninghui Li & John C. Mitchell. *DATALOG with Constraints : A Foundation for Trust Management Languages.* In Verónica Dahl & Philip Wadler, editeurs, Practical Aspects of Declarative Languages, 5th International Symposium, PADL 2003, New Orleans, LA, USA, January 13-14, 2003, Proceedings, volume 2562 of *Lecture Notes in Computer Science*, pages 58–73. Springer-Verlag, 2003. 49, 63, 72, 74, 76, 97

[Li04a] Ninghui Li, Ziad Bizri & Mahesh V. Tripunitara. *On mutually-exclusive roles and separation of duty*. In CCS '04 : 11th ACM conference on Computer and communications security, pages 42–51, New York, NY, USA, 2004. ACM Press. 38, 39

[Li04b] Ninghui Li & Mahesh V. Tripunitara. *Security analysis in role-based access control*. In Trent Jaeger & Elena Ferrari, editeurs, SACMAT'04 : 9th ACM Symposium on Access Control Models and Technologies, New York, pages 126–135, New York, NY, USA, 2004. ACM Press. 63

[Li07] Ninghui Li, Ji-Won Byun & Elisa Bertino. *A Critique of the ANSI Standard on Role-Based Access Control*. IEEE Security and Privacy, vol. 5, no. 6, pages 41–49, IEEE Computer Society, Los Alamitos, California, 2007. 28, 34, 54, 107, 113, 114, 120, 128, 204

[Liu04] Alex X. Liu, Mohamed G. Gouda, Huibo H. Ma & Anne H. H. Ngu. *Firewall Queries.* In Teruo Higashino, editeur, OPODIS'04 :

8th International Conference on Principles of Distributed Systems, Grenoble, France, volume 3544 of *Lecture Notes in Computer Science*, pages 197–212. Springer-Verlag, 2004. 46, 217

[Liu05] Peng Liu, Jian bin Hu & Zhong Chen. *A Formal Language for Access Control Policies in Distributed Environment*. In WI'05 : IEEE/WIC/ACM International Conference on Web Intelligence, pages 766–769, Washington, DC, USA, 2005. IEEE Computer Society. 63

[Maher96] Michael J. Maher & Divesh Srivastava. *Chasing Constrained Tuple-Generating Dependencies.* In Richard Hull, editeur, 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Montreal, Canada, pages 128–138. ACM Press, 1996. iii, 81, 87, 88, 90, 91, 127, 182, 185, 204, 247, 250

[Maher97] Michael J. Maher. *Constrained dependencies*. Theoretical Computer Science, vol. 173, no. 1, pages 113–149, Elsevier, Essex, 1997. 83

[Maher00] Michael J. Maher & Junhu Wang. *Optimizing Queries in Extended Relational Databases.* In Mohamed T. Ibrahim, Josef Küng & Norman Revell, editeurs, DEXA'00 : 11th International Conference on Database and Expert Systems Applications, London, volume 1873 of *Lecture Notes in Computer Science*, pages 386–396. Springer-Verlag, 2000. 83, 85

[Mayer06] Alain J. Mayer, Avishai Wool & Elisha Ziskind. *Offline firewall analysis.* International Journal of Information Security, vol. 5, no. 3, pages 125–144, Springer-Verlag, Berlin, 2006. 46

[Meyers05] Scott Meyers. Effective C++, third edition. Addison-Wesley, 2005. 113

[Miège05] Alexandre Miège. *Dénition d'un environnement formel d'expression de politiques de sécurité. Modèle Or-BAC et extensions.* Doctorat, Ecole Nationale Supérieure des Télécommunications,Paris, 2005. 29, 30, 45, 46, 63, 76, 108, 117

[Mossakowski03] Till Mossakowski, Michael Drouineaud & Karsten Sohr. *A temporal-logic extension of role-based access control covering dynamic separation of duties*. In 10th international symposium on temporal representation and reasoning, 4th international conference on temporal logic, volume 00, page 83, Los Alamitos, California, 2003. IEEE Computer Society. 43, 53

[Nash90] M. J. Nash & K. R. Poland. *Some Conundrums Concerning Separation of Duty.* In IEEE Symposium on Security and Privacy, pages 201–209, 1990. 32, 38, 39, 45

[Neumann02] Gustaf Neumann & Mark Strembeck. *A scenario-driven role engineering process for functional RBAC roles*. In SACMAT'02 : 7th ACM Symposium on Access Control Models and Technologies,, pages 33–42, New York, NY, USA, 2002. ACM Press. 155

[Nourine02] Lhouari Nourine & Olivier Raynaud. *A fast incremental algorithm for building lattices.* Journal of Experimental & Theoretical Artificial Intelligence, vol. 14, no. 2-3, pages 217–227, Taylor and Francis Group., London, 2002. 163

[Osborn00] Sylvia L. Osborn, Ravi S. Sandhu & Qamar Munawer. *Configuring role-based access control to enforce mandatory and discretionary access control policies*. ACM Transactions on Information & System Security, vol. 3, no. 2, pages 85–106, ACM Press, New York, 2000. 28

[Pasquier99] Nicolas Pasquier, Yves Bastide, Rafik Taouil & Lotfi Lakhal. *Discovering Frequent Closed Itemsets for Association Rules.* In Catriel Beeri & Peter Buneman, editeurs, ICDT'99 : 7th International Conference on Database Theory, Jerusalem, Israel, volume 1540 of *LNCS*, pages 398–416. Springer-Verlag, 1999. 163

[Postel81] J. Postel. *Internet ControlL Message Protocol*. RFC 792, Internet Engineering Task Force, September 1981. 217

[Revesz95] Peter Z. Revesz. *Constraint Databases : A Survey.* In Leonid Libkin & Bernhard Thalheim, editeurs, Selected Papers from Semantics in Databases Workshop, Prague, Czech Republic, volume 1358 of *Lecture Notes in Computer Science*, pages 209–246. Springer-Verlag, 1995. 83, 216

[Richters01] Mark Richters & Martin Gogolla. *OCL - Syntax, Semantics and Tools*. In Tony Clark & Jos Warmer, editeurs, Advances in Object Modelling with the OCL, volume 2263 of *LNCS*, pages 43–69. Springer-Verlag, Berlin, 2001. 53

[Rihaczek91] Karl Rihaczek. *The harmonized ITSEC evaluation criteria*. Computers and Security, vol. 10, no. 2, pages 101–110, Elsevier, Oxford, 1991. 13, 14

[Roeckle00] Haio Roeckle, Gerhard Schimpf & Rupert Weidinger. *Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization.* In ACM Workshop on Role-Based Access Control, pages 103–110, 2000. 25, 155

[Rozenberg97] G. Rozenberg, editeur. Handbook of graph grammars and computing bygraph transformations,vol. 1 : Foundations. World Scientific, London, New-Jersey, 1997. 53

[SAIC04] SAIC. *Role Engineering Process*. Rapport technique, Science Applications International Corporation, The Healthcare RBAC Task Force, 2004. 25

[Salvat96] Eric Salvat & Marie-Laure Mugnier. *Sound and Complete Forward and backward Chainingd of Graph Rules.* In Peter W. Eklund, Gerard Ellis & Graham Mann, editeurs, ICCS'96 : 4thInternational Conference on Conceptual Structures, Sydney,Australia, volume 1115 of *Lecture Notes in Computer Science*, pages 248–262. Springer-Verlag, 1996. iv, 92, 145

[Sandhu93] Ravi S. Sandhu. *Lattice-Based Access Control Models*. Computer, vol. 26, no. 11, pages 9–19, IEEE Computer Society, Los Alamitos, California, 1993. 20, 21

[Sandhu96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein & CharlesE. Youman. *Role-Based Access Control Models.* IEEE Computer, vol. 29, no. 2, pages 38–47, IEEE Computer Society, Los Alamitos, California, 1996. 25, 26, 32, 69, 117, 156

[Sandhu98] Ravi S. Sandhu & Venkata Bhamidipati. *An Oracle implementation of the PRA97 model for permission-role assignment*. In RBAC'98 : 3rd ACM workshop on Role-based access control, pages 13–21, New York, NY, USA, 1998. ACM Press. 35

[Sandhu99] Ravi S. Sandhu & Qamar Munawer. *The ARBAC99 Model for Administration of Roles.* In ACSAC'99, 15th Annual Computer Security Applications Conference,Scottsdale, Arizona, pages 229–240. IEEE Computer Society, 1999. 35, 117

[Sandhu04] Ravi S. Sandhu. *A Perspective on Graphs and Access Control Models.* In Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce & Grzegorz Rozenberg, editeurs, ICGT'04 : 2nd International Conference on Graph Transformations, Rome, Italy, volume 3256 of *Lecture Notes in Computer Science*, pages 2–12. Springer-Verlag, 2004. 37, 46

[Schlegelmilch05] Jürgen Schlegelmilch & Ulrike Steffens. *Role mining with ORCA*. pages 168–176, New York, 2005. ACM Press. 191

[Sieka01] Jeremy G. Sieka, Lie-Quan Lee & Andrew Lumsdaine. Boost graph library : User guide and reference manual. C++ In-Depth Series. Addison-Wesley, 2001. 184

[Simon97] Richard Simon & Mary Ellen Zurko. *Separation of Duty in Role-based Environments*. In CSFW '97 : 10th IEEE workshop on Computer Security Foundations, page 183, Washington, DC, USA, 1997. IEEE Computer Society. 39

[Sohr05] Karsten Sohr, Gail-Joon Ahn, Martin Gogolla & Lars Migge. *Specification and Validation of Authorisation Constraints Using UML and OCL.* In Sabrina De Capitani di Vimercati, Paul F. Syverson & Dieter Gollmann, editeurs, ESORICS'05, 10th European Symposium

on Researchin Computer Security, Milan, Italy, volume 3679 of *Lecture Notes in Computer Science*, pages 64–79. Springer-Verlag, 2005. 63

[Solomon05] Michael G. Solomon & Mike Chapple. Information security illuminated. Jones and Bartlett Publishers, Inc., USA, 2005. 14, 130

[Sowa84] John F. Sowa. Conceptual structures : Information processing in mind and machine. Addison-Wesley, 1984. 143, 146

[Stroustrup97] Bjarne Stroustrup. The C++ programming language, third edition. Addison-Wesley, 1997. 183

[TCS85] *Trusted Computer System Evaluation Criteria, DoD Standard 5200.28-STD*. Rapport technique, United States Department of Defense, December 1985. 17

[Thion06a] Romuald Thion & Stéphane Coulondre. *Modeling and Inferring on Role-Based Access Control Policies Using Data Dependencies.* In 17th International Conference on Database and Expert Systems Applications, DEXA'06, LNCS, pages 914–923, 2006. 7

[Thion06b] Romuald Thion & Stéphane Coulondre. *Representation and Reasoning on Role-Based Access Control Policies with Conceptual Graphs.* In 14th International Conference on Conceptual Structures, ICCS'06, volume 4068 of *LNCS*, pages 427–440, 2006. 7

[Thion06c] Romuald Thion & Stéphane Coulondre. *Un Modèle Homogène pour la Confidentialité et l'Intégrité des Données Relationnelles.* In Dominique Laurent, editeur, BDA'06, 22èmes Journées Bases de Données Avancées, Lille, 2006. 7

[Thion07a] Romuald Thion. Cyber warfare and cyber terrorism, chapitre Access Control Models. Information Science Reference. IDEA Group Publishing, Pennsylvania, may 2007. Colarik Andrew M. and Janczewski Lech eds., ISBN 978-1-59140-991-5. 6

[Thion07b] Romuald Thion. *Découverte automatisée de hiérarchies de rôles pour les politiques de contrôle d'accès - Prix du meilleur article Jeune Chercheur*. In Hermes, editeur, XXVème congrès INFORSID, INFORSID'07, pages 139–154, may 2007. 7

[Thomas97a] Roshan K. Thomas. *Team-based access control (TMAC) : a primitive for applying role-based access controls in collaborative environments*. In RBAC '97 : Proceedings of the second ACM workshop on Role-based access control, pages 13–19, New York, NY, USA, 1997. ACM Press. 31, 44

[Thomas97b] Roshan K. Thomas & Ravi S. Sandhu. *Task-Based Authorization Controls (TBAC) : A Family of Models for Active and Enterprise-Oriented Autorization Management.* In Tsau Young Lin & Shelly

Qian, editeurs, IFIP'98 : 11th International Conference on Database Security, Lake Tahoe, California, volume 113 of *IFIP Conference Proceedings*, pages 166–181. Chapman & Hall, 1997. 27, 32, 34

[Tidswell01] Jonathon E. Tidswell & John M. Potter. *A graphical definition of authorization schema in the DTAC model*. In SACMAT '01 : Proceedings of the sixth ACM symposium on Access control models and technologies, pages 109–120, New York, NY, USA, 2001. ACM Press. 37, 46, 53, 63, 211

[Vaidya06] Jaideep Vaidya, Vijayalakshmi Atluri & Janice Warner. *RoleMiner : mining roles using subset enumeration*. In CCS'06 : 13th ACM conference on Computer and Communications Security, pages 144–153, New York, NY, USA, 2006. ACM Press. 192, 193

[Vaidya07] Jaideep Vaidya, Vijayalakshmi Atluri & Qi Guo. *The role mining problem : finding a minimal descriptive set of roles*. In Volkmar Lotz & Bhavani M. Thuraisingham, editeurs, SACMAT'07 : 12th ACM Symposium on Access Control Models and Technologies, Sophia Antipolis, France, pages 175–184. ACM Press, 2007. 192, 210, 211, 212

[Vigna03] Giovanni Vigna. *A Topological Characterization of TCP/IP Security.* In Keijiro Araki, Stefania Gnesi & Dino Mandrioli, editeurs, FME '03 : Formal Methods, International Symposium of Formal Methods Europe, Proceedings, volume 2805 of *LNCS*, pages 914–939, Berlin, 2003. Springer-Verlag. 217

[Wainer03] Jacques Wainer, Paulo Barthelmess & Akhil Kumar. *W-RBAC - A Workflow Security Model Incorporating Controlled Overriding of Constraints.* International Journal of Cooperative Information Systems, vol. 12, no. 4, pages 455–485, World Scientific, London, 2003. 32

[Wainer07] Jacques Wainer, Akhil Kumar & Paulo Barthelmess. *DW-RBAC : A formal security model of delegation and revocation in workflow systems.* Information Systems, vol. 32, no. 3, pages 365–384, Elsevier, Amsterdam, 2007. 32

[Wang02] Junhu Wang. *Exploiting Constraints for Query Processing*. Doctorat, Griffith University, Brisbane, Queensland, Australia, 2002. 81, 83, 87, 91, 127, 185, 204

[Wang05] Junhu Wang, Rodney W. Topor & Michael J. Maher. *Rewriting Union Queries Using Views.* Constraints, vol. 10, no. 3, pages 219–251, Springer-Verlag, Amsterdam, 2005. 204

[Wermelinger95] Michel Wermelinger. *Conceptual Graphs and First-Order Logic*. In Gerard Ellis, Robert Levinson, William Rich & John F. Sowa, edi-

teurs, ICCS'95 : 3rd International Conference on Conceptual Structures, Santa Cruz, California, volume 954 of *Lecture Notes in Computer Science*, pages 323–337. Springer-Verlag, 1995. 144, 145

[Wille97] Rudolf Wille. *Conceptual Graphs and Formal Concept Analysis*. In ICCS'97 : 5th International Conference on Conceptual Structures, pages 290–303, 1997. 157, 212

[Witkowski07] Andrew Witkowski, Srikanth Bellamkonda, Hua-Gang Li, Vince Liang, Lei Sheng, Wayne Smith, Sankar Subramanian, James Terry & Tsae-Feng Yu. *Continuous queries in oracle*. In VLDB'07 : 33rd international conference on Very Large DataBases, pages 1173–1184. VLDB, 2007. 180

[Wool04] Avishai Wool. *A Quantitative Study of Firewall Configuration Errors*. Computer, vol. 37, no. 6, pages 62–67, IEEE Computer Society Press, Los Alamitos, California, 2004. 46, 217

[Zhang97] Xubo Zhang & Z. Meral Özsoyoglu. *Implication and Referential Constraints : A New Formal Reasoning*. IEEE Transactions on Knowledge & Data Engineering, vol. 9, no. 6, pages 894–910, IEEE Computer Society, Los Alamitos, California, 1997. 81