Multi-OrBAC : un modèle de contrôle d'accès pour les systèmes multi-organisationnels

Anas Abou El Kalam
LIFO-ENSI de Bourges ; anas.abouelkalam@ensi-bourges.fr

Yves Deswarte
LAAS - CNRS ; Yves.Deswarte@laas.fr

Résumé

Cet article propose un nouveau modèle de contrôle d'accès pour les applications complexes, hétérogènes, interopérables et distribuées : " Multi-OrBAC " (pour MultiOrganization-Based Access Control). Ce modèle permet de spécifier, dans un cadre homogène, plusieurs politiques de sécurité pour des organisations hétérogènes devant coopérer. Le but est d'offrir à chacune de ces organisations une certaine souplesse (par exemple, sur le choix d'une plate-forme, de services, de mécanismes) tout en respectant les contraintes imposées par une politique globale de sécurité. Multi-OrBAC est un modèle offrant deux présentations complémentaires, une présentation de type génie-logiciel - avec UML - destinée à la mise en œuvre, et une présentation formelle - avec la programmation logique par contraintes (PLC) - destinée à la validation. La PLC est utilisée pour spécifier les règles de fonctionnement du système ainsi que les règles de la politique de sécurité. Elle sert également à interroger la politique de sécurité, détecter et résoudre les conflits, vérifier la cohérence de la politique et gérer les problèmes de hiérarchie et de délégation. Enfin, un prototype a été développé pour illustrer l'utilisation du modèle pour décrire une politique de sécurité, ainsi que la résolution de conflits et la vérification de cohérence dans le cas d'un système multi-organisationnel.

Mots-clés : Politiques et modèles de sécurité, systèmes distribués interopérables et hétérogènes, détection et résolution de conflits, formalisation des règles d'héritage.

1 Introduction

Avec l'avènement de la «société de l'information», l'informatisation s'impose dans de nombreuses applications actuelles et émergentes : télémédecine, commerce électronique, administrations centrales et collectivités territoriales, etc. Il devient donc de plus en plus nécessaire de faire face aux risques et de renforcer

la sécurité des systèmes pour permettre d'avoir une confiance justifiée dans les traitements et la distribution des données et services informatiques.

Une large panoplie de techniques, mesures et outils de sécurité existent tant dans la littérature que sur le marché. Ces mécanismes, aussi robustes soient-ils, ne permettent d'atteindre un niveau de sécurité satisfaisant que s'ils s'inscrivent dans une démarche globale fondée sur une politique de sécurité. Celle-ci doit décrire de manière claire et non-ambiguë, les objectifs de sécurité à atteindre ainsi que les règles qui régissent la manière dont l'information sensible et les autres ressources sont gérées, protégées et distribuées [1]. Néanmoins, la définition d'une politique de sécurité ne garantit pas à elle seule un fonctionnement correct du système. La politique peut en effet être mal conçue ou violée intentionnellement (par malveillance) ou accidentellement (par maladresse).

Il est donc nécessaire de contraindre le système à respecter les règles à l'aide de mécanismes appropriés (en particulier de contrôle d'accès), de détecter les conflits éventuels, et de faire en sorte qu'aucune séquence valide d'applications des règles de la politique de sécurité ne puisse amener le système dans un état tel qu'un objectif de sécurité soit violé. Ceci suppose l'utilisation d'un modèle formel pour décrire et abstraire la politique de sécurité afin de réduire sa complexité, mais aussi pour vérifier que cette politique est complète et cohérente, et que la mise en œuvre par le système de protection est conforme aux propriétés attendues du système. Dans cette optique, cet article préconise l'utilisation :

- d'UML pour représenter les concepts de base de la politique de sécurité et faciliter son intégration dans une conception génie-lociciel du système d'information ;
- de la programmation logique par contraintes pour modéliser les règles de fonctionnement du système, formaliser et raisonner sur la politique de sécurité et traiter les problèmes de délégation et de hiérarchie.

La réflexion menée dans cet article s'articule autour de six sections : la section 2 expose les points forts et les limites des modèles classiques de sécurité. La section 3 présente Multi-OrBAC comme une extension du modèle OrBAC. Multi-OrBAC couvre la richesse des systèmes collaboratifs, distribués et interopérables, et permet de prendre en compte d'éventuelles améliorations ou modifications de politique de sécurité (modèle extensible et modulable). La section 4 décrit l'utilisation de Multi-OrBAC à travers une étude de cas. Ensuite, la section 5 présente le formalisme logique associé à Multi-OrBAC. Enfin, la section 6 dresse les conclusions et propose des extensions possibles de ce travail.

2 Modèles traditionnels de sécurité

Les politiques et modèles classiques de sécurité sont mal adaptés aux applications actuelles qui sont de plus en plus complexes, coopératives et distribuées. Pour illustrer cette limite, discutons des politiques et modèles discrétionnaires, obligatoires, à base de rôles et d'organisations. Un état de l'art plus détaillé peut être trouvé dans les références [2], [3], [4].

Une politique est dite discrétionnaire si l'entité qui possède un objet peut

gérer et manipuler librement les droits sur cet objet (comme c'est le cas, par exemple, sous UNIX). Cette classe de politiques de sécurité présente des inconvénients, comme les risques de fuite d'informations et d'attaques par chevaux de Troie. Le modèle de Lampson [5] (comme ses améliorations HRU [6] et Take-Grant [7]), est un modèle discrétionnaire de base, structuré sous forme d'une machine à états où chaque état est représenté par un triplet (S,O,M) : S désigne un ensemble de sujets (entités actives), O un ensemble d'objets (entités passives, conteneurs d'information) et M une matrice de contrôle d'accès. Chaque cellule M(s, o) de cette matrice contient les droits que le sujet s possède sur l'objet o. Ce modèle rend difficile la mise à jour de la politique de sécurité comme il augmente les risques d'erreurs de l'administrateur de sécurité. En effet, à chaque fois qu'un nouvel objet, un nouveau sujet ou une nouvelle action sont ajoutés, il faut modifier les règles qui portent sur ces entités.

Pour résoudre les problèmes des politiques discrétionnaires, les politiques obligatoires décrètent des règles incontournables. Ainsi, les politiques multiniveaux affectent-elles aux objets et aux sujets des attributs non modifiables par les usagers, et donc qui limitent leur pouvoir de gérer les accès aux informations qu'ils possèdent - en interdisant par exemple à tout sujet de lire une information de classification supérieure à son habilitation. Les politiques dites de Bell et LaPadula [8] et de Biba [9] en sont les exemples les plus anciens. D'autres exemples ont été développés pour les systèmes commerciaux [10] et pour les institutions financières [11]. Les politiques obligatoires sont très spécifiques aux domaines pour lesquels elles ont été développées. Leur mise en œuvre impose des contraintes fortes aux organisations et aux utilisateurs. De plus, elles sont vues comme centralisées et donc mal adaptées à une prise de décision en réseau ou au travers de systèmes réellement répartis.

A l'inverse, le modèle de contrôle d'accès basé sur les rôles (RBAC pour «Role-Based Access Control») [12], [13] est plus adapté aux entreprises. Il permet une administration plus facile et réduit les coûts de gestion des droits. En effet, dans RBAC, les privilèges ne sont plus associés d'une façon directe aux utilisateurs mais à travers des rôles : chaque rôle correspond à une fonction dans l'entreprise, et reçoit les droits (ou privilèges) nécessaires pour assumer la fonction ; d'autre part, chaque utilisateur se voit assigner un ou plusieurs rôles. Malgré cette souplesse, RBAC (dans sa version de base) ne permet pas de tenir compte du contexte pour affiner le contrôle d'accès. De plus, le concept de privilège est donné sous une forme générique, et doit être détaillé séparément pour chaque application. Nous pensons à l'inverse qu'il serait préférable de raffiner cette notion de «permissions» et d'intégrer les détails de sa structure au modèle [14].

Pour pallier ces problèmes, nous avons contribué - dans le cadre du projet RNRT MP6 - au développement du modèle Or-BAC «Organization-based Access Control» [14]. Dans OrBAC, la politique de sécurité est définie uniquement avec des entités abstraites : organisation, rôle, vue, activité, contexte. L'organisation peut être définie comme une entité ayant une autorité statutaire bien défini ; les rôles sont utilisés (comme dans RBAC) pour regrouper les utilisateurs ayant les mêmes fonctions ; les vues regroupent les objets qui satisfont

une propriété commune ; les activités correspondent à des opérations sur les vues, c'est-à-dire qu'elles regroupent les actions sur les objets correspondants. Les règles de sécurité OrBAC sont exprimées sous la forme suivante : Permission|Interdiction|Obligation|Recommandation(org ; r ; v ; a ; contexte). Par exemple, la règle «Permission(Rangueil, Médecin, Modification, Dossier_Medical, Medecin_Traitant)» signifie que dans l'organisation «Rangueil», les médecins ont la permission de modifier les dossiers médicaux dans le contexte «médecin traitant»

OrBAC est un modèle innovant qui permet d'abstraire et de maîtriser la complexité de la politique de sécurité. Il est toutefois peu adapté aux systèmes distribués, hétérogènes et interopérables. En effet, examinons les questions suivantes :

1. Comment définir des règles qui concernent plusieurs organisations ? autrement dit, est-il possible de définir des règles (par exemple, des permissions) pour des utilisateurs appartenant à d'autres organisations, ou même à des sous-organisations d'une même organisation ?
2. Est-il possible d'avoir - dans une même organisation - des objets sur lesquels on réalise des actions différentes ?

Étant donné qu'une règle OrBAC ne porte que sur une seule organisation, il est impossible de définir des règles qui concernent des organisations indépendantes. OrBAC couvre donc difficilement les besoins de distribution et d'interopérabilité. La réponse à la deuxième question est également non. En effet, comme le

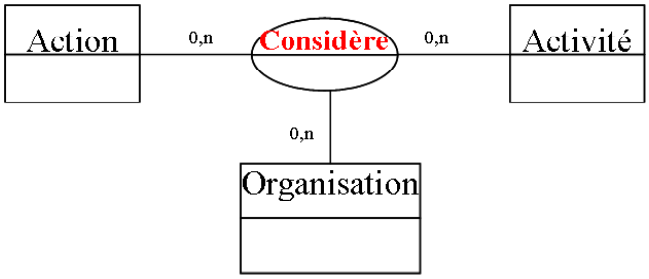

FIG. 1 – Relation entre les actions, les activités et les organisations dans OrBAC.

montre la figure **??**, la relation entre action et activité est définie de façon unique pour une organisation, indépendamment de la vue. Ainsi, si on suppose qu'une certaine organisation possède les vues «dossier médical», «dossier administratif» et «dossier fournisseur», l'action (correspondant à une certaine activité, par exemple lire) qu'on peut exécuter sur les trois vues doit être la même. De manière concrète, si une organisation, possède des documents de traitement de texte, tous les objets (consultables) de cette organisation ne peuvent être soumis qu'à une même action ReadDocFile(). Purpan ne peut donc pas avoir à la fois des fichiers XML (ou Word, ou autres) pour les dossiers administratifs et des fichiers texte (ou bases de données, etc.) pour les dossiers des fournisseurs. OrBAC ne satisfait donc pas les besoins d'hétérogénéité.

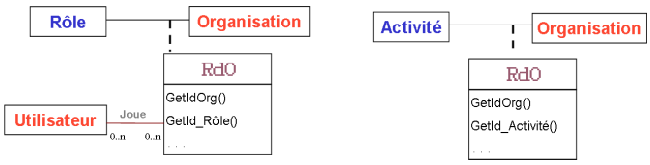

FIG. 2 – Ébauches du diagramme de classe UML représentant les RdO et AdO

Il semble donc nécessaire d'étendre Or-BAC afin de l'adapter aux besoins de distribution, de collaboration et d'hétérogénéité.

3 Le modèle Multi-OrBAC

3.1 Rôle dans Organisation 'RdO'

Dans un contexte multi-organisationnel, même si un utilisateur peut jouer plusieurs rôles, il n'a pas forcément le droit de les jouer dans n'importe laquelle des organisations. Le modèle Multi-OrBAC traite ce problème en ajoutant la notion de «rôle dans organisation»(RdO). On peut imaginer des cas où un utilisateur Pierre joue les rôles «infirmier à l'hôpital de Purpan» et «radiologue assistant à Rangueil», mais pas forcément «radiologue assistant à Purpan», ni «infirmier à Rangueil». Un RdO est une association entre le rôle et l'organisation, il possède des attributs, et peut participer à d'autres relations. Un RdO présente donc les caractéristiques d'une classe et d'une association et peut à ce titre être décrit par une classe association selon la notation UML [15] (**??**). Bien évidemment, rien n'empêche qu'un certain utilisateur puisse jouer plusieurs rôles dans une même organisation ; dans ce cas, cet utilisateur est tout simplement associé à plusieurs RdO (où l'organisation est la même).

3.2 Activité dans Organisation 'AdO'

De la même manière que nous avons présenté la notion de RdO, nous introduisons l'entité «activité dans organisation» (notée AdO) comme classe-association entre les activités et les organisations (figure 2). Remarquons que là encore, l'objectif est de permettre à des organisations de structurer différemment les mêmes activités. Pour illustrer cela, supposons que l'hôpital de Purpan utilise un système de fichiers, et que Rangueil utilise une base de données ; si nous considérons l'activité «lecture», celle-ci peut correspondre, dans l'organisation Purpan, à l'action «fopen()», mais peut tout aussi bien correspondre à l'action «select» dans Rangueil (figure 3).

3.3 Vue dans Organisation 'VdO'

Dans la mesure où les vues caractérisent la manière dont les objets sont utilisés dans l'organisation, nous introduisons la classe-association «Vue dans

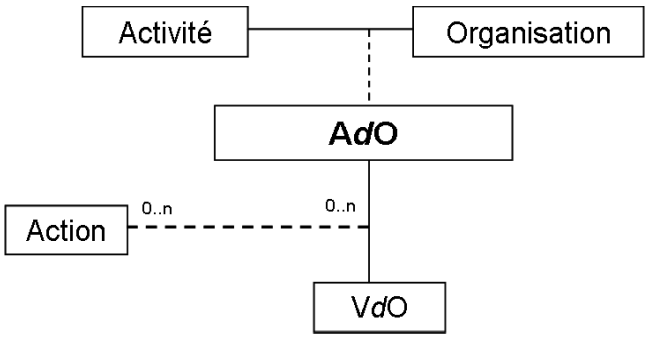

FIG. 3 – Ébauches du diagramme de classe UML représentant les VdO.

Organisation», (VdO), et nous associons les objets aux VdO (figure 3). Cette manière de structurer les organisations, les objets et les vues permet aux organisations de donner des définitions différentes à une même vue ; par exemple, la vue «dossier médical»peut être définie à Purpan comme un ensemble de documents textes, tandis qu'à Rangueil, cette même vue correspond à des attributs ou à des tables de la base. Notons que dans multi-OrBAC, l'action dépend, non seulement de l'activité et de l'organisation, mais aussi de la vue. On peut donc tout à fait avoir - dans la même organisation - des vues hétérogènes (c'est-à-dire sur lesquels on peut exécuter des actions différentes).

3.4 Contexte dans Organisation

Même si dans Or-BAC de base [14] la notion de contexte est ambiguë, certains travaux plus récents ont tenté d'éclaircir cette notion [2], [16]. Nous pensons que le contexte peut être défini comme toute information qui peut être utilisée pour spécifier les circonstances concrètes de l'accès, et ainsi fournir un contrôle plus fin. Cela correspond donc à une contrainte, c'est-à-dire un prédicat logique associé à une règle d'accès et qui lie la validité de la règle une à l'état du système (par exemple en fonction de certaines valeurs d'attributs). Par exemple, la règle «l'utilisateur u a la permission d'exécuter l'action a seulement si le contexte $[@\_Requtet = @\_Station_Y et\ temps\_requte \in periode_P]$». Dans Multi-OrBAC, comme pour RdO, VdO et AdO, le concept de «Contexte dans Organisation» (CdO) peut être représenté comme une classe association entre le contexte et l'organisation. Une instance de CdO pourrait être par exemple «urgence dans Rangueil». Bien évidemment, l'administrateur sécurité doit spécifier les conditions qui seront évaluées en temps réel pour décider si, à un instant donné, un CdO est vrai ou pas.

3.5 Le métamodèle Multi-OrBAC

Dans Multi-OrBAC, une politique définit des règles du type : dans le CdO c, l'organisation org accorde au RdO r la permission de réaliser l'AdO a sur

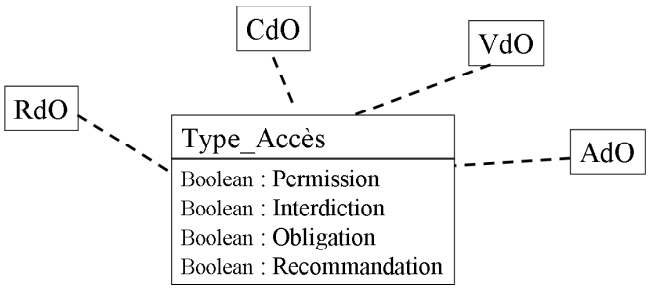

FIG. 4 – Ébauche du diagramme de classe représentant les règles Multi-OrBAC.

la VdO v. Nous relions ainsi les entités «RdO, VdO, AdO et CdO» avec une classe-association notée «type d'accès» et qui correspond à une permission, interdiction, obligation ou recommandation (figure 4). En réalité, les règles Multi-OrBAC peuvent être simplifiées en Type_Accès(RdO, AdO, Vue, Contexte) dans la mesure où l'activité, la vue et le contexte doivent appartenir à la même organisation, celle dans laquelle sera exécutée l'action. Bien évidemment, cette représentation inclut le cas particulier où la règle ne concerne qu'une seule organisation (c'est le cas d'OrBAC) ; dans ce cas, le CdO peut préciser que les organisations (du RdO, VdO, et AdO) doivent être identiques.

Si de plus, on veut contrôler la manière selon laquelle les règles sont ajoutées dans le système, on peut envisager des règles Multi-OrBAC sous la forme Type_Accès(Org, RdO, AdO, Vue, Contexte) ; où Org est l'organisation qui définit la règle. Dans ce cas, lors de l'ajout d'une règle, le système doit vérifier que l'organisation qui définit la règle (Org) est une super-organisation de celle qui contient les éléments de la vue, comme il doit vérifier que l'administrateur qui ajoute cette règle appartient à Org (ou à une super organisation de Org). De cette manière, une organisation ne peut accorder des permissions, interdictions et recommandations que sur les objets qui lui appartiennent ou qui appartiennent à ses sous-organisations. La politique de sécurité comporte ainsi des faits du type : Permission(Purpan, Médecin-Dans-Rangueil, Lecture-Dans-Purpan, DossierMédical, Urgence). Cette règle implique deux organisations différentes et signifie que «l'hôpital Purpan accorde aux médecins de l'hôpital de Rangueil la permission de consulter les dossiers médicaux (de Purpan) dans un contexte d'urgence».

Au moment de la requête, et avant d'accorder ou rejeter l'accès, le système doit calculer et vérifier le contexte courant en fonction des relations qui existent entre le sujet demandant l'accès, l'objet invoqué, l'action demandée, les organisations impliquées ainsi que d'autres informations (par exemple, le temps) sur le système.

Il est important de souligner que, dans Multi-OrBAC (comme dans OrBAC), les règles de sécurité ne sont pas spécifiées pour chaque objet, sujet et action, mais seulement en utilisant des entités abstraites : organisations, rôles, vues, activités et contextes. Pour autant, le contrôle d'accès de bas niveau doit per-

mettre de décrire les actions concrètes que réalisent les sujets sur les objets. Nous distinguons ainsi deux niveaux d'abstraction :

- niveau abstrait, portant uniquement des entités abstraites (organisation, rôle, vue, activité, contexte) ; la politique de sécurité y est exprimée à travers la classe association «Type d'accès» (permission, obligation, interdiction ou recommandation) ;
- niveau concret, portant sur des autorisations (ou obligations ou interdictions ou recommandations) concrètes associées, dans le contexte courant, à un sujet si, un objet oj et une action ak. Ces types d'accès sont déduits, à un moment donné, par instanciation des règles de la politique de sécurité.

Une représentation UML du modèle Multi-Orbac est donnée dans la figure 5. Pour enrichir l'expressivité et diminuer la complexité, cette représentation spé-

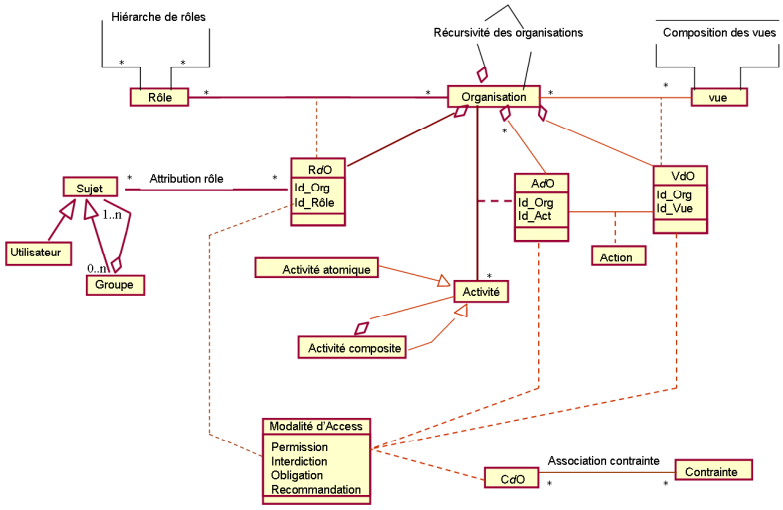

FIG. 5 – Représentation UML du méta modèle Multi-OrBAC

cifie la hiérarchie des rôles, la récursivité des organisations ainsi que la composition des sujets, vues et activités. Regardons, à titre d'exemple, la structure des sujets : les groupes de sujets (ou équipes) comme les utilisateurs, sont des sujets (relation d'héritage) et d'autre part, les groupes sont composés de sujets, c'est-à-dire d'utilisateurs ou d'autres équipes (relation de composition). Cette manière de faire assure la flexibilité et la récursivité, dans la mesure où il est possible de former des équipes, qui à leur tour peuvent être regroupées pour former d'équipes plus grandes, et ainsi de suite. Le même raisonnement peut être appliqué aux activités, organisations, vues et objets.

Notons qu'UML est parfaitement adapté pour représenter plusieurs aspects reliés à la politique de sécurité. Nous venons de voir que les notions RdO, AdO, VdO et CdO et «type d'accès» peuvent être spécifiées par des classes-associations ; les rôles (resp. vues, activités) sont modélisés par des classes dont le nom est celui du rôle (resp. vue, activité) et dont les attributs décrivent les

propriétés des fonctions des sujets (resp. objet, action) jouant le rôle (resp. participant à la vue ou à l'activité) ; la spécialisation (entre rôles, par exemple) peut être modélisée par un héritage ; les organisations peuvent être représentées soit par des classes (dans le diagramme de classes), des objets composites (dans le diagramme d'objets) ou par l'élément de spécification UML «sous-système» (dans le diagramme de déploiement).

Les objets (au sens de la sécurité) du système sont modélisés par des objets UML. Le fait qu'un objet puisse être tantôt actif (un patient qui consulte son dossier médical, et joue à ce titre le rôle patient) tantôt passif (lorsque l'infirmière lui fait une injection par exemple) est parfaitement modélisable à l'aide de l'héritage multiple entre classes.

Afin de préciser la sémantique de certains éléments comme les organisations et les rôles, on peut éventuellement utiliser les stéréotypes d'UML.

Pour exprimer les expressions contextuelles, plusieurs alternatives s'offrent à nous :

- utiliser des contraintes de multiplicité sur les relations ou sur les classes ; par exemple pour restreindre le nombre d'utilisateurs jouant un rôle ou appartenant à une organisation, contraindre la structure d'une organisation, etc. ;
- utiliser des expressions du langage OCL (Object Constraint Languge) [17] ; celui-ci permet d'exprimer de manière formelle, des contraintes sous forme d'expressions booléennes prédéfinies ou personnalisées ; néanmoins, le langage OCL reste peu expressif, il ne dispose pas d'une sémantique explicite et ne permet pas d'exprimer certaines contraintes importantes pour la sécurité comme les contraintes temporelles ;
- utiliser du pseudo-code ou une description en langage naturel, bien entendu, de manière informelle.

En plus des aspects structuraux, UML permet également de décrire plus en détail certains aspects comportementaux. Par exemple, les interactions entre les organisations peuvent être détaillées par des transitions entre des diagrammes d'objets. La transition doit être étiquetée par la règle de sécurité définissant le type d'accès (permission, interdiction, obligation ou recommandation) de l'action. Si cette règle est conditionnelle (par exemple, de la forme : il est permis que X seulement si Y), les conditions sont données par des expressions logiques (par exemple en OCL). La figure 6 représente une règle de sécurité par une relation de dépendance entre deux sous-systèmes ; cette relation est labellisée par le nom de la règle et possède une note avec la condition de garde ; le sous-système «avant» représente les conditions nécessaires pour que l'action soit réalisée, tandis que le sous-système «après» représente les effets de l'action sur le système.

4 Etude de cas

Dans cette section, nous décrivons une architecture sécurisée basée sur Multi-OrBAC. La figure 7 présente un exemple de scénario montrant les différentes

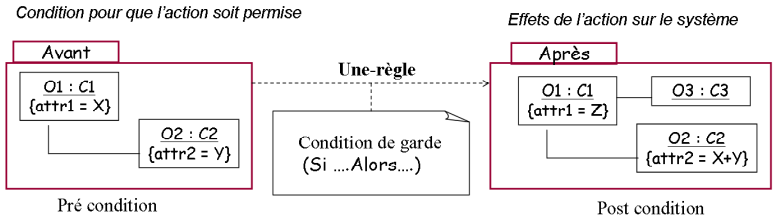

FIG. 6 – Exemple de représentation UML d'une règle de sécurité

étapes (possibles) parcourues entre une requête d'accès d'un utilisateur et la réponse à sa demande.

Deux entités importantes peuvent être distinguées dans cette figure :

- un «moniteur de référence» (MR) [18] : intermédiaire entre les sujets et les objets, il vérifie que chaque accès d'un sujet vers un objet est autorisé par un droit d'accès dans la politique ;
- un «serveur d'autorisation» (SA) : il consulte les règles de sécurité et génère les preuves d'autorisation.

Dans ce scénario, l'utilisateur envoie une requête avec un ensemble de paramètres comme son RdO (ex. radiologiste dans Rangueil) et l'objet invoqué. Une fois authentifié (par le MR), la requête est envoyée au SA. Ce dernier effectue les tâches suivantes :

- il interroge la politique de sécurité (contenue par exemple dans une base de données dont les champs sont «Type d'accès ; RdO ; AdO ; Vue ; Contexte»),
- il extrait les règles qui permettent de décider dans le cas courant (ex. les règles où figure le RdO envoyé en paramètres de la requête),
- il combine les différentes séquences de règles possibles (ex. par le biais du mécanisme d'héritage) et évalue les paramètres de la requête dans ces règles,
- il résout les conflits éventuels,
- il prend une décision (est-ce que l'action est permise, interdite, obligatoire ou recommandée ?),
- il génère un ensemble de preuves d'autorisation (un ticket, par exemple) sous la forme : RdO ;(Permission, AadOrgdest, vuea, contexte1) ;(Interdiction, AbdOrgdest, vueb, contexte2), ...,
- il chiffre l'ensemble avec la clé publique du MR d'Orgdest et signe avec sa clé privée.

Quand l'utilisateur (ou plutôt son MR) reçoit le ticket, il l'envoie à Orgdest. Le MR d'Orgdest vérifie la signature du SA, déchiffre le ticket, et autorise ou refuse l'accès. Si l'utilisateur souhaite accéder à des objets appartenant à d'autres organisations, il demande un autre ticket au SA.

L'architecture système ainsi que le diagramme de déploiement UML correspondant à notre scénario sont illustrés par la figure 8. Il ne s'agit que d'une architecture simplifiée avec deux organisations, la première utilise un système de base de données (BD) relationnelle, tandis que la deuxième utilise un système

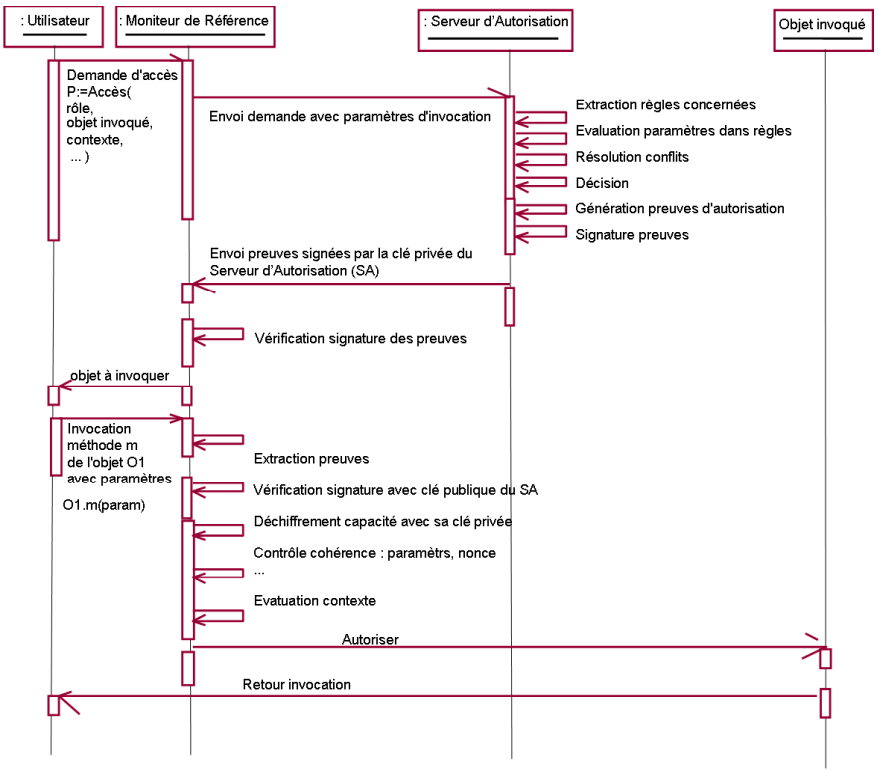

FIG. 7 – Diagramme de séquence UML détaillant un scénario de la phase d'autorisation.

de fichiers XML. La politique de sécurité Multi-OrBAC contient les règles gérant les interactions entre les organisations et est stockée dans une base accessible par le SA. Le diagramme de déploiement représente les éléments de réalisation (fichiers, modules, etc.), et décrit l'environnement d'exécution et de déploiement du code.

5 Modèle formel

5.1 Intérêt d'une approche formelle et choix du langage

L'objet de cette section est de présenter un cadre logique qui permet de spécifier les aspects liés à la sécurité du système et qui offre des outils d'aide au raisonnement sur la politique de sécurité. Le principal atout de l'utilisation d'une approche formelle dans la spécification d'une politique de sécurité réside dans l'élimination d'un certain nombre des ambiguïtés de la spécification, et d'aider l'administrateur à spécifier, à définir et à formaliser la politique de sécurité [19]. Les profits en sont multiples, par exemple :

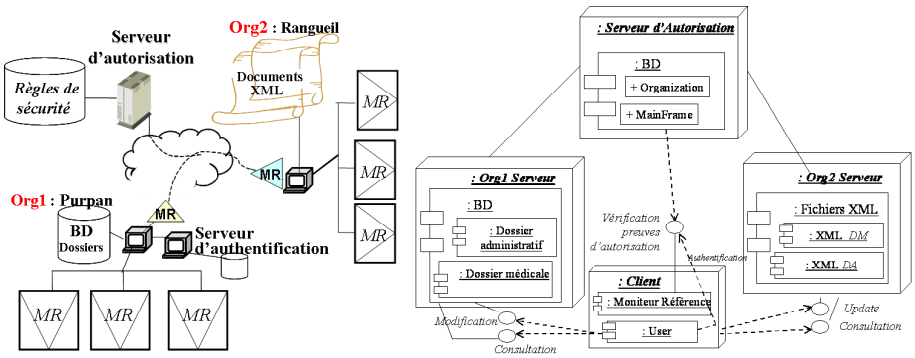

FIG. 8 – Architecture et diagramme de déploiement UML du scénario proposé.

- L'interrogation d'une politique de sécurité afin d'avoir une réponse aux requêtes du type «qui a des privilèges (et lesquels) sur un (des) objet(s) donné(s), et dans quel contexte ?».
- La vérification de la cohérence d'une politique de sécurité. Globalement, on peut distinguer quatre types d'incohérence dans une politique de sécurité :
  - Certains objectifs de sécurité sont en contradiction les uns avec les autres (les objectifs ne peuvent pas être tous satisfaits en même temps).
  - Des règles de sécurité de la politique se contredisent (une action est à la fois permise par une règle et interdite par une autre).
  - Les règles de fonctionnement du système sont incompatibles avec les objectifs ou les règles de sécurité.
  - Les règles ne garantissent pas la satisfaction des objectifs : partant d'un état sûr (satisfaisant les objectifs de sécurité), il est possible, tout en respectant les règles de sécurité, d'atteindre un état non-sûr (qui compromet l'un des objectifs de sécurité).
- La garantie et la justification que l'implémentation du système d'information, et en particulier des mécanismes de contrôle d'accès, permet bien de garantir les propriétés de sécurité souhaitées.
- La vérification de la complétude d'une politique notamment en montrant que, face à chaque risque identifié, il existe une règle dans la politique de sécurité qui définit la conduite à tenir face à ce risque.
- La gestion des problèmes de fusion de politiques de sécurité, par exemple dans le cadre d'une restructuration entre deux organismes. Un premier aspect concerne la définition de rôles et de structures organisationnelles qui soient compatibles. Un autre aspect concerne ensuite la détection de conflits dans la politique obtenue par fusion, puis la proposition d'une méthode permettant de résoudre ces conflits.

Pour résoudre une bonne partie de ces problèmes, nous avons utilisé la programmation logique par contraintes. Tout d'abord, la programmation logique (PL) est un paradigme de programmation qui a été introduit par R. Kowalski [20] et A. Colomerauer [21]. Elle est basée sur un sous-ensemble de la logique des

prédicats du premier ordre (clauses de Horn). Nous l'avons choisi en raison de :

- sa déclarativité, qui permet de décrire les connaissances d'un problème d'une façon simple ;
- sa puissance, par l'utilisation de processus d'unification et de résolution pour inférer la solution du problème à résoudre à partir de sa description ;
- sa souplesse, offerte par la possibilité de résoudre différents types de problèmes suivant l'instantiation des arguments.

La programmation logique par contraintes (PLC) est une extension de la PL [22]. Elle constitue une combinaison pratique entre le processus de déduction logique et un ensemble d'algorithmes incrémentaux de résolution de contraintes (solveurs de contraintes).

C'est pourquoi nous avons choisi la PLC pour spécifier les règles de fonctionnement du système étudié et les règles de la politique de sécurité pour exprimer les règles d'héritage et de composition (d'organisations, de rôles, de vues et d'activités), et finalement pour détecter et résoudre - à l'aide des solveurs de contraintes - les conflits éventuels.

5.2 Spécification d'une règle de sécurité Multi-OrBAC

Tout d'abord, les relations de Multi-OrBAC - telles qu'elles ont été définies dans la figure 5 - sont transformées en prédicats logiques, par exemple : Joue(sujet, RdO), Sous-Rôle(Rôle-junior, Rôle senior). Une instance du premier prédicat serait du type Joue(David, Médecin dans Purpan) tandis qu'une instance du deuxième peut être Sous-Rôle(Chirurgien, Médecin) exprimant qu'un chirurgien est nécessairement un médecin. Les règles de sécurité (décrites dans la figure 4) sont spécifiées sous la forme :
$Permission(RdO, AdO, VdO, CdO) \wedge$
$Joue(s, RdO) \wedge$
$Correspond\_à(o, VdO) \wedge$
$Appartient\_à(a, AdO) \wedge$
$Est_Vrai(CdO) \rightarrow Est_permis(s, a, o).$
Une instance de cette règle peut être par exemple :
$\forall s \in Sujet \forall a \in Action \forall o \in Objet$
$Permission(Médecin - dans - Rangueil, Lecture - dans - Purpan,$
$Doss\_Médical, urgence) \wedge$
$Joue(David, Médecin - dans - Rangueil) \wedge$
$Correspond\_à(f1.xml, Doss_Médical - dans - Purpan) \wedge$
$Appartient\_à(Read - xml, Lecture - dans - Purpan) \wedge$
$Est_Vrai(urgence - dans - Purpan)$
$\rightarrow Est\_permis(s, a, o).$
Cette règle signifie : si une règle (niveau abstrait) donne, dans le contexte urgence-dans-Purpan, à Médecin-dans-Rangueil la permission Lecture-dans-Purpan sur Doss_Médical-dans-Purpan, si le sujet David joue Médecin-dans-Rangueil, si f1.xml correspond à Doss_Médical-dans-Purpan, si l'action Read-xml appartient à l'AdO Lecture-dans-Purpan et si le contexte urgence-dans-Purpan est vrai, alors David a la permission d'exécuter Read-xml sur f1.xml (niveau concret).

5.3 Spécification des règles d'héritage

Afin de réduire la complexité de la politique de sécurité, Multi-OrBAC dispose d'un mécanisme d'héritage (relation d'ordre partiel) qui porte sur les rôles, les organisations et les vues.

1. Intuitivement, si un rôle junior hérite d'un rôle senior, alors il peut acquérir les permissions accordées au rôle senior. Ceci peut être spécifié par la règle suivante :
   $Permission(Rôle_{senior}\ dans\ Org_a, AdO, VdO, CdO) \wedge$
   $sous\_rôle\ (Rôle_{junior}, Rôle_{senior}) \rightarrow$
   $Permission(Rôle_{junior}\ dans\ Org_a, AdO, VdO, CdO)$.
   Par exemple,$Permission(Médecin-dans-Rangueil, AdO, VdO, CdO) \wedge$ $sous\_rôle(Chirurgien, Médecin). \rightarrow Permission(Chirurgien - dans - Rangueil, AdO, VdO, CdO)$.
2. La composition des organisations peut également être exploitée par les règles d'héritage. Généralement, si des permissions sont accordées à un RdO (ex. Médecin-dans-Rangueil), alors on peut imaginer que ces permissions peuvent s'étendre aux sous-organisations (ex. sont accordées aux médecins de toutes les sous-organisations de Rangueil : département de radiologie, service des urgences, etc.). La règle correspondant à ce type de raisonnement est :
   $Permission(RôledansOrg_1, AdO, VdO, CdO) \wedge sous\_organisation(Org_2, Org_1) \rightarrow Permission(RôledansOrg_2, AdO, VdO, CdO)$.
3. On peut également raisonner sur la composition des objets et imaginer une règle du type : «si un rôle possède des permissions sur des objets d'une certaine organisation Org_1, alors il a ces permissions sur tous les objets des sous-organisations d'Org_1». Par exemple :
   $Permission(Médecin-dans-Rangueil, Lecture-dans-Purpan, VdO, CdO) \wedge sous\_organisation(Département_1, Purpan) \rightarrow$
   $Permission(Médecin-dans-Rangueil, Lecture-dans-Départment_1, VdO, CdO)$.
   La règle peut être spécifiée de la manière suivante :
   $Permission(RdO, ActivitédansOrg_{senior}, VuedansOrg_{senior}, ContexteinOrg_{senior}) \wedge sous\_org(Org_{junior}, Org_{senior})$
   $\rightarrow Permission(RdO, ActivitdansOrg_{junior}, VuedansOrg_{junior}, ContextedansOrg_{junior})$.

5.4 Détection et résolution de conflits

Lorsqu'une politique de sécurité contient des permissions, mais aussi des interdictions, voire des obligations et des recommandations, il est nécessaire de s'assurer qu'elle ne peut pas générer de conflit, c'est-à-dire, garantir qu'il n'existe pas de situation dans laquelle un utilisateur aurait simultanément la permission (ou l'obligation ou la recommandation) et l'interdiction d'effectuer une action sur un objet. Cette situation n'est pas tout à fait impossible dans la mesure où,

à un moment donné, on peut avoir plusieurs règles (permissions, interdictions, recommandations, obligations) pour un quadruplet donné (RdO, AdO, v, c) :

- les règles figurant dans la base des faits
- celles déduites par l'application des règles d'héritage (propagation «par héritage») et des mécanismes logique de déduction (en exploitant l'axiomatique du langage, ex. toute recommandation est une permission (Recommandation(RdO, AdO, v, c) → Permission(RdO, AdO, v, c), etc.).
- celle déduite d'une intervention de l'administrateur pour modifier la base des faits.

Afin de remédier à ce type de problèmes, une vérification - hors ligne - de la cohérence de la politique de sécurité s'impose. Il faut également s'assurer que toute intervention temporaire de l'administrateur pour ajouter/modifier une règle/entité laisse la politique dans un état de cohérence.

Notons que le problème de détection des conflits dans les politiques de sécurité a fait l'objet de plusieurs travaux antérieurs, notamment pour RBAC [23], OrBAC [24] et les bases de données [25]. L'approche que nous suggérons s'appuie sur la programmation logique par contraintes (PLC).

L'idée est d'associer des priorités (un entier naturel) aux règles. Ainsi une règle classique Multi-OrBAC (figure 9-a) sera en faite remplacée par celle de la figure 9-b).

$Permission(RdO, AdO, VdO, CdO)$ $\wedge\ Joue(s, RdO) \wedge$ $Correspond\_à(o, VdO) \wedge$ $Appartient\_à(a, AdO) \wedge$ $Est\_Vrai(CdO)$ $\rightarrow Est\_permis(s, a, o).$	$Permission(RdO, AdO, VdO, CdO,$ $Priorité) \wedge Joue(s, RdO) \wedge$ $Correspond\_à(o, VdO) \wedge$ $Appartient\_à(a, AdO) \wedge$ $Est_{V}rai(CdO)$ $\rightarrow Est\_permis(s, a, o, Priorité)$

FIG. 9 – a) Règle classique Multi-OrBAC. b) Règle Multi-OrBAC avec priorités.

Lors de l'ajout d'une nouvelle règle :

- elle est maintenue dans une base temporaire ;
- si un conflit est détecté, le système identifie et liste les éléments de la politique (règles) à l'origine du conflit (soit les sous-ensembles minimaux de règles générant ce conflit)
- le système propose des corrections en invitant l'utilisateur à
  - à reformuler l'une des règles impliquées,
  - à modifier un objectif sécurité ou une règle fonctionnement,
  - ou à changer la priorité de la règle à ajouter.

Le travail présenté dans cet article a été implémenté en utilisant le langage Java ; la partie logique (base des faits "politique de sécurité ", interrogation de la politique, détection et résolution de conflits) est toutefois déléguée à un module développé en Prolog. Ce langage logique permet d'énoncer les connaissances d'un problème d'une manière déclarative tout en fournissant un mécanisme de résolution. Un programme Prolog est constitué d'un ensemble de clauses ; une

clause est une affirmation portant sur des atomes logiques exprimant une relation entre des termes ; les termes sont les objets de l'univers. Décrivons quelques règles de notre implémentation :

```
/* Exemple de règles décrivant le système*/
org(hospital).
user(jean).
role(Physician).
/* Exemple de règles d'héritage */
subject(X) :-org(X).
subject(X) :-personne(X)
/* Quelques relations du modèle Multi-OrBAC */
Play(David, Physician in Purpan).
Consulting Physician(Jean,David).
age(Jean,2).
/* Exemples de règles d'héritage */
Play_inheritance(A,X in Z) :- sub_role(Y,Z),Play(A,X in Y)
Play_inheritance(A,X in Y) :- sub_org(B,A),Play(B,X,Y).
/* Ajout d'une entité ou d'un prédicat */
Add(org(A)) :- \+ org(A), asserta(org(A)), !.
Add(org(A)) :-
org(A),nl,write('No :'),nl,write('org('),write(A),write(')'),
tab(1),write('already'),tab(1),write('exist.').
```

6 Conclusions et perspectives

Cet article présente un nouveau modèle de contrôle d'accès (Multi-OrBAC) qui couvre les particularités des systèmes complexes, hétérogènes et distribués. Comme dans OrBAC, dans Multi-OrBAC les règles de sécurité ne sont représentées qu'à l'aide d'entités abstraites. Ceci permet de structurer les sujets, objets et actions, de séparer la spécification de la politique de sécurité de son implémentation, de spécifier qu'un utilisateur peut jouer différents rôles dans différentes organisations, de montrer qu'une même vue peut correspondre à différents objets (selon l'organisation), et d'implémenter de différentes façons la même activité.

Une modélisation UML de Multi-OrBAC et des phases d'authentification, d'autorisation et de déploiement ont été très utiles pour intégrer la sécurité aux différentes phases de développement.

Par ailleurs, un formalisme logique basé sur la programmation logique par contraintes est associé à Multi-OrBAC, puis utilisé pour spécifier les règles de fonctionnement ainsi que les règles de sécurité. Il sert également à interroger la politique de sécurité, dériver les permissions par héritage, vérifier la cohérence de la politique de sécurité et détecter et résoudre les conflits.

Nous avons utilisé le langage Java pour implémenter notre prototype, sauf la partie logique qui est déléguée à un module développé en Prolog.

Actuellement, nous étudions comment automatiser le passage de la conception UML au formalisme logique (et vice-versa), comment gérer les délégations, comment intégrer des mécanismes existants d'authentification (ex. Kerberos) et les infrastructures de contrôle d'accès (ex. XACML [26]) dans notre prototype et le mettre à disposition de la communauté scientifique, comment adapter Multi-OrBAC aux services Web et d'autres types d'organisations (dynamiques, citiques, etc.), comment combiner Prolog avec d'autres langages plus puissants, et finalement comment maîtriser les problèmes liés à la complexité de notre système.

Références

[1] Common Criteria for Information Technology Security Evaluation, CC v3, Part 1 : Introduction and general model, 79 p., ISO/IEC 15408-1, July 2005.

[2] A. Abou El Kalam, Modèles et politiques de sécurité pour les domaines de la santé et des affaires sociales, Thèse de doctorat, Institut National Polytechnique de Toulouse, 198 pp., 4 décembre 2003, rapport LAAS-CNRS n°03578.

[3] F. Cuppens et N. Cuppens-Boulahia "Les modèles de sécurité",in Sécurité des systèmes d'information V.2, Traité IC2, série Réseaux et télécommunications, dir. Ludovic Mé Yves Deswarte, ISBN 2-7462-1259-5, à paraître (juin 2006).

[4] Y. Deswarte, "La sécurité des systèmes d'information et de communication", in Sécurité des réseaux et des systèmes répartis, (Yves Deswarte Ludovic Mé, eds), Traité IC2, Hermès, ISBN : 02-7462-0770-2, 264 pp, octobre 2003.

[5] B. Lampson, "Protection", 5th Princeton Symposium on Information Sciences and Systems, 1971.

[6] M.A. Harrison, W.L. Ruzzo, J.D. Ullman, "Protection in Operating Systems", Communications of the ACM, Vol. 19. N° 8, pp.461-470, 1976.

[7] A.K. Jones, R.J. Lipton, L. Snyder, "A Linear Time Algorithm for deciding Security", 17th Annual Symposium on Fundations of Computer Science, Houston, USA, 1976.

[8] D.E.Bell, L.J.LaPadula, Secure Computer Systems : Unified Exposition and Multics Interpretation, The MITRE Corporation, Technical Report, ESD-TR-73-306, 1975.

[9] K.J.Biba, Integrity Consideration for Secure Computer Systems, The MITRE Corporation, Technical Report, ESD-TR-76-372 MTR-3153, 1977.

[10] D.Clark, D.Wilson, "A comparison of Commercial and Military Computer Security Policies", IEEE Symposium on Security and Privacy, Oakland (CA, USA), 27-29 avril, 1987.

[11] D.Brewer, M.Nash, "The Chinese Wall security policy", IEEE Symposium on Security and Privacy, Oakland (CA, USA), 1-3 mai, 1989, pp 206-214.

[12] S.I. Gavrila, J.F. Barkley "Formal Specification for Role Based Access Control", Third ACM Workshop on RBAC, Fairfax (VA, USA). 22-23 octobre 1998,

[13] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn and R. Chandramouli "A Proposed Standard for Role-Based Access Control", ACM Transactions on Information and System Security, Volume 4, Number 3, août 2001.

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

[15] G. Booch, J. Rumbaugh and I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, ISBN 0-201-57168-4, USA, 1999.

[16] F. Cuppens et A. Miège, "Modelling Contexts in the Or-BAC Model", 19th Annual Computer Security Applications Conference (ACSAC'03), Las Vegas, 8-12 décembre, 2003, IEEE Computer Society.

[17] J.B. Warmer and A. Kleppe, "The Object Constraint Language", Addison-Wesley, ISBN 321179366, USA, 2003.

[18] TCSEC, Trusted Computer System Evaluation Criteria, Department of Defense (DoD), DoD Standard, DoD 5200.28-STD, 1985, 122 pp.

[19] F. Cuppens, C. Saurel, "Specifying a Security Policy : a Case Study", 9th IEEE Computer Security Foundations Workshop, Kenmare, County Kerry, Irlande, 10-12 juin 1996, IEEE Computer Society Press, pp. 123-134.

[20] R.A. Kowalski, "Predicate Logic as Programming Language", in Proc. IFIP Congress, pp. 569-574, North Holland, Amsterdam, 1974.

[21] A. Colmerauer, "An introduction to Prolog III", Communication of the ACM, vol. 33, N° 7 pp. 70-90, juillet 1990.

[22] J. Jaffar and J.L. Lassez, "Constraint Logic Programming", Communication of the ACM, vol. 33, N° 7, pp. 52 - 68, juillet 1990.

[23] M. Strembeck, "Conflict Checking of Separation of Duty Constraints in RBAC" Conference on Software Engineering, 17-19 février 2004, Innsbruck, Autriche.

[24] S. Benferhat, R. El Baida, F. Cuppens, "A Stratification-Based Approach for Handling Conflicts in Access Control", 8th ACM Symposium on Access Control Models and Technologies, Côme, Italie, 2-3 juin 2003, 189-2003, ACM press.

[25] E. Bertino, S. Jajodia et P. Samarati, "Supporting Multiple Access Control Policies in Database Systems" IEEE Symposium on Security and Privacy, Oakland (CA, USA), 1996.

[26] A. Matheus, "How to Declare Access Control Policies for XML Structured Information Objects using OASIS' eXtensible Access Control Markup Language (XACML)", 38th Annual Hawaii International Conference (HICSS), Hawaii (USA), 3-6 janvier 2005, IEEE CS Press, vol. 07, no. 7, p. 168a, 10 pp.