

MotOrBAC : un outil d'administration et de simulation de politiques de sécurité

Frédéric Cuppens, Nora Cuppens-Boulahia et Céline Coma

GET/ENST Bretagne
2 rue de la Châtaigneraie, BP 78
35512 Cesson Sévigné Cedex
France

Face à la complexité grandissante des systèmes d'information et aux difficultés de mettre en place une politique de sécurité cohérente, les administrateurs ont besoin d'outils de gestion simples et efficaces. Dans cet article, nous présentons le prototype MotOrBAC qui permet de centraliser dans un modèle unique, l'expression et l'administration de la politique de sécurité du système d'information. MotOrBAC analyse et simule une politique de sécurité spécifiée conformément au modèle OrBAC (Organization Based Access Control) [Orb, ABB⁺03, CM04b]. Ce prototype met également en œuvre l'ensemble des fonctions introduites dans le modèle AdOrBAC (Administration Organization Based Access Control) [CM04a] qui permet l'administration de OrBAC.

Nous présentons dans cet article les avantages que présente le modèle OrBAC pour développer un tel prototype. Nous étudions ensuite les diverses fonctionnalités de MotOrBAC : (1) spécification d'une politique de sécurité, (2) simulation, (3) analyse de la cohérence et (4) administration de la politique.

Mots-clés: modèle de sécurité, modélisation en logique, administration d'une politique de sécurité, OrBAC

1 Introduction

Les systèmes d'information (SI) deviennent de plus en plus complexes. Ces systèmes combinent souvent des infrastructures de réseau fixes et mobiles (sans fil), reposant sur divers systèmes d'exploitation (Windows, Linux, Unix, MacOS, etc) et fournissent de nombreuses applications (messageries, navigateurs, serveurs de bases de données, services web, etc).

Dans ce contexte, définir et ensuite gérer une politique de sécurité est une tâche complexe pour les administrateurs. Dans la grande majorité des cas, cette tâche est actuellement réalisée de façon artisanale ; les administrateurs doivent configurer manuellement les différents composants de sécurité de l'architecture du SI dont ils ont la responsabilité.

Des outils d'administration tels que Firewall Builder ou Solsoft Net-Partitioner existent. Ces logiciels sont incontestablement utiles mais se limitent à l'administration des composants de sécurité *réseau*. Les outils pour administrer les composants de sécurité système (par exemple, un système d'exploitation tel que SE-Linux) ou applicatif (par exemple un système de gestion de base de données tel que Oracle) sont beaucoup plus limités et nécessitent une expertise importante.

Le but de cet article est de présenter un prototype, MotOrBAC, qui a été développé pour administrer une politique de sécurité. MotOrBAC fournit les fonctions suivantes : (1) Expression d'une politique de sécurité basée sur le modèle OrBAC [Orb, ABB⁺03, CM04b], (2) simulation de la politique, (3) analyse de la cohérence de la politique et (4) spécification de la politique d'administration.

L'objectif de MotOrBAC est de centraliser dans un modèle de sécurité unique l'expression de toutes les exigences de sécurité réseau, système ou applicatives. Pour cela, MotOrBAC est basé sur le modèle OrBAC. OrBAC est un modèle qui permet d'exprimer une politique de sécurité au niveau *organisationnel*, c'est-à-dire indépendamment de l'implantation qui sera ensuite faite de cette politique. Il est ainsi possible d'exprimer l'ensemble des exigences de sécurité du SI et ensuite de distribuer ces exigences sur les

différents composants de sécurité, vus comme autant de sous-organisations de l'organisation que constitue le SI. Cette distribution porte également sur les responsabilités d'administration que l'on peut confier à des sujets affectés à des rôles différents.

Une fois la politique de sécurité exprimée au niveau organisationnel, l'administrateur de sécurité peut utiliser MotOrBAC pour introduire les entités concrètes (sujets, actions et objets) auxquelles la politique s'applique. La fonction de simulation permet alors de tester la politique de sécurité organisationnelle sur ces entités concrètes.

L'intérêt de centraliser l'expression de la politique de sécurité est de disposer d'une vision globale de la politique dont on peut analyser la cohérence. Comme le modèle OrBAC offre la possibilité d'exprimer une politique de sécurité mixte incluant des privilèges positifs (permissions) et négatifs (interdictions), des conflits peuvent apparaître. L'objectif de la fonction d'analyse de la cohérence est de détecter ces conflits et ensuite d'offrir la possibilité de définir des stratégies pour résoudre ces conflits.

Enfin, la fonction d'administration sert à spécifier qui a la responsabilité d'exprimer et ensuite mettre à jour la politique de sécurité du système d'information. Actuellement, la plupart des modèles de sécurité repose sur un administrateur de sécurité unique qui a les pleins pouvoirs pour définir la politique de sécurité (réseau, système ou applicative). Cette hypothèse d'un administrateur unique n'est plus réaliste dans les infrastructures de SI de plus en plus distribuées. La fonction d'administration implantée dans MotOrBAC repose sur le modèle AdOrBAC [CM04a] qui permet de définir une politique d'administration en utilisant les mêmes concepts que OrBAC (ce qui fait de OrBAC un modèle auto-administré). En utilisant AdOrBAC, il est possible de distribuer l'administration de la politique sur plusieurs rôles n'ayant chacun que des droits d'administration restreints. Dans MotOrBAC, l'expression de la politique d'administration est utilisée comme politique de contrôle d'accès appliquée aux administrateurs qui accèdent à MotOrBAC.

Le plan du reste de l'article est le suivant. Dans la première section, nous motivons l'implémentation d'un prototype de simulation et d'analyse de politique de sécurité reposant sur le modèle OrBAC. La section 2 justifie les différentes fonctionnalités offertes par MotOrBAC, rappelle les concepts introduits dans le modèle OrBAC et présente les principes généraux du prototype que nous avons développé. La section 3 explique comment définir une politique de sécurité en utilisant MotOrBAC. La section 4 décrit la fonction de simulation de la politique de sécurité au niveau concret et la section 5 présente la fonction de gestion des conflits. Dans la section 6, nous montrons comment MotOrBAC implante le modèle d'administration AdOrBAC. Cet article se conclut sur la section 7 par les perspectives d'évolution de MotOrBAC.

2 Nécessité d'un outil d'administration

2.1 Motivation

Actuellement, les sociétés privées et les organismes publics rencontrent de nombreux problèmes pour mettre en place la politique de sécurité de leur SI. Pour illustrer certains de ces problèmes, intéressons-nous à une organisation et à ses besoins de définition et de gestion de la sécurité de son SI. Une société *WorldCompany* possède de nombreuses filiales (*FranceCompany*, *EnglandCompany*,...). Elle sous-traite à d'autres compagnies (*TaiwanSousTrait*) une partie des ses activités. Elle possède plusieurs administrateurs (*François*, *Ali* et *Juda*). Toutes les filiales de la *WorldCompany* traitent le même produit. Leur hiérarchie, leur politique et les métiers qui y sont définis sont similaires. *WorldCompany* souhaite définir une politique de sécurité identique et cohérente pour toutes ses filiales. Cependant, *WorldCompany* rencontre des difficultés. En effet, *FranceCompany* et *EnglandCompany* sont des filiales situées dans deux pays différents, ne possédant pas la même législation. Les administrateurs de *WorldCompany* doivent adapter les politiques de ces filiales afin qu'elles soient compatibles avec les législations du pays où elles se situent. Lors de cette adaptation, *WorldCompany* se rend compte que l'un de ses administrateurs, *Juda*, définit des règles dans la politique de sécurité qui laissent des informations trop vulnérables. Ne sachant pas si *Juda* est un administrateur malveillant, incompetent ou s'il lui manque les éléments nécessaires pour avoir une visibilité globale sur l'ensemble du SI, *WorldCompany* souhaite réduire les droits de cet administrateur. D'autre part, *WorldCompany* doit autoriser un certain nombre d'accès à ses sous-traitants, et les modifier selon les rôles qu'ils jouent dans la société. De plus, les administrateurs de *WorldCompany* doivent gérer

les départs en retraite, les stagiaires, etc. Ils doivent pouvoir modifier facilement la politique de sécurité en cas de changement radical de la politique, comme le passage des 39 heures aux 35 heures, en France.

Il est clair que, parmi les problèmes rencontrés par la *WorldCompany* et qui peuvent se poser à n'importe quelle organisation désireuse de mettre en place une politique de sécurité, certains sont liés à des restructurations, d'autres à la vérification de la cohérence de la politique ou encore à la centralisation et la spécification des droits d'administration. Il est difficile pour les administrateurs d'avoir une vision globale de la politique de sécurité afin de la gérer correctement. C'est l'une des raisons pour laquelle l'utilisation d'un logiciel d'administration d'utilisation aisée leur permettraient de visualiser et d'obtenir rapidement les informations dont ils ont besoin. Cependant, afin que ce logiciel soit efficace, il doit reposer sur un modèle qui offre les propriétés nécessaires pour répondre aux problèmes de spécification et de gestion de la politique de sécurité tels que ceux qui se posent à la société *WorldCompany*. Ce modèle doit permettre de gérer simplement de nombreuses entités, possédant divers modes administratifs (centralisé ou non, un super administrateur ou plusieurs administrateurs ayant des droits restreints, contextuels,...) et permettant de vérifier la cohérence de la politique. L'état de l'art dans le domaine nous a amenés à la conclusion que le modèle OrBAC est l'un des modèles qui convient le mieux aux spécifications requises par ce logiciel.

2.2 OrBAC

L'objectif d'OrBAC [Orb, ABB⁺03, CM04b] est de modéliser une politique de sécurité centrée sur l'organisation qui la définit ou en a la charge. Par conséquent, une entreprise est une organisation mais un composant de sécurité tel qu'un pare-feu correspond également à une organisation. La spécification d'une politique OrBAC se fait au niveau organisationnel (dit abstrait) indépendamment de son implantation. La politique implantée (dite concrète) est dérivée de la politique organisationnelle. Cette approche rend toute politique exprimée dans le modèle OrBAC reproductible et évolutive. En effet, elle ne nécessite aucun réajustement au niveau concret qui pourrait introduire des incohérences difficilement récupérables ; tout se fait au niveau organisationnel. Le niveau concret est constitué de sujets, d'actions et d'objets. Le niveau organisationnel contient les rôles, les activités et les vues. Le rôle (respectivement l'activité, la vue) est un ensemble de sujets (respectivement d'actions, d'objets) sur lesquels sont appliquées les mêmes règles de sécurité.

Le modèle OrBAC repose sur un formalisme basé sur la logique du premier ordre avec négation. Cependant, comme la logique du premier ordre est en général indécidable, nous avons restreint le modèle pour qu'il soit compatible avec un programme Datalog stratifié [Ull89]. Un programme Datalog ne doit pas utiliser de termes fonctionnels et ne doit inclure que des règles définies et sûres. Une règle est définie si chaque variable qui apparaît dans la conclusion de la règle apparaît également dans la prémisse. Une règle est sûre si elle ne permet de dériver qu'un ensemble fini de faits. Dans toute règle, les variables sont universellement quantifiées. Les littéraux négatifs sont autorisés dans la prémisse mais les règles doivent pouvoir être *stratifiées*. La stratification d'un programme Datalog consiste à ordonner les règles ; si une règle contient un littéral négatif, alors les règles qui définissent ce littéral sont évaluées d'abord. Un programme Datalog stratifié est calculable en temps polynomial.

Dans la suite, toutes les règles définissant la politique de sécurité doivent donc correspondre à un programme Datalog stratifié. Pour exprimer ces règles, nous allons en fait utiliser une notation à la Prolog[†]. Les termes commençant par une majuscule correspondent à des variables alors que les termes commençant par une minuscule, par exemple `jean`, correspondent à des constantes. Un fait, tel que `parent(jean,marie)`, indique que `jean` est un parent de `marie`. Une règle telle que

$$\text{grand_parent}(X, Z) \text{ :- } \text{parent}(X, Y), \text{parent}(Y, Z).$$

indique que `X` est un grand-parent de `Z` s'il existe un sujet `Y` tel que `X` est un parent de `Y` et `Y` est un parent de `Z`.

En utilisant ce formalisme, chaque organisation peut ensuite définir des règles de sécurité pour spécifier que certains rôles ont la permission, l'interdiction ou l'obligation de réaliser certaines activités sur certaines vues. Ces règles de sécurité ne sont pas statiques mais leur activation dépend au contraire de conditions contextuelles. Pour cela, le concept de *contexte* est explicitement introduit dans OrBAC et intervient

[†] Il est cependant important de signaler que les administrateurs introduisent les règles via une interface et qu'ils n'ont, en général, pas à introduire de règles logiques lorsqu'ils utilisent MotOrBAC.

dans la définition des règles de sécurité. Ainsi, en utilisant notre formalisme basé sur la logique du premier ordre, les règles de sécurité sont représentées en utilisant trois prédicats d'arité 5 : *permission*, *prohibition* et *obligation*. Par exemple :

- *permission(org, role, activity, view, context)* signifie que, dans l'organisation *org*, les sujets affectés à *role* ont la permission de réaliser *activity* sur la vue *view* dans le contexte *context*.

Les prédicats *prohibition* et *obligation* sont définis de façon similaire. Par exemple, la permission suivante :

- *permission(hospital, nurse, consult, medical_record, urgency)*

spécifie que, dans l'organisation *hospital*, les infirmiers ont la permission de consulter les dossiers médicaux dans un contexte d'urgence.

Tous ces concepts d'organisation, de rôle, d'activité, de vue et de contexte, peuvent être structurés hiérarchiquement. Les permissions, interdictions et obligations sont alors hérités lorsqu'on s'élève dans la hiérarchie (voir [CCBM04] pour plus de détails). Comme une politique de sécurité peut inclure des règles de sécurité conflictuelles (par exemple conflit entre une permission et une interdiction), il est nécessaire de définir des stratégies de résolution de conflits. Comment ce problème est résolu dans OrBAC est présenté dans la section 4.

Une fois définie la politique de sécurité organisationnelle, il est possible de tester comment cette politique s'applique aux entités concrètes que sont les sujets, actions et objets. Pour cela, nous avons introduit 3 prédicats d'arité 3 pour affecter un sujet à un rôle (resp. une action à une activité) (resp. un objet à une vue) :

- *empower(org, subject, role)* : spécifie que dans l'organisation *org*, le sujet *subject* est affecté au rôle *role*.
- *consider(org, action, activity)* : spécifie que dans l'organisation *org*, l'action *action* implante l'activité *activity*.
- *use(org, object, view)* : spécifie que dans l'organisation *org*, l'objet *object* est utilisé dans la vue *view*.

Par exemple, le fait *empower(hospital, jean, physician)* spécifie que *jean* est affecté au rôle *physician* dans l'organisation *hospital*.

Les contextes sont définis par des règles logiques qui caractérisent dans quelles conditions le contexte est actif. Dans le modèle OrBAC, ceci est modélisé par des règles logiques ayant le prédicat *hold* comme conclusion. *hold* est un prédicat d'arité 5 défini de la façon suivante :

- *hold(org, subject, action, object, context)* : spécifie que dans l'organisation *org*, le sujet *subject* réalise l'action *action* sur l'objet *object* dans le contexte *context*.

En utilisant ce modèle, il est ensuite possible de dériver les autorisations concrètes qui s'appliquent aux sujets, actions et vues. Le principe général de dérivation des autorisations concrètes à partir des autorisations organisationnelles est présenté dans la section 5 et est utilisé pour simuler la politique de sécurité au niveau concret.

2.3 MotOrBAC

Afin que des administrateurs puissent définir facilement une politique de sécurité basée sur le modèle OrBAC, nous avons développé le prototype MotOrBAC. L'architecture de MotOrBAC est présentée dans la figure 1. Ce prototype est composé de quatre modules : (1) l'analyseur de cohérence de la politique, (2) la sauvegarde des données (en XML ou Prolog), (3) le module de communication et (4) l'interface graphique (cf. figure 2).

Le prototype MotOrBAC assure cinq fonctionnalités :

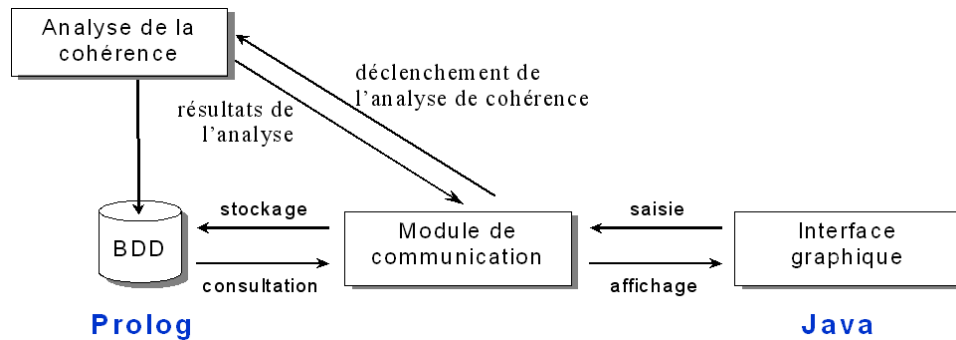


Fig. 1: Architecture de MotOrBAC

- Saisie d'une politique de sécurité : l'administrateur peut introduire avec MotOrBAC les différentes entités spécifiques au SI dont il gère la sécurité (organisations et sous-organisations, rôles, activités, vues et contextes) et les règles de sécurité associées (voir section 3).
- Simulation de la politique : MotOrBAC permet de simuler la politique en saisissant les sujets, actions et objets de l'organisation et en dérivant automatiquement la politique au niveau concret à partir de la politique organisationnelle introduite par les administrateurs. Les sujets, actions, objets sont caractérisés par des attributs (voir la section 4).
- Vérification de la cohérence de la politique de sécurité : MotOrBAC permet de détecter les conflits au niveau concret ou abstrait de la politique de sécurité spécifiée par l'administrateur (voir section 5.1).
- Résolution de conflits : une fois des conflits détectés, MotOrBAC intègre des stratégies de résolution de conflits qu'il applique à la politique de sécurité introduite par l'administrateur pour rétablir la cohérence (voir la section 5.2).
- Gestion des droits d'administration : MotOrBAC permet de spécifier les droits donnant à un sujet affecté à un rôle d'administration la possibilité de gérer tout ou partie d'une politique de sécurité OrBAC (voir section 6).

3 Spécification d'une politique de sécurité

3.1 Les entités organisationnelles d'OrBAC dans MotOrBAC

Lorsqu'un administrateur désigné souhaite définir la politique de sécurité d'une organisation, MotOrBAC va d'abord demander à l'administrateur d'introduire le nom de cette organisation. Cet administrateur peut ensuite définir plusieurs sous-organisations. Ceci conduit à l'insertion dans la politique de sécurité de faits de la forme :

- `sub_organisation(org1, org2)` : org1 est une sous-organisation de org2.

L'administrateur a alors le choix de visualiser uniquement la politique de sécurité liée à une sous-organisation ou à une organisation et à toutes ses sous-organisations grâce à un menu déroulant.

L'administrateur peut ensuite définir pour chaque organisation, les différents rôles, activités et vues ainsi que les contextes. Ainsi, pour créer un rôle via MotOrBAC, l'onglet *Role* prévu à cet effet permet de renseigner le nom du rôle ainsi que les relations d'héritage existant entre ce rôle et les autres rôles déjà définis. Par exemple, supposons que l'administrateur crée, dans l'organisation `hospital`, le rôle `head_nurse` et spécifie que ce rôle est hiérarchiquement supérieur au rôle `nurse`. Ceci conduit à l'insertion dans la politique de sécurité des faits suivants :

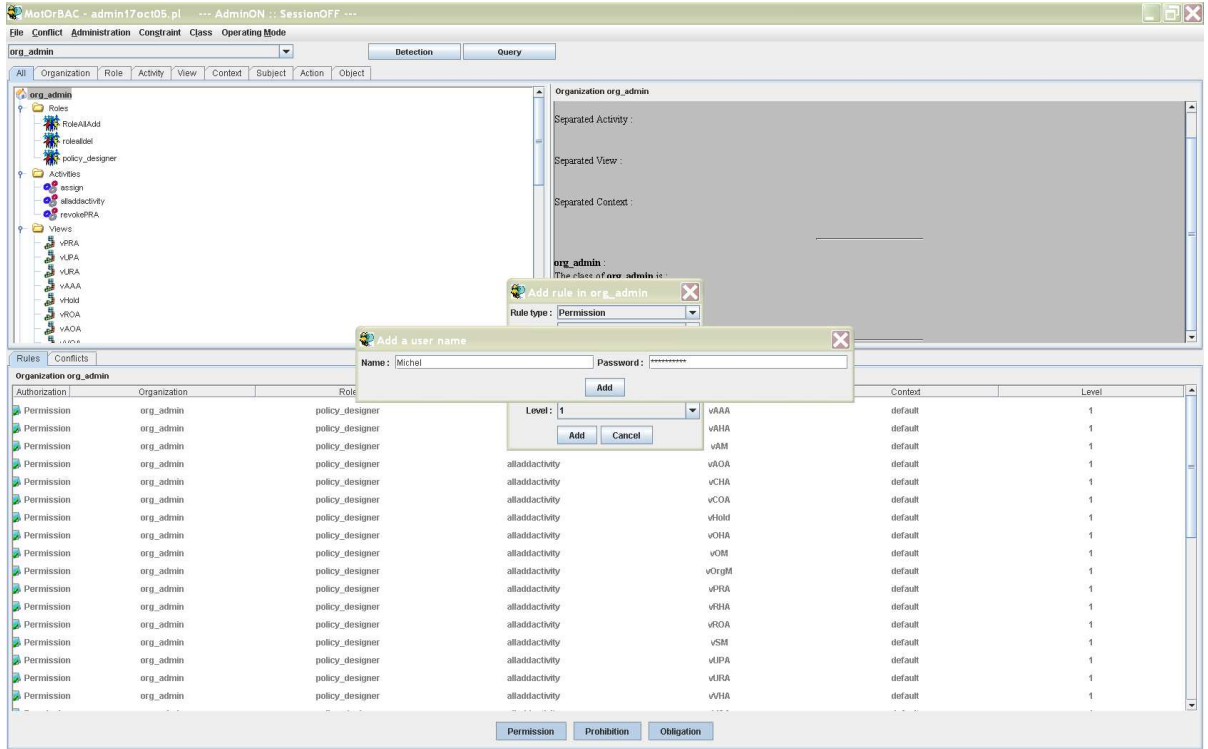


Fig. 2: Interface de MotOrBAC

- $\text{use}(\text{hospital}, \text{head_nurse}, \text{role})$: l'organisation hospital utilise head_nurse dans la vue role[‡].
- $\text{senior_role}(\text{hospital}, \text{head_nurse}, \text{nurse})$: dans l'organisation hospital, le rôle head_nurse est hiérarchiquement supérieur au rôle nurse.

Lors de la création des entités organisationnelles, il est également possible de définir des *contraintes* que doivent respecter ces entités. Toute mise à jour de la politique de sécurité qui viole une de ces contraintes est rejetée. Dans OrBAC, une contrainte est modélisée par une règle qui dérive sur le prédicat `error` d'arité 0. Certaines contraintes comme la séparation de rôle ont un format générique prédéfini dans OrBAC. Cette contrainte utilise le prédicat `separated_role` d'arité 4 et correspond à la règle suivante :

`error :-`

```
separated_role(Org1, Role1, Org2, Role2),
empower(Org1, Subject, Role1),
empower(Org2, Subject, Role2).
```

Cette règle spécifie que si une règle de séparation de rôles existe entre `Role1` et `Role2` dans les organisations `Org1` et `Org2`, alors aucun sujet `Subject` ne peut être simultanément affecté au rôle `Role1` dans l'organisation `Org1` et au rôle `Role2` dans l'organisation `Org2`.

MotOrBAC permet de spécifier simplement ce type de contraintes de séparation entre rôles. Des contraintes de séparation similaires peuvent être spécifiées pour les vues, les activités et les contextes. La définition d'autres types de contrainte est possible dans MotOrBAC en saisissant une expression Prolog qui définit la règle correspondant à la contrainte.

Enfin, MotOrBAC offre naturellement à l'administrateur un menu permettant de créer des règles de sécurité. Une règle de sécurité est définie dans l'organisation spécifiée (sélectionnée dans un menu déroulant),

[‡] Le modèle OrBAC inclut quatre vues prédéfinies `role`, `activity`, `view` et `context` qui permettent respectivement de gérer les rôles, activités, vues et contextes de chaque organisation.

pour un type de privilège (permission, interdiction, obligation). Cette règle s'applique à un rôle, une activité, une vue et un contexte de l'organisation courante et possède un niveau de priorité. Ce niveau de priorité associé à chaque règle de sécurité est utilisé pour gérer les éventuels conflits entre règles (voir section 5 ci-dessous). Par exemple, on peut définir la règle de sécurité suivante :

```
permission(hospital, nurse, consult, medical_record, urgency, 1).
```

Cette règle spécifie que, dans l'organisation `hospital`, la permission accordée aux infirmiers de consulter les dossiers médicaux dans un contexte d'urgence, a un niveau de priorité égal à 1.

L'interface (cf. fig. 2) permet également de visualiser toutes les entités organisationnelles (rôle, activité, vue, contexte,...) ainsi que les informations qui les concernent (privilèges, entités concrètes affectées, ...).

3.2 Héritages

Afin de gérer plus facilement des sous-organisations, en automatisant la dérivation des privilèges, OrBAC permet de définir des hiérarchies de rôles, d'activités, de vues et de contextes [CCBM04]. Les entités dans les niveaux hauts des hiérarchies, héritent des privilèges des entités hiérarchiquement inférieures. MotOrBAC permet d'introduire facilement des relations d'héritage, en demandant systématiquement lors de la création d'une entité abstraite si l'administrateur souhaite en faire une sous-entité d'une autre entité. Si c'est le cas, l'administrateur peut faire appel à un menu déroulant qui lui propose les entités auxquelles il peut associer une relation de hiérarchie avec l'entité qu'il souhaite créer.

Il existe quatre grands mécanismes d'héritage dans OrBAC :

L'héritage des privilèges liés à la hiérarchie des entités abstraites : dans une même organisation, si une règle de sécurité s'applique à une entité abstraite (rôle, activité ou vue), alors ses sous-entités (respectivement sous-rôle, sous-activité ou sous-vue) héritent de cette règle de sécurité. Par exemple, l'héritage des permissions dans la hiérarchie de rôles correspond à la règle suivante :

```
permission (Org, Senior_role, Activity, View, Context, Priority) :-  
    permission (Org, Junior_role, Activity, View, Context, Priority),  
    senior_role (Org, Senior_role, Junior_role).
```

L'héritage des contraintes de séparation : les contraintes de séparation sont héritées à travers les hiérarchies d'entités abstraites à l'instar de l'héritage des privilèges.

L'héritage des privilèges liés à la hiérarchie d'organisation : les organisations héritent des privilèges à travers la hiérarchie d'organisations si les rôles, les activités, les vues et les contextes impliqués dans les règles de sécurité sont définis dans ces organisations. Supposons qu'une règle de sécurité r est associée à une organisation `Org` et que soit définie une sous-organisation `Sorg` de `Org`. `Sorg` hérite de r si le rôle, l'activité, la vue et le contexte impliqués dans r sont définis dans `Sorg` :

```
permission (Sorg, Role, Activity, View, Context, Priority) :-  
    sub_organization (Sorg, Org),  
    use (Sorg, Role, role),  
    use (Sorg, Activity, activity),  
    use (Sorg, View, view),  
    use (Sorg, Context, context),  
    permission (Org, Role, Activity, View, Context, Priority).
```

L'héritage de la définition des contextes : si un contexte est défini dans une organisation et que cette organisation possède une sous-organisation, la sous-organisation hérite de la définition du contexte sauf mention explicite du contraire.

4 Simulation

4.1 Création des entités concrètes

Afin de pouvoir simuler une politique de sécurité au niveau concret, il convient d'introduire, dans MotOrBAC, les entités concrètes du modèle OrBAC. Nous suivons pour cela une approche orientée-objet. Toutes les entités du modèle correspondent donc à des objets. Chaque objet a un identifiant et possède un ensemble d'attributs utilisés pour décrire l'objet.

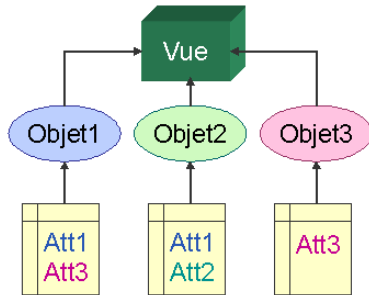


Fig. 3: Vues et attributs

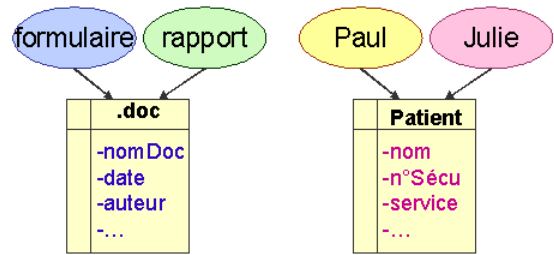


Fig. 4: Classes et attributs

Dans le formalisme logique du modèle OrBAC, les attributs sont représentés par des prédicats binaires. Par exemple, le fait $\text{age}(s1, 20)$ exprime que $s1$ a un attribut age qui a pour valeur 20. Les objets appartiennent à des classes ce qui est représenté en utilisant un prédicat binaire class . Par exemple, le fait $\text{class}(s1, \text{student})$ indique que $s1$ appartient à la classe student .

Les sujets et les actions correspondent en fait à deux sous-classes particulières d'objets. Les sujets sont des entités actives qui peuvent réaliser des actions sur d'autres objets. Les actions sont des programmes concrets que l'on peut voir comme des méthodes, mais ceci reste optionnel dans le modèle OrBAC. Les organisations du modèle sont aussi des objets. Elles possèdent donc des attributs et appartiennent à des classes, par exemple la classe des hôpitaux.

En utilisant l'interface de MotOrBAC, un utilisateur peut créer de nouveaux objets, spécifier la classe d'appartenance de ces objets et renseigner leurs différents attributs. L'utilisateur peut ensuite affecter une entité concrète à une entité organisationnelle. Pour cela, il doit sélectionner une organisation et affecter un objet à une vue, respectivement un sujet à un rôle ou bien une action à une activité. Si la politique de sécurité de l'organisation considérée autorise cette affectation, alors une nouvelle instance du prédicat use , empower ou consider sera insérée dans la politique de sécurité.

Il faut insister sur le fait que le concept de vue du modèle OrBAC est distinct de celui de classe. Une classe est un concept taxonomique utilisé pour structurer la description des objets alors qu'une vue est un concept organisationnel qui sert à structurer la spécification de la politique de sécurité. On peut également remarquer que les objets affectés à une vue peuvent avoir des attributs différents (cf. figure 3), alors que ceux appartenant à une classe ont les mêmes attributs, seules leurs valeurs diffèrent (cf. figure 4).

Cependant, une classe peut être utilisée comme une vue organisationnelle si cette classe sert à spécifier la politique de sécurité. Un objet peut appartenir à une classe sans être utilisé par une organisation dans la vue correspondante. Par exemple, si dossier_medical est à la fois une classe et une vue, alors certains objets de cette classe peuvent être utilisés dans l'organisation purpan_hospital mais pas dans l'organisation rangueil_hospital . De plus, plusieurs organisations peuvent dans certains cas partager des objets dans la même vue ou dans des vues différentes. De façon similaire, un même sujet peut être affecté, dans plusieurs organisations, à des rôles différents.

L'interface de MotOrBAC permet de gérer ces différentes situations.

4.2 Dérivation des privilèges concrets

Une fois les entités concrètes définies dans MotOrBAC, on peut appliquer la politique de sécurité organisationnelle aux entités concrètes pour dériver si tel sujet a la permission de réaliser telle action sur tel objet. Pour cela, nous introduisons le prédicat is_permitted d'arité 4 et appliquons le principe général de dérivation des permissions concrètes :

```
is_permitted(Subject, Action, Object, Priority) :-
    permission(Org, Role, Activity, View, Context, Priority),
    empower(Org, Subject, Role),
    consider(Org, Action, Activity), use(Org, Object, View),
    hold(Org, Subject, Action, Object, Context) .
```

Cette règle spécifie que dans une certaine organisation si (1) une permission existe pour un rôle, une activité et une vue dans un certain contexte, (2) un sujet est habilité dans ce rôle, (3) une action implémente cette activité, (4) un objet est utilisé dans cette vue et (5) ce contexte est actif pour ce sujet, cette action et cette vue, alors ce sujet a une permission concrète de réaliser cette action sur cet objet. Cette permission concrète a le même niveau de priorité que la permission organisationnelle qui a permis de la dériver.

Des règles similaires permettent de dériver les interdictions et les obligations au niveau concret. Un administrateur peut demander à MotOrBAC d'appliquer ces règles pour connaître l'ensemble des privilèges dérivables au niveau concret. En utilisant l'interface, l'administrateur peut également formuler une requête logique pour connaître l'ensemble des privilèges qui s'appliquent aux entités (sujets, actions et objets) qui satisfont la requête.

Le niveau de priorité associé au privilège dérivé permet de résoudre les éventuels conflits entre permission et interdiction d'une part et entre obligation et interdiction d'autre part. Ceci permet de tester la présence de conflits au niveau concret compte tenu de l'ensemble d'entités concrètes considérées et de vérifier si ces conflits peuvent être résolus en appliquant les niveaux de priorité.

Cependant, il est possible que la création de nouvelles entités concrètes créent de nouveaux conflits qui ne pourraient pas être résolus. C'est la raison pour laquelle nous proposons, dans la section 5, une démarche pour résoudre les conflits au niveau organisationnel. L'objectif de cette démarche est de pouvoir garantir que s'il n'existe pas de conflit au niveau organisationnel, alors il ne pourra pas en exister au niveau concret.

5 Analyse

5.1 Les conflits dans OrBAC

La fonction de simulation décrite dans la section précédente permet de mettre en évidence des conflits au niveau concret. Dans cette section, nous présentons une autre fonction offerte par MotOrBAC pour analyser les conflits au niveau organisationnel. Pour détecter ces conflits, nous introduisons d'abord le prédicat d'arité 0 `conflict`. La présence de conflits est ensuite détectée en appliquant la règle suivante :

`conflict :-`

```
permission(Org1, Role1, Activity1, View1, Context1, Priority1),
prohibition(Org2, Role2, Activity2, View2, Context2, Priority2),
not (separated_role(Org1, Role1, Org2, Role2)),
not (separated_activity(Org1, Activity1, Org2, Activity2)),
not (separated_view(Org1, View1, Org2, View2)),
not (separated_context(Org1, Context1, Org2, Context2)),
not (Priority1 < Priority2),
not (Priority2 < Priority1).
```

Cette règle indique que si (1) une permission et une interdiction organisationnelles existent dans les organisations `Org1` et `Org2`, (2) il n'existe pas de contraintes de séparation entre les rôles, activités, vues et contextes et (3) les priorités associées respectivement à la permission et à l'interdiction ne sont pas comparables, alors un conflit est dérivé.

En effet, si cette situation se produit, il est possible d'affecter un sujet simultanément aux rôles `Role1` et `Role2` (idem pour une action aux activités `Activity1` et de `Activity2` et un objet aux vues `View1` et de `View2`) et de dériver une permission et une interdiction conflictuelle au niveau concret en appliquant le principe général de dérivation des privilèges présenté dans la section 4.2. A noter que ce conflit au niveau concret est seulement *potentiel* dans la mesure où le sujet, l'action et l'objet pouvant générer le conflit n'existe peut-être pas au niveau concret.

Une règle similaire à la règle ci-dessus permet de détecter les conflits entre les interdictions et les obligations.

Lorsqu'il n'est pas possible de dériver le prédicat `conflict`, on dit que la politique de sécurité est *cohérente*. L'intérêt de cette démarche de détection des conflits au niveau organisationnel est le suivant. Une fois que la politique organisationnelle est cohérente, l'utilisateur peut affecter des entités concrètes aux entités abstraites sans risque d'introduire de conflit au niveau concret. En effet, on peut démontrer que s'il n'existe pas de conflit au niveau organisationnel, alors il n'en existera pas au niveau concret [CCB05].

En utilisant la fonction de détection de MotOrBAC, l'utilisateur peut, grâce à une option, demander de visualiser aussi bien les conflits organisationnels que concrets.

5.2 Résolution des conflits dans MotOrBAC

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

Modifier une des règles conflictuelles : L'utilisateur peut considérer que le conflit détecté est dû à une erreur dans la spécification de la politique. Dans ce cas, il peut mettre à jour la politique en modifiant une des règles conflictuelles.

Ajouter une (ou plusieurs) contrainte(s) de séparation : L'utilisateur peut ajouter des contraintes de séparation. Par exemple, en introduisant une contrainte de séparation entre les rôles *nurse* et *physician*, on a l'assurance que les privilèges de ces deux rôles ne peuvent plus rentrer en conflit.

Modifier le niveau de priorité d'une des règles conflictuelles : Changer le niveau de priorité d'une règle est un moyen simple pour résoudre le conflit. Cependant, il faut s'assurer que cette modification n'a pas pour conséquence de rendre l'une des règles redondante ou inapplicable. Nous ne développons pas dans cet article le problème des règles redondantes. Il s'agit d'un problème complexe et nous renvoyons à [CCBGA05] où est défini un algorithme pour détecter ce type d'anomalie dans une politique de sécurité réseau.

Ignorer le conflit : L'utilisateur peut tout simplement ignorer le conflit. Cependant tout conflit non résolu au niveau organisationnel peut générer des conflits au niveau concret.

6 AdOrBAC

6.1 Administration par vue

Un modèle de contrôle d'accès est généralement constitué de deux parties distinctes : un modèle pour exprimer une politique d'autorisation et un modèle d'administration de cette politique. Le modèle d'administration permet de spécifier qui possède des privilèges pour mettre à jour la politique d'autorisation et sous quelles conditions. Par exemple, le modèle ARBAC [SBM97, SM99] a été défini pour administrer le modèle de contrôle d'accès RBAC [SCFY96, FSG⁺01].

Le modèle OrBAC possède également son modèle d'administration, le modèle AdOrBAC [CM04a]. AdOrBAC permet de spécifier une politique d'administration dans le même formalisme logique et en utilisant les mêmes concepts que ceux introduits pour définir le modèle OrBAC. AdOrBAC fait donc du modèle OrBAC un modèle *auto-administré*. Le principe de base du modèle AdOrBAC est de considérer que toutes les opérations d'administration de la politique de sécurité doivent être réalisées en gérant, c'est-à-dire en créant, modifiant ou supprimant, des objets particuliers appelés objets d'administration. L'expression de la politique d'administration consiste à spécifier qui possède des privilèges de gérer ces objets d'administration.

Pour être conforme avec le modèle OrBAC, les objets d'administration sont regroupés dans différentes vues (voir figure 5). Comme dans OrBAC, les objets affectés à ces vues sont spécifiés en utilisant le prédicat *use*. AdOrBAC permet d'administrer les activités suivantes : (1) gestion des permissions (*pra*), (2) gestion des affectations de rôles aux sujets et d'activités aux actions (*ura*, *aaa*), (3) gestion des hiérarchies d'entités organisationnelles (*rha*, *aha*, *vha*, *oha*, *cha*), (4) gestion des définitions de contextes (*hold*). Nous étudions de façon plus détaillée la gestion des privilèges via la vue *pra* dans la section suivante. Les autres fonctions d'administration sont présentées dans [CM04a].

6.2 Gestion des privilèges

La vue *pra* (Permission Role Assignment) permet de gérer les affectations des privilèges aux rôles. Cette vue est aussi une classe munie d'attributs permettant de décrire les différents paramètres d'un privilège. Ainsi, on définit les attributs suivants pour la classe *pra* : (1) *type* (le type de privilège {*permission*, *prohibition*, *obligation*}), (2) *authority* (l'organisation dans laquelle s'applique le privilège),

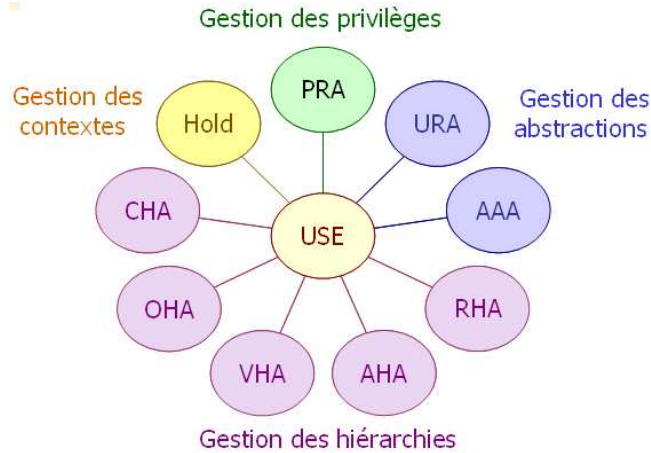


Fig. 5: Les vues administratives

(3) *grantee* (le rôle qui possède le privilège), (4) *privilege* (l'activité qui pourra être réalisée lorsque le privilège est utilisé), (5) *target* (la vue sur laquelle porte le privilège), (6) *context* (le contexte d'utilisation du privilège) et (7) *level* (le niveau de priorité associé au privilège et qui est utilisé pour résoudre les conflits).

L'insertion d'un nouvel objet dans la vue *pra* est interprétée comme la création d'un nouveau privilège. Pour cela, nous définissons une règle pour dériver automatiquement une permission à partir des objets affectés à la vue *pra* et ayant *permission* comme valeur pour l'attribut *type* :

```
permission(Auth, Role, Activity, View, Context, Priority) :-
    use(Org, ObjAdmin, pra),
    type(ObjAdmin, permission),
    authority(ObjAdmin, Auth),
    grantee(ObjAdmin, Role),
    privilege(ObjAdmin, Activity), target(ObjAdmin, View),
    context(ObjAdmin, Context), level(ObjAdmin, Priority),
    sub_organization(Org, Auth).
```

On peut remarquer dans cette règle qu'un objet administratif appartenant à la vue *pra* n'est interprété comme une permission que si l'organisation *Auth* apparaissant dans l'attribut *authority* est une sous-organisation de l'organisation *Org* qui utilise cet objet administratif. Cette contrainte est appelée *principe de confinement hiérarchique* et impose qu'une organisation ne peut contrôler la politique de sécurité que d'une de ses sous-organisations.

Deux autres règles similaires à la règle ci-dessus permettent de dériver des interdictions et des obligations lorsqu'un objet appartenant à la vue *pra* a une valeur de l'attribut *type* égale à *prohibition* ou *obligation*.

6.3 AdOrBAC dans MotOrBAC

MotOrBAC inclut une fonction d'administration implantant le modèle AdOrBAC. Une fois authentifiée par MotOrBAC, un utilisateur ayant des privilèges administratifs, c'est-à-dire ayant la permission de créer des objets appartenant aux vues administratives du modèle AdOrBAC, peut ainsi créer une politique de sécurité correspondant au modèle OrBAC. Dans [CM04a], nous avons montré que notre approche permettait de modéliser une politique d'administration distribuée entre plusieurs administrateurs ayant chacun des privilèges d'administration restreints. On peut ainsi s'affranchir de l'hypothèse du super administrateur unique réunissant tous les droits d'administration. On peut également spécifier des politiques d'administration flexibles, par exemple une politique qui donne des privilèges d'administration sous certaines conditions (pour la durée d'une journée ou lorsque un administrateur titulaire est absent).

Une fois que la politique d'administration de l'organisation est définie, cette politique est appliquée pour contrôler les actions effectuées par les utilisateurs de MotOrBAC. Ainsi, chaque fois qu'un utilisateur de MotOrBAC tente d'exécuter une action (par exemple, insérer un objet dans la vue `pra` pour créer une nouvelle permission), la politique d'administration est appliquée pour vérifier que cet utilisateur a effectivement la permission de créer cet objet. Si tel n'est pas le cas, la tentative de création sera rejetée.

7 Conclusion

Nous avons montré dans cet article qu'il est possible de fournir à des administrateurs de sécurité les moyens de spécifier de façon aisée leur politique de sécurité et d'en avoir la maîtrise : (1) un modèle de sécurité structuré autour d'entités organisationnelles auxquelles l'administrateur est rapidement familiarisé (organisation, rôle, activité, vue, contexte), (2) une séparation claire entre le niveau organisationnel et le niveau concret (sujet, action, objet), (3) un modèle de privilèges riches incluant permission, interdiction et obligation, (4) un outil de simulation, de visualisation et d'analyse de la politique, (3) une automatisation de la détection et de la résolution des conflits éventuels et (5) une administration décentralisée de la politique de sécurité. L'administrateur peut introduire sa politique de sécurité avec MotOrBAC directement à partir de l'expression de cette politique dans le modèle OrBAC. Les entités manipulées dans ce modèle étant assez intuitives, il devrait pouvoir les spécifier de façon naturelle. Les enrichissements apportés au modèle OrBAC de base sont également pris en compte dans le prototype MotOrBAC. Ainsi, l'administrateur peut affiner sa politique de sécurité et alléger de ce fait sa gestion par l'introduction des différentes hiérarchies d'entités OrBAC ainsi que des contraintes qui régissent les différentes affectations d'entités organisationnelles. Les entités concrètes sont décrites en utilisant une approche orientée-objet puis affectées aux entités organisationnelles adéquates. Toutes les entités (organisationnelle ou concrète) et toutes les relations (affectation, hiérarchie, privilège, contrainte) associées à ces entités introduites en utilisant l'interface graphique de MotOrBAC se traduisent par des insertions dans la base de faits et la base de règles décrivant la politique de sécurité. Enfin, l'administration de la politique de sécurité, connue pour être un grand casse-tête, est également une des fonctionnalités offertes par le prototype. Au moyen d'une utilisation combinée des concepts de vue OrBAC et d'objet, l'attribution d'un privilège administratif consiste en l'affectation d'un *objet privilège* à la *vue administrative* adéquate. Le modèle implanté dans ce prototype étant auto-administré, la gestion de ces droits administratifs est alors faite à l'instar de la gestion de toute politique de sécurité OrBAC. Dans MotOrBAC, l'administration n'est pas forcément confiée à un seul sujet. Les privilèges des différents sujets affectés au rôle d'administrateur sont activés après identification/authentification au niveau de l'interface graphique.

La définition d'une politique de sécurité dédiée, telle qu'une politique de sécurité réseau et qui se traduit par la dérivation de règles génériques de configuration d'un firewall, a également été envisagée. Basée sur les travaux que nous avons menés dans [CCBSM04], une première version expérimentale a été implantée dans MotOrBAC.

La prise en compte de la délégation n'a pas été décrite dans cet article en raison du manque de place disponible. Elle est gérée dans notre prototype comme un privilège administratif et consiste à affecter un objet à une vue particulière de délégation, approche que nous avons adoptée pour la gestion de tout privilège administratif. L'objet en question (le ticket) contient les caractéristiques de la délégation, en particulier : (1) l'autorité qui délègue, (2) le bénéficiaire de cette délégation, (3) le privilège accordé par le biais de cette délégation et (4) son contexte d'application.

MotOrBAC gère ainsi différentes fonctionnalités permettant de définir une politique de sécurité cohérente. Plusieurs enrichissements du prototype sont prévus, en particulier, implanter une plus large variété de contextes [CM03] et compléter les travaux effectués dans le cadre d'une politique dédiée réseau au déploiement de la politique sur d'autres composants de sécurité.

References

- [ABB⁺03] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *4th IEEE International*

- Workshop on Policies for Distributed Systems and Networks (Policy'03)*, Como, Italie, June 2003.
- [CCB05] F. Cuppens and N. Cuppens-Boulahia. High level conflict management strategies in advanced access control models. Technical report, ENST-Bretagne, 2005.
- [CCBGA05] Frédéric Cuppens, Nora Cuppens-Boulahia, and J. Garcia-Alfaro. Detection and Removal of Firewall Misconfiguration. In *2005 IASTED International Conference on Communication, Network and Information Security (CNIS 2005)*, Phoenix, AZ, USA, November 2005.
- [CCBM04] Frédéric Cuppens, Nora Cuppens-Boulahia, and Alexandre Miège. Inheritance hierarchies in the Or-BAC Model and Application in a network environment. In *2nd Foundation of Computer Security Workshop*, Turku, Finlande, July 2004.
- [CCBSM04] Frédéric Cuppens, Nora Cuppens-Boulahia, Thierry Sans, and Alexandre Miège. A formal approach to specify and deploy a network security policy. In *Second Workshop on Formal Aspects in Security and Trust (FAST)*, Toulouse, France, 26-27 August 2004.
- [CM03] Frédéric Cuppens and Alexandre Miège. Modelling Contexts in the Or-BAC Model. In *19th Annual Computer Security Applications Conference (ACSAC '03)*, Las Vegas, USA, 2003.
- [CM04a] Frédéric Cuppens and Alexandre Miège. Administration Model for Or-BAC. *Computer Systems Science and Engineering (CSSE'04)*, 19(3), May 2004.
- [CM04b] Frédéric Cuppens and Alexandre Miège. Or-BAC (Organization Based Access Control). In *DistRibUtion de Données à grande Echelle (DRUIDE)*, Le Croisic, France, May 2004.
- [FSG⁺01] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security*, 4(3):224–274, August 2001.
- [Orb] OrBAC.org, <http://www.orbac.org/>.
- [SBM97] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, 1997.
- [SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *Computer*, 29(2):38–47, 1996.
- [SM99] Ravi Sandhu and Qamar Munawer. The ARBAC99 Model for Administration of Roles. In *15th Annual Computer Security Applications Conference (ACSAC '99)*. IEEE Computer Society, 1999.
- [Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II*. Computer Science Press, 1989.