THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Université Toulouse III - Paul Sabatier*
Discipline ou spécialité : *Informatique*

Présentée et soutenue par *Marwan CHEAITO*
Le *09/03/2012*

Titre : *Un cadre de spécification et de déploiment de politiques d'autorisation*

JURY

Président :	*Mme. Francine KRIEF, Professeur, ENSEIRB, Bordeaux*
Rapporteurs :	*M. Frédéric CUPPENS, Professeur, ENST-Bretagne, Rennes*
	M. Abdelmadjid BOUABDALLAH, Professeur, Université de Technologie Compiègne
Examinateurs :	*Mme. Maryline LAURENT, Professeur, TELECOM & Management SudParis, Paris*
	M. Abdelmalek BENZEKRI, Professeur à l'Université Toulouse III, Toulouse
	M. François BARRERE, Maître de conférences à l'Université Toulouse III, Toulouse
	M. Romain LABORDE, Maître de Conférences à l'Université Toulouse III, Toulouse

---

Ecole doctorale : *ED MITT : Mathématiques Informatique Télécommunications de Toulouse*
Unité de recherche : *IRIT-UMR 5505*
Directeur(s) de Thèse : *M. Abdelmalek BENZEKRI*
Co-Encadrants : *M. François Barrere*
M. Romain Laborde

Résumé

Notre travail propose une méthodologie de conception et de développement d'un système d'autorisation adaptable aux différentes facettes que peut recouvrir le contrôle d'accès dans les organisations telles que l'hétérogénéité des pratiques organisationnelles, des technologies utilisées et des contextes à considérer.

Pour y répondre, deux approches de gestion ont guidé nos études : le contrôle d'accès basé sur des attributs et la gestion à base de politiques.

- Le contrôle d'accès basé sur des attributs permet de spécifier des permissions par rapport à toute caractéristique liée de la sécurité des utilisateurs, des actions, des ressources et de l'environnement. Cette approche répond aux problèmes liés à l'expressivité des langages de politiques d'autorisation.
- La gestion à base de politiques, quant à elle, vise à permettre l'adaptabilité dynamique du comportement d'un système par le biais de politiques de gestion. Elle repose sur une architecture mettant en exergue deux entités : le Policy Decision Point (PDP) et le Policy Enforcement Point (PEP). Le PDP est une entité indépendante de l'élément géré, qui prend des décisions de gestion par rapport à une politique donnée. Le PEP fait l'interface entre le PDP et l'élément géré. Lorsqu'une requête est effectuée par un utilisateur sur une ressource, le PEP envoie une demande de décision au PDP et l'applique. Ce type d'architecture favorise l'intégration du système de gestion dans un environnement à gérer. Nous avons donc choisi le standard XACML comme technologie cible car il met en œuvre ces deux approches.

Si d'un point de vue théorique XACML semble répondre aux problèmes d'adaptabilité, les systèmes conçus selon ce standard aujourd'hui sont limités à une situation donnée. En effet, un système XACML développé pour des besoins et un environnement technologique donnés ne peut pas être facilement réutilisé dans un autre contexte. Il est donc nécessaire de définir une méthodologie de conception et de développement pour rendre un tel système d'autorisation adaptable dans la pratique. Notre approche consiste à définir un système d'autorisation minimal qui puisse être facilement étendu pour des besoins spécifiques et contextuels (e.g. capacité d'expression et de mise en œuvre de politiques incluant des contraintes complexes, adaptation à un environnement technologique particulier, etc). Ceci amène les questions suivantes :

- Quelles extensions doivent être apportées au système d'autorisation minimal ? Est-il possible de réutiliser des extensions pour différentes situations données ? Par exemple, le nombre de technologies utilisées pour stocker les accréditations des utilisateurs est limité (e.g. LDAP, MySQL, Active Directory). Cependant, les accréditations ainsi que la structuration de ces données d'accréditation peuvent différer d'une organisation à une autre.
- Comment gérer le cycle de vie des extensions dans le système d'autorisation ? En effet, il existe un lien fort entre l'utilisation d'une extension et les besoins exprimés dans une politique d'autorisation. Par exemple, l'existence dans le système d'autorisation d'une extension permettant de récupérer le rôle de l'utilisateur dans une base de données MySQL n'est nécessaire que si le système d'autorisation doit évaluer au moins une politique incluant des contraintes sur le rôle des utilisateurs. Dans le cas contraire, cette extension n'a pas lieu d'être présente dans le système d'autorisation minimal. De la même manière, si une politique incluant des contraintes sur les rôles des utilisateurs est chargée dans le système d'autorisation, l'extension associée doit être rajoutée.

En nous basant sur les différents travaux issus du domaine du génie logiciel liés au concept d'adaptabilité, nous avons défini les besoins d'adaptation des systèmes d'autorisation. Ce travail nous a permis de déterminer une classification des extensions par rapport aux besoins d'adaptabilité des systèmes d'autorisation de type XACML. Nous avons aussi proposé une méthode permettant de faciliter la création d'extensions et favorisant leur réutilisabilité.

Enfin, nous avons traité le problème de gestion du cycle de vie des extensions dans le système d'autorisation en proposant le concept de « politique auto-contenue ». Une politique auto-contenue comprend les règles d'autorisation ainsi que toute information nécessaire au système d'autorisation pour qu'il l'interprète et qu'il l'exécute (i.e., les extensions et leurs configurations). Nous avons décrit une architecture pour spécifier et déployer des politiques auto-contenues en utilisant l'approche composants orientés-services. Notre architecture est flexible et dynamiquement adaptable. Nous l'avons implémentée en utilisant le cadriciel OSGi afin de valider notre travail à la fois en termes de faisabilité mais aussi de performance. De plus, nous avons réalisé un éditeur de politique auto-contenue qui facilite l'écriture des règles d'autorisation ainsi que la configuration des extensions utilisées.

Remerciements

Ce travail de thèse n'aurait pas été aussi fructueux sans l'aide de plusieurs personnes. Je remercier tous ceux qui m'ont soutenu et encouragé pendant ces cinq années de thèse et qui ont, à leur manière, permis à ce travail d'aboutir.

Tout d'abord, je remercie, M. Abdelmalek BENZEKRI, Professeur à l'IUT-A de l'université Toulouse III, et directeur de l'équipe SIERA, pour m'avoir accueilli dans son équipe. Ainsi, J'exprime ma gratitude la plus profonde vers lui, pour les précieux conseils et l'expérience dont il a su me faire profiter.

Mes plus grands remerciements sont naturellement pour mes deux encadrants : messieurs François BARRERE et Romain LABORDE. Romain Laborde est une personne géniale, toujours présent pour me soutenir durant toutes ces années. Grâce à ses idées, sa disponibilité, ses remarques et nos nombreuses discussions, a joué un grand rôle dans l'élaboration de ce travail. François BARRERE est une personne pleine d'humeur. Ma considération pour lui est inestimable. Ses remarques, ses encouragements, et ses critiques pertinentes m'ont conduit vers la bonne voie dans l'élaboration de ce travail. Je l'avoue, la phase de rédaction, ce n'était pas toujours facile pour eux ! François et Romain, un GRAND Merci pour tout !!

Je tiens à remercier chaleureusement M. Frédéric CUPPENS et M. Abdelmadjid BOUABDALLAH pour avoir accepter de rapporter ma thèse et pour leurs remarques pertinentes sur le manuscrit. Je remercie également tous les autres membres du Jury : Maryline LAURENT et Francine KRIEF, pour l'honneur qu'ils m'ont fait en acceptant de participer à mon jury de thèse malgré des emplois de temps chargés.

Un merci particulier à tout le reste des permanents de l'équipe SIERA, car l'expérience de chacun d'eux, m'appris des choses même en dehors de mon domaine: Thierry DESPRATS, Michelle SIBILLA, Emmanuel LAVINAL, Julien BROISIN, André AOUN, Philippe VIDAL, Daniel MARQUIÉ, Michel GALINDO, Martine de PERETTI, et Olivier CATTEAU.

Je tiens à remercier mes collègues présents et passés (déjà docteurs ou actuels thésards), avec qui j'ai partagé de discussions et échangés des idées : Michel KAMEL, Samer WAZAN, Tony TOUEIR, Bashar KABBANI, Ludi AKUE, Valentine BUTOIANU, Hicham EL KHOURY, Triomphe RAMANDALAHY, et Audrey MOUI. Je pense particulièrement à Samer, Tony, et Bashar pour leur soutien ces derniers jours. Je tiens à remercier aussi tous les stagiaires passés (en particulier Lucas et Mathieu) qui ont m'aider dans la partie implémentation de mon travail de thèse.

Mes remerciements s'adressent également à l'ensemble des services de l'IRIT, qui permettent à chacun de travailler dans les meilleures conditions. Je pense particulièrement à Martine LABRUYERE, Agathe BARITAUD, et Arnaude CARIOU. Ainsi, merci à Jacqueline ARNILLAS, pour ces discussions que changer d'air et laisse nous oublier le stress!

J'adresse également une pensée à tous mes amis à Toulouse (en particulier Raphael, Mona, Hanna, Oussama, et Wael) pour leur soutien, les franches rigolades et les soirées interminables. Un grand merci à eux. Il ne faut pas oublier mes amis au Liban qui m'ont soutenu malgré la distance qui nous sépare, et je pense toujours à mes amis que j'ai perdus sans pouvoir les voir. Un merci particulier à mon ex Maria, qui j'ai partagé avec elle des moments agréables. Enfin, j'aimerais remercier ma famille ainsi que tous les proches à qui je dois beaucoup. Ma mère, je ne vous oublierai jamais, tu resteras toujours dans mon cœur. ***A tous Mille Merci !!***

Liste de Publications

- ***Conférences et workshops internationaux :***
  - *Marwan Cheaito, Romain Laborde, François Barrère, Abdelmalek Benzekri. A deployment framework for self-contained policies. Dans: IEEE/IFIP 6th International Conference on Network and Service Management (CNSM 2010), Niagara Falls -CANADA, 25/10/2010-29/10/2010.*
  - *Romain Laborde, Marwan Cheaito, François Barrère, Abdelmalek Benzekri. A deployment framework for self-contained policies (short paper). Dans: IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY 2010), George Mason University -USA, 21/07/2010-23/07/2010.*
  - *Marwan Cheaito, Romain Laborde, François Barrère, Abdelmalek Benzekri. An extensible XACML authorization decision engine for context aware applications (regular paper). Dans : International Conference on Pervasive Computing and Applications (ICPCA 2009), Tamkang University -Taiwan, 03/12/2009-05/12/2009, IEEE, (support électronique), décembre 2009.*
  - *Romain Laborde, Marwan Cheaito, François Barrère, Abdelmalek Benzekri. An extensible XACML authorization web service: Application to dynamic web sites access control. Dans: 3rd workshop Security and Privacy in Telecommunications and Information System (SEPTIS 2009), Marrakech, Morocco, November 29th - December 4th, 2009.*
- ***Conférences et workshops nationaux :***
  - *Romain Laborde, Marwan Cheaito, François Barrère, Abdelmalek Benzekri. Vers un système d'autorisation comme un service. Dans : 9th edition of the Francophone Conference on management of Networks and Services (GRES 2010), Montréal-CANADA, 12/10/2010-14/10/2010.*
  - *Marwan Cheaito, Romain Laborde, François Barrère, Abdelmalek Benzekri. Configurable Data Types in Policy Based Access Control Management: A Specification and Enforcement Framework. Dans : Conférence sur la Sécurité des Architectures Réseaux et Systèmes d'Information (SAR-SSI 2010), Menton -France, 18/05/2010-21/05/2010.*

Table des matières

Résumé I
Remerciements III
Liste de Publications V
Table des matières VII
Table des Figures XI
Introduction Générale 1
Chapitre1. Evolution des exigences de sécurité des organisations 7
1.1 Introduction 7
1.2 Gestion des identités et des accès 8
1.2.1 Concept d'identité 9
1.2.2 Concept de gestion des accès 10
1.2.3 Changements de perspectives dans la gestion des identités et des accès 12
1.3 Autorisation 20
1.3.1 Les architectures d'autorisation AAA 20
1.3.2 Les architectures d'autorisation à base de politiques 23
1.4 Conclusion 24
Chapitre2. Evolution des modèles formels de contrôle d'accès 25
2.1 Introduction 25
2.2 Modèles de contrôle d'accès classiques 26
2.2.1 Modèle de contrôle d'accès discrétionnaire 26
2.2.2 Modèle de contrôle d'accès obligatoire 28
2.2.3 Synthèse 28
2.3 Modèles de contrôle d'accès à base de rôle 29
2.4 Modèles de contrôle d'accès à base des tâches 30
2.5 Modèles de contrôle d'accès dérivés de RBAC 31
2.5.1 Modèles de contrôle d'accès à base d'équipes 31
2.5.2 Modèles de contrôle d'accès prenant en compte la localisation 32
2.5.3 Modèles de contrôle d'accès à base de contexte temporel 33
2.5.4 Modèles de contrôle d'accès à base de contexte 33
2.5.5 Synthèse des extensions du modèle RBAC 34
2.6 Modèle de contrôle d'accès à base d'organisation 35
2.6.1 Synthèse 38
2.7 Conclusion 38
Chapitre3. Adaptabilité et systèmes d'autorisation 41
3.1 Introduction 41
3.2 Adaptabilité 42
3.2.1 Quelques définitions informelles sur l'adaptabilité 42
3.2.2 Les techniques pour mettre en œuvre l'adaptabilité 44

3.3 Etude de l'adaptabilité d'un système d'autorisation 48
3.4 Conclusion 51
Chapitre4. Etat de l'art sur les systèmes d'autorisation 53
4.1 Introduction 53
4.2 Les approches intégrant plusieurs PDPs 53
4.2.1 RSBAC 53
4.2.2 L'infrastructure d'autorisation avancée de Chadwick et Fatema 55
4.2.3 Synthèse 56
4.3 L'approche d'un langage de politique générique 56
4.3.1 MotOrBAC 56
4.3.2 Les systèmes d'autorisation à base d'ontologies 58
4.3.3 Ponder 63
4.4 XACML 64
4.4.1 Introduction 64
4.4.2 Les extensions du langage de politiques XACML 67
4.4.1 Etude des implémentations XACML 69
4.4.2 Etude des éditeurs de politiques XACML 71
4.5 Conclusion 73
Chapitre5. Analyse des besoins de XACML 75
5.1 Introduction 75
5.2 Pourquoi avoir choisi XACML 76
5.2.1 Langage de politiques XACML extensible 76
5.2.2 Architecture flexible 78
5.3 Analyse de XACML dans un environnement réel 79
5.3.1 XACML en pratique 79
5.4 Définition des besoins pour un moteur de décision XACML extensible 81
5.5 Conclusion 84
Chapitre6. Une méthodologie pour créer des extensions réutilisables 85
6.1 Introduction 85
6.2 Les types de données réutilisables 85
6.2.1 Exemples 86
6.2.2 Types de données prédéfinis 90
6.2.3 Type de données configurable scalaire 91
6.2.4 Type de données configurable composite 95
6.3 Les PIPs réutilisables 95
6.3.1 Exemple 96
6.3.2 Méthodologie 97
6.3.3 Exemple de mise en œuvre de la méthodologie 97
6.4 Conclusion 99

Chapitre7. Implémentation d'un cadriciel pour l'édition et l'exécution de politiques d'autorisation XACML 101
7.1 Politique auto-contenue ... 101
7.2 Les exigences d'une politique auto-contenue ... 103
7.3 Outillage pour la construction de politique auto-contenue ... 103
7.4 Proposition d'architecture d'un moteur de décisions d'autorisation XACML extensible ... 104
7.4.1 Architecture d'édition d'une politique auto-contenue ... 105
7.4.2 Architecture d'exécution d'une politique auto-contenue ... 107
7.5 Scenario d'une politique auto-contenue ... 109
7.5.1 Application du scenario ... 110
7.6 Implémentation ... 118
7.6.1 Description ... 119
7.6.2 Résultats expérimentaux ... 120
7.6.3 Schéma global ... 122
7.7 Conclusion ... 123
Conclusion générale ... 125
Acronymes ... 129
Références ... 131

Table des Figures

Figure 1 La relation entre les entités, les identités, et les identifiants/attributs 10
Figure 2 Architecture d'autorisation AAA 20
Figure 3 le modèle Agent 21
Figure 4 Le modèle Push 21
Figure 5 Le modèle Pull 22
Figure 6 Architecture d'autorisation centralisée 22
Figure 7 Architecture de services distribués 22
Figure 8 Système d'autorisation 24
Figure 9 Modèle RBAC 29
Figure 10 Le modèle LRBAC 32
Figure 11 Le modèle Or-BAC 36
Figure 12 Contexte dans Or-BAC 38
Figure 13 Contraintes d'une solution informatique adaptée 41
Figure 14 Symboles de la définition formelle de l'adaptation 43
Figure 15 Exemple d'aspect en AspectJ 46
Figure 16. Architecture orientée service 47
Figure 17. Le contexte de système d'autorisation 49
Figure 18Architecture RSBAC 54
Figure 19 Architecture pour l'infrastructure d'autorisation avancée 55
Figure 20 Architecture de MotOrBAC 57
Figure 21 Architecture de KAOS 59
Figure 22 Architecture d'une infrastructure d'autorisation REI 60
Figure 23 Architecture de POLIPO 62
Figure 24 Exemple d'une politique d'autorisation en Ponder 63
Figure 25 Exemple d'une politique XACML 65
Figure 26 Architecture de XACML 66
Figure 27 La hiérarchie des rôles avec le profil RBAC 68
Figure 28 Exemple d'une règle GeoXACML 69
Figure 29 Une capture d'écran de l'éditeur XML PAD 71
Figure 30 Une capture d'écran de l'éditeur proposé par UMU 72
Figure 31 Exemple d'une condition 72
Figure 32 Exemple d'une politique XACML 77
Figure 33 Architecture PBM 79
Figure 34 Exemple d'utilisation de XACML 80
Figure 35 Notre vision d'un moteur de décisions d'autorisation XACML extensible 82
Figure 36 Hiérarchie des rôles 87
Figure 37 Hiérarchie de localisations logiques 88
Figure 38 Le groupe scenario 88
Figure 39 Méthodologie pour étendre le langage avec types de données configurables scalaires 91
Figure 40 Méthodologie d'un type de données « Rôle » configurable scalaire 92
Figure 41 Ontologie pour type de donnée configurable scalaire 94
Figure 42 POSET définition 94
Figure 43 Contraintes de relation d'ordre avec PAL 94
Figure 44 Processus de configuration d'un PIP 96
Figure 45 Scénario PIP MySQL 97
Figure 46 Concept d'une politique auto-contenue 102
Figure 47 Architecture générale du système d'autorisation proposé 104
Figure 48 Architecture de la spécification d'une politique auto-contenue 106
Figure 49 Architecture de déploiement d'une politique auto-contenue 108
Figure 50 Hiérarchie de rôles dans UPS 109
Figure 51 Exemple d'une table MySQL 110
Figure 52 Politique auto-contenue du scenario 110

Figure 53 Architecture d'édition de la politique auto-contenue du scenario 111
Figure 54 Projet d'une politique auto-contenue 112
Figure 55 Configuration d'un type de données configurable scalaire 112
Figure 56 Ajouter un fichier de configuration de type de données 112
Figure 57 Première fenêtre pour configurer un PIP MySQL 113
Figure 58 Les informations à spécifier pour un PIP MySQL 114
Figure 59 Tester l'extraction d'informations de la base de données 114
Figure 60 Structure des valeurs provenant dans une base de données MySQL 115
Figure 61 Format attendue par type de données 115
Figure 62 Fenêtre principale pour la configuration des attributs PIP 115
Figure 63 Liste des valeurs possibles pour un type de données configurable scalaire 116
Figure 64 Liste de noms possibles d'URN XACML 117
Figure 65 Architecture d'exécution de la politique auto-contenue du scénario 118
Figure 66 La décomposition de notre politique 118
Figure 67 Temps nécessaire en µ-secondes pour chaque test 121
Figure 68 Schéma global 122

Introduction Générale

Depuis quelques années, les technologies de l'information et de la communication révolutionnent tous les secteurs, qu'ils soient industriels, commerciaux, administratifs et même jusqu'à la vie privée des individus. Les progrès et les innovations alliés à l'explosion de la bulle « Internet » nous permettent aujourd'hui d'exploiter des dispositifs de plus en plus adaptés à la mobilité et connectés à des réseaux polymorphes (sans fil, GPRS, UMTS, satellites, filaires, optiques…). Ils ont ouvert la voie à de nouvelles applications et de nouveaux usages qui sont en passe de révolutionner des techniques de travail largement établies. Des bénéfices particuliers sont par exemple attendus des champs d'application de l'informatique ubiquitaire, du « cloud computing », de la construction de réseaux sociaux d'entreprise ou grand public. Les réseaux et leurs applications deviennent le théâtre privilégié des interactions homme machine, machine à machine, ou entre individus que ce soit dans un cadre professionnel d'entreprises, dans les rouages des états ou encore dans l'organisation de la vie privée des individus.

Cependant, ces changements de paradigme dans la gestion et l'organisation des entreprises ont amené à des questionnements en matière de sécurisation des systèmes d'information en regard des risques encourus. Nombre de travaux portant sur la conception d'équipements, d'architectures et/ou de normalisation ont été conduits ces dernières années, afin d'accompagner ces mutations organisationnelles. Aujourd'hui, la prise de conscience en matière de sécurité devrait également conduire toute entreprise à décrire les axes à suivre dans un document de contrôle général appelé politique de sécurité, recueillant la nature des objectifs de sécurité à garantir. La constitution de ce recueil est une phase importante puisqu'il fait partie d'une action stratégique visant à protéger tout ou partie du « système d'information », considéré de plus en plus comme un « actif » au même titre que d'autres biens mobiliers ou immobiliers. Cependant le passage de ce recueil à une politique opérationnelle n'est pas toujours immédiat, d'autant plus que pour nombre d'entreprises ou organisations la gestion de la sécurité n'est pas un objectif métier, mais une activité à laquelle il faut répondre sans avoir forcément une formation de spécialiste dans ce domaine.

L'ouverture des systèmes d'information pour répondre à des contraintes de disponibilité, de traitement et de puissance de stockage des systèmes d'information est inéluctable. La contrepartie de cette ouverture vient de l'éventualité des menaces ou risques de sécurité qu'elle engendre. Aujourd'hui, la vie des organisations dépend plus que jamais du cycle de vie des informations dont elles sont propriétaires et/ou qu'elles gèrent. Les conséquences d'une violation de la sécurité risquent de provoquer des situations graves et d'impacter le fonctionnement des organisations La presse spécialisée se fait régulièrement l'écho de situations où certaines données d'une organisation ont été mal modifiées. Parfois l'interruption d'un service est la conséquence d'une violation de sécurité entrainant une indisponibilité pour les utilisateurs légitimes. D'autres cas signalent que des données confidentielles ont été divulguées dans le domaine public [Vimercati et al. 2003], comme suite à une modification d'une règle au sein d'un pare-feu, routeur. Des situations décrites font également suite à un accès illégal aux données personnelles : ainsi récemment, le géant japonais de l'électronique Sony a été victime d'un accès illégal aux données personnelles des abonnés et engendré une perte 122 millions d'euros au cours de ce gigantesque piratage [Le Monde 2011].

Afin d'apporter des solutions à ces problèmes récurrents, les méthodes et les référentiels de normes proposent d'aborder la sécurité dans son ensemble. En les appliquant, il est alors possible de dégager les exigences de sécurité et d'identifier les situations dans lesquelles un système au sein de l'organisation peut être exposé à des attaques ou risques [Oliveira 2008]. Dans le domaine de la sécurité, la confidentialité, l'intégrité, la disponibilité, et la traçabilité constituent les propriétés ou objectifs de base à atteindre. Ils représentent les points les plus importants qui guident le processus d'expression des « objectifs de sécurité ». D'autres éléments peuvent venir les compléter comme l'authenticité (la propriété d'être vrai) ou la non-répudiation (l'impossibilité de nier être l'auteur d'un accès ou d'une opération).

Les documents de politique de sécurité décrivent les règles, les lois ou les pratiques qui régissent le traitement des informations sensibles et l'utilisation des ressources par le matériel et le logiciel d'un système d'information ou d'un produit [Nasser 2006]. Une politique de sécurité reste cependant difficile à définir, à exprimer et à construire, car elle est établie par des acteurs ayant des niveaux de connaissance différents sur l'organisation, sur la stratégie des organisations, sur les environnements technologiques. De plus la croissance et l'évolution permanente des organisations, de l'économie numérique mondiale, de l'étendue des applications d'Internet et des technologies de l'information et de la communication rendent encore plus complexe cette tâche. Chaque nouvelle vague de technologie apporte aussi avec elle de nouveaux risques de sécurité et de nouveaux défis de gestion de la sécurité. Cela a pour conséquence la nécessité de réviser et/ou de changer certains documents de politique de sécurité dont le contenu a été arrêté à un instant donné [Tech-Brief-CA 2011].

Aujourd'hui, l'une des retombées en matière de sécurisation de la constitution des politiques de sécurité conduit à la mise en œuvre de systèmes de gestion des identités et des accès (IAM). En effet, toute politique de sécurité nécessite au minimum qu'un contrôle sur les usagers et les ressources soit mis en œuvre. Ces solutions ne sont qu'un maillon clé de la chaine de sécurité des organisations. Elles occupent aujourd'hui un espace important dans le cadre de l'ouverture des systèmes d'information. En effet, le contexte de la mondialisation des marchés impose que les organisations soient réactives et ouvrent tout ou partie de leur systèmes d'informations à des partenaires potentiels, tout en garantissant des niveaux pointus de protections de leurs ressources. Les solutions d'IAM évoluent en parallèle des avancées technologiques qui marquent différents domaines (ubiquitaire, mobilité, virtualisation… etc.), et tentent de couvrir de nouveaux défis de sécurité. En particulier, la gestion des identités ne se limite plus à gérer le cycle de vie des identités des employés internes à une organisation unique, elle doit offrir des moyens de gérer des acteurs externes tels que des clients, des partenaires, des sous-traitants… Le contrôle d'accès est une solution de gestion dont l'objectif principal est de garantir l'accès à des ressources (matérielles, logicielles) présentant des degrés divers de sensibilité. Le contrôle d'accès devrait garantir l'accès aux seules entités sensées être habilitées à exercer des actions sur ces ressources. Dans le concept de gestion des accès, le contrôle d'accès est généralement réalisé par la séquence de processus AAA (Authentification, Autorisation, Compatibilité). Pour que le contrôle d'accès revête un caractère opérationnel, des solutions intégrant un système d'autorisation, un système d'authentification et un système d'auditabilité doivent y être implantés. Dans ce domaine, notre contribution majeure est de proposer un système d'autorisation adaptable.

Généralement, le processus d'autorisation est une fonction qui consiste à déterminer et à décider si une entité est autorisée ou non à réaliser une ou plusieurs activités ou actions sur une ressource ou un service. Pour régir l'accès aux ressources, une politique d'autorisation doit être mise en place au sein d'un système d'autorisation. Elle permet d'exprimer de manière plus ou moins abstraite les besoins de sécurité pour les organisations. Les politiques d'autorisation sont une manière particulière de mettre en œuvre les exigences de sécurité décrites dans des documents de politiques de sécurité [Oliveira 2008]. Elles sont définies comme étant des directives (règles) de haut niveau qui spécifient quelle entité a la permission d'exercer quelle action sur quelle ressource d'un système.

Aujourd'hui, plusieurs modèles formels de contrôle d'accès considérant le triplet (sujet, objet, action) existent pour aider à écrire et vérifier la cohérence (par exemple l'absence de conflits entre politiques), et aussi exprimer les exigences de sécurité dans les politiques d'autorisation. A l'origine, la mise en œuvre d'une politique d'autorisation s'adressait à des ressources parfaitement identifiées (i.e. des fichiers ou répertoires d'unités disques). L'évolution des architectures de systèmes vers la construction de véritables systèmes d'information, a donné de nouveaux élans à la modélisation des systèmes de gestion des accès. A ce jour, des critères aussi divers que les rôles, la localisation, les tâches, l'organisation, les contextes etc. donnent lieu à la formalisation de nouveaux modèles de contrôle d'accès, pour répondre à des exigences que l'on peut décrire au sein de politiques d'autorisation. Cette évolution des modèles formels de contrôle d'accès vient de la volonté de pouvoir intégrer dans les politiques des contraintes ou des approches applicatives rendues possibles par la

technologie (réseaux sans fil, informatique mobile, ubiquitaire, communautaire, service web etc.). Ces modèles proposent une organisation des droits d'accès au plus près des « politiques d'autorisation », c'est-à-dire l'ensemble des règlements qui régissent la façon dont sont protégées les ressources d'un système.

Sur un plan de mise en œuvre, les mutations des exigences des organisations et des modèles formels de contrôle d'accès ont pour conséquence de devoir adapter en permanence le fonctionnement des systèmes d'autorisation. Il existe plusieurs propositions et implémentations de systèmes d'autorisation. Elles fournissent un langage de politiques pour la spécification de règles d'autorisation et un moteur de décisions d'autorisation pour répondre à une politique donnée et l'appliquer. Beaucoup de ces propositions sont issues de situations spécifiques qui ont motivé leur développement sans que pour autant l'adaptabilité ait été traitée. Il en est ainsi de SELinux [SELinux 2011] qui implémente le modèle MAC pour le système d'exploitation Linux ou encore de PERMIS [Permis 2011] qui implémente le modèle RBAC etc. Au contraire, d'autres systèmes d'autorisation existent et ont pris en compte cette problématique de l'adaptation en fournissant soit un langage ou un modèle de politiques générique (comme XACML [Xacml 2011], PONDER [Ponder 2011], MotOr-BAC [MotOr-BAC 2011] etc.) ou bien en permettant l'intégration de multiples moteurs de décision de politiques (Policy Decision Point) (comme RSBAC [RSBAC 2011], et l'infrastructure d'autorisation proposée par Chadwick et Fatema [Chadwick et Fatema 2009] etc...).

Comme nous l'avons annoncé, notre travail porte sur la définition d'une méthode de conception et de développement d'un système d'autorisation adaptable. Il présente la faculté d'être facilement étendu à toute exigence de contrôle d'accès actuelle et suffisamment ouvert pour en garantir la pérennité pour des exigences futures. Deux approches de gestion ont guidé notre étude : le contrôle d'accès basé sur des attributs (ABAC) et la gestion à base de politiques (PBM) :

- ABAC permet de spécifier des permissions par rapport à toute caractéristique liée à la sécurité des utilisateurs, des actions, des ressources et de l'environnement. Cette approche nous semble appropriée pour l'expressivité de la spécification des politiques d'autorisation. Elle donne les moyens d'écrire des politiques, intégrant une diversité d'informations de sécurité et associées à des exigences propres aux organisations. Elle est considérée comme une solution de gestion de politiques d'autorisation de manière générique.

- Par ailleurs, le choix de l'approche PBM est justifié par les motivations premières d'un tel système. Il garantit la flexibilité et l'adaptation à l'infrastructure existante. De plus la gestion du changement semble adaptée à ce service d'autorisation. Dans l'architecture classique d'un système de gestion à base de politique, on distingue deux entités principales : le Policy Decision Point (PDP) qui interprète la politique et prend des décisions de gestion par rapport à elle, et le Policy Enforcement Point (PEP) qui rend exécutables les décisions du PDP par le système contrôlé. L'objectif premier de cette approche est de rendre dynamique un certain nombre de tâches de gestion sans stopper ou recoder le système. Cette approche permet d'obtenir une architecture flexible pour le système d'autorisation favorisant l'intégration du système de gestion dans un environnement à gérer.

Dans notre travail, nous avons eu une attention particulière pour le standard OASIS XACML car il intègre nativement la notion d'extensibilité et qu'il possède un nombre important d'implémentations. D'un point de vue théorique XACML semble répondre aux problèmes d'adaptabilité, mais dans la pratique les systèmes basés sur ce standard sont limités à une situation donnée.

XACML donne la possibilité d'étendre le langage en ajoutant de nouveaux éléments (attributs, types des données, fonctions...) si besoin, ce qui lui confère la caractéristique de langage extensible. Cette caractéristique d'extensibilité est nécessaire pour l'expressivité et la flexibilité de la spécification des politiques. A tout nouveau besoin caractérisé, on ajoute les extensions du langage dans les politiques. Parmi les exigences imposées pour étendre le langage XACML, il faut chercher à identifier

les besoins principaux pouvant conduire à la construction d'une extension. De plus il faut essayer de déterminer comment architecturer ces extensions afin qu'elles soient réutilisables dans des situations différentes. Par exemple, la construction d'un système de gestion d'accès aux ressources pourrait donner lieu à une extension, sachant que l'accès aux ressources dépend des accréditations données aux utilisateurs, sachant également que plusieurs technologies (e.g., LDAP, MySQL, Active Directory) existent pour supporter le stockage d'informations diverses et/ou des accréditations et qui plus est, que leur structuration peut différer d'une organisation à une autre.

En s'appuyant sur les différents travaux liés au concept d'adaptabilité des systèmes informatiques dans le domaine du génie logiciel, nous avons étudié les besoins d'adaptabilité des systèmes d'autorisation. Ce travail nous a permis de déterminer une classification des extensions par rapport aux besoins d'adaptabilité des systèmes d'autorisation de type XACML. Nous avons aussi proposé une méthode permettant de faciliter la création des extensions et favorisant leur réutilisabilité.

D'autre part, le moteur de décision d'autorisation XACML doit être modifié pour prendre en compte des politiques intégrant des extensions du langage standard durant l'évaluation. Cela est fait en important les modules des extensions du langage (par exemple importer le module qui implémente le type de données permettant de manipuler des informations GPS) au moteur de décision. Cette solution pose un problème d'ingénierie de développement car il ne s'agit pas d'une simple mise à jour logicielle du moteur de décision. Le problème est de nature conceptuelle. Les extensions du langage standard peuvent être spécifiques à une politique. Dans ce contexte, la modification apportée au moteur de décision n'a de véritable intérêt que lorsqu'il doit évaluer une politique intégrant les extensions au langage standard. Dès lors comment faire le lien entre les politiques et leurs extensions ? Conceptuellement et à des fins de contrôle et de maîtrise du système, il y a un intérêt manifeste à coupler le cycle de vie de l'implémentation du moteur de décision avec le cycle de vie des politiques.

Une réponse doit être donnée à la problématique d'extension du langage de politique et de la mise en œuvre par un moteur de décision. Nous avons introduit le concept de **«politique auto-contenue»**, qui représente à nos yeux une solution d'avenir pour le déploiement des politiques. Une politique auto-contenue comprend les règles d'autorisation ainsi que toute information nécessaire au système d'autorisation pour qu'il l'interprète et qu'il l'exécute (i.e., les extensions et leurs configurations). Nous considérons dans cette approche que : l'implémentation native du moteur de décision inclut uniquement le code requis pour que le processus de prise de décision soit le plus performant possible. C'est au sein d'une politique auto-contenue que l'on trouve le code correspondant aux extensions du langage ainsi que les configurations qui conviennent.

Il faut noter que la gestion du cycle de vie et de l'implémentation du moteur de décision s'en trouve facilitée. En effet, lorsqu'une politique est chargée par le moteur d'autorisation, seul le code nécessaire est installé ; lorsque la politique est déchargée, le code associé est retiré. Cela évite donc de maintenir du code « inutile » (car associé à d'autres politiques et non utilisé) au sein du moteur de décision. Une architecture supportant les politiques auto-contenues doit répondre à ces exigences. Pour atteindre cet objectif, nous proposons une architecture basée sur l'approche service orienté composant. Cette approche garantit la flexibilité, et l'adaptabilité dynamique. De plus, elle offre la réutilisabilité de composants et la gestion de la dépendance de nos modèles orientés composants. Une implémentation a été développée pour établir la preuve du concept, en particulier, valider les performances de notre approche. De plus, nous avons réalisé un éditeur de politique auto-contenue qui facilite l'écriture des règles d'autorisation ainsi que la configuration des extensions utilisées.

Bien que nous ayons choisi l'environnement XACML pour la réalisation et l'implémentation de nos travaux, nous tenons à souligner qu'elle n'est pas spécifique à un langage particulier. Notre approche est suffisamment générique pour être applicable à tout autre système et langage à base d'attributs où les expressions des règles sont préfixées.

Ce document de thèse s'articule autour de sept chapitres. Chaque chapitre comporte une introduction, nos réflexions, et une conclusion synthétisant le chapitre.

Dans le premier chapitre nous rappelons les changements majeurs qui ont marqué l'organisation des entreprises au fil du temps. En effet, l'adoption des technologies de l'information et de la communication conduit au fur et à mesure à trouver des solutions en matière de sécurité en regard des risques encourus. Pour cela, nous présentons dans ce chapitre un état de l'art sur les évolutions qui ont marqué les systèmes de gestion des identités et des accès. Nous présentons également les architectures d'autorisation et nous donnons notre point de vue sur la conception architecturale de ces systèmes d'autorisation.

Dans le deuxième chapitre, nous présentons une synthèse des principaux modèles formels de contrôle d'accès proposés et de leur capacité à répondre aux évolutions technologiques.

Dans le troisième chapitre, nous présentons les différents travaux liés au concept d'adaptabilité des systèmes informatiques dans le domaine du génie logiciel. En se basant sur cette étude, nous analysons les besoins d'adaptabilité d'un système d'autorisation.

Dans le quatrième chapitre, nous donnons un état de l'art sur les principaux systèmes d'autorisation qui ont abordé la notion d'adaptabilité. Nous analysons également la capacité d'adaptation de ces systèmes d'autorisation par rapport à notre étude effectuée dans le troisième chapitre. Ainsi, nous analysons plus en détails le standard OASIS XACML qui nativement intègre la notion d'extensibilité et qui possède un nombre important d'implémentations.

Le cinquième chapitre est consacré à l'intérêt que présente la construction d'un langage de politiques extensible. L'extensibilité est offerte par un langage générique suffisamment flexible et expressif pour construire des politiques en prenant en compte n'importe laquelle des informations de sécurité. Puis, nous nous attachons à l'intérêt que suscite la mise en œuvre d'une architecture flexible. La flexibilité permet l'utilisation d'un langage de politiques unique pour contrôler les différents applications/systèmes ou ressources dans un système. Compte tenu des travaux menés depuis plusieurs années et de son adoption, nous avons choisi de nous appuyer sur l'environnement logiciel XACML pour étayer notre architecture. Nous rappelons comment les prises de décisions d'autorisation sont réalisées dans un environnement réel. Ensuite, nous citons les besoins d'extensibilité d'un moteur de décisions d'autorisation XACML.

Dans le sixième chapitre, nous présentons une méthode qui permet de déterminer une structuration des extensions par rapport à l'analyse de besoins d'extensibilité du moteur de décisions d'autorisation XACML présentés dans le chapitre précédent. Cette méthode facilite la création des extensions et favorise leur réutilisabilité dans un moteur de décisions d'autorisation XACML.

Finalement, nous présentons dans le septième chapitre l'implémentation que nous avons faite du moteur de décisions d'autorisation XACML. L'architecture de notre implémentation est basée sur l'approche de composants orientés services. Cette approche garantit les critères d'extensibilité et d'adaptabilité dynamique de notre système dont nous avons soulignés l'importance. De plus, notre approche offre la réutilisabilité du code afin de limiter la quantité de code au sein des systèmes aux seules extensions nécessaires au langage de politiques XACML, et éviter la redondance de codes inutiles dans le système.

Chapitre1. Evolution des exigences de sécurité des organisations

1.1 Introduction

La fin du XX^{e} siècle a été marquée par la révolution des technologies de l'information et des communications (TIC). L'informatique est passée, en l'espace de quelques décennies, du statut de curiosité scientifique et de prototype militaire à celui d'outil incontournable et massivement répandu dans tous les secteurs d'activités de la vie professionnelle ou de la vie privée. Les particuliers tout comme les organisations sont et seront amenés à s'équiper de matériels électroniques toujours plus puissants et dont l'architecture revêt une complexité plus ou moins grande afin d'acquérir, traiter, transmettre, stocker des informations… [Thion 2008].

Par ailleurs, une organisation est un ensemble d'individus, regroupés au sein d'une structure régulée, dans le but de répondre à ou d'atteindre un ou des objectifs déterminés [ISO 9000 2000]. Dans toute organisation apparaissent des responsabilités, des pouvoirs, des échanges, des relations. De tout temps les organisations se sont structurées sous des formes différentes afin de répondre aux objectifs pour lesquels elles sont créées. Ainsi l'organisation d'une entreprise diffère de celle d'une université ou d'une association humanitaire.

Cependant, l'introduction des technologies impacte les modes de fonctionnement et/ou les méthodes de travail mis en œuvre au sein des organisations. L'introduction des nouvelles technologies a bouleversé et continue de remettre en cause les relations traditionnelles qui étaient celles de l'organisation des tâches et des relations entre partenaires. Sans chercher à établir une genèse complète dès l'origine des civilisations, des organisations se sont mises en place pour gérer les relations et la vie de communautés, de collectivités, pour étendre des zones d'influence, pour commercer etc. Aujourd'hui, les organisations se caractérisent comme des systèmes complexes où des équilibres se forment à partir de différents facteurs (des objectifs à atteindre, la nature des tâches à exercer, la nécessité d'équilibrer des budgets financiers, le capital de connaissances à gérer, la gestion des compétences des acteurs de l'organisation....). Pendant de nombreuses années, les interactions au sein des organisations ont été menées entre des êtres humains. Maintenant, la révolution technologique a conduit à un changement tel qu'en complément des interactions entre humains (man to man), les interactions homme à machine (man to machine), et machine à machine (machine to machine) gagnent de plus en plus de terrain. Si l'introduction de la mécanisation a simplifié en son temps la réalisation des activités par un être humain, l'ère industrielle a ensuite apporté une rationalisation dans la fabrication d'un certain nombre de produits. Aujourd'hui, le recul sur l'introduction des technologies n'est pas encore suffisant pour en décliner sous une forme exhaustive la totalité des impacts. Nous pouvons noter que son appropriation par les acteurs humains peut parfois être rapide, mais que sa maitrise est plus difficile à obtenir. Par exemple, la technologie des réseaux, l'informatique distribuée et l'Internet favorisent l'exploitation d'applications, l'accès à de nombreuses informations par des profils d'utilisateurs différents, et la mise en œuvre du Web2.0 a étendu aux internautes la faculté de devenir eux-mêmes des producteurs d'information et non plus de n'être que de simples consommateurs.

Aujourd'hui, si la technologie continue d'ouvrir la porte à de nouvelles formes d'organisations, elle donne lieu à de nombreuses réflexions dans le domaine de la sécurité. En effet, en repoussant au fur et à mesure les frontières de ce qui semblait difficile à mettre en œuvre en termes de dessin des contours des organisations, les exigences de sécurité continuent de changer. Les principes qui étaient ceux appliqués aux balbutiements de l'Internet ne sont plus les mêmes, et aujourd'hui nous pouvons constater des changements très forts voire parfois un revirement total sur les principes guidant la mise en œuvre de politiques sécuritaires. Sans pour autant en constituer une liste exhaustive, les éléments suivants soulignent certains de ces changements :

- D'une situation où la conception des architectures était guidé par le principe de restreindre la connaissance des ressources et par voie de conséquence d'en limiter l'accès aux seules entités (personnes) habilitées, nous sommes aujourd'hui à l'ère de l'application du principe de partage. En conséquence, collaboration et partage d'information deviennent des axes majeurs de l'ouverture des systèmes d'information. Cependant ces deux qualificatifs sont très loin d'être synonymes d'accès permissifs. Ils imposent bien au contraire la mise en œuvre de mécanismes de contrôle de plus en plus élaborés. D'une situation où l'on concevait des architectures (propriétaires) fermées, où seuls des utilisateurs préalablement connus, déclarés, identifiés et autorisés pouvaient exploiter des ressources, on prône aujourd'hui l'ouverture. Celle-ci englobe en particulier la possibilité d'accepter des utilisateurs dont on ne connait rien a priori, et de donner des accès à des informations en appliquant de nouveaux mécanismes de partage d'informations. Parmi ces nouveaux modes de partage, certains posent problèmes en regard des difficultés inhérentes au respect des exigences en termes de confidentialité et de vie privée (privacy). En intégrant ces dimensions, le contrôle d'accès aux ressources n'en devient que plus complexe.
- D'une administration centralisée des utilisateurs dépendant d'une organisation unique, on en vient à déléguer l'administration des identités et des droits des utilisateurs. Au lieu de centraliser, le principe est aujourd'hui de conférer cette gestion à des entités déportées (propriétaires de processus, de données et de systèmes). Dans le cadre de l'établissement d'une coopération entre une organisation A et une organisation B, cela peut permettre à l'organisation A de demander à ce qu'un utilisateur dépendant de l'organisation B soit authentifié par l'organisation à laquelle il appartient (organisation B) pour exploiter des ressources appartenant à l'organisation A.

Si pendant longtemps l'organisation de la gestion des identités constituait un processus indépendant des programmes et des applications qui étaient exécutées, on envisage de plus en plus aujourd'hui l'intégration de ce processus au sein des activités métiers des organisations. En conséquence, le contrôle d'accès aux ressources se voit de plus en plus impacté par un lien très étroit. Les mutations dans l'organisation des entreprises sont telles que l'on doit envisager l'organisation du contrôle d'accès en tenant compte de ces nouveaux paradigmes. Dans ce contexte, l'identité prend un sens tout particulier : elle devient un élément fondamental pour l'organisation des accès sur des ressources, des applications, des programmes, voire la mise en œuvre de processus dans un domaine d'activités. Les identités sont un préalable à l'exécution d'une activité ; elles sont intimement liées à l'organisation du contrôle d'accès. Les identités sont utilisées pour contrôler l'accès aux ressources et aux données. Celles-ci doivent être protégées afin d'assurer de garantir les propriétés de sécurité qui leur sont associées. La gestion des identités et celle des accès sont fondamentalement liées et ne peuvent pas être séparées [Vacca 2009].

1.2 Gestion des identités et des accès

Le concept de gestion des accès et des identités (IAM en anglais) fait partie intégrante de la sécurité des organisations [Vacca 2009]. La gestion des identités prend en considération la problématique de création, modification et destruction des comptes et privilèges associés à des utilisateurs. Elle permet également de s'assurer de l'authentification des utilisateurs, de garantir l'accès aux ressources et services en fonction de critères polymorphes. De plus, elle doit permettre d'historiser certains accès tout en garantissant la protection des libertés individuelles, et de traiter les autorisations.

1.2.1 Concept d'identité

Sur le plan humain, l'approche des identités a donné lieu à de nombreuses études dans le domaine de la psychologie sociale. La notion d'identité est profondément paradoxale. Les résultats font état d'une articulation où se mêlent similitude, différence, unicité et communauté. L'identité résulte d'interactions complexes qui se nouent entre un individu et d'autres appartenant à un groupe. L'identité se crée en considérant entre autres des déterminants sociaux, des positions intimes, la perception de soi et le regard d'autrui. Lipiansky [Lipiansky 1992] propose que l'identité du sujet puisse être définie comme s'articulant autour d'une distinction entre identité du moi et identité du soi.

Le concept d'identité n'est donc pas simple à établir. De plus si l'on s'en réfère au dictionnaire de l'académie française le mot identité est défini sous la forme suivante : « Ce qui fait qu'une chose est la même qu'une autre, que deux ou plusieurs choses ne sont qu'une ou sont comprises sous une même identité ».

La mise en place d'une identité se justifie par la nécessité de répondre à une problématique donnée. Ainsi la délivrance d'une carte d'identité professionnelle dans une entreprise, permet à un employé (a minima) d'affirmer son appartenance à cette entreprise. La délivrance d'une carte de séjour pour un étudiant étranger permet de justifier de la régularité de sa présence sur un territoire. Un passeport permet de justifier un lien avec le pays qui en a assuré la délivrance.

On peut décrire sous une forme générique la notion d'identité comme l'existence d'une entité pour d'autres entités. Une identité est une représentation d'une entité (une personne ou une organisation ou une chose) dans un domaine d'application spécifique. Une entité peut avoir plusieurs identités dans un domaine donné. Par exemple, une personne peut être à la fois un parent et un enseignant dans une école, et avoir ainsi deux identités dans le système scolaire [Josang 2007]. Un ressortissant d'un pays peut être à la fois titulaire d'un passeport et d'une carte nationale d'identité. Une identité permet de singulariser, nommer, isoler une entité prise parmi un ensemble [Kamel 2008]. La délivrance d'une identité vis-à-vis de l'ensemble du système pour lequel cette identité est définie, doit permettre d'identifier de manière unique chaque élément de cet ensemble. Le numéro de passeport identifie de manière unique tout passeport.

Dans le monde numérique toute entité (sujet, objet) peut de manière semblable se voir associer une ou plusieurs identités dans un ou plusieurs référentiels. On associe aux utilisateurs des comptes (user account) sur un serveur de fichiers ou encore des comptes de messagerie électronique. Pendant longtemps, l'identité numérique a été considérée comme l'équivalent de l'identité classique d'un être humain ou d'une personne morale. Une identité numérique (Figure 1) se compose d'un ensemble de données descriptives (caractéristiques ou attributs) que l'on peut désigner par un identifiant. Cet identifiant est une valeur unique dans l'espace des valeurs servant à l'identification d'une entité dans le référentiel concerné (messagerie électronique, serveurs de fichiers). Lorsque l'identité est associée à une personne physique, un ou plusieurs profils peuvent être établis à partir de sous–ensembles des attributs (voire à partir de la totalité). L'identité numérique est une représentation « informatique » d'une entité.

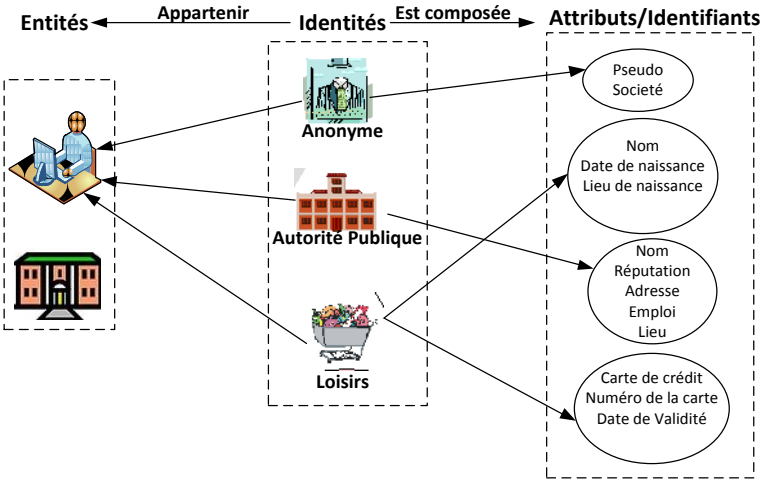

Figure 1 La relation entre les entités, les identités, et les identifiants/attributs

Comme dans le domaine classique, une identité numérique n'a de sens que si elle a une utilité : une entité est amenée à interagir avec un ou plusieurs systèmes pour la réalisation d'un ensemble d'activités. Ainsi un usager d'une entreprise accède à des serveurs de messagerie, des serveurs de fichiers, certains administrent des systèmes d'exploitation, d'autres accèdent à une base de données… Généralement un lien est établi entre un élément d'identité, ou un profil généré à partir d'une identité pour répondre à un besoin fonctionnel dans le cadre de l'accès à une ressource.

Toute identité présente un cycle de vie. La création d'une identité intervient lorsque l'on souhaite qu'une entité puisse exister dans un référentiel donné. Dans la création d'une identité, plusieurs attributs sont renseignés. Certains de ces attributs sont statiques, alors que d'autres sont dynamiques. Par exemple le nom d'un utilisateur humain est statique, alors que les pseudos qu'il utilise peuvent changer dans le temps. La modification ou maintenance des attributs fait partie du cycle de vie. Une identité peut être détruite lorsqu'une entité n'a plus de raison d'exister dans le référentiel où elle figurait.

Le cycle de vie d'une identité est classique dans le sens où il comporte une phase de création, de maintenance et de destruction. La durée de l'existence d'une identité est variable. La création d'un compte au sein d'un système informatique peut être réalisée pour une durée de plusieurs mois ou de plusieurs années. La création d'un processus associée à l'exécution d'un programme n'aura pour durée de vie que le temps de l'exécution d'un programme.

Le concept de gestion des identités est généralement décrit comme la combinaison de technologies et de pratiques pour représenter et reconnaître les entités comme des identités numériques. Aujourd'hui, elle ne se limite pas seulement à la gestion des identités pour des usagers au sein d'un système unique, ou encore au sein d'une organisation physique unique. Elle a pris un sens différent en la considérant étendue dans un réseau où est assurée la reconnaissance de personnes physiques externes voire de processus externes (clients, fournisseurs, partenaires, web services etc.). La gestion du cycle de vie recouvre l'inscription ou enregistrement, la maintenance, l'exploitation et la désinscription des identités [Toelen 2005].

1.2.2 Concept de gestion des accès

Le concept de gestion des accès fait référence au contrôle et à la sécurisation des accès aux ressources ou services, et donc à la protection des informations contre tout accès non-autorisé. Le contrôle d'accès recouvre donc un sens particulier car il garantit la confidentialité des informations. La définition du contrôle d'accès reste ambiguë. Dans certains travaux et notamment dans la norme ISO 7498-2 [ISO7498-2 1989] et le travail de Sandhu et Samarati [Sandhu et Samarati 1994], le processus

d'autorisation est présenté sous le nom du « contrôle d'accès ». Dans les travaux de l'IETF RFC 2989 [Aboba et al. 2000], la signification du « contrôle d'accès » recouvre trois processus nommés « authentification », « autorisation » et « audit ». En sécurité informatique, le « contrôle d'accès » se réfère à l'utilisation de mécanismes pour permettre aux entités authentifiées d'exécuter des actions en fonction de leur niveau d'autorisation et d'empêcher les entités de réaliser des actions non autorisées. La gestion des accès qui comporte le contrôle d'accès est généralement réalisée par la séquence de processus AAA (Authentification, Authorization, Accounting) [AAA 2000].

Le processus d'authentification est une étape préliminaire permettant de vérifier et de valider l'identité numérique d'une entité qui veut accéder à une ressource ou un service [Chong 2004]. L'absence d'un tel processus ne peut que menacer les ressources sur lesquelles portent les accès. Une ressource offerte en libre accès, l'est en particulier à des utilisateurs malicieux, qui peuvent alors profiter de cette situation pour détourner l'utilisation prévue de la ressource. Un résultat erroné d'une tentative de vérification de l'identité peut également conduire des utilisateurs malicieux à s'introduire auprès d'un système ou d'exploiter sous une forme non légitime les ressources concernées. Ce processus doit permettre de vérifier qu'un utilisateur qui réclame l'accès à une ressource est bien celui qu'il prétend être. La solution la plus courante est celle qui consiste à gérer les accès sur la base d'un couple (nom de compte d'utilisateur, mot de passe). Une autre solution est d'utiliser des certificats électroniques X.509 comme moyen d'authentification. Suivant le domaine d'application métier, ce processus de vérification peut être réalisé de manière récurrente. Ainsi, dans le domaine bancaire, l'accès à un certain nombre de serveurs n'est possible qu'après avoir décliné son identité, plusieurs fois, sur des serveurs différents et dans un temps imparti. Ce principe est équivalent à celui utilisé dans les aéroports internationaux où la production de son passeport est exigée à de nombreux endroits pour atteindre un avion ou pour pénétrer le territoire d'un pays [Reed 2004].

Le processus d'autorisation est celui qui permet de déterminer et de décider si une entité dont l'identité a été préalablement vérifiée est habilitée ou non à réaliser une ou plusieurs activités ou actions sur une ressource ou un service. Ce processus permet donc de refuser à certaines entités (non autorisées) l'accès aux ressources ou services sensibles ou non.

La traçabilité d'un certain nombre d'événements est obtenue en réalisant une journalisation de ces événements. Ainsi, la création de comptes, la modification et la suppression donnent lieu à historisation pour déterminer les utilisateurs d'un système. La journalisation des « logs » donne la possibilité de rechercher à quel moment le titulaire d'un compte a cherché à atteindre une ressource (voire la modifier). Au-delà de la vérification d'identité et de la possibilité ou non d'atteindre une ressource, les éléments d'audit et de comptabilité (accounting) permettent de retracer et/ou de suivre qui a eu accès et à quel niveau sur une ressource donnée. Il s'agit là d'une information capitale dans le contexte de la sécurité puisque ce processus permet de réaliser des analyses comportementales d'usagers sur des ressources. Dans certains cas de figure, cela permet de repérer des comportements malveillants et/ou des failles de sécurité qui peuvent affecter certains produits ou services.

Dans le domaine de la gestion des accès, ces trois processus complémentaires viennent constituer le standard d'or de la sécurité [Reed 2004].

Aujourd'hui, l'organisation des systèmes d'information est telle que les systèmes de gestion des accès ne portent plus sur des ressources uniques ou sur des organisations unitaires. La complexité est telle que le contrôle d'accès doit être établi à partir de caractéristiques propres à la ressource ou au service, en considérant des éléments de localisation géographique, temporels, contextuels… [Thion 2008]. Ainsi le contrôle d'accès peut porter sur des applications spécifiques, avec franchissement de pare-feux, exploitation de VPN, d'unités de stockage, des bases d'informations maintenues soit au sein de sa propre organisation soit au sein d'une organisation partenaire… La complexité est aujourd'hui si forte que les travaux de recherche portent sur la prise en compte de la complémentarité des aspects en termes d'organisation, de formalisation, de sémantique, d'architecture… pour offrir un système de gestion des accès qui soit le plus adapté aux exigences les plus ouvertes possibles. Cette complexité vient de la diversité des profils d'usagers, des ressources, des équipements, des relations établies «

Man to Man », « Machine to Machine », « Man to Machine ». Si à l'origine, la gestion des accès était réglée sous une forme simple et directe (ex : accès à un shell sur une machine), aujourd'hui il apparait de plus en plus comme le résultat d'évaluations de situations composites : par exemple l'accès à une donnée peut engendrer des accréditations différentes si l'on est dans l'enceinte d'une entreprise ou si l'on est à l'extérieur de celle-ci. Les solutions de gestion des accès doivent évoluer pour être en adéquation avec l'évolution des exigences naissant de l'adoption des technologies au sein des organisations.

1.2.3 Changements de perspectives dans la gestion des identités et des accès

Au cours de la préparation de cette thèse, notre travail principal ayant porté sur la conception et le développement d'un système d'autorisation adaptable, nous avons souhaité établir dans cette section un état de l'art sur l'adaptation dans le temps des solutions de gestion des identités et des accès. Cet état de l'art est établi en considérant, le niveau d'intégration des technologies informatiques ainsi que l'évolution des exigences de sécurité des organisations.

1.2.3.1 De la création de simples comptes à la gestion des identités

La première rencontre avec une solution de gestion des identités et des accès est venue de la volonté de satisfaire deux exigences fondamentales des métiers de la gestion d'entreprise : améliorer la sécurité des systèmes d'information et réaliser des économies financières en s'intéressant à l'architecture de ces systèmes.

L'exploitation de l'informatique dans le secteur des administrations et des services date du début des années 60. Les coûts associés aux équipements matériels sont tels qu'il n'est pas alors envisageable d'en équiper toutes les entreprises et organisations. En conséquence les seuls bénéficiaires sont les grands comptes ou des sociétés spécialisées en ingénierie informatique. Le système informatique est organisé autour d'une unité centrale, ou mainframe, à laquelle sont connectés des terminaux constitués sous forme de grappes. Les opérateurs travaillent directement à partir de ces terminaux. L'introduction de l'informatique dans les entreprises ou la commercialisation de services permet de rationaliser certaines activités du fonctionnement des organisations [Akram 2005]. Les programmes principaux sont dans certains cas spécifiques à un métier donné (calcul de structures, simulation, etc.), ou sont encore des programmes à portée générale dans le monde de la gestion d'entreprise.

En matière de sécurisation, les risques principaux sont davantage liés à la fragilité des équipements matériels retenus pour fabriquer les calculateurs et au fonctionnement des programmes. La sûreté de fonctionnement est un domaine qui concentre de nombreux efforts. Les solutions de gestion des accès mis en œuvre sont relativement sommaires et visent principalement à se protéger de maladresses. Le contrôle porte davantage sur les conséquences de l'exécution des programmes pour éviter de perdre les résultats fournis (contrôle d'exécution) et sur des mécanismes de protection de l'accès aux données stockées sur les machines (droits d'accès aux fichiers et aux commandes).

Une conception différente du déploiement de la technologie est apparue avec la commercialisation massive de microordinateurs ou ordinateurs personnels au début des années 80. Les organisations ont commencé à se tourner vers ce type de systèmes pour répondre à des demandes métiers émanant de secteurs autres que ceux de la gestion. La mise en réseaux de ces types de systèmes a conduit rapidement à la nécessité de sécuriser les flux de données transmis et de protéger les informations présentes au sein de ces systèmes [Mann 2007]. Le contrôle d'accès prend alors place au sein des systèmes d'exploitation. L'authentification est faite par un couple (nom d'utilisateur, mot de passe) et l'autorisation est donnée en fonction de droits (rwx sous Unix) associés à des répertoires et fichiers présents au sein des systèmes. L'accounting est également mis en place au sein des

systèmes d'exploitation principalement pour suivre les heures de connexion et de déconnexion des usagers.

Par la suite, le déploiement des infrastructures conduit à donner un sens différent à la gestion de la sécurité. La nécessité d'avoir une infrastructure en permanence opérationnelle (contrainte de disponibilité) a conduit à consolider toute la partie architecture réseaux et systèmes. En particulier les premiers antivirus ont été déployés pour contrecarrer les menaces de ce type, et la mise en place de serveurs de fichiers au sein des organisations a permis de mieux organiser l'accès aux données. Un serveur de fichiers permet de partager des données entre utilisateurs à travers un réseau interne. La gestion des accès a été essentiellement déployée pour limiter l'accès à des ressources de type fichiers présents sur les serveurs et postes de travail en s'appuyant sur une gestion de comptes usagers, et aussi pour protéger les flux de données échangés sur le réseau. Généralement, une organisation typique est constituée d'une grande variété de systèmes d'exploitation et d'applications.

La première identité numérique est donc apparue lorsque l'on a associé la paire (identifiant, mot de passe) ou tout d'autre secret partagé à un utilisateur physique. Le couple (identifiant, mot de passe) est utilisé pour réaliser l'authentification lors de la connexion à un compte ou un répertoire. Il est sensé « prouver » l'identité de l'utilisateur à condition de respecter un certain nombre de consignes strictes sur la non diffusion des identifiants et mots de passe ou secrets partagés. Dans le cas contraire, il n'y a aucune preuve possible. La fragilité de ce système a conduit à l'introduction d'accréditations. La multiplication des systèmes au sein d'un réseau et la multiplication des bases de comptes utilisateurs a rendu complexe les opérations de gestion et de suivi des comptes utilisateurs et de leurs accréditations. La gestion des identités est alors apparue dans cette période des années 80. Le premier système de gestion des identités a été donné par la publication de la recommandation X.500 [ITU 2011], élaboré par l'ITU et couvrant les services d'annuaire tels que Directory Access Protocol (DAP) pour stocker et organiser les identités. L'ISO a également été associée au développement de ce standard. Comme beaucoup de standards de l'ITU, celui-ci était très lourd et complexe [Josang 2007]. Une version légère publiée au début des années 1990 pour le DAP, a été proposée dans le cadre LDAP. Normalisé par l'IETF, cette solution d'annuaire a été adoptée par Netscape. Microsoft a inventé « Active Directory » équivalent d'annuaire LDAP. Ce sont des solutions pour organiser et stocker de manière sûre les identités.

D'autre part, le contrôle d'accès (dans le sens d'autorisation) recouvre les mécanismes permettant de prévenir l'utilisation non autorisée de ressources. Les mécanismes de contrôle d'accès sont par exemple, dans le cas des systèmes d'exploitation multiutilisateurs, les indicateurs permettant de définir les droits des utilisateurs sur les fichiers, c'est-à-dire, lire, écrire, exécuter, etc. Il peut s'agir également de l'association d'une liste de contrôle d'accès associant des comptes utilisateurs à des ressources, ou encore de listes de règles de contrôle d'accès définies sur des routeurs filtrants, firewalls, commutateurs, etc. La protection de chacune des ressources ou applications, est faite par la construction d'une structure de données contenant une liste de comptes et une liste de contrôle d'accès (ACL : Access Control List). Ces éléments sont définis pour identifier les utilisateurs ayant droit d'accéder à la ressource et leur accorder les droits qui leurs sont assignés. Très souvent efficace au sein d'une organisation, cette solution est adoptée pour gérer les droits d'accès d'utilisateurs internes à une organisation unique [Kamel 2008].

La spécialisation de certains systèmes et la multiplication des serveurs a, au fil du temps, conduit à une croissance du nombre d'identités à gérer et à la multiplication d'identités associées à un même utilisateur physique. Dans un grand nombre d'organisations, les utilisateurs typiques sont amenés à se connecter plusieurs fois, parfois sur différents systèmes, pour pouvoir lancer l'exécution d'applications propres à l'exercice d'un métier. Du point de vue de l'utilisateur, ces multiples connexions et la nécessité de retenir de nombreux mots de passe peuvent conduire à de mauvaises expériences. Les utilisateurs des différents systèmes doivent avoir différents comptes et donc rentrer les mêmes informations sur leur identité ce qui augmente la difficulté de gestion. Par ailleurs, les utilisateurs sont amenés à retenir des identifiants et mots de passe multiples ce qui devient à terme un obstacle aux bons usages. Un vrai problème est celui de l'oubli des mots de passe et parfois des

identifiants de comptes dû le plus souvent à une utilisation rare de certains d'entre eux. Du point de vue de la direction (gouvernance), les mots de passe oubliés augmentent considérablement les coûts de gestion et conduisent à de mauvais réflexes pour les utilisateurs (qui les inscrivent sur des blocs-notes, dans des fichiers par exemple). Ces oublis sont la porte ouverte aux failles de sécurité [Vacca 2009].

Le concept d'authentification unique (Single Sign-On, SSO) est apparu comme une solution d'authentification pour éliminer les problèmes de mots de passe multiples et dangereux et simplifier l'administration des systèmes. Conceptuellement, il permet aux utilisateurs de se voir accorder l'accès à toutes leurs applications ou données avec un identifiant et mot de passe unique. Par exemple, « Kerberos » [Kerberos 2011] est un protocole d'identification réseau conçu pour fournir un mode d'authentification unique et forte pour les applications clients/serveurs en utilisant le principe de chiffrement à clé secrète.

La conduite de partenariats entre des organisations (facilités en particulier par l'ouverture des systèmes d'information et par le déploiement des structures réseau) a conduit à s'interroger sur l'approche qui devait être faite en matière de gestion des utilisateurs et des ressources mises à disposition dans le cadre d'un partenariat. Ce contexte coopératif a montré les limites de l'approche d'une gestion des identités centralisée. Certes, la centralisation facilite au sein d'une entité administrative unique la gestion des identités des entités. Cependant dans un contexte multi-administratif comme l'environnement des organisations virtuelles, il impose d'exporter soit une partie, soit la totalité des informations maintenues dans les annuaires de chacune des organisations vers un tiers. Cet export n'est pas toujours possible, compte tenu des politiques de sécurité en vigueur dans les organisations (problématique de confiance). De plus, il induit une lourdeur dès que le nombre de partenaires commence à devenir important (passage à l'échelle). Il nécessite aussi que des procédures soient clairement établies pour prendre en compte sans délai de latence toute demande de modification qu'un partenaire souhaite voir opérer sur les droits d'un utilisateur, ou sur des ressources (prise en compte d'une fin de contrat d'un employé ayant jusque là des droits sur des ressources, changement d'affectation d'un employé, etc.). Des solutions ont alors été étudiées pour promouvoir une décentralisation de la gestion des identités dans le domaine de coopération entre des organisations [KAMEL 2008] grâce à l'utilisation de solutions de fédérations d'identités telles que SAML [SAML 2011] et WS-Federation [Goodner et al. 2007].

1.2.3.2 De l'affectation de droits élémentaires à une gestion complexe de l'accès aux ressources

La gestion des accès aux ressources s'est complexifiée au fur et à mesure de l'adoption des technologies de traitement de l'information. A l'origine, la gestion des accès se limitait à associer à des comptes utilisateurs, un ensemble de droits d'accès à des fichiers et/ou dossiers. Par la suite des pouvoirs ont été octroyés à certains comptes utilisateurs pour qu'ils puissent exercer des actions particulières sur certaines ressources (organisation de sauvegarde, gestion de périphériques, etc.). Afin de simplifier la tâche de l'administrateur qui était sollicité pour toute modification des droits accordés à des utilisateurs lambda, des solutions de délégation ont été proposées. C'est ainsi que la notion d'utilisateurs privilégiés ou avec pouvoir a été introduite au sein des organisations. Afin de conserver la maîtrise et la cohérence des actions exercées par l'administrateur principal et les utilisateurs avec pouvoir, des solutions ont été recherchées pour organiser au mieux cette opération de délégation.

En pratique, le travail assuré par un administrateur principal ou délégué est celui d'accorder ou de supprimer des droits d'accès et des privilèges à des entités. Si à l'origine les entités concernées étaient des utilisateurs et des ressources matérielles, aujourd'hui ces entités sont aussi bien des utilisateurs que des applications, des ressources matérielles, logicielles ou encore des organisations. Le processus permettant d'accorder des privilèges pour l'accès à des ressources s'inscrit aujourd'hui dans un cadre plus large que celui de l'autorisation dont nous avons rappelé l'existence dans la section précédente. Sa mise en pratique est faite par l'application de règles, de stratégies, où les restrictions apportées aux ressources sont liées aux fonctions et aux données métiers apparaissant dans une organisation. Dès lors la construction de ce système d'autorisation est réalisée en considérant les

ressources et les entités qui exercent des actions sur celles-ci. Il faut en particulier inventorier la liste des actions qui peuvent affecter chaque ressource. Il faut essayer de regrouper les entités qui vont exercer ces actions sur les ressources afin de simplifier la gestion des habilitations. Par exemple, au sein d'une entreprise, un groupe « Manager » pourrait être identifié afin de donner aux entités y figurant la possibilité d'appeler le service d'approbation des emprunts. Dans ce cas, la définition du « métier Manager » pourrait refléter les responsabilités des entités exerçant les activités de ce métier. En raisonnant ainsi, un système d'autorisation ne prend plus simplement en compte la reconnaissance d'un droit d'accès en fonction de l'identifiant d'une entité, mais il accorde l'accès à une ressource par rapport à la fonction d'une entité dans une organisation. Cela permet donc d'envisager la cohérence de la gestion des utilisateurs avec une stratégie d'entreprise, voire de maintenir une cohérence entre des entités programmes et cette même stratégie d'entreprise.

Cette volonté de maîtriser les actions que peuvent exercer les utilisateurs (au sens général) sur des ressources (tout aussi générales) a conduit à des propositions de modèles formels qui seront rappelés dans le chapitre 2. Ces modèles formels, lorsqu'ils sont appliqués amènent à définir avec précision les exigences de sécurité auxquelles une organisation souhaite répondre. Ils amènent également une modélisation des utilisateurs, des entités ou plus généralement des sujets. Ils permettent aussi une modélisation des ressources ou plus généralement des objets et des actions qu'un sujet peut exercer sur un objet.

Dans le passé, la réalisation des accès sur des ressources était guidée par l'analyse de listes de contrôle d'accès (ACL), ces dernières étaient utilisées traditionnellement dans les systèmes de contrôle d'accès discrétionnaires (*DAC : Discretionary Access Control*) [Harrison *et al* 1976]. Aujourd'hui l'heure est à la construction de politiques de contrôle d'accès. Au lieu d'exprimer des contrôles en s'attachant aux ressources de bas niveau, les politiques de contrôle d'accès permettent d'exprimer à haut niveau les actions réalisables par des sujets sur des objets. Elles apportent une flexibilité autre que celle proposée antérieurement au sein des ACLs. Les ACLs ont montré leurs limites lorsque, en particulier le nombre d'utilisateurs (internes et externes) et d'applications à gérer augmente. Adaptée aux situations stables, les ACLs s'accommodent mal d'un accroissement du nombre d'éléments à gérer et de destructions, créations et modifications portant sur les éléments gérés.

De plus, les politiques ne se limitent pas uniquement à spécifier des règles permettant d'accorder explicitement l'accès, appelées règles de permission. Parfois, il est plus simple d'exprimer dans les politiques des règles en termes d'interdictions pour restreindre explicitement l'accès, appelées règles d'interdiction [Cuppens-Boulahia 2008]. Lorsqu'une politique contient ces deux types de règles, des conflits peuvent apparaître et doivent être résolus par la solution d'autorisation mise en place [Nasser 2006].

Si la construction des ACLs a reposé sur l'exploitation de commandes, aujourd'hui la construction des politiques de contrôle d'accès repose sur l'utilisation de langages permettant de spécifier une politique et d'en construire les règles. Des plateformes et/ou des outils connexes peuvent permettre de détecter des situations de conflits entre règles ou entre des politiques. Le critère d'extensibilité des langages peut aussi apporter davantage de flexibilité aux politiques, ce que nous avons proposé dans notre travail.

1.2.3.3 Modèles de coopération des organisations et fédération d'identités

L'adoption puis le déploiement du réseau Internet à l'échelle mondiale, alliés à la pression de la globalisation économique, a des conséquences depuis plus de dix ans sur l'organisation et le fonctionnement des organisations. Les crédos de la flexibilité et de l'ouverture ont amené les entreprises à ouvrir tout ou partie de leurs systèmes d'information sur le monde extérieur. Les réseaux se sont organisés autour d'intranet, d'extranet et la création de portails Internet a gagné au fur et à mesure les organisations. Les réseaux internes des entreprises se sont ainsi ouverts et l'exploitation de

services d'EDI (échanges de données informatisées), et d'outils de groupware ont permis le développement du travail collaboratif à distance. Cette ouverture des réseaux d'entreprise impacte depuis le contrôle d'accès dans la mesure où l'accès à des objets dépendant d'une première organisation doit être procuré à des sujets dépendant d'une autre organisation. Cette situation complexifie l'écriture des politiques de contrôle d'accès et leur mise en œuvre

Les solutions de collaboration ponctuelles et menées à l'origine au sein d'une même organisation ont amené à la formation d'organisations virtuelles (OV). Les organisations virtuelles redessinent les frontières des entreprises en ce sens qu'elles donnent la possibilité à plusieurs organisations de se réunir dans le but de répondre communément à un projet, à une activité économique. Chaque organisation apporte dans le cadre de l'OV ce qu'elle sait faire de mieux, en s'appuyant sur le savoir-faire des autres et en échangeant des informations par le biais des technologies d'information (TI) notamment en utilisant le Web [Nasser 2006]. Chaque organisation apporte des ressources nécessitant la mise en œuvre de systèmes de gestion des accès spécifiques, pour n'offrir aux partenaires que le strict accès nécessaire dans le cadre du respect d'accords contractuels.

En changeant le regard et les usages en matière de partage d'informations, les organisations doivent ouvrir leurs systèmes d'information pour accueillir des utilisateurs externes et offrir de nouveaux types de partage d'information, tout en veillant strictement aux exigences de sécurité de la vie privée (privacy) et de la confidentialité. L'ouverture des systèmes d'information en environnements réseaux a nécessité dans un premier temps l'introduction d'équipements spécifiques tels que des routeurs et/ou des pare-feux pour organiser la protection des ressources et des équipements connectés aux réseaux d'entreprise. De telles solutions n'ont pas permis de protéger efficacement les différentes entités. Très rapidement, des exigences en termes d'authentification forte, de confidentialité et d'intégrité ont été ressenties. Pour apporter ces niveaux de service, des techniques de chiffrement ont été exploitées. Ces dernières ont été intégrées au sein de piles de protocoles de communication. Par exemple le protocole SSL/TLS a offert le chiffrement de données transmises entre une entité cliente et une entité serveur. Cette pile de protocole a été utilisée pour le développement de services sécurisés tels HTTPs, FTPs... La mise en place de tels services repose sur l'exploitation de certificats. Un certificat numérique est une carte d'identité numérique dont l'objet est d'identifier une entité physique (utilisateur ou machine) (cf. 1.2.1). Il contient tout un ensemble d'informations sur l'entité concernée. La norme ITU-T X.509 [RFC5280 2008] relative aux infrastructures à clés publiques (ICPs) décrit entre autres les formats standards des certificats électroniques et offre les moyens d'en vérifier la validité.

Bien que les ICPs soient fortement déployées aujourd'hui, exploitées au sein des navigateurs WEB, elles posent cependant des problèmes de confiance [Wazan 2011], car les autorités de certification peuvent être inégales. De plus, la complexité des relations qui unissent les organisations membres d'un réseau collaboratif fait qu'elles peuvent ne pas être adaptées dans leur état d'utilisation actuel à l'établissement d'une chaîne de certification [Kamel 2008].

Les fédérations d'identité [Clusif 2007] ou authentification répartie ont été proposées pour des architectures ouvertes et à grande échelle comme une solution de partage d'informations d'identité. Elles donnent la possibilité de contrôler les accès réalisés à distance sur des ressources appartenant à des organisations différentes. Le principe majeur est de demander à l'organisation dont dépend une entité, d'en vérifier l'identité lorsqu'elle souhaite accéder à une ressource qui dépend d'une organisation partenaire. La fédération d'identité se concrétise pour un groupement d'organisations, par une interconnexion des services d'authentification et sur la base d'un ensemble d'attributs d'identité communs. Un fournisseur d'identités (IdP : Identity Provider) gère un ensemble d'utilisateurs (identificateurs et attributs). Un fournisseur de services (SP) est une entité (e.g. établissement, administration, entreprise, etc.) qui propose une ressource numérique en ligne au sein de la fédération. Techniquement, les relations de confiance entre les membres d'une fédération reposent sur des certificats électroniques comme les certificats à clés publiques X.509 et des métadonnées partagées (avec la limite indiquée précédemment). En outre, la confiance s'établit administrativement entre les participants de la fédération au travers de conventions de partenariats (e.g. signature de contrats entre

organisations). Une même organisation peut participer à plusieurs fédérations et gérer des partenariats de manière bilatérale.

SAML [SAML 2011] et WS-Federation [Goodner et al. 2007] sont des exemples de solutions pour l'authentification unique (SSO) sur le Web et la fédération d'identité pour la gestion des identités et des processus d'authentification. SAML et WS-Federation proposent l'échange de données d'authentification et d'autorisation entre des domaines de sécurité distribués. SAML permet aux partenaires de générer des assertions concernant l'identité, les attributs et les droits d'une entité (utilisateur, machine) et, les transférer à d'autres entités (organisations, applications, etc.). Il définit la syntaxe et les règles pour demander, créer, communiquer et utiliser les assertions SAML au format spécifié. Les assertions SAML, encapsulées dans des messages et transférées par le biais du protocole HTTP, permettent à la fédération d'outrepasser les limites qui pourraient naître des différences marquant les infrastructures déployées chez différents partenaires. WS-Federation se focalise davantage sur la gestion des identités et l'authentification des services Web mis en œuvre au cours de l'exécution d'une application.

1.2.3.4 Organisation de la gestion des accès dans un environnement collaboratif

Bien que les infrastructures à clé publiques et la fédération d'identité fournissent des réponses aux problématiques de la gestion des identités et de leur authentification (utilisateurs, ordinateurs, applications, programmes… appartenant à un environnement collaboratif), elles restent incapables de fournir des solutions complètes et matures pour la gestion des accès et en particulier le processus d'autorisation.

En effet, lorsqu'une organisation est amenée à partager des ressources critiques et à grande échelle (e.g. des bases de données, des applications de simulation) avec des partenaires extérieurs potentiels, elle doit être en mesure de déterminer qui est autorisé à accéder à ses ressources et qui ne l'est pas. Idéalement, le fournisseur de services (par le biais d'un administrateur) doit être capable de définir la politique de contrôle d'accès exprimant « qui » est autorisé à faire « quoi ». L'infrastructure d'autorisation quant à elle applique la politique et les règles d'accès. Ainsi, une infrastructure d'autorisation offre aux fournisseurs de services le moyen de séparer les règles d'accès des ressources à protéger permettant ainsi un passage à l'échelle et une souplesse (i.e. les règles d'accès doivent être modifiées si les contextes d'accès ont changé). Lorsque plusieurs partenaires sont engagés dans une collaboration, même s'ils peuvent s'accorder sur une architecture de l'infrastructure d'autorisation, ils peuvent avoir différents points de vue concernant la gestion des accès [Nasser 2006]. En effet, les approches de gestion des accès dans un environnement collaboratif peuvent être centralisées ou distribuées. Dans la gestion centralisée, toutes les décisions pour savoir si un utilisateur d'une organisation « A » est en mesure d'accéder à une ressource donnée dans une organisation « B », sont prises et mises en œuvre en un point central qui peut être une « super-organisation ». La centralisation implique la délégation de la gestion de l'organisation collaborative à une autorité centrale comme la solution de contrôle d'accès proposée par Globus « Community Authorization Service » CAS [Pearlman et al. 2002]. Ainsi, le système VOMS (Virtual Organisation Management System) [Alfieri et al. 2005] partage avec CAS la même vision de l'organisation collaborative mais considère que la décision d'accès à une ressource est propre à son propriétaire. Aussi, ce système propose l'utilisation de « rôles » pour la gestion des utilisateurs et de leurs droits dans une OV [Alfieri et al. 2005]. Cette solution centralisée devient vite compliquée quand le nombre de partenaires commence à devenir important (passage à l'échelle), et que les activités deviennent un ensemble de processus métiers dans lesquels plusieurs ressources et utilisateurs peuvent intervenir selon les besoins et les contraintes propres de chaque organisation membre.

Dans la gestion distribuée, chaque organisation est responsable de ses propres décisions de contrôle d'accès et de leur mise en œuvre. La notion de délégation d'autorité est remplacée par la reconnaissance d'autorité. Un certain niveau de confiance est établi entre les différents partenaires pour permettre à chacun de gérer ses utilisateurs et ses ressources. Le propriétaire des ressources

contrôle les droits sur ses ressources. D'un autre côté, les utilisateurs qui vont accéder à ces ressources sont nommés par leur organisation. Le propriétaire des ressources fait confiance à son partenaire pour qu'il décide des utilisateurs qui s'engagent à une « bonne exploitation des ressources ». A l'inverse, une organisation fait confiance au propriétaire des ressources pour fournir les bonnes ressources et s'assurer de leur bon fonctionnement.

La politique de contrôle d'accès ou d'autorisation qui régit l'accès doit contrôler et spécifier les contextes dans lesquels des utilisateurs d'un certain partenaire peuvent accéder à des ressources (chez ce partenaire ou d'autres membres). Etant donné que les différentes organisations maintiennent le contrôle sur leurs utilisateurs et ressources chacune de son côté, le système de gestion distribué de contrôle d'accès doit permettre aux différentes autorités de faire partie du processus de gestion. En effet, XACML (eXtensible Access Control Markup Language) [XACML 2011] propose un langage standard qui intègre nativement la notion d'extensibilité pour l'expressivité des politiques contenant des informations de sécurité diverses et associées à des exigences propres aux organisations. Ainsi, XACML est basé sur une architecture flexible qui rend dynamique un certain nombre de tâches de gestion sans avoir à stopper ou recoder le système, et qui favorise l'intégration du système de gestion dans un environnement à gérer. Par contre, un système d'autorisation qui implémente le standard XACML peut être utilisé comme une solution de gestion distribuée. Cependant, ce système doit être facilement étendu et suffisamment ouvert pour prendre en compte des exigences spécifiques à ces environnements distribués dans des situations différentes (différentes organisations, différents besoins d'autorisation, différents environnements informatiques). Pour cela, une étude des besoins d'adaptation de ce système est nécessaire, ce que nous proposons dans cette thèse.

1.2.3.5 Mobilité et ubiquité versus gestion des accès et des identités

Aujourd'hui, l'essor des technologies a ouvert la voie à de nouvelles applications et de nouveaux usages qui sont en passe de révolutionner des techniques de travail largement établies depuis des années. Des bénéfices particuliers sont par exemple attendus des champs d'application de l'informatique ubiquitaire, du Cloud computing [Escoffier 2008] [Somashekar 2010], de la construction de réseaux sociaux d'entreprise ou grand public. L'informatique ubiquitaire, appelée également informatique pervasive, a pour but de rendre accessible toutes sortes de services, n'importe où, avec l'ambition de remettre en cause les modes de travail classiques des utilisateurs en regard de l'utilisation des équipements informatiques (terminaux d'accès, exécution de programmes…). Cette volonté d'affranchir l'utilisateur des contraintes actuelles d'utilisation comme celle d'être assis devant un ordinateur, d'utiliser un clavier, une souris doit amener à restituer la liberté d'actions et de mouvement aux utilisateurs. En ce sens le développement de l'ubiquité nécessite d'aborder les problématiques inhérentes à celles de la mobilité.

Les réseaux et leurs applications deviennent ainsi le théâtre privilégié des interactions homme à machine, machine à machine, ou entre individus que ce soit dans un cadre professionnel d'entreprises, dans les rouages des états ou encore dans l'organisation de la vie privée des individus. La commercialisation de nouveaux terminaux toujours plus évolués (type Smartphone, tablettes…) alliée à l'extension et la diversification des réseaux d'accès (ADSL, FTTH, 3G, Wi Max...), repoussent les frontières géographiques traditionnelles du monde de l'entreprise ou du domaine privatif. A titre privatif, l'utilisation d'équipements du type ordinateur portable, smartphone, tablette… peut permettre une prolongation de la vie familiale hors du domicile privatif. Sans porter de jugement de valeur sur les conséquences sociales et humaines de ces nouveaux usages à court, moyen et long terme, nous pouvons noter qu'ils sont dans un premier temps associés à un engouement pour la technologie. A minima, ils permettent dans certaines situations d'assouvir une curiosité en regard des possibilités offertes pour la reconstruction de certaines interactions humaines. Ainsi, les consoles de jeux proposent aujourd'hui une restitution de la mobilité en donnant la possibilité de réaliser une partie de golf, de bowling… et d'organiser parfois ces jeux en réseau sans-fils. A titre professionnel, les équipements tablettes, ordinateurs portables sont pour certains employés, le moyen de prolonger à leur domicile (ou ailleurs) l'activité qu'ils ont au sein de l'entreprise. Dans le cadre de l'exercice d'un

télétravail l'utilisation de ces équipements constituent la porte d'entrée au sein d'un bureau (parfois virtuel).

La gestion des accès à des ressources prend ici une dimension autre. La prise en compte de la mobilité amène à s'interroger sur la nature des mécanismes et outils à proposer pour répondre le mieux possible à cette prolongation des univers de l'entreprise ou privatif. La conception de moteurs d'autorisation et/ou de contrôle d'accès doit être faite en considérant les conséquences liées à la mobilité et traiter de contraintes propres à des domaines applicatifs spécifiques. Elle nécessite qu'une attention particulière soit portée sur la manière de constituer des identités et de les gérer. Dans la section consacrée à l'identité numérique, nous avons rappelé qu'une identité se compose d'un ensemble de données descriptives dont un identifiant garantissant l'unicité d'une entité dans le système pour lequel l'identité est créée, des caractéristiques statiques et d'autres qui sont dynamiques. Le travail de réflexion sur la création d'une identité peut amener nombre d'interrogations dont l'opportunité de mettre en place une autorité unique de délivrance d'une identité ou de considérer l'existence de plusieurs autorités et de construire des fédérations d'identité comme préalable à l'exploitation de ressources. Des questions relatives aux caractéristiques de l'identité peuvent également être soulevées pour déterminer comment garantir le respect de la vie privée, pour déterminer ou non l'opportunité d'y faire figurer des informations associées à une personne physique, sur l'opportunité de traiter de l'anonymat etc.

Si ce questionnement doit conduire à déterminer la nature des caractéristiques statiques, variables voire dynamiques à retenir pour constituer une identité [Arabo et al. 2009], d'autres questionnements lient le contrôle d'accès à la continuité d'exécution d'un service. Par exemple, il est commun dans le cadre de l'établissement de communications téléphoniques, qu'un usager équipé d'un smartphone se déplace sans que pour autant la communication engagée avec un interlocuteur soit rompue. De la même façon il peut continuer de consulter sa messagerie. Cette possibilité doit être étendue à d'autres services ubiquitaires. Il faut être à même de déterminer ce qui conditionne l'exécution d'un service, et de manière identique ce qui conditionne l'accès à une ressource. La mobilité conduit ainsi à considérer la problématique de la contextualisation pour aborder dans toutes ses dimensions le contrôle d'accès [Hengartner 2005]. Le contexte peut être défini comme un ensemble d'informations utiles pour caractériser les contraintes, les situations, les entités… qui influent sur le comportement de l'exécution d'un processus ou qui guident les interactions entre entités. Ces informations peuvent être d'ordre temporel, spatial, géographique, historique etc.

Dans le domaine du contrôle d'accès, l'utilisation de la notion de contexte été proposée par plusieurs modèles formels dont par exemple le modèle Or-BAC [Cuppens et al. 2003]. L'une des attentes était par exemple de guider le contrôle d'accès à un dossier médical réalisé par un médecin (consultation, modification) en retenant la localisation physique du médecin ou de la plage horaire de consultation du dossier. Que ce soit pour Or-BAC ou d'autres modèles formels, les difficultés rencontrées, proviennent des conséquences des changements des valeurs des informations contextuelles, mais également des changements qui affectent la construction ou la modification des règles de contrôle d'accès et au-delà de la prise en compte de nouvelles technologies. Compte tenu de l'importance que revêt le contexte pour les applications futures, il faut envisager de concevoir et développer un système d'autorisation suffisamment flexible pour qu'il puisse supporter toutes les contraintes exposées. Dans le cadre du travail mené au cours de cette thèse, nous avons élaboré une synthèse de l'évolution des modèles formels de contrôle d'accès. Elle est présentée au chapitre 2 alors qu'au chapitre 3 nous traitons par ailleurs de la notion d'adaptabilité afin de proposer un moteur de décisions d'autorisation suffisamment flexible pour répondre à ces évolutions.

1.3 Autorisation

Rappelons que l'autorisation a pour fonction d'évaluer la capacité de décider d'autoriser ou non des sujets (utilisateur, programme en cours d'exécution, calculateur, etc.) à réaliser des activités/actions sur un ou plusieurs objets/ressources (fichier, dossier, bâtiment, application informatique spécifique, base de données, serveur etc.).

1.3.1 Les architectures d'autorisation AAA

Les architectures d'autorisation ont été bâties dans un premier temps autour d'un serveur unique permettant de garantir les trois fonctionnalités de contrôle d'accès qui sont l'authentification, l'autorisation et l'audit (AAA), rappelées dans la section (1.2.2). Pour répondre aux besoins relevés en termes de gestion des accès (qui a le droit de faire quoi, comment et dans quelles circonstances), les entités de base participant à une autorisation [RFC2904 2000] sont les suivantes (Figure 2) :

1. Un utilisateur qui demande l'accès à un service ou une ressource
2. Le serveur AAA du fournisseur de service qui autorise l'accès au service en respect du contrat signé avec l'organisation de tutelle de l'utilisateur
3. L'équipement du fournisseur de services ou gestionnaire de ressources qui assure l'exécution du service proprement dit

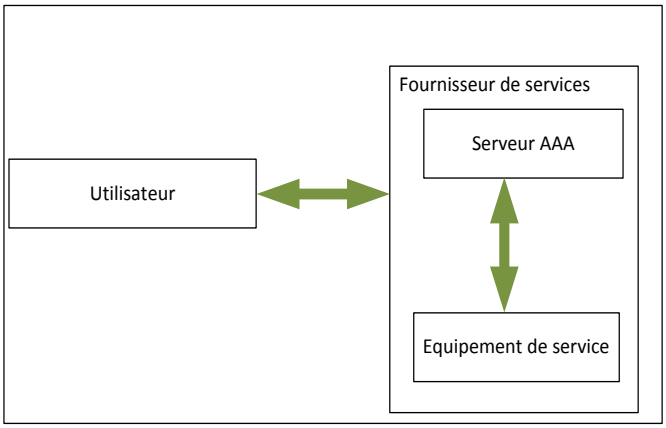

Figure 2 Architecture d'autorisation AAA

Ces architectures d'autorisation ont amené à introduire une dissociation entre les ressources et les processus d'authentification et d'autorisation alors que traditionnellement les mécanismes de protection étaient intégrés aux applications et/ou associées aux ressources. C'était au moment de l'exécution d'un programme que l'on vérifiait les habilitations d'un utilisateur. Pour les ressources, une structure de données contenant une liste de comptes d'utilisateurs et une liste de contrôle d'accès (*ACL : Access Control List*) étaient établies jusque-là. Les interactions dans les architectures d'autorisation AAA peuvent suivre les modèles de déploiement (Agent, Pull, Push) décrits ci-après.

1.3.1.1 Les modèles de déploiement des architectures AAA

Dans cette section, nous présentons les différents modèles de déploiement des architectures d'autorisation AAA.

Le modèle Agent

L'utilisateur formule sa requête de demande d'accès à un équipement de service auprès d'une partie tierce (serveur d'autorisation AAA). Celle-ci est l'agent situé entre l'utilisateur et l'équipement de service (Figure 3). Le serveur AAA applique la politique qui a été préalablement établie pour la demande en cours. Si l'accès est autorisé, le serveur transfère la requête de service ; sinon, il renvoie un message d'interdiction à l'utilisateur ayant formulé la requête. L'équipement de service retourne le résultat de la requête au serveur AAA qui à son tour renvoie la réponse à l'utilisateur.

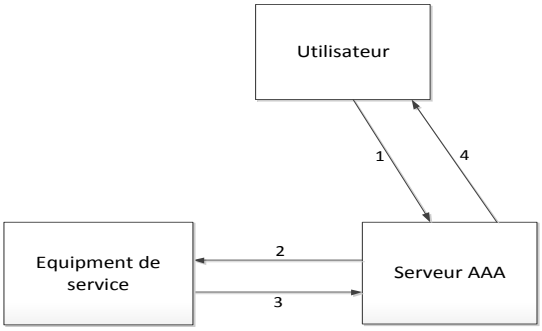

Figure 3 le modèle Agent

Le modèle Push

En premier lieu, l'utilisateur demande l'autorisation de sa requête au serveur AAA (par exemple, via serveur d'autorisation) (Figure 4). Si le serveur AAA autorise la requête du sujet, il envoie à l'utilisateur un message sécurisé (jeton ou certificat) qui agit comme une preuve de droit (assertion d'autorisation). Généralement une telle assertion a un temps de validité qui lui est associée. L'assertion peut ensuite être utilisée par l'utilisateur pour demander un service spécifique en communiquant avec l'équipement de service. L'équipement de service applique la décision qui sera d'accepter ou de refuser l'assertion d'autorisation. Un tel système est mis en œuvre dans la distribution de tickets comme Kerberos [Kerberos 2011].

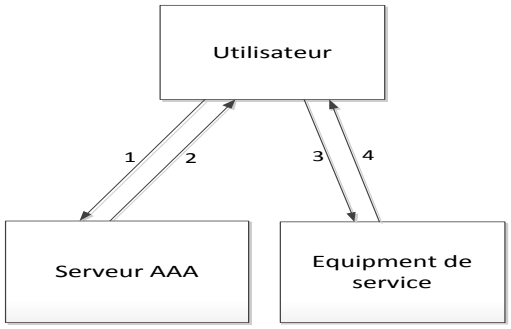

Figure 4 Le modèle Push

Le modèle Pull

En premier, l'utilisateur contacte l'équipement de service directement avec une requête. Pour accorder ou refuser la requête de l'utilisateur, l'équipement de service contacte l'autorité d'autorisation (serveur AAA) (Figure 5). Ensuite le serveur AAA prend une décision d'autorisation et renvoie le résultat via un message à l'équipement de service. L'équipement de service accordera ou non le service à l'utilisateur en analysant le contenu du message contenant le résultat de la décision.

Des exemples de tels systèmes se trouvent dans le monde des réseaux avec des systèmes utilisant le protocole RADIUS [RADIUS 2011].

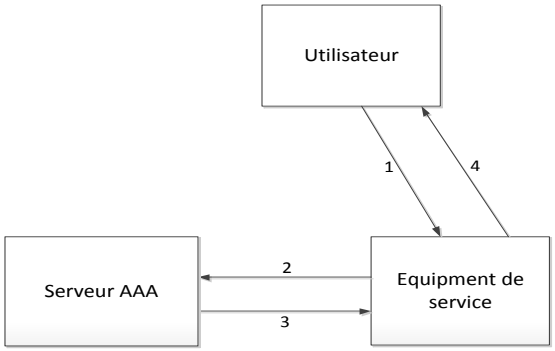

Figure 5 Le modèle Pull

1.3.1.2 Un exemple de système AAA : RADIUS

Les architectures AAA ont été mises en place afin d'offrir en particulier des accès à des ressources à distance. Le Remote Authentication Dial-In User Service (RADIUS) (Figure 6) [RADIUS 2011, le RFC 2865 en 2000 ayant rendu obsolète le RFC 2138 et 2058 datant de 1997], développé par la société Livingston Enterprises, est un protocole client/serveur basé sur architecture AAA et le modèle de déploiement « Pull » en permettant de centraliser les données d'authentification. L'architecture de RADIUS repose principalement sur un serveur RADIUS relié à une base d'identification (base de données, annuaire, etc.), un serveur d'accès réseaux (Network Access Server - NAS) qui est le client RADIUS, et un utilisateur final. L'ensemble des transactions entre le client RADIUS et le serveur RADIUS est chiffré et authentifié grâce à un secret partagé. Le serveur RADIUS prend les décisions d'autorisation (accepter, refuser ou challenge) en évaluant une requête d'un client RADIUS avec une liste de contrôle d'accès (ACL). Ensuite, le NAS applique la décision prise par le serveur au client.

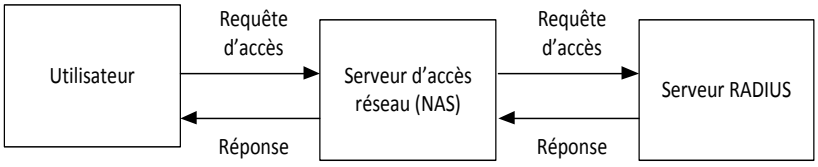

Figure 6 Architecture d'autorisation centralisée

L'architecture AAA de RADIUS a été ensuite étendue au sein du protocole DIAMETER [DIAMETER 2011] afin de proposer la gestion d'accès distants établis en s'appuyant sur de nouvelles technologies d'accès dont les accès sans fil, DSL, IP mobile. Les serveurs d'accès réseau (NAS) ont gagné en complexité et en nombre de demandes. L'architecture centralisée de RADIUS a évolué pour prendre en charge des services distribués (services web, applications et bases de données...) avec la nécessité de mettre en place des mécanismes de redirection et d'agents relais. Un exemple d'architecture d'autorisation pour les agents « proxy » est illustré dans (Figure 7).

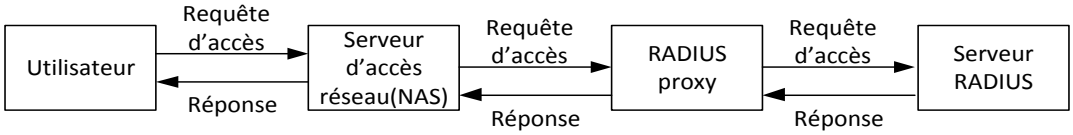

Figure 7 Architecture de services distribués

RADIUS a initialement été défini afin de permettre aux fournisseurs d'accès à Internet d'authentifier les utilisateurs distants à partir de multiples serveurs mais d'une seule base d'utilisateurs, évitant ainsi la duplication des attributs d'identité. C'est pourquoi, l'autorisation dans RADIUS est limitée à un nombre réduit d'attributs d'identité des utilisateurs (principalement le couple nom login/mot de passe). RADIUS ne peut donc s'adapter à d'autres situations plus complexes où les politiques d'autorisation intègrent pour contrôler l'accès aux différents applications/services des éléments variés tels que les rôles, les équipes, le temps, etc. Pour cela, les architectures d'autorisation à base de politiques ont été développées, où des politiques sont utilisées à la place des ACLs. La politique peut contenir des informations diverses associées à des exigences de sécurité autres que des informations sur les identités des utilisateurs et donc permet de spécifier des règles plus fines que de simples ACLs.

1.3.2 Les architectures d'autorisation à base de politiques

Les architectures à base de politiques sont une autre solution pour contourner la difficulté inhérente aux solutions qui consistaient entre autres à construire des ACLs. Tout comme les architectures AAA décrites précédemment, cette architecture présente l'avantage d'externaliser le processus de décisions de contrôle d'accès en dehors de l'application gérée. De plus, les langages de politique ont une capacité d'expressivité plus importante que les ACLs. Cette architecture rend la gestion plus dynamique et évolutive, car une autorisation est explicitement décrite par deux fonctionnalités complémentaires :

1. Une fonction de prise de décisions d'autorisation réalisée après consultation des politiques et,
2. Une fonction d'application des décisions.

Ces deux actions sont accomplies par deux entités distinctes (Figure 8) nommées respectivement PDP (Policy Decision Point) et PEP (Policy Enforcement Point) :

- Un PDP est une entité logique qui prend des décisions d'autorisation en considérant les informations suivantes [RFC2906 2000] :
  - Un contexte de requête qui inclut des informations sur l'entité qui demande l'accès, l'opération à effectuer (consultation, modification, etc.) et la ressource à protéger,
  - Et la politique d'autorisation qui définit les autorisations.
- Un PEP est une entité logique qui applique la décision d'autorisation prise par le PDP. C'est le PEP qui réalise techniquement l'attribution ou le refus d'une demande d'accès à une ressource.

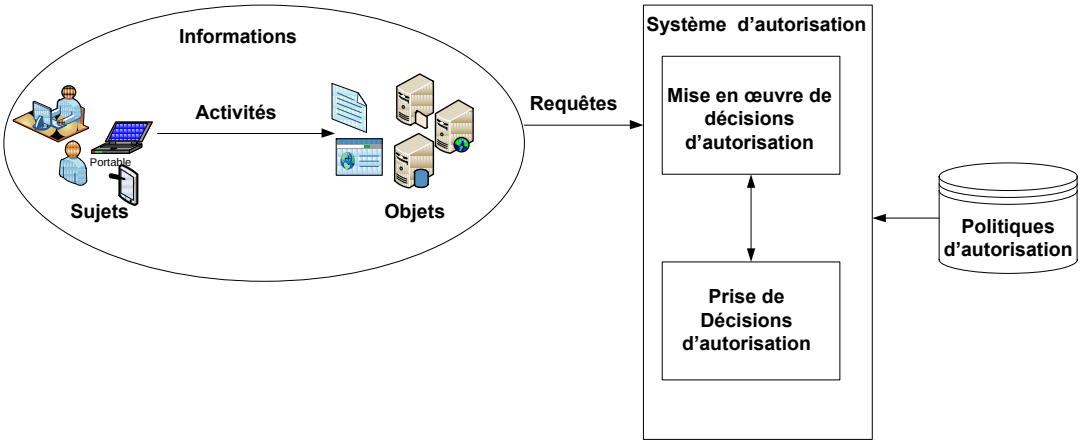

Figure 8 Système d'autorisation

Les interactions dans une architecture à base de politiques entre PDPs et PEPs peuvent suivre l'un des modèles de déploiement des architectures AAA (Agent, Pull, Push) décrits ci-dessus. Les fonctions de prise de décisions et de mise en œuvre d'un système d'autorisation à base de politiques peuvent être implantées au sein d'une même application ou au sein de composants séparés et distants.

Si l'externalisation de la fonction de décision est un élément important, elle n'implique pas forcément que le système d'autorisation puisse répondre dans le temps à l'évolution des besoins et des technologies (téléphonie mobile, internet fixe et mobile, informatique ubiquitaire, cloud computing, etc.). Ce type de système doit reposer sur un langage de politique flexible et évolutif. La solution des ACLs longtemps pertinente est de plus en plus dépassée aujourd'hui. Les langages de politique d'autorisation proposés aujourd'hui doivent donc prendre en compte cette évolution et anticiper des besoins futurs. Une analyse plus détaillée de ce problème de flexibilité et d'évolutivité est présentée dans le chapitre 3.

1.4 Conclusion

Dans ce chapitre, nous avons inventorié les changements majeurs qui ont marqué l'organisation des entreprises au cours de ces dernières années. Les besoins actuels en terme d'usage des technologies de l'information et de la communication rendent plus aigus les questionnements en matière de sécurisation des systèmes d'information. Aujourd'hui, la prise de conscience en matière de sécurité devrait amener toute entreprise à décrire les axes à suivre dans un document de contrôle général appelé politique de sécurité, recueillant la nature des objectifs de sécurité à garantir.

En regard des systèmes d'information, la mise en œuvre de la politique de sécurité doit permettre de répondre à l'intégrité, la disponibilité, la confidentialité, et la traçabilité, qui sont les points pivots des objectifs de sécurité. Le concept de gestion des identités et des accès est un moyen de protéger les systèmes d'information des organisations en garantissant les objectifs de sécurité. Dans le concept de gestion des accès le contrôle d'accès est l'un des moyens de mis en œuvre pour sécuriser les ressources (matérielles et logicielles) qu'elles soient ou non sensibles. Il recouvre sous une forme générale tout ou partie de mécanismes tels que la traçabilité, l'autorisation, l'authentification… Au cours de cette thèse, nous avons porté un intérêt particulier aux systèmes d'autorisation, car ils doivent répondre aux défis engendrés par l'exploitation des technologies de l'information et de la communication et en particulier à leur constante évolution.

Chapitre2. Evolution des modèles formels de contrôle d'accès

2.1 Introduction

En général, la sécurité des systèmes dépend des moyens mis en œuvre pour garantir les objectifs ou propriétés de sécurité fondamentales que sont la confidentialité, l'intégrité, la disponibilité. Le contrôle d'accès dans la gestion des identités et des accès (IAM) est mis en œuvre pour garantir ces propriétés fondamentales décrites lors de la définition puis de l'implémentation d'une politique de sécurité. Sous une forme générale, on les regroupe sous le vocable de politique de contrôle d'accès ou d'autorisation.

En outre, l'utilisation d'un modèle formel permet d'exprimer à l'aide de méthodes formelles un problème de manière non ambigüe afin de pouvoir l'évaluer, en particulier mathématiquement, et de trouver des solutions répondant à ce problème. Dans le cadre du contrôle d'accès, la formalisation a pour but de s'assurer que les propriétés ou objectifs de sécurité attendus sont bien atteints. La formalisation amène à construire des modèles qui décrivent de manière abstraite un système réel et offrent une analyse de son comportement sur la base d'un ensemble d'états dans lequel le système peut entrer [Laborde 2005]. DAC, MAC, RBAC et des dérivés tels LRBAC, Or-BAC etc. font partie de l'ensemble des modèles formels traitant du contrôle d'accès.

A l'origine les modèles de contrôle d'accès ont essayé de répondre à des besoins ou problèmes inhérents à la mise en œuvre des systèmes d'exploitation et des systèmes de gestion de bases de données. L'approche retenue a consisté à considérer qu'un système d'exploitation peut être appréhendé comme un ensemble d'entités appelées sujets (processus, utilisateurs, etc.) qui effectuent des actions sur des objets (ressources, fichiers, sockets etc.). Une opération effectuée par un sujet sur un objet peut alors être représentée par un triplé (sujet, objet, type d'accès). En adoptant cette représentation d'un système d'exploitation, une politique de contrôle d'accès classique peut alors être élaborée à partir des entités fondamentales suivantes :

- Un ensemble de sujets (S) : Un sujet peut être un processus, un utilisateur, une application etc.
- Un ensemble d'objets (O) : Un objet est un conteneur d'informations, sur lequel un sujet peut effectuer des actions (exemples : fichiers, sockets de communication, périphériques matériels, etc.)
- Un ensemble d'actions (A) : représente l'action à traiter par le sujet sur l'objet. (exemples : lecture, écriture, exécution d'un fichier, envoi de signaux ou de messages inter-processus etc.)

Un modèle formel de contrôle d'accès tente de modéliser de manière formelle les politiques de contrôle d'accès. Il tente par conséquent de décrire de manière non ambiguë les différentes entités répondant à des exigences d'une politique de sécurité au sein de l'organisation.

En s'appuyant sur les entités (S, O, A), deux approches de modèles de contrôle d'accès dits classiques se sont rapidement dégagées. Les modèles de contrôle d'accès discrétionnaire (DAC) ont pour caractéristique principale que les utilisateurs (**sujets**) peuvent être propriétaires des données et par conséquent attribuer les permissions sur les objets qu'ils possèdent. C'est le type de mécanisme mis en œuvre dans les systèmes d'exploitation traditionnels tels UNIX. Le modèle de contrôle d'accès obligatoire (MAC) est utilisé lorsque la politique de sécurité impose que les décisions de protection ne soient plus confiées au propriétaire des objets concernés. Ainsi, différemment de DAC, le MAC impose une entité supérieure (typiquement un administrateur externe de la politique de sécurité d'où mandataire) dont la tâche est de gérer l'attribution des permissions sur les fichiers et processus du

système. Ce système consolide la gestion de la sécurité en appliquant le principe de la séparation des pouvoirs.

Au fil du temps, ces modèles sont devenus insuffisants pour prendre en compte tout un ensemble des caractéristiques provenant du passage à l'échelle et de l'évolution des besoins utilisateurs. Ces limites ont conduit à la proposition du modèle (Role Based Access Control – RBAC) pour s'appuyer sur la notion de rôles dans les règles de politiques. Bien que ce modèle ait séduit les grands comptes, par son approche d'administration des droits, il s'avère aujourd'hui insuffisant pour prendre en considération la construction de règles répondant par exemple à l'utilisation de dispositifs ubiquitaires. Ainsi, plusieurs modèles de contrôle d'accès tentent de combler les manques constatés (LRBAC, Geo-RBAC, CRBAC, etc). L'activité autour des modèles de contrôle d'accès montre que toute proposition de modèle est faite pour répondre à une contrainte ou spécificité particulière d'une organisation ou d'un contexte précis qui n'a pas être traité par les modèles proposés antérieurement.

Dans les paragraphes suivants nous présentons sous forme synthétique, les apports des modèles de contrôle d'accès proposés, couvrant à notre sens les exigences des organisations en suivant l'axe d'évolution technologique.

2.2 Modèles de contrôle d'accès classiques

Comme nous venons de le préciser en introduction, les modèles de contrôle d'accès discrétionnaire (DAC) et le modèle de contrôle d'accès mandataire (MAC) se sont rapidement imposés pour faciliter la gestion des accès aux ressources au sein des systèmes d'exploitation et des systèmes de gestion de bases de données. Ces deux modèles reposent sur l'exploitation du triplet sujet (S), objet (O), Action (A)) pour représenter les politiques de contrôle d'accès.

2.2.1 Modèle de contrôle d'accès discrétionnaire

TCSEC (Trusted Computer System Evaluation Criteria) présente le contrôle d'accès discrétionnaire comme : "*un moyen de restriction d'accès aux objets basé sur l'identité des sujets et/ou groupes auxquels ils appartiennent. Les contrôles sont dits discrétionnaires dans le sens où le sujet est capable de transférer les permissions d'accès à d'autres sujets*" (La transmission des droits est exercée à la discrétion du sujet) [NCSC 1987].

Le triplet <*sujet, objet, action*> est utilisé pour construire des règles d'autorisation (ou d'interdiction). Le contrôle d'accès discrétionnaire (DAC) a été principalement implanté au sein des systèmes d'exploitation (Microsoft Windows, Solaris, Linux, FreeBSD). Dans ces systèmes les règles d'autorisation sont exprimées sous forme positive ou négative. Une règle d'autorisation positive spécifie l'ensemble des *sujets* qui peuvent accéder aux *objets*. Une règle d'autorisation négative spécifie l'ensemble des *sujets* qui ne peuvent pas accéder aux *objets*. La construction d'une politique de contrôle d'accès peut être qualifiée de fermée ou d'ouverte. Dans une politique fermée, l'accès aux *objets* est par défaut interdit. Les règles construites donnent des droits au fur et à mesure aux *sujets* sur les *objets*. Dans une politique ouverte, l'accès aux *objets* est par défaut autorisé. Les règles construites limitent au fur et à mesure l'accès aux *objets.*

Le modèle DAC laisse au sujet propriétaire d'un objet toute latitude pour décider quels autres sujets peuvent exercer des actions sur l'objet. Ajouté à cela, les langages de spécifications de politiques DAC donnent la possibilité de constituer et de gérer des « sujet_groupe ». On constitue au besoin, des regroupements de sujets présentant des caractéristiques communes. Cela permet alors d'éviter la présence dans une base de politiques, d'une multiplicité de règles quasiment identiques. On regroupe au sein d'une règle unique un comportement qui devrait être défini nominativement pour chaque sujet. De plus les langages de politique DAC peuvent être plus ou moins performants en

combinant ou non des règles positives et/ou des règles négatives. Ils offrent alors la possibilité aux sujets une flexibilité dans l'expression des besoins d'accès aux ressources.

L'implantation de ce modèle a donné lieu à la constitution de matrices d'accès initialement introduite en 1971 par Lampson [Lampson 1971] qui a été généralisée en 1976 par Harison, Ruzzo et Ullman (HRU) [Harrison et al. 1976]. Dans ce dernier, l'état du système est défini par un triplé (*S*, *O*, *M*) où *S* représente l'ensemble des sujets (e.g. utilisateur, processus etc.) pouvant exercer un ensemble d'actions. *O* représente l'ensemble des objets (e.g. fichier, table, classe, programme etc.). Enfin, *M* représente la matrice d'accès, où les lignes correspondent aux sujets et les colonnes correspondent aux objets (Tableau 1).

Sujets \ Objets	Fichier	Table
Alice	Lire Ecrire Exécuter	Exécuter
Bob	Lire Ecrire	Exécuter Lire

Tableau 1 Exemple d'une matrice d'accès

Les droits correspondent généralement à des actions élémentaires comme *lire, écrire, exécuter ou posséder* (mais ne sont pas limités à ces derniers). En effet, si de nouveaux objets, de nouveaux sujets ou de nouvelles actions sont ajoutés dans le système, il devient nécessaire d'enregistrer toutes les permissions accordées pour ces nouvelles entités [Nasser 2006].

Il existe en pratique deux approches pour implémenter la matrice d'accès :

- Par une liste de contrôle d'accès (ou ACL pour Access Control List) : la matrice est stockée par colonne. A chaque objet est associée une liste de règles indiquant pour chaque utilisateur les actions pouvant être exercées par ce dernier sur cet objet.
- Par une liste de capacité (ou capability) : la matrice est stockée par ligne. A chaque utilisateur correspond une liste, appelée liste de capacité, indiquant pour chaque objet les actions que l'utilisateur est en droit d'effectuer sur cet objet.

Cependant, le modèle DAC limite l'accès aux objets uniquement en se basant sur un élément correspondant le plus souvent à l'identité de l'utilisateur, ou un élément générique pour désigner une ressource commune. C'est pour cela que le modèle DAC est également appelé, IBAC (Identity Based Access Control). Le problème principal de cette approche de modélisation est le contrôle de la propagation de droits dû à la capacité de possession qui a amené des recherches sur le problème de protection (safety problem). Ce problème consiste à analyser s'il existe une séquence d'actions qui amène à un droit à se retrouver dans une case de la matrice d'accès où il ne devrait pas être. De plus les recherches effectuées sur ce sujet ont montré que le problème de protection est indécidable dans le cas général [Harrison et al. 1976].

Le modèle de contrôle d'accès obligatoire ou MAC (plus précisément le modèle multi-niveaux) en éliminant le droit de possession résout cette problématique.

2.2.2 Modèle de contrôle d'accès obligatoire

Afin d'apporter une solution aux problèmes de fuites d'information des modèles de contrôle d'accès discrétionnaires, les modèles obligatoires (Mandatory Access Control) viennent fixer des règles incontournables destinées à forcer le respect de la politique de sécurité de l'organisation. Ainsi, le modèle multi-niveaux affecte aux sujets et aux objets des niveaux de sécurité qui sont non modifiables par les utilisateurs et, par conséquent, limite leurs accès. Par conséquent, les droits de type *posséder* n'existent pas. Le premier modèle, appelé modèle de Bell et La Pudula [Bell et La Padula 1976], a été développé pour le département de la défense américain et vise, plus particulièrement, à assurer la confidentialité des données dans le contexte de l'utilisation partagée de mainframes. Le deuxième modèle, appelé modèle de Biba [Biba 1977] qui prend en compte les exigences commerciales s'intéresse plutôt à l'intégrité des informations.

Prenons le modèle multi-niveaux de Bell et La Padula qui vise la confidentialité. Ce modèle est basé sur la classification des sujets et des objets. Le principe consiste à attribuer une classe d'accès à chaque sujet et à chaque objet. Généralement, une classe d'accès est constituée de deux composants : un niveau de sécurité et un ensemble de catégories. Le niveau de sécurité est un élément d'un ensemble hiérarchique ordonné, tel que Top Secret > Secret > Confidential > Unclassified. L'ensemble des catégories est un sous-ensemble d'un ensemble non ordonné, dont les éléments représentent soit une compétence, une région, un département. Pour éviter les fuites d'information, l'accès aux objets doit obligatoirement respecter les deux principes fondamentaux :

- No read up : un sujet est autorisé à lire un objet donné uniquement si sa classe d'accès domine la classe d'accès de l'objet.
- No write down : un sujet est autorisé à écrire dans un objet donné uniquement si la classe d'accès de l'objet domine sa classe d'accès.

D'après la politique de contrôle d'accès, un fichier contenant des informations sensibles ne peut être accédé que par un utilisateur exécutant une application d'un niveau de sécurité *secret*. Si un utilisateur invoque l'application avec un niveau de sécurité *unclassified*, l'opération de lecture de ce fichier sera bloquée (Le principe de No-read up). Alors, en respectant les deux principes, le modèle MAC résout le problème de fuite d'information des modèles DAC. Il est quand même un modèle très rigide, car il ne permet pas de gérer les exceptions entre les différents niveaux de sécurité. Par exemple, un utilisateur de niveau de sécurité secret ne peut pas accéder, pour des raisons exceptionnelles, aux données de niveau de sécurité top secret.

2.2.3 Synthèse

Les modèles de contrôle d'accès classiques définissent une relation directe entre les sujets et les objets. Ces modèles classiques sont développés pour résoudre des problèmes de sécurité traditionnels comme la confidentialité et l'intégrité. Mais, ils ont trouvé leurs limites : trop rigides, insuffisamment sûrs ou difficiles d'administration. A l'usage, une limite importante de ces modèles est apparue : la politique d'autorisation devient rapidement complexe à exprimer et administrer. Il est en effet nécessaire d'énumérer les autorisations pour chaque sujet, action ou objet. En particulier, lorsqu'un nouveau sujet ou objet est créé, il est nécessaire de mettre à jour la politique d'autorisation pour définir les nouvelles permissions associées à ce sujet ou objet. Si ces modèles permettent de construire des «sujet_groupe », ils n'indiquent pas comment constituer ou gérer ces groupes.

Pour cela, d'autres modèles qui brisent cette relation directe entre sujet/objet en y insérant de nouveaux concepts : les tâches, les rôles, les équipes etc. ont été proposés par la suite. Dans la section suivante, nous présentons les modèles à base de rôles. Ils sont plus populaires et les extensions nombreuses prenant en compte de nouvelles notions par rapport à l'évolution technologique.

2.3 Modèles de contrôle d'accès à base de rôle

Le modèle de contrôle d'accès à base de rôle (RBAC) a été proposé pour présenter une nouvelle organisation des droits centrée sur le concept de rôle. Un rôle représente une fonction dans le cadre d'une organisation. Utiliser le rôle comme intermédiaire entre les sujets et les permissions facilite et simplifie les tâches d'administration en diminuant le nombre d'affectations à manipuler. Ce modèle est largement adopté par les entreprises et les industriels [Thion 2008] et a été appliqué dans de grandes structures. Les logiciels commerciaux Trusted Solaris, Windows Authorization Manager, Oracle 9 et Sybase Adaptive Server ont mis en œuvre tout ou partie des principes des modèles RBAC. La spécification de modèle RBAC [Ferraiolo et al. 2007] [NIST 2011] comprend les sous-modèles suivants (Figure 9):

- Le modèle $RBAC_0$ ou « the flat model », qui présente les concepts et relations de base c.à.d. le noyau du modèle,
- Le modèle $RBAC_1$ ou « the hierarchical model », qui reprend le modèle $RBAC_0$ et introduit la notion de hiérarchie entre rôles,
- Le modèle $RBAC_2$ ou « the constrained model », qui reprend le modèle $RBAC_0$ et introduit la notion de contrainte,
- Le modèle $RBAC_3$ ou « the symmetric model », qui reprend les modèles $RBAC_1$ et $RBAC_2$ et prend en compte les interactions entre contraintes et hiérarchie.

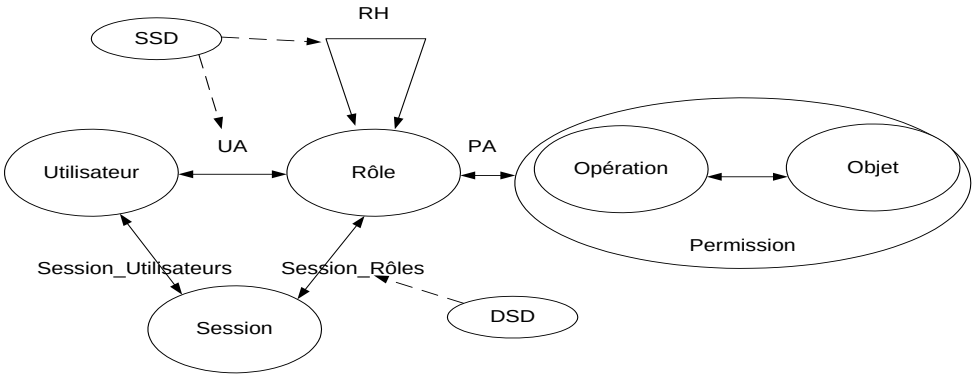

Figure 9 Modèle RBAC

Le modèle $RBAC_0$ définit les concepts ou entités de base pour spécifier les politiques de contrôle accès RBAC. Le modèle $RBAC_1$ ajoute la possibilité de construire une hiérarchie de rôles, i.e. une approche pour structurer cette notion de rôle. Les versions 0 et 1 du modèle RBAC introduisent les ensembles suivants :

- Utilisateur : l'ensemble des utilisateurs, où un utilisateur est une entité active, humaine ou logicielle
- Rôle : l'ensemble des rôles, où un rôle est une fonction de travail dans le cadre d'une organisation liée à une autorité et des responsabilités
- Permission : l'ensemble des autorisations afin d'effectuer des opérations sur un ou plusieurs objets protégés
- Opération : l'ensemble des opérations
- Objet : l'ensemble des objets ou des ressources
- Session : une correspondance entre un utilisateur et un ensemble de rôles autorisés
- UA $\subseteq$ Utilisateur $\times$ Rôle : permet d'affecter des rôles aux utilisateurs

- PA $\subseteq$ Permission $\times$ Rôle : permet d'affecter des permissions aux rôles
- RH $\subseteq$ Rôle $\times$ Rôle : définit un ordre partiel sur l'ensemble Rôle, appelée héritage. Elle est aussi écrite par $\geq$ tel que $rôle_1 \geq rôle_2$ implique que les permissions de $rôle_2$ sont aussi des permissions de $rôle_1$
- Session_Utilisateurs : Session $\rightarrow$ Utilisateur, permet d'établir l'utilisateur d'une session,
- Session_Rôles : permet d'établir l'ensemble des rôles associés à une session.

Les deux modèles $RBAC_2$ et $RBAC_3$ apportent la possibilité de gérer les éventuels conflits entre rôles en ajoutant des contraintes pour exprimer la séparation de tâches et l'exclusion mutuelle entre rôles. Ainsi, pour interdire à un utilisateur d'être affecté à deux rôles qui sont en conflit, RBAC propose deux types de contraintes: les séparations statiques (Static Separation of Duties ou SSD) et les séparations dynamiques (Dynamic Separation of Duties ou DSD). La SSD interdit l'affectation d'un utilisateur à deux rôles en conflit, et empêche qu'une hiérarchie de rôles amène un utilisateur à posséder les permissions de deux rôles en conflit. La DSD évite qu'un utilisateur possède deux rôles en conflit en même temps dans une même session.

Il faut bien différencier le concept de rôle de celui d'un groupe. Un rôle est une fonction dans une organisation. Un groupe est un ensemble d'entités. Par conséquent, l'ensemble des sujets assignés à un même rôle forme un groupe [Laborde 2005]. La hiérarchie de rôle (RH) est un moyen naturel pour structurer les rôles afin de refléter les lignes d'une organisation en considérant l'autorité et la responsabilité des utilisateurs. Il suffit d'affecter le(s) rôle(s) que l'utilisateur assurera au sein de l'organisation pour lui accorder les droits associés. Sa réaffectation à d'autres fonctions est très simple, il suffit de désactiver son ancien rôle et de lui affecter au(x) nouveau(x) rôle(s). L'organisation des rôles en hiérarchie facilite encore plus l'administration des contrôles d'accès.

Le concept de rôle sépare la relation sujet/objet. Ainsi, il facilite l'administration des politiques de contrôle d'accès. Par exemple dans une organisation où un groupe de 100 utilisateurs travaillant sur un même projet doivent avoir l'accès à un fichier, il est plus simple de créer un rôle projet et d'y associer le droit de lire le même fichier qui se trouve dans un serveur. Dans un modèle RBAC, on spécifie une permission (+lire, fichier) et un rôle *projet* qui représente les membres participant au projet. Lorsque cette permission est accordée aux membres du projet, ceci est décrit en assignant le rôle *projet* aux membres de projets par relation UA et en assignant la permission (+lire, fichier) au rôle *projet* dans la relation PA. Au moment où le projet se termine, pour enlever la permission accordée (+lire, objet), il suffit de supprimer l'élément (*projet*, (+lire, objet)) de la relation PA. Aussi, le modèle RBAC sépare les notions d'assignation et d'activation grâce aux contraintes.

L'inconvénient de RBAC réside dans la difficulté de gérer des règles dépendant du contexte de l'utilisateur, par exemple, de types « les étudiants ont le droit d'accéder uniquement à leurs données personnelles » ou « Seuls les étudiants qui se trouvent dans le bâtiment A ont le droit d'accéder au serveur de leur université » [Thion 2008]. Pour le premier exemple, une solution envisageable est de créer pour chaque étudiant un rôle privé. Théoriquement c'est une solution mais en pratique cela n'est pas faisable car s'il existe un très grand nombre d'étudiants, cela fait perdre à RBAC sa simplicité d'administration. En outre, pour le deuxième exemple un nouvel élément doit être introduit dans le modèle correspondant à la « location physique de l'utilisateur ». Ainsi, un nombre important de nouveaux modèles qui étendent le modèle RBAC ont été proposés afin de répondre à des besoins de sécurité non définis dans le modèle RBAC.

2.4 Modèles de contrôle d'accès à base des tâches

En parallèle des travaux originaux sur RBAC, le modèle TBAC (Task Based Access Control) [Thomas et Sandhu 1993] a été conçu afin d'activer une permission par rapport aux tâches effectuées

par l'utilisateur. L'idée essentielle de ce modèle consiste à ajouter la notion de tâche dans des règles d'autorisation. Cela permet de définir les permissions qu'un sujet peut activer selon la tâche qui est en cours. Chaque étape d'autorisation correspond à certaines activités ou tâches dans le contexte plus large d'un workflow de l'organisation. Le modèle TBAC fut le premier modèle à introduire le concept de tâche. TBAC va au-delà des modèles IBAC où les actions correspondent généralement à des commandes élémentaires (comme la lecture du contenu d'un objet ou l'écriture dans un objet) pour structurer et contrôler la réalisation d'actions composites, appelées tâches ou activités. Il ajoute une étape d'autorisation (authorization step) qui permet de définir les permissions (enabled permissions) qu'un sujet (executor-trustee) peut activer selon la tâche qui est en cours. Ainsi, TBAC offre une approche pour différencier l'affectation et l'activation des permissions par rapport à des tâches données aux utilisateurs au sein de l'organisation.

La notion de tâche permet de contrôler les activités exercées par les utilisateurs d'un système d'information au sein de l'organisation. TBAC peut parfaitement être adapté et intégrer la notion de rôle. C'est dans cet esprit que le modèle TR-BAC (Task and Rôle BAC [Bertino et al. 2001]) a été défini. Dans ce cas, les droits sont activés en fonction d'un rôle et portent sur la réalisation des tâches. De plus, TBAC ou TR-BAC présente l'inconvénient de ne pas prendre en compte des contraintes sur les horaires ou périodes d'accès pendant lesquels les utilisateurs sont en charge de la réalisation de leurs activités [Joshi et al. 2005]. Le manque constaté est couvert par d'autres modèles formels de contrôle d'accès.

2.5 Modèles de contrôle d'accès dérivés de RBAC

Le modèle RBAC a été largement adopté par l'industrie et par la communauté de recherche. Il a déclenché un renouveau des modèles de contrôle d'accès et plusieurs propositions ont été faites pour ajouter de nouveaux concepts ou notions au modèle RBAC de base : par exemple le temps, la localisation, le contexte spatial, la position géographique de l'utilisateur etc. Nous classons dans cette section les modèles qui couvrent la plupart de ces nouvelles notions ou concepts dérivés de RBAC :

- La notion d'équipes introduite par le modèle TMAC [Thomas et Sandhu 1997]
- La notion de localisation et d'information spatiale dans des applications mobiles introduites par le modèle LRBAC [Zhang et al. 2006]
- La notion d'information spatiale et plus particulièrement la position physique de l'utilisateur dans des dispositifs comme GPS introduit par le modèle Geo-RBAC [Bertino et al. 2005]
- La notion de temps et des contraintes de temps dans les systèmes de « workflow » introduite par le modèle GTRBAC [Joshi et al. 2005]
- La notion de contexte dans un environnement ubiquitaire (pervasif) introduit dans le modèle CRBAC [Park et al. 2006]

2.5.1 Modèles de contrôle d'accès à base d'équipes

La notion d'équipe a été proposée dans le modèle TMAC [Thomas et Sandhu 1997] (TeaM-based Access Control). Les permissions sont associées aux rôles ainsi qu'aux équipes. La notion d'équipe a été introduite pour représenter des aspects transversaux des rôles qui ne sont pas directement exprimables dans les modèles RBAC [Thion 2008]. Dans TMAC, l'objectif est d'accorder à chaque utilisateur membre d'une équipe des permissions accordées aux autres membres de l'équipe qui sont actifs. On distingue deux types d'informations [Thomas et Sandhu 1997] :

- le contexte utilisateur, i.e., les utilisateurs qui font partie de l'équipe à tout moment

- le contexte objet, i.e., l'ensemble des instances des objets dont l'équipe a besoin pour accomplir sa tâche

Il faut noter qu'ici les relations de ce modèle sont dynamiques, car les permissions sont accordées par rapport au contexte des membres d'une équipe dans des sessions actives. Ce modèle prend seulement les requêtes actives dans une session. Ce modèle offre les avantages d'une administration simplifiée d'une modélisation RBAC mais aussi, il fournit un contrôle plus fin sur l'activation d'autorisation d'utilisateurs individuels et des objets.

2.5.2 Modèles de contrôle d'accès prenant en compte la localisation

2.5.2.1 Le modèle LRBAC

LRBAC [Zhang et al. 2006] étend le modèle RBAC pour que le contrôle d'accès puisse être établi en prenant en compte les informations de localisation. Un tel modèle a été proposé pour autoriser ou interdire l'accès lorsque les systèmes sont dans ou hors d'une zone d'opération définie. Ce modèle, proposé dans le cadre de la prolifération d'équipements mobiles, utilise la localisation logique des utilisateurs et/ou des systèmes comme paramètre contextuel. La localisation logique exprime les frontières ou les secteurs (i.e. départements, bureaux individuels, secteur public etc.) dans l'organisation. Ainsi, les domaines de localisation logique définissent les limites de l'espace dans lequel les rôles peuvent être assumés par l'utilisateur. Comme RBAC, LRBAC comprend un noyau et un modèle hiérarchique (Figure 10). Le modèle de base comprend les éléments de base suivants :

- Utilisateurs, Rôles, Sessions, Permissions, Operations et Objets ont la même signification que dans RBAC,
- LDOM (Domaine de localisation logique) est l'ensemble des domaines de localisation logique,
- SR (Spatial Role) caractérise l'ensemble des rôles spatiaux. Un rôle spatial est un couple présenté par (r, ldom), où r définit le rôle comme dans RBAC et ldom est le nom du domaine de localisation logique. Un exemple de SR est (Manager, BâtimentA) : Manager est le nom de rôle ; BâtimentA est l'endroit logique où ce rôle est valide.

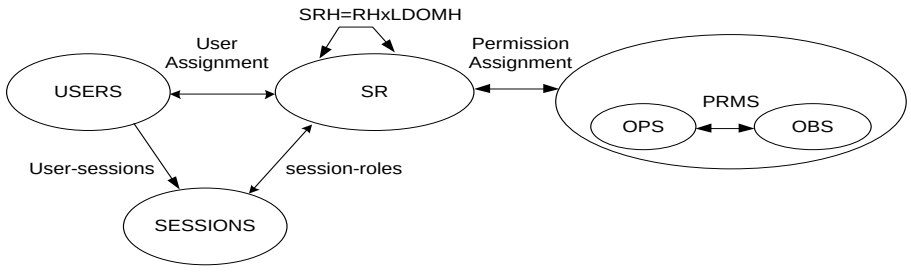

Figure 10 Le modèle LRBAC

Le modèle hiérarchique introduit une hiérarchie entre les rôles spatiaux (SR) qui se composent de deux hiérarchies : une pour les rôles de base et l'autre pour les domaines de localisation logique (ldom). La hiérarchie des rôles est définie comme dans le modèle RBAC. La hiérarchie des *ldom* est un ordre partiel $\supseteq$ où « ldom1$\supseteq$ldom2 » signifie que le domaine *ldom2* inclut le domaine *ldom1*. Ainsi, la hiérarchie entre les rôles spatiaux « SRH$\subseteq$ SR×SR » est un ordre partiel « $\geq$ », où par exemple « (r1, ldom1) $\geq$ (r2, ldom2) » signifie qui r1 hérite r2 c.à.d. r1$\geq$r2 et ldom1 $\supseteq$ ldom2.

2.5.2.2 Le modèle Geo-RBAC

Le modèle Geo-RBAC [Bertino et al. 2005], étend le modèle RBAC en définissant de nouveaux concepts spatiaux pour représenter la position des sujets et celles des objets. Ces nouveaux concepts sont utilisés pour limiter géographiquement l'utilisation des rôles. Le principe proposé dans Geo-RBAC est de comparer une position physique, supposée obtenue de façon fiable (par exemple la localisation GPS), à des positions logiques (exemples : route, ville, région) auxquelles sont associées des rôles géographiques [Thion 2008].

Il est composé comme RBAC de sous-modèles de base, hiérarchique, et contraint. Le modèle Geo-RBAC comprend tous les concepts de base de RBAC et ajoute des nouveaux concepts : la notion de rôle spatial, et de position réelle/logique. Ce modèle est très précis pour répondre à la nécessité de prendre en compte la localisation géographique, dans la construction d'une règle de politique de contrôle d'accès.

2.5.3 Modèles de contrôle d'accès à base de contexte temporel

GTRBAC [Joshi et al. 2005] étend le modèle RBAC afin d'exprimer un large éventail de contraintes temporelles. En particulier, le modèle permet d'exprimer le temps et des contraintes temporelles sur les rôles, l'affectation des rôles aux utilisateurs, et l'affectation des permissions aux rôles. Ce modèle répond aux besoins précis d'applications avec une contrainte temporelle forte, comme les systèmes intégrant des workflows où la notion de temps est importante. Ces systèmes sont utilisés par des organisations désirant spécifier des règles d'autorisation qui permettent ou interdisent l'accès à des ressources pendant un intervalle de temps donné.

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

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

Tout comme le modèle RBAC, GTRBAC propose de structurer les informations sous forme de hiérarchie [Thion 2008]. De plus, ce modèle inclut l'étude de conflits entre les éléments : rôles, utilisateurs, et contraintes temporelles. Ainsi, ce modèle répond à un contexte précis impliquant l'utilisation de la notion de « temps » dans les politiques de contrôle d'accès au sein de l'organisation.

2.5.4 Modèles de contrôle d'accès à base de contexte

Avec l'utilisation de l'informatique ubiquitaire (pervasive computing) au sein des organisations prenant en compte des caractéristiques par essence dynamiques, comme la position géographique d'un utilisateur, l'état des équipements ou l'horaire d'utilisation, il est devenu nécessaire d'intégrer les aspects contextuels évolués dans les modèles formels de contrôle d'accès. La prise en compte de la notion de contexte dans les politiques à base de rôles est un domaine d'investigation actif aujourd'hui. La définition du contexte en informatique est délicate, une définition communément admise est proposée par Dey [Dey 2001]: « *le contexte est l'ensemble de toutes les informations qui peuvent être utilisées pour caractériser la situation d'une entité. Une entité pouvant être un acteur, un lieu, ou un objet de l'environnement considéré comme utile à l'interaction entre un utilisateur et une application, y compris l'utilisateur et l'application eux mêmes* ».

Le modèle formel de contrôle d'accès CRBAC [Park et al. 2006] introduit en plus du « rôle » la notion de « contexte ». Le modèle de base comprend les éléments de base suivants :

- Utilisateurs, Rôles, Sessions, Permissions, Operations et Objets ont la même signification que dans RBAC,
- CR (context role): CR représente un ensemble de rôles contextuels. Le rôle contextuel est utilisé pour capturer des informations de contexte de sécurité pertinentes sur l'environnement pour une utilisation dans les politiques de contrôle d'accès. Le rôle contextuel peut être lié au temps, à la localisation, etc.
- C (contexte): représente l'ensemble des informations contextuelles du système. L'ensemble « C » capture toutes les informations de contexte qui sont utilisées pour définir le rôle contextuel «CR». Les informations peuvent être le temps, la localisation, la température etc.

Dans CRBAC, les rôles sont composés : rôles de sujets comme dans RBAC, et rôles contextuels pour capturer des informations de sécurité liées au contexte. Enfin, ce modèle fournit un contrôle d'accès basé sur le contexte pour les exigences des applications sensibles au contexte dans les systèmes ubiquitaires ou pervasifs.

2.5.5 Synthèse des extensions du modèle RBAC

Le principe général des modèles dérivés de RBAC est :

- d'organiser les droits de la façon la plus proche possible de la structure des organisations, afin de permettre aux administrateurs de manipuler les droits d'une façon plus intuitive
- de limiter le nombre d'affectations des permissions, pour éviter les erreurs en réduisant la taille des politiques
- d'exprimer de nouvelles règles et contraintes, qui permettront de traduire facilement les politiques exprimées en langage naturel
- de prendre en compte des notions autres que le rôle tel qu'il est introduit dans RBAC, pour spécifier des politiques de contrôle d'accès qui répondent à des besoins spécifiques

Ces divers modèles tentent de répondre à un contexte spécifique qui ne fait pas partie des éléments considérés dans le modèle RBAC. Ainsi, le contexte précis proposé dans TMAC est de prendre en compte dans politiques de contrôle d'accès les deux notions (équipe et rôle), afin de répondre aux exigences spécifiques de la collaboration entre membres d'équipes au sein de l'organisation. Pour répondre aux exigences de l'informatique mobile, la notion de localisation dans les politiques de contrôle d'accès devient primordiale. Par exemple, un utilisateur portant un terminal mobile change régulièrement de position et peut demander l'accès à certaines ressources de plusieurs sites. Ces sites peuvent avoir différents niveaux de fiabilité ou bien peuvent relever de différents domaines administratifs et engendrer une disponibilité des ressources variables d'un site à l'autre. Ainsi, dans les organisations où l'accès à des ressources sensibles ne peut être exercé que dans un périmètre donné, les services de localisation (servant de médiateur) doivent être mis en œuvre pour déterminer si un usager équipé d'un terminal mobile est ou non situé dans le périmètre défini.

Le modèle LRBAC a été proposé pour guider l'accès aux ressources en fonction de la présence des utilisateurs à l'intérieur ou l'extérieur de l'organisation. Cette notion est nécessaire dans des systèmes mobiles, dès l'instant où il est souhaitable de voir le contrôle d'accès à des ressources être restreint ou augmenté. Cependant, aujourd'hui la localisation ne saurait s'arrêter à la simple prise

en compte d'un bâtiment ou d'une zone dans un secteur particulier. Pour la construction de politiques de contrôles d'accès, il peut être important de s'appuyer sur la localisation réelle d'un équipement ou d'un usager. Aujourd'hui, nombre d'usagers exploitent le système GPS pour effectuer une opération de localisation (routes, points d'intérêt, flotte de véhicule etc.). Le modèle Geo-RBAC propose d'associer à une position réelle une représentation des objets et de décliner des rôles en fonction de la position spatiale d'un utilisateur. Comme les autres modèles il est spécifique et traite du contexte géographique. Il ne prend pas en compte des autorisations qui pourraient par ailleurs dépendre de contextes temporels (par exemple une permission pendant les heures de travail). La proposition du modèle GTRBAC est venue apporter cette dimension.

L'avènement des dispositifs ubiquitaires, pour lesquels les autorisations ne sont pas statiques mais dépendent de conditions de l'environnement, a amené un nouveau modèle de contrôle d'accès, qui permet d'activer dynamiquement les autorisations quand les conditions sont satisfaites. Dans ce cas, les autorisations sont contextuelles. Plusieurs propositions ont donné lieu à la prise en compte du contexte dans les modèles de contrôle d'accès pour spécifier des *autorisations contextuelles*. Le contexte peut être celui de la localisation, du temps, de la situation géographique... Le modèle CRBAC traite du « contexte » avec la notion de rôle contextuel pour capturer des informations de sécurité liées au contexte associé aux rôles des utilisateurs. Le modèle CA-RBAC proposé par [Kulkarni et Tripathi 2008] donne la possibilité de spécifier des permissions personnalisées, pour certains membres des rôles définis, en fonction de contraintes contextuelles aux liaisons dynamiques des objets avec des services spatiaux actifs.

Mais la notion de contexte reste délicate à au moins deux égards [Thion 2008] :

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

L'exemple donné entre CRBAC et CA-RBAC donne une idée de la complexité sous-jacente.

2.6 Modèle de contrôle d'accès à base d'organisation

Le modèle de contrôle d'accès Or-BAC (Organization Based Access Control) [OrBAC 2011] [Cuppens et Miège 2003] [Abou El Kalam et al. 2003] vise à résoudre certains problèmes rencontrés par les premiers modèles de contrôle d'accès des années 90 et à établir une politique de contrôle d'accès plus abstraite. Il s'intéresse, non seulement aux permissions, mais aussi aux interdictions, obligations et recommandations dans une politique de sécurité. Or-BAC prend le concept de rôle dans RBAC. En plus de ce concept, il ajoute des nouveaux concepts pour structurer les sujets, les objets et les actions.

Le concept central de ce modèle est la notion d'organisation comme son nom l'indique. Une organisation peut être un groupe structuré de sujets jouant des rôles déterminés. Ce peut être un hôpital, une clinique médicale, un service d'urgence… L'organisation représente l'ensemble des rôles, des activités, et des vues qui représentent les abstractions respectives des utilisateurs, des opérations et des objets par rapport à une organisation donnée. Par exemple, un utilisateur est lié à un ensemble de rôles pour une organisation donnée. Il peut être affecté à d'autres rôles pour une autre organisation.

Le fait d'introduire ce concept « organisation » comme un élément de base dans le modèle de contrôle d'accès permet de structurer les droits en rassemblant plusieurs notions comme le rôle de RBAC et l'équipe de TMAC. Ces derniers définissent des relations binaires entre l'utilisateur et le rôle dans RBAC, ou entre l'utilisateur et l'équipe dans TMAC. Or-BAC définit des relations ternaires entre les organisations, les sujets et les rôles : un sujet joue un rôle dans une organisation. Ce qui veut dire que l'utilisateur ayant plusieurs rôles peut activer soit tous les rôles soit un sous-ensemble de ses rôles, dans n'importe quelle équipe à laquelle il participe. Dans la pratique, même si un utilisateur possède plusieurs rôles, il n'a pas forcément le droit de les jouer dans toutes les équipes auxquelles il appartient [Abou El Kalam et al. 2003].

La Figure 11 montre les différentes notions du modèle Or-BAC [OrBAC 2011] [Abou El Kalam et al. 2003]. Ce schéma fait apparaitre les deux niveaux de politique Or-BAC (abstrait et concret) ainsi que les différentes relations existant entre les entités de ces deux niveaux.

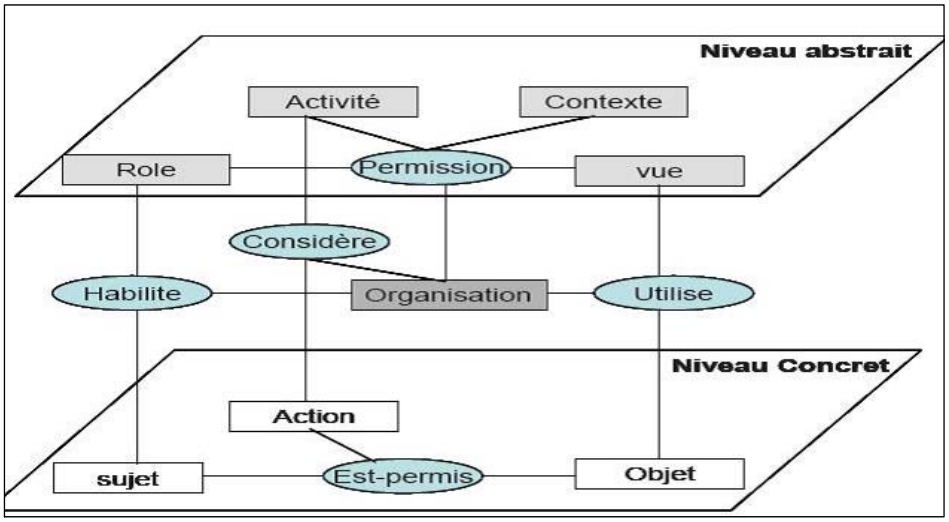

Figure 11 Le modèle Or-BAC

Or-BAC considère les ensembles suivants : Organisation (ensemble des organisations), Sujet (ensemble des sujets), Action (ensemble des actions), Objet (ensemble des objets), Rôle (ensemble des rôles), Activité (ensemble des activités), Vue (ensemble des vues), et Contexte (ensemble de contexte). Chaque entité peut avoir certains attributs. Ceci est représenté par des fonctions qui associent des entités avec la valeur de ces attributs. Par exemple si sujet s $\in$ Sujet, alors nom(s)=n et adresse(s)=m représentent les attributs *nom* et *adresse* utilisés pour désigner le nom et l'adresse du sujet s, où *n* est la valeur de *nom* et *m* est la valeur de l'*adresse*.

Nous décrivons les relations existant entre les entités du niveau concret et les entités de niveau abstrait d'Or-BAC :

- Relation entre sujets et les rôles : le sujet représente soit un utilisateur (Marie, Pierre etc.), soit une organisation (service d'urgence de l'hôpital de Purpan). Dans Or-BAC, l'entité rôle structure le lien entre les sujets et les organisations. La relation *Habilite (Purpan, Marie, médecin)* signifie « l'organisation Purpan habilite le sujet Marie dans le rôle médecin ».
- Relation entre les objets et les vues : les objets représentent des entités passives (comme les fichiers, les dossiers administratifs, les dossiers médicaux etc.). L'entité vue permet de structurer les objets (notion similaire aux rôles par rapport aux sujets) et elle caractérise la façon dont les objets sont utilisés dans l'organisation. Par conséquent, une relation, appelée *utilise* a été définie pour relier les organisations, les objets et les vues :

Si *org* est une organisation, *o* est un objet, et *v* est une vue alors *utilise (org, o, v)* signifie que *org* utilise l'objet *o* dans la vue *v*.

- Relation entre les actions et les activités : les actions représentent les opérations qui peuvent être effectuées par les sujets sur les objets. Dans ce modèle, l'entité Action représente les actions élémentaires lire, écrire etc. Les Activités correspondent aux actions qui ont le même objectif, par exemple consulter, modifier… Là encore, l'objectif est de permettre à des organisations de structurer différemment les mêmes activités. La relation *considère* entre les actions et les activités est utilisée pour associer les trois entités Organisation, Action et Activité.

La politique de contrôle d'accès est définie par les relations permission, obligation et interdiction. Ce sont des relations sur Organisation × Rôle × Activité × Vue × Contexte. La relation *Permission (o, r, a, v, c)* indique que dans une organisation *o,* un rôle *r* est autorisé de faire une activité *a* sur une vue *v* dans un contexte *c*. Les autorisations concrètes de type (sujet, action, objet) sont dérivées des relations définies ci-dessus. Tout comme le modèle RBAC, il peut exister une hiérarchie de rôles qui implique des héritages de permissions (Activités, Vues). Or-BAC permet aussi de spécifier une hiérarchie d'activités et de vues et même d'organisations [Cuppens et al. 2004].

En outre, pour prendre en compte dans les organisations, les nouvelles technologies support de l'informatique ubiquitaire et/ou mobile, Or-BAC définit l'entité contexte qui joue un rôle important dans la définition des politiques de contrôle d'accès spécifiques. Elle permet d'exprimer des politiques dépendantes du contexte ou de l'environnement (circonstances concrètes) dans lesquelles les organisations accordent les permissions de réaliser des activités sur des vues. Par exemple, les étudiants ont le droit d'accéder aux dossiers d'examens uniquement entre 8 :00 am et 9 :00 pm. Or-BAC comme d'autres modèles à base de contexte tente de définir de manière générale les différents types de « contexte » comme le contexte temporel, spatial, ou encore provisionnel etc.

Les contextes définis dans Or-BAC font partie de 5 catégories [OrBAC 2011] [Cuppens et Miège 2003] (Figure 12):

- Contexte spatial : la localisation logique et géographique de l'utilisateur peut être un paramètre à prendre en considération. Un client peut être autorisé à accéder au document de recrutement depuis un certain endroit dans le campus de l'organisation pour empêcher la fraude. Ceci peut être vérifié par des technologies de localisation comme le GPS.
- Contexte temporel: la notion de temps est considérée dans les politiques de contrôle d'accès comme dans le modèle GTRBAC expliqué plus haut.
- Contexte déclaré : dans certain cas, un sujet doit déclarer qu'un contexte est valide. Par exemple, un professeur peut déclarer un "contexte d'examen" et activer ainsi les permissions associées pour que les étudiants accèdent au document d'examen.
- Contexte prérequis : c'est le cas où la permission est donnée à un sujet seulement si certaines contraintes sont satisfaites. Par exemple, un étudiant ne peut voir accès qu'aux notes du cours multimédia.
- Contexte provisionnel : permet d'activer des permissions (interdictions ou obligations) en fonction des actions précédemment réalisées.

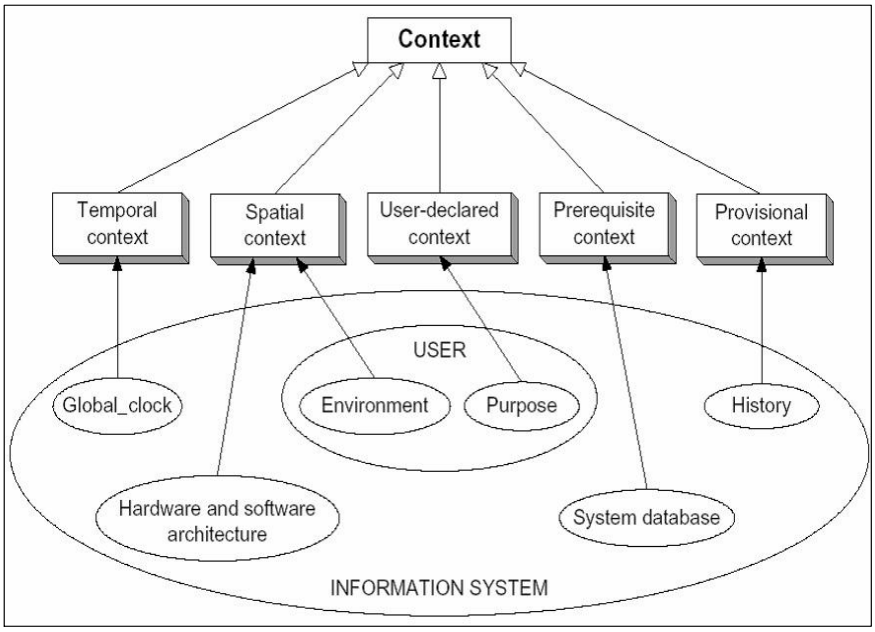

Figure 12 Contexte dans Or-BAC

2.6.1 Synthèse

En introduisant ces différentes entités, Or-BAC représente un modèle très générique qui reflète la structure des droits et relations au sein de l'organisation. Les entités « rôle », « activité », « vue » rendent la politique de contrôle d'accès flexible dans le sens où une modification des entités concrètes « sujet, action, objet » n'invalide pas la politique elle-même. Le modèle Or-BAC prend ainsi la notion de contexte pour spécifier des permissions ou autorisations dynamiques.

Cependant, l'extension O2O (Organization to Organization) d'OrBAC [Cuppens et al. 2006b], le modèle Multi-OrBAC [Abou El Kalam et Deswarte 2006], ou encore le travail de Nasser [Nasser 2006] ont été proposés pour étendre le modèle Or-BAC ; afin qu'il puisse répondre à une situation qui est la collaboration inter-organisationnelle. Ces travaux montrent la difficulté de réaliser un modèle de contrôle d'accès générique stable.

De plus, le modèle Or-BAC en apportant de plus en plus de notions (organisation, contexte, rôle, équipe, etc.) devient en même temps de plus en plus complexe. Ainsi, est-il nécessaire d'appliquer un tel modèle dans des cas que nous pourrons qualifier de « simples » ? Par exemple, une politique de contrôle d'accès à un manège dans un parc d'attraction qui stipulerait que « le manège n'est accessible que si le sujet fait plus d'un mètre » ou encore une règle contrôlant l'accès à un film selon les critères d'âge définis par le CSA « l'âge du sujet doit être supérieur à 16 ans pour regarder le film A ».

2.7 Conclusion

Nous avons présenté un bilan des divers modèles formels de contrôle d'accès existants. Ces modèles ont été conçus pour traiter du contrôle d'accès dans des situations bien différentes. Ils introduisent de multiples concepts pour spécifier les politiques de contrôle d'accès. Le but de ces divers modèles est d'offrir la palette la plus vaste possible d'expression de politiques de contrôle d'accès.

Tout d'abord, les modèles de contrôle d'accès classiques (DAC, MAC) ont été proposés pour répondre aux exigences de systèmes d'exploitation. Ensuite, la famille des modèles RBAC, la famille des modèles TBAC, et les modèles qui les ont étendus ont fait l'objet d'études approfondies sur des situations différentes. D'autres modèles ont préféré compléter le concept de rôle dans des modèles RBAC par une ou plusieurs notions pour s'adapter à d'autres situations, comme le contexte dans Or-BAC ou CRBAC.

Dans ces divers modèles, il est important d'identifier des concepts centraux pour que les administrateurs puissent structurer efficacement les droits d'accès. Mais, la structuration du contrôle d'accès est incontournable pour une gestion sûre et efficace des droits, et en conséquence, les approches non structurées sont trop délicates à utiliser, en particulier car elles manquent d'outils pratiques associés. En effet, pour une petite structure, un modèle contextuel simple est un choix satisfaisant : il est envisageable de gérer plusieurs contextes associés aux utilisateurs si leur nombre est réduit. À l'inverse, pour une multinationale, il sera certainement nécessaire d'avoir une notion de rôle générale accompagnée de concepts et de relations spécifiques, capables de prendre en compte de multiples facettes des métiers [Thion 2008].

Nous pouvons donc constater que les modèles formels de contrôle d'accès évoluent en même temps que les technologies et les besoins utilisateurs. En effet, les modèles de contrôle d'accès sont liés aux préoccupations de sécurité existant au moment de leur conception. ***Ainsi, prendre comme hypothèse qu'il est possible de définir un modèle de contrôle d'accès générique et stable dans le temps ne nous semble pas être approprié. De notre point de vue, il nous semble plus envisageable de définir un langage d'expression de politiques d'autorisation et un moteur de décision associé qui soient génériques et extensibles pour mettre en œuvre les différents besoins provenant des technologies, des utilisateurs, des organisations tout en respectant la formalisation proposée par le(s) modèle(s) formel(s) de contrôle d'accès le(s) plus approprié(s) à ces besoins. C'est pourquoi, nous proposons dans cette thèse de concevoir un système d'autorisation adaptable.***

Chapitre3. Adaptabilité et systèmes d'autorisation

3.1 Introduction

Un système informatique représente une solution répondant à des besoins métiers qui est exécutée dans un contexte particulier (e.g. ressources matérielles ou logicielles disponibles, etc.). Un système informatique est dit « adapté » s'il répond correctement à ces différentes contraintes. Lorsque l'une de ces contraintes change, le système informatique initial peut ne plus être adapté (e.g. l'application ne répond plus aux besoins métiers qui ont évolué, elle ne supporte pas le changement du format de données produites par une application tiers dont elle dépend). Il est alors nécessaire de considérer l'adaptation du système informatique pour qu'il reste toujours adapté aux contraintes qui lui sont imposées et qui évoluent (Figure 13).

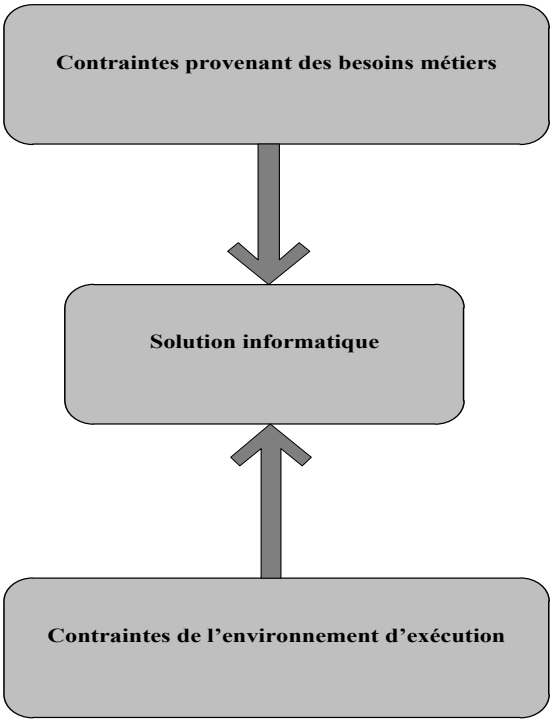

Figure 13 Contraintes d'une solution informatique adaptée

L'objectif de notre travail étant de fournir un système d'autorisation pouvant être réutilisé dans différentes situations (différentes organisations, différents besoins d'autorisation, différents environnements informatiques), la notion d'adaptation est un élément important à considérer. En effet, la réutilisabilité du système d'autorisation que nous visons implique qu'un certain nombre de décisions d'implémentation basées sur des suppositions quant à la nature de l'environnement d'utilisation soient reportées au moment où ces informations sont connues. Ce moment correspond à l'installation du logiciel mais aussi tout au long du cycle d'utilisation. En effet, les besoins d'autorisation et l'environnement d'exécution évoluent dans le temps.

Nous présentons donc dans ce chapitre une introduction sur la notion d'adaptabilité, les concepts associés ainsi que les principales techniques provenant du domaine du génie logiciel afin d'analyser les besoins d'adaptabilité des systèmes d'autorisation.

3.2 Adaptabilité

L'adaptabilité est la propriété qui définit la capacité d'un système, à pouvoir prendre en compte les événements survenant dans son environnement et pouvoir y réagir de façon correcte. En effet, l'adaptation est un mécanisme permettant à un système de fournir ses services sous des conditions particulières ou nouvelles et lui permet d'assurer les modifications nécessaires d'une manière transparente. Elle désigne l'action de s'ajuster et de réagir face aux variations des contraintes de l'environnement ou des besoins métiers [Belaramani 2002]. Il est essentiel de différencier les termes adaptabilité et adaptation. L'adaptabilité représente la capacité ou le degré d'adaptation. L'adaptation est l'action entreprise pour atteindre l'adaptabilité par le changement d'une certaine propriété, d'un paramètre ou d'une métrique [Hacini 2008].

De nombreux travaux de recherche dans le domaine du génie logiciel ont étudié la notion d'adaptabilité [Subramanian et Chung 1999], [Subramanian et Chung 2001],[McKinley et al. 2004], [Escoffier 2007], [Grine et al. 2007] et [Tarvainen 2008]. Ces travaux ont proposé des techniques et des méthodes d'adaptation à différents niveaux du cycle de vie des logiciels. Les besoins d'adaption à considérer peuvent être divers comme répondre aux changements des besoins utilisateurs/applicatifs mais aussi permettre l'accélération du développement logiciel ou encore faciliter la maintenance logicielle.

Nous proposons dans cette section de définir la notion d'adaptabilité et de présenter les principales techniques utilisées dans ce contexte.

3.2.1 Quelques définitions informelles sur l'adaptabilité

Les problèmes liés au besoin d'adaptation sont récurrents dans les systèmes informatiques. Le nombre de travaux et de définitions relatifs à ce terme montre la difficulté de traiter ce concept de manière globale. Afin d'illustrer la diversité des besoins et solutions d'adaptation, nous présentons quelques définitions :

- L'adaptabilité est définie comme la facilité avec laquelle un système ou des parties d'un système peuvent être adaptés aux changements des exigences [Tekinerdogan et Aksit 1996]
- Un programme est appelé adaptable s'il peut être facilement modifié. Un programme est appelé adaptatif s'il change son comportement automatiquement en fonction de son contexte [Lieberherr 1995]
- L'adaptation est la capacité d'une organisation à modifier ses règles internes de fonctionnement, en réponse à des stimuli externes [Highsmith 1999]
- Au fil de temps, l'environnement d'origine (par exemple, la CPU, le système d'exploitation, les besoins métiers ou d'autres caractéristiques externes) pour lequel le logiciel a été développé est susceptible de changer. La maintenance adaptative résulte dans la modification du logiciel pour s'adapter aux changements de son environnement externe [Pressman 1997]
- L'évolution adaptative change un logiciel afin qu'il s'exécute dans un nouvel environnement [Oreizy et al. 1998]
- L'objectif de la maintenance adaptative est de faire évoluer un système pour répondre aux besoins des utilisateurs et métiers [Mikkonen et al. 2000]
- Un logiciel auto-adaptatif est un logiciel qui modifie son propre comportement en réponse aux changements de son environnement d'exploitation [Oreizy et al. 1999]

- Les mesures d'adaptation : une catégorie de mesures de qualité qui adresse la facilité d'un système à évoluer ou à migrer [SEI 2011]
- Etc.

3.2.1.1 Une définition formelle de l'adaptabilité

De manière générale, l'adaptation représente le changement d'un système afin de répondre aux variations de son environnement. Devant la multitude de définitions liées à ce concept, nous avons choisi d'en retenir une qui est à la fois générique et qui réduit les ambiguïtés par son approche formelle. Selon Subramanian et Chung [Subramanian et Chung 2001], l'adaptation d'un système *S* est causée par un changement Δ_E intervenu dans un environnement *E* qui devient un nouvel environnement *E'*. Cette adaptation résulte en un nouveau système *S'* qui idéalement répond aux besoins du nouvel environnement *E'*. Ainsi, l'adaptation peut être représentée par une fonction (Figure 14) : *Adaptation: E×E'×S→S', où meet(S', need (E'))*.

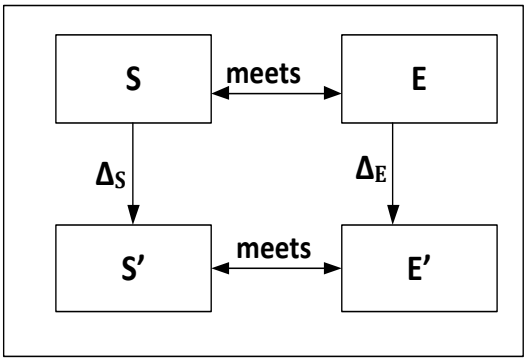

Figure 14 Symboles de la définition formelle de l'adaptation

Par conséquent, un système est adaptable si une telle fonction d'adaptation existe, et l'adaptabilité fait référence à la capacité du système à mettre en œuvre cette fonction d'adaptation. La réalisation de cette fonction d'adaptation implique trois fonctionnalités : 1) la capacité de reconnaître le changement d'environnement Δ_E, 2) la capacité à déterminer le changement à apporter sur le système Δ_S, 3) la capacité à effectuer ce changement pour générer le nouveau système *S'*. Ces trois fonctionnalités sont résumées par les fonctions suivantes :

- *EnvChangeRecognition* : $E' \times E \rightarrow \Delta_E$
- *SysChangeRecognition* : $\Delta_E \times S \rightarrow \Delta_S$
- *SysChange* : $\Delta_S \times S \rightarrow S'$, *où meet(S', need(E'))*

La fonction *EnvChangeRecognition* représente la capacité d'observation. Cette capacité d'observation doit permettre de détecter les variations de l'environnement. En effet, ce sont les variations de l'environnement qui déclenchent les adaptations. Lorsque la variation de l'environnement consiste en une modification des besoins métiers, l'utilisateur du système est la seule entité apte à détecter cette variation. Si cette variation de l'environnement se situe au niveau de l'environnement d'exécution, le système doit inclure des mécanismes d'observation. Si cette observation se limite à une analyse de l'environnement d'exécution au moment de l'initialisation/déploiement du système, l'adaptation ne pourra être que statique. Par contre, si les mécanismes d'observation effectuent une surveillance continue du système d'exécution (via des mécanismes de supervision par exemple), alors l'adaptation peut être une adaptation dynamique.

La fonction *SysChangeRecognition* représente la capacité de décision. Cette fonctionnalité doit inclure des mécanismes permettant de définir une adaptation par rapport à une variation de

l'environnement. Cette activité implique à la fois la compréhension de la variation de l'environnement mais aussi une bonne connaissance du système à modifier pour décider quelle adaptation est nécessaire. Tout comme l'action d'observation, la décision d'une modification à apporter au système peut être prise par plusieurs acteurs. Le programmeur du système peut prendre cette décision lorsque l'utilisateur lui notifie une modification des besoins métiers par exemple. Comme, il connaît la structure interne du programme, il est apte à déterminer les modifications à apporter. L'utilisateur du système peut aussi dans certains cas prendre cette décision. Par exemple, un administrateur d'un serveur Apache 2.0 peut décider qu'il est nécessaire d'utiliser le module mod-auth-ldap pour pouvoir authentifier ses utilisateurs sur son serveur Web avec le serveur LDAP nouvellement installé. Finalement, le système peut lui même décider d'une modification à apporter. Cela permet à l'adaptation d'être dynamique. Par contre cela, implique qu'il ait à la fois une compréhension et une capacité de raisonnement sur lui-même. Des solutions intermédiaires consistent à définir des stratégies d'adaptation. La stratégie d'adaptation [Marangozova 1999] définit les paramètres importants influençant le fonctionnement d'une application et spécifie les actions à entreprendre lorsque ces paramètres ne vérifient pas certaines propriétés. Ces stratégies peuvent être prédéfinies et statiques. Le programmeur fournit plusieurs implémentations d'un même code et spécifie dans quelles conditions elles doivent être utilisées. Par exemple, avoir plusieurs codes d'une même page web pour prendre en compte les différences d'affichage des navigateurs web utilisés (Internet Explorer vs Firefox ou encore affichage smartphone vs écran classique). Les stratégies d'adaptation peuvent aussi être dynamiques, i.e., modifiables au cours de l'exécution.

Enfin, la fonction *SysChange* représente la capacité d'action. Une fois qu'une adaptation a été décidée, il est nécessaire de déterminer les modifications concrètes à réaliser sur le système. Cette action de modification du système doit s'appuyer sur de la validation/vérification afin de confirmer que le nouveau système « *S'* » est bien *adapté* au nouvel environnement *E'*. Cette modification du système peut se situer au niveau de la topologie de l'architecture du logiciel (e.g. création ou migration de blocs de code/composants ou encore modification des connexions entre composants) ou au niveau des composants même (e.g. ajout/retrait/modification de fonctionnalités). Une autre approche consiste à modifier le comportement du système assurant l'exécution du logiciel à adapter via des middlewares adaptables (Open-ORB [Blair and Coulson, 1997], Olan [Balter et al. 1998], R-RIO [Loques et al. 2000] ou encore [David 2001]). Tout comme la fonction de décision, la modification peut être effectuée par le programmeur, l'utilisateur (administrateur qui modifie le fichier apache2.conf pour utiliser le module mod-auth-ldap) ou encore le système lui-même.

3.2.2 Les techniques pour mettre en œuvre l'adaptabilité

Il existe différentes techniques permettant de traiter le problème d'adaptabilité de manière plus ou moins complète. Nous présentons brièvement les principales techniques.

3.2.2.1 La configuration

La configuration est la forme la plus simple, et certainement la plus ancienne, d'adaptation. L'application possède alors un *nombre prédéfini de paramètres variables* qui vont guider l'exécution de l'application. Par exemple, la table de routage d'un routeur, le nombre de connexions simultanées d'un serveur web Apache, ou encore l'adresse IP d'une machine. Les éléments variables peuvent être configurés ou personnalisés au moment de leur instanciation (adaptation statique) ou durant l'exécution (adaptation dynamique). Le mécanisme de configuration peut être fourni soit en tant que paramètres lors du lancement de l'application, soit par l'intermédiaire d'un fichier de configuration ou encore par l'utilisation d'un gestionnaire de configuration. La dernière solution est celle qui offre le plus de flexibilité, mais elle est aussi la plus délicate à mettre en œuvre [Desertot 2007].

Lorsque les logiciels peuvent aussi être configurés (ou reconfigurés) dynamiquement, la gestion

du comportement et du cycle de vie du logiciel ou de ses composants peut devenir "délicate". Rendre une nouvelle configuration opérationnelle nécessite de prendre en compte la transition entre les deux configurations. Par exemple, le module de routage peut devoir terminer d'exécuter une file de datagrammes IP en attente de routage avant de prendre en compte la nouvelle table de routage.

3.2.2.2 La réflexivité

La réflexivité est la capacité d'un système à raisonner sur lui-même voire même de changer sa structure interne. Pour cela, la réflexivité repose sur trois concepts :

- La réification ou chosification qui consiste à concrétiser une notion abstraite. Par exemple, les notions de classes et de méthodes du langage Java sont concrétisées par les classes java.lang.reflect.Class et java.lang.reflect.Method.
- L'introspection permet à un programme de consulter des informations concernant son propre état en utilisant les mécanismes de réification. Par exemple, la méthode java.lang.reflect.Class.getMethods permet de retrouver toutes les méthodes d'une classe donnée.
- L'intercession permet à un programme de modifier sa propre exécution. Par exemple pour le langage Java, le Java Runtime Environment inclut un chargeur de classes Java qui permet de charger dynamiquement des classes dans la machine virtuelle Java. Les méthodes de ces classes peuvent ensuite être utilisées via les méthodes « java.lang.reflect.Class.newIstance » et « java.lang.reflect.Method.invoke ». Le langage python propose ce type de mécanisme de manière plus évoluée. Par exemple, « inspect.getsourcelines » permet de récupérer le code source, « compile » permet de compiler le fichier modifié, et « exec » permet de l'exécuter.

De plus la réflexivité peut s'appliquer à différents niveaux :

- La réflexivité structurelle : se focalise sur la structure des éléments constituants d'un système (composants ou objets par exemple).
- La réflexivité architecturale : se focalise sur les connexions entre les éléments constituants d'un système (composants ou objets par exemple).
- La réflexivité comportementale : se focalise sur l'exécution. Par l'intermédiaire d'une description du comportement du système, il est possible d'influer sur le déroulement de l'exécution du système.

3.2.2.3 La programmation orientée aspect

La programmation orientée aspect est un paradigme de programmation qui permet de simplifier l'écriture d'un programme en améliorant la séparation des préoccupations fonctionnelles, qui correspondent au cœur de métier, et techniques qui sont liées à l'exécution de l'application. L'exemple qui est classiquement donné est la mise en œuvre de la journalisation des événements d'une application qui est souvent programmée de manière éparpillée dans tout le code de l'application. Pour cela, cette approche fournit des mécanismes complémentaires aux paradigmes de programmation classiques (e.g., procédural ou objet) pour mieux « modulariser » les différentes préoccupations d'une application et ainsi de rendre les applications plus faciles à faire évoluer. La programmation orientée aspect fournit des outils pour exprimer des aspects et des tisseurs qui permettent d'intégrer ces aspects au code existant. Pour ce faire la POA a introduit les concepts suivants :

- Les points de jonction et de coupe représentent les endroits du code où doivent s'exécuter les aspects ; un point de coupe étant une combinaison de points de jonction.
- Les méthodes d'un aspect, qui sont associées à un point de coupe, indiquent ce que l'aspect doit réaliser.

```
aspect UnAspect{
  private pointcut coupe() : call(void UneClasse.uneMethode());

before() : coupe(){
System.out.println("exécution de l'aspect");
  }
}
```

Figure 15 Exemple d'aspect en AspectJ

L'exemple de la Figure 15 présente un aspect qui affiche dans la sortie standard le message « exécution de l'aspect » avant que la méthode Une Classe.uneMethode() ne soit exécutée. Ainsi, il est possible de spécifier des changements d'un système sans modifier le code source original.

3.2.2.4 L'inversion de contrôle

L'inversion de contrôle est un patron d'architecture logiciel dans lequel le contrôle du flot d'exécution de l'application n'est plus effectué par l'application elle-même mais par un cadriciel sous-jacent. Dans cette approche, ce n'est pas l'application qui gère les appels au cadriciel mais l'inverse. Le programme utilisateur est alors composé de blocs de codes utilisant l'API du cadriciel, sans que la relation entre ces blocs de codes ne soit écrite « en dur ». La technique d'inversion de contrôle permet donc de réaliser des adaptations dynamiques des logiciels car tout composant déployé délègue la gestion de ses dépendances au cadriciel. L'API SAX (Simple API for XML) est un exemple d'API qui utilise l'inversion de contrôle.

3.2.2.5 L'approche orientée service

L'approche orientée service favorise grandement l'adaptabilité. Cette approche est de plus en plus employée, notamment dans le commerce électronique (Business to Business) ou dans l'intégration d'applications d'entreprise. Il existe différentes définitions pour le concept de service ou pour les architectures orientées service. Parmi les principales nous présentons celle du W3C : *« Une architecture orientée service est un ensemble de composants pouvant être invoqué, et dont les descriptions d'interfaces peuvent être publiées et découvertes »*. Une architecture orientée service (ou Service-Oriented Architecture, SOA) repose sur trois acteurs fondamentaux communs qui sont le fournisseur de service, le consommateur et le registre (Figure 16).

- Le fournisseur de service spécifie un service et veut le rendre découvrable par des consommateurs. Pour cela, il l'enregistre dans un registre de services (également appelé annuaire).
- Le registre de services expose la spécification des services disponibles à tout consommateur à la recherche d'un service spécifique.
- Après négociation auprès du registre, le consommateur obtient une ou plusieurs références vers des implantations du service conformes au contrat recherché. Le consommateur est alors capable de les invoquer.

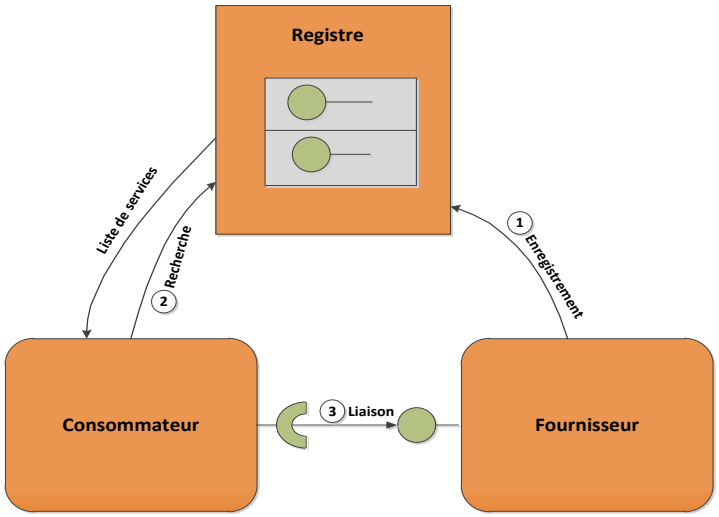

Figure 16. Architecture orientée service

En plus, l'architecture orientée service est dynamique puisque les fournisseurs et consommateurs de services peuvent joindre et quitter l'architecture au cours de l'exécution. L'architecture orientée service apporte aussi une réponse aux problèmes que rencontrent actuellement les entreprises au niveau de la réutilisabilité, de l'interopérabilité et de la réduction de couplage entre les différents systèmes constituant leur système d'information.

3.2.2.6 Discussion

La technique de configuration impose une connaissance complète et totale du système. Elle doit s'appuyer sur d'autres techniques pour modifier la structure interne du programme informatique c.à.d. le code.

La réflexivité apporte la faculté de pouvoir modifier les structures internes de haut niveau du programme (par exemple ses objets) lors de son exécution. Si cette technique offre des possibilités de changer le système de manière très importante, elle reste une solution assez coûteuse car elle impose d'avoir une représentation très fine du système devant être adapté.

La technique de programmation Orientée Aspect introduit une meilleure flexibilité que la technique de réflexivité en simplifiant l'écriture des programmes informatiques, en les rendant plus modulaires et donc plus faciles à faire évoluer. Cependant, cette technique nécessite elle aussi une connaissance très fine du programme à adapter. Cette approche n'implique cependant pas que toutes les briques qui composent l'application soient clairement définies. De plus elle est limitée dans son utilisation dynamique. Introduire ou supprimer (tisser ou détisser) des aspects dans le logiciel durant son exécution est possible (par exemple JAC [JAC 2011] ou SpringAOP [SpringAOP 2011]). Cependant, cela implique que tous les appels effectués par l'application soient interceptés par la plateforme POA et donc cela ralentit l'exécution de l'application [Legond-Aubry 2005].

L'inversion de contrôle et l'approche orientée service proposent une meilleure modularité pour l'assemblage des applications. Elles offrent l'extensibilité et l'adaptabilité dynamique des architectures du système. La différence fondamentale entre ces deux techniques est la façon dont les injections des dépendances sont fournies au composant qui les requiert. Dans le cas de l'inversion de contrôle, les injections de dépendances de tout service requis sont fournies explicitement, ce qui implique une difficulté au niveau du débogage du logiciel et une mise en œuvre coûteuse. Dans l'approche orientée

service, les injections de dépendances sont implicites et nécessitent uniquement l'accès à un registre de services.

3.3 Etude de l'adaptabilité d'un système d'autorisation

Un système d'autorisation peut être décomposé en trois entités (Figure 17). La **fonctionnalité de mise en œuvre des décisions** d'autorisation (PEP) qui transforme des informations brutes de bas niveau en informations pouvant être analysées par la fonctionnalité de prise de décision et applique les décisions d'autorisation sur le système cible. La **fonctionnalité de prise de décisions** d'autorisation (PDP) analyse les informations transmises par la fonctionnalité de mise en œuvre par rapport à des politiques d'autorisation écrites dans un langage donné et rend une décision d'autorisation. Il existe donc un lien de dépendance forte entre la fonctionnalité de mise en œuvre et celle de décision correspondant aux informations échangées entre elles. De plus, la fonctionnalité de prise de décision peut si nécessaire récupérer des informations complémentaires. Finalement, les **politiques** (P) sont l'expression de contraintes d'autorisation. La syntaxe et la sémantique dans des règles exprimées dans le langage de politique doivent être connues de la fonctionnalité de prise de décision. Ainsi, il existe aussi une relation de dépendance entre la fonctionnalité de prise de décision et le langage de politique.

Nous retrouvons ces éléments dans des systèmes d'autorisation couramment utilisés aujourd'hui, comme par exemple, le pare-feu inclus dans le système d'exploitation Linux qui est composé de Netfilter et IPtables. Netfilter est un ensemble d'hameçons (hooks) au niveau du noyau linux permettant aux modules du noyau d'enregistrer des fonctions de rappel dans le processus de traitement de la pile réseau. Une fonction de rappel est alors appelée pour chaque datagramme IP qui traverse cet hameçon dans la pile réseau. Le module ip_tables et la commande iptables fournissent respectivement une structure générique de règles d'autorisation et un système de décision, et une interface utilisateur pour la spécification de règles d'autorisation.

Nous basons notre analyse des besoins d'adaptation d'un système d'autorisation sur la formalisation de Subramanian et Chung [Subramanian et Chung 2001] présentée dans la section 3.2.1.1. Nous considérons donc un état *S* tel que le système d'autorisation est adapté à son environnement *E*, i.e. dans le modèle de Subramanian et Chung la fonction *meet(S, need(E))* retourne vrai.

Dans le contexte des systèmes d'autorisation, l'environnement (E) peut être raffiné en trois éléments (Figure 17) :

- L'environnement technologique contrôlé (ECTRL) représente l'entité physique qui est contrôlée par le système d'autorisation ainsi que les informations disponibles au niveau de cette entité. Par exemple, l'ECTRL par IPtables/Netfilter est le flux d'exécution de la pile réseau Linux, ainsi que toutes les informations disponibles qui sont structurées selon le format spécifique de la pile réseau Linux.
- L'environnement technologique complémentaire (ECOMP) représente l'ensemble des technologies informatiques disponibles à l'administrateur sécurité dans lesquelles se trouvent des informations additionnelles, i.e. non fournies par l'ECTRL, liées aux besoins de sécurité. Par exemple, un serveur LDAP ou Active Directory qui contient des informations sur les utilisateurs ou encore une base d'information de gestion SNMP dans laquelle se trouve des informations sur des équipements, etc.
- L'environnement de besoins de sécurité (EBSEC) représente les exigences de sécurité qui doivent être mises en œuvre par le système d'autorisation. Ces exigences doivent donc être formulées dans le langage de politique du système d'autorisation. Par exemple, mise en œuvre d'une politique RBAC donnée, définition de contraintes d'accès par rapport à la position GPS des utilisateurs, etc.

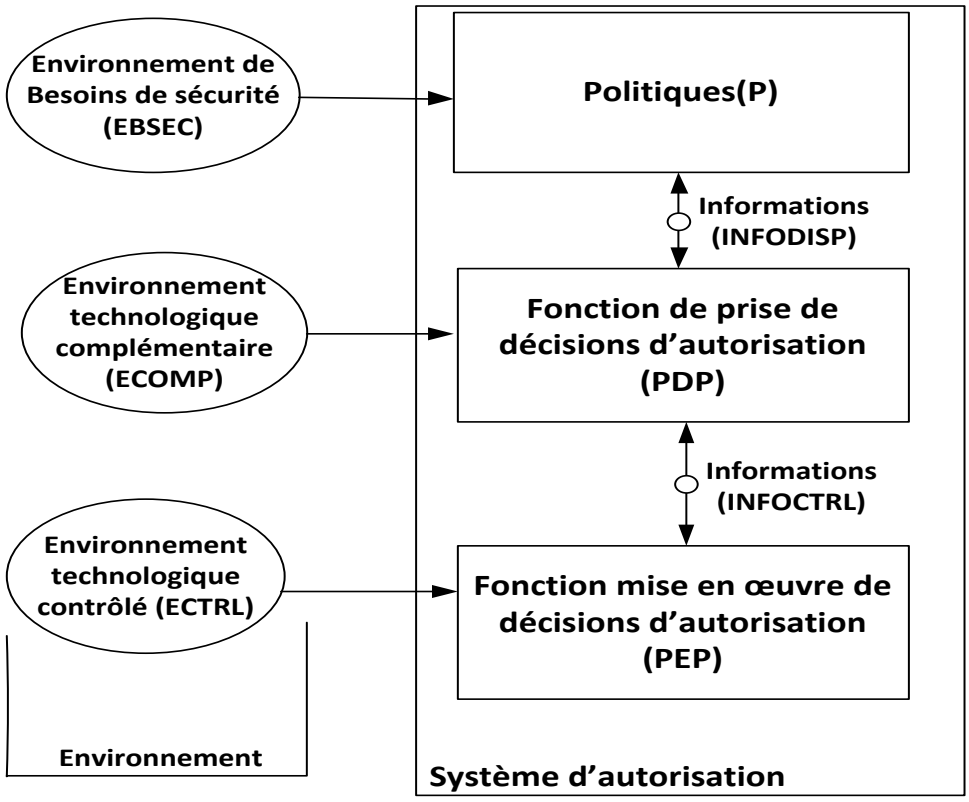

Figure 17. Le contexte de système d'autorisation

Comme expliqué dans l'introduction de cette section, il existe des dépendances entre les entités qui composent un système d'autorisation. Nous définissons ces relations par (Figure 17) :

- L'interface (INFOCTRL) représente les informations échangées entre le PEP et le PDP. Cette interface comprend le format et la structure des informations véhiculées dans les requêtes formulées par le PEP et les décisions prises par le PDP.
- L'interface (INFODISP) représente les informations manipulées dans les politiques d'autorisation au travers du langage de politique. Ces informations possèdent un format et une structure compris par le PDP. INFODISP intègre donc l'ensemble des informations transmises par le PEP auxquelles viennent se rajouter les informations additionnelles provenant de l'environnement technologique complémentaire ECOMP. De plus, INFODISP représente aussi le langage de politique permettant de spécifier des expressions sur les informations de sécurité. INFODISP constitue donc l'ensemble de la connaissance du PDP pour prendre une décision.

Etudier les besoins d'adaptabilité qui émergent des variations possibles de l'environnement E, nous amène à examiner l'impact des variations d'ECTRL, d'ECOMP et d'EBSEC sur le système d'autorisation. Nous analysons donc l'impact de chacune de ces entités.

1. **Variation de l'environnement technologique contrôlé** (ECTRL)

Une variation de l'ECTRL (Δ_{ECTRL}) peut résulter d'une modification de l'API utilisée pour contrôler le flux de traitement/d'exécution d'une requête (par exemple, passage de Apache 1.3 à Apache 2.0), ou encore de la modification des structures internes utilisées pour stocker les informations brutes, comme l'ajout/suppression d'une information ou encore la modification du format d'une information.

La variation Δ_{ECTRL} implique que la modification du système comporte une modification du PEP et/ou de l'interface INFOCTRL. Ceci peut se représenter sous la forme : *SysChangeRecognition*(Δ_{ECTRL}, S) = Δ_S tel que $\Delta_{PEP} \cup \Delta_{INFOCTRL} \subseteq \Delta_S$.

- S'il existe Δ_{PEP} tel que *meet(SysChange*(Δ_{PEP},S),need(E')) alors l'adaptation se limite au niveau de l'entité PEP car il n'est pas nécessaire de modifier l'interface INFOCTRL. Ce type de changement intervient lorsque l'API d'accès au flux de traitement d'une requête change. Il est aussi possible de modifier le PEP lorsque le format d'une information de sécurité a été modifié afin que cette modification n'ait aucun impact sur le PDP.
- Par contre, s'il n'existe pas de Δ_{PEP} tel que *meet(SysChange*(Δ_{PEP},S),need(E')) alors $\Delta_{PEP} \cup \Delta_{INFOCTRL} \cup \Delta_{PDP} \cup \Delta_{INFODISP} \subseteq \Delta_S$, c'est à dire qu'il est nécessaire d'étudier la modification de l'interface INFOCTRL et donc du PDP et/ou de l'interface INFODISP.
    - S'il existe $\Delta_S = \Delta_{PEP} \cup \Delta_{INFOCTRL} \cup \Delta_{PDP}$ tel que *meet(SysChange*(Δ_s,S),need(E')) alors une modification sur les éléments PEP, INFOCTRL et PDP existe pour adapter le système.
    - Sinon, il faut aussi modifier l'entité politique (par exemple, modifier une règle). Par conséquent, $\Delta_{PEP} \cup \Delta_{INFOCTRL} \cup \Delta_{PDP} \cup \Delta_{INFODISP} \cup \Delta_P = \Delta_S$. Si *meet(SysChange*($\Delta_s$,S),need(E')) alors le système peut s'adapter au changement Δ_{ECTRL}, sinon le système ne peut pas s'adapter et les besoins de sécurité EBSEC ne peuvent plus être satisfaits.

2. **Variation de l'environnement technologique complémentaire** (ECOMP)

Une variation de l'ECOMP (Δ_{ECOMP}) peut consister dans la mise à jour d'une base de données, le changement du format des données stockées, le changement de la technologie utilisée pour stocker les données, etc.

Du point de vue de notre modèle, la variation Δ_{ECOMP} implique donc une modification du PDP et/ou de l'interface INFODISP. Ceci peut se représenter sous la forme : *SysChangeRecognition*(Δ_{ECOMP}, S) = Δ_S tel que $\Delta_{PDP} \cup \Delta_{INFODISP} \subseteq \Delta_S$.

Une modification du PDP peut adapter le système sans altérer l'interface INFODISP. Dans ce cas, il existe Δ_{PDP} tel que*meet(SysChange*(Δ_{PDP},S),need(E')). Toutefois, s'il est nécessaire de modifier INFODISP, alors il sera aussi nécessaire de modifier la politique $\Delta_{PDP} \cup \Delta_{INFODISP} \cup \Delta_P \subseteq \Delta_S$. Si une telle modification satisfait *meet(SysChange*($\Delta_{PDP} \cup \Delta_{INFODISP} \cup \Delta_P$,S),need(E')), le nouveau système reste adapté. Dans le cas contraire, le système ne peut s'adapter au nouvel environnement.

Prenons par exemple, une politique d'autorisation qui fait référence à une organisation d'origine de l'utilisateur. Considérons que la valeur de cette organisation d'origine est une chaîne de caractères déterminée à partir de l'adresse email de l'utilisateur qui est de la forme « nom@organisation.fr ». La récupération de l'organisation est effectuée par l'application d'une expression régulière qui découpe l'adresse email et produit la chaîne de caractères organisation d'origine. Les adresses email sont stockées dans une base de données MySQL. Une requête SQL effectuée par rapport à l'identifiant de l'utilisateur (information donnée par le PEP) permet de récupérer l'adresse email. Un changement de l'environnement complémentaire ECOMP peut être ici : une modification du format des adresses email qui devient « nom@département.organisation.fr », ou encore la migration des comptes utilisateurs sur un serveur LDAP. Cet exemple met en avant deux éléments clés de Δ_{ECOMP} : le format de stockage de l'information et la technologie d'accès à l'information.

Dans le cas de la modification de l'adresse email, Δ_{PDP} tel que *meet(SysChange*(Δ_{PDP},S),need(E')) peut consister à modifier l'expression régulière pour prendre en compte le nouveau format d'adresse email. Si ce n'est pas possible, l'interface INFODISP est

modifiée car l'élément organisation d'origine est maintenant de la forme « département.organisation ». Pour que le système puisse s'adapter, le langage de politique (INFODISP) doit pouvoir permettre de manipuler différemment cette nouvelle information sous la forme d'une nouvelle règle (Δ_P). Par exemple, en modifiant l'expression « organisation d'origine est IRIT » en « organisation d'origine se termine par IRIT ».

Dans le cas d'une migration de la base de données dans un serveur LDAP, il est nécessaire de trouver Δ_{PDP} tel que *meet(SysChange*(Δ_{PDP},S),need(E')) pour prendre en compte la nouvelle technologie d'accès à l'information.

3. **Variation de l'environnement de besoins de sécurité** (EBSEC)

Une variation de l'EBSEC (Δ_{EBSEC}) représente un changement provenant des besoins de sécurité comme l'application d'un nouveau modèle de contrôle d'accès, la prise en compte de nouveaux éléments de sécurité, etc.

Si ces nouveaux besoins peuvent être exprimés par le langage de politique sans nécessiter de modifier la connaissance du PDP alors il existe Δ_P tel que *meet(SysChange*(Δ_P,S),need(E')). Si par contre, ces nouveaux besoins imposent l'utilisation de nouvelles informations et/ou de nouvelles fonctionnalités dans le langage alors $\Delta_{PDP} \cup \Delta_{INFODISP} \cup \Delta_P \subseteq \Delta_S$. Dans ce cas, la modification du PDP consiste à ajouter des fonctionnalités pour récupérer cette nouvelle information depuis l'environnement complémentaire ou alors pour étendre le langage de politique. Finalement, les besoins de sécurité peuvent aussi impacter l'environnement contrôlé, $\Delta_{PEP} \cup \Delta_{INFOCTRL} \cup \Delta_{PDP} \cup \Delta_{INFODISP} \cup \Delta_P \subseteq \Delta_S$. S'il n'existe pas $\Delta_S = \Delta_{PEP} \cup \Delta_{INFOCTRL} \cup \Delta_{PDP} \cup \Delta_{INFODISP} \cup \Delta_P$ tel que *meet(SysChange*(Δ_S,S),need(E')) alors le système n'est pas capable de mettre en œuvre les besoins de sécurité.

En conclusion, les variations (Δ_E) de l'environnement (E) se reflètent par des modifications dans les trois éléments ECTRL et/ou ECOMP et/ou EBSEC. Ces modifications apportent des changements (Δ_S) d'une partie ou toutes parties du système d'autorisation. La détection et la mise en œuvre de tout ou partie de ces changements peut soit être effectué statiquement (adaptation statique) ou encore à la volée sans avoir à arrêter le système (adaptation dynamique).

3.4 Conclusion

La notion d'adaptabilité est importante dans le contexte des systèmes d'autorisation dont l'objectif est à la fois de répondre aux nouveau besoins de sécurité mais aussi de s'intégrer dans des environnement technologies différents. Nous avons donc présenté dans ce chapitre une étude générale sur l'adaptabilité par rapports aux différents travaux effectués dans le domaine du génie logiciel. Nous avons étudié les différentes facettes liées au concept d'adaptabilité ainsi que les principales techniques développées pour rendre un système informatique adaptable. De plus, nous avons analysé les besoins d'adaptabilité des systèmes d'autorisation en nous basant sur la formalisation proposée par Subramanian et Chung [Subramanian et Chung 2001].

Dans le chapitre suivant, nous présentons un état de l'art des principaux systèmes d'autorisation existants en se focalisant sur leur capacité d'adaptabilité.

Chapitre4. Etat de l'art sur les systèmes d'autorisation

4.1 Introduction

Les systèmes d'autorisation sont considérés comme un sous-ensemble des mécanismes de contrôle d'accès [Vimercati et al. 2007]. Un système d'autorisation est responsable de prendre des décisions d'autorisation (comme accorder, refuser...) et de les appliquer lorsqu'une entité demande l'accès à une ressource/service. L'évolution à la fois des besoins de sécurité mais aussi des environnements technologiques à contrôler fait que les systèmes d'autorisation doivent s'adapter en permanence. Nous avons défini dans le chapitre précédent les besoins d'adaptation des systèmes d'autorisation. Dans ce chapitre, nous analysons la capacité d'adaptation des systèmes d'autorisation existants par rapport à cette étude.

Il existe plusieurs propositions d'implémentations des systèmes d'autorisation. Cependant, beaucoup d'entre elles ont été développées pour un contexte donné sans prendre en compte les besoins d'adaptation. Nous ne traiterons donc pas ces solutions dans ce chapitre. Par exemple, SELinux [SELinux 2011] implémente le modèle MAC pour le système d'exploitation Linux, PERMIS [PERMIS 2011] implémente le modèle formel RBAC, GSAS (geospatial role-based authorisation system) [Atluri et Chun 2007] implémente le modèle formel GSAM (geo-spatial authorization model) pour l'environnement Web, CAS [Pearlman et al. 2002] qui traite la notion de « groupe » pour l'organisation virtuelle, VOMS [Alfieri et al. 2005] qui traite la notion de « groupe » et de « rôle » pour l'organisation virtuelle etc.

Un nombre plus restreint de travaux sur les systèmes d'autorisation ont abordé cette notion d'adaptation. Nous avons classifié ces travaux en deux groupes. Le premier groupe, qualifié de multi-PDPs, propose d'étendre la capacité d'expression de besoins de sécurité et de prise de décision en permettant d'ajouter des PDPs ; chaque PDP étant spécialisé pour un type de politique donné. Le deuxième groupe est constitué de travaux qui proposent que ce soit le langage ou le modèle de politique qui soit extensible. Nous analysons plus en détails le standard OASIS XACML qui nativement intègre la notion d'extensibilité et qui possède un nombre important d'implémentations.

4.2 Les approches intégrant plusieurs PDPs

Dans cette section, nous présentons le système RSBAC qui permet d'intégrer plusieurs PDPs localement ainsi que les travaux de Chadwick et Fatema qui ont proposé la même approche mais de manière distribuée.

4.2.1 RSBAC

RSBAC (Rule Set-Based Access Control pour Linux) [RSBAC 2011] [Blanc 2006] est un cadre de contrôle d'accès flexible pour les noyaux des systèmes d'exploitation Linux. Il implémente le modèle MAC et ajoute la notion de rôle du modèle RBAC. Il est basé sur le travail fait par Abrams et La Padula [Abrams et al.1995] sur le modèle GFAC (Generalized Framework for access control) [Blanc 2006]. GFAC est un modèle d'implémentation de mécanismes de contrôle d'accès dans les systèmes d'exploitation Linux. Par conséquent, RSBAC décompose le système d'autorisation en deux composants (Figure 18):

- Access Control Decision Facility (ADF) : la partie qui est responsable des décisions d'autorisation (cet élément est équivalent au PDP). Cette entité appelle tous les modules de décisions actives et génère une décision combinée finale. Les décisions sont fondées sur le

type d'accès (type de requête), l'objectif d'accès et sur les valeurs des attributs attachés au sujet et la ressource cible à accéder. D'autres attributs indépendants peuvent être utilisés par des modules individuels, par exemple, le module responsable de la vie privée ou privacy (PM).

- Access Control Enforcement Facility (AEF) : cette entité non modulaire est responsable pour appliquer la décision finale de l'ADF via des appels système.

Dans RSBAC, l'ADF est un ensemble de modules de décision spécifiques à des « modèles de sécurité »[1]. De nombreux modules de décisions d'autorisation sont disponibles (Figure 18) comme :

- Module AUTH (Authenticated User) : ce module gère les permissions de changement d'UID (user identifier). Dans ce modèle de sécurité, chaque processus est lié à une liste d'UID possibles (capabilities).
- Module UM (User Management) : il est responsable de la gestion des utilisateurs et groupes standards de Linux directement dans l'espace noyau.
- Module RC (Role Compatibility) : ce module apporte la notion de rôles suivant le modèle RBAC.
- Module ACL (Extensive Access Control Lists) : ce module implémente les listes de contrôle d'accès (ACL) en y ajoutant si nécessaire la notion de rôles. Le sujet peut être vu comme un utilisateur, un rôle RC ou un groupe ACL et l'action de type RSBAC ou spécifique à ACL.
- Module MAC (Mandatory Access Control) : ce module est l'implémentation directe du modèle MAC. Il autorise l'usage de niveaux de classification et de sensibilité dans des règles
- Etc.

De plus, RSBAC offre la possibilité d'écrire de nouveaux modules pour d'autres types de politiques.

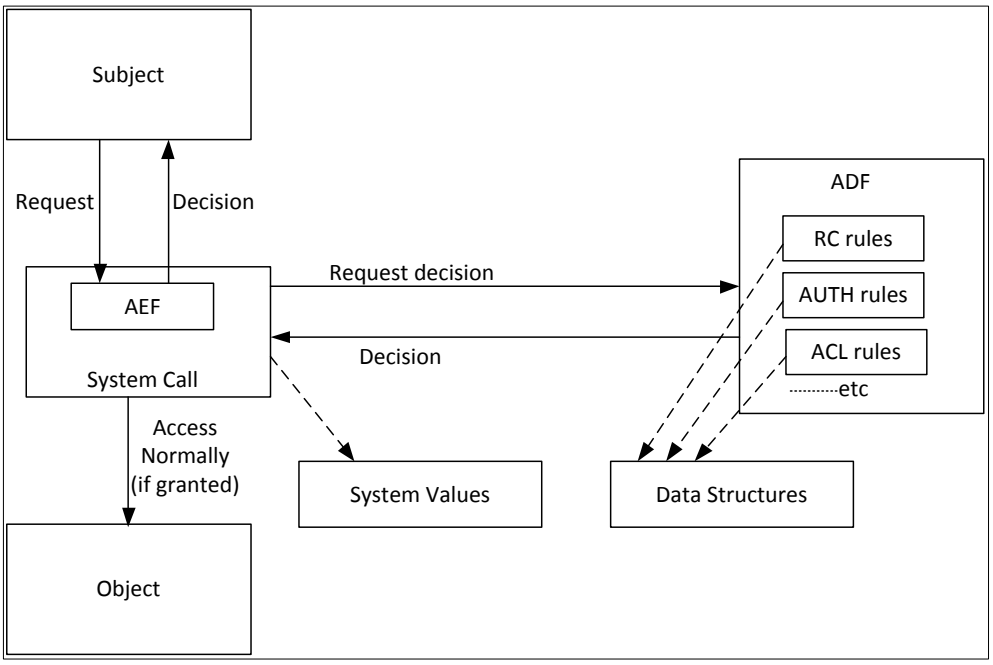

Figure 18Architecture RSBAC

[1]Le terme modèle de sécurité employé dans RSBAC n'est pas synonyme de modèle formel de contrôle d'accès.

4.2.2 L'infrastructure d'autorisation avancée de Chadwick et Fatema

L'infrastructure d'autorisation proposée par Chadwick et Fatema [Chadwick et Fatema 2009] reprend la même idée que RSBAC mais permet aux différentes entités d'être distribuées. Cette architecture a été définie initialement pour traiter du problème des politiques collées. Une politique collée est une politique qui reste attachée à la ressource qu'elle protège. Un PDP qui va prendre une décision par rapport à une politique collée doit être capable de comprendre ce langage. L'idée des auteurs est donc d'avoir un PDP maître qui peut appeler des PDPs spécifiques à une politique donnée (par exemple, PERMIS, Keynote, etc). Cette architecture (Figure 19) ajoute aux classiques PEP et PDP les composants suivants : un PEP indépendant des applications (AIPEP - Application Independant PEP), un service de validation des accréditations (CVS – Credential Validation Service) et un PDP maître (Master PDP).

Le composant AIPEP est une fonction de mise en œuvre de décisions d'autorisation indépendante de l'application qui est vu comme un proxy entre le PEP et le MasterPDP. Il est responsable de coordonner les différentes actions de l'infrastructure d'autorisation (validation des accréditations, communication avec le Master PDP et mise en œuvre des obligations). Il présente une interface unique pour le PEP dépendant de l'application, afin de faciliter l'intégration.

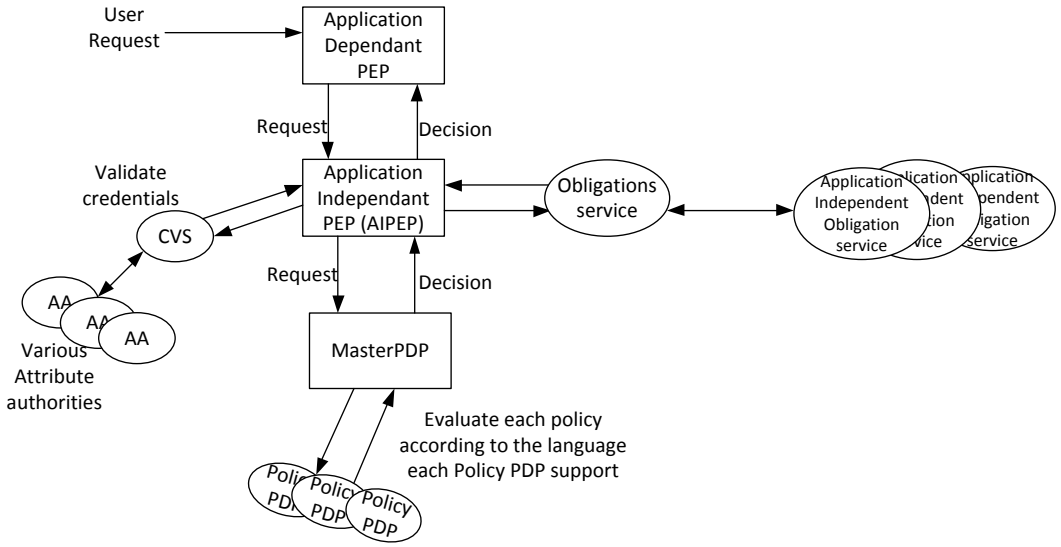

Figure 19 Architecture pour l'infrastructure d'autorisation avancée

Le CVS est responsable de récupérer et valider les informations additionnelles pour les accréditations des utilisateurs non incluses dans les requêtes des entités. Il est configuré avec une politique de validation des accréditations qui lui indique quelles accréditations sont valides en termes d'autorités de confiance (AA), des attributs de confiance émis et pour quels groupes d'utilisateurs. Le CVS peut travailler dans les modes Push, ou Pull, ou Push/Pull. Le mode Pull signifie que l'entité qui demande l'accès ne dispose pas des accréditations et nécessite que le CVS doive les récupérer. Le mode Push signifie que le message contenant la demande d'accès contient l'ensemble des accréditations qui doivent être validées par le CVS.

Enfin, le MasterPDP est responsable de transmettre la requête à différents PDP (où chaque PDP traite un langage de politique spécifique), et d'obtenir les décisions d'autorisation de ces PDP, et enfin de résoudre les conflits entre ces décisions avant de retourner la décision d'autorisation finale au composant AIPEP. De plus, le MasterPDP doit être configuré pour connaitre le PDP auquel envoyer la politique collée en fonction du langage de politique utilisé. Les échanges entre les différents composants se font via les standards SAML ou WS-Trust.

4.2.3 Synthèse

Ces deux systèmes proposent d'intégrer plusieurs PDPs par une entité appelée ADF ou Master PDP. Cette entité est capable de transmettre des requêtes à différents PDPs et d'assembler les différentes décisions en résolvant si nécessaire les conflits pour produire une décision finale grâce à une politique de combinaison des décisions. Cette approche permet donc d'étendre partiellement une politique (Δ_P), le langage de politique ($\Delta_{INFODISP}$) et la fonction de décisions d'autorisation (Δ_{PDP}).

Cependant, cette approche pose deux problèmes. Tout d'abord, si les besoins de sécurité imposent l'écriture d'une règle d'autorisation faisant appel à des concepts existant dans deux langages X et Y implémentés par deux PDP différents (PDP_X et PDP_Y). Par exemple, « *si* (C_1^X **ET** C_2^Y) **OU** C_3^X *alors* accepter » où C_A^B est la condition A exprimée dans le langage B. Les conditions C_1^X et C_3^X sont évaluées par le PDP_X et C_2^Y par le PDP_Y. De plus les opérateurs logiques **ET** et **OU** doivent être exprimés dans le langage de la politique de combinaison car analysé par l'ADF ou le Master PDP. Par conséquent, écrire une telle règle va nécessiter de combiner les deux langages X et Y ainsi que le langage de la politique de combinaison des décisions. De plus, lorsque les PDPs spécifiques peuvent partager des données en commun, l'ordre dans lequel ces PDPs seront appelés est très important et il peut être différent entre deux règles. Cette approche peut donc entraîner une complexité très importante dans l'écriture de règles et donc favoriser des erreurs d'écriture.

Le travail de Chadwick et Fatema impose au master PDP d'avoir préalablement un PDP pour chaque langage de politique qu'il peut potentiellement rencontrer. Ce système ne traite pas le cas où le master PDP reçoit une politique collée écrite dans un langage pour lequel il n'a pas de PDP associé. Enfin, ces deux systèmes sont limités par rapport aux informations de l'environnement technologique complémentaire. Le CVS du travail de Chadwick et Fatema est limité aux accréditations des utilisateurs. De plus, les modules de décision de RSBAC ont une architecture monolithique. Si une information complémentaire change de format ou de technologie d'accès, il faut modifier tout le module de décision responsable de cette information.

4.3 L'approche d'un langage de politique générique

Dans cette section, nous présentons le système MotOrBAC qui implémente le modèle formel de contrôle d'accès Or-BAC, les systèmes d'autorisation qui implémentent un langage de politiques à base d'ontologies, et le système Ponder qui propose un langage de politiques orienté-objet.

4.3.1 MotOrBAC

4.3.1.1 Introduction

MotOrBAC [Autrel et al. 2008] [MotOrBAC 2011] est un système de contrôle d'accès qui a été développé pour administrer et implémenter une politique de sécurité spécifiée conformément au modèle formel de contrôle d'accès Or-BAC. L'objectif de MotOrBAC est d'unifier dans un modèle formel de sécurité générique l'expression de toutes les exigences de sécurité réseau, système ou applicatives. Or-BAC est un modèle de contrôle d'accès qui permet d'exprimer une politique de sécurité au niveau organisationnel (voir Chapitre 2). 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) dans le modèle Or-BAC [Cuppens et al. 2006b].

L'architecture de MotOrBAC (Figure 20) comprend une interface graphique d'administration (MotOrBAC-GUI) et un moteur d'évaluation de politiques qui correspond à un PDP (OrBAC API). La spécification d'une politique de sécurité Or-BAC nécessite de définir plusieurs entités abstraites et leurs relations hiérarchiques. L'interface de MotOrBAC (MotOrBAC-GUI) permet de définir

graphiquement (sous forme d'arborescence) les différentes entités du modèle Or-BAC ainsi que leurs relations : les entités abstraites (organisations, rôles, activités, vues, et les contextes) et les entités concrètes (sujets, actions, objets) sont spécifiées en utilisant une approche orientée objet.

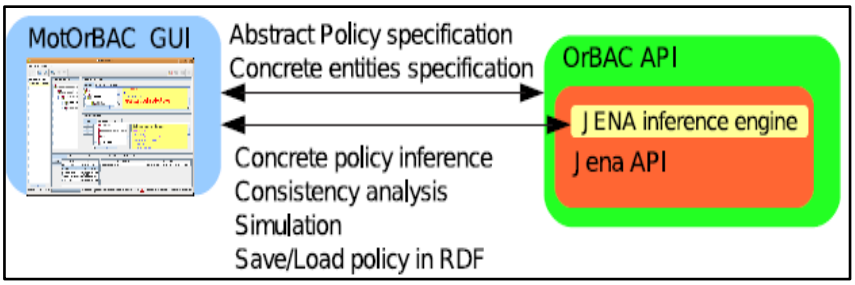

Figure 20 Architecture de MotOrBAC

MotOrBAC GUI peut être utilisé pour effectuer plusieurs tâches sur les politiques de sécurité Or-BAC :

- Edition : L'administrateur peut introduire les différents entités spécifiques au système d'information basé sur le modèle Or-BAC (telles que les organisations et sous-organisations, les rôles, les activités, les vues et les contextes) durant la spécification de politiques de contrôle d'accès.
- Simulation : L'interface permet de visualiser toutes les règles de sécurité concrètes inférées à partir des règles de sécurité abstraites. Elle 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 au niveau abstrait ou organisationnel introduite par les administrateurs.
- Analyse : L'interface permet de détecter les conflits au niveau concret ou abstrait, et d'intégrer des stratégies de résolution de conflits.
- Administration : L'interface inclut une fonction d'administration implémentant le modèle Ad-OrBAC [Cuppens et Miège 2004] pour administrer les politiques Or-BAC. La fonction d'administration est utilisée pour exprimer ou de mettre à jour la politique de sécurité du système d'information. Dans MotOrBAC, l'expression de la politique d'administration est utilisée pour contrôler l'accès des administrateurs qui ont accès à MotOrBAC.

Cette interface graphique a été développée pour spécifier et administrer des politiques Or-BAC. La politique en sortie est un fichier XML de type RDF (Resource Description Framework) [RDF 2011]. En outre, le moteur d'évaluation de politiques (Or-BAC API) est utilisé pour gérer les politiques de type RDF générées par l'interface graphique MotOrBAC-GUI et de les interpréter. Il utilise la bibliothèque java de Jena [JENA 2011] pour représenter une politique Or-BAC comme un graph RDF. Jena dispose d'un moteur d'inférence qui est utilisé par l'API OrBAC pour inférer sur les politiques Or-BAC et les conflits.

De plus, l'API permet d'évaluer des politiques Or-BAC contenant des informations contextuelles. L'interface MotOrBAC GUI permet de spécifier une partie des contextes du modèle Or-BAC, qui sont les contextes temporels, les contextes déclarés par l'utilisateur et les contextes exprimant une condition sur les attributs des entités concrètes. La composition de contextes est aussi implémentée. L'API Or-BAC est programmée en Java en suivant l'approche de la programmation orientée aspects (AOP) pour séparer les préoccupations de sécurité d'autres préoccupations relatives à l'application. Ainsi, l'API implémente un système de plug-in basé sur la technologie OSGi. Pour cela, MotOrBAC peut être étendu avec l'aide de ce système de plug-ins pour ajouter de nouvelles

fonctionnalités. Finalement, les politiques Or-BAC peuvent aussi être définies directement en Java grâce à l'API Or-BAC [Autrel et al. 2008].

4.3.1.2 Synthèse

La solution MotOrBAC part de l'hypothèse que le modèle Or-BAC permet de représenter tout besoin de sécurité. Notre hypothèse est différente. Au regard du nombre de modèles de contrôle d'accès existants et qui apparaissent chaque année, nous ne pensons pas qu'il puisse exister un unique modèle de contrôle d'accès stable dans le temps. Cette évolution se retrouve au niveau d'Or-BAC avec par exemple multi-OrBAC [Kalam et Deswarte 2006] ou encore O2O [Cuppens et al. 2006a]. C'est pourquoi, nous avons pris comme hypothèse qu'il existe une multitude de modèles de contrôle d'accès et que les personnes qui définissent leur politique de contrôle d'accès (par exemple, les administrateurs de sécurité dans les organisations ou les utilisateurs si pas d'organisation) peuvent choisir le modèle qui convient à leur situation. Ainsi, nous désirons que ce soit le système d'autorisation qui soit capable d'exprimer tout modèle de contrôle d'accès voire une combinaison de modèles.

Par ailleurs, l'administrateur doit définir les entités réelles ainsi que leurs attributs dans l'interface MotOrBAC afin que cela soit importé dans le fichier RDF. Dans le cadre d'une organisation, ces informations peuvent déjà exister dans diverses bases de données ou annuaires. Cette approche amène à dupliquer l'information et donc à complexifier sa gestion. Par exemple, le rôle des utilisateurs peut se trouver dans un annuaire LDAP (par exemple, le rôle peut être retrouvé à partir du nom). Il va falloir dupliquer cette information utilisateur-rôle pour qu'il soit pris en compte dans le système MotOrBAC. Cette tâche peut être complexe s'il existe un nombre élevé d'utilisateurs. De plus, changer le rôle d'un utilisateur implique modifier les données de l'annuaire LDAP mais aussi le fichier de politique Or-BAC. Ceci complexifie aussi la gestion du cycle de vie de l'information technologique complémentaire.

4.3.2 Les systèmes d'autorisation à base d'ontologies

4.3.2.1 KAoS

KAoS – Knowledge Agent oriented System [Uszok et al. 2004] est l'un des premiers efforts pour représenter la politique de sécurité à l'aide d'un langage de web sémantique OWL, c.à.d. basé sur les ontologies. Au cours de l'initialisation, KAoS charge en premier « core KAoS Policy Ontology » [Uszok et al. 2004], qui définit les concepts de base pour représenter un environnement générique d'acteurs ou de clients ainsi que les politiques dans ce contexte. C'est l'ontologie de politique de base. Puis, KAoS charge des ontologies supplémentaires, en étendant les concepts de « core KAoS Policy Ontology », avec des notions spécifiques à l'environnement particulier à contrôler et le domaine d'application. L'ensemble des ontologies de politiques dans KAOS est nommé KPO (KAOS Policy Ontology).

La Figure 21, montre l'architecture pour mettre en œuvre KAoS [Uszok et al. 2008] qui intègre les composants suivants : Distributed Directory Service (DDS) pour la gestion des politiques KAoS, et le garde KAoS pour déterminer la réponse d'autorisation ou d'obligation à une action demandée par un client. Un garde KAoS intègre à la fois les fonctionnalités de PEP ET PDP.

Tout d'abord, les administrateurs de sécurité spécifient les politiques d'ontologies KAoS au format OWL avec l'aide de l'API KAoS (KPAT GUI). Ensuite, ces politiques sont distribuées dans les annuaires « Distributed Directory Service (DDS) » qui sont accessibles directement par les gardes. Lorsque les politiques sont ajoutées ou modifiées, ou quand un garde se connecte ou se reconnecte à l'annuaire de service distribué. Le DDS rassemble la mise à jour des informations appropriées pertinentes à un garde particulier. Il tient un registre sur les intérêts de chaque garde et une histoire de

ce qui lui a déjà été envoyé. Le DDS envoie les politiques préconfigurées et/ou les mises à jour qui sont demandées par un garde pour que celui-ci puisse prendre des décisions d'autorisation.

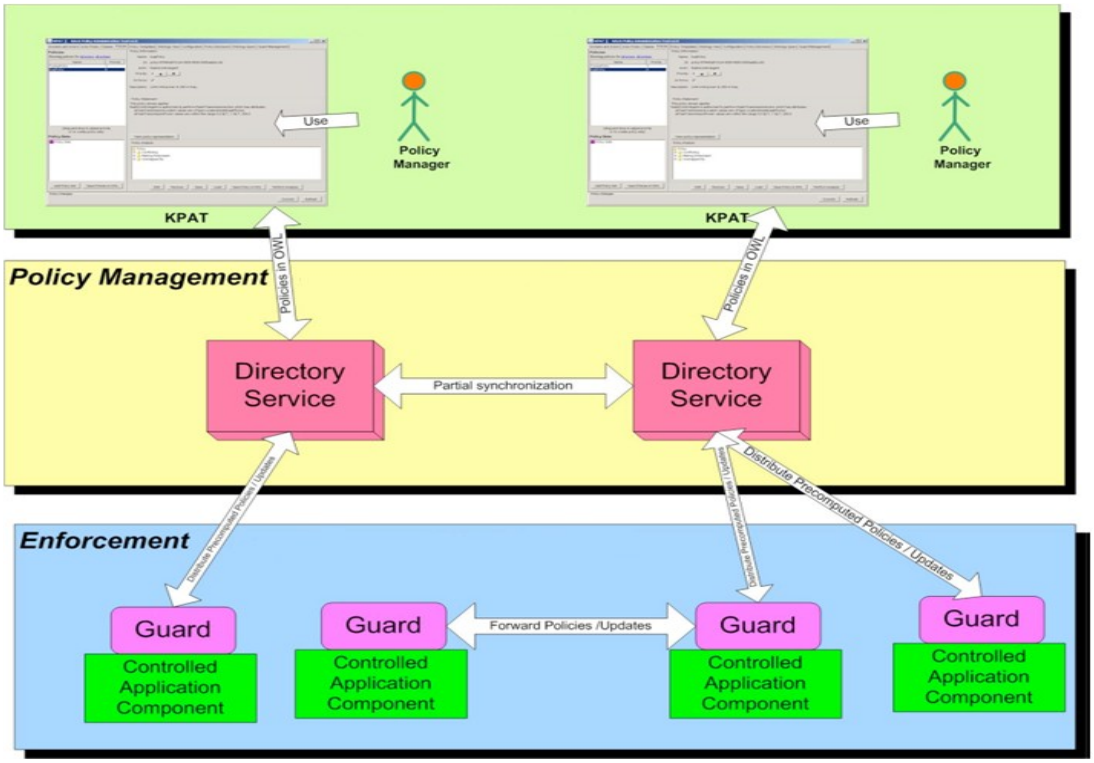

Figure 21 Architecture de KAOS

Les gardes sont installés sur les applications à contrôler. Ainsi, les gardes doivent pouvoir retrouver non seulement les informations sur les politiques, mais aussi sur l'état du système et des entités gérées. Les gardes n'effectuent pas la surveillance directement. Ils offrent des interfaces pour brancher des mécanismes de surveillance ou des connecteurs à des bases de données.

4.3.2.2 REI

REI [Rei 2011] est un autre langage de spécification de politiques basé sur OWL-Lite [OWL 2011]. Il permet d'écrire des politiques avec des ontologies et des prédicats en prolog. Les ontologies sont utilisées pour capturer les connaissances sur un certain domaine d'intérêt et prolog pour inférer des informations dynamiquement. Les ontologies décrivent les concepts, les individus (c.à.d. les instances des concepts), et les relations qui s'établissent entre ces concepts dans un certain domaine d'intérêt. REI définit une politique comme un ensemble de règles décrivant des concepts déontiques tels que l'autorisation, l'interdiction, l'obligation et l'exemption. L'exemple suivant est une interdiction appelée « CannotUseIR », qui interdit à une entité dénotée par « JohnDoe » d'accomplir une action dénotée par « InfraredPortUsage » :

```
<deontic:Prohibition rdf:ID="CannotUseIR>
<deontic:actor rdf:resource="#JohnDoe"/>
<deontic:action rdf:resource="#InfraredPortUsage"/>
</deontic:Prohibition>.
```

REI n'inclut pas de fonction de mise en œuvre de politiques (PEP), mais seulement un moteur de décisions ou PDP qui est utilisé pour raisonner sur des politiques REI afin de fournir des décisions d'autorisation. Pour cela, plusieurs implémentations de REI existent (comme le travail de Kagal [Kagal et al. 2004] pour les services web, et le travail de Patwardhan [Patwardhan et al. 2004] pour l'environnement ubiquitaire) qui tentent de présenter une infrastructure d'autorisation de REI pour répondre à un contexte donné. Par exemple, le travail de Patwardhan présente une architecture et une

preuve de concept de mise en œuvre d'une infrastructure de sécurité à base de REI pour les appareils mobiles dans un environnement ubiquitaire. La Figure 22 montre l'architecture de cette infrastructure d'autorisation.

Dans la partie client, le « Policy Enforcer » (équivalent à un PEP) est le médiateur d'accès situé sur le périphérique demandant l'accès aux ressources. Il est responsable de la mise en œuvre des décisions d'autorisation (appelées dans ce travail « politique de décision ») envoyées par le serveur de politiques. Si aucune politique n'est reçue par le PEP, le PEP applique une politique par défaut qui est très restrictive et qui permet une communication minimale, limitée pour aller chercher une nouvelle décision dans le serveur de politiques.

Dans la partie serveur, il existe deux composants : le moteur de politiques REI et le serveur de politiques. La politique de sécurité est décrite en utilisant l'ontologie REI. Une ontologie sur un domaine spécifique peut également être utilisée pour décrire des informations d'un domaine spécifique c.à.d. des informations complémentaires. Le moteur de REI a une interface Java et utilise Prolog pour le raisonnement. Le serveur de politiques est responsable de transférer les requêtes envoyées par le PEP au moteur de politique REI. Ensuite, le moteur de REI génère une décision qui est encapsulée dans un certificat. Le certificat contient l'ensemble des décisions d'autorisation (accorder ou refuser l'accès). Le PEP valide la signature, la période de validité et l'identification de l'émetteur du certificat.

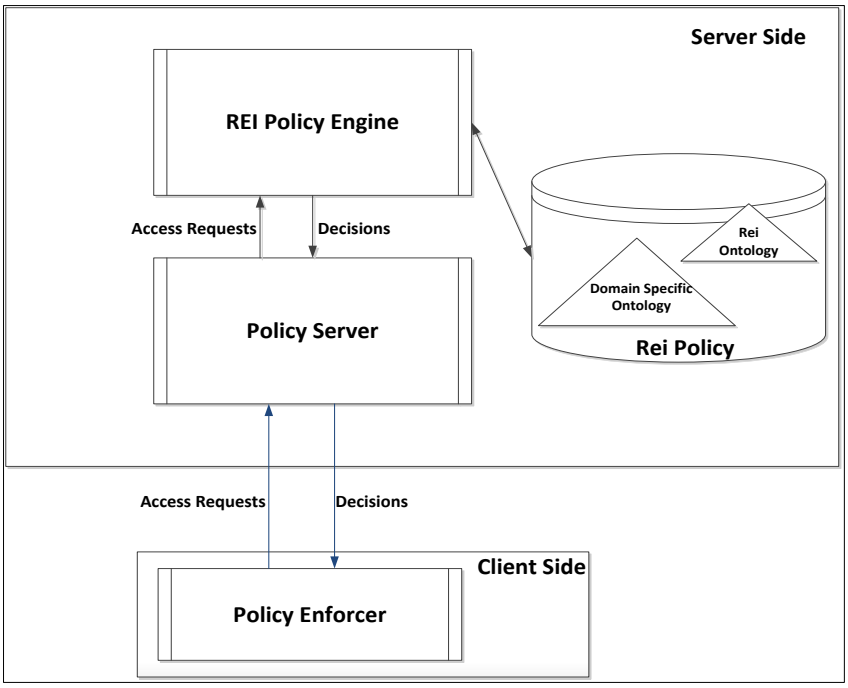

Figure 22 Architecture d'une infrastructure d'autorisation REI

4.3.2.3 POLIPO

POLIPO est un cadre qui combine le contrôle d'accès distribué avec les ontologies dans un système de systèmes (SOS). Un SOS est un système dont les composants sont eux-mêmes des systèmes complexes [Trivellato et al. 2009]. Les auteurs de POLIPO utilisent le même concept que KAoS et REI avec l'utilisation d'un langage de web sémantique pour la spécification de politiques. Les politiques POLIPO sont spécifiées avec l'aide de quatre constructions :

- Les éléments atomiques de l'ontologie sont utilisés pour interroger la base de connaissances représentée par des ontologies. Chaque concept ou classe dans une ontologie est identifié comme *ConceptURI*. *ConceptURI(a)* exprime que *a* est une instance du concept *ConceptURI*. Par exemple *psd:Client('A') et psd:Organisation('C')* sont des concepts.

RelationshipURI permet de définir les relations entre les concepts. Par exemple, *psd:Employépar('A', 'C')* est une relation entre les concepts *'A'* et *'C'* et représente le fait que le client 'A' travaille pour l'organisation 'C'.

- Les éléments atomiques des accréditations représentent les déclarations signées numériquement par un émetteur sur un attribut d'un sujet. Les accréditations sont de la forme : cred(issuer ; attr ; subject ; [[c1... cn]; [a1... an]]). Où « issuer » est le nom unique de l'entité qui a signé l'accréditation, « attr » est l'attribut pour lequel l'accréditation est délivrée, « subject » est le nom unique pour le sujet de cet attribut, et « [[c1... cn]; [a1... an]] » est une liste de propriétés optionnelles.
- Les éléments atomiques des autorisations dénotent la permission d'un sujet à effectuer une action sur un objet. Une autorisation est de la forme perm(action, sujet, objet), où action est une relation entre le concept sujet et le concept objet.
- Les contraintes sont spécifiées à l'aide de la programmation de contraintes logiques (CLP) (par exemple, =, >, <, etc.) ou de prédicats définis par l'utilisateur (par exemple, aboutSurveillance('File')).

A partir de ces éléments de base [Trivellato et al. 2009], les politiques POLIPO sont formalisées dans le paradigme de la programmation logique. Les règles POLIPO sont des clauses de« Horn » de la forme : $h \leftarrow b_1,...,b_n$. Où « h » est appelé la tête qui est un élément atomique, et $b_1,...,b_n$ appelés le corps sont des éléments atomiques avec $n \geq 0$. POLIPO distingue la spécification de trois types de règles :

- Les règles d'autorisation pour spécifier les règles de permissions de sujets à effectuer des actions sur des objets ou des ressources. Par exemple, la politique tirée de [Trivellato 2009] « subject X is authorized to read object Y if Y is about surveillance as defined by the policy owner and X provides a credential issued by the Blue Star Navy stating that he is a senior officer » est formalisée dans POLIPO comme suit : *perm('psd:read', 'X', 'Y')* ← *aboutSurveillance('Y'), cred('BlueStarNavy', 'psd;SeniorOfficer', 'X', [])*.
- Les règles de définition de prédicats pour spécifier des règles de prédicats de type (CLP). Par exemple, la politique tirée de [Trivellato 2009] « An object is about surveillance if it concerns mission 'Surveillance' and has sensitivity level less than 3 according to the Blue Star Navy » est formalisée dans POLIPO comme suit : *aboutSurveillance('X')* ← *bs:aboutMission('X'; 'Surveillance'),bs:sensitivityLevel('X'; 'Y'), Y < 3*.
- Les règles de délivrance d'accréditations pour spécifier des règles sur les accréditations données aux sujets. Par exemple, la politique tirée de [Trivellato 2009] « The BlueStar Navy releases a psd:SeniorOfficer credential to entities defined as senior officers » est formalisée dans POLIPO comme suit : *cred('BlueStarNavy'; 'psd:SeniorOfficer'; X; [])*← *psd:SeniorOfficer(X)*

La Figure 23 montre l'architecture de POLIPO [Trivellato et al. 2009]. Deux types de requêtes existent dans POLIPO : les demandes d'accréditations « credential release request » et les demandes d'accès aux ressources « access request ». Le PEP (Policy enforcement point) prend une requête et l'envoie au PDP (Policy décision point) complétées, si nécessaire, par des informations additionnelles existant dans une base de données afin de l'évaluer. En fonction du type de requête, le PDP consulte soit la politique des accréditations soit la politique d'autorisation pour prendre sa décision. Cette décision résulte d'une couche de prédicats qui permet d'inférer sur les politiques avec l'ontologie locale (qui est basée sur l'ontologie globale). Ces prédicats peuvent être définis par l'utilisateur à l'aide des règles de prédicats. Enfin, le PDP donne la décision au PEP qui l'applique.

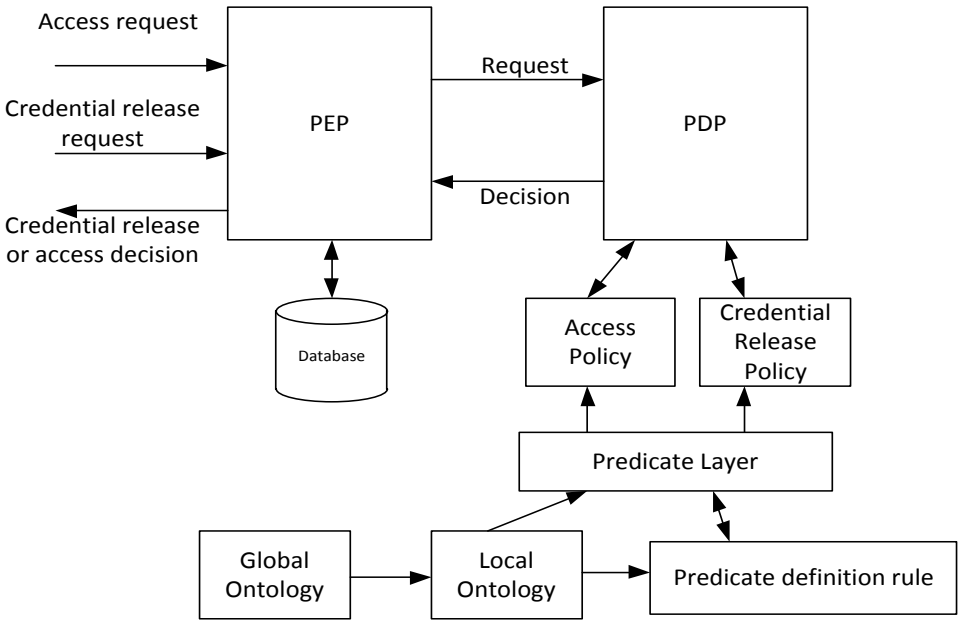

Figure 23 Architecture de POLIPO

4.3.2.4 Synthèse

Les langages de politiques à base d'ontologies (comme KAoS, REI ou POLIPO) sont une approche intéressante. En effet, les ontologies offrent naturellement des capacités d'extensibilité. Par conséquent, un PDP capable de raisonner sur des ontologies peut normalement raisonner sur les changements appliqués sur l'ontologie.

Toutefois, définir un concept qui soit universellement accepté est un défi difficile à relever [Kingston 2008]. Définir de manière universelle la relation « est un » est sujet à divers problèmes comme la confusion des sens, la réduction des sens, la sur-généralisation, ou encore la confusion de taxonomie des rôles. Par exemple, les emplois de « assisant professor » aux États-Unis et « maître de conférence » en France se ressemblent, mais sont en même temps différents. Ajouté à cela, les réponses de l'expert peuvent être inexacts (malentendu lors de la définition), les niveaux de détails choisis (le niveau d'abstraction choisi est généralement déterminé par le problème abordé ce qui crée un problème de généricité), etc. De plus, les langages du web sémantique utilisés pour la représentation d'ontologie présentent encore certaines difficultés comme une syntaxe complexe, une description déclarative longue, et des liens hypertextes et de référence vers des ressources externes qui font que la spécification de la politique devient très difficile à comprendre. En outre, l'intégration d'ontologies différentes peut être au moins aussi difficile que l'intégration des ressources qu'elles décrivent [Horrocks 2008]. En général, la réutilisation des ontologies est indécidable et algorithmiquement insoluble pour la logique de description sous-jacente OWL DL [Grau et al. 2008].

D'un point de vue architectural, ces trois systèmes imposent que le PEP doive fournir toutes les informations nécessaires au PDP. Il doit donc transformer toutes les informations brutes de l'environnement technologique contrôlé dans le format de l'ontologie. Toutefois, rien n'indique ni comment ni où le PEP récupère ces informations. Comment le PEP s'adapte à un changement de technologie de stockage de l'information brute ? Comment le PEP s'adapte à un changement de format de l'information ? De plus, lorsque l'information est stockée directement sous la forme de l'ontologie, la même information est dupliquée et pose les problèmes de gestion que nous avons décrits dans la synthèse sur MotOrBAC (section 4.3.1.2).

Enfin, la généricité de cette approche pose aussi un problème de performance du moteur de décision. En effet, les outils comme Protégé [Protégé 2011] intègrent des plugins comme « Kazuki »

[Kazuki 2011] qui transforment une ontologie dans un langage de programmation comme Java. Cependant, le code Java généré par ces outils à partir d'une ontologie est loin d'être optimisé. Par exemple nous avons créé une ontologie très simple qui consiste uniquement dans la définition d'un concept appelé « point ». Nous avons défini deux propriétés appelées x et y de type entier et une relation en SWRL [SWRL 2011] appelée *estADroite* entre les instances du concept point tel que *estADroite(?a, ?b)* si *?a.x < ?b.x*. Le générateur de code « Kazuki » a produit 13 classes et 43 méthodes pour ce simple exemple.

4.3.3 Ponder

4.3.3.1 Introduction

Ponder [Ponder 2011] [Twidle et al. 2009] est un langage déclaratif orienté-objet développé par *Imperial College* dont l'objectif est de spécifier à la fois des politiques de gestion et de sécurité. Ainsi, Ponder permet d'écrire et de mettre en œuvre des politiques d'autorisation et des politiques d'obligation de type événement/condition/action pour les besoins de gestion.

L'une des particularités de Ponder est que tout élément (une politique, une requête, un objet, un sujet etc.) est considéré comme étant un objet géré (« Managed object »). L'administrateur peut configurer ces objets gérés, y compris modifier la politique, en cours d'exécution du système Ponder grâce à un langage de haut niveau appelé PonderTalk qui est basé sur SmallTalk [Smalltalk 2011]. Un objet géré doit donc être capable de recevoir et de répondre aux messages envoyés par PonderTalk. D'un point de vue concret, les objets gérés sont écrits en Java. Le lien entre les mots clés des messages PonderTalk et les méthodes Java se fait via des annotations Java (par exemple @Ponder2op ()).

Les objets gérés Ponder sont regroupés dans des domaines. La notion de domaine est proche de celle de répertoire dans les systèmes de fichiers hiérarchiques [Cakic 2003]. Un domaine n'encapsule pas les objets gérés, mais contient les références des interfaces des objets gérés. Les objets gérés peuvent être membres de plusieurs domaines. Un domaine peut être membre d'un autre domaine. Dans ce cas, il est appelé sous-domaine du domaine parent. Les domaines sont identifiés de manière unique par le chemin de domaine tout comme les chemins absolus dans les systèmes de fichiers. Par exemple « A/D » représente le sous-domaine « D » du domaine « A ».

```
root/tauthdom at: "a1" put:
(newauthpol
subject: root/personnel/nurse/
ward1/nurse1
action: checkpatient()
target: root/patient/ward1/patient1
focus: 'subject')
```

Figure 24 Exemple d'une politique d'autorisation en Ponder

La Figure 24 présente une commande qui ajoute une nouvelle politique d'autorisation Ponder. Cette politique est un objet géré créé dans le domaine (/tauthdom). Une politique d'autorisation doit avoir quatre arguments (le sujet, l'action, l'objet, et le focus). Les sujets et les objets sont spécifiés en utilisant les chemins des domaines. Les actions représentent les opérations ou les méthodes définies dans la classe d'un objet géré. Le premier argument est le domaine de l'objet géré qui représente le sujet. Dans l'exemple de la Figure 7, il s'agit de « nurse1 » qui fait partie du domaine « ward1 » qui est un sous-domaine du domaine « nurse », etc. Le deuxième argument « checkpatient() » est une méthode de l'objet géré « infirmière ». Le troisième argument représente le domaine de l'objet ou de

la ressource. Dans notre exemple, la ressource est l'objet géré « patient1 » qui fait partie du domaine « ward1 », qui est un sous domaine de « patient », etc. Il faut noter ici que « root/personnel/nurse/ward1 » et « root/patient/ward1 » sont deux domaines différents.

Ponder permet aussi de définir des politiques de type gestion au travers de politique d'obligation. Ces politiques sont de type événement/condition/action. Si un événement est détecté et que une ou plusieurs conditions sont vérifiées alors le système Ponder exécute un certain nombre d'actions, c'est à dire des méthodes sur des objets gérés [Twidle et al. 2009].

4.3.3.2 Synthèse

Ponder est une contribution importante dans le système de gestion à base de politiques dont l'objectif est de pouvoir couvrir un large éventail d'environnements et d'applications.

Le concept de domaine est important dans Ponder. Cette structuration des objets gérés s'adapte très bien à la hiérarchie des rôles du modèle RBAC [Damianou et al. 2002]. Cependant, il est difficile d'envisager des structures plus complexe comme la hiérarchie (rôle, localisation) du modèle LRBAC qui utilise une combinaison de deux hiérarchies (une pour les rôles et une autre pour les domaines des localisations logiques). Alors, en imposant une structure spécifique, Ponder est limité dans le nombre de modèles de contrôle d'accès qu'il est capable de représenter.

De plus, la mise en œuvre actuelle « Ponder2 » permet d'étendre le PDP avec du code Java supplémentaire pour des objets gérés, afin de représenter tous les besoins de sécurité non pris. De plus, la modification du PDP peut être faite de manière dynamique avec des messages PonderTalk. Cependant, cela peut poser un problème de performance du système, car des codes résiduels inutiles peuvent subsister. Par exemple, le code d'objet géré peut subsister dans le système alors que plus aucune politique n'y fait référence. Par conséquent, il manque un mécanisme qui gère le cycle de vie des extensions par rapport à la politique en cours d'exécution. Enfin, le problème de réutilisation des extensions Ponder n'a pas été traité. Par conséquent le code d'un objet géré ne peut pas être réutilisé.

4.4 XACML

4.4.1 Introduction

XACML (eXtensible Access control MarkupLanguage) est un langage de contrôle d'accès basé sur XML, qui a été standardisé par OASIS [XACML 2011]. XACML fournit un langage de politiques de contrôle d'accès basé sur les attributs. De plus, ce langage fournit une architecture pour la mise en œuvre du contrôle d'accès : un protocole de type requête/réponse donne les moyens d'exprimer des requêtes d'accès et les réponses appropriées.

Dans ce protocole, une demande d'accès aux ressources est un contexte de requête qui spécifie les éléments suivants [Mankai 2005] :

- Les attributs des sujets (Subjects),
- Les attributs de la ressource (Resource),
- Les attributs de l'action (Action),
- Les attributs de l'environnement i.e. toute information (telle que temps, localisation, contexte...) autre que celles relatives au sujet, objet et action.

Comme suite à une requête, la réponse générale peut être accorder *(permit)* ou refuser (*deny*) l'accès. Par contre, d'autres réponses sont possibles pour faire face à des situations exceptionnelles [Mankai 2005]. Quand une erreur inattendue survient ou que le système n'arrive pas à répondre, la réponse est indéterminée (Indeterminate). Par contre, si l'évaluation se déroule correctement mais qu'aucune règle (politique ou ensemble de politiques) n'est appliquée à des requêtes, alors la réponse sera non-applicable (Not Applicable).

Le langage de politique XACML décrit les exigences de contrôle d'accès en termes de contraintes sur les attributs des sujets, des ressources, des actions et de l'environnement. Plus précisément, les attributs peuvent être des caractéristiques du sujet, de la ressource, de l'action, ou de l'environnement dans lequel la requête est faite. Les attributs ont un identifiant, l'Uniform Resource Name (URN), un type de données, et une valeur [XACML 2011] [Scaglioso et al. 2008].

La politique dans XACML est un ensemble de règles et une cible (Target), et peut également être un ensemble facultatif d'éléments d'Obligations. L'élément cible permet d'identifier la politique ou les règles applicables à une requête d'accès. Il spécifie les conditions que le sujet, la ressource et l'action doivent vérifier afin qu'une politique ou une règle soit applicable à la ressource requise. Une règle de contrôle d'accès est un ensemble de conditions et une décision (permit, ou deny). Ainsi, il est encore possible de regrouper les politiques de contrôle d'accès en des ensembles de politiques (PolicySet) et ce, pour structurer les politiques et les règles. La (Figure 25) illustre une politique XACML, qui contient une règle qui permet à tous les sujets à la condition d'être membres du groupe « developers » de faire l'action « read » sur n'importe quelle ressource.

```
<Policy PolicyId="ExamplePolicy"
       RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">http://server.example.com/code/docs/developer-guide.htm
          <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#anyURI"
                                       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
  <Rule RuleId="ReadRule" Effect="Permit">
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
            <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                                       AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
    <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                                    AttributeId="group"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">developers</AttributeValue>
    </Condition>
  </Rule>
</Policy>
```

Figure 25 Exemple d'une politique XACML

L'architecture de gestion XACML décrit les différentes entités et leurs rôles liés au processus de décision de contrôle d'accès. Une version globale de l'architecture est représentée dans (Figure 26). Le « context handler » est l'entité du système qui est responsable de convertir les requêtes d'accès dans le format natif de l'application en une forme de contexte XACML. Il convertit également les décisions d'autorisation de la forme XACML en une réponse dans le format natif de l'application. Le modèle opère selon les étapes suivantes [XACML 2011] [Scaglioso et al. 2008] :

1. Le Policy Administration Point (PAP) prend les politiques écrites par les administrateurs de sécurité et les rend disponibles au PDP. Ces politiques représentent la politique de gestion complète qui contrôle les décisions prises par le PDP.
2. Le demandeur d'accès envoie une requête d'accès au PEP.
3. Le PEP envoie la requête d'accès au context handler au format natif (langage supporté par le PEP), optionnellement incluant des attributs pour le sujet, la ressource et l'environnement.
4. Le context handler construit un contexte de requête XACML et l'envoie au PDP.
5. Le PDP peut demander des attributs additionnels du context handler pour le sujet, la ressource et l'environnement qui n'existent pas dans la requête.
6. Le context handler demande ces attributs du Policy Information Point (PIP).
7. Le PIP obtient les attributs demandés d'une source extérieure (base de données SQL, annuaire LDAP,..) et retourne les attributs demandés au context handler.
8. Le context handler envoie les attributs demandés au PDP qui évalue la politique.
9. Le PDP retourne la décision d'autorisation au context handler.
10. Le context handler traduit le contexte de réponse XACML au format de réponse natif du PEP. Le context handler retourne la réponse au PEP qui applique la décision d'autorisation.

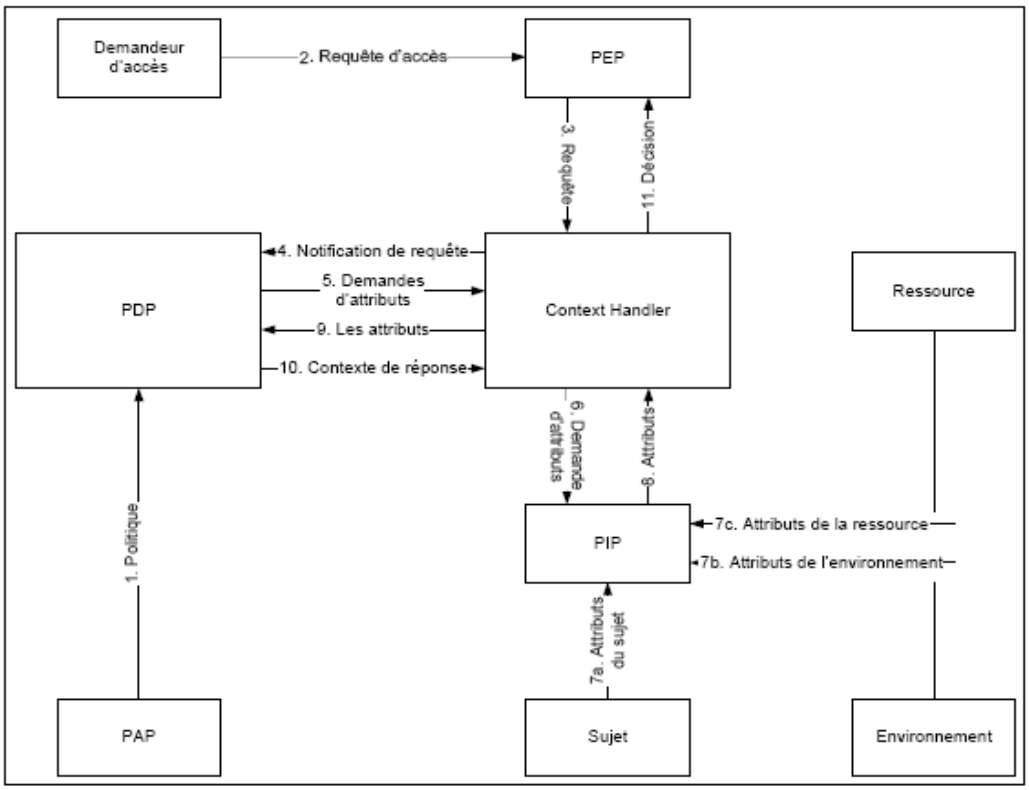

Figure 26 Architecture de XACML

Pour mieux illustrer le comportement typique de XACML analysons un scénario où un sujet (utilisateur, application, etc.) veut accéder à une ressource. Le sujet envoie sa requête à l'entité qui protège la ressource (par exemple un serveur web). Cette requête est capturée par le PEP. Cette requête est envoyée au Context Handler qui la transforme en une requête XACML contenant les attributs de l'utilisateur, la ressource demandée, l'action requise et, d'autres informations relevant du contexte. Ensuite, le Context Handler envoie la requête XACML au PDP. Le PDP examine la requête et récupère la politique XACML applicable à cette requête. Ensuite, le PDP prend la décision d'accès conformément aux règles de la politique. Cette réponse est alors retournée au PEP qui se charge de l'appliquer. Ainsi, un PDP peut gérer un nombre croissant de politiques XACML et peut communiquer avec plusieurs PEPs afin de prendre les décisions d'autorisation qui seront appliquées par les PEPs.

XACML peut être utilisé dans des environnements distribués où les PEPs et PDPs sont répartis au niveau des infrastructures hétérogènes. XACML sépare les politiques des applications protégées et propose un standard pour l'expression des autorisations. Ainsi, XACML permet aux systèmes de sécurité hétérogènes de partager les politiques. En conséquence, les administrateurs de sécurité qui utilisent le langage de politiques XACML ne sont plus obligés d'écrire leurs politiques avec différents langages.

Enfin, la version 3.0 actuelle du standard XACML permet d'exprimer des politiques d'administration et de délégation. Une politique d'administration permet de répondre à la question « Qui est autorisé à rédiger des politiques et à propos de Quoi » [Kamel 2008]. Une politique d'administration permet de contrôler les types de politiques que les individus peuvent créer et modifier. La version 2.0 du standard XACML ne traite pas les conditions sous lesquelles les politiques sont créées. Elle considère que des politiques existent et qu'elles sont utilisées par le PDP. De plus, une politique de délégation spécifie "permettre à autrui d'exercer une action qu'un sujet est autorisé à accomplir ". Son intérêt est de permettre à des utilisateurs de définir des politiques à durée limitée pour déléguer des droits à d'autres utilisateurs. Dans la version 2.0 du standard XACML, des politiques de délégation du type « un utilisateur peut faire des actions à la place d'un autre utilisateur » peuvent être spécifiées. Cependant, il ne permet pas à des utilisateurs comme dans la version 3.0 de générer des politiques à la volée, de type « pendant que je suis en vacances, un autre utilisateur peut approuver des requêtes à ma place ».

4.4.2 Les extensions du langage de politiques XACML

Le langage XACML inclut des types de données (entier, réel, ensemble, etc.) et des fonctions (entier-supérieur, appartient à, etc.) basiques. Ces types de données et fonctions sont limités pour représenter toutes les exigences de sécurité qui émanent des modèles formels. Ainsi, si l'on prend l'exemple de modèles considérant le rôle, on trouve la nécessité d'exploiter une structure mathématique (ensemble d'ordre partiel POSET) pour adresser les relations d'héritage du modèle RBAC. Pour construire la hiérarchie entre les valeurs de rôle, il faut vérifier les trois propriétés mathématiques de POSET que sont la réflexivité, l'antisymétrie, et la transitivité. Le standard de XACML ne peut pas présenter cette structure mathématique et la relation d'ordre entre les valeurs de rôles avec des types de données tels que chaîne de caractères, entier, booléen... Ces types de données nécessitent des fonctions de comparaison entre valeurs et on ne peut pas exprimer la relation d'héritage avec ces fonctions.

OASIS a donc spécifié un ensemble de profils qui raffinent ou ajoutent de nouveaux éléments à XACML pour répondre à des situations données. Nous avons classifié ces profils en deux groupes. L'approche du premier groupe consiste à jouer avec les possibilités de XML pour étendre le langage XACML sans le modifier (c.à.d. sans ajouter de nouveaux attributs/fonctions/types de données au langage). Le profil RBAC [RBAC_Profile 2005] ou encore le profil Or-BAC [Abi Haidar et al. 2006] appartiennent à ce groupe. Le deuxième groupe propose d'étendre le langage de politiques avec de

nouveaux attributs/fonctions/types de données. Le profil GeoXACML [GeoXACML 2011] appartient à ce groupe.

Tout d'abord, le profil RBAC est défini par OASIS afin de supporter l'utilisation du modèle formel de contrôle d'accès RBAC dans les politiques. L'idée consiste à créer des liens entre documents XML grâce aux chemins XML (XPath). L'utilisation de ce mécanisme XML permet de spécifier la notion de hiérarchie de RBAC sans modifier le code des PDPs.

Pour décrire la hiérarchie des rôles, le profil RBAC XACML suggère l'utilisation de deux types particuliers d'ensembles de politiques (PolicySets) de type XML :

- L'ensemble des rôles « RolePolicySet (RPS) » : contient l'ensemble des RPSs qui permet à l'administrateur de spécifier des rôles. Un RPS est créé pour chaque rôle identifié (par exemple le rôle du sujet est égal à "manager").
- L'ensemble des permissions « Permission PolicySet (PPS) » qui contient l'ensemble des autorisations pour un rôle (par exemple le sujet avec le rôle « manager » peut faire l'action « lire » sur la ressource « clients.html »).

Considérons par exemple, que l'Université Paul Sabatier (UPS) propose à ses étudiants une liste des offres d'emploi d'été dans le site web « www.jobsUPS.fr ». La politique d'autorisation pourrait être formellement donnée par l'administrateur de sécurité de l'organisation « UPS » comme «Seuls les étudiants de l'organisation UPS peuvent accéder au site web www.jobsUPS.fr ». L'organisation UPS délivre trois types de diplômes : « phd », « master », et « licence ». La hiérarchie de rôles de l'UPS définie par l'administrateur de sécurité avec la relation d'héritage « ≥ » est : phd≥student, master≥student, et licence≥student.

En appliquant le profil RBAC pour cet exemple (Figure 27), chaque rôle a son RPS. Puis, il y a un PPS pour les étudiants qui contient la permission d'accéder au site web « www.jobsUPS.fr » et ce PPS est référencé dans la RPS de l'étudiant. La hiérarchie de rôles est spécifiée par des références ou « Xpath links » entre PPS. Selon l'exemple donné, toutes les permissions attribuées au rôle « étudiant » sont également affectés au rôle « phd » ou « master » ou « licence », car le rôle « phd » ou « master » ou « licence » sont supérieurs au rôle « étudiant ».

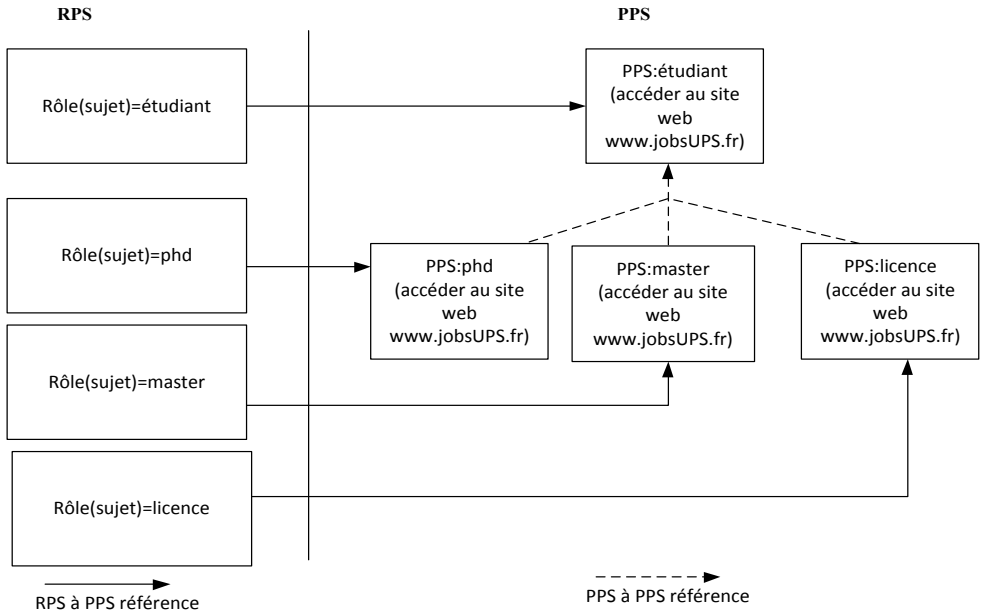

Figure 27 La hiérarchie des rôles avec le profil RBAC

L'idée de ce profil est de rendre XACML capable de mettre en œuvre des politiques RBAC sans pour autant le modifier. Dans des situations très simples comme celle décrite dans notre exemple où nous ne considérons que trois rôles et une règle, de telles politiques restent compréhensibles et gérables. Mais, cette approche n'est pas efficace dans le cas général, car cela peut conduire à des politiques extrêmement complexes. Le problème vient du nombre de liens « XPath » ou références qui augmentent avec le nombre de rôles et les autorisations attribuées aux rôles. Prenons par exemple, le cas du modèle LRBAC qui étend le modèle RBAC avec la notion de localisation logique du sujet. Dans LRBAC, les rôles spatiaux, sont un produit cartésien des rôles et des domaines de localisation logique, pour attribuer des autorisations. Avec le profil RBAC pour représenter ce modèle, il devrait y avoir « le nombre de rôles » x «le nombre de domaines de localisation logique » des ensembles RPSs et PPSs. Nous avons décrit un tel scénario dans [Cheaito et al. 2010a]. Cette complexité vient du fait que le profil ne veut pas ajouter de nouveaux types de données et fonctions. Ceci impose aussi à l'administrateur de sécurité d'être à la fois expert dans le langage XACML et XML. Par conséquent, cette approche permet d'étendre le langage XACML sans modification du PDP. Par contre, elle entraîne une complexité dans l'écriture et la compréhension des politiques.

Le profil GeoXACML [GeoXACML 2011], défini pour prendre en compte les données géographiques, utilise une autre approche pour étendre le langage. Il ajoute de nouveaux types de données et fonctions pour manipuler des données géographiques dans les politiques XACML. Par exemple, les données géographiques avec l'utilisation d'un dispositif GPS (Global Positiong system). La Figure 28 montre une règle d'une politique XACML spécifiée avec l'aide du profil GeoXACML pour ajouter des informations géographiques. Cette règle contient un type de données « geometry » non défini dans le standard XACML mais inclus dans le profil GeoXACML et connu par l'URN (Uniform Resource Name) « urn :ogc :def :dataType :geoxacml :1.0 :geometry ». La fonction « geomtery-contains » du profil GeoXACML est liée à ce type de données. Pour implémenter cette règle dans un système d'autorisation qui respecte le standard XACML, il faut ajouter du code supplémentaire au niveau du PDP afin de supporter ces nouvelles informations. Cette approche permet d'étendre le langage XACML de façon plus pertinente. Par contre, il nécessite que le PDP soit modifié pour prendre en compte ces nouvelles informations.

```
<Rule RuleId="TestRule" Effect="Permit">
<Condition FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-contains"">
<ResourceAttributeDesignator AttributeId="requested-area"
DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry"/>
<AttributeValue DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry">
<gml:Polygon gid="North and South America"
srsName=" http://www.opengis.net/gml/srs/epsg.xml#4326"
xmlns:gml="http://www.opengis.net/gml">
  <gml:outerBoundaryIs><gml:LinearRing>
   <gml:coordinates cs="," ts=" ">
-180,60 -180,47 -137,55 -125,35 -110,17 -80,5 -87,-5
-74,2 -78,-53 -67,-58 -60,-48 -35,-23 -27,-3 -55,12 -60,22
-75,30 -67,42 -50,45 -60,85 -60,85 -85,85 -120,80 -130,75
-160,75 -168,70 -180,60
</gml:coordinates>
  </gml:LinearRing></gml:outerBoundaryIs >
</gml:Polygon>
</AttributeValue>
</Condition>
</Rule>
```

Figure 28 Exemple d'une règle GeoXACML

4.4.1 Etude des implémentations XACML

Plusieurs implémentations existent qui sont toutes conformes au standard XACML. Elles supportent tous les types de données et les fonctions standards de XACML. Nous citons ici les

implémentations suivantes : SUNXACML [SUNXACML 2011], Java Enterprise XACML [EXACML 2011], extensible XACML [Cheaito et al. 2010a] et HERASAF [HERASAF 2011].

SUNXACML fournit une implémentation open source du standard XACML version 2.0 écrite en langage de programmation JAVA. Cette implémentation fournit un support pour la majorité des caractéristiques de XACML (analyse de politiques et de requêtes, détermination de l'applicabilité d'une politique, évaluation de requêtes auprès de politiques). En plus, une API (Application Programming Interfaces) Java permet de programmer des PEPs ou PIPs capables d'interagir avec le PDP, et d'ajouter de nouveaux types de données/fonctions au niveau du PDP. Par contre, écrire un PIP afin de supporter des informations technologiques complémentaires (par exemple des informations dans une base des données) nécessite d'arrêter le système et modifier le code source. La modification est faite en utilisant l'API java pour ajouter des codes supplémentaires au PDP ou changer le code du PDP. SUNXACML s'adapte statiquement pour effectuer le changement du système venant des variations de l'environnement.

Java Enterprise XACML est une implémentation Java du standard XACML version 2.0 indépendant des autres implémentations de XACML. Elle fournit des extensions avec des API Java qui permettent d'ajouter des fonctions non standards, et des PIPs pour récupérer des attributs sur des systèmes externes sans modifier la structure interne du PDP. Cette implémentation ne supporte pas l'ajout des nouveaux types de données pour répondre à des contextes donnés. Par exemple, il est impossible d'implémenter les types des données définis dans GeoXACML. De plus, comme SunXACML, il nécessite d'arrêter le système et modifier le code source du PDP (c.à.d. adaptation statique) pour chaque nouvelle fonctionnalité à ajouter.

Le cadre extensible de XACML que nous avons proposé dans un premier temps est une implémentation d'un moteur de décisions d'autorisation XACML ou PDP extensible. Il comprend un composant de base ou PDP qui implémente le standard XACML et qui peut supporter des modules supplémentaires pour représenter des besoins de sécurité qui impliquent des extensions du langage de politiques XACML (tels que des nouveaux types de données et fonctions non standards). La solution proposée permet d'étendre l'architecture sans pour autant avoir besoin de modifier l'ensemble du système. Mais, le système doit être redémarré chaque fois qu'un nouveau module doit être inclus (par exemple un nouveau module PIP pour récupérer des informations complémentaires), ce qui implique que le système s'adapte statiquement pour effectuer chaque changement du système.

HERASAF comprend un module de base ou PDP qui implémente le standard XACML et qui peut supporter des modules supplémentaires. Il permet d'ajouter dynamiquement des modules possédant des fichiers de configuration. Par exemple, un nouveau type de données non standard à spécifier dans une politique XACML, nécessite l'écriture d'un nouveau module et l'ajout dynamique au système, afin que cette extension soit prise en compte.

Le cadre COCOA [Chin et al. 2010] est une implémentation de l'architecture de XACML pour l'environnement ubiquitaire. Il a été conçu pour prendre en compte des informations contextuelles dans des requêtes de sujets qui sont de nature dynamique (comme localisation de l'utilisateur, temps d'accès, niveau de confiance...). Il étend l'architecture de XACML par l'entité CSS « Context session service», et remplace les deux entités PIP et PDP par « CTX PIP » et « CTX PDP ». L'entité « CTX PIP » extrait les attributs contextuels et les données du sujet, et vérifie leur authenticité. Si la vérification est positive, il passe les valeurs d'attributs contextuels du sujet au Policy Decision Point contextuel (CTX PDP). Le CTX PDP utilise des attributs contextuels du sujet en conjonction avec d'autres informations fournies par d'autres PEPs (par exemple le nom du sujet et l'opération demandée) pour prendre une décision d'autorisation. Cette décision est retournée au PEP qui applique la décision. Dans le même temps, le CTX PDP envoie des détails sur le sujet (identité du sujet, URI des attributs contextuels, décision d'autorisation, horodatage, et ressources demandées) à l'entité CSS qui prend les attributs contextuels du sujet et émet un message de souscription. Le CSS est notifié sur toute modification des données contextuelles liées au sujet en recevant un message de notification, ensuite il demande une révision des décisions d'autorisation auprès de l'entité CTX PDP. Le cadre

COCOA est fait pour que XACML puisse répondre à un contexte donné qui est l'environnement ubiquitaire. Le CTX PIP s'adapte seulement à des informations complémentaires qui sont des assertions contextuelles sur des requêtes de sujets. La politique de COCOA est le langage standard de XACML. Ainsi, les contraintes contextuelles sont représentées comme une condition dans une règle XACML utilisant des types de données et fonctions standards. Cela signifie que le langage de politiques COCOA est limité car il est basé sur des attributs standards de XACML. Ainsi, le CTX PDP est spécifique à un contexte donné qui est de traiter des informations contextuelles dans des requêtes de sujets. Pour traiter d'autres besoins tels que le rôle, le système doit être arrêté (adaptation statique) pour soit ajouter un PDP pour rôle ou modifier le CTX PDP.

Le problème majeur lié à l'implémentation d'extensions du langage XACML par de nouveaux types de données ou fonctions est que le PDP doit posséder une partie de code qui implémente ces extensions. Même si le moteur de décision supporte l'ajout de module dynamiquement, il n'existe aucun lien entre l'extension du langage de politique et l'extension du PDP afin de garantir que le système soit consistent. D'un autre côté, un PDP peut contenir des parties de code/modules d'extension inutiles (qui implémentent des fonctions ou types de données non utilisés dans la politique) qui diminueront sa performance. Il manque un guide qui explique comment créer des extensions et les utiliser dans le système efficacement.

4.4.2 Etude des éditeurs de politiques XACML

Différents travaux menés par la communauté de recherche traitent de la spécification des politiques XACML. Premièrement, les politiques XACML étant des documents XML, tout éditeur XML comme XMLPAD [XMLPAD 2011] peut être utilisé. XML PAD est un éditeur pour les politiques XACML qui vérifie la structure XML de politiques en important le schéma de XACML. Il propose toutes les balises XML (Figure 29) possibles pour spécifier de politiques XACML conformes au schéma XACML importé dans l'éditeur. Cet éditeur permet de spécifier tous les attributs standards (URIs, type de données, fonctions...).

Mais, il ne permet pas de spécifier de nouveaux attributs non définis dans le schéma XML comme les attributs GeoXACML et qui seront nécessaires pour que le système d'autorisation prenne des décisions d'autorisation. XMLPAD étant un éditeur pur XML, il prend en compte tous les changements qui peuvent exister dans tout schéma XML importé. Cependant, il ne fournit pas de mécanisme qui aide à écrire des politiques XACML.

Figure 29 Une capture d'écran de l'éditeur XML PAD

L'éditeur proposé par UMU (University of Murcia) [UMU 2011] est dédié pour faciliter la tâche de spécification de politiques XACML conformes au standard version 2.0 et 3.0. Le but

principal de ce projet est de développer un éditeur graphique de définition des politiques mises en œuvre en langage Java, cachant tous les détails XML. Les éléments ou balises XML de la politique XACML sont représentés par une arborescence graphique. Dans chaque élément, l'administrateur peut choisir une liste de valeurs conformes au standard XACML ou peut entrer sa propre valeur qui n'existe pas dans cette liste. Cet éditeur propose la manipulation des politiques avec des attributs standards ou des attributs non-standards si nécessaire.

La Figure 30 montre comment spécifier une politique XAMCL avec cet éditeur. Cet éditeur facilite l'écriture de politique XACML. Cependant, lorsque des types de données ou fonctions non standards sont introduits, il n'y a pas de moyen de vérifier s'il s'agit d'une erreur de frappe ou s'il s'agit d'un nouveau type de données ou fonction.

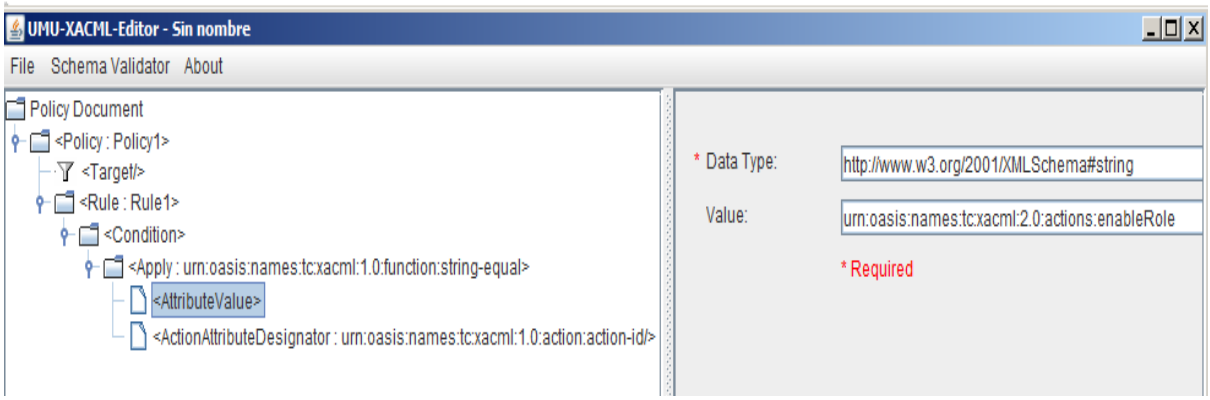

Figure 30 Une capture d'écran de l'éditeur proposé par UMU

Une nouvelle approche très prometteuse a été développée par l'Université d'Ottawa [Stepien et al. 2010]. Leur idée est d'offrir un éditeur de politiques pour des utilisateurs non-techniciens. Ils s'attachent à créer une notation non-technique proche du langage naturel qui rend les politiques XACML beaucoup plus lisibles par les êtres humains. Cela, permet de réduire le nombre de politiques et les règles, une meilleure évolutivité des outils d'analyse (par exemple détecter les conflits entre deux politiques). De plus, cet éditeur facilite l'utilisation des conditions complexes dans des règles de politiques XACML en augmentant leur lisibilité. Par exemple, la spécification d'une condition qui déclare « la nourriture peut être achetée uniquement le lundi ou le mercredi tandis que les billets de voyage peuvent être achetés uniquement pendant les week-ends » peut être spécifiée avec cet éditeur (Figure 31) dans une seule règle appelée « rule1 » qui permet cette condition au lieu de deux règles en utilisant une condition complexe.

```
Rule: rule_1 : Permit
condition:
Merchandise is food
    provided that DayOfTheWeek is one of
Monday, Wednesday,
or
  Merchandise is travel
    provided that DayOfTheWeek is one of
 Saturday, Sunday
```

Figure 31 Exemple d'une condition

Ainsi, il permet de spécifier des politiques avec des types de données et fonctions non-standards et générer en sortie un fichier XACML prêt à déployer dans un système d'autorisation de type XACML. L'éditeur permet aux administrateurs de sécurité de définir des fonctions non standards et de préciser des types de données dans un fichier de configuration (par exemple week est une variable qui peut prendre comme valeur Monday, Tuesday, etc.). Cependant, la déclaration des fonctions non standards n'implique pas que le PDP les implémente et les types de données construits dans le fichier de configuration restent simples.

Le problème majeur de ces éditeurs de politique est qu'ils ne garantissent pas à l'administrateur que la politique générée soit interprétée ou non par le PDP cible. De plus, les outils qui facilitent l'écriture de règles sont limités par exemple, à la vérification de la syntaxe d'une expression qui comprend des fonctions non standard.

4.5 Conclusion

Dans ce chapitre, nous avons présenté un bilan sur les principaux systèmes d'autorisation qui offrent des mécanismes d'adaptabilité. Nous les avons classés en deux groupes : approche d'intégration de plusieurs PDPs, et l'approche où le langage ou le modèle de politiques est générique. Nous avons analysé la capacité d'adaptation de ces systèmes d'autorisation existants afin de répondre aux différentes situations (différentes organisations, différents besoins d'autorisation, différents environnements informatiques) par rapport à notre étude effectuée dans le chapitre 3.

Ainsi, nous avons analysé plus en détails le standard OASIS XACML qui nativement intègre la notion d'extensibilité et qui possède un nombre important d'implémentations. Nous avons étudié la capacité d'adaptation du langage de politiques, des systèmes d'autorisation de type XACML, et des éditeurs de politiques XACML. Nous avons choisi XACML comme technologie cible.

Dans le chapitre suivant nous présentons en détails les besoins d'adaptabilité pour implémenter un écosystème qui respecte la philosophie de XACML.

Chapitre5. Analyse des besoins de XACML

5.1 Introduction

En général, le système d'autorisation a pour fonction d'évaluer la capacité de donner (autoriser ou non) à des sujets ou demandeurs d'accès (utilisateur, programme en cours d'exécution, calculateur, fichier etc.) la possibilité de faire des activités ou actions sur un ou plusieurs objets (fichiers, dossiers, bâtiments, applications informatiques spécifiques bases de données, serveurs…). Cependant, ce système fournit :

- Un langage de politiques pour la spécification des politiques d'autorisation
- Un moteur de décisions d'autorisation pour interpréter les politiques et prendre la décision sur des requêtes des entités demandeurs d'accès fondées sur ces politiques
- Parfois, d'un mécanisme d'application ou de mise en œuvre de décisions d'autorisation (par exemple PEP pour système de gestion à base de politiques) qui s'assure de la bonne exécution des décisions prises

Comme nous l'avons signalé, l'évolution des exigences au sein des organisations impose que ce système soit adaptable pour répondre aux variations des exigences de l'environnement. Rappelons que dans un tel système, l'environnement géré (E) se reflète par :

- Environnement des besoins de sécurité (EBSEC) : Il en est ainsi des exigences venant des modèles formels de contrôle d'accès tels que les rôles, temps, localisations
- Environnement technologique complémentaire (ECOMP) Un choix technologique d'implémentation d'une base de données MySQL ou un annuaire LDAP pour stocker les comptes des utilisateurs donne lieu à des contraintes spécifiques
- Environnement technologique contrôlé (ECTRL) comme utiliser internet pour faire une requête d'accès au site web

Rappelons, qu'une variation de l'environnement géré (Δ_E) implique un changement d'une partie ou toutes parties du système (Δ_S).Le résultat (S') conduit en pratique soit à la mise en œuvre d'un nouveau système d'autorisation à contrôler (adaptation statique), soit sans arrêter le système d'autorisation existant (adaptation dynamique). Le résultat (S') doit bien sûr répondre idéalement au nouvel environnement (E') causé par la variation de l'environnement (Δ_E). La conduite du changement dans un système d'autorisation adaptable doit être faite à partir de la connaissance de la variation de l'environnement ($\Delta_E=\Delta_{EBSEC}\cup\Delta_{ECOMP}\cup\Delta_{ECTRL}$), pour déterminer les changements à apporter au système.

Pour pouvoir construire un système d'autorisation adaptable (capable de réaliser le changement du système (Δ_S)), il nous faut tout d'abord préciser le contexte et « savoir » qui utilise le système d'autorisation afin d'en déduire les exigences qu'un système d'autorisation devrait satisfaire. Nous avançons les hypothèses suivantes à la base des tâches qu'un administrateur de sécurité au sein de l'organisation doit accomplir dans un système d'autorisation :

- **Hypothèse 1**. Les systèmes d'autorisation sont gérés par des administrateurs de sécurité. Cela implique que la perception du système peut être différente d'une organisation à une autre.
- **Hypothèse 2**. Les administrateurs de sécurité ont ou bien peuvent avoir une certaine connaissance des techniques de programmation, mais ce n'est pas le cœur de leurs tâches.

Ainsi dans de nombreuses situations, ils ne sont pas supposés être des programmeurs (chevronnés ou non) d'applications.

- **Hypothèse 3**. Les administrateurs de sécurité peuvent avoir une bonne connaissance des concepts de sécurité. Surtout, ils ont la capacité de comprendre les concepts de sécurité introduits par des modèles de contrôle d'accès. Par exemple, les administrateurs de sécurité peuvent comprendre le concept ou la notion de rôle et comment l'utiliser dans leur organisation.
- **Hypothèse 4**. Les administrateurs de sécurité ne sont pas supposés avoir de bonnes connaissances en mathématiques. En particulier, l'exercice de cette fonction n'est pas spécifiquement lié à des facultés particulières en termes de compréhension de structures et propriétés mathématiques, de relations d'ordre introduites par les modèles de contrôle d'accès. Par exemple, les administrateurs de sécurité peuvent appréhender ce que recouvre la hiérarchie des rôles, sans pour autant être férus dans la compréhension et la manipulation des structures et propriétés mathématiques de la relation d'ordre de rôle qui est un ensemble d'ordre partiel (POSET).

En admettant les hypothèses données ci-dessus, nous pouvons en déduire quelques exigences sur ce qu'un système d'autorisation adaptable devrait satisfaire :

- **Exigence 1**. Un système d'autorisation devrait pouvoir être intégré à une infrastructure existante, sans avoir systématiquement besoin de repenser ou reproduire l'ensemble du système.
- **Exigence 2**. Un système d'autorisation devrait être capable d'interpréter et d'exécuter n'importe quelle politique d'autorisation. Il est important de satisfaire la diversité des applications/services et exigences de toute organisation.
- **Exigence 3**. Un système d'autorisation devrait fournir des fonctionnalités de configuration ou outils qui permettent aux administrateurs de sécurité de l'adapter à leurs besoins spécifiques, sans programmer quoi que ce soit.

5.2 Pourquoi avoir choisi XACML

Pour satisfaire les exigences et les hypothèses citées ci-dessus, nous avons choisi XACML comme technologie cible. XACML offre deux avantages : un langage de politiques extensible et une architecture flexible.

5.2.1 Langage de politiques XACML extensible

Le langage de politiques XACML est basé sur l'approche de contrôle d'accès à base d'attributs (ABAC). ABAC permet de spécifier des permissions par rapport à toute caractéristique liée à la sécurité des sujets, des actions, des ressources et de l'environnement. Cette approche nous semble appropriée pour l'expressivité de la spécification des politiques d'autorisation.

Par exemple, considérons une politique d'autorisation écrite de manière formelle comme suit : « *tous les étudiants localisés dans le bâtiment IRIT peuvent lire le fichier examen.html à partir de 10 :00 am* ». Cette politique contient des exigences de sécurité diverses figurant en totalité ou sous forme de notions dans différents modèles formels de contrôle d'accès : le rôle, la localisation physique, et le temps d'accès. Avec ABAC, une telle politique d'autorisation est spécifiée formellement en utilisant des attributs de la manière suivante : *=(rôle(sujet), « étudiant »)* ∧ *=(localisation(sujet), « IRIT »)* ∧*=(nom(action), « lire »)* ∧ *=(nom(ressource), « examen.html »)* ∧

≥*(temps(environnement), « 10 am »)* . Où, le *sujet, l'action, la ressource et l'environnement* sont des entités qui représentent des caractéristiques pour des attributs. Les exigences de sécurité (*rôle, nom, localisation et temps*) sont exprimées dans la politique ABAC avec des attributs pour ces entités. Les relations (*=, et ≥)* sont soit des relations entre les attributs ou soit des relations entre un attribut et un ou plusieurs valeurs d'attributs. Alors, ABAC donne les moyens d'écrire formellement des règles à base des attributs, intégrant une diversité d'informations de sécurité et associées à des exigences propres aux organisations. Cette approche est considérée comme une solution de gestion de politiques d'autorisation de manière générique. Donc, le langage de politiques XACML qui est de type ABAC peut être considéré comme une solution générique de spécification de politiques d'autorisation.

Cependant, le langage XACML fournit des types de données et des fonctions standards (par exemple entier, chaîne de caractères, booléen...) pour manipuler les attributs ABAC. Dans XACML, un attribut pour une entité ABAC est un triplé <*attr-id, attr-dt, attr-value*>, où « *attr-id* » est l'identificateur de l'attribut, « *attr-dt* » est le type de donnée de l'attribut, et « *attr-valeur* » est la valeur de l'attribut. Par exemple, <*name, string, access*>*(subject)* désigne l'attribut pour le sujet dont l'id est « *name* », de type de données « *string* » et une valeur «*access*». Un attribut peut avoir plusieurs valeurs. Les relations des attributs sont obtenues par des fonctions de comparaison. Par exemple, la fonction "=" pour le type de données « string » est une fonction de comparaison entre les caractères de deux chaînes de caractères. L'exemple dans la Figure 32 est écrit dans le langage de politiques XACML avec l'aide de types de données et fonctions standards.

```
<Apply FunctionId: urn:xacml:and>
<Apply FunctionId: urn:xacml:string-equals>
<AttributeValue Datatype: string>étudiant</AttributeValue>
<SubjectAttributeDesignator AttributeId: rôle Datatype: string/>
</Apply>
<Apply FunctionId: urn:xacml:string-equals>
<AttributeValue Datatype: string>lire</AttributeValue>
<ActionAttributeDesignator AttributeId: nom Datatype: string/>
</Apply>
<Apply FunctionId: urn:xacml:string-equals>
<AttributeValue Datatype: string>examen.html</AttributeValue>
<ResourceAttributeDesignator AttributeId: nom Datatype: string/>
</Apply>
<Apply FunctionId: urn:xacml:string-greater-than>
<AttributeValue Datatype: string>10am</AttributeValue>
<EnvironementAttributeDesignator AttributeId: temps Datatype: string/>
</Apply>
</Apply>
```

Figure 32 Exemple d'une politique XACML

Dans le langage de politiques XACML, les fonctions et les types de données sont connus par leurs URN (Uniform Resource Name) comme des identificateurs. La fonction « = » (Figure 32) est connue par l'URN « **urn :xacml :string-equals** ». Cette fonction prend deux entrées : une variable dont la valeur peut exister dans une requête XACML ou dans une technologie complémentaire, et une ou plusieurs valeurs d'attributs. La fonction « **urn :xacml :string-equals** » est l'URN pour type de données standard « **string** ». Les valeurs des attributs XACML sont des données qui décrivent les caractéristiques des entités sujets, ressources, actions et environnement.

De plus, bien que leur existence soit considérable, il n'est pas possible avec les types de données et les fonctions standards de spécifier toutes les contraintes venant des toutes les exigences des organisations. Par contre, le fait que le langage XACML soit extensible, est une caractéristique importante pour les travaux que nous avons entrepris. Le langage XACML reconnaît la définition de nouveaux types de données et des fonctions non standards [Cheaito et al. 2010b]. L'extensibilité confère à ce langage une certaine flexibilité pour la spécification de politiques. Il est donc possible dans XACML d'ajouter de nouvelles extensions pour exprimer de nouvelles exigences dans des politiques. Cependant, cette extensibilité du langage pose un problème de déploiement du système d'autorisation XACML. En effet, la fonction de prise de décisions d'autorisation ne peut pas toujours conduire à une exécution d'une politique contenant des extensions. Il peut être nécessaire de vérifier la

présence ou d'introduire du code supplémentaire dans le moteur d'autorisation. La conception d'une architecture flexible qui permet d'avoir une implémentation extensible, se positionne comme une solution candidate pour résoudre ce problème.

5.2.2 Architecture flexible

L'approche de gestion à base de politiques (PBM) est une approche reconnue maintenant pour la gestion des systèmes complexes. PBM dont les motivations premières sont la flexibilité et l'adaptation de l'architecture d'un système à l'infrastructure existante et à la gestion du changement semble adaptée à ce service d'autorisation [Sloman et Lupu 2002]. Les systèmes de gestion à base de politiques rendent la gestion plus dynamique et évolutive. Avec une telle approche, deux actions doivent être réalisées :

- Une décision d'autorisation doit être prise concernant les requêtes effectuées par des demandeurs d'accès après la consultation des politiques
- La décision doit être appliquée ou mise en œuvre

L'architecture d'un PBM peut comprendre les trois entités qui sont **(**Figure 33) :

- Le Policy Decision Point (PDP) qui interprète la politique et prend des décisions de gestion par rapport à elle
- Le Policy Enforcement Point (PEP) qui rend exécutables les décisions du PDP par le système contrôlé. C'est le PEP qui réalise techniquement l'attribution ou le refus d'une demande d'accès à une ressource
- Le Policy Information Point (PIP) (étape 3 dans la Figure 33) qui est responsable de récupérer des informations complémentaires dans des endroits spécifiques (fichiers, bases de données, LDAP...) non inclus dans des requêtes. Il est un composant qui agit comme un serveur d'annuaire qui stocke les valeurs d'attributs et les rend disponibles pour les processus de décisions d'autorisation par le PDP

Dans cette approche l'externalisation du processus de décision en dehors de l'application gérée permet d'envisager l'utilisation d'un langage de politique unique pour contrôler les différents applications/systèmes dans les systèmes de type PBM. Dans cette approche, les politiques représentent une logique *externalisée* pour contrôler le comportement d'un système géré sans avoir à le recoder ou le stopper [Agrawal et al. 2005]. A cause de la diversité des ressources gérées et les variations des exigences des organisations, il semble naturel de favoriser le développement de langages de politiques qui sont à la fois extensibles et indépendants du système géré.

XACML est basé sur l'architecture PBM et les interactions entre entités PDPs, PIPs et PEPs suivent le modèle d'autorisation *Pull*. Cette architecture donne l'avantage d'ajouter/supprimer une politique avec ou sans extension sans pour autant arrêter ou stopper le système (adaptation dynamique). Cependant, pour autant que nous le sachions [Cheaito et al. 2010b], aucune solution ne peut garantir le lien de dépendance entre les politiques, les extensions apportées au langage, et le moteur de décisions de politiques ou PDP dans un système d'autorisation.

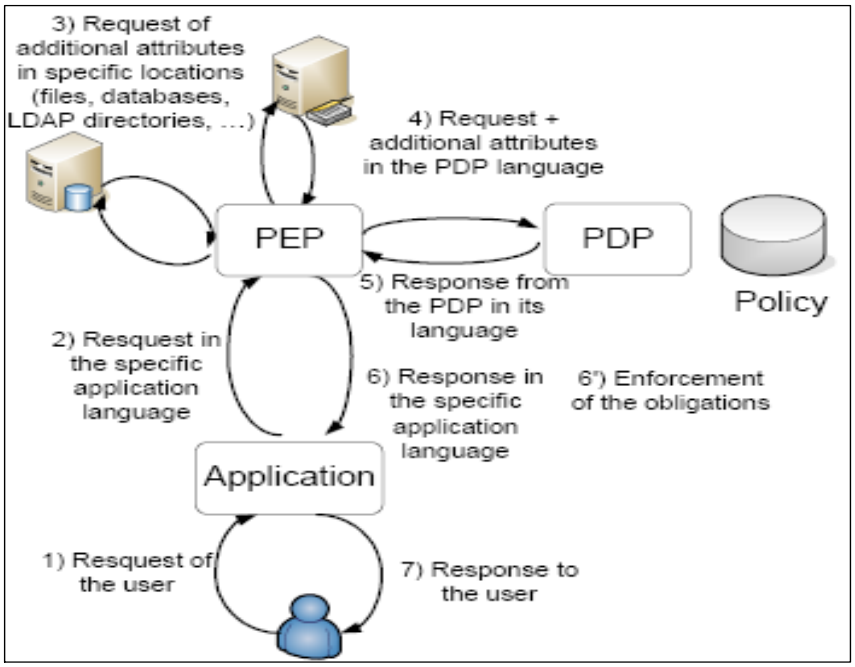

Figure 33 Architecture PBM

5.3 Analyse de XACML dans un environnement réel

Dans cette section, nous présentons un exemple pour montrer les questions liées à la cohérence du langage de politiques et le moteur de décision de politiques dans un système d'autorisation XACML.

5.3.1 XACML en pratique

Nous présentons un exemple pris de [Laborde et al. 2010] afin de comprendre la problématique liée au déploiement d'un système d'autorisation XACML dans un environnement réel. Cet exemple se focalise uniquement sur le côté moteur de décisions de politiques ou PDP.

5.3.1.1 Scenario Geo-XACML

Considérons une entreprise, appelée « MY-ORGANIZATION », qui décide d'installer un service web cartographique. Ce service peut fournir des cartes insérées par l'entreprise et en particulier des vues détaillées de ses bâtiments. Cette entreprise désire restreindre l'accès à ces vues détaillées uniquement à ses employés. Tous les comptes utilisateurs sont déjà stockés dans une base de données « MySQL ». Les employeurs qui travaillent pour cette entreprise peuvent provenir de différentes entreprises et en particulier de sous-traitants « sub-contractors ». Pour différencier les employés, les comptes utilisateurs possèdent un champ appelé « organization-name » qui contient le nom de l'entreprise d'origine.

5.3.1.2 Application du scenario dans XACML

Pour construire sa politique, l'administrateur de sécurité désire utiliser le profil « GeoXACML », dont il connait l'existence (développé par l'OGC [GeoXACML 2011]). En effet, ce standard propose des types de données et des fonctions pour manipuler des données géographiques dans des politiques XACML. La politique de sécurité du scenario proposé (Figure 34) devient la

suivante : « Si l'attribut de l'utilisateur « organization-name » est égal à « my-organization » et que la zone géographique demandée (attribut de la ressource « requested-area ») se trouve dans la zone « MY-ORG-AREA » alors accepter ; sinon refuser l'accès ».

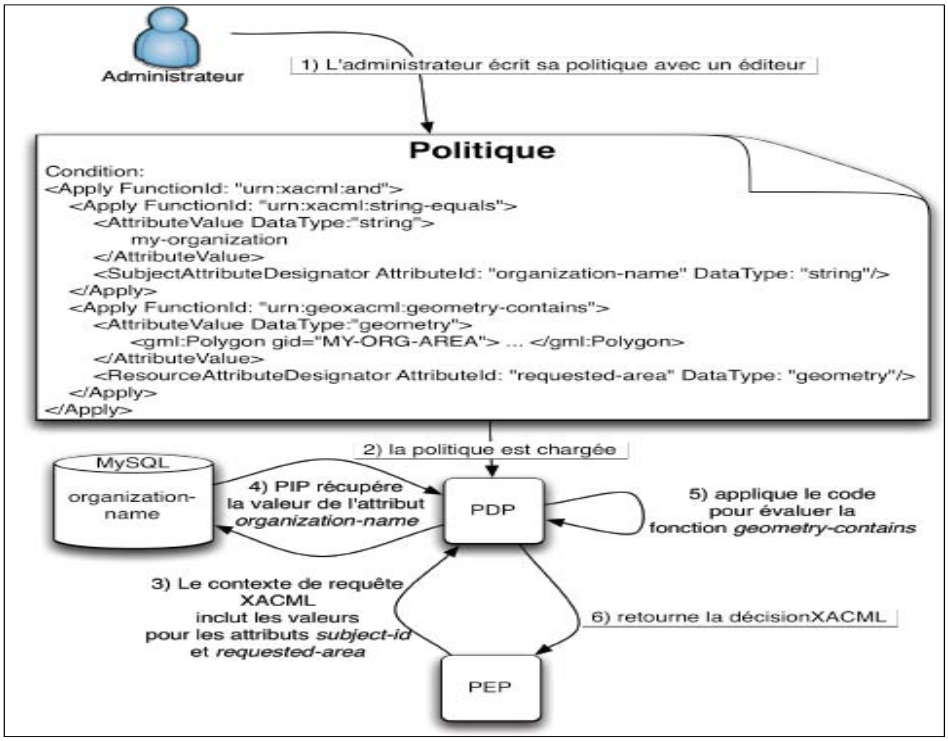

Figure 34 Exemple d'utilisation de XACML

Ainsi, en supposant que le système d'autorisation implémente le profil « GeoXACML », l'administrateur de sécurité écrit cette politique avec le langage XACML (une version simplifiée est donnée dans la Figure 34). Le processus d'exécution est alors le suivant. L'administrateur écrit sa politique en XACML et la charge dans le moteur de décision de politiques ou PDP via le Policy Administration Point (PAP) (étapes 1 et 2 dans Figure 34).

Quand un utilisateur tente d'accéder à une carte, le PEP récupère la requête et envoie une requête XACML, appelée contexte de requête, au PDP (étape 3 dans dans Figure 34). Cette requête contient les valeurs pour les attributs « subject-id » et « requested-area ». Dans notre exemple, l'attribut « organization-name » n'est pas envoyé par le PEP.

Lorsque le PDP évalue la politique, il doit faire appel au module Policy Information Point (le PIP est l'entité XACML chargée de récupérer des valeurs d'attributs non présents dans le contexte de requête du PEP) pour l'obtenir depuis la base de données MySQL (étape 4 dans Figure 34). Ceci implique que l'implémentation XACML doit intégrer un code PIP implémentant les requêtes « MySQL » et qu'il possède la configuration nécessaire pour accéder au serveur « MySQL » (comme adresse du serveur, login, mot de passe, schéma de la base ou la requête SQL). Ensuite, le PDP évalue la fonction « geometry-contains ». Ceci implique que le PDP doit avoir aussi le code qui implémente cette fonction, et ainsi le code pour le type de données associé à cette fonction (étape 5 dans Figure 34**)**. Finalement, le PDP retourne sa décision au PEP (étape 6 dans Figure 34).

Cet exemple montre plusieurs aspects dont au moins les deux suivants :

1. Etendre le langage XACML n'est pas si simple. En effet, les notations, au format XML devraient être accompagnées du code permettant au PDP d'exécuter ces écrits (dans le cas présent un nouveau code du type de données GeoXACML et sa fonction et un nouveau code du PIP MySQL). Les changements du système sont causés par des variations du langage de politique XACML. Ici, le changement de politique est causé par

l'extension du langage de politiques par un nouveau type de données et/ou un nouvel attribut PIP. En effet, cette situation implique d'effectuer des changements sur un moteur de décisions de politiques ou PDP avec des codes supplémentaires pour prendre en compte le nouveau type de données et/ou le nouvel attribut PIP.

2. Les extensions du langage doivent elles mêmes être spécifiques aux besoins du langage de politiques XACML. Par exemple si un nouveau besoin de sécurité à prendre dans le langage de politiques conduit à ajouter un nouvel attribut. Dans cette situation, le PDP doit prendre en compte ce nouvel attribut durant le processus de prise de décision d'autorisation.

5.4 Définition des besoins pour un moteur de décision XACML extensible

La diversité des ressources à gérer mais aussi des besoins des organisations impose que les langages de politique soient le plus indépendant possible du système géré. Ils doivent être également extensibles afin de pouvoir prendre en compte toute contrainte spécifique à une organisation particulière. Dans le domaine de la sécurité, XACML propose un langage où les permissions sont décrites en termes d'attributs sur les sujets, les actions, les ressources et l'environnement. Dans ce langage de contrôle d'accès, toute nouvelle caractéristique considérée comme importante pour la sécurité est représentée par un attribut. XACML fournit des types de données et des fonctions standards pour manipuler ces attributs dans une politique. De plus, ce langage donne la possibilité d'être étendu par de nouveaux attributs ou types de données ou fonctions si besoin est.

De notre point de vue, la caractéristique d'extensibilité du langage doit également s'accompagner d'une extensibilité du moteur de décisions d'autorisation XACML (Figure 35). Quand un administrateur de sécurité écrit une politique avec l'aide d'un éditeur, il peut souhaiter inclure un nouveau besoin c.à.d. un nouvel attribut ou type de données ou fonction pour les besoins de son organisation. Dans ce cas, il consulte une base de données de modules (un module peut comporter des codes et des fichiers de configuration) qui contient le code associé à l'extension ou l'exigence à spécifier dans sa politique. Il choisit les modules qui sont spécifiques à sa politique. Ensuite, il charge la politique avec ces modules spécifiques au moteur de décisions d'autorisation XACML qui implémente à l'origine le seul standard XACML. Ainsi, le moteur est capable d'implémenter en plus du standard XACML les extensions non-standards nécessaires aux politiques, car chaque module chargé vient avec son propre code.

Notre idée est inspirée d'Apache [Kew 2007] qui est composé d'un module de base qui implémente les fonctionnalités les plus élémentaires nécessaires à un serveur web et un ensemble de modules standard pour le traitement d'une requête HTTP. Par exemple, quand un administrateur de sécurité veut authentifier les utilisateurs avec un serveur LDAP, il suffit d'ajouter le module « modauth-ldap » et de le configurer. Cette fonctionnalité est intéressante, car reprise dans notre contexte, elle permet de construire un moteur de décisions d'autorisation flexible et extensible. Par exemple, dans un scenario GeoXACML, la politique GeoXACML doit être chargée avec ses modules spécifiques pour être exécutée par le moteur de décisions d'autorisation ; les modules nécessaires sont : le module « MySQL » PIP qui contient le code pour les attributs MySQL , et la configuration pour se connecter à une base de données « MySQL » (tel que adresse IP du serveur, login/mot-de-passe...), le module « geometry » qui contient les codes permettant d'évaluer le type de données « geometry » et la fonction associée au type de données « geometry-contains », et le module pour « polygon » qui contient le code permettant d'évaluer le type de donnée « polygon ».

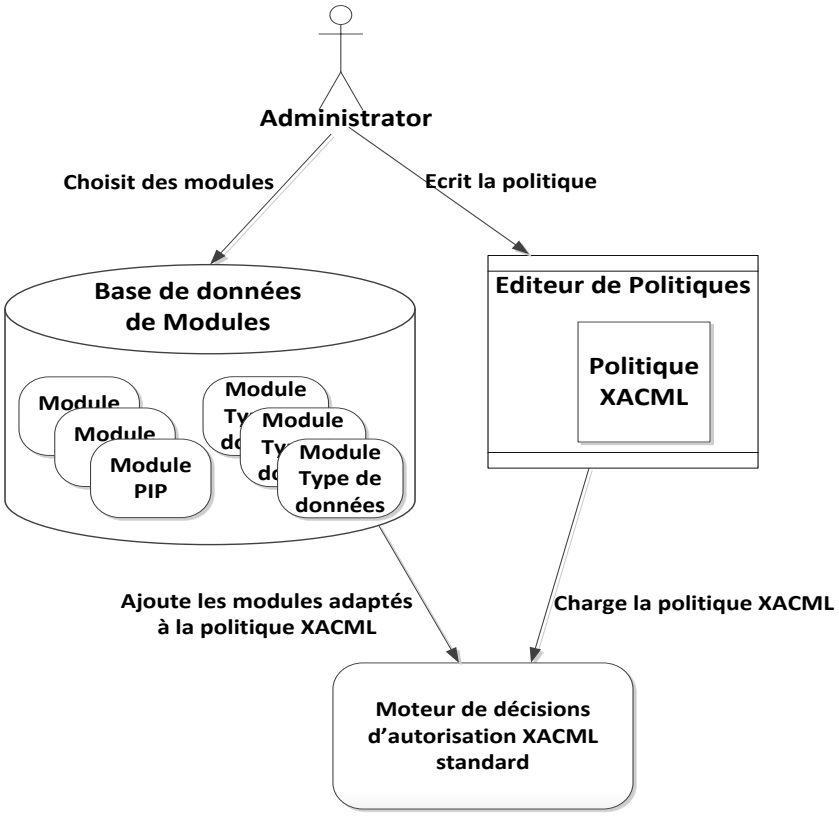

Figure 35 Notre vision d'un moteur de décisions d'autorisation XACML extensible

Cependant, de nombreuses implémentations du moteur de décisions d'autorisation XACML existent. Des implémentations existantes telles que SUNXACML [SUNXACML 2011], Enterprise Java XACML [EXACML 2011], XACMLight [XACMLLight 2011] sont conformes au standard XACML c.à.d. qu'elles soutiennent tous les types de données et les fonctions standards de XACML. Cependant, une extension du langage de politiques pour supporter de nouvelles fonctionnalités (comme définir un nouveau type de données ou un attribut PIP) nécessite soit de modifier le code source du moteur de décisions d'autorisation, soit de changer le moteur. Une telle situation nécessite de stopper le système pour que le nouveau moteur soit chargé et puisse prendre en compte cette extension. Cela pose une difficulté car de telles opérations sont lourdes à mettre en œuvre et imposent pendant cette phase de mise à jour d'arrêter les évaluations des requêtes le temps de l'opération de maintenance. Ces implémentations s'adaptent statiquement pour effectuer le changement du système.

De plus, le cadre « extensibleXACML »[Cheaito et al. 2010a] propose une implémentation d'un moteur de décisions d'autorisation XACML extensible. Il comprend un composant de base ou PDP qui implémente le standard XACML et qui peut supporter des modules supplémentaires pour toutes extensions du langage de politiques XACML (tels que des nouveaux types de données et fonctions non standards). La solution proposée permet d'étendre l'architecture sans pour autant requérir de modifier l'ensemble du système, c.à.d. que cette solution propose des changements dynamiques sous une certaine forme. Mais, le système doit être redémarré chaque fois qu'un nouveau module doit être inclus, ce qui implique une adaptation statique là-encore.

HERASAF [HERASAF 2011] permet d'ajouter dynamiquement des extensions au langage XACML avec des modules (comme nouveau type de données) possédant des fichiers de configuration ce qui confère une adaptation dynamique du moteur de décisions pour effectuer le changement du système. Cependant, ce projet ne définit aucune méthode qui explique comment structurer les extensions du langage efficacement. En conséquence, il n'y a pas de garantie qu'une politique avec extensions puisse être interprétée, même si le moteur de décisions d'autorisation est extensible dynamiquement. Il manque une **méthodologie** qui explique comment créer les extensions du langage

ou modules qui seront utilisables par le moteur de décisions d'autorisation. Il manque également un guide d'utilisation des extensions du langage de politiques XACML pour le moteur de décisions de politiques. Pour résoudre ce problème, l'implémentation du moteur de décisions d'autorisations doit considérer l'exigence suivante :

- **Rendre les modules des extensions du langage XACML réutilisables :** Le moteur de décisions d'autorisation XACML doit être capable de ne réutiliser que les seuls codes des modules qui sont spécifiques aux politiques à évaluer durant le processus de prise de décision.

Ces implémentations modifient le moteur de décisions par inclusion de code supplémentaire pour les extensions du langage XACML, sans qu'un lien soit maintenu entre une politique et les extensions qui lui sont spécifiques. De ce fait, lorsqu'une politique est supprimée, le code des extensions qu'il a été nécessaire d'installer pour son exécution reste toujours dans le moteur de décisions. Dans ce cas, on peut avoir des codes inutiles dans le moteur de décision ce qui peut conduire à terme à des problèmes de performance. Si l'on souhaite s'affranchir de ce problème le moteur de décisions d'autorisation doit alors satisfaire à l'exigence suivante :

- le moteur de décisions d'autorisation doit être capable de **gérer le cycle de vie des modules** des extensions du langage installés dans moteur de décisions d'autorisation XACML**.** Ce moteur doit être capable de gérer le lien de dépendance entre les politiques et ces modules réutilisables qui lui sont spécifiques. La suppression d'une politique doit conduire au retrait des modules spécifiques ajoutés pour son exécution.

Au-delà de ces problèmes techniques il faut souligner le rôle majeur des administrateurs. Ils sont responsables de la mise en place des exigences des organisations, et donc de la spécification de politiques d'autorisation, et de leur bonne exécution. Ainsi ce sont les administrateurs de sécurité qui sont responsables de l'écriture des politiques XACML. Or suivant la nature des organisations, les administrateurs peuvent être ou non des spécialistes. Il y a donc lieu d'essayer de faciliter la tâche de spécification des politiques.

Différents travaux menés par la communauté de recherche traite de la spécification des politiques XACML. Premièrement, les politiques XACML étant des documents XML, tout éditeur XML comme par exemple XMLPAD [XMLPAD 2011] peut être utilisé. Les éditeurs XML peuvent vérifier la structure XML des politiques XACML en important le schéma XACML. Un éditeur dédié XACML [UMU 2011] a été créé pour faciliter la tâche de spécification. Il fournit aux utilisateurs une interface graphique cachant tous les détails XML. Il propose la manipulation de tous les types de données et les fonctions standards ; de plus de nouveaux types de données et fonctions peuvent être utilisés si nécessaire. Enfin, une nouvelle approche très prometteuse a été développée par l'Université d'Ottawa [Stepien et al. 2010]. Leur idée est de faire un éditeur de politiques pour des utilisateurs non-techniques. Ils s'attachent à créer une notation pour rendre les politiques XACML beaucoup plus lisibles par les êtres humains. Leur éditeur de politiques génère en sortie un fichier XACML. La facilité d'utilisation du langage XACML a été améliorée.

Bien que ces outils conduisent tous à la génération d'un fichier XML (respectant la structure XACML). Nous avons pu noter l'absence d'une **méthodologie** qui explique quelles extensions apporter au langage XACML et comment spécifier ces extensions dans les politiques. De plus nous avons également noté que rien ne permet d'assurer qu'une politique contenant des extensions puisse être interprétée par un moteur de décisions de politiques donné.

5.5 Conclusion

Nous avons choisi XACML comme technologie cible car il est basé sur les deux approches : ABAC pour la spécification des politiques et PBM pour l'architecture. Ces deux approches apportent le critère de flexibilité au langage de politiques et à l'architecture. Le langage de politiques XACML fournit des types de données et des fonctions standards (tels que entier, double, booléen etc.) pour manipuler des attributs ABAC. De plus, ce langage est extensible et donne la possibilité d'ajouter de nouveaux attributs, types de données, et fonctions.

Cependant, l'extensibilité du langage qui est une caractéristique nécessaire pour la spécification de politique de manière générique engendre un problème de déploiement. En effet, un moteur de décisions d'autorisation ou PDP respectant le standard XACML peut ne pas reconnaître un nouveau type de données ou une nouvelle fonction qui aurait été ajoutée au langage. La programmation d'une extension à ce PDP est obligatoire afin de le rendre capable d'interpréter ces nouvelles informations. Cela pose un problème de consistance : Un moteur de décisions d'autorisation donné est-il capable d'interpréter une extension donnée du langage ?

Par ailleurs, plusieurs implémentations pour moteur de décisions d'autorisation XACML existent. Vis-à-vis des extensions du langage de politique, certaines supportent leur prise en compte par le PDP statiquement alors que d'autres dynamiquement. Cependant aucune de ces implémentations ne fournit une **méthodologie** expliquant comment structurer ces extensions efficacement. **Pour cela, dans le prochain chapitre nous allons présenter une méthode qui explique : quelles extensions considérer dans le langage XACML (nouveaux attributs, types de données, et fonctions), comment créer des extensions réutilisables pour moteur de décisions d'autorisation et comment spécifier les extensions dans les politiques XACML. Enfin, cette méthode aide à la conception de notre moteur d'autorisation XACML extensible et adaptable.**

Chapitre6. Une méthodologie pour créer des extensions réutilisables

6.1 Introduction

Comme nous l'avons conclu dans le chapitre 5, la mise en œuvre de moteurs de décisions d'autorisation adaptables et extensibles doit permettre de répondre à la diversité des situations rencontrées lors de l'écriture de politiques. Le langage XACML présente des possibilités en matière d'extensibilité. L'ajout d'une nouvelle extension au langage XACML (attributs, fonctions, type de données etc.) n'a de sens que si cette extension facilite l'écriture de règles ou de politiques. Pour qu'une extension soit prise en compte, il faut que le moteur de décision d'autorisation ou PDP ait la capacité de la traiter c'est-à-dire de reconnaitre son occurrence et conduire à une décision [Cheaito et al. 2009] [Laborde et al. 2009].

Il nous a semblé important que l'écriture de politiques et leur exécution soit facilitée. Dans cet esprit, il est nécessaire d'avoir une **méthode** qui explique quelles extensions du langage XACML sont à retenir pour exprimer les exigences des organisations. Une telle méthode doit préciser comment créer les modules des extensions du langage afin d'être exécutées par un moteur de décisions d'autorisation et spécifiés dans les politiques. En effet, lorsqu'un administrateur de sécurité manifeste le besoin d'étendre un langage, il cherche bien sûr les modules qui correspondent à ses besoins. Une fois identifiés, ces modules sont chargés dans le moteur de décisions d'autorisation qui implémente le standard XACML afin d'être exécutés.

Ce chapitre est consacré à l'étude des problématiques d'extensibilité du langage XACML et des solutions que nous proposons. La problématique relative à l'édition et à l'exécution des politiques avec des extensions sera présentée dans le dernier chapitre de cette thèse.

6.2 Les types de données réutilisables

Le langage XACML permet à l'administrateur de sécurité de manipuler dans les politiques des expressions composées :

- Des types de données de base : chaîne de caractères, booléen, entier, double, date, URI, NomX500 etc.
- D'exécuter des fonctions de base de manipulation de ces types de données : comparaison de chaînes de caractères (égalité, supériorité, infériorité…), comparaison de dates, comparaison d'entiers etc.

Ces types de données et fonctions de base permettent d'exprimer certaines exigences de sécurité dans les politiques à condition qu'elles restent simples. Cependant, certaines situations imposent que des types de données et des fonctions plus complexes soient présentes dans des règles de politiques. Par exemple, la notion de rôle est mise en avant dans nombre de politiques de contrôle pour guider l'accès à des ressources. Dans le modèle formel de contrôle d'accès RBAC, le Rôle est défini comme un ensemble d'ordre partiel (POSET). La relation d'ordre (≥) entre les valeurs de rôles possède les propriétés mathématiques de réflexivité, antisymétrie, et transitivité. Une ambiguïté sur l'utilisation d'une fonction identifiée par la relation d'ordre (≥) pourrait naitre. Lorsqu'une telle fonction est associée au type de données « chaîne de caractères », cette fonction s'arrête à la comparaison de deux chaînes. Lorsqu'elle est associée à la manipulation de rôles, cette fonction devrait permettre de positionner hiérarchiquement un rôle par rapport à un autre. Tels que les types de données standards ont été définis, et telle que la sémantique de la fonction (≥) a été également définie,

il n'est pas possible de la réutiliser pour l'associer à la hiérarchie de rôles figurant dans le modèle RBAC. Il faut donc étendre le langage XACML pour proposer aux administrateurs de sécurité un nouveau type de données et de nouvelles fonctions de manipulation de celui-ci.

Plus généralement, le standard XACML limite la facilité d'expression des diverses exigences de sécurité. Alors que des modèles formels de contrôle d'accès voient régulièrement le jour pour apporter des solutions à des problématiques particulières, il est difficile d'en bénéficier lors de l'utilisation du langage standard XACML. La seule solution est d'une part d'étendre ce langage ; d'autre part, il faut également proposer des solutions pour structurer l'écriture des modules mettant en œuvre les extensions qui peuvent être apportées au langage. Il faut également proposer une architecture pour rendre ces modules réutilisables. Enfin, il faut que ces modules soient installés au sein du moteur de décisions d'autorisation pour que celui-ci puisse interpréter et exécuter la politique contenant les extensions. Cette architecture sera présentée dans le dernier chapitre.

De notre point de vue, les exigences de sécurité qui conduisent à la production d'extensions du langage XACML peuvent être classées en deux groupes d'informations :

- Certaines informations retenues dans la composition d'une règle de politiques peuvent ne pas être spécifiques à un contexte d'utilisation. La syntaxe et la sémantique sont en règle générale connues et prédéfinies. Il en est ainsi par exemple des informations GPS (Global Positioning System), des adresses MAC, des adresses IP, high precision real etc.
- D'autres informations de sécurité sont au contraire spécifiques à un contexte d'utilisation. De telles informations sont qualifiées de spécifiques car elles singularisent une forme d'organisation, un projet de collaboration, un système et donnent un sens à une règle de sécurité à appliquer. On peut abréger ces informations par le terme de SSI pour informations de sécurité spécifique. Elles nécessitent le concours de l'administrateur de sécurité pour qu'elles soient modélisées au sein des politiques (par exemple, rôles, localisations physique ou logique, contexte, temps d'accès etc.). Les administrateurs de sécurité ne sont pas supposés être des programmeurs chevronnés même s'ils ont une bonne connaissance des concepts de sécurité. L'exécution d'une politique repose de plus en plus sur la manipulation de modèles formels de contrôle d'accès, car ils intègrent des structures mathématiques spécifiques pour garantir les propriétés mathématiques. En conséquence, les administrateurs de sécurité doivent respecter les structures mathématiques intrinsèques quand ils définissent leur information de sécurité et leurs politiques d'autorisation.

Enfin, Le langage XACML présente ici un intérêt particulier car il donne la possibilité d'exprimer les exigences de sécurité en utilisant les extensions du langage au travers de la formulation de nouveaux types de données ou fonctions si besoin.

6.2.1 Exemples

Nous présentons des exemples afin de préciser les exigences de sécurité qui conduisent à la production d'extensions du langage XACML auxquelles un administrateur est confronté et les implications qu'elles ont sur l'écriture de politiques en langage XACML. Nous considérons les scenarios suivants :

1. Une adresse IP (avec IP pour Internet Protocol) dans un pare-feu est un numéro d'identification qui est attribué à chaque branchement d'appareil à un réseau informatique utilisant l'Internet Protocol. Il existe des adresses IP de version 4 et de version 6. La version

4 est actuellement la plus utilisée : elle est généralement représentée en notation décimale avec quatre nombres compris entre 0 et 255, séparés par des points, ce qui donne par exemple : 212.85.150.134. Une politique d'autorisation pour adresse IP version 4 pourrait être écrite comme suit : si l'adresse IP de l'ordinateur de l'utilisateur est égale à «192.168.10.20» alors permettre l'accès aux ressources du système d'adresse IP égale à « 194.154.192.3». Ce premier exemple classique permet de souligner la manipulation de paramètres que l'on peut considérer comme triviaux dans une politique.

2. L'Université Paul Sabatier (UPS) propose à ses étudiants une liste de propositions d'emplois de vacances contenues dans le fichier « jobs.html » situé dans un serveur. La politique d'autorisation écrite par administrateur de sécurité de l'organisation « UPS » pourrait être : «Seuls les étudiants de UPS peuvent lire le fichier jobs.html ». L'organisation UPS délivre trois diplômes : « phd », « master », et « licence ». Si l'on se réfère à la hiérarchie de rôles représentée dans (Figure 36), les utilisateurs « étudiant » qui sont inscrits dans la préparation de ces diplômes (« phd », « master » ou « licence ») peuvent donc accéder au fichier « jobs.html », car ces trois rôles sont sous-rôles du rôle « étudiant » [Cheaito et al. 2010a]. Ce deuxième exemple permet de souligner qu'une politique peut comprendre des arguments spécifiques à une organisation ; ici on parle d'étudiants mais ailleurs ce pourrait être des employés, des contractuels, des saisonniers, des ressources affectées à un projet…

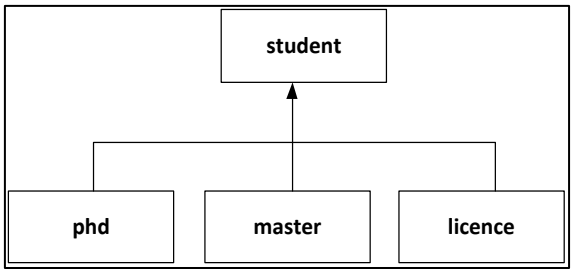

Figure 36 Hiérarchie des rôles

3. Ce scénario complète le scénario précédent, en considérant une hiérarchie de rôles. Au sein de l'université Paul Sabatier (UPS) figure un laboratoire de sciences informatique appelé « IRIT » et un laboratoire de mathématiques appelé «IMT». L'« IRIT » occupe deux immeubles. Des domaines de localisations logiques sont naturellement identifiés : « UPS », « IRITLab » (IRIT) , « MATHLab » (IMT) , « Batiment1 » et « Batiment2 », et une hiérarchie de localisations logique peut en découler (Figure 37). L'IRIT propose aux doctorants en « phd » de l'UPS d'accéder à une liste de propositions d'emplois en informatique quand ils sont situés dans l'un des bâtiments de l'IRIT. Ils peuvent être soit à l'intérieur du « Batiment1 » soit dans le « Batiment2 ». Cette politique d'autorisation peut être spécifiée comme : « les utilisateurs avec rôle « phd » et de localisation logique « IRITLab » ont l'autorisation de « (lecture,CS-jobs.html) ». Selon la hiérarchie, les étudiants de doctorat « phd » à l'intérieur de « Batiment1 » et « Batiment2 » peuvent également lire le fichier « CS-jobs.html ». Ce scénario permet d'illustrer que des contraintes de localisation géographiques spécifiques à un établissement peuvent apparaître au sein d'une politique.

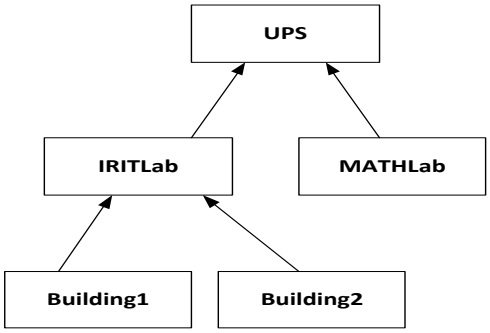

Figure 37 Hiérarchie de localisations logiques

4. Considérons un site web où les usagers peuvent constituer des groupes pour partager des documents/informations entre les membres du même groupe (Figure 38). Chaque groupe comprend un seul gestionnaire, qui est responsable de la gestion du groupe appelé « manager », et des membres, qui peuvent accéder aux seuls documents partagés par le groupe. Une même personne peut faire partie de plusieurs groupes différents et y tenir le rôle « manager » ou le rôle « member ». Afin d'assigner un rôle à un groupe spécifique, il faut considérer les accréditations des utilisateurs qui sont de la forme « <group-role-value, group-id> », où le groupe-rôle-value est la valeur de rôle soit « manager » soit « member », et le groupe-id est le nom du groupe. Dans la hiérarchie, le rôle « member » est un sous-rôle de rôle « manager ». Autrement dit, les rôles « manager » se voient accorder tous les privilèges des rôles « member ». La politique pour le rôle « manager » avec ABAC est : « ≥(get-role-attribute(credential(U)), manager) ∧ =(get-group-attribute(credential(U)), group(R)) ∧ =(id(R),management-page) ∧ =(id(A),access) ». Où, le credential(U) représente les accréditations de l'utilisateur, group(R) représente le nom du groupe de la ressource, id (R) représente l'identificateur de la ressource, et id (A) représente l'identificateur de l'action. Cette politique signifie qu'un utilisateur peut faire l'action « accès » sur la ressource « management-page » ; si l'une de ses accréditations comprend au moins le rôle « manager » dans la hiérarchie de rôles (≥(get-role-attribute (credential(U)),manager)) pour le même groupe (=(get-groupattribute (credential(U)), group(R))). En plus la politique associée au rôle « member » avec ABAC est spécifiée comme: « ≥(get-role-attribute(credential(U)), member) ∧ =(get-group-attribute(credential(U)), group(R)) ∧ =(id(R),shared-documents) ∧ isEqual(id(A),access) ». Cette politique stipule qu'un utilisateur peut accéder aux documents partagés d'un groupe si l'une de ses accréditations comprend au moins le rôle « member » pour le même groupe.

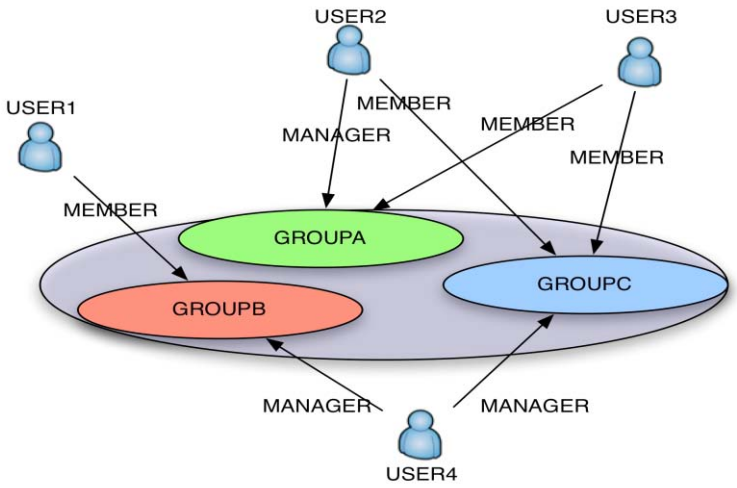

Figure 38 Le groupe scenario

6.2.1.1 Analyse

Le premier scenario est exprimé dans le langage de politiques XACML en choisissant le module qui implémente le type de données appelé « adresse IP ». Ce type de données est classé comme une exigence de sécurité qui n'est pas spécifique à un contexte d'utilisation. Car l'adresse IP a une syntaxe et une sémantique prédéfinies. Les valeurs et les relations entre les adresses IP V4 sont définies par défaut. On sait qu'une valeur de type de données « adresseIP » est structurée de quatre nombres compris entre 0 et 255, séparés par des points (si elle est V4). Les fonctions associées à ce type de données sont des fonctions simples. Par exemple, la fonction « = » retourne vrai lorsque deux valeurs de type « adresse IP » sont identiques. Définir l'adresse IP comme un type de données non-standard donne la possibilité d'avoir des fonctions de manipulation autres que des fonctions de comparaison dans le standard qui répondent idéalement aux besoins propres de l'administrateur de sécurité.

En plus, la politique du scenario « adresse IP » peut être spécifiée dans le langage de politiques XACML comme suit $[\text{adresseIP(Sujet)}]_{\text{adresseIP}}$ $[\text{isEqual}]_{\text{adresseIP}}$ $[\text{192.168.10.20}]_{\text{adresseIP}}$ ∧ $[\text{adresseIP(Ressource)}]_{\text{adresseIP}}[\text{isEqual}]_{\text{adresseIP}}$ $[\text{194.154.192.3}]_{\text{adresseIP}}$ ∧ $[\text{nom(Action)}]_{\text{String}}$ $[\text{isEqual}]_{\text{String}}$ $[\text{permit}]_{\text{String}}$ où, les valeurs d'attributs (192.168.10.20, et 194.154.192.3), la fonction (isEqual) et le nom d'attribut (adresseIP) sont en corrélation avec le type de données « adresseIP ». Enfin, pour exécuter une telle politique, l'administrateur doit choisir le module qui implémente le type de données « adresse IP » (contenant la sémantique et la syntaxe de valeurs et les fonctions associées à ce type). Les adresses MAC, la position réelle sur terre… sont d'autres exemples de types de données simples pour exprimer des exigences non spécifiques à un contexte d'utilisation. Nous leur donnons le qualificatif de « type de données prédéfini » pour les types de données et fonctions simples et standards.

Dans le deuxième scenario de « Rôle », on a l'exigence de rôle qui est définie dans des modèles formels de contrôle d'accès comme RBAC. Dans le troisième scenario on a les exigences de localisation logique et rôles qui figurent dans des modèles formels de contrôle d'accès comme LRBAC. Ces exigences sont spécifiques à un contexte d'utilisation dépendant d'une organisation. Les valeurs et hiérarchies de rôles ou de localisations logiques sont définies par l'administrateur de sécurité pour son organisation. Les valeurs et les hiérarchies entre valeurs des rôles ou des localisations logiques ne sont pas définies par défaut. Elles doivent être configurées par l'administrateur de sécurité qui doit s'assurer de leur conformité avec les structures mathématiques définies par des modèles formels de contrôle d'accès.

La politique dans le deuxième scenario peut être spécifiée par l'administrateur dans XACML avec l'aide du module « Rôle », qui implémente le type de données pour rôles comme suit : $[\text{rôle(Sujet)}]_{\text{Rôle}}$ $[\text{isAtLeast}]_{\text{Rôle}}$ $[\text{étudiant}]_{\text{Rôle}}$ ∧ $[\text{nom(Ressource)}]_{\text{String}}$ $[\text{isEqual}]_{\text{String}}$ $[\text{jobs.html}]_{\text{String}}$ ∧ $[\text{nom(Action)}]_{\text{String}}$ $[\text{isEqual}]_{\text{String}}$ $[\text{lire}]_{\text{String}}$. Où, la fonction « isAtLeast » représente la relation d'ordre. Mais, la politique XACML du troisième scenario peut être spécifiée par l'administrateur de sécurité avec l'aide de deux modules « Rôle » pour les rôles et « Localisation » pour les localisations logiques. La politique peut être spécifiée comme suit: $[\text{rôle(Suject)}]_{\text{Rôle}}[\text{isAtLeast}]_{\text{Rôle}}$ $[\text{phd}]_{\text{Rôle}}$∧ $[\text{localisation(Subjet)}]_{\text{Localisation}}$ $[\text{isInside}]_{\text{Localisation}}$ $[\text{IRITLab}]_{\text{Localisation}}$∧ $[\text{nom(Ressource)}]_{\text{String}}$ $[\text{isEqual}]_{\text{String}}$ $[\text{CS-jobs.html}]_{\text{String}}$∧ $[\text{nom(Action)}]_{\text{String}}[\text{isEqual]String[lecture}]_{\text{String}}$ Où, la fonction « isInside » représente la relation d'ordre de localisation logique pour le type de données *Localisation*, et « isAtLeast » représente la relation d'ordre de rôles pour le type de données *Rôle*. Pour répondre à ces exigences de sécurité, l'administrateur choisit les deux modules « Rôle » et « Localisation » pour spécifier les exigences de rôles et localisations logiques dans l'écriture de la politique XACML. Aussi, l'administrateur de sécurité doit configurer les deux modules pour définir les valeurs et les hiérarchies entre les valeurs. Enfin, tous les types de données comme rôle et localisation logique sont classés comme des exigences spécifiques à un contexte d'utilisation. Nous donnons le nom « type de données configurable scalaire » pour ces catégories d'exigences de sécurité.

Le troisième scenario représente une exigence de sécurité appelée « credential » qui regroupe deux exigences de sécurité : l'une est le rôle, l'autre est le groupe. Le type de données « credential » caractérise un besoin de l'administrateur qui souhaite regrouper plusieurs exigences de sécurité. Les fonctions nécessaires à ce type de données sont des fonctions de récupérations de valeurs. Dans le scenario proposé, le type de données « credential » regroupe le type de données « Rôle » pour rôle et le type de données standard « String » pour groupe. Par exemple, l'administrateur choisit le module « credential » pour spécifier dans le langage XACML une politique avec cette exigence comme suit : [get-role-attribute [credential(Sujet)] $_{credential}]_{Rôle}$ $[isAtLeast]_{Rôle}$ $[manager]_{Rôle}\wedge$ [get-group-attribute [credential(sujet)] $_{credential}]_{String}$ $[isEqual]_{String}$ $[group(ressource)]_{String}\wedge$ $[id(ressource)]_{String}$ $[isEqual]_{String}$ $[management\text{-}page]_{String}\wedge$ $[id(action)]_{String}$ $[isEqual]_{String}$ [access] $_{String}$. La fonction « get-role-attribute » qui récupère la valeur de rôle, et la fonction « get-group-attribute » qui récupère le nom du groupe sont des fonctions du type de données composite « credential ». La fonction « isAtLeast » représente la relation d'héritage pour le type de données « Rôle », et la fonction « isEqual » pour le type de données standard « String ». Mais, les fonctions de « credential » sont des fonctions qui accèdent aux valeurs de champs de types de données Rôle et String. Nous donnons le nom « type de données configurable composite » pour ces catégories d'exigences de sécurité.

6.2.2 Types de données prédéfinis

Il s'agit des types de données venant en complément des types du standard XACML (entiers, chaîne de caractères...), tels des types de données simples (par exemple, l'adresse MAC, la position réelle des utilisateurs dans des dispositifs GPS, adresse IP etc.). Ces types de données dont la syntaxe et la sémantique sont prédéfinies permettent de prendre en compte des exigences de sécurité qui ne sont pas spécifiques à un contexte d'utilisation particulier. Nous présentons ici un autre exemple (différent de la manipulation d'une adresse IP) qui utilise la position réelle des utilisateurs sur terre (user's real position) par un dispositif GPS [Cheaito et al. 2009].

Le GPS est un système qui fournit des services de positionnement fiables, de navigation, et de synchronisation pour des utilisateurs ou des objets. Dans un dispositif GPS, la géométrie d'un objet peut être de type point, ligne ou polygone [Bertino et al. 2005]. Les dispositifs GPS sont utilisés pour définir les positions que peut occuper un élément Cela signifie que les valeurs données à partir de dispositifs GPS sont prédéfinies et représentent les positions réelles. Ainsi, la position réelle est un type géométrique, soit de type point qui représente en trois dimensions (latitude, longitude et altitude) une position donnée sur terre. Ces données GPS sont exprimées en utilisant des formats différents, comme le GPS eXchange format standard (GPX) [GPX 2011], Keyhole Markup Langage (KML) [KML 2011]...

Les trois éléments (latitude, longitude et altitude) de la position réelle sont obtenus à partir d'une représentation fidèle à un type de données existant (par exemple KML utilise le type de données *double* qui est standard dans XACML) et ses fonctions pour manipuler des expressions (par exemple, les fonctions du type de données *double* dans XACML) avec une sémantique et une syntaxe prédéfinies. Cela signifie que le type de données position réelle est composé de types de données standards et que l'ensemble des valeurs possibles sont connues par défaut. Ainsi, ce type peut permettre de calculer des valeurs au travers des fonctions comme isBelow, isAbove, isBetween ... qui sont des fonctions standards existantes.

Ainsi les types de données prédéfinis qui n'ont pas besoin d'être configurés par l'administrateur de sécurité peuvent être un type de données standard ou simple ou des champs de structure, séquences, composition de types de données standards ou simples. On peut associer à ces types de données des fonctions de comparaisons de valeurs. Pour être utilisés, les modules de ces types de données doivent être installés dans le moteur de décisions.

6.2.3 Type de données configurable scalaire

Ce type de données caractérise les types de données abstraits pour lesquels les valeurs et relations entre valeurs ne sont pas prédéfinies. Ces types de données permettent d'exprimer des exigences de sécurité spécifiques à un contexte d'utilisation (SSI). Par exemple, le modèle RBAC ne fournit pas les valeurs de rôles. Si l'administrateur doit exprimer des contraintes basées sur les rôles, il doit spécifier les valeurs de rôles et fournir la hiérarchie de rôles pour que ces valeurs puissent être manipulées. Dans ce contexte, nous proposons une méthodologie (Figure 39) pour étendre le langage XACML avec de nouveaux types de données configurables scalaires.

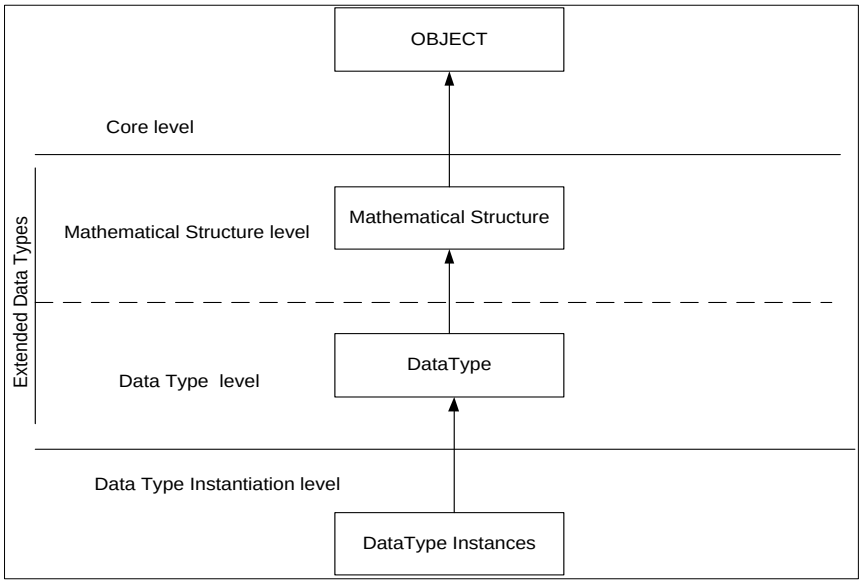

Figure 39 Méthodologie pour étendre le langage avec types de données configurables scalaires

Cette méthode est décomposée en trois niveaux [Cheaito et al. 2010a] :

- *Core level* : Ce niveau représente tout. Nous utilisons ce niveau pour intégrer toutes les extensions du langage XACML c.à.d. tous les types de données et leurs fonctions. Nous avons été inspirés par le langage JAVA qui considère que toutes les classes héritent de la classe *Object*.
- *Extended data types level* : Ce niveau est divisé en deux sous-niveaux :
  - *Mathematical Structure sub-level* : représente les structures mathématiques, y compris leurs propriétés mathématiques et les algorithmes associés. Ce niveau constitue une base mathématique générique pour traiter les informations de sécurité au sein des modèles formels de contrôle d'accès. Par exemple, la structure mathématique et les propriétés mathématiques de la relation d'ordre de rôles du modèle RBAC sont définies à ce niveau.
  - *Data Type sub-level* : Ce niveau représente les extensions du langage XACML avec de nouveaux types de données utilisés dans des politiques de contrôle d'accès. Ils comprennent également les fonctions nécessaires pour écrire les politiques de contrôle d'accès. C'est à ce niveau que l'administrateur de sécurité définit les exigences de sécurité telles que le rôle, la localisation logique, le temps... qui sont spécifiques à un contexte d'utilisation.
- *Data Type Instantiation level* : Ce niveau représente les instances définies par les administrateurs de sécurité pour les types de données définis au niveau supérieur. L'administrateur de sécurité spécifie, pour chaque instance, les valeurs et les hiérarchies adaptées à son propre contexte. Par exemple dans le niveau *Data Type,* l'administrateur de sécurité choisit le module associé au type de données Rôle pour prendre en compte la

notion de rôles du modèle RBAC dans des politiques. Dans ce niveau, l'administrateur crée une instance de type de données *Rôle* appelée *RoleOrg1*pour l'organisation *Org1*. Pour cette instance, l'administrateur peut spécifier les valeurs et les hiérarchies correspondant à l'organisation *Org1*.

Ainsi, la méthodologie appliquée pour le type de données configurable scalaire « Rôle » dans le scenario donné ci-dessus sera définie de la manière suivante (Figure 40):

- *Core level* : Ce niveau représente tous les types de données et les fonctions configurables scalaires.
- *Extended data types level : Ce niveau est divisé en deux sous-niveaux :*
  - *Mathematical Structure sub-level* : représente les structures mathématiques des types de données configurables au cas où ils existent. Pour le type de données *Rôle* la structure mathématique prise du modèle RBAC est un ensemble d'ordre partiel (*POSET*) et les propriétés mathématiques (réflexivité, antisymétrie, et transitivité) sont définies à ce niveau par l'administrateur de sécurité. Pour sa mise en œuvre, un outil doit être fourni qui permet à l'administrateur de définir les structures et les propriétés mathématiques.
  - *Data Type sub-level* : Ce niveau représente la définition de types de données configurables scalaires. Alors, le type de données *Rôle* est défini dans ce niveau comme un type abstrait spécifique de la structure mathématique *POSET*. Ce type de donnée inclut des fonctions abstraites comme *isAtLeast* qui représente la relation d'ordre définie dans RBAC.

- *Data Type Instantiation level :* Ce niveau contient les instances de tous les types de données abstraits ou informations de sécurité SSI définis par les administrateurs de sécurité. Par exemple, dans le scénario de RBAC, l'administrateur de sécurité définit sa propre instance de type de données Rôle pour l'organisation Université Paul Sabatier appelé UPSROLE. Pour cette instance, l'administrateur de sécurité spécifie les valeurs de rôle (student, phd, master et licence) et leur hiérarchie correspondante. Pour sa mise en œuvre, un outil doit être fourni à l'administrateur afin qu'il spécifie les instances, les valeurs, et les hiérarchies.

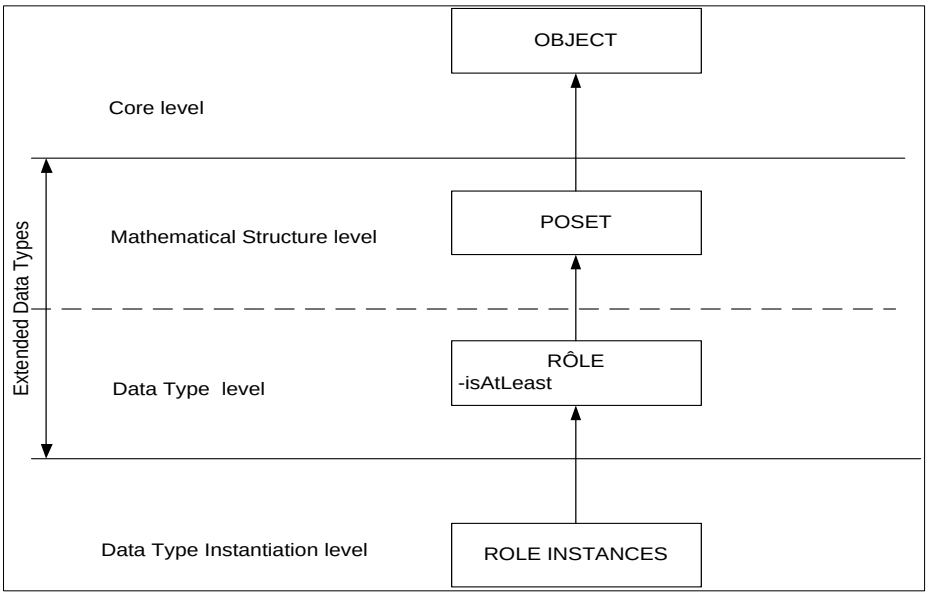

Figure 40 Méthodologie d'un type de données « Rôle » configurable scalaire

6.2.3.1 Exemple d'une mise en œuvre d'un type de données configurable scalaire

Rappelons (**Exigence 3** dans la section 5.1 du chapitre 5**)** que le système d'autorisation doit fournir des fonctionnalités de configuration pour les administrateurs de sécurité. Dans la section précédente, nous avons expliqué comment appliquer une méthodologie pour étendre le langage de politique XACML avec des types de données configurables scalaires. Nous offrons un outil dans notre système d'autorisation mettant en œuvre cette méthodologie de telle sorte que les administrateurs puissent avoir la possibilité d'exprimer des politiques sans s'attacher à analyser et comprendre tous les fondamentaux mathématiques associés à la création de nouveaux types de données.

Le concept d'ajout de nouveaux types de données est traité en considérant les travaux menés sur les ontologies et en le considérant comme un moyen [Cheaito et al. 2010a]. Les ontologies [Horridge 2009] sont utilisées pour décrire des concepts dans le domaine, les individus (c.à.d. des instances de concepts) dans le domaine, et les relations qui s'établissent entre ces concepts. Cette décomposition s'applique à notre méthodologie où les deux niveaux « Mathematical Structure level » et « Data Type level » définissent des concepts généraux et abstraits et le niveau « Data Type Instantiation level » définit des instances de ces concepts. Nous utilisons dans notre système d'autorisation l'outil Protégé [Protégé 2011][Horridge 2009] comme un éditeur de l'ontologie, car il permet :

- La mise en œuvre de notre méthodologie pour les types de données en spécifiant les concepts, les individus et les relations entre les concepts
- La spécification et la vérification des contraintes mathématiques pour des relations entre les concepts en utilisant le plug-in « PAL constraint » [PAL 2011]
- Le raisonnement avec des individus, types de données définis par l'utilisateur, et débogage des ontologies à l'aide du plug-in étendu pour les moteurs de raisonnements, comme Pellet [Sirin et al. 2007]

Par exemple l'ontologie dans la Figure 41, représente la mise en œuvre de la méthodologie des types de données configurables scalaires « Rôle » et « Localisation logique ». Cette ontologie suit la méthodologie en définissant d'abord le concept « THING » qui représente l'ensemble contenant tous les individus et les concepts définis dans le niveau « Core level ». Pour cette raison, tous les concepts sont dérivés du concept « THING ». Aussi, le concept « DATATYPES » qui représente l'ensemble de tous les types de données est dérivé du concept « THING ». Avec ces deux concepts on est au premier niveau de la méthodologie.

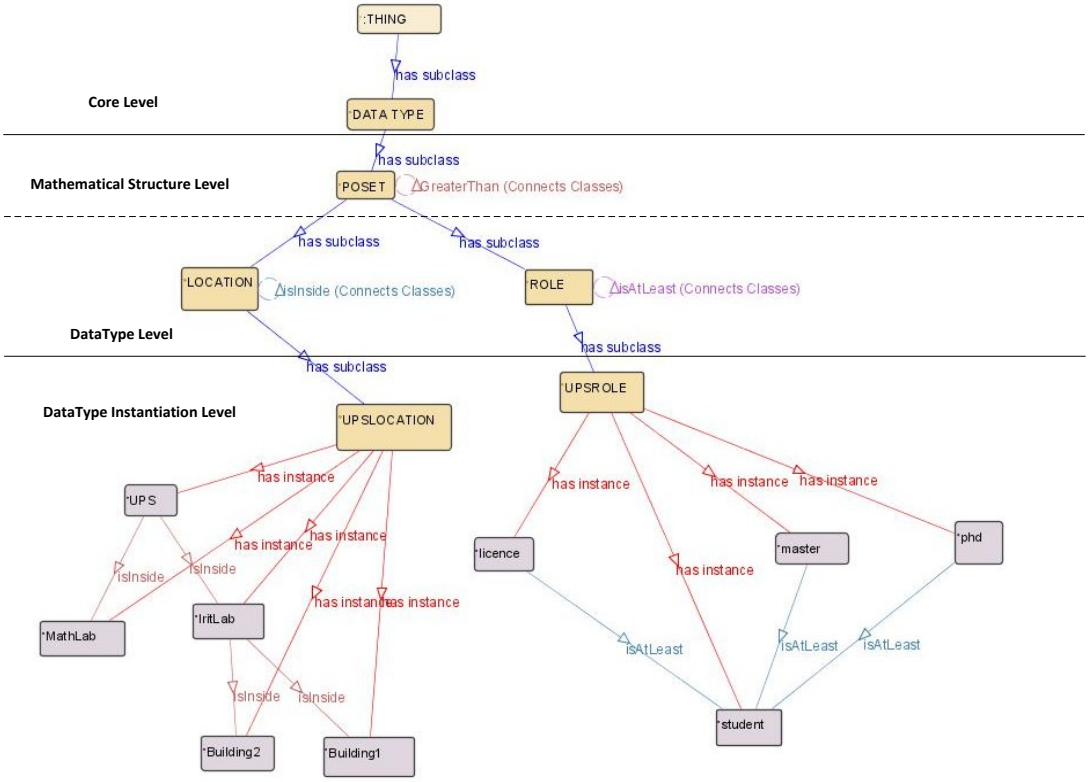

Figure 41 Ontologie pour type de donnée configurable scalaire

Ensuite, il faut définir la structure mathématique du type de données « Rôle » et « Localisation ». Le concept « POSET » représente cette structure mathématique. Il est aussi dérivé du concept de « DATATYPE ». La relation « Greater Than » pour le concept « POSET » spécifie la relation d'ordre définie dans les modèles formels RBAC et LRBAC. La sémantique de cette relation (réflexive, antisymétrique et transitive) est définie en utilisant le plug-in « PAL » (Figure 42). Les concepts «ROLE» et « LOCATION » représentent respectivement les types de données abstraits « Rôle » et « Localisation » définis dans la méthodologie au niveau « Data Type level » ; ils sont dérivés du concept « POSET ». La relation « isAtLeast » et « isInside » sont sous des noms différents dérivés de la relation « Greater Than ».

```
Reflexive:  (forall ?x
                (≥ ?x ?x))

Antisymmetric:  (forall ?x(forall ?y
                (=> (and ( (≥ ?x ?y)
                           (≥ ?y ?x)) )
                    (= ?x ?y) ) ))

Transitive:  (forall ?x(forall ?y(forall ?z
                (=> (and ((≥ ?x ?y)
                          (≥ ?y ?z)) )
                    (≥ ?x ?z) ) )))
```

Figure 42 POSET définition

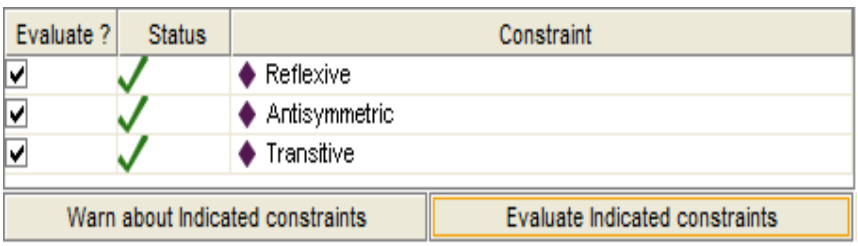

Figure 43 Contraintes de relation d'ordre avec PAL

L'administrateur de sécurité doit également définir le concept « UPSRole » qui est une instance du type de donnée configurable scalaire « Rôle ». Ce concept contient l'ensemble des valeurs (instances du concept) et la hiérarchie entre valeurs (relations entre instances) au regard du niveau « Data Type Instantiation level » de la méthodologie. L'administrateur utilise la relation « isAtLeast » pour le concept « ROLE », afin de définir la relation entre valeurs ou individus. En outre, l'administrateur de sécurité peut évaluer cette spécification en utilisant les propriétés mathématiques des contraintes prédéfinies par le plug-in « PAL » sur la relation d'ordre *POSET*, afin de vérifier les violations entre les individus (Figure 43). Ainsi, l'administrateur de sécurité peut utiliser le moteur de raisonnement pour vérifier la cohérence et la syntaxe des concepts, des individus et des relations entre concepts.

Par ailleurs, les types de données définis par les administrateurs de sécurité peuvent être exportés en utilisant « Protégé » aux différents formats comme le langage Web Ontology (OWL) ou XML [Horridge 2009][OWL 2011]. Ainsi, les administrateurs de sécurité peuvent écrire leurs politiques de contrôle d'accès en utilisant les types de données du fichier de configuration d'ontologie généré. Étant définis en dehors de la politique, ces types de données ne sont pas limités à une seule politique et peuvent être utilisés dans différentes politiques. Enfin, quand un administrateur de sécurité exprime le besoin d'utiliser un nouveau type de données ou une nouvelle exigence de sécurité, il suffit d'implémenter la méthodologie du nouveau type de donnée avec l'ontologie.

6.2.4 Type de données configurable composite

Les types de données configurables composites sont des types de données qui peuvent être construits comme des champs de structure, de séquence, de composition d'autres types de données prédéfinis ou scalaires ou même composites. Pour l'administrateur de sécurité, ces types de données sont des regroupements de deux groupes d'exigences de sécurité. Ces types peuvent être composés des champs de types de données standards ou simple seulement. La différence entre ce type et le type de données prédéfini vient des fonctions qui ne sont que des opérations de récupération des valeurs de champs mais pas des fonctions de comparaison (comme dans le cas du type de données prédéfini position réelle).

Prenons par exemple le scénario « credential » il est composé d'un « tuple »=<Rôle, String> où le type de données « Rôle » est configurable scalaire et où le type de données « String » est prédéfini. Les fonctions de « credential » sont utilisées pour récupérer les valeurs de champs des types de données du « tuple ». Ainsi, la fonction « get-rôle-attribute » est mise en œuvre pour récupérer les valeurs de rôle dans les accréditations de l'utilisateur. Les valeurs et les relations d'ordre entre valeurs des accréditations des utilisateurs ou « credential » ne sont pas connues par défaut, et doivent donc être configurées par l'administrateur de sécurité.

Pour s'adapter aux politiques contenant un type de données composite, l'administrateur de sécurité doit choisir le module qui implémente ce type de données en plus des modules propres à chaque type de données défini dans le champ de ce type de données composite. Ces modules doivent être configurés et chargés dans le moteur de décisions d'autorisation avec la politique pour qu'ils puissent être interprétés et exécutés. La configuration de ces modules doit être fournie avec des outils comme dans le cas du type de données configurable scalaire.

6.3 Les PIPs réutilisables

Rappelons, que le PIP (Policy Information Point) est la partie dans l'architecture de XACML qui est responsable de la récupération des valeurs d'attributs manquants (à partir de sources externes : base de données MySQL, LDAP, fichiers ...). De nombreuses solutions de stockage de données existent (chaque organisation est responsable du choix effectué pour stocker les comptes des

utilisateurs). L'administrateur de sécurité est alors tenu d'utiliser la solution de stockage retenue au sein de son organisation. Du point de vue de l'administrateur, ces solutions sont des exigences venant de l'environnement technologique de récupération des informations. De plus, ces différentes technologies ont des mécanismes d'accès et d'extraction qui leur sont propres (par exemple URL, Login, Password, requête SQL… pour une base de données MySQL).

Pour s'adapter à ces environnements technologiques différents, le moteur de décisions d'autorisation devrait proposer un module PIP par technologie de stockage. Chaque module, permet à l'administrateur de renseigner les différents attributs de connexion à la boite de stockage de données et de le spécifier dans des politiques XACML. Par exemple, le module « MySQL PIP » est responsable de la récupération des valeurs d'attributs qui sont stockées dans une base de données « MySQL ». Mais, le module « LDAP PIP » est responsable de la récupération des valeurs d'attributs dans l'annuaire « LDAP ». L'administrateur de sécurité de l'organisation va donc (Figure 44) :

- Choisir le module PIP correspondant à la technologie de stockage de son organisation
- Configurer ce dernier avec les attributs de connexion à la technologie de stockage
- Ecrire ses politiques avec des attributs de connexion configurée pour étendre le langage de politiques avec des extensions qui répondent aux exigences de son organisation.
- Charger la politique et les modules PIP spécifiques dans le moteur de décisions d'autorisation pour interprétation et exécution.

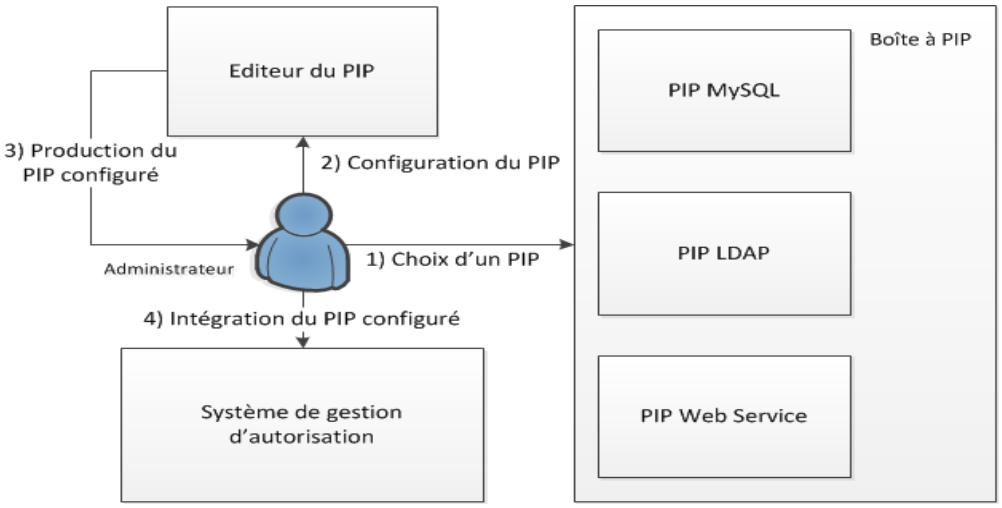

Figure 44 Processus de configuration d'un PIP

6.3.1 Exemple

Considérons que l'organisation possède deux bases de données MySQL distinctes (Figure 45). Les deux stockent les noms et prénoms des utilisateurs en fonction de leurs identifiants, mais la structure de données est différente dans chacune d'elles. La base de données « A » a une structure de données « STRUCTURE 1 », et la base de données « B » a une structure de données « STRUCTURE 2 ». Admettons que le module PIP MySQL comprenne les deux structures « STRUCTURE 1 » et « STRUCTURE 2 » pour récupérer les valeurs de l'attribut PIP MySQL « userID_pip_mysql ».

Dans XACML, l'administrateur a besoin d'étendre le langage de politiques XACML avec le nouvel attribut PIP MySQL « userID_pip_mysql ». Chaque attribut PIP MySQL est spécifié dans la politique avec un nom d'attribut de connexion qui est l'identifiant ou userID et le type de données de l'attribut. Par exemple, si l'administrateur souhaite ajouter un attribut pour un PIP MySQL, il doit spécifier dans la politique la ligne suivante: <SubjectAttributeDesignator AttributeId : « userID »

DataType : « String »/>. A partir de cet instant, le module PIP MySQL permet d'interpréter et d'exécuter des politiques contenant l'attribut PIP MySQL au sein du système d'autorisation.

De plus, le module PIP MySQL ne se limite pas à la compréhension des données de bases de données « A » et « B ». Il faut que le module PIP MySQL soit capable de transformer des données de format « STRUCTURE 1 » et des données de format « STRUCTURE 2 » en données de format « STRUCTURE 3 » qui est la structure des valeurs requises par le type de données de l'attribut PIP MySQL « userID_pip_mysql ». Le module PIP MySQL doit offrir une solution pour transformer les structures des données extraites depuis les bases afin qu'elles soient compatibles avec les structures de données requises par le système d'autorisation.

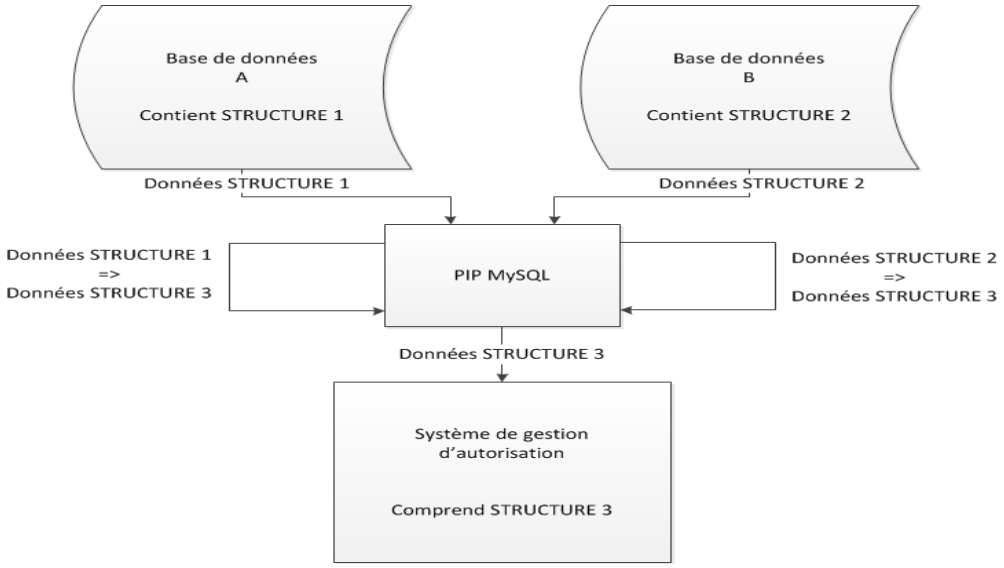

Figure 45 Scénario PIP MySQL

6.3.2 Méthodologie

Chaque module, par exemple *PIP MySQL*, devra être configurable de telle sorte que les besoins de l'administrateur de sécurité puissent être satisfaits. Le module PIP doit être capable d'extraire puis de modifier les informations des conteneurs de données de l'organisation pour qu'elles soient exploitables par le système d'autorisation. Sa configuration doit être facile, interfacée, et doit à chacune de ses étapes procéder à des validations. Les étapes pour avoir un module PIP réutilisable sont :

1. Réaliser l'extraction d'informations de connexion propres à la technologie de stockage. Pour se connecter à la base de données MySQL le fichier de configuration du module *PIP MySQL* contient des informations comme l'URL de la base de données, le login de connexion, le mot de passe correspondant et la requête SQL à exécuter.
2. Transformer les données précédemment extraites pour que le système d'autorisation puisse les comprendre et les exploiter.
3. Etre facilement configuré par n'importe quel administrateur de sécurité de l'organisation.

6.3.3 Exemple de mise en œuvre de la méthodologie

En premier lieu il est intéressant de développer une interface graphique qui facilite la création d'un fichier de configuration pour un module PIP comme par exemple le module PIP MySQL. Ce fichier de configuration doit contenir des informations extraites et nécessaires pour la connexion à la

technologie de stockage (URL, login, mot de passe... pour MySQL). Ensuite, imaginons par exemple que le système d'autorisation attende du module PIP MySQL, des données de l'identifiant de format : « prénom+nom ». On constate que les informations « prénom » et « nom » sont bien présentes dans la base de données MySQL, mais que leur structure ne peut être exploitée dans un système d'autorisation. Le PIP MySQL doit pouvoir transformer par exemple la valeur « marilyn.monroe» extraite de la base de données dans un format « marilyn+monroe » attendu par le système d'autorisation. Ainsi, lors de la configuration du module PIP, il faut que ce problème de structure de données soit résolu, pour qu'à l'utilisation du module dans le système d'autorisation, la traduction des données soit effectuée automatiquement.

Pour satisfaire à ces nouvelles exigences, nous avons décidé d'utiliser la technologie des « expressions régulières ». Cette technologie permet de définir la structure syntaxique d'une chaîne de caractères. Les « expressions régulières » dans Java présentent des particularités caractéristiques dues à un caractère spécial commun : « \ ». Après étude de la technologie des expressions régulières, nous avons décidé d'y ajouter une convention d'écriture, toute variable sera ainsi incorporée à l'expression régulière : elle aura un nom symbolique encadré par « (* » et « *) ».

Lors de la configuration du module PIP MySQL, l'administrateur a décrit la structure syntaxique de la colonne « identifiant » de la façon suivante : « (*chaîne1*).(*chaîne2*)». Pour que notre module PIP puisse identifier chaîne1, et chaîne2 de la valeur « marilyn.monroe», l'administrateur doit affecter à chacune d'elles une « expression régulière » qui définit leur structure littérale au sein de la base de données. Une fois les chaînes de caractères « marilyn » et « monroe » identifiées, le PIP MySQL est capable de composer « marilyn+monroe » conformément au format attendu. La composition du fichier de configuration a été enrichie dans le but de remplir l'objectif (rappel : « structurer les données extraites de la base de données MySQL pour le système d'autorisations puisse les exploiter) :

- Paramètre : est le nom de l'attribut PIP MySQL écrit dans la politique pour lequel ce fichier de configuration soit utilisable. Le module PIP MySQL est choisi pour permettre à l'administrateur d'écrire des politiques contenant l'attribut PIP MySQL
- URL : adresse de la base de données MySQL nécessaire à l'extraction des données voulues
- Login : identifiant de connexion à la base de données
- Password : mot de passe de connexion à la base de données
- SQLrequest : requête SQL que le PIP MySQL devra exécuter sur la base de données pour en extraire les informations désirées
- DataTypeFormat : format requis par le système d'autorisation qui doit être au final retourné par le PIP, « (*prénom*)+(*nom*) » dans notre exemple ».
- DBColumnMap : liste des colonnes à extraire de la base de données, chacune étant associée à son format (exemple : la colonne « identifiant » est associée à « (*chaîne1*).(*chaîne2*) »).
- DataTypeMap : liste des références du DataTypeFormat associées à leur format final (exemple : (*prénom*)=(*chaîne1*) et (*nom*)=(*chaîne2*)).
- DataType : libellé du type de données de l'attribut associé au DataTypeFormat présent

L'interface graphique ou tout autre outil pour faciliter correctement la configuration des informations du module PIP doit être fournie. L'administrateur de sécurité utilise cet outil pour écrire les informations nécessaires pour tout nouvel attribut PIP ou toute extension du langage XACML.

6.4 Conclusion

Dans ce chapitre nous avons présenté comment créer des extensions du langage de politiques XACML. Nous avons proposé une méthodologie pour ajouter de nouveaux types de données et une autre pour ajouter de nouveaux attributs PIPs. Avec cette méthode, nous avons un moyen de faciliter le travail des administrateurs de sécurité. Ils ont la possibilité de définir une extension et de la configurer en particulier avec l'aide d'outils comme les ontologies pour configurer de nouveaux types de données configurables scalaires. Un autre avantage de la méthode est par voie de conséquence de faciliter la spécification de politiques contenant des extensions du langage.

De plus, les modules (avec fichiers de configurations ou non) seront utilisés par notre implémentation du moteur de décisions d'autorisation XACML extensible. Pour cela, nous allons montrer dans le prochain chapitre comment interpréter et exécuter les extensions du langage XACML dans notre système, en proposant tout d'abord le nouveau concept de « *politique auto-contenue* ». L'implémentation de ce concept doit satisfaire les deux exigences : *la réutilisabilité des modules des extensions du langage XACML, et la gestion du cycle de vie de ces modules*.

Chapitre7. Implémentation d'un cadriciel pour l'édition et l'exécution de politiques d'autorisation XACML

7.1 Politique auto-contenue

Dans le chapitre précédent, nous avons montré que les singularités propres à certaines situations conduisent à la conception de modules pour étendre le langage de politiques XACML. Ces modules peuvent être configurés par les administrateurs de sécurité pour être inclus dans le moteur de décisions d'autorisation XACML. Un moteur de décisions d'autorisation XACML basé sur une architecture de gestion à base de politiques (PBM) attribue à la fonction de prise de décisions, ou (PDP), la capacité d'interpréter les informations d'une politique. En d'autres termes, lorsque le langage de politique est étendu, le PDP doit également être étendu si l'on veut maintenir cette caractéristique d'interprétation et d'exécution. En conséquence, la plupart des implémentations XACML (citées précédemment dans la section 5.4 du chapitre 5) nécessitent que soient constituées des extensions à intégrer au PDP (ajouter des codes supplémentaires ou des modules externes), afin d'interpréter et d'exécuter une politique contenant des extensions. En effet, quand bien même toutes les implémentations du PDP respectent le standard XACML, ce n'est pas pour autant qu'elles sont en mesure de reconnaître un nouvel attribut, un nouveau type de données ou une nouvelle fonction qui aurait été ajoutée au langage. La programmation d'une extension qui sera installée au sein d'un PDP est donc obligatoire afin de lui donner la possibilité d'interpréter ces nouvelles informations.

L'incertitude peut être de mise et des problèmes de consistance peuvent voir le jour : lorsqu'un PDP est activé, on peut toujours se demander s'il est capable d'interpréter ou non telle ou telle extension donnée du langage lorsque l'on écrit une politique. D'après nos connaissances, aucune solution réelle n'existe pour répondre à ce problème. En conséquence, lorsqu'un administrateur écrit une politique, il lui appartient de vérifier si le PDP cible possède la capacité d'exécuter sa politique. Plus précisément, l'administrateur doit vérifier les points suivants lorsqu'il écrit la politique XACML :

- le PDP cible possède t-il le code PIP lui permettant de retrouver les attributs de sa politique ? Si oui, est-ce que le PIP possède la configuration nécessaire ?
- le PDP implémente-t-il tous les types de données et les fonctions utilisées dans la politique ?

Si le PDP ne possède pas ces caractéristiques, l'administrateur doit soit modifier sa politique, soit étendre le PDP. S'il modifie sa politique, il doit alors n'utiliser que les types de données ou des fonctions connues du PDP car en dehors des types de données et fonctions standards, il n'a aucune information sur l'existence d'éventuelles extensions du langage.

La réécriture d'une politique est peu efficace. Sans documentation facilement accessible, sur d'éventuelles extensions apportées à un PDP, l'administrateur court le risque de réécrire ce qui existe déjà. Qui plus est, sans gestion du cycle de vie des extensions, le PDP pourrait contenir des codes inutiles, si l'on ne prend pas garde de supprimer les codes ou modules externes implantés lors de l'extension d'un PDP. Cela pourrait donc nuire aux performances d'ensemble du système.

Dans notre travail, nous nous sommes attachés à proposer une solution architecturale pour traiter ces problématiques. Cette solution permet de justifier quand et pourquoi des modules d'extensions doivent être ajoutés ou supprimés du PDP. Qui plus est, au-delà de la gestion stricte de ce cycle de vie, nous avons imaginé une solution dynamique, pour que le système ne soit pas arrêté, puis relancé pour prendre en compte les extensions, ce qui dans le cas contraire nuirait aux principes mêmes de changements dynamiques de politiques.

Notre approche est de concevoir un système où un administrateur peut inclure dans une politique tous les éléments nécessaires à son exécution (Figure 46). Nous lui avons donné le nom de « *politique auto-contenue* » car dans cette approche :

- une politique auto-contenue inclut tous les codes ainsi que les configurations nécessaires pour que le PDP puisse exécuter des politiques avec des extensions. Par exemple, la politique inclut le module qui contient le code pour un PIP qui récupère les valeurs des attributs sur une base de données « MySQL » et la configuration nécessaire pour extraire et structurer les informations. Ainsi, la politique inclut le module qui donne le code pour implémenter le type de données non standard, et la configuration de ce type de données.
- L'implémentation du PDP inclut uniquement le code nécessaire au processus de prise de décisions de base du PDP. Il ne contient que le code lui permettant de prendre des décisions sur des politiques contenant des types de données et des fonctions du standard XACML e.g. SUNXACML PDP implémentation du standard [SUNXACML 2011].

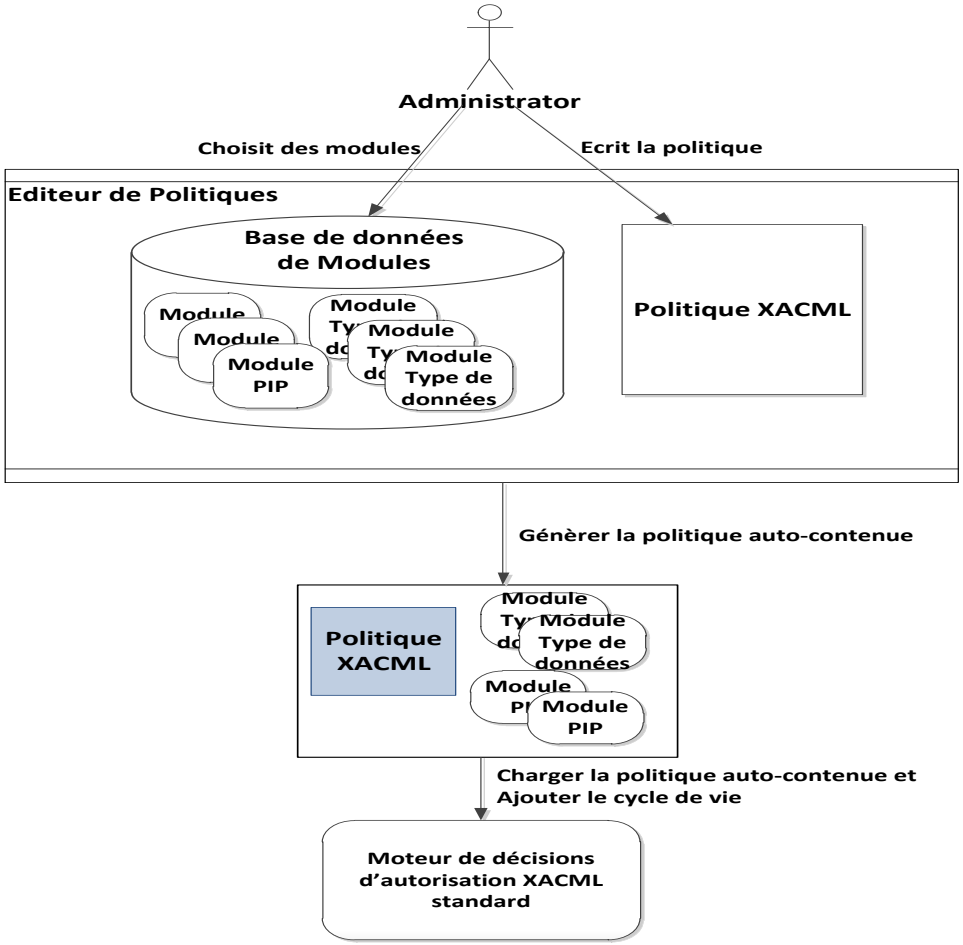

Figure 46 Concept d'une politique auto-contenue

Cette approche facilite la gestion du cycle de vie de l'implémentation du moteur de décisions d'autorisation. Chaque politique doit fournir toutes les informations nécessaires pour interpréter, exécuter, et gérer son cycle de vie dans le système d'autorisation. Quand une politique est chargée, les modules (codes et configurations) qui lui sont spécifiques sont installés dans le système d'autorisation. Lorsque la politique est retirée, les modules (codes et configurations) qui lui sont associés sont également retirés du système d'autorisation.

7.2 Les exigences d'une politique auto-contenue

Nous nous sommes fixés de répondre aux exigences suivantes pour que les politiques auto-contenues présentent un réel intérêt :

Exigence 1 (auto-suffisance) : Une politique auto-contenue doit être auto-suffisante. Une politique auto-contenue doit inclure toutes les informations nécessaires pour qu'elle soit exécutée par un moteur de décisions d'autorisation. Par exemple, la politique auto-contenue doit inclure le code et la configuration du PIP, la structuration et la configuration de données non standards afin que le moteur de décisions d'autorisation soit capable d'interpréter et d'exécuter la politique avec ses extensions.

Exigence 2 (auto-description) : Une politique auto-contenue doit contenir les informations pour faciliter la gestion de son cycle de vie. La politique auto-contenue doit fournir une description des éléments qu'elle contient (i.e. PIP, fonctions, données etc.) et la façon de les utiliser, et la liste des éléments qui doivent être installés pour son exécution. Le chargement d'une politique est réalisé en tenant compte de ces informations. De la même façon lorsqu'une politique est supprimée, la description sera utilisée pour décider des modules qui doivent être enlevés par le moteur d'autorisation. Par exemple, si un moteur de décisions installe un PIP pour que le PDP accède aux attributs du PIP et installe un module de type de données pour que le PDP évalue les fonctions de ce type de données, ces éléments devront être désinstallés lorsque la politique auto-contenue sera supprimée.

Exigence 3 (dynamicité): Les situations ubiquitaires étant de plus en plus nombreuses, il nous semble important que les changements en temps réel ou dynamiques soient répercutés au sein des moteurs de décisions d'autorisation. Ainsi, une politique auto-contenue doit pouvoir être installée et désinstallée dynamiquement. Cette exigence découle de la définition fondatrice de la gestion à base de politiques dans [Sloman et Lupu 2002]. Le système doit être capable de charger et décharger dynamiquement une politique auto-contenue sans avoir besoin d'arrêter ou de recoder le système.

7.3 Outillage pour la construction de politique auto-contenue

Dans notre système, nous avons retenu l'approche ABAC pour constituer des politiques XACML génériques et extensibles. Ainsi, l'architecture est basée sur la gestion à base de politiques (PBM) pour avoir un système unique flexible et indépendant de l'application gérée dès que possible. Au-delà des PBM et pour répondre au besoin de l'adaptabilité dynamique, notre architecture est également basée sur l'approche de programmation composants orientés services. Cette dernière approche nous permet de concevoir un système qui s'adapte aux exigences sans arrêter ou stopper le système, qui gère le cycle de vie de politiques auto-contenues dans le PDP, et qui résout la problématique de réutilisabilité des modules des extensions de politique auto-contenue au sein du système d'autorisation.

Cette approche offre à la fois les avantages d'adaptabilité dynamique, comme l'intégration et le dynamisme des architectures, mais aussi la réutilisabilité et la gestion des dépendances des modèles orientés composants [Cervantes et Hall 2004]. L'architecture orientée services est basée sur l'idée que les applications peuvent découvrir et invoquer des services pour accomplir certaines tâches [Papazoglou et al. 2007].

Rappelons que l'approche orientée services est organisée en trois parties : un fournisseur de service, un consommateur de service et un annuaire de services. Un annuaire de services typique se compose d'un ensemble d'entités donnant la possibilité d'accéder et de récupérer les informations sur les services d'intérêt [Tsai et al. 2007] pour des parties externes ou des consommateurs de services. Les fournisseurs de services publient leurs services à tout moment dans l'annuaire et les

consommateurs de services peuvent invoquer un service fournisseur enregistré dans le service annuaire pour un contrat ou propriété spécifique. En plus, l'approche orientée services permet aux services consommateurs d'être notifiés de façon dynamique des nouveaux services fournisseurs enregistrés dans le service annuaire.

Cependant, la programmation composants orientés services traite de la réutilisabilité de blocs logiques de programmes qui implémentent une ou plusieurs interfaces. Un logiciel orienté composants est un assemblage de composants. La notion d'interface étant très similaire à la notion d'interfaces de service, la programmation orientée composants a été utilisée pour mettre en œuvre des services. Un service peut être mis en œuvre par un ou plusieurs composants.

7.4 Proposition d'architecture d'un moteur de décisions d'autorisation XACML extensible

Nous présentons maintenant l'architecture globale de notre système d'autorisation générique et extensible (Figure 47). Cette architecture est composée de quatre entités : un éditeur de politiques auto-contenues (Self-contained Policy Editor), une interface d'administration de politiques auto-contenues (Policy Admin Interface), un moteur de base de décision d'autorisation (Core Authz Decision Engine), et un point d'accès de décision de base d'autorisation (Core Authz Decision Access Point).

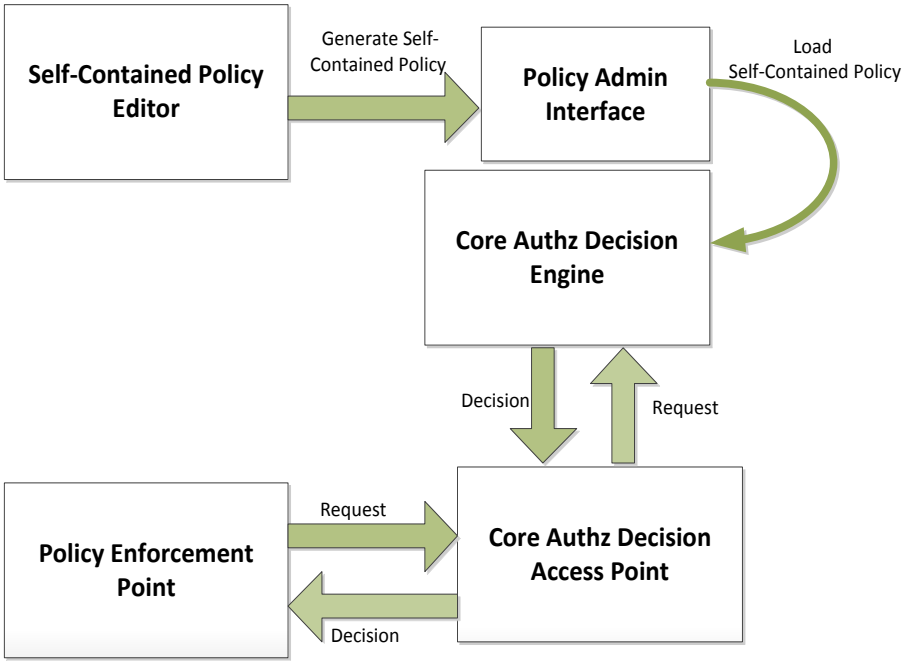

Figure 47 Architecture générale du système d'autorisation proposé

1. **Self-contained Policy Editor :**

   Il s'agit de l'éditeur de politiques auto-contenues. Il a été élaboré pour répondre aux intentions suivantes :

   - Faciliter l'écriture de politiques XACML avec des extensions du langage. Par exemple offrir un éditeur XML qui prend en compte les fonctionnalités d'auto-complétions pour les éléments XACML (fonctions, attributs, types de données...).
   - Choisir les modules des extensions du langage que la politique auto-contenue (en phase d'écriture) doit comprendre. Par exemple, l'administrateur choisit le module PIP

MySQL. Cela permet d'inclure dans les politiques XACML de nouveaux attributs PIP de type MySQL, et permet au moteur de décisions d'interpréter cette politique.

- Faciliter les fonctionnalités de configuration des modules. Par exemple, l'éditeur offre à l'administrateur de sécurité une interface graphique pour la configuration d'un module PIP MySQL.

Enfin, cet éditeur est utilisé pour l'édition de politiques auto-contenues. Il permet également de générer des fichiers de politiques auto-contenues compréhensibles ou compatibles avec un moteur de décisions d'autorisation afin d'être exécutées.

2. **Policy Admin Interface :**

   C'est l'interface qui prend le fichier de la politique auto-contenue générée par l'éditeur de politiques et qui réalise le chargement de la politique auto-contenue dans le moteur de décisions d'autorisation.

3. **Core Authz Decision Engine :**

   Ce module est celui qui supporte l'exécution des politiques auto-contenues et permet de gérer le cycle de vie des modules au sein du moteur de décisions d'autorisation XACML. Dans notre implémentation ce module est construit par un moteur de décisions d'autorisation XACML. Il comprend l'entité de prise de décisions d'autorisation responsable de l'exécution des politiques auto-contenues. Il implémente le standard XACML strictement. L'entité point d'administration est responsable de la gestion du cycle de vie des politiques auto-contenues qui ont été chargées. Ce module comprend également l'entité « context handler » responsable de la fourniture des requêtes XACML à l'entité de prise de décisions d'autorisation pour que cette dernière puisse évaluer avec les politiques XACML dans leur contexte.

4. **Core Authz Decision Access Point :**

   C'est un point d'accès exploité (par exemple une API, un socket, un serveur web...) pour relayer à l'entité « Core Authz Decision Engine » des requêtes provenant de demandeurs d'accès envoyées par l'entité PEP (Policy Enforcement Point). C'est en retour le point d'accès exploité pour restituer à l'entité PEP les décisions d'autorisation prises par l'entité « Core Authz Decision Engine ».

Après avoir décrit les éléments fonctionnels de notre système, nous proposons deux architectures : une pour l'étape d'édition d'une politique auto-contenue et une pour l'étape d'exécution d'une politique dans notre système.

7.4.1 Architecture d'édition d'une politique auto-contenue

La Figure 48 présente l'architecture de spécification de politiques auto-contenues basée sur l'approche de programmation composants orientés services. Dans cette architecture, nous séparons clairement les deux entités en interaction que sont l'éditeur de politiques auto-contenues « Self-contained Policy Editor » et la politique auto-contenue « Self-contained Policy ». Le composant « Self-contained Policy Editor » est un service consommateur qui invoque les services fournisseurs de composants de la politique auto-contenue afin de faciliter l'écriture de la politique auto-contenue par les administrateurs de sécurité.

Le résultat du travail assuré par le composant « Self-contained Policy Editor » est la génération d'un fichier contenant la « politique auto-contenue ». Ce fichier est prêt à être analysé et

traité par l'entité « Core Authz Decision Engine ». Ce fichier contient les composants illustrés dans la Figure 48.

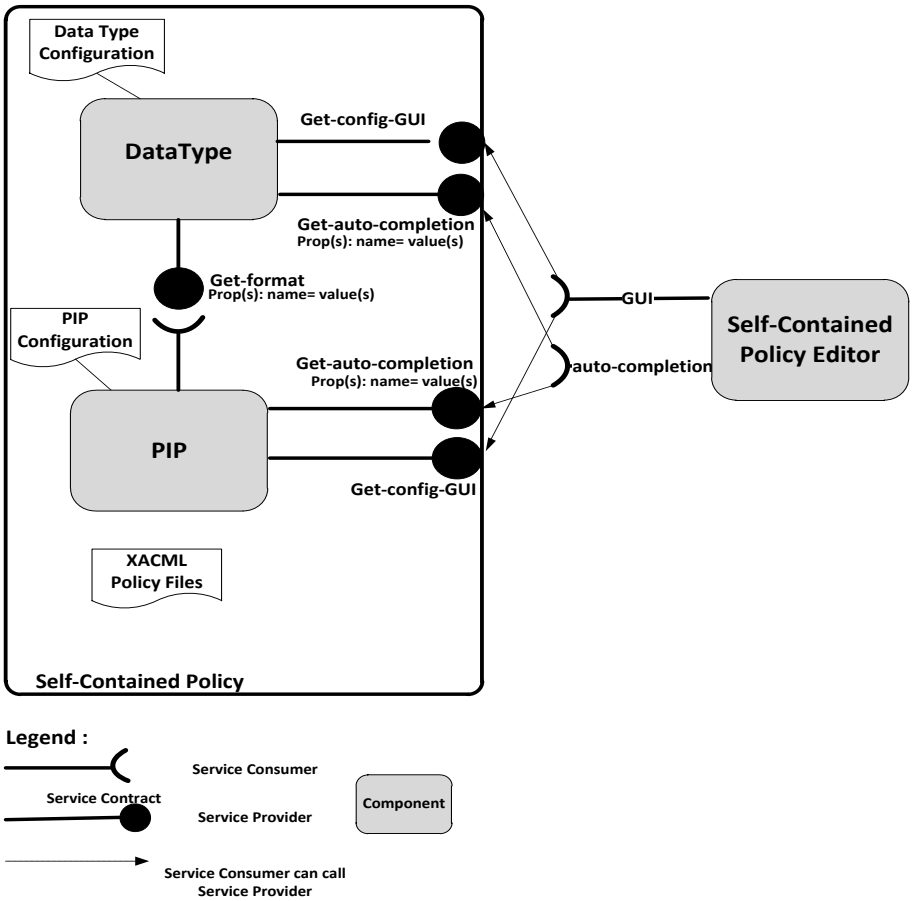

Figure 48 Architecture de la spécification d'une politique auto-contenue

La spécification d'une politique auto-contenue comprend trois types de composants : les composants d'accès à la politique (Policy Access), les composants PIP et les composants type de données (DataType). La raison d'être de l'éditeur de politiques auto-contenues est d'interagir avec les deux composants PIP et type de données pour permettre à un administrateur d'ajouter les fonctionnalités dont il a besoin.

PIP : Les composants PIP sont dédiés à la récupération d'attributs non présents dans le contexte de requêtes XACML depuis diverses sources externes (LDAP, MySQL, Microsoft SQL server....). Ils publient deux services fournisseurs pour l'éditeur de politiques qui sont « Get-config-GUI » et « Get-auto-completion ».

- Le service « Get-config-GUI » est invoqué par l'éditeur de politiques pour faciliter la création du fichier de configuration PIP. Chaque composant PIP est générique pour chaque technologie de stockage (un module PIP MySQL générique pour la technologie de stockage MySQL, un module PIP LDAP générique pour la technologie de stockage LDAP...). Ainsi, chaque composant a sa propre configuration. Par exemple la configuration d'un module PIP MySQL impose de connaitre l'adresse IP du serveur, le login/mot-de-passe, la requête SQL... Ce service offre une interface graphique pour aider l'administrateur à configurer ces informations pour le module PIP MySQL.
- Le service « Get-auto-completion » fournit de nombreuses informations pour les administrateurs de sécurité au moment des spécifications des politiques d'autorisation. Ce composant peut fournir des fonctionnalités d'auto-complétion pour éviter les erreurs durant l'écriture des politiques avec des extensions. Par exemple avec ce service l'éditeur

de politiques peut proposer à l'administrateur à travers des menus déroulants les noms possibles qui aident l'administrateur dans sa tâche d'écriture dans la politique d'un URN de l'identificateur d'attributs PIP, et d'un URN du type de données de l'attribut PIP.

Data Type : Les composants « Data Type » implémentent les types de données et les fonctions non-standards utilisées dans la politique XACML. Ils fournissent à l'éditeur deux services qui sont « Get-config-GUI » et « Get-auto-completion ».

- Le service « Get-config-GUI » comme dans le composant PIP offre à l'éditeur de politiques des interfaces graphiques pour configurer des types de données configurables scalaires ou composites. Un type de données configurable doit être configuré avant de pouvoir être référencé dans une politique. Chaque composant de type de données fournit des services par rapport au type de données qu'il implémente.
- Le service « Get-auto-completion » fournit de nombreuses informations pour les administrateurs de sécurité durant l'écriture de politiques avec des extensions du langage dans l'éditeur. Ce composant peut fournir à l'éditeur de politiques toutes les informations utiles dans le fichier de configuration pour les types de données configurables scalaires ou composites. Il peut fournir par exemple tous les noms possibles d'un URN de type de données, les instances de type de données, URN des fonctions... De plus, il peut fournir à l'éditeur toutes les valeurs possibles et la description de la syntaxe de chaque valeur pour un type de données non standard. Grace à ce service, l'éditeur de politiques peut fournir aux administrateurs de sécurité des fonctionnalités d'auto-complétion de politiques avec extensions du langage.
- Le troisième service « Get-format » : Il est invoqué par les composants PIP. Le rôle d'un composant PIP est de récupérer des informations (valeurs des attributs prises en considération dans une politique). Ces valeurs sont retournées dans des formats divers qui dépendent des sources technologiques qui ont permis leur création et stockage (Champ LDAP, table dans une base de données etc.). Ce service donne au composant PIP, la structure attendue par les types de données des attributs. Le composant PIP peut alors transformer ces informations pour qu'elles puissent être exploitées.

7.4.2 Architecture d'exécution d'une politique auto-contenue

La Figure 49 présente l'architecture de déploiement ou exécution de politiques auto-contenues basée sur l'approche de programmation composants orientés services. Le composant « Core Authz Decision Access Point » est un point d'accès entre le PEP et le moteur d'autorisation. Le moteur d'autorisation ou « Core Authz Decision Engine » qui supporte les politiques auto-contenues comprend trois entités actives : le PDP, Le Context-Handler et le PAP.

- L'entité **PDP** qui est un moteur de décision d'autorisation. Le rôle du composant PDP est similaire à celui présent dans l'architecture XACML c.à.d. la prise de la décision d'autorisation en évaluant les politiques avec des requêtes. Il fournit le service « Get-decision » pour donner les décisions d'autorisation prises.
- Le **Context-Handler** récupère la requête d'une entité du composant « Core Authz Decision Access Point » et la transforme en format XACML. Ensuite, il fournit le service « Get-XACML-request » pour donner la requête XACML aux autres composants.
- Le rôle du **PAP** est de gérer le cycle de vie de la politique auto-contenue en fournissant le service « PAP-admin » qui est responsable de :

  1) **Charger une politique auto-contenue :** le PAP charge tous les composants de la politique dans le moteur d'autorisation. Cependant, les services ne sont pas enregistrés dans l'annuaire.

2) **Activer une politique auto-contenue :** le PAP active tous les composants de la politique et le PDP est notifié qu'il peut exécuter la politique.

3) **Désactiver une politique auto-contenue :** Les services sont supprimés de l'annuaire et le PDP n'exécute plus la politique.

4) **Supprimer la politique auto-contenue :** les composants de la politique auto contenue sont enlevés de l'environnement du moteur d'autorisation.

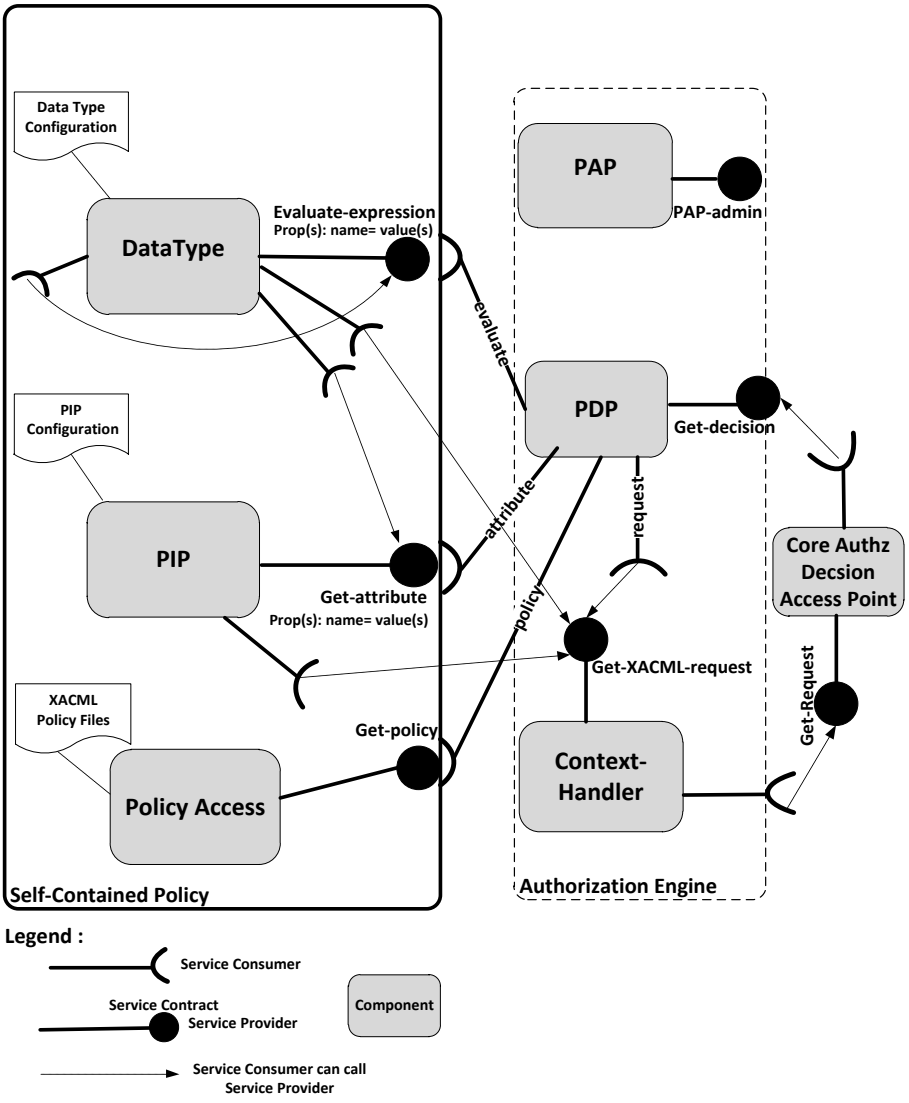

Figure 49 Architecture de déploiement d'une politique auto-contenue

Pour le déploiement, l'architecture d'une politique auto-contenue comprend trois types de composants : les composants d'accès à la politique (Policy Access), les composants PIP et les composants types de données.

Policy Access : Le composant d'accès à la politique fournit un seul service, appelé « Get-policy », permettant au PDP d'accéder aux politiques XACML. Lorsque le composant PAP active la politique auto-contenue dans le moteur, ce service est automatiquement enregistré. Le PDP est alors notifié qu'il peut charger les politiques XACML pour pouvoir l'utiliser afin de mettre en œuvre le processus de décisions d'autorisation.

PIP : Les composants PIP sont dédiés à la récupération d'attributs dont il faut connaitre la valeur mais qui ne figurent pas dans le contexte des requêtes XACML. Ils peuvent provenir de plusieurs sources. Les composants PIP publient le service « Get-attribute » qui ajoute les couples <attribut, valeur> dans le contexte des requêtes. Un composant PIP peut être responsable d'un ou plusieurs attributs. En conséquence, chaque PIP doit indiquer le nom du ou des attributs (en XACML il s'agit d'un URN). Il est responsable de retrouver leurs valeurs au moment de la phase d'enregistrement au service annuaire. L'approche composants permet de réutiliser des fonctionnalités couramment employées dans des politiques comme l'accès à des informations dans une base MySQL ou dans un répertoire LDAP. Ainsi, si un composant PIP MySQL existe, il suffit de le configurer pour pouvoir l'utiliser dans une politique XACML. Les composants PIP peuvent être mis à disposition afin de satisfaire à des utilisations par d'autres composants tels que le PDP ou des types de données. En plus, ce composant peut invoquer le service fourni par le composant « Context-Handler » (« Get-XACML-request ») pour utiliser des informations dans des requêtes XACML et qui sera utile pour la récupération de valeurs d'attributs. De plus, le fichier de configuration du PIP contient toutes les informations nécessaires pour que le composant PIP soit capable d'extraire et de transformer toutes les valeurs de l'attribut au format attendu ou requis par le type de donnée de cet attribut.

Data Type : Les composants de types de données implémentent les types de données et les fonctions non-standards utilisées dans la politique XACML. Les composants types de données doivent fournir les services pour permettre au composant PDP d'évaluer les expressions dans la politique grâce au service « Evaluate-expression ». Ils indiquent au PDP l'ensemble des fonctions (le nom d'une fonction est un URN unique dans XACML) qu'ils implémentent dans le contrat ou la propriété liée au service. Il n'est pas nécessaire d'indiquer les types de données pris en charge car XACML est un langage préfixé. Ainsi, dès que le PDP trouve une fonction non standard dans la politique il invoque le service associé à cette fonction en lui passant en paramètre tout le bloc d'expression à évaluer. Ce processus est récursif. Si le composant pour la fonction « XYZ » trouve une sous expression <Apply FunctionId:"urn:ABC">…</Apply>, il peut invoquer le service associé à cette fonction de la même manière, etc. De plus, un composant type de données peut aussi invoquer le service « Get-XACML-request » ou le service « Get-attribute » d'un composant PIP au besoin. Tout comme les composants PIP, les composants types de données peuvent nécessiter des informations qui résident dans des fichiers de configurations des types de données qu'il implémente [Cheaito et al. 2010a]).

7.5 Scenario d'une politique auto-contenue

Considérons que l'Université Paul Sabatier (UPS) (Figure 50) propose à ses étudiants qui appartiennent au groupe « developers » la permission d'accéder au site web « www.example.com ». La politique d'autorisation est écrite formellement par l'administrateur de sécurité de l'organisation « UPS » comme : « Seuls les étudiants de l'organisation UPS qui sont développeurs peuvent accéder au site web www.example.com ». Comme nous l'avons indiqué dans le descriptif des scenarii, l'organisation UPS délivre trois types de diplômes : « phd », « master », et « licence ». La hiérarchie de rôles de l'UPS est définie par l'administrateur de sécurité et représentée dans la **Figure 50**. En se reportant à la hiérarchie donnée, tous les utilisateurs de l'organisation « UPS » qui ont le rôle « phd-UPS », ou « master-UPS », ou « licence-UPS » peuvent accéder au site web, car ces trois rôles sont des sous-rôles du rôle « étudiant-UPS ».

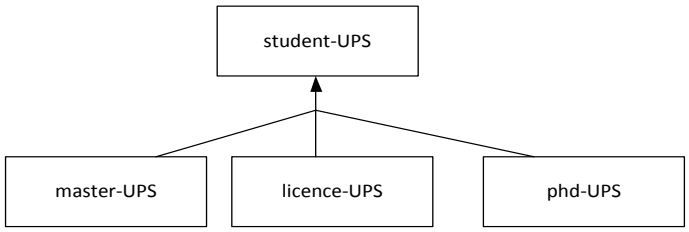

Figure 50 Hiérarchie de rôles dans UPS

L'organisation « UPS » désire restreindre uniquement à ses étudiants l'accès à ces vues détaillées. Tous les comptes utilisateurs de l'organisation UPS sont stockés dans une base de données « MySQL » appelé « SCP_DB ». Les comptes des utilisateurs qui sont étudiants de « UPS » sont différenciés des autres comptes en mettant en place un champ appelé « roleorg » qui contient le rôle de l'étudiant dans l'organisation d'origine. L'administrateur de sécurité renseigne les informations sur les comptes des utilisateurs étudiants de l'UPS dans la base de données « SCP_DB » dans une table appelée « RoleOrgInfo » (Figure 51). Les valeurs de cette table sont stockées dans une seule colonne avec structure « chaine1@chaine2 », où chaine1 est le nom du rôle que peut posséder un étudiant, et chaine2 est le nom de l'organisation dont dépend l'etudiant.

userJob	userName
phd@UPS	samer
licence@UPS	mathieu
phd@UPS	marwan
student@UPS	john

Figure 51 Exemple d'une table MySQL

Pour appliquer ce scenario dans le système d'autorisation de type XACML, l'administrateur doit étendre le langage de politiques XACML pour prendre en considération le rôle et l'attribut PIP « roleorg ». L'administrateur choisit de créer le module « RoleOrg » pour implémenter le type de donnée pour rôle, et de créer le module « MySQL PIP » pour récupérer des valeurs de l'attribut « roleorg ». La politique auto-contenue de ce scénario doit inclure dans la politique XACML, un module pour implémenter le type de données configurable scalaire « RoleOrg » et sa fonction « isAtLeast » qui représente la relation d'héritage de rôles, et un module « MySQL » PIP pour implémenter l'attribut « roleorg ». Chaque module doit être configuré par l'administrateur de sécurité pour que la politique auto-contenue (Figure 52) soit prête à être chargée dans le moteur de décisions d'autorisation.

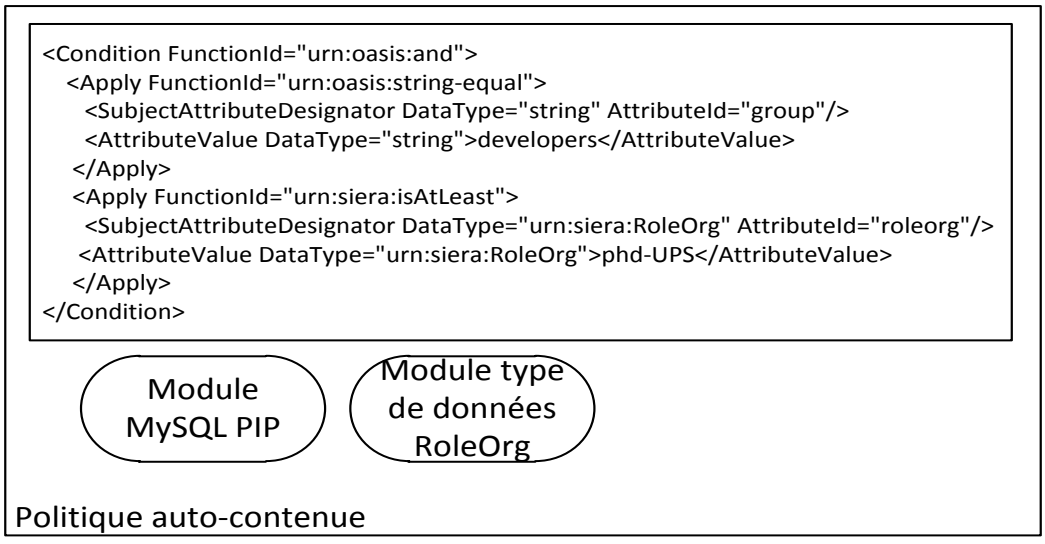

Figure 52 Politique auto-contenue du scenario

7.5.1 Application du scenario

Dans cette section, nous présentons le processus de spécification et le processus d'exécution du scénario ci-dessus dans notre système.

7.5.1.1 Architecture d'édition de la politique auto-contenue

Maintenant, nous présentons le processus de spécification de ce scenario dans notre éditeur de politiques auto-contenues. Tout d'abord, la politique auto-contenue est composée des composants ou modules suivants (Figure 53):

- Un composant pour implémenter le type de donnée « RoleOrg ». Ce composant implémente aussi la fonction d'héritage ou relation d'ordre « isAtLeast ». La configuration contient toutes les informations nécessaires sur ce type de donnée pour être utilisé par d'autres composants.
- Un composant PIP dont la fonction principale est d'accéder à une base de données MySQL et de réaliser une requête SQL. La configuration du PIP MySQL contient des informations telles que l'adresse IP du serveur MySQL, le login/mot de passe, la requête SQL à exécuter... Ces informations sont utilisées par le composant PIP pour récupérer les valeurs de l'attribut PIP « roleorg ».
- Le fichier de la politique XACML, où la politique est spécifiée avec des attributs ABAC comme suit : isEqual(group(Sujet), « developers ») ∧ isAtLeast(roleorg(Sujet), « phd-UPS ») ∧ isEqual(nom(Action)= « acceder ») ∧ isEqual(nom(Ressource), « http://www.example.com »).

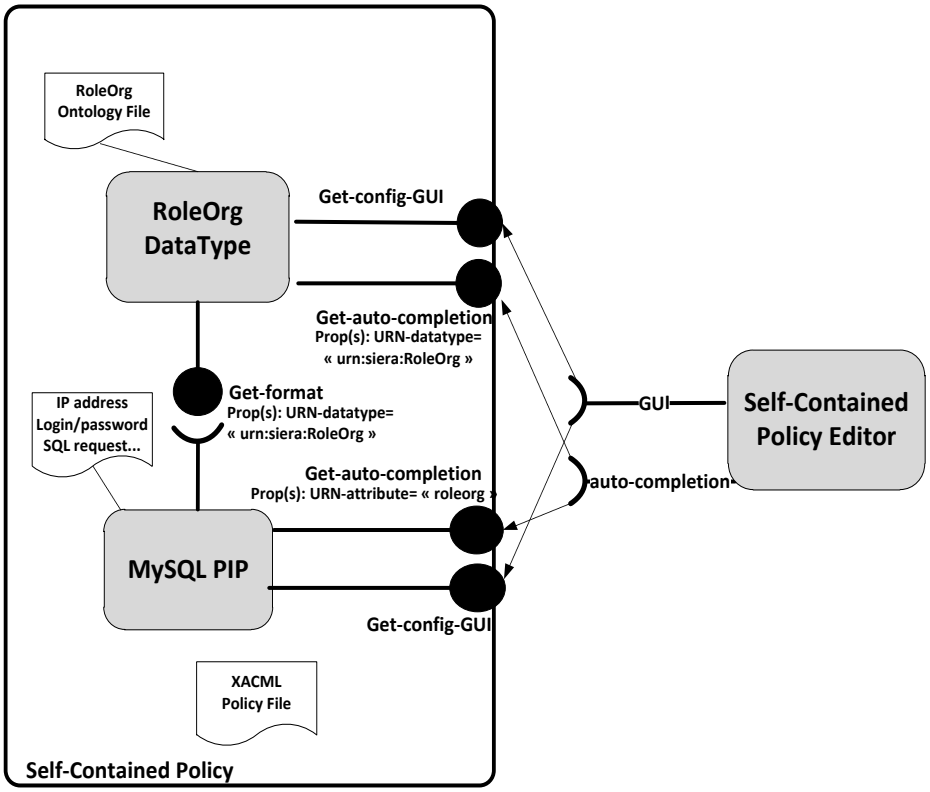

Figure 53 Architecture d'édition de la politique auto-contenue du scenario

Lorsque notre éditeur de politiques auto-contenues est lancé, l'administrateur peut commencer à créer des projets qui représentent l'interface graphique de base d'une politique auto-contenue. Concernant les projets, l'administrateur doit alors retrouver à chaque création d'un projet, trois dossiers (Figure 54) par défaut :

- Le dossier « Policies » qui contient toutes les politiques XACML créées ou ajoutées par l'administrateur de sécurité

- Le dossier « DataTypes » qui contient une représentation graphique de tous les modules pour les types de données non-standards dans les politiques XACML
- Le dossier « PIP » qui contient une représentation graphique de tous les modules PIP pour les nouveaux attributs PIP dans les politiques XACML

Figure 54 Projet d'une politique auto-contenue

L''administrateur de sécurité commence alors à choisir et à charger les modules qui représentent les exigences ou extensions du langage XACML nécessaires pour son organisation. Dans ce scenario, l'administrateur de sécurité doit charger dans l'éditeur de politiques les deux modules « RoleOrg » et « MySQL PIP ». Selon notre scenario, le processus d'édition d'une politique auto-contenue dans l'éditeur est composé de trois étapes :

1- **Configurer le composant de type de donnée non standard « RoleOrg »** : Le service qui est invoqué par l'éditeur est « Get-config-GUI ». Ce service donne la première fenêtre qui permet à l'administrateur de sécurité de configurer le type de données configurable scalaire « RoleOrg » comme présenté dans la Figure 55.

Figure 55 Configuration d'un type de données configurable scalaire

L'interface graphique donne la possibilité à l'administrateur de sécurité d'ajouter les instances du type de données « RoleOrg » suivant la méthode décrite précédemment pour ce type de données. Pour le scenario en question, l'administrateur écrit le nom de l'instance de ce type de données « UPSRole » et choisit le fichier de configuration (fichier d'ontologie) par le menu déroulant illustré dans la Figure 56. Ce fichier contient toutes les informations nécessaires pour écrire des politiques XACML avec ce type de données. Enfin, l'administrateur clique sur le bouton « save » de la première fenêtre (Figure 55) pour finaliser la configuration du type de données « RoleOrg ».

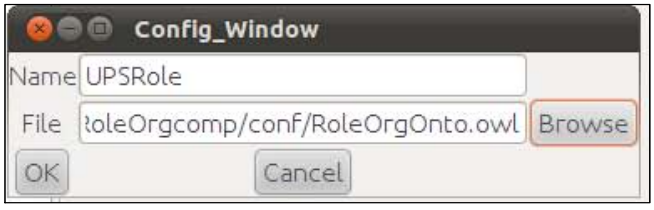

Figure 56 Ajouter un fichier de configuration de type de données

2- **Configurer le composant MySQL PIP de l'attribut « roleorg » :** Ce composant PIP fournit deux services à l'éditeur de politiques. « Get-config-GUI » est appelé par l'éditeur, quand l'administrateur de sécurité a besoin de configurer l'attribut PIP « roleorg ». Le service « Get-format » est mis en œuvre pour donner la structure ou le format attendu par le type de données « RoleOrg » au composant « MySQL PIP ».

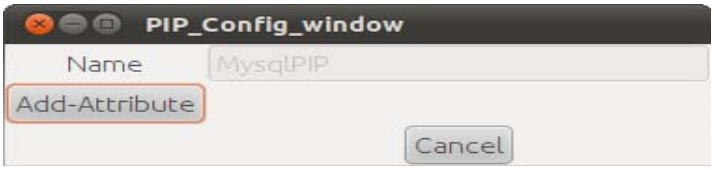

Figure 57 Première fenêtre pour configurer un PIP MySQL

Dans le scenario décrit, l'administrateur a besoin de configurer l'attribut PIP « roleorg » dans l'éditeur de politiques. Le composant « Get-config-GUI » est invoqué et donne accès à la fenêtre dans la Figure 57. Le processus associé à cette fenêtre permet d'ajouter l'attribut « roleorg » du PIP MySQL en utilisant le bouton « Add-Attribute ». Cela nous amène au formulaire suivant dans la Figure 58 qui permet de renseigner différents champs nécessaires à la configuration du PIP MySQL :

- Name : nom de l'attribut PIP, pour le scenario, le nom est « roleorg »
- URL : adresse de la base de données MySQL
- DataType : sélection dans une liste d'un type de données existant, qui détermine le format que retournera notre PIP. Pour le scenario, l'administrateur choisit le type de données « urn :siera :RoleOrg » pour « roleorg »
- Login : identifiant de connexion à la base de données
- Pwd : mot de passe de connexion à la base de données (non renseigné dans l'exemple donné car inexistant pour la base actuelle)
- SQLrequest : requête SQL que notre PIP exécutera dans le but d'extraire les informations voulues. On remarque ici la présence d'une convention d'écriture mise en place : « $urn:oasis:names:tc:xacml:1.0:subject-id$ ». En effet les « $ » encadrent la référence à un attribut dont la valeur existe dans la requête du demandeur d'accès XACML et nécessaire à l'exécution de la requête SQL. Cet attribut est le lien entre la valeur du nom de l'utilisateur dans la requête XACML et les noms des utilisateurs stockés dans la base de données MySQL pour l'organisation UPS.

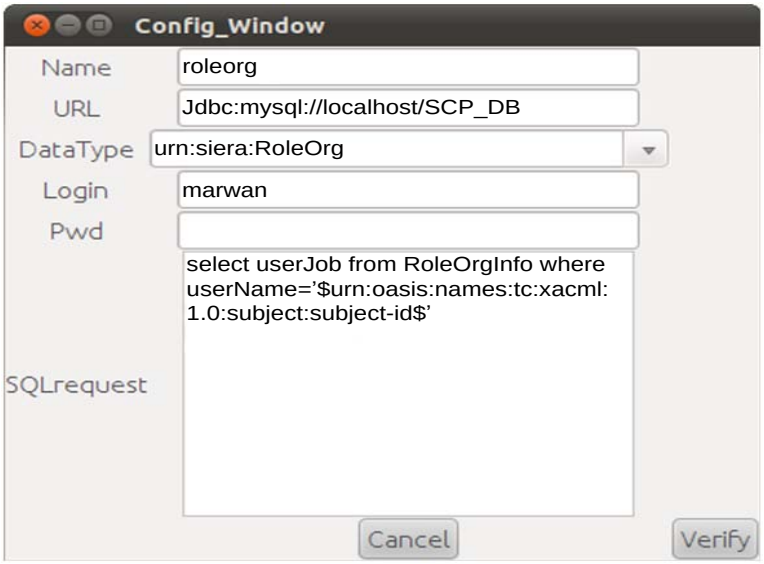

Figure 58 Les informations à spécifier pour un PIP MySQL

Les champs URL, Login, Pwd et SQLrequest sont exploités pour interroger la base de données MySQL de l'organisation « UPS ». Le bouton « Verify » permet à l'administrateur d'accéder à la fenêtre (Figure 59) qui permet de tester l'extraction d'informations de la base de données, et donc de valider ou d'invalider les données saisies jusque-là en suivant les objectifs décrits dans la méthodologie du PIP.

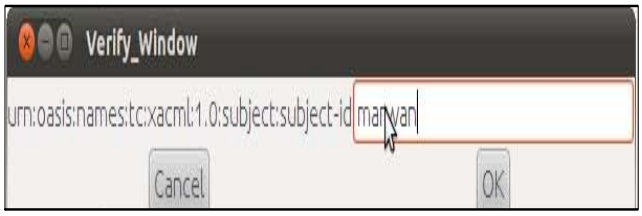

Figure 59 Tester l'extraction d'informations de la base de données

Une valeur de test doit être donnée à la référence « urn:oasis:subject-id » de la requête SQL. En cliquant sur OK, l'utilisateur exécute la requête SQL suivante : « select userJob from RoleOrgInfo where userName='marwan'». Cette requête permet de sélectionner toutes les lignes de la base de données dont la colonne « userName » a pour valeur « marwan », et d'en extraire les valeurs de la colonne « userJob ». A l'affichage de la fenêtre dans la Figure 60, l'administrateur constate que l'extraction de données est un succès. La configuration est jusque-là valide. Le premier objectif de notre PIP MySQL est atteint. Cette fenêtre est divisée en deux parties distinctes :

- La première est le résultat de la requête SQL précédente. Afin de ne pas alourdir le traitement et l'affichage, seule la première ligne extraite de la base de données est affichée.
- La seconde permet à l'utilisateur de définir le format de la colonne « userJob » de la base de données, ici : (*userRole*) pour « chaine1 » de la colonne (*userOrg*) pour « chaine2 » de la colonne « userJob » séparée par la référence « @ ».

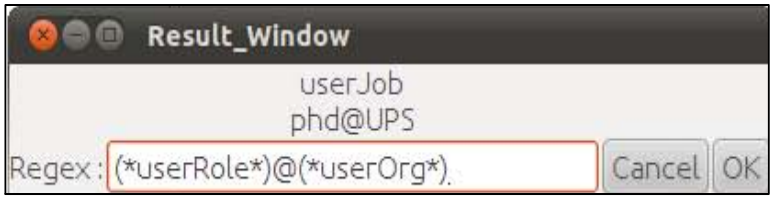

Figure 60 Structure des valeurs provenant dans une base de données MySQL

En appuyant sur le bouton « OK », le service « Get-Format » est alors invoqué par le composant « MySQL PIP » pour récupérer le format attendu du type de données « RoleOrg ». L'administrateur voit la fenêtre (Figure 61) qui permet de sélectionner le format du type de données « RoleOrg » désiré pour l'attribut « roleorg ». Il identifie le format final des données extraites de la base. Dans le scenario présenté, le format de « RoleOrg » qui lui est associé est « (*Role*)-(*Org*)». L'administrateur doit maintenant allouer à chaque référence de ce format, un agencement des références extraites de la base de données. Il lui est possible d'utiliser le bouton « Define » comme précédemment afin d'accéder à la liste des références disponibles. Le champ « Org » fait référence au (*userOrg*), et le champ « Rôle » fait référence à (*userRole*).

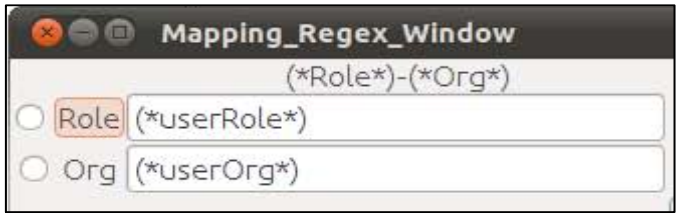

Figure 61 Format attendue par type de données

Cette fenêtre permet à l'administrateur de contrôler la validité de la configuration à laquelle il vient d'aboutir en visualisant les valeurs que prendront les références du format final du type de données. Ces informations seront exploitées par le processus de prise de décisions du moteur de décisions d'autorisation, notamment lorsque ce dernier sera amené à transformer les données de la base de données « MySQL » au format attendu de type de données. Ensuite, l'administrateur accède à la fenêtre principale (Figure 62) de configuration qui contient la liste des attributs définis pour MySQL PIP. Dans le cas présent cette liste ne comporte qu'un élément, l'attribut « roleorg » précédemment créé avec la possibilité d'en ajouter, d'en modifier ou encore d'en supprimer.

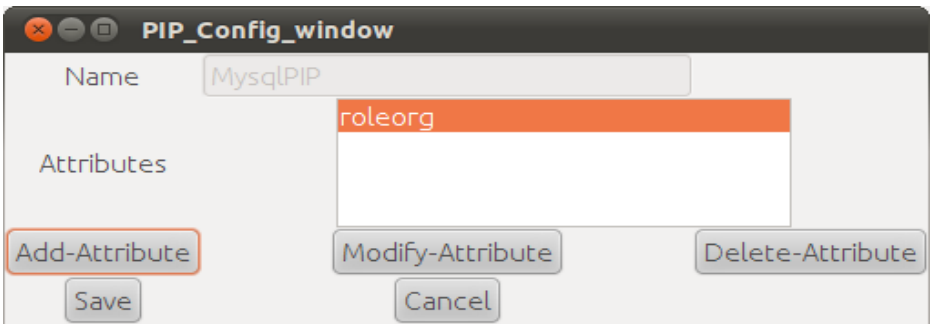

Figure 62 Fenêtre principale pour la configuration des attributs PIP

Enfin, en cliquant le bouton « Save », l'administrateur termine la configuration du module MySQL PIP. Le fichier de configuration est alors généré et peut alors être exploité par le moteur de décisions d'autorisation.

3- Faciliter l'écriture de la politique auto-contenue :

Une fois que la configuration des modules des extensions du langage est achevée, l'éditeur de politiques invoque le service « Get auto-completion » pour chaque composant des modules d'extension pour offrir à l'administrateur de sécurité des fonctionnalités d'auto-complétion concernant l'écriture du type de données non standard « RoleOrg » et de l'attribut PIP « roleorg ». Les politiques XACML étant des documents XML, nous avons déjà souligné que tout éditeur de XML peut être utilisé. Parmi les éditeurs, nous avons choisi l'éditeur XML dans Netbeans [Netbeans 2011], car il fournit déjà des fonctionnalités pour faciliter l'écriture de documents XML. Il permet en particulier de spécifier des politiques XACML conformément au standard et peut donner des fonctionnalités pour l'écriture de politiques XACML non standard, ce qui est par exemple le cas dans le scenario décrit. Cet éditeur propose tous les tags, les éléments et les attributs XML possibles pour des politiques XACML en respectant le schéma du standard XACML.

Dans notre scenario l'éditeur invoque le service « Get auto-completion » pour offrir des fonctionnalités d'auto-complétion pour le type de données « RoleOrg ». Cela permet de limiter les erreurs typographiques durant l'écriture de type de données par l'administrateur de sécurité. De plus, l'administrateur de sécurité peut bénéficier de toutes les informations venant du fichier de configuration de type de données configurable scalaire. Par exemple, l'éditeur de politiques peut proposer aux administrateurs de sécurité toutes les valeurs possibles pour le type de données « RoleOrg » avec une description syntaxique de chaque valeur sélectionnée (cf Figure 63).

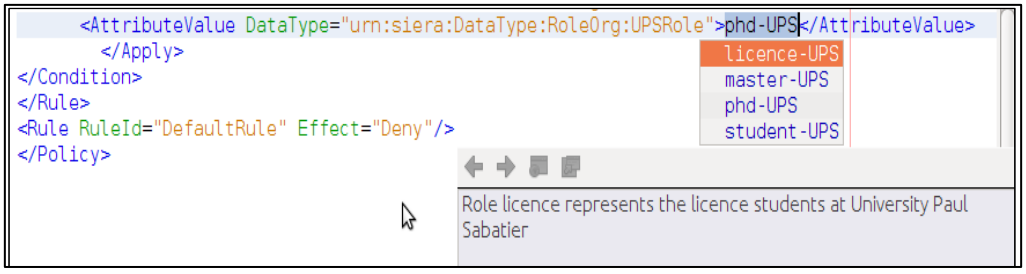

Figure 63 Liste des valeurs possibles pour un type de données configurable scalaire

L'éditeur de politiques peut également aider l'administrateur de sécurité dans l'écriture des noms d'URN de l'instance « UPSRole » de type de données « RoleOrg » , idem pour l'URN de la fonction « isAtLeast », ainsi que des descriptions syntaxiques de chaque URN comme on le voit pour « isAtLeast » dans la Figure 64. Les mêmes fonctionnalités sont offertes par l'éditeur de politiques durant la phase d'écriture de l'attribut PIP « roleorg ». L'éditeur invoque le service « Get auto-completion » du composant « MySQL PIP » pour atteindre ces fonctionnalités.

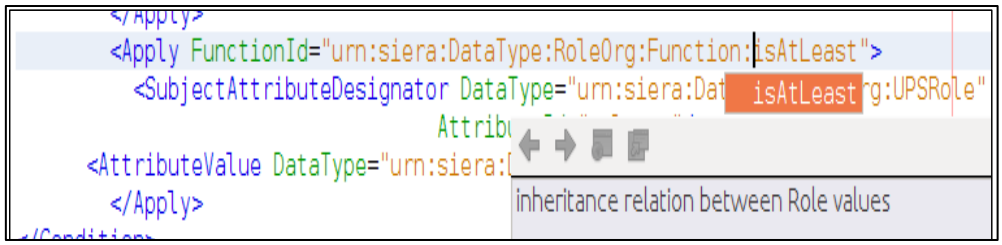

Figure 64 Liste de noms possibles d'URN XACML

Enfin, quand l'édition de la politique auto-contenue est terminée, l'administrateur de sécurité peut générer un fichier pour la politique auto-contenue du scenario. Cette politique (contenant tous les composants, fichiers de configurations, et politiques XACML) est compatible et compréhensible par le moteur de décisions d'autorisation ou « Core Authz Decision Engine » qui va l'exécuter.

7.5.1.2 Architecture d'exécution de la politique auto-contenue

Nous présentons maintenant le processus de l'exécution du scenario dans la section 7.5. La politique auto-contenue contient les composants suivants (Figure 65) :

- Un composant pour implémenter le type de données « RoleOrg ». Ce composant implémente également la fonction « isAtLeast »
- Un composant MySQL PIP dont la fonction principale est d'accéder à une base de données MySQL et de récupérer la valeur de l'attribut « roleorg » dans la politique XACML
- Un composant d'accès à la politique pour récupérer le fichier de la politique XACML

Lorsque le composant « PAP Interface » invoque le service « PAP-admin » du composant « PAP » pour charger la politique auto-contenue, il « installe » tous les composants de cette politique. Puis, lorsque la politique auto-contenue est activée, il « active » tous les composants qui publient leurs services (le dernier étant « Get-policy » service). Quand tous les services sont activés, le composant PDP est alors notifié et peut invoquer les services de la politique auto-contenue. Le PDP maintenant, peut évaluer les requêtes XACML en fonction de cette politique XACML en invoquant le service « Get-policy » pour récupérer et charger le fichier de la politique. Selon notre exemple, le processus d'évaluation d'autorisation par PDP est le suivant (Figure 65 et Figure 66) :

- La requête XACML contient l'attribut « subject-id »
- Evaluation de l'expression « expression 1 » : Il s'agit de la fonction « urn :xacml :and ». Le PDP peut évaluer cette fonction, car c'est une fonction standard XACML
- Evaluation de l'expression « expression 2 » : il s'agit de la fonction "urn:xacml:string:equals". Le PDP peut aussi évaluer cette fonction, car c'est une fonction standard XACML
- Evaluation de l'expression « expression 3 » : il s'agit de la fonction « urn:siera:isAtLeast ». Le PDP sait que cette fonction n'est pas standard. En conséquence, il appelle le service « Evaluate-expression » (avec la propriété Function-URN= urn:siera :isAtLeast') et en donnant l'expression « expression 3 » comme paramètre d'entrée. Le composant du type de donnée « RoleOrg » reçoit toute l'expression « expression 3 » et l'évalue en fonction de la requête XACML. Dans l'expression « expression 2 », le composant « RoleOrg »voit qu'il a besoin d'évaluer l'attribut « roleorg » qui n'existe pas dans la requête XACML. Alors, il appelle le service « Get-attribute » (avec la propriété du service URN-Attribute='roleorg'). Ce service est responsable de la récupération les valeurs de l'attribut dans une base de données MySQL

et en fonction de la requête XACML (pour récupérer le nom d'utilisateur connu « subject-id » dans la requête XACML). Les valeurs récupérées de l'attribut PIP « roleorg » sont ensuite transformées au format attendu du type de donnée « RoleOrg », en utilisant les informations stockées dans le fichier de configuration du composant « MySQL PIP ». Maintenant, le composant « RoleOrg » peut évaluer toute l'expression (expression 3). Ensuite le composant du type de données envoie le résultat d'évaluation au PDP

- Evaluation de l'expression « expression 1 » : Après que le PDP ait reçu le résultat de l'expression « expression 3 ». Le PDP peut alors évaluer toute l'expression « expression 1 » et prendre sa décision d'autorisation

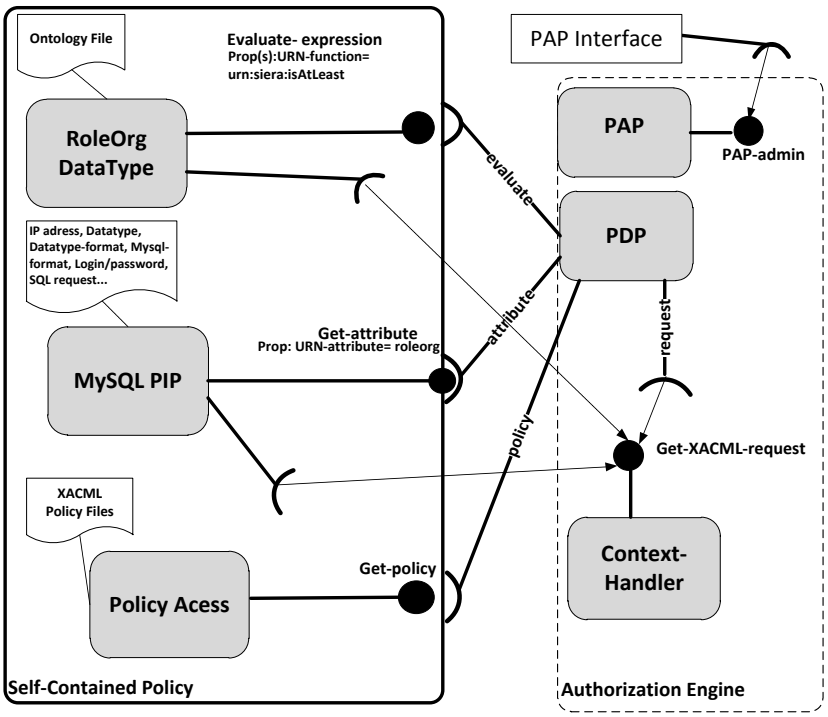

Figure 65 Architecture d'exécution de la politique auto-contenue du scénario

```
<Condition FunctionId="urn:oasis:and">
  <Apply FunctionId="urn:oasis:string-equal">
    <SubjectAttributeDesignator DataType="string"
AttributeId="group"/>
    <AttributeValue DataType="string">developers</
AttributeValue>
  </Apply>
  <Apply FunctionId="urn:siera:isAtLeast">
    <SubjectAttributeDesignator DataType="urn:siera:RoleOrg"
AttributeId="roleorg"/>
    <AttributeValue DataType="urn:siera:RoleOrg">phd-UPS</
AttributeValue>
  </Apply>
</Condition>
```

expression1 — expression2 — expression3

Figure 66 La décomposition de notre politique

7.6 Implémentation

Nous avons développé notre système basé sur l'architecture de programmation composants orientés services, en utilisant le cadriciel OSGi «Open Services Gateway Initiative » [OSGi 2011]

[Tavares et Valente 2008]. Nous expliquons dans cette section comment nous répondons aux exigences de la politique auto-contenue. Ensuite, nous expliquons le processus de spécification et d'exécution d'une politique auto-contenue dans notre système. Notre prototype nous a également permis d'évaluer et d'analyser l'impact de performance de l'utilisation de la technologie OSGi pour le processus de décision du contrôle d'accès par notre système.

7.6.1 Description

OSGi est une technologie promue par l'Alliance OSGi pour la construction de systèmes dynamiques et modulaires en Java. Le cœur de cette spécification est un cadriciel (framework) qui définit un modèle de gestion du cycle de vie des composants, un répertoire ou annuaire (registry) de services, un environnement d'exécution et des modules.

Les composants appelés « bundles » dans OSGi, peuvent être installés, arrêtés, démarrés, mis à jour et désinstallés de manière distante sans nécessiter de redémarrage c.à.d. dynamiquement. Ils sont l'unité physique de déploiement de Java contenant les fichiers de classes Java et les ressources associées comme les images, les icônes... Ils comprennent également un fichier de manifeste « manifest-file » pour déclarer les paquets du bundle d'import/d'export (c'est à dire les dépendances entre bundles) et les autres informations utiles. La gestion du cycle de vie est effectuée à travers une API en appliquant une politique de gestion des téléchargements distants. Le répertoire (registry ou annuaire) de services permet aux bundles de détecter l'addition de nouveaux services, ou la suppression de services et de s'y adapter.

De plus, les « bundles » peuvent fournir des services aux autres « bundles ». Un service est un objet Java standard qui est publié dans l'annuaire de services d'OSGi natif à l'aide d'un ou plusieurs types d'interface et des propriétés. Les propriétés de services sont constituées de paires <attribute,value> pour différencier les implémentations de service. Nous avons utilisé cette fonction pour indiquer quel URN est manipulé par les services de type de données ou PIP. Par exemple, la propriété associée au service « Evaluate-expression » pour un type de donnée est l'URN de ses fonctions. Cette fonction est utile par exemple pour le composant PDP, car il sait quel composant type de donnée gère quelle expression.

Un aspect intéressant de l'annuaire de services est sa nature dynamique. Dès qu'un bundle « A » publie un service dont un autre bundle « B » est à la recherche, l'annuaire de services va lier ces deux bundles ou composants. Cette fonctionnalité donne la possibilité d'implémenter la notification dynamique de PDP quand un « Get-policy » service est publié/non-publié, ce qui est une exigence de notre système en regard de la gestion du cycle de vie des politiques auto-contenues.

De plus, le cadre OSGi fournit des mécanismes de gestion pour installer, activer, désactiver, mettre à jour, et supprimer des « bundles ». L'état d'un « bundle » peut être changé à tout moment tandis que le cadre OSGi est en marche. Depuis OSGi RFC 132 [RFC132 2011], il est possible d'étendre la console ou le noyau de l'environnement OSGi avec de nouvelles commandes définies par des utilisateurs (user-defined commands). Pour cela, nous avons implémenté quatre commandes (load, activate, deactivate et unload) dans le composant qui interagit avec la console des commandes OSGi et qui invoque le service « PAP-Admin » du composant PAP contenant le code associé à chaque commande. Cette étape est intéressante pour contrôler le cycle de vie des politiques auto-contenues. Elle assure la cohérence du cycle de vie de politiques auto-contenues en gérant le cycle de vie des bundles OSGi de politiques auto-contenues. En conséquence, les composants ou « bundles » internes d'OSGi n'existent pas du point de vue des administrateurs de la politique.

D'autre part, différentes implémentations du cadre OSGi existent comme Equinox [Equinox 2011], Concierge [Concierge 2011], Knoplerfish [Knoplerfish 2011], ou Apache Felix [Apache Felix 2011]. Nous avons choisi Apache Felix pour notre système d'autorisation car il implémente la dernière

version de la spécification OSGi, y compris la RFC 132. En outre, il inclut un sous-projet appelé iPOJO [iPOJO 2011], qui propose un nouveau modèle flexible et extensible de composants orientés services se basant sur l'approche SOA [Escoffier 2008]. Le but derrière iPOJO est double. Le premier objectif est de simplifier le développement d'applications qui se basent sur OSGi. Le deuxième objectif est de pousser au-delà des mécanismes de bases d'OSGi et d'essayer d'incorporer de façon plus transparente des fonctionnalités avancées, y compris la configuration, la synchronisation, et la composition du composant.

L'implémentation de notre moteur de décision de politiques ou PDP est basée sur SUN XACML [SUNXACML 2011] qui supporte pleinement le standard XACML version 2.0 [XACML 2011] avec des types de données et fonctions standards. Nous développons un éditeur de politiques auto-contenues pour simplifier la spécification et la génération de politiques auto-contenues. Notre éditeur est basé sur un éditeur XML de Netbeans [Netbeans 2011] afin de fournir des fonctionnalités d'auto-complétions pour les politiques XACML qui sont des formats XML. En plus, notre éditeur est développé comme un « bundle » OSGi pour bénéficier des services des « bundles » OSGi de politiques auto-contenues.

7.6.2 Résultats expérimentaux

OSGi fournit la flexibilité et l'extensibilité que nous attendons dans notre approche. Afin de valider sa faisabilité, nous avons testé l'impact sur les performances en utilisant OSGi dans les moteurs de décision de la politique ou PDP. Le but de cette expérience n'est pas de faire une évaluation de la performance de notre prototype. Dans [Turkmen et Crispo 2008] une étude de comparaison de performance a été faite entre différentes implémentations de PDP XACML en fonction du nombre de règles de la politique, les requêtes XACML... Notre implémentation composants orientés services étant indépendante de l'implémentation du PDP, nous pouvons utiliser n'importe quelle autres implémentations du PDP par « bundlizing » c.à.d. faire des bundles « OSGi » pour ces implémentations.

Ainsi, nous avons étudié les deux prototypes suivants pour tester l'impact de l'utilisation de OSGi:

- Prototype1 : Ce prototype est basé sur l'implémentation SUNXACML [SUNXACML 2011], qui fournit une API Java pour ajouter de nouvelles fonctions, types de données et les PIPs. Nous appelons ce prototype « Classic XACML »
- Prototype2 : La seconde est notre implémentation OSGi de composants orientés services [Cheaito et al. 2010b] pour des politiques auto-contenues. Nous appelons ce prototype « OSGi XACML »

La validation de notre approche a été faite en considérant dans les deux cas que le PDP est basé sur SUNXACML et que le code qui implémente les fonctions de types de données non-standards et le PIP est le même. L'expérimentation consiste en trois tests (Figure 67) :

- **Test1 - type de données non standard :** La politique contient un type de données non-standard. Dans ce test, aucun PIP n'est nécessaire.
- **Test2 – appel à un PIP :** Dans ce cas, un attribut est manquant dans la requête XACML. Le PDP peut demander à un PIP l'obtention des valeurs d'attribut dans une base de données MySQL. Les types de données et les fonctions dans la politique sont standards.
- **Test3 - type de données non standard et appel à un PIP :** Dans ce cas, nous testons comment le PDP réagit avec une politique qui contient une expression contenant une fonction non-standard. En outre, l'expression contient une contrainte sur une valeur d'attribut

qui n'est pas dans le contexte de la requête XACML. Ainsi, le PIP devrait être appelé pour obtenir cette valeur à partir d'une base de données MySQL.

Pour chaque test, nous avons mesuré le temps moyen « average time » de prise de décision de contrôle d'accès en microsecondes (µs) basé sur 1000 requêtes envoyées par le PEP (Figure 67). Notre environnement de test est le suivant : Un ordinateur de bureau « HP compaq dc 7800 » qui a un processeur « Intel core2 duo E6550 2.33 Ghz », une mémoire RAM 2 GB, et un disque dur 150 GB. Le système d'exploitation est Windows 7. L'implémentation de composants orientés services est basée sur « Apache Felix version 3.0.1 » et sur « iPOJO version 1.6 ». La machine virtuelle java est « JAVA 1.6.0 SDK ». Aussi, le serveur MySQL est « Oracle MySQL server version 5.1 ».

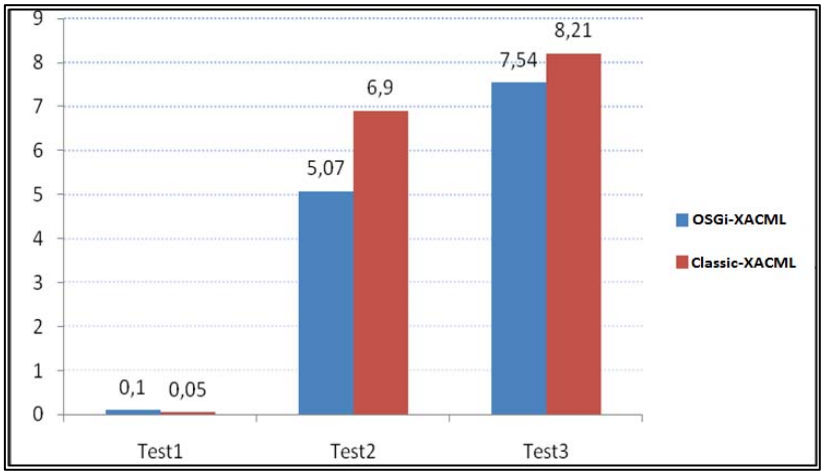

Figure 67 Temps nécessaire en µ-secondes pour chaque test

7.6.2.1 Analyse des résultats

Dans cette section, nous analysons les résultats de performance pour chaque test :

- **Test1:** « Classic-XACML » est un peu plus rapide que « OSGi XACML ». La raison est que dans « Classic-XACML » le module du type de données non-standard utilise directement l'API spécifique du PDP pour ajouter un nouveau type de données (ici c'est l'API SUNXACML). Ainsi, ce code est ajouté à l'intérieur de l'implémentation du PDP et celui-ci peut appeler directement les méthodes. « OSGi XACML », étant indépendant de l'implémentation du PDP, le PDP cherche le service fourni par le composant type de données non-standard qui implémente la fonction afin de l'invoquer. Il doit appeler le registre des services pour obtenir les références du service fournisseur. Actuellement, la communauté OSGi prend en compte cette question de performance. Des travaux existent [Frénot et al. 2010] [Rellermeyer et al. 2007] qui tentent d'améliorer l'interaction entre les services et composants dans l'environnement d'exécution OSGi. En conséquence, la performance de notre prototype pourrait être améliorée dans le futur.
- **Test2 et Test3:** « OSGi XACML » se place mieux que le prototype « Classic XACML ». Le prototype « Classic XACML » utilise « java.lang.ClassLoader » pour charger les classes requises par le pilote « JDBC MySQL » afin de se connecter en Java à la base de données MySQL. Cela prend beaucoup de temps par rapport au processus de chargement de la classe avec « java.lang.ClassLoader » dans OSGi. Cela vient du fait qu'avec la version java standard les chargeurs de classes comme « java.lang.ClassLoader » sont organisés dans une arborescence hiérarchique, alors qu'ils sont organisés sous forme d'un graphe dans OSGi. En conséquence, le chemin

de dépendance des chargeurs de classes dans OSGi est plus court que dans le standard Java et les performances sont meilleures. Pour plus d'informations sur la comparaison entre le chargement de classes dans Java standard et dans OSGi voir [Bartlett 2009].

7.6.3 Schéma global

La Figure 68 présente le schéma global de tous les services fournisseurs et services consommateurs de notre implémentation. Les services fournisseurs pour l'édition et l'exécution d'une politique-auto-contenue sont les suivants :

- Services fournisseurs pour la spécification d'une politique auto-contenue : « Get-config-GUI » qui construit l'interface graphique des extensions, « Get-auto-completion » qui est utilisé pour les fonctionnalités d'auto-complétion dans l'éditeur, et « Get-format » qui donne le format du type de données de l'attribut PIP requis par les composants PIP.
- Services fournisseurs pour le déploiement d'une politique auto-contenue : « Get-attribute » qui est responsable de récupérer les valeurs des attributs PIP, « Get-policy » qui est responsable de récupérer les fichiers de politiques XACML , « Evaluate-expression » qui est responsable de l'évaluation des expressions envoyées par le composant PDP ou les composants types de données, « Get-XACML-request » qui est responsable de la récupération de la requête XACML , « PAP-admin » qui est responsable de la gestion du cycle de vie des composants des politiques auto-contenues, et « Get-decision » qui prend les décisions d'autorisation prises par un composant PDP.

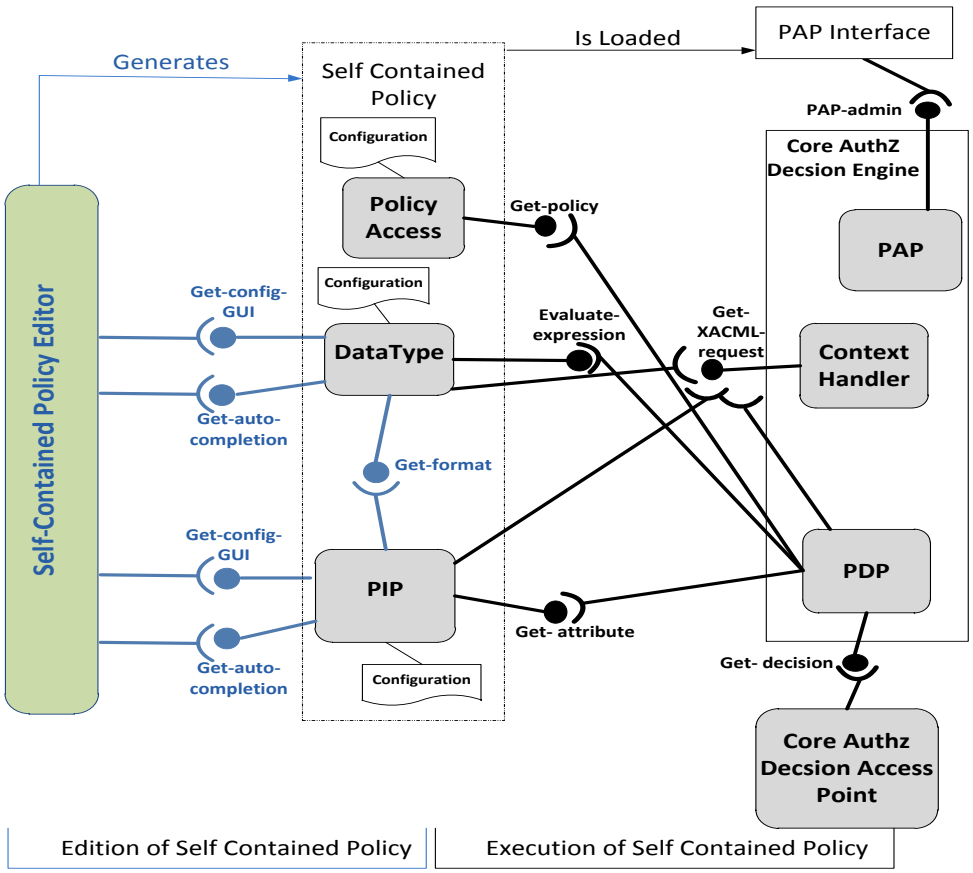

Figure 68 Schéma global

7.7 Conclusion

La contribution majeure de notre travail porte donc sur la définition d'un système d'autorisation générique et ouvert qui puisse s'adapter aux exigences de contrôle d'accès dépendant des organisations au présent et dans le futur. Notre travail repose sur deux approches de gestion, le contrôle d'accès à base d'attributs (ABAC) et la gestion à base de politiques (PBM). Un langage de politique basé sur ABAC donne la possibilité de spécifier des règles de contrôle d'accès de manière générique c.à.d. écrire n'importe quelles informations dans des règles. D'autre part, une architecture basée sur les PBM donne l'avantage d'avoir un système unique qui est ouvert et flexible pour répondre aux exigences. Comme nous l'avons indiqué, nous avons choisi pour notre travail le langage XACML alliant les avantages de ces deux approches. Mais, un langage de politiques XACML extensible, creuse un écart entre l'interprétation et l'exécution d'une politique contenant des extensions du langage par le moteur de décisions de politiques. Il n'y a aucune garantie qu'une politique contenant des extensions du langage puisse être interprétée et exécutée par un moteur de décision de politiques.

Pour cela, nous avons proposé le concept de « *politiques auto-contenues* ». Dans cette approche, une politique contient toutes les informations requises afin d'être interprétée et exécutée par un moteur de décisions d'autorisation. Nous avons présenté dans ce chapitre les caractéristiques auxquelles doit répondre une politique auto-contenue. Un cadre basé sur l'approche composants orientés services permet d'en implémenter le concept et de répondre aux critères d'extensibilité, de réutilisabilité, et d'adaptation dynamique aux besoins. Une évaluation expérimentale a été faite qui prouve la faisabilité de notre approche.

Enfin, nous avons également eu l'ambition de doter les responsables de l'écriture des politiques d'un système réduisant la complexité du passage d'une phase de conception à l'exécution d'une politique. Ainsi, nous avons proposé un composant qui est un éditeur de politiques auto-contenues. Cet éditeur conçu en environnement graphique, permet aux administrateurs de sécurité de réutiliser et configurer les composants de politiques auto-contenues existants (par exemple, le composant PIP MySQL).

Conclusion générale

L'objectif de cette thèse était de définir une méthodologie de conception et de développement d'un système d'autorisation adaptable aux différentes facettes que peut recouvrir le contrôle d'accès dans les organisations. Il prend en considération l'hétérogénéité des pratiques organisationnelles, le support de technologies différentes et des contextes à considérer. Deux approches de gestion ont guidé nos études - le contrôle d'accès à base d'attributs et la gestion à base de politiques - :

- Le contrôle d'accès à base d'attributs permet de spécifier des permissions par rapport à toute caractéristique liée à la sécurité des utilisateurs, des actions, des ressources et de l'environnement. Cette approche répond aux problèmes liés à l'expressivité des langages de politiques d'autorisation.

- La gestion à base de politique, quant à elle, vise à permettre l'adaptabilité dynamique du comportement d'un système par le biais de politiques de gestion. Cette architecture comporte deux entités : le Policy Decision Point (PDP) et le Policy Enforcement Point (PEP). Le PDP est une entité indépendante de l'élément géré, qui prend des décisions de gestion par rapport à une politique donnée. Le PEP est l'interface entre le PDP et l'élément géré. Lorsqu'une requête est lancée par un utilisateur sur une ressource, le PEP envoie une demande de décision au PDP puis l'applique lorsque le PDP s'est prononcé. Ce type d'architecture favorise l'intégration du système de gestion dans un environnement à gérer.

Au cours du travail, nous avons choisi le standard XACML comme technologie cible car il met en œuvre ces deux approches. Si d'un point de vue théorique XACML semble répondre aux problèmes d'adaptabilité, dans la pratique les systèmes construits selon ce standard sont limités à une situation donnée. En effet, lorsqu'un système XACML a été développé pour des besoins et un environnement technologique particuliers, il ne peut pas être facilement réutilisé dans une autre situation. Il est donc nécessaire de définir une méthodologie de conception et de développement pour rendre un tel système d'autorisation adaptable dans la pratique.

Notre approche a consisté à définir un système d'autorisation minimal (i.e. un noyau conforme au standard XACML) pouvant être facilement étendu pour des besoins spécifiques à une situation donnée (e.g. capacité d'expression et de mise en œuvre de politiques incluant des contraintes complexes, adaptation à un environnement technologique particulier, etc.). Les problèmes que nous avons traités sont :

1. Identifier les besoins principaux pouvant conduire à la construction d'une extension
2. Architecturer ces extensions afin qu'elles soient réutilisables par le système d'autorisation dans des situations différentes (différentes organisations, différents besoins d'autorisation, différents environnements informatiques)
3. Gérer le cycle de vie des extensions dans le système d'autorisation

Pour arriver à ces résultats, nous avons étudié tout d'abord l'évolution des exigences au sein des organisations, et plus particulièrement les exigences des politiques de sécurité. La définition d'une « politique de sécurité » est une phase clé dans le contrôle de la sécurité d'un système au sein d'une organisation. Une politique est difficile à construire, car elle se caractérise par la formulation d'une ou de plusieurs lignes directrices. Une politique de sécurité reste difficile à définir, à exprimer et à construire, car elle est établie par des acteurs ayant des niveaux de connaissance différents sur l'organisation, sur la stratégie des organisations, sur les environnements technologiques. D'une organisation à une autre les mots figurant dans une politique peuvent recouvrir des sens différents. Le domaine d'application d'une politique d'une organisation est très vaste comparé à celui d'un modèle formel de contrôle d'accès. De plus, la capacité d'interprétation d'une politique par des êtres humains dépasse pour l'instant largement celle que peut donner l'utilisation d'une entité informatique. Sa mise

en œuvre ne peut être faite qu'en regard des besoins de l'organisation où elle s'applique, tout en ayant un regard très attentif sur les technologies choisies et leurs évolutions qui peuvent l'impacter. En effet, la croissance et l'évolution permanente des organisations, de l'économie numérique mondiale, de l'étendue des applications d'Internet et des technologies de l'information et de la communication rendent encore plus complexe cette tâche. Chaque nouvelle vague de technologie apporte aussi avec elle de nouveaux risques de sécurité et de nouveaux défis de gestion de la sécurité. Cela a pour conséquence la nécessité de réviser et/ou de changer certains documents de politiques de sécurité dont le contenu a été arrêté à un instant donné. [Tech-Brief-CA 2011].

Aujourd'hui, l'une des retombées en matière de sécurisation de la constitution des politiques de sécurité est la mise en œuvre des systèmes de gestion des identités et des accès (IAM). Nous en avons rappelé les fondements principaux, et souligné les limites en regard de l'ambition attendue des solutions de contrôle d'accès. Dans l'absolu, le contrôle d'accès doit garantir l'accès à des services ou des ressources qu'elles soient matérielles ou logicielles. La politique de contrôle d'accès définit les circonstances dans lesquelles cet accès peut être octroyé. Nombre de paramètres influent sur l'autorisation ou le refus à un sujet d'exploiter un service ou une ressource. La complexité qui en découle a conduit en terme architectural à construire des moteurs d'autorisation afin que les circonstances d'exploitation d'une ressource puissent être évaluées au sein de ce module spécifique.

Plusieurs modèles formels de contrôle d'accès ont été proposés pour la gestion des accès. Ils se différencient par les caractéristiques retenues quant à la manière de structurer les concepts (comme sujets, objets, actions), l'expression de la politique d'autorisation et sa gestion. Un modèle formel décrit un système réel et par conséquent il définit son comportement. L'évolution des architectures de systèmes, puis la construction de véritables systèmes d'information, ont donné un nouvel élan à la modélisation des systèmes de gestion des accès. A ce jour, des critères aussi divers que les rôles, la localisation, les tâches, l'organisation, les contextes… donnent lieu à de nouveaux modèles formels de contrôle d'accès, pour répondre à des exigences de sécurité que l'on peut décrire au sein de politiques d'autorisation. En conséquence, nous avons proposé une synthèse des principaux modèles formels de contrôle d'accès proposés et de leur capacité à répondre au cas par cas à des exigences de sécurité au sein des organisations.

Ces deux études nous amènent à conclure simplement que si l'on veut réellement mettre en œuvre les politiques de contrôle d'accès, il faut impérativement construire des moteurs d'autorisation ouverts et qui répondent à la fois à des exigences propres aux organisations et aux exigences venant de l'évolution des technologies :

- En particulier, la portée (sémantique) des règles définies par l'administrateur d'une organisation mérite le plus souvent d'être précisée, adaptée au contexte d'une organisation. Il faut donc donner à un administrateur la possibilité de définir son cadre et les éléments propriétaires sur lesquelles doivent porter les règles.

- La technologie évoluant, des modèles sont entre autres proposés pour formaliser des situations particulières de contrôle d'accès. Conscient que les politiques peuvent évoluer il faut donc qu'un moteur d'autorisation soit suffisamment ouvert et flexible pour intégrer toute nouveauté qui pourrait affecter l'exploitation d'une ressource ou d'un service.

C'est dans cet état d'esprit que nous nous sommes appuyés sur les différents travaux dans le domaine du génie logiciel liés au concept d'adaptabilité des systèmes informatiques. Nous avons défini les besoins d'adaptation des systèmes d'autorisation. Ce travail nous a permis de déterminer les extensions à ajouter aux systèmes d'autorisation de type XACML. Nous avons aussi proposé une méthode permettant de faciliter la création des extensions. Elle explique comment étendre le langage de politiques dans notre système XACML minimal. Cela permet de construire des extensions pour prendre en compte les évolutions des exigences de sécurité au sein des organisations. Cette méthode propose également la manière de configurer ces extensions dans des modules réutilisables par notre système d'autorisation XACML.

Comme nous l'avons souligné, il existe un lien fort entre l'utilisation d'une extension et les besoins exprimés dans une politique d'autorisation. Par exemple, la présence dans le système d'autorisation d'une extension permettant de récupérer le rôle de l'utilisateur dans une base de données MySQL n'est nécessaire que si le système d'autorisation doit évaluer au moins une politique incluant des contraintes sur le rôle des utilisateurs. Dans le cas contraire, cette extension n'a pas lieu d'être ajoutée au système d'autorisation minimal. De la même manière, si une politique incluant des contraintes sur les rôles des utilisateurs est chargée dans le système d'autorisation, l'extension associée doit être ajoutée.

Ceci nous a naturellement conduit à étudier et proposer une solution en regard de la problématique du cycle de vie des extensions dans le système d'autorisation. Dans le concept de « politique auto-contenue » que nous avons défini, une politique auto-contenue comprend les règles d'autorisation ainsi que toute information nécessaire au système d'autorisation pour qu'il puisse l'interpréter et l'exécuter (i.e., les extensions et leurs configurations). Nous avons décrit une architecture pour spécifier et déployer des politiques auto-contenues en utilisant l'approche composants orientés services. Notre architecture est flexible et dynamiquement adaptable. Nous l'avons implémentée en utilisant le cadriciel OSGi afin de valider notre travail à la fois en termes de faisabilité mais aussi de performance. De plus, nous avons réalisé un éditeur de politique auto-contenue qui facilite l'écriture des règles d'autorisation ainsi que la configuration des extensions utilisées.

Au-delà des aspects conceptuels traités dans cette thèse, nous avons aujourd'hui un prototype.

- Il permet d'éditer et de déployer une politique auto-contenue : l'éditeur permet sous une forme conviviale de définir de nouveaux attributs non standards (types de données, intervalles de valeurs, fonctions, chemin d'accès (URI)...), de les configurer, ce qui permet à un administrateur de personnaliser ses règles de politiques et de générer un fichier prêt à être déployé et interprété.
- Le prototype offre une gestion dynamique du cycle de vie de la politique auto-contenue, afin d'éviter que des codes inutiles ne viennent alourdir les performances attendues d'un PDP.
- Il propose la réutilisabilité des composants de la politique auto-contenue, de telle sorte que dans des situations analogues il n'y ait pas à reconstruire tout un module d'extension.
- Le prototype offre des performances d'évaluation d'une politique XACML comportant des extensions acceptables. Les résultats expérimentaux ont permis de comparer les temps de prise de décision du moteur d'autorisation entre une politique définie sous forme auto-contenue (application directe de notre travail), et la même politique où les extensions ont été rajoutées sous forme statique à une implémentation du standard XACML.

Bien que cela n'ait pas été adressé dans cette thèse, il nous semblerait utile dans une continuité de ce travail, d'évaluer notre solution dans un contexte inter-organisationnel. Plusieurs problématiques pourraient donner lieu à des études. L'une d'entre elles est de pouvoir conjuguer des politiques auto-contenues de sécurité propres à chaque organisation, avec une politique de sécurité commune aux organisations naissant de la coopération engagée.

Des études complémentaires pourraient être menées pour identifier les conflits qui pourraient apparaitre à différents niveaux. En particulier nous avons souligné que dans une politique la sémantique d'une règle pouvait donner lieu à des interprétations différentes ; cette sémantique est décrite dans la politique auto-contenue. Il faudrait investiguer pour qu'au sein d'un éditeur de politique la dimension inter-organisationnelle puisse donner lieu à une vue commune. Il faudrait également que l'on puisse mettre en place au sein de ce même éditeur un moteur permettant de détecter les conflits qui pourraient apparaitre. Sur un plan architectural il faudrait également

déterminer s'il y a lieu de mettre en place un moteur de décision unique ? Centralisé mais où ? , ou bien distribué ce qui amènerait chaque organisation à évaluer les demandes d'accès aux ressources faites dans le cadre de la politique commune et de gérer des politiques locales à une organisation. Il faudrait également aborder la problématique d'accès à une ressource pour laquelle la décision d'autorisation implique un ou plusieurs partenaires. En complément, puisque nous avons envisagé un prototype permettant de traiter la réutilisabilité des extensions, il faudrait alors envisager de construire un cadriciel pour que des extensions conçues à un niveau local d'une organisation puissent être exportées dans un contexte commun. Cela pose en particulier le problème de nommage et de structuration au sein d'un cadriciel d'extensions. Il faudrait également traiter des aspects performances et de qualité des extensions dans un tel contexte.

Enfin, notre travail dans cette thèse s'est concentré uniquement sur l'étude des besoins d'adaptabilité de l'entité de prise des décisions d'autorisation (PDP : Policy Decision Point). Une solution plus globale pourrait également traiter de la problématique d'adaptabilité des entités de mise en œuvre des décisions d'autorisation : les PEPs (Policy Enforcement Point).

Acronymes

AAA: Authentification, Authorization, Accounting

ABAC: Attribute based access control

ACL: Access control list

ADF: Access control decision facility

Ad-OrBAC: Administration model for Or-BAC

AEF: Access control Enforcement facility

AOP: Aspect oriented programming

CA-RBAC: Context aware role based access control

CAS: Community authorization service

COCOA: Context-constrained Authorisation framework

CRBAC: Context role based access control

DAC: Discretionary access control

DAP: Directory access protocol

FTP: File transfer protocol

Geo-RBAC: Spatially aware role based access control

GeoXACML: Geospatial eXtensible access control markup language

GPS: Global positioning system

GSAM: Geo spatial authorization model

GSAS: Geo spatial authorization system

GTRBAC: Generalized temporal role based access control

HERASAF: Holistic enterprise-ready application security architecture framework

HTTP: Hypertext transfer protocol

IAM: Identity and Access management

Idp: Identity provider

IP: Internet Protocol

ITU: International telecommunication union

KAoS : Knowledge Agent oriented System

LDAP: Lightweight directory access protocol

LRBAC: Location-aware role-based access control

MAC: Mandatory access control

NAS: Network access server

Or-BAC: Organization based access control

OSGi: Open services gateway initiative framework

OV: Virtual organization

OWL: Web ontology language

PBM: Policy based management

PEP: Policy enforcement point

PDP: Policy decision point

PIP: Policy information point

PKI: Public key infrastructure

POLIPO: Policies and ontologies for interoperability, portability, and autonomy

POSET: Partially ordered set

RADIUS: Remote Authentication dial in user service

RBAC: Role based access control

RDF: Resource description framework

REI: Universal

RSBAC: Rule-set based access control framework for linux

SAML: Security assertion markup language

SELinux: Security enhanced linux

SOA: Service oriented architecture

SP: Service provider

SSL: Secure sockets layer

SSO: Single sign on

TBAC: Task based access control

TCSEC: Trusted computer system evaluation criteria

TIC: Technologies de l'Information et de la communication

TLS : Transport layer security

TMAC: Team based access control

VOMS: Virtual organization membership service

XACML: eXtensible access control markup language

XML: Extensible markup language

Références

[Abi Haidar et al. 2006] D. Abi Haidar, N. Cuppens-Boulahia, F. Cuppens et H. Debar, « An extended RBAC profile of XACML », proceedings of 2006 ACM Secure Web Services Workshop (SWS), Fairfax, VA, USA, Novembre 2006.

[Aboba et al. 2000] B. Aboba, P. Calhoun, S. Glass, « Criteria for Evaluating AAA Protocols for Network Access », RFC 2989, IETF, Nov. 2000.

[Abou El Kalam et al. 2003] A. Abou El Kalam, R. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel et G. Trouessin, « Organization Based Access Control », IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), Lake Come, Italy., 2003.

[Abou El Kalam et Deswarte 2006] A. Abou El Kalam, Y. Deswarte, « Multi-OrBAC: un modèle de contrôle d'accès pour les systèmes multi-organisationnels », 1st Conference on Security in Network Architectures and Information Systems (SAR-SSI 2006), pp. 67-85, juin 2006.

[Abrams et al. 1995] M. D. Abrams , L. J. La padula, , et S. Jajodia, « Information Security: An Integrated Collection of Essays », dans IEEE Computer Society Press, U.S., 1995.

[Agrawal et al. 2005] D. Agrawal, K.W. Lee, et J. Lobo, « Policy-based Management of Networked Computing Systems », Dans IEEE Communications Magazine. Vol. 43, n° 10, 2005.

[Akram 2005] M. Akram, « Managing Changes to Service Oriented Enterprises » , Master sotenu en 2005.

[Alfaro el al. 2006] J. G. Alfaro, F. Cuppens, et N. Cuppens-Boulahia, « Analysis of Policy Anomalies on Distributed Network Security Setups »,dans ESORICS 2006: 496-511, 2006.

[Alfieri et al. 2005] Alfieri R., R. Cecchini, V. Ciaschini, d. A. Luca, K. Frohner, L. Rentey et F. Spataro, « From gridmap-file to VOMS: Managing Authorization in a Grid Environment », Future Gener. Comput. Syst. 21(4): pp. 549-558, 2005.

[Apache Felix 2011] Apache felix projet, Site web: http://felix.apache.org, 2011.

[Arabo et al. 2009] A. Arabo, Q. Shi, et M. Merabti, « Context-Aware Identity Management in Pervasive Ad-hoc Environments », International Journal of Advanced Pervasive and Ubiquitous Computing 1 (4) 29-42, 2009.

[Atluri et Chun 2007] V. Atluri, et S. A. Chun, « A geotemporal role-based authorisation system », dans International Journal of Information and Computer Security, Vol. 1, No. 1/2, 2007.

[Autrel et al. 2008] F. Autrel, F. Cuppens, N. Cuppens-Boulahia, et C. Coma-Brebel, « MotOrBAC 2: a security policy tool. » , dans SARSSI'08 : 3ème conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information, 13-17 octobre, Loctudy, France, 2008.

[Balter et al., 1998] Balter, R., Bellissard, L., Boyer, F., Riveill, M., et Vion-Dury, J.-Y., « Architecturing and configuring distributed applications with Olan », dans Proceedings Int. Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), 1998.

[Bartlett 2009] N. Bartlett, « OSGi In Practice », 11, January 2009.

[Belaramani 2002] N. M. Belaramani, « A component-based software system with functionality adaptation for mobile computing ». Master's thesis, 2002.

[Bell et La Padula 1976] E. Bell, J. La Padula, « Secure computer systems: Unified Exposition and Multics interpretation », the MITRE Corporation, Technical report, 1976.

[Bertino et al. 2001] E. Bertino, P. A. Bonatti, et E. Ferrari, « TRBAC: A temporal role-based access control model », ACM Trans. Inf. Syst. Secur. 4(3): 191-233, 2001.

[Bertino et al. 2005] E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca, « GEO-RBAC: a spatially aware RBAC », dans SACMAT 2005.

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

[Blair and Coulson, 1997] Blair, G. S. et Coulson, G., « The case for reflective middleware », Technical report, Distributed Multimedia Research Group, Department of Computing, Lancaster University, 1997.

[Blanc 2006] M. Blanc, « Sécurité des systèmes d'exploitation répartis : architecture décentralisée de méta-politique pour l'administration du contrôle d'accès obligatoire », Thèse sotenu en 2006.

[Brief-CA 2007] Brève de CA sur solution IAM, « Comment la gestion des identités et des accès (IAM) peut-elle m'aider à améliorer la mise en conformité réglementaire et augmenter les performances de l'entreprise ? », Site web : http://www.ca.com/files/solutionbriefs/ca-iam_solution-brief_fr_200711.pdf, 20007.

[Cakic 2003] J. Cakic, « A high level framework for Policy-Based management of distributed systems », thése soutenu en 2003.

[Cervantes et Hall 2004] H. Cervantes, et R. S. Hall, « Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model », 26th ICSE, 2004.

[Chabrier 2007] T. Chabrier, "Access Control Mechanisms, " Master sotenu en 2007.

[Chadwick et al. 2008] D. Chadwick, G. Zhao, S. Otenko, R. Laborde, L. Su et T. A. Nguyen, « PERMIS: a modular authorization infrastructure ». Concurrency And Computation: Practice And Experience. Volume 20, Issue 11, Pages 1341-1357, 10 August 2008. Online ISSN: 1532-0634. Print ISSN: 1532-0626. DOI: 10.1002/cpe.1313.

[Chadwick et Fatema 2009] D. W. Chadwick, et K. Fatema, « An advanced policy based authorisation infrastructure », dans Digital Identity Management 2009: 81-84.

[Chadwick et Otenko 2003] D. Chadwick et S. Otenko, « A Comparison of the Akenti and PERMIS Authorization Infrastructures », Dans Proceedings of the ITI First International Conference on Information and Communications Technology (ICICT 2003) Cairo University, pages 5-26, 2003.

[Cheaito et al. 2009] M. Cheaito, R. Laborde, F. Barrère, et A. Benzekri, « An extensible XACML authorization decision engine for context aware applications », Dans : International Conference on Pervasive Computing and Applications (ICPCA 2009), Tamkang University - Taiwan, 03/12/2009-05/12/2009, IEEE, (support électronique), décembre 2009.

[Cheaito et al. 2010a] M. Cheaito, R. Laborde, F. Barrère, et A. Benzekri, « Configurable Data Types in Policy Based Access Control Management: A Specification and Enforcement Framework », Dans : Conférence sur la Sécurité des Architectures Réseaux et Systèmes d'Information (SAR-SSI 2010), Menton -France, 18/05/2010-21/05/2010.

[Cheaito et al. 2010b] M. Cheaito, R. Laborde, F. Barrère, et A. Benzekri, « A deployment framework for self-contained policies », Dans : IEEE/IFIP 6th International Conference on Network and Service Management (CNSM 2010), Niagara Falls -CANADA, 25/10/2010-29/10/2010.

[Chin et al. 2010] J. Chin, N. Zhang, A. Nenadic and O. Bamasak, « A Context-constrained Authorisation (CoCoA) Framework for Pervasive Grid Computing », the Wireless Networks

(WINET) Journal (the Journal of Mobile Communication, Computation and Information); ISSN: 1022-0038 (print version); ISSN: 1572-8196 (electronic version); Journal no. 11276; Springer US.

[Chong 2004] F. Chong, brève sur « gestion de l'identité et de l'accès », Site web : http://msdn.microsoft.com/fr-fr/library/aa480030.aspx, publié 2004.

[Clusif 2007] Fédération d'identité, « Gestion des Identités », dossier technique, groupe "Gestion des identités", juillet 2007.

[Cuppens et Miège 2003] F. Cuppens and A. Miège, « Modelling contexts in the Or-BAC model », ACSAC 2003.

[Cuppens et al. 2004] Cuppens F., N. Cuppens-Boulahia et A. Miège, « Inheritance hierarchies in the Or-BAC Model and application in a network environment », 2nd Foundations of Computer Security Workshop, 2004.

[Cuppens et al. 2006a] F. Cuppens, N. Cuppens-Boulahia, et C. Coma « O2O: Virtual Private Organizations to Manage Security Policy Interoperability », second International Conference on Information Systems Security (ICISS'06), Calcutta, Inde, décembre 2006.

[Cuppens et al. 2006b] F. Cuppens, N. Cuppens-Boulahia, et C. Coma-Brebel, « MotOrBAC : un outil d'administration et de simulation de politiques de sécurité ». First joint conference on security in network architectures (SAR) and security of information systems (SSI), 6-9 June, Seignosse, Landes, France, 2006.

[Cuppens-Boulahia 2008] N. Cuppens-Boulahia, F. Cuppens, D. Abi Haidar, et H. Debar, « Negotiation of Prohibition: An Approach Based on Policy Rewriting », dans 23rd International Information Security Conference (SEC 2008). Milan, Italy. September 2008.

[Cuppens et Miège 2004] F.Cuppens et A.Miège, « Administration Model for Or-BAC », dans International Journal of Computer Systems Science and Engineering (CSSE), 2004.

[Damianou et al. 2002] N. Damianou, N. Dulay, E. Lupu, M. Sloman et T. Tonouchi, « Policy Tools for Domain Based Distributed Systems Management », dans IFIP/IEEE Symposium on Network Operations and Management. Florence, Italy, Apr. 2002.

[David 2001] P. C. David, « Une infrastructure pour middleware adaptable », Rapport de Stage de DEA, < URL : http://rangiroa.essi.fr/rapports/2001/01-dea-david.pdf>, 2001.

[Desertot 2007] M. Desertot, « Une architecture flexible et adaptable pour les serveurs d'applications », Thèse sotenu en 2007.

[Dey 2001] A. K. Dey, « Understanding and Using Context. Personal Ubiquitous Computing », vol. 5, no. 1, pages 4–7, Springer-Verlag, London, 2001. 41.

[DIAMETER 2011] DIAMETER based protocol, Site web: http://tools.ietf.org/html/rfc3588#page-6, 2011.

[Eclipse 2011] Eclipse IDE Site Web: http://www.eclipse.org/, 2011.

[Escoffier 2008] C. Escoffier, " iPOJO : Un modèle à composant à service flexible pour les systèmes dynamiques," Thèse sotenu en 2008.

[Equinox 2011] Equinox projet, Site web: http://www.eclipse.org/equinox/, 2011.

[EXACML 2011] Enterprise Java XACML: http://code.google.com/p/enterprise-java-xacml/, 2011[Ferraiolo et al. 2007] D.F. Ferraiolo, R. Kuhn, R. Sandhu, « RBAC Standard Rationale: comments on a Critique of the ANSI Standard on Role Based Access Control, » Dans IEEE Security & Privacy 2007, vol. 5, no. 6, pp. 51-53, 2007.

[Ferraiolo et al. 2007] D.F. Ferraiolo, R. Kuhn, R. Sandhu, « RBAC Standard Rationale: comments on a Critique of the ANSI Standard on Role Based Access Control, » Dans IEEE Security & Privacy 2007, vol. 5, no. 6, pp. 51-53, 2007.

[Frénot et al. 2010] S. Frénot, N. Ibrahim, F. Le Mouël, A. Ben Hamida, J. Ponge, M. Chantrel, et D. Beras. « ROCS: a Remotely Provisioned OSGi Framework for Ambient Systems », Dans Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010), Osaka, Japan, April 2010.

[GeoXACML 2011] GeoXACML,url: http://www.opengeospatial.org/standards/geoxacml, last access 2011.

[Goodner et al. 2007] M. Goodner , M. Hondo, A. Nadalin, M. McIntosh, D. Schmidt, « Understanding WS-Federation », version 1.0, May 2007.

[GPX 2011] Official GPX site web: http://www.topografix.com/gpx.asp, 2011.

[Grau et al. 2008] B. C. Grau, I. Horrocks, , Y. Kazakov, et U. Sattler, « Modular reuse of ontologies: Theory and practice. » , Dans Journal of Artificial Intelligence Research 31 (Apr. 2008), 273—318.

[Grine et al. 2007] H. Grine, T. Delot, et S. Lecomte, «Evaluation de requêtes adaptable dans les environnements pervasifs : une approche à base de composants récursifs », Dans Actes de l'Atelier Gestion de Donnees dans les Systemes d'Information Pervasifs (GEDSIP), 2007.

[Hacini 2008] S. Hacini, « Sécurité des Systèmes d'Information : Mise en œuvre de la confiance et de l'adaptabilité pour la protection de l'agent mobile », thèse soutenue en 2008.

[Hansen et Oleshchuck 2003] F. Hansen et V. Oleshchuk, « SRBAC: A Spatial Role-Based Access Control Model for Mobile Systems, » In 8th NordSec 2003.

[Harrison et al. 1976] Michael A. Harrison, Walter L. Ruzzo et Jeffrey D. Ullman. « Protection in Operating Systems ».Communications of the ACM. 19(8):461–471, août 1976.

[Hengartner 2005] U. Hengartner, « Access control to information in Pervasive Computing Environments », Ph.D. Thesis, available as Technical Report CMU-CS-05-160, 2005.

[HERASAF 2011] HERASAF, site web: http://www.herasaf.org/, 2011.

[Herault 2005] C. Herault, « Adaptabilité des Services Techniques dans le Modèle à Composants», Thèse sotenu en 2005.

[Highsmith 1999] Highsmith, J.A. Adaptive Software Development – A Collaborative Approach to Managing Complex Systems, Dorset House Publishing, New York, 1999.

[Horrocks 2008] I. Horrocks, « Ontologies and the semantic web », Communications of the ACM, v.51 n.12, December 2008.

[Horridge 2009] M. Horridge, « A practical guide to building owl ontologies using protégé 4 and co-ode tools edition 1.2 », tutoriel 2009.

[Hui et al 2011] M. L. C. Hui, N. Elmarzouqi, et C. H. Yong, « A Context-Aware based Authorization System for Pervasive Grid Computing », Dans World Academy of Science, Engineering and Technology 74, 2011.

[iPOJO 2011] iPOJO projet, Site web: http://felix.apache.org/site/apache-felix-ipojo.html, 2011.

[ISO7498-2 1989] ISO 7498-2 Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture. ISO/IEC 1989.

[ISO 9000 2000] ISO 9000, « Systèmes de management de la qualité - Principes essentiels et vocabulaire », Ed Afnor,2000.

[ITU 2011] Site Web: International telecommunication union (ITU), Geneva, http://www.itu.int/fr/pages/default.aspx, 2011.

[JAC 2011] The Java Aspect Components (JAC) project, site web: http:// http://jac.ow2.org/, 2011.

[Jaume et Morisset 2008] M. Jaume, et C. Morisset, « Un cadre sémantique pour le contrôle d'accès, » Technique et Science Informatiques 27(8): 951-976, 2008.

[JENA 2011] Jena – A Semantic Web Framework for Java, Site web : http://jena.sourceforge.net/, 2011.

[Josang 2007] A. Josang, M. Al Zomai et S. Suriadi, « Usability and Privacy in Identity Management Architectures », Dans ACSW '07 proceedings of the fifth Australasian symposium on ACSW frontiers, 2007.

[Joshi et al. 2005] J.B.D. Joshi, E. Bertino, U. Latif, et A. Ghafoor, « A Generalized Temporal Role-Based Access Control Model », Dans IEEE Transactions on Knowledge and Engineering, VOL. 17, NO. 1, pp. 4-23, 2005.

[Kagal et al. 2004] L. Kagal, M. Paoucci, N. Srinivasan, G. Denker, T. Finin, et K. Sycaraet, « Authorization and Privacy for Semantic Web Services », Article, IEEE Intelligent Systems (Special Issue on Semantic Web Services), July 2004.

[Kamel 2008] M. Kamel, « Patrons organisationnels et techniques pour la sécurisation des Organisations virtuelles, » Thèse sotenu en 2008.

[Kazuki 2011] Kazuki java generator for ontologies, site web: http://semwebcentral.org/projects/kazuki/, 2011.

[Kerberos 2011] Kerberos, Site web: http://www.ietf.org/rfc/rfc4120.txt, 2011.

[Kew 2007] N. Kew, « The Apache Modules Book », Prentice Hall, 2007.

[KeyNote 2011] The KeyNote Trust-Management System Version 2, site web: http://www.crypto.com/papers/rfc2704.txt, 2011.

[Kingston 2008] J. Kingston, « Multi-perspective ontologies: Resolving common ontology development problems », An International Journal Expert Systems with Applications, Vol 34 , Issue 1, Pages: 541-550, 2008.

[KML 2011] KML site web: http://code.google.com/intl/fr/apis/kml/documentation/, 2011.

[Knoplerfish 2011] Knoplerfish projet, Site web: http://www.knopflerfish.org/, 2011.

[Kulkarni et Tripathi 2008] D. Kulkarni, et A.Tripathi, « Context-aware role-based access control in pervasive computing systems, » in SACMAT 2008.

[Laborde 2005] R. Laborde, « Un Cadre formel pour le raffinement de l'information de gestion de sécurité réseau : Expression et Analyse, » Thèse sotenu en 2005.

[Laborde et al. 2008] R. Laborde, M. Kamel, F. Barrère, et A. Benzekri, « Pep = point to enhance particularly », Dans IEEE POLICY Conference, 2008.

[Laborde et al. 2009] R. Laborde, M. Cheaito, F. Barrère, et A. Benzekri, « An extensible XACML authorization web service: Application to dynamic web sites access control », Dans: 3rd workshop Security and Privacy in Telecommunications and Information System (SEPTIS 2009), Marrakech, Morocco, November 29th - December 4th, 2009.

[Laborde et al. 2010] R. Laborde, M. Cheaito, F. Barrère, et A. Benzekri, « Vers un système d'autorisation comme un service », Dans : 9th edition of the Francophone Conference on management of Networks and Services (GRES 2010), Montréal-CANADA, 12/10/2010-14/10/2010.

[Lampson 1971] Lampson B., « Protection », 5th Princeton Symposium of Information Science and Systems, pp. 437-443, 1971.

[Legond-Aubry 2005] F. Legond-Aubry, « Un modèle d'assemblage de composants par Contrat et Programmation Orientée Aspect », thèse, soutenue le 11 juillet 2005.

[Le Monde 2011] Sony prévoit 2,3 milliards d'euros de pertes, Site web : http://www.lemonde.fr/technologies/article/2011/05/23/sony-prevoit-2-3-milliards-d-euros-de-pertes_1526020_651865.html, article publié le 23 mai 2011.

[Lieberherr 1995] K. Lieberherr, « Workshop on Adaptable and Adaptive Software Report », Addendum to the Proceedings of the 10th Annual Conference on Object-Oriented Programming Systems, Languages and Applications, Oct. 15-19, 1995, Austin, TX, USA.

[Lipiansky 1992] E. M. Lipiansky, « Identité et Communication », Presses universitaires de France - PUF, 1992, ISBN 978-2130442004.

[Lorch et Sankar 2004] M. Lorch, K. Sankar « Conceptual Grid Authorization Framework and Classification », Dans memo GFD-I.038 Authorization Frameworks and Mechanisms –WG, novembre 2004.

[Loscocco et al. 1998] P. A. Loscocco , S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell, « The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments, » In 21st NISSC 1998.

[Loques et al., 2000] Loques, O., Leite, J., Lobosco, M., et Sztajnberg, A., « On the integration of configuration and meta-level programming approaches », dans Object-Oriented Reflection and Software Engineering, volume 1826 of LNCS, pages 191-210. Springer-Verlag, 2000.

[Mankai 2005] M. Mankai, « Vérification et analyse des politiques de contrôle d'accès application au langage XACML », mémoire soutenuen 2005.

[Mann 2007] B. Mann, rapport de CA sur evolution d'IAM, « The Evolution of Identity and Access Management", 2007.

[Marangozova 1999] Marangozova, « Adaptabilité », Rapport de magister, < URL : http://rangiroa.essi.fr/rapports/1900/99-magistere3-marangozova.pdf>, 1999.

[Mckinley et al. 2004]P. K. Mckinley, S. M. Sadjadi, E. P. Kasten , B. H. C. Cheng, "A taxonomy of compositional adaptation", Technical Report MSU-CSE-04-17, 2004.

[Medjdoub 2005] S. Medjdoub, « Modèle de contrôle d'accès pour XML : Application à la protection des données personnelles, » Thèse sotenu en 2005.

[Mikkonen et al. 2000] Mikkonen, T., Lahde, E., Niemi, J., Siiskonen, M. « Managing Software Evolution with the Service Concept », Proceedings of the International Symposium on Principles of Software Evolution, Nov. 2000, Japan, IEEE Computer Press, pp. 46 – 50.

[MotOrBAC 2011] MotOrbac une implémentation du modèle Or-BAC, Site web : http://motorbac.sourceforge.net/index.php?page=home&lang=fr, 2011.

[Nasser 2006] B. Nasser, « Organisation Virtuelle : Gestion de politique de contrôle d'accès inter domaine, » Thèse sotenu en 2006.

[Netbeans 2011] Netbeans Module site web: http://www.netbeans.org/kb/trails/platform.html, 2011.

[NCSC 1987] National Computer Security Center (NCSC), « A Guide to Understanding discretionary Access Control in Trusted systems », 1987.

[NIST 2011] "Nationale Institute of standards and Technology (NIST)," Site Web: http://crs.nist.gov/groups/SNS/rbac/, 2011.

[Oliveira 2008] A. S. de Oliveira, « Réécriture et Modularité pour les Politiques de Sécurité, » Thèse sotenu en 2008.

[OrBAC 2011] Le modèle de contrôle d'accès Or-BAC Site web : http://orbac.org/, 2011.

[Oreizy et al. 1998] Oreizy, P., Medvidovic, N. and Taylor, R.N. « Architecture-Based Runtime Software Evolution », Proceedings of the International Conference on Software Engineering", Kyoto, Japan, April 1998, pp. 177 - 186.

[Oreizy et al. 1999] Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., et Wolf, A.L. « An Architecture-Based Approach to Self-Adaptive Software », IEEE Intelligent Systems, May/June 1999, pp. 54-62.

[OSGi 2011] OSGi, Site web: http://www.osgi.org/, 2010.

[OWL 2011] Web ontology langauage (OWL), Site Web: http://www.w3.org/TR/owl-features/, 2011.

[PAL 2011] The protégé axiom language: http://protege.stanford.edu/plugins/paltabs/pal-quickguide/, 2011.

[Papazoglou et al. 2007] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, « Service-Oriented Computing: State of the Art and Research Challenges », Computer, v.40 n.11, p.38-45, November 2007.

[Park et al. 2006] S. Park, Y. Han, et T. Chung, « Context-Role Based Access Control for Context-Aware Application, » in HPCC 2006.

[Patwardhan et al. 2004] A. Patwardhan V. Korolev, L. Kagal, et A. Joshi, « Enforcing Policies in Pervasive Environments », Dans Proceedings, International Conference on Mobile and Ubiquitous Systems: Networking and Services, August 2004.

[Pearlman et al. 2002] Pearlman L., V. Welch, I. Foster, C. Kesselman et S. Tuecke, « A community authorization service for group collaboration », In The IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, juin, 2002.

[PERMIS 2011] PERMIS Site Web: « http://sec.cs.kent.ac.uk/permis/ », 2011.

[Ponder 2011] Ponder2 Site Web: "http://www.ponder2.net/", 2011.

[Pressman 1997] Pressman, R.S. Software Engineering – A Practitioner's Approach, 4th Edition, McGraw-Hill Companies, Inc., New York, 1997.

[Protégé 2011] Protégé ontology editor, site web : http://protege.stanford.edu/, 2011.

[RADIUS 2011] RFC2865 Site Web: http://www.ietf.org/rfc/rfc2865.txt, 2000.

[RBAC_Profile 2005] « Core and hierarchical role based access control (RBAC) profile of XACML v2.0 », OASIS Standard, February 2005.URL: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbacprofile1-spec-os.pdf.

[RDF 2011] Resource Description Framework (RDF), Site web : http://www.w3.org/RDF/, 2011.

[Reed 2004] A. Reed, « The Definitive Guide to Identity Management », livre electronique publié par realtimepublishers, 2004.

[2011] Rei policy specification language, Site web : http://rei.umbc.edu/, 2011.

[Rellermeyer et al. 2007] J. S. Rellermeyer, G. Alonso, and T. Roscoe. « R-OSGi: Distributed Applications Through Software Modularization », Dans Proceedings of the ACM/IFIP/USENIX 8th International Middleware Conference, p. 1-20, November 2007.

[RFC132 2011] OSGi Drafts, RFC 132, Site web: http://www.osgi.org/Specifications/Drafts, 2011.

[RFC2904 2000] AAA (Authentication Authorization Accounting) Authorization Framework, « http://www.ietf.org/rfc/rfc2904.txt?number=2904 », 2000.

[RFC2906 2000] AAA (Authentication Authorization Accounting) Authorization Requirements, « http://www.ietf.org/rfc/rfc2906.txt?number=2906 », 2000.

[RFC5280 2008] RFC5280, « Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile », Site web: http://tools.ietf.org/html/rfc5280, 2008.

[RSBAC 2011] Rule set based access control for linux (RSBAC), Site Web: http://www.rsbac.org/, 2011.

[Russello 2007] G. Russello, C. Dong, and N. Dulay, « Authorisation and Conflict Resolution for Hierarchical Domains », dansIEEE Workshop on Policies for Distributed Systems and Networks, 2007.

[SAML 2011] OASIS Security Assertion Markup Language (SAML), Site Web: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security,2011.

[Sandhu et Samarati 1994] R. S. Sandhu , et P. Samarati, « Access control: principles and practice ». Dans IEEE Communications Magazine, 1994.

[Scaglioso et al. 2008] P. G. Scaglioso, C. Basile, A. Lioy, « Modern Standard-based Access Control in Network Services: XACML in action », dans IJCSNS, VOL.8 No.12, Décembre 2008.

[SEI 2011] Software Engineering Institute's (SEI) website: http:\\www.sei.cmu.edu, 2011.

[SELinux 2011] Security Enhanced Linux, Site web: http://www.nsa.gov/research/selinux/, 2011.

[Shibboleth 2011] Shibboleth Site Web: http://shibboleth.internet2.edu/, 2011.

[Sirin et al. 2007] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, et Y. Katz, « Pellet: A practical owl-dl reasoned », Elsevier Journal of Web Semantics: Science, Services and Agents on the World Wide Web, 5:51–53, 2007.

[Sloman et Lupu 2002] M. Sloman, et E. Lupu, « Security and Management Policy Specification », IEEE Network: Volume 16: Issue 2: pp.10–19: March 2002.

[Smalltalk 2011] Smalltalk an object-oriented, dynamically typed, reflective programming language, Site web: http://www.smalltalk.org/versions/ANSIStandardSmalltalk.html, 2011.

[Somashekar. 2010] S. Somashekar, «Le cloud computing : quelles opportunités pour votre organisation ? ». Dans livre blanc strategie produits, janvier 2010.

[SpringAOP 2011] Aspect Oriented Programming with Spring (SpringAOP), site web: http://static.springsource.org/spring/docs/2.0.8/reference/aop.html, 2011.

[Stepien et al. 2010] B. Stepien, S. Matwin, et A. Felty, « Strategies for Reducing Risks of Inconsistencies in Access Control Policies », Dans ARES 2010.

[Subramanian et Chung 2001] N. Subramanian, et L. Chung, « Software Architecture Adaptability: An NFR Approach, » Dans IEEE Proceedings of International Workshop on Principles of Software Evolution, 2001.

[Subramanian et Chung 1999] N. Subramanian, et L. Chung, « Metrics for software adaptability, » Dans Applied Technology Division, Anritsu Company, pp. 95-108, 1999.

[SUNXACML 2011] SUN XACML PDP implémentation, Site web : http://sunxacml.sourceforge.net/, 2011.

[SWRL 2011] Semantic web rule language (SWRL), site web : http://www.w3.org/Submission/SWRL/, 2011.

[Tarvainen 2008] P. Tarvainen, « Adaptability Evaluation at Software Architecture Level", Dans The Open Software Engineering Journal, 2008.

[Tavares et Valente 2008] A. L.C. Tavares , et M. T. Valente, « A gentle introduction to OSGi », Dans ACM SIGSOFT Software Engineering Notes, v.33 n.5, September 2008.

[Tech-Brief-CA 2011] Technologie brève de CA, « An Integrated Architecture for Identity and Access Management »,
Site web : http://www.pinnacle-international.com/pdf/iam_tech_brief.pdf, téléchargé le 21 mars 2011.

[Tekinerdogan et Aksit 1996] B. Tekinerdogan et M. Aksit, « Adaptability in Object-Oriented Software Development Workshop Report », 10th European Conference on Object-Oriented Programming, July 8-12, 1996, Linz, Austria.

[Thomas et Sandhu 1993] R. K. Thomas et R. Sandhu, « Towards a task-based paradigm for flexible and adaptable access control in distributed applications », 2nd New Security Paradigm Workshop, 1993.

[Thomas et Sandhu 1997] R. K. Thomas et R. S. Sandhu. Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Authorization Management. In Tsau Young Lin & Shelly Qian, editeurs, IFIP'98 : 11th International Conference on Database Security, Lake Tahoe, California, volume 113 of IFIP Conference Proceedings, pages 166–181. Chapman & Hall, 1997. 27, 32, 34.

[Thion 2008] R. Thion, « Structuration Relationnelle des politiques de contôrle d'accès Resprésentation, Raisonnement et Vérification, » Thèse sotenu en 2008.

[Toelen 2005] J. O. Toelen, « Identity and Access Management », Master.thesis, 2008.

[Trivellato et al. 2009] D. Trivellato, F. Spiessens, N. Zannone, et S. Etalle, « POLIPO: Policies & OntoLogies for Interoperability, Portability, and autOnomy », in IEEE POLICY 2009.

[Tsai et al. 2007] W. T. Tsai, X. Zhou, Y. Chen, B. Xiao, R. A. Paul, et W. Chu, « Roadmap to a Full Service Broker in Service-Oriented Architecture », Dans ICEBE 2007: 657-660.

[Turkmen et Crispo 2008] F. Turkmen, et B. Crispo, « Performance evaluation of XACML PDP implementations », Dans SWS 2008 p. 37-44.

[Twidle et al. 2009] K. Twidle, N. Dulay, E. Lupu et M. Sloman, « Ponder2: A Policy System for Autonomous Pervasive Environments », Dans The Fifth International Conference on Autonomic and Autonomous Systems ICAS, April 2009.

[UMU 2011] UMU editor: http://umu-xacmleditor.sourceforge.net/, 2011.

[Uszok et al. 2004] A. Uszok, J. M. Bradshaw, M. Johnson, R. Jeffers, A. Tate, J. Dalton, et S. Aitken. « KAoS Policy Management for Semantic Web Services », Dans IEEE Intelligent Systems, 19(4):32–41, 2004.

[Uszok et al. 2008] A. Uszok, J. M. Bradshaw, J. Lott, M. Breedy, L. Bunch, P. Feltovich, M. Johnson, et H. Jung, « New Developments in Ontology-Based Policy Management: Increasing the Practicality and Comprehensiveness of KAoS », Dans Proceedings of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks, Washington DC, USA 2008.

[Vacca 2009] J. Vacca, « Computer and Information Security », ISBN : 978-0-12-374354-1, 2009.

[Vimercati et al. 2003] S. D. C. d. Vimercati, S. Paraboschi, et P. Samarati, « Access control: principles and solutions ». Dans Journal Software Practice Experience, pp. 397-421, avril 2003.

[Vimercati et al. 2007] S. D. C. d. Vimercati, S. Foresti, et P. Samarati, «Access control policies and languages ». Dans International Journal of Computational Science and Engineering, Vol. 3,No. 2, 2007.

[Wang et al. 2004] L. Wang, D. Wijesekera, et S. Jajodia, « A Logic-based Framework for Attribute based Access Control », Dans 2nd ACM Workshop on FMSE, 2004.

[Wazan 2011] A. S. Wazan, « Gestion de la confiance dans les infrastructures à clés publiques », thèse, 2011.

[XACML 2011] OASIS eXtensible Access Control Markup Language (XACML) Site web: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, 2011.

[XACMLLight 2011] XACMLight, site web: http://sourceforge.net/projects/xacmllight/, 2011.

[XMLPAD 2011] XMLPAD, Site web: http://www.wmhelp.com/xmlpad3.htm, 2011.

[Zhang et al. 2006] H. Zhang, Y. He, et Z. Shi. « Spatial context in role-based access control», Dans ICISC Conference, 2006.

[Zhou et al 2005] W. Zhou, V. H. Raja, et C. Meinel, « An authentication and authorization system for virtual organizations », Dans WMSCI 2005: 9th World Multi-Conference on Systemics, Cybernetics and Informatics, Vol 7, pp. 150-155, 2005.