Université du Québec en Outaouais

Spécification des exigences de contrôle d'accès en modèles de systèmes de décision

Thèse présentée comme exigence du programme de Doctorat en sciences et technologies de l'information

Université du Québec en Outaouais

Département d'informatique et d'ingénierie

Thèse intitulée :

Spécification des exigences de contrôle d'accès en modèles de systèmes de décision

Un travail dirigé par

Karim El Guemhioui, directeur de recherche

Luigi Logrippo, codirecteur de recherche

Présenté par

Jamal Abd-Ali

Novembre 2016

Table des matières

Liste des abréviations et des acronymes

ACMM : Access control metamodel (41)

ADA : arbre de décisions ascendantes (102)

BLP : Bell LaPadula (59)

CCA : concept de contrôle d'accès (adopté dans le modèle du langage MACL) (86)

CLP : Constraint Logic Programming (14)

CW : Chinese Wall (54)

ComAl : Combining Algorithm (11)

ComLang : langage proposé pour la spécification des algorithmes de composition de systèmes de décision (78)

DCW : Dynamic Chinese Wall (55)

ECAE : exigence de contrôle d'accès en entreprise (2)

EPAL : Enterprise Privacy Authorization Language (3)

G2G : modèle de contrôle d'accès dit Group2Group (46)

IDL : Interface Definition Language normalisé par OMG (42)

IM : Integration Metamodel (86)

MACL : Modular Access Control Language (6)

MOF : Meta Object Facility (40)

NHS: niveau d'habilitation de sécurité (57)

NS: niveau de sensibilité (57)

OCL : Object Constraint Language (27)

OMG : le consortium Object Management Group (27)

PDL : Policy Description Language (33)

PDP : Policy Decision Point (entité du système qui évalue une politique applicable et retourne une décision d'autorisation) (18)

PEP : Policy Enforcement Point (entité du système qui effectue le contrôle d'accès en faisant les requêtes de décision et en forçant les décisions d'autorisation) (18)

PIP : Policy Information Point (entité du système qui agit en tant que source de valeurs des attributs) (18)

RBAC : Role Based Access Control (23)

SAML : Security Assertion Markup Language (25)

SCA: Système de contrôle d'accès (25)

SOD: Separation Of Duty (7)

SMT : Satisfiability Modulo Theory (87)

V&V : Validation et vérification (81)

XACML : eXtensible Access Control Markup Language (3)

Résumé

L'étendue des activités humaines et des processus critiques qui sont sujets à des traitements automatiques, généralement distribués, font que les exigences de contrôle d'accès aux ressources ne cessent de gagner en importance et d'atteindre des niveaux de complexité inégalés. Le nombre des usagers et des ressources a explosé avec l'avènement des applications distribuées sur des réseaux étendus sur les cinq continents. En conséquence, la spécification des exigences de contrôle d'accès est devenue une tâche laborieuse et complexe qui se base sur différents critères qui ne sont pas tous simples d'où le risque élevé d'avoir des énoncés de spécifications de contrôle d'accès contradictoires et incomplets. On se trouve alors face à la problématique de vouloir forcer dans un moule de règles élémentaires de contrôle d'accès (sujet, mode d'accès, objet), des exigences de contrôle d'accès exprimant généralement des principes généraux et des buts de contrôle d'accès des entreprises.

En vue de rendre la spécification des exigences de contrôle d'accès plus simple et moins exposée aux erreurs, nous avons élaboré un langage permettant la spécification de ces exigences en termes de principes de contrôle d'accès et fournissant des moyens de vérification de propriétés et de détection d'erreurs.

Notre approche se base sur l'identification des concepts de contrôle d'accès au niveau des buts de contrôle d'accès des entreprises. Un but de contrôle d'accès exprime un concept de contrôle d'accès qui justifie la raison pour laquelle on veut limiter l'accès à certaines ressources. Comme exemples de concepts de contrôle d'accès on peut considérer: l'application du principe d'octroyer le minimum nécessaire de permissions d'accès; ou la limitation de l'ensemble des ressources accessibles par tout sujet à un niveau qui l'empêche d'accomplir, sans l'aide d'un autre sujet, une tâche critique. Des éléments de notre langage représentant ces concepts vont permettre une expressivité au niveau de ces concepts. Ceci étant, notre vision d'une spécification de contrôle d'accès dans une entreprise consiste en une structure de plusieurs concepts de contrôle d'accès

qui sont couplés en modules dont chacun est capable d'émettre une décision de contrôle d'accès en réponse à toute demande d'accès. Cette modularité permet une meilleure séparation des problèmes et une réduction de la complexité des spécifications. Elle va caractériser le langage qu'on propose et lui donner son nom, à savoir, Modular Access Control Language (MACL).

De plus, nous définissons un mapping de MACL vers un langage cible qui lui permet un cadre formel de vérification de propriétés des spécifications des exigences de contrôle d'accès écrites dans ce langage. Ce cadre formel fournira les moyens d'automatisation des tâches de vérification de propriétés et de détection d'erreurs.

Chapitre 1 **Introduction**

Dans ce chapitre on va identifier l'insatisfaction qu'on ressent vis-à-vis des moyens de spécification des politiques de contrôle d'accès et on va donner notre point de vue sur des pistes de recherche pour améliorer cette situation.

Le *contrôle d'accès* concerne la limitation des activités des usagers légitimes d'un système qui ont déjà été authentifiés avec succès. Le contrôle d'accès a été historiquement implémenté par un ensemble de règles au niveau des systèmes d'exploitation, des pare-feu et des systèmes de gestion de bases de données.

La spécification de contrôle d'accès doit permettre de traiter une demande d'accès à une ressource informatique par une certaine entité qu'on appelle sujet. La forme la plus simple de cette spécification est un ensemble de *règles*, chacune permettant de fournir une réponse de contrôle d'accès à certaines demandes d'accès; on dit alors que la règle est applicable à ces demandes d'accès. Par exemple, spécifier qu'une certaine personne est autorisée à lire un certain fichier requiert une règle de spécification de contrôle d'accès qui relie cette personne au dit fichier, avec un privilège d'accès en mode lecture. Ce modèle de règles de spécification de contrôle d'accès est fonctionnel lorsque les exigences de contrôle d'accès peuvent être exprimées en termes de contraintes simples sur les attributs courants des sujets et des ressources impliqués, ce qui n'est généralement pas le cas des exigences de contrôle d'accès en entreprise.

Un langage de spécification de contrôle d'accès est un langage qui permet d'exprimer des exigences de contrôle d'accès dans un énoncé qui servira de base pour produire des décisions de contrôle d'accès à des demandes d'accès. En général, ce type de langage utilise uniquement une forme de règles simples de contrôle d'accès pour exprimer des exigences de contrôle d'accès. De plus, ces règles peuvent être regroupées selon différents critères, tels que les motifs qui les sous-tendent, les conditions d'applicabilité ou les parties qui les ont exigées. Les ensembles constitués par ces regroupements de règles définissent ce qu'on appelle des *politiques* de contrôle d'accès.

Un bon nombre de langages d'expression de politiques de contrôle d'accès est déjà disponible. Malgré cela, il n'y a pas de langage qui permet toujours d'exprimer des exigences de contrôle d'accès avec une expressivité satisfaisante où une exigence correspond à une formulation simple et bien définie dans le langage. Notons que les exigences d'une seule politique de contrôle d'accès peuvent correspondre à un grand nombre de règles pour l'exprimer, ce qui augmente les risques d'erreurs, d'incohérence et de conflit avec d'autres exigences ou règles.

Dans ce travail, nous visons l'élaboration d'un langage qui dépasse les limitations d'expression des langages existants en offrant un moyen simple d'exprimer de façon appropriée des exigences de politiques de contrôle d'accès en entreprise. Ces exigences seront désignées par *exigences de politique de contrôle d'accès en entreprise* (ECAE). Elles sont définies à une échelle globale comme des principes à respecter qui reposent sur des concepts dont l'interprétation ne se limite pas à un seul sujet et un seul objet dans un contexte déterminé. Par exemple, un principe de contrôle d'accès à respecter au sein d'une entreprise peut consister en l'exigence que l'accès à tout dossier de client de l'entreprise soit limité à l'employé qui est en charge du client et qui lui offre les services de l'entreprise. Nous pouvons ainsi définir le concept de « chargé de » qui exprime qu'un sujet est chargé d'un ensemble d'objets et permet de définir, une fois pour toutes, les privilèges d'un sujet sur les objets qu'il prend en charge.

Le langage visé doit offrir un moyen d'exprimer de façon simple des exigences de contrôle d'accès semblables à celle de l'exemple précédent, et ce en utilisant des concepts prédéfinis dans ce langage comme celui de « chargé de ». En effet, en l'absence de ce genre de concept, de telles ECAE basées sur « qui se charge de quoi » ne peuvent facilement être exprimées sous forme de règles reliant un sujet, un objet, une condition et un privilège d'accès.

Pour spécifier une (des) exigence(s) de contrôle d'accès en règles de contrôle d'accès, on dispose déjà de plusieurs langages : certains sont des standards comme XACML, d'autres constituent des solutions propriétaires comme EPAL [1] de IBM; ces langages seront discutés dans le chapitre 2. Le langage qu'on va élaborer sera doté d'un pouvoir expressif adéquat aux ECAE qui le distingue des langages de spécification de contrôle d'accès existants par son degré d'expressivité et ses moyens de vérification de propriétés.

1.1 Règles et modèles de contrôle d'accès

Notre investigation des besoins des entreprises en contrôle d'accès, nous a permis de constater qu'une exigence de contrôle d'accès en entreprise n'est pas toujours exprimable sous la forme d'une règle simple de contrôle d'accès reliant un sujet à certains objets par le biais d'un privilège d'accès. Dans cette section, nous allons expliquer le besoin de disposer de moyens qui permettent une expressivité plus adéquate que les règles simples.

Nous verrons également ce qu'est un modèle de contrôle d'accès. À cet effet, nous commencerons par une explication préliminaire introduisant de façon *abstraite* la notion de modèle de contrôle d'accès. Suite à une analyse rapide des ECAE, nous avons remarqué que des règles simples de contrôle d'accès étaient insuffisantes car, pour certaines ECAE, l'octroi d'un privilège dépend d'un ensemble de conditions qui ne sont pas assez simples pour être exprimées dans une seule règle, mais nécessitent plutôt une spécification laborieuse d'un ensemble de règles. De plus, certaines ECAE se basent sur

[1] http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/

des ensembles de conditions qui présentent une similarité remarquable; cette similarité est due au fait que tous ces ensembles de conditions sont exprimables en fonction d'une unique structure de concepts. Cette structure de concepts, partagée par ces ECAE, sera dite un *modèle* de contrôle d'accès de ces ECAE. L'utilité d'un modèle de contrôle d'accès réside dans son potentiel comme moyen d'expression simple de ces ECAE. Nous allons chercher à spécifier une ECAE par : (1) une spécification d'un modèle de contrôle d'accès; (2) une spécification des particularités qui distinguent l'ECAE considérée des autres ECAE qui partagent le même modèle. Ces ECAE exprimables en se basant sur un modèle de contrôle d'accès représentent des applications de ce modèle et elles sont dites des *instances* de ce modèle de contrôle d'accès.

Ainsi, la spécification d'un modèle de contrôle d'accès est fournie une fois pour toutes, et chaque application de ce modèle sera une réutilisation de sa spécification se distinguant par les particularités spécifiques aux ECAE considérées. Cette intégration de ces particularités est dite une « personnalisation » ou « customization ».

1.1.1 Capturer la logique des ECAE dans des modèles

Un modèle de contrôle d'accès, formé d'un ensemble de concepts, représente un principe de contrôle d'accès. Une caractéristique principale d'un modèle de contrôle d'accès est que son application est une spécification de contrôle d'accès permettant de fournir des décisions en réponse à des demandes d'accès.

On définit un *système de décision* comme étant une spécification de contrôle d'accès qui, interprétée individuellement, permet des décisions de contrôle d'accès en réponse à des demandes d'accès. Ainsi dans le cadre du présent travail de recherche, une application d'un modèle de contrôle d'accès est un système de décision. De plus, un principe ou un modèle dont l'application n'est pas un système de décision n'est pas désigné par le terme « modèle de contrôle d'accès ».

Ceci étant, nous sommes d'opinion qu'un seul modèle de contrôle d'accès représentant un principe de contrôle d'accès, n'est pas suffisant pour exprimer de manière claire et concise tous les besoins de contrôle d'accès des entreprises.

Dans la suite de cette section, ce point sera expliqué en utilisant comme exemple le modèle de rôle.

Contrôler l'accès aux ressources informatiques d'une entreprise est souvent étroitement lié à la spécification des mandats du personnel de l'entreprise. Si ce contrôle n'est pas bien conçu, ce contrôle pourrait limiter l'accès à des informations indispensables à l'accomplissement des activités de quelques membres du personnel de l'entreprise. Ainsi, l'administrateur du contrôle d'accès doit jongler avec un bon nombre de sujets et de ressources en considérant la question de qui fait quoi dans l'entreprise.

En effet, le besoin de spécifier, avec des groupements de sujets et d'objets, les exigences de contrôle d'accès de l'entreprise était derrière l'émergence du bien connu concept de rôle. Ce concept se base sur la catégorisation des ressources et des sujets demandant un accès. Ce concept a été défini dans [1] et [2], et il est largement adopté car il facilite la gestion du contrôle d'accès.

Prenons, par exemple, un système de protection des données basé sur les rôles. Si on désire exprimer un tel système en utilisant un langage de règles simples de contrôle d'accès, il faut commencer par bâtir une sémantique incluant le concept de rôle. Il serait évidemment plus facile d'utiliser des éléments d'expression dans lesquelles cette sémantique est prédéfinie. Pareillement, on pourrait penser à plusieurs autres concepts utilisés par d'autres modèles, qui pourraient aussi être inclus dans la sémantique du langage que nous envisageons. Entre autres, on pourrait penser aux modèles basés sur la confidentialité comme celui de Bell-LaPadula [3].

Cette situation non satisfaisante lors de l'expression d'ECAE avec des langages à règles simples ou limités à un seul modèle de contrôle d'accès, nous a donné l'idée d'élaborer un langage qui sera plus approprié en permettant d'exprimer des ECAE en termes d'instances de modèles de contrôle d'accès connus et dont la logique de décision est déjà intégrée dans la sémantique des éléments de ce langage. Ainsi, un modèle de contrôle d'accès et sa logique de décision sont spécifiés une fois pour toutes, ensuite ce modèle sera réutilisé et sa logique sera réutilisée aussi chaque fois qu'on applique ce modèle. Cet

aspect du langage sera désigné par *la réutilisation de la logique de décision*. En plus d'avoir une meilleure expressivité, nous projetons que notre langage, qu'on appellera MACL, sera muni de moyens de résolution d'éventuelles contradictions entre spécifications provenant de différents modèles de contrôle d'accès, ainsi que d'autres moyens de vérification de propriétés.

1.1.2 Règles et modèles de contrôle d'accès

Dans cette section, nous allons investiguer deux manières différentes de spécifier les ECAE : (1) en utilisant des règles simples, et (2) en utilisant des modèles de contrôle d'accès comme base de notre langage de spécification d'ECAE. Nous allons voir que la spécification des ECAE avec des règles simples de contrôle d'accès augmente les possibilités d'erreurs et rend difficile l'identification des motifs sous-tendant ces règles.

Pour commencer, expliquons la structure des règles de contrôle d'accès ayant une forme simple. Les règles expriment des assertions simples comme : suite à sa demande, un certain sujet peut accéder en mode lecture à une certaine ressource si une certaine condition est satisfaite. En effet, une règle s'applique pour traiter un **événement** de demande d'accès et consiste en un n-uplet exprimant une association multiple entre : **sujet, ressource, action, condition, décision** d'autorisation. Un **événement** représente une demande d'accès à une ressource et permet de fournir des informations provenant de l'environnement d'exécution comme, par exemple, la date et l'heure. Un **sujet** représente l'entité demandant l'accès. Une **ressource** représente l'information ciblée par la demande d'accès. Une **condition** concerne un ensemble de valeurs d'attributs de la ressource et certains attributs de l'environnement. Une **action** spécifie l'opération que l'événement demande. Une **décision** est la réponse à l'événement demandant l'exécution de l'action sur la ressource. Une décision prend généralement l'une des valeurs : « Permit », « Deny », « Indeterminate », « NotApplicable ».

Un ensemble de règles du type mentionné est appelé une *politique* et représente, généralement, un ensemble d'exigences reliées qui permettent de caractériser cette

politique par une description sommaire informelle basée sur un but de l'entreprise; ce but assure une certaine cohésion aux règles d'une même politique.

Considérons maintenant les caractéristiques des exigences de contrôle d'accès d'une entreprise. Une ECAE est généralement exprimée en termes de buts de l'entreprise, de règles de discipline générales, d'éléments organisationnels de l'entreprise et de normes spécifiant le respect de la confidentialité des informations. Une ECAE représente un ou plusieurs critères de contrôle d'accès qui peuvent toucher un ensemble de ressources et un ensemble de sujets, en tenant compte d'un ensemble de concepts pas nécessairement simples.

Comme exemple simple d'ECAE, citons l'énoncé suivant : en aucun cas une même personne ne peut accéder à tous les objets d'un certain ensemble d'objets. Ce principe de contrôle d'accès s'inscrit dans un modèle de contrôle d'accès connu sous le nom de *Separation of Duty (SOD) ou séparation des tâches*. Cette ECAE concerne généralement un ensemble d'objets ayant des effets complémentaires nécessaires pour accomplir une tâche critique voire dangereuse. Cette unique ECAE inclut une révision de l'historique des accès aux ressources et aura une éventuelle incidence sur toute autre ECAE car elle concerne tous les sujets et plusieurs objets.

Essayons d'exprimer l'exemple précédent en règles simples selon la forme du n-uplet déjà vu (sujet, ressource, action, condition, décision), en faisant abstraction de l'événement qui est une demande d'accès.

Notons par OP l'ensemble des objets concernés par notre exemple; on veut spécifier qu'aucun sujet n'a le droit d'accéder à la totalité des objets de OP, et ce indépendamment de l'ordre d'accès adopté. Admettons que l'action considérée soit une lecture et désignons par HasAccessed(s) l'ensemble des objets de OP déjà accédés (selon un mode d'accès quelconque) par un sujet s. Les règles pour exprimer l'exemple précédent pourront être :

Règles à formuler indépendamment des autres spécifications de contrôle d'accès

Sujet : s; Ressource : o; Action : Lecture; Condition : o $\in$ OP et OP = (HasAccessed(s) U {o}); Décision : « Deny ».

Il faut bien sûr effectuer une mise à jour de l'ensemble HasAccessed(s) chaque fois que le sujet s accède à un objet de l'ensemble OP. Notons que la notion d'historique d'accès ne fait pas partie de la sémantique d'un langage de règles simples.

Règles pour les conflits actuels

La règle qu'on vient de mentionner peut être en conflit avec toute autre règle qui stipule, par exemple, que tout sujet avec un certain rôle aura la permission d'accéder à tout objet de l'ensemble OP. Ceci soulève le besoin de spécifier que l'exigence d'empêcher l'accès à la totalité des éléments de OP est prioritaire par rapport aux autres exigences de contrôle d'accès.

Règles pour les conflits futurs

Lors de la spécification ou de la modification des ECAE, si une nouvelle règle est ajoutée, qui permet à un certain sujet un accès aux éléments d'un ensemble d'objets non disjoint de l'ensemble OP, alors il faut gérer un conflit possible entre cette nouvelle règle et la première règle exprimant l'interdiction de l'accès à la totalité de OP.

Cet exemple a permis de montrer comment une seule ECAE d'une entreprise n'a pas pu être formulée par une règle simple. Il a plutôt fallu recourir à trois groupes de règles : (1) un groupe de règles pour définir l'ECAE, (2) un autre groupe de règles pour garantir que l'ECAE considérée n'est en conflit avec aucune autre règle déjà existante ; (3) et un troisième groupe de règles pour résoudre les éventuels conflits entre l'ECAE considérée et toute nouvelle règle ajoutée pour exprimer une nouvelle ECAE ou la modification d'une ECAE existante. On constate que les règles résultant d'une seule ECAE de type SOD sont dispersées dans plusieurs groupes; ceci nuit à leur lisibilité au sein d'une spécification des ECAE. À l'évidence, certaines ECAE ne peuvent pas être spécifiées avec un ensemble de règles confinées dans un module cohésif (dénotant une politique), ce qui permettrait de réduire la complexité de leur spécification.

Une spécification des ECAE utilisant les modèles de contrôle d'accès

Une meilleure solution pour spécifier l'exemple précédent, commence par la définition du concept d'ensemble d'objets, inaccessible dans sa totalité. Pour un tel ensemble, on doit empêcher qu'un même sujet puisse accéder à la totalité des éléments de cet ensemble. Un tel ensemble sera dit ***partiellement accessible*** et se verra attribuer une valeur indiquant le nombre maximal d'objets de cet ensemble auquel un même sujet peut accéder.

Ceci étant, pour spécifier l'ECAE : « un ensemble d'objets ne doit pas être accessible dans sa totalité par un même sujet », il suffit de fournir une seule assertion qui définit cet ensemble comme étant *partiellement accessible*. Cette assertion pourra avoir la forme suivante :

Partiellement-accessible ($\{o_1, \ldots\ldots, o_n\}$, n-1)

Cette spécification signifie qu'un même sujet peut accéder au maximum à n-1 objets parmi les n objets $o_1, \ldots\ldots, o_n$.

Une telle assertion spécifiant une ECAE n'a pas la forme d'une règle simple, elle représente une application d'un modèle de contrôle d'accès qu'on peut appeler *partiellement accessible.* Remarquons ici, qu'on n'a pas à spécifier la logique donnant l'interprétation de *partiellement accessible*, ce qui allège et améliore la lisibilité de la spécification de nos ECAE. De plus, nous prévoyons que les règles de résolution de conflits seront simplifiées dans le cas de conflits entre des applications de plusieurs modèles de contrôle d'accès. Dans notre exemple, on pourrait spécifier, une fois pour toutes, si une règle construite selon le concept d'ensemble *partiellement accessible* prévaudra ou non, par rapport aux autres règles qui pourraient être présentes. Remarquons qu'on passe de règles verbeuses pour chaque ECAE, à des spécifications basées sur des ensembles de concepts prédéfinis au niveau des modèles de contrôle d'accès. On peut alors spécifier comment régler les conflits au niveau des applications « instances » des différents modèles de contrôle d'accès.

Par ailleurs, notons que la notion courante en contrôle d'accès de l'exclusion mutuelle pour l'accès à l'un ou l'autre de deux objets o1 et o2, peut se formuler par :

Partiellement-accessible ({o1, o2}, 1)

Pour conclure, on a vu qu'en absence de la spécification d'une application du modèle de contrôle d'accès *partiellement accessible*, on a besoin d'un ensemble de règles pour formuler une ECAE exprimant le concept de *partiellement accessible*. Cet ensemble est formé de règles disséminées dans trois groupes. La spécification de cette ECAE perd sa cohésion de forme, ce qui nuit à la modularité recommandée dans la gestion d'un système de contrôle d'accès pour des raisons relatives à la réduction de la complexité, à la maintenance et à l'évolutivité du système. Avec des modèles de contrôle d'accès, on se dirige plutôt vers une situation où l'ajout d'une ECAE se limite à l'ajout d'un élément représentant une instance d'un modèle de contrôle d'accès. Cela permet une meilleure gestion des ECAE régissant un système de contrôle d'accès d'entreprise. Il s'agit donc de considérer la spécification des ECAE comme instances de modèles de contrôle d'accès et cela sera dit une spécification des ECAE au ***niveau des modèles***. Pour sa part, l'expression du contrôle d'accès sous forme de règles sera appelée, dans la suite de ce document, une spécification au ***niveau des règles***.

1.1.3 Conflits, incohérences et algorithmes de composition

On considère dans cette section les systèmes de décisions qui sont par définition des spécifications de contrôle d'accès qui, interprétées individuellement, permettent des décisions de contrôle d'accès en réponse à des demandes d'accès. Ainsi, les règles de contrôle d'accès, les politiques de contrôle d'accès et certaines spécifications des exigences de contrôle d'accès sont des systèmes de décision. La structure de la règle qu'on a déjà expliquée indique le fait qu'une règle s'interprète par une décision de contrôle d'accès en réponse à certaines demandes d'accès. Il en est de même pour une politique ou une spécification de contrôle d'accès qui consiste en un ensemble de règles dont l'interprétation mène à une décision de contrôle d'accès.

Remarquons qu'une spécification de contrôle d'accès n'est pas nécessairement un système de décision. En effet, considérons une spécification S qui stipule qu'un certain sujet joue un rôle r1 sans fournir les privilèges associés au rôle r1. Alors S n'est pas suffisante pour prendre une quelconque décision de contrôle d'accès; S n'est pas un système de décision.

Deux systèmes de décision sont *en conflit* si, pour une même demande d'accès, ils aboutissent à deux décisions de contrôle d'accès différentes. À titre d'exemple, considérons le cas d'un sujet qui, d'une part, tient un rôle lui donnant le privilège de lire un objet o et, d'autre part, est concerné par une autre règle le privant de ce privilège. En absence d'une règle de prévalence entre ces deux dernières règles, on aura une spécification qui retourne deux décisions contradictoires (l'une de « Permit » et l'autre de « Deny ») pour la même demande d'accès.

De façon générale, si la spécification des ECAE d'une entreprise renferme deux systèmes de décision présentant un conflit, elle mènera à des décisions différentes (contradictoires) pour au moins une demande d'accès. La spécification est alors dite *incohérente*; mais elle pourra devenir *cohérente* si on la munit de moyens permettant de conclure, à partir des différentes décisions des systèmes de décision impliqués, une décision unique. Ces moyens sont spécifiés par ce qu'on appelle un algorithme de composition ou de combinaison de décision (en anglais *combining algorithm)* (ComAl).

Un ComAl est une fonction qui prend, en entrée, une liste de décisions et retourne, en sortie, une décision unique. Certains ComAl se limitent à faire prévaloir certaines décisions par rapport à d'autres, alors que d'autres ComAl sont plus compliqués à spécifier et peuvent tenir compte des valeurs de variables additionnelles pour retourner une décision unique, en fonction de la liste des décisions d'entrée.

Certains langages, tels que XACML [4], permettent de spécifier des méthodes de résolution de conflits entre les règles et entre les politiques de contrôle d'accès. Considérons, par exemple, l'algorithme de *Deny-overrides* qui stipule qu'une décision « Deny » en conflit avec n'importe quelle autre décision, résultera en une décision

« Deny ». Cet algorithme ne tient pas compte de la source de chaque décision, en termes de politiques d'entreprise, ce qui n'est pas convenable pour des ECAE où le motif derrière une décision détermine et justifie généralement sa prévalence par rapport aux autres décisions. Il importe de remarquer que ces algorithmes de combinaison prédéfinis dans XACML traitent les règles en conflit sans prendre en charge les concepts de modèles de contrôle d'accès qui les sous-tendent. De tels algorithmes ne sont pas suffisants pour exprimer comment régler toute sorte de conflit entre des règles de contrôle d'accès. De plus, nous pensons que les conflits entre les spécifications de contrôle d'accès seront plus simples à spécifier quand ces spécifications seront catégorisées selon des instances de modèles de contrôle d'accès.

Cette situation d'insatisfaction nous incite à considérer l'intégration, dans le langage que nous envisageons, de moyens d'expressivité convenables pour la résolution de conflits par des ComAl; un sujet de recherche qui a intéressé plusieurs chercheurs [5], [6] et [7].

La vérification de propriétés

L'élaboration d'un langage de spécification d'ECAE vise non seulement une expressivité convenable mais aussi l'établissement d'une base pour la vérification de certaines propriétés des ECAE. On s'intéresse principalement aux propriétés de cohérence et d'incomplétude qu'on va expliquer dans cette section.

Une *spécification incohérente* de contrôle d'accès a une interprétation qui n'est vraie dans aucun contexte. Dans la section précédente, on a vu un exemple de spécification de contrôle d'accès incohérente car les deux systèmes de décision qui interprétaient cette spécification retournaient des décisions contradictoires (l'un « Permit » et l'autre « Deny »). Cette situation de conflit n'est pas la seule forme d'incohérence que peut avoir une spécification de contrôle d'accès. En effet, considérons le cas d'un langage de spécification d'exigences de contrôle d'accès qui permet de spécifier les différents rôles qu'un sujet peut tenir, en associant une liste de rôles à chaque sujet. Ce même langage permet de spécifier les différents sujets affectés à un rôle, en associant à chaque rôle une liste des sujets qui peuvent tenir ce rôle. On pourrait alors avoir, pour un sujet s1, une

spécification de liste de rôles contenant un rôle r1, alors que pour le rôle r1, la liste des sujets pouvant tenir ce rôle pourrait avoir été spécifiée en omettant s1. Ces deux spécifications sont contradictoires, mais la détection de cette contradiction n'est pas toujours évidente.

La détection, par un procédé automatique, des incohérences des exigences de contrôle d'accès est primordiale et indispensable quand on envisage des ECAE, en raison de leur grand volume et de leur complexité qui dépassent les capacités non automatiques de détection d'incohérence. Il est donc nécessaire que le langage de spécification d'ECAE envisagé se prête au traitement automatique.

Après l'incohérence, l'incomplétude est un autre souci de tout administrateur de contrôle d'accès. Elle correspond à l'existence de cas de demandes d'accès pour lesquels aucune règle de contrôle d'accès n'est applicable. En revanche, une spécification de contrôle d'accès qui permet de retourner une décision de contrôle d'accès en réponse à toute demande d'accès est dite complète. Ceci définit aussi la caractéristique de complétude d'une spécification de contrôle d'accès. L'incomplétude et l'incohérence sont **les anomalies** les plus courantes d'une spécification de contrôle d'accès. Leur prise en charge sera toujours un but à atteindre dans tout cadre de travail de spécification des ECAE. Un exemple simple illustrant ces anomalies est fourni dans l'énoncé suivant des exigences de contrôle d'accès d'une machine qui dispense du tabac.

La décision d'accès à l'achat automatisé de tabac pour un client qui s'est enregistré et identifié, est fonction de certains attributs de ce client, conformément aux règles suivantes :

- Étudiant universitaire => Permit
- Étudiant CEGEP => Deny
- Âgé de plus de 18 ans => Permit

Cet ensemble de règles n'est pas complet car il ne spécifie pas quelle décision prendre dans le cas où le client n'est ni étudiant, ni âgé de plus de 18 ans. De plus, cette spécification est incohérente s'il y a des étudiants de CEGEP qui sont âgés de plus de 18

ans, car les deux dernières règles s'appliqueront à eux en fournissant deux décisions contradictoires de « Deny » et de « Permit ».

1.2 Un langage modulaire avec un langage cible

Notre contribution pour la solution des problèmes que nous avons décrits sera sur l'élaboration d'un nouveau langage de spécification de contrôle d'accès pour les applications d'entreprise. Le modèle abstrait de ce langage sera basé sur l'intégration de plusieurs instances de différents modèles de contrôle d'accès; chaque instance est un système de décision et elle représente un module de contrôle d'accès. Notre langage sera appelé *Modular Access Control Language* (MACL).

En plus de l'expressivité, la détection des anomalies constitue un but essentiel de MACL. Notre approche pour atteindre ce but, va se baser sur un mapping de tout énoncé en MACL en un énoncé dans **un langage cible** qui est basé sur la logique de premier ordre (FOL). Plus précisément, le mapping que nous introduirons dans la Section 5.4 sera orienté vers une forme restrictive du FOL qui est le langage du Constraint Logic Programming et est limitée aux *definites clauses* [8]. Notons que le langage des *definites clauses* est indécidable comme FOL [8]. Remarquons aussi que dans le cas de domaines finis, qui seraient suffisants pour les spécifications des exigences de contrôle d'accès, tant FOL que CLP sont décidables. Cependant, le langage des *definites clauses* bénéficie de procédures de résolution efficaces surtout *SLD Resolution* [41], ce qui n'est pas le cas pour FOL. Ceci convient à notre prototype d'implémentation présenté au Chapitre 8 qui est basé sur le Constraint Logic Programming (CLP) [8]. Notre prototype implémente une partie de nos objectifs offrant ainsi une illustration de leur faisabilité, alors que notre travail conceptuel se veut libre des restrictions du CLP pour préparer le terrain à toute implémentation future visant bénéficier des possibilités fournies par des formes moins restrictives de FOL. Pour simplier, dans la suite de ce document nous allons qualifier par CLP le langage cible en faisant abstraction des quelques cas où nous nous libérons de certaines de ses restrictions.

Le mapping vers une logique CLP, doit permettre de fournir une représentation précise et complète de la sémantique de chaque module, ainsi que de l'intégration des différents modules. Ce mapping est essentiel pour MACL et sera exploité pour la spécification formelle des ComAl.

Ainsi, tel que détaillé dans les Sections 7.2.1 et 9.5.2, toute spécification d'ECAE dans MACL sera transformable en un énoncé du langage cible qui conviendra à un traitement automatique de détection d'incohérence ou de vérification de propriétés exprimables dans des énoncés de ce langage. Le point fort du langage proposé réside dans le fait qu'il utilise des concepts de contrôle d'accès (modèles) qu'on trouve dans les spécifications informelles des ECAE. À noter ici, que les exemples présentés dans ce chapitre ont servi le but d'illustrer les règles et les modèles, mais dans la suite de notre étude nous allons élaborer sur d'autres exemples qui conviendrons mieux pour illustrer un sous-ensemble choisi de MACL.

On mentionne ici, que notre travail se limite à l'élaboration d'un langage de spécification de contrôle d'accès sans s'étendre à d'autres approches de recherche qui s'intéressent à l'émission de décisions dans les domaines des systèmes experts en intelligence artificielle et des bases de données. Notre approche a un but précis et avec plus de généralité nous risquerions de perdre de vue notre domaine d'application. Aussi, la considération d'applications différentes et plus générales nous conduirait à des solutions plus complexes et moins efficaces.

1.3 Récapitulation et conclusion

Dans ce chapitre, nous avons identifié notre problématique et exploré les moyens de la résoudre. Les éléments de la problématique sont :

1. La situation d'insatisfaction résultant de la grande difficulté de spécifier des ECAE en utilisant les règles simples de contrôle d'accès.
2. L'inadéquation totale des règles simples pour spécifier certaines exigences de contrôle d'accès.

Les éléments de l'approche envisagée sont :

1. Un langage de spécification de contrôle d'accès appelé MACL.
2. L'expression de chaque ECAE en tant qu'instance d'un modèle de contrôle d'accès.
3. Chaque instance d'un modèle de contrôle d'accès constituera un bloc de construction de MACL; et un énoncé en MACL consistera en une ou plusieurs instances de modèles de contrôle d'accès, qu'on appellera module(s).
    a. MACL fournira une méthode pour la définition et l'intégration des modules;
    b. L'intégration des modules reposera sur des algorithmes de règlement de conflits entre modules, utilisant le concept présenté de *combining algorithms*.
4. Le langage MACL sera doté de moyens de détection des anomalies que nous avons décrites. À cet effet, nous définirons un mapping des énoncés en MACL vers un langage qui supporte l'analyse automatique de ses énoncés. Ce mapping aura deux volets :
    a. Un pour les modules de MACL qui sont des instances de modèles de contrôle d'accès;
    b. Un autre pour les spécifications des *combining algorithms* permettant l'intégration des modules de MACL.

Ainsi, MACL permettra : (1) des expressions moins laborieuses et moins longues des ECAE; (2) une meilleure séparation modulaire entre les énoncés exprimant les différents ECAE d'une entreprise selon ses différents buts; (3) une prise en charge de la détection des anomalies.

Chapitre 2 État de l'art

Dans ce chapitre, on fera un survol des langages de spécification de contrôle d'accès. On s'intéressera à la pertinence de ces langages pour l'expression des ECAE et la détection des anomalies. Particulièrement, on va constater que pour l'ensemble des modèles de contrôle d'accès, aucun des langages existants ne permet: (1) l'expression des ECAE en termes d'application de modèles de contrôle d'accès; (2) la réutilisation de la logique de décision introduite dans la Section 1.1.2. Une représentation tabulaire illustre ceci à la Section 12.2. On vise à tirer des leçons pour élaborer un langage qui amène une contribution dans le domaine.

2.1 Les langages normalisés de contrôle d'accès

Dans le contexte courant où plusieurs parties fournissent des données et partagent la responsabilité d'en contrôler l'accès, un langage standardisé de spécification des ECAE est un moyen utile facilitant l'intégration des politiques de contrôle d'accès en provenance des différentes parties.

2.1.1 XACML

Le standard eXtensible Access Control Markup Language (XACML) [4] normalise un langage de spécification des ECAE. XACML se base sur une architecture de contrôle d'accès décrivant les interactions entre différents systèmes impliqués dans le contrôle

d'accès. De plus, ce standard définit un ensemble d'éléments XML avec leurs attributs, dans le but d'offrir un schéma de langage de spécification de contrôle d'accès.

Les différents systèmes impliqués dans le contrôle d'accès selon XACML, échangent et traitent des données pour effectuer les actions reliées au contrôle d'accès. Cet échange de données est illustré dans le diagramme de flots de données de la Figure 2.1-1. Tout d'abord, le contrôle d'accès a pour but de limiter l'accès aux objets en se conformant à un ensemble de politiques de contrôle d'accès dit *Policy Set.* L'entité responsable de la fourniture du *Policy Set* est dite *Policy Administration Point* (PAP). Le PAP rend son ensemble de politiques de contrôle d'accès, le *Policy Set,* disponible à un système dit le *Policy Decision Point* (PDP) qui est responsable de la détermination de décision de contrôle d'accès conforme au *Policy Set.* Le déroulement des traitements de contrôle d'accès est initié par un demandeur d'accès (*access requester*). Sa demande d'accès (*access request*) renferme des informations l'identifiant comme sujet demandant l'accès, et identifiant la ressource à laquelle il désire accéder, l'action représentant le mode d'accès demandé et optionnellement des valeurs de certains attributs du sujet, de la ressource et de l'environnement d'exécution.

Le demandeur d'accès initie les traitements de contrôle d'accès en soumettant sa demande d'accès (*access request*) à un système informatique responsable d'assurer le contrôle d'accès. Ce système est dit *Policy Enforcement Point* (PEP). Le PEP reformule la demande d'accès pour l'adresser à une entité qu'on appelle *context handler. Le context handler* est responsable de passer cette demande au PDP selon le format approprié, pour obtenir une réponse à la demande d'accès. Notons aussi que le *context handler* est responsable de fournir, suite à une demande du PDP, des informations supplémentaires sur les attributs des sujets, des ressources, de l'action et de l'environnement. Ces informations sont à chercher par le *context handler* en communiquant avec un système agissant en tant que source de valeurs des attributs, qui est appelé *Policy Information Point* (PIP).

Ayant reçu la demande d'accès et les valeurs des attributs nécessaires pour aboutir à une décision de contrôle d'accès conforme au Policy Set, le PDP va produire cette décision qu'on appelle *response context* selon le diagramme de la Figure 2.1-1. Le *context handler* passe ce dernier message au PEP. Le PEP est responsable de la mise en application (e*nforcement*) de cette réponse-décision de contrôle d'accès. Cette action d'*enforcement* prise en charge par le PEP est nommée « Effect » et elle comprend deux volets. Le premier est de rendre l'action demandée sur la ressource disponible ou hors portée vis-à-vis du demandeur d'accès, selon que la réponse est positive (égale à « Permit ») ou négative (égale à « Deny »). Le deuxième est de forcer l'exécution de certains traitements exigés par les politiques applicables à la demande d'accès et des obligations.

Ayant discuté le modèle architectural de contrôle d'accès de XACML, on va examiner de près l'élément *Policy Set* de cette architecture, car il représente les spécifications de contrôle d'accès qui alimentent le PDP. Le *Policy Set* est un ensemble de politiques de contrôle d'accès, et chaque politique de contrôle d'accès (policy selon le diagramme) est un ensemble de règles de contrôle d'accès. Une politique de contrôle d'accès est associée aussi à une ou plusieurs obligations. Expliquer le Policy Set revient à expliquer le concept de règle et d'obligation.

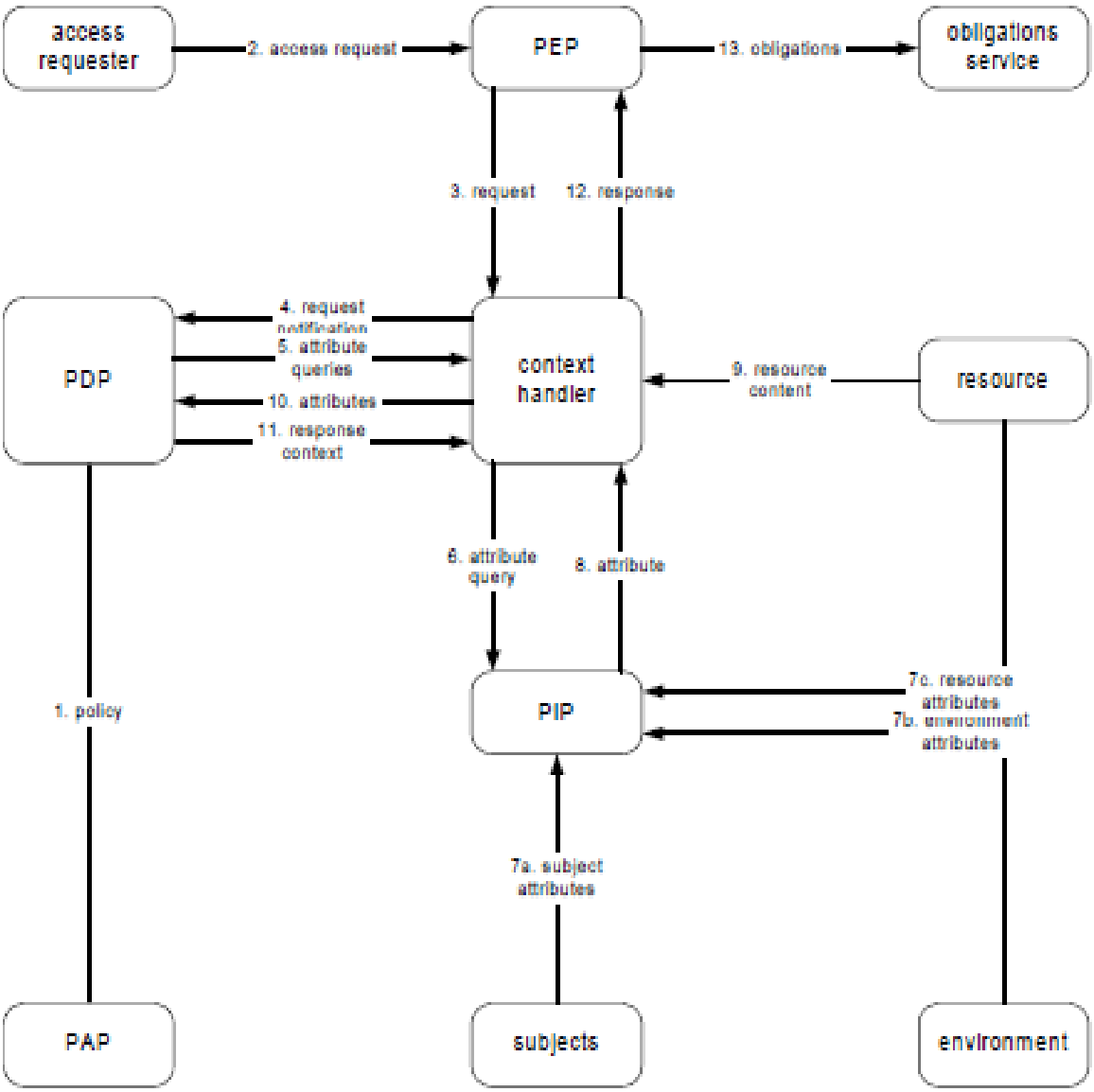

Figure 2.1-1 Diagramme de flots de données [4]

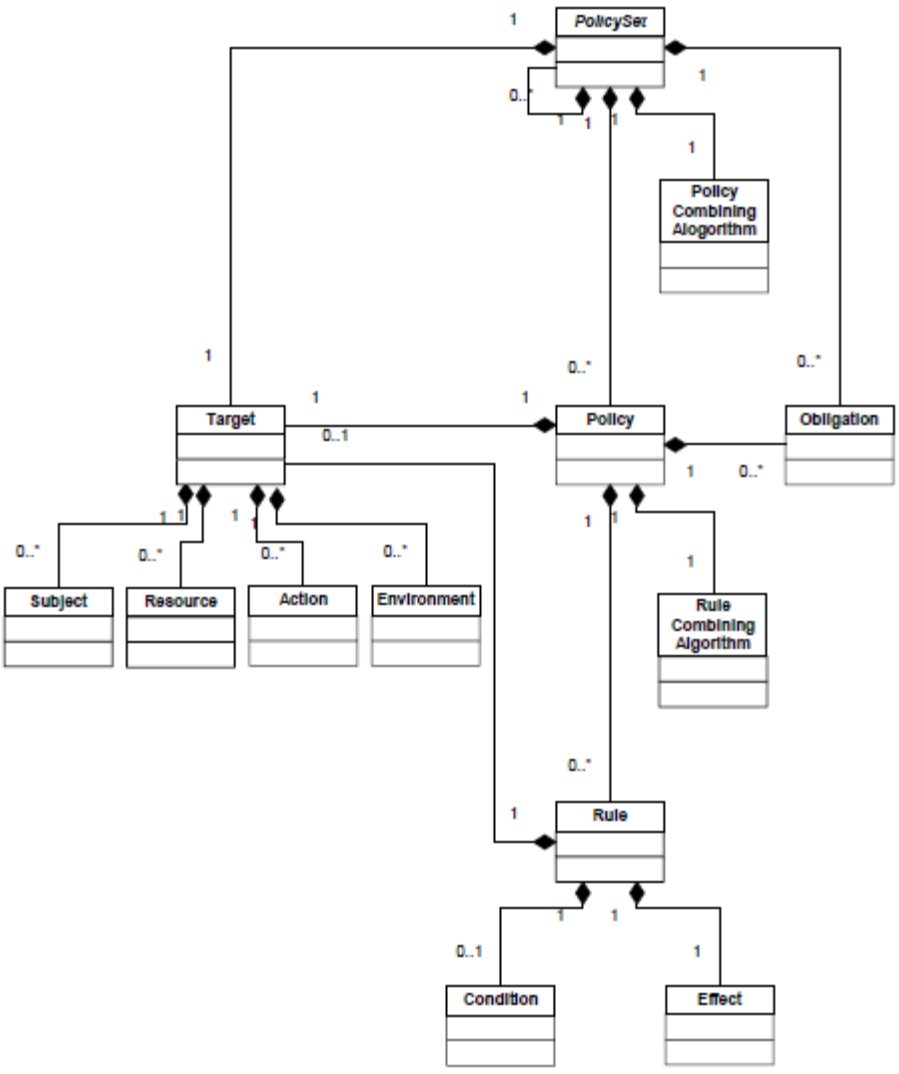

Figure 2.1-2 Illustration du modèle de XACML [4]

La Figure 2.1-2 donne une illustration du schéma du *Policy Set*. On y trouve le concept de règle qui est à la base de la spécification XACML; une règle applique une condition élémentaire de prédicat pour retourner, suite à une demande d'accès, une réponse représentée par l'élément « Effect ». Cet élément appartient à une énumération de décisions d'autorisation qui se limite à « Permit » et « Deny ». Notons que le concepteur d'une règle définit cette règle pour l'appliquer à certaines demandes d'accès. Ces demandes sont ainsi ciblées par cette règle et forment ce qu'on appelle le « Target » de cette règle. Selon le modèle ci-contre, une instance de l'élément « Target » représente un ensemble de demandes d'accès caractérisées par des combinaisons de sujets, de

ressources, de modes d'accès (actions) et de valeurs de certaines variables de l'environnement. Un mode d'accès représente la nature de l'action qu'un sujet peut exécuter sur une ressource accessible, entre autres, lecture, écriture, et exécution. Ces derniers modes d'accès sont désignés dans la suite par : Read, Write, Execute.

Toute instance de target est associée à un « PolicySet », une « Policy » ou une « Rule » (règle), pour indiquer une relation d'applicabilité de ces dernières afin de décider des demandes d'accès représentées par l'instance de « Target ».

D'autre part, une obligation représente une opération, spécifiée dans une politique ou un ensemble de politiques, qui doit être exécutée par le PEP en conjonction avec l'*enforcement* d'une décision d'autorisation. La notion d'obligation est supportée par XACML même, avec prise en charge du cas de plusieurs obligations émanant de plusieurs politiques ou ensemble de politiques applicables pour répondre à une seule requête de décision. Ainsi, si plusieurs politiques s'appliquent à une demande d'accès, alors plusieurs obligations sont à exécuter en conjonction avec l'*enforcement* de la décision. On retient pour exécution toutes les obligations de toutes les politiques applicables à une requête avec une décision égale à la décision retournée par le PDP. Remarquons qu'une politique de plusieurs règles applicables peut retourner une décision de contrôle d'accès différente de celle retournée par le PDP qui se base sur l'ensemble de politiques applicables du *Policy Set* pour retourner une décision. Ainsi, les obligations associées à une politique applicable à une demande d'accès et retournant une décision différente de celle retournée par le PDP, ne seront pas à exécuter.

XACML prend en considération le besoin de résoudre les conflits entre les règles ou entre les politiques. Il permet de spécifier les règles de résolution des conflits en choisissant un algorithme de combinaison « Combining algorithm » parmi un ensemble prédéfini dans la spécification XACML. Un algorithme de combinaison de plusieurs politiques ou règles est la spécification qui permet de déduire une décision unique à partir des décisions retournées par les dites politiques ou règles. Le standard permet qu'on spécifie ses

propres algorithmes de combinaison et qu'on les associe aux politiques et ensembles de politiques, mais XACML ne fournit pas de moyens formels pour spécifier ces algorithmes.

De plus, comme son nom l'indique, eXtensible Access Control Markup Language (XACML) permet d'étendre sa sémantique par le biais de définitions de nouveaux attributs. Ceci va dans la direction d'une meilleure expressivité des particularités spécifiques de certaines ECA. Cette extensibilité de XACML a déjà été exploitée et a résulté en une multitude de profils couvrant entre autres :

- L'exigence d'un lien fort entre le but de collecte des informations et l'usage de ces informations;
- le modèle RBAC [9];
- Les exigences d'authentification et d'intégrité avec les signatures numériques.

Chaque extension résout un problème au niveau d'un modèle de contrôle d'accès. On peut en tirer des leçons sur comment se baser sur des règles simples pour bâtir des artefacts exprimant des concepts de contrôle d'accès au niveau des modèles de contrôle d'accès. De plus, les concepts de politiques et d'ensembles de politiques avec leurs algorithmes de combinaisons aident à gérer un grand nombre de règles, mais ceci n'est pas suffisant pour atteindre le niveau des modèles comme on l'a déjà expliqué dans l'introduction.

XACML s'avère assez flexible pour l'expression des exigences de contrôle d'accès, mais il n'est pas « orienté » vers le niveau des modèles de contrôle d'accès avec des moyens d'expressivité convenable, d'implémentation et de détection des anomalies d'incohérence et d'incomplétude. De plus, les moyens de réutilisation de la logique de décision échappent à ce langage. Son extensibilité et ses mécanismes de combinaison de politiques et de règles seront deux points forts qui inspireront notre recherche et laisseront la porte ouverte à des travaux futurs de conception de profils de XACML en réponse aux besoins émergeants dans le domaine.

Notons enfin qu'une implémentation du PDP basée sur la spécification de contrôle d'accès en XACML a été fournie par Sun Microsystems [10].

La spécification XACML définit un ensemble de termes reliés étroitement aux éléments de son modèle illustré dans la Figure 2.1-1 et la Figure 2.1-2 . Nous allons fournir ces définitions car nous allons nous en servir comme vocabulaire de base du domaine de contrôle d'accès dans la suite de notre travail. Les termes sont cités par ordre alphabétique, les mots en caractères gras réfèrent à des termes définis dans la même liste :

Accès : exécution d'une opération sur une ressource.
Algorithme de combinaison de politiques : procédure qui permet de retourner une **décision** en se basant sur des décisions retournées par un ensemble de politiques.
Condition : expression booléenne qui s'évalue à *vrai* ou faux.
Contexte : représentation d'une requête et d'une décision d'autorisation.
Décision d'autorisation (Décision): résultat de l'évaluation d'une politique applicable, qui est retournée par le **PDP** au **PEP**. C'est une fonction dont la valeur est un élément de l'énumération « Permit », « Deny », « Indeterminate », « NotApplicable » et éventuellement un ensemble d'**obligations**.
Obligation : opération spécifiée dans une politique ou un ensemble de politiques qui doit être exécutée par le PEP en conjonction avec la mise en application (*enforcement*) d'une décision d'autorisation
Policy Decision Point (PDP) : entité du système qui évalue une politique applicable et retourne une décision d'autorisation.
Policy Enforcement Point (PEP) : entité du système qui performe le contrôle d'accès en faisant les requêtes de décision et en forçant les décisions d'autorisation.
Policy Information Point (PIP) : entité du système qui agit en tant que source de valeurs des attributs.
Privacy officer : personne chargée de spécifier les exigences de contrôle d'accès dans une entreprise.

Ressource (Objet) : donnée, service ou un composant du système.

Requête de décision (requête) : demande d'un PEP à un PDP de retourner une décision d'autorisation en fonction de la spécification fournie par le PEP du sujet, de l'objet et de l'action à exécuter sur l'objet.

Système de contrôle d'accès (SCA) : ensemble des données résultant d'une spécification des ECAE et des données de l'environnement sur lesquelles se base un PDP pour produire les décisions d'autorisation.

Target ou cible: ensemble de requêtes de décision, identifiées par ressource, sujet et action qu'une règle, politique ou ensemble de politiques sont responsables d'évaluer.

2.1.2 SAML

Un autre standard à mentionner est Security Assertion Markup Language (SAML) [11] qui normalise la forme d'un ensemble d'assertions à générer ou à échanger. Ces assertions expriment en général : (1) un accomplissement de l'authentification d'un sujet; (2) une association entre un sujet et un attribut; (3) une décision d'autorisation positive ou négative d'accès à une ressource par un sujet. SAML est conçu dans le souci d'échange d'informations qui concernent la sécurité entre différentes parties, et ce en conformité avec des schémas reliant une syntaxe bien déterminée à des sémantiques de sécurité bien documentées.

SAML ne présente pas de moyens d'expression appropriés pour les modèles de contrôle d'accès permettant la réutilisation de la logique de décision, mais comme XACML il représente une norme qui peut privilégier toute élaboration de modèle de contrôle d'accès qui s'y conforme.

2.2 Langages de spécification de contrôle d'accès

En survolant l'état de l'art du domaine des langages de spécification des exigences de contrôle d'accès, on s'aperçoit qu'il y a un créneau de recherche bien établi visant des objectifs en relation avec les nôtres. Ces travaux s'inscrivent dans le but général de permettre une spécification d'ECA plus robuste vis-à-vis des anomalies de cohérence et

de complétude. La plupart de ces travaux ne dépassent pas le niveau des règles de contrôle d'accès. Certains travaux ont proposé des moyens appropriés pour l'expression de certaines ECAE au niveau des modèles mais les résultats sont restés isolés et partiels et n'offrent pas un moyen de réutiliser la logique de décision.

2.2.1 Ponder

Ponder [12] est un langage de spécification d'exigences de contrôle d'accès qui contrôle l'accès par des règles structurées en événement-condition-action. C'est un langage fortement typé qui se base sur des **déclarations** qui supportent, entre autres, les rôles hiérarchiques, les structures organisationnelles d'une entreprise, la délégation des droits d'accès, l'enregistrement des usagers, le filtrage des informations accessibles d'un objet en fonction de certains attributs et paramètres, et l'autorisation négative. Certains de ces concepts sont au cœur de la spécification de contrôle d'accès au niveau des modèles et font que Ponder va être, pour nous, un bon langage de référence que nous voulons dépasser en ciblant une orientation plus axée sur le contrôle d'accès au niveau des modèles et couvrant des concepts supplémentaires comme le Chinese Wall[13], Bell-LaPadula [3] et la combinaison de politiques de contrôle d'accès. L'explication de ces concepts fera l'objet du chapitre suivant.

Ponder est basé sur un ensemble de règles dont la forme ne se limite pas à l'attribution conditionnelle d'un privilège d'accès à un sujet. On a plusieurs catégories de règles qui permettent les déclarations mentionnées dans le paragraphe précédent, comme les rôles hiérarchiques et les structures organisationnelles. De plus, une réutilisation des déclarations est aisée en utilisant la définition de types et l'instanciation de ces types. Cela permet de créer des règles génériques prêtes à être réutilisées selon différentes valeurs de paramètres.

Les obligations de Ponder spécifient des actions à déclencher suite à des événements. Elles offrent plus de flexibilité que les obligations au sens XACML qui sont associées à des actions d'exécution de décisions d'accès. Cela permet de spécifier des fonctionnalités de surveillance réactive pour un système décrit par Ponder.

À noter aussi l'utilisation d'un sous-ensemble de l'Object Constraint Language (OCL) [14] dans l'expression de prédicats et dans la recherche et la sélection d'objets et de leurs attributs. OCL a fourni une contribution majeure à l'élaboration de Ponder par sa convenance au paradigme orienté objet et par le fait que OCL est un standard bien connu du consortium Object Management Group[2] (OMG).

OCL a permis aussi la spécification aisée des **contraintes** au niveau d'une politique de contrôle d'accès ainsi qu'au niveau de plusieurs politiques. Les contraintes sont considérées comme des métapolitiques quand elles contraignent plusieurs politiques. Ces contraintes déclenchent des **exceptions** avec des paramètres, quand elles sont violées. Ces métapolitiques ne sont pas suffisantes pour spécifier la composition de plusieurs politiques. La composition de politiques permet de résoudre des conflits et de les rendre complémentaires plutôt que sources de décisions de contrôle d'accès divergentes. Par exemple, considérons deux politiques de contrôle d'accès, p1 et p2, qui fournissent des décisions membres de l'ensemble Σ = {Permit, Deny}. Ainsi, en réponse à une requête d'accès, chacune de politiques p1 et p2 fournit une décision, ce qui résulte en un couple de décisions élément de Σ^2 au lieu d'une décision unique. Une composition de ces deux politiques consiste à associer à tout couple de décisions de contrôle d'accès appartenant à Σ^2 une décision unique. Néanmoins, ces métapolitiques peuvent spécifier des conditions d'applicabilité des politiques et empêcher certains conflits entre elles.

Soulignons aussi les avantages que Ponder a tiré du paradigme orienté objet. Ponder jouit d'un contrôle sur les types lors de la compilation ce qui offre un premier pas dans la validation. Ponder profite aussi de l'héritage qui permet une bonne structuration des objets favorisant des spécialisations de contrôle d'accès à partir des types déjà définis. Notons que Ponder part d'un nombre limité de types prédéfinis correspondant aux déclarations supportées par ce langage et mentionnées dans le premier paragraphe de

[2] http://www.omg.org/

cette section, comme les rôles hiérarchiques et les délégations. La création de nouveaux types est limitée aux types prédéfinis et aux types qui en sont dérivés.

Comme exemple illustrant la spécification avec Ponder de contrôle d'accès au niveau des modèles, on va considérer la spécification de la séparation des tâches ou de « duty » (SOD) et de la délégation. Selon ce principe, dont on a donné un exemple dans la section 1.1.2, on veut empêcher qu'une même personne puisse émettre un chèque puis l'approuver sans avoir besoin de l'intervention d'une autre personne. En d'autres termes, l'émetteur d'un chèque et la source de son approbation doivent être deux personnes distinctes. Selon l'exemple présenté dans [12], la séparation des tâches est exprimable en Ponder par une règle spécifique d'autorisation pour chaque couple d'actions à séparer. On utilise une règle d'autorisation simple avec ajout d'une condition d'exclusion mutuelle. L'objet chèque doit avoir une propriété qui s'appelle *Issuer* et qui est comparée avec le demandeur d'accès à la méthode d'approbation de ce chèque. Cela permet de spécifier une condition d'exclusion mutuelle particulière concernant l'émission de chèque et son approbation, mais cela laisse à désirer car la condition d'exclusion mutuelle dépend de la spécification de propriétés (*Issuer* dans notre exemple) de certains objets qui gardent trace de l'historique des accès à ces objets en plus de la formulation de conditions d'accès en termes de ces propriétés.

En effet, si une deuxième exigence de SOD est à spécifier, la formulation du dernier exemple ne sera pas réutilisable. On doit chercher une expression d'une condition d'accès qui ne ressemble pas nécessairement à celle du premier cas; il se peut qu'aucun des objets ne garde de trace de ses utilisateurs. Le niveau d'expression qu'on vise pour les ECAE est celui qui permet de spécifier toute exigence de SOD selon un moyen **dédié** de ce langage. Ce moyen d'expression dédié devrait être **prédéfini** dans le langage pour spécifier toute exigence de type SOD.

Remarquons aussi que la condition d'exclusion mutuelle de Ponder ne peut pas faire l'objet d'une instanciation pour une réutilisation avec de nouvelles valeurs de paramètres, car elle renferme la condition d'exclusion mutuelle dépendant de l'historique

des accès aux objets qui n'est pas un concept supporté par Ponder. Cet exemple montre qu'on peut exprimer des ECAE au niveau des règles, avec Ponder ou d'autres langages. Cette façon d'exprimer une ECAE est à comparer à l'expression des ECAE en utilisant des éléments du langage de spécification de contrôle d'accès dédiés à des concepts au niveau des modèles de contrôle d'accès. Ces éléments seront réutilisables chaque fois qu'un de ces concepts est appelé dans une spécification des ECAE. Le concept d'exclusion mutuelle a été évoqué au premier chapitre à travers l'exemple de Partiellement-accessible de la page 9, et il sera envisagé dans l'explication de nos objectifs et de notre approche dans les Sections 3.3.6 et 4.1.

À mentionner ici que l'application de certaines contraintes à l'affectation de rôle permet d'assurer une exclusion mutuelle des tâches. Mais elle ne permet pas d'assurer l'exclusion mutuelle en fonction de l'historique des accès. Par exemple, on peut exiger que quelqu'un avec un rôle r1 qui permet d'émettre un chèque ne puisse alors avoir aucun autre rôle permettant d'approuver un chèque. Remarquons que ceci n'est pas suffisant pour spécifier qu'une personne a le droit de signer un chèque et d'approuver un autre chèque émis par quelqu'un d'autre.

Comme autre exemple de concept au niveau des modèles de contrôle d'accès pris en charge par Ponder, on va considérer La **délégation.** La délégation exprime la possibilité de transférer des privilèges d'accès. Ponder permet de spécifier qu'un sujet peut octroyer des privilèges précisés qu'il possède, à un autre sujet pour agir en son nom, avec des restrictions exprimées sous forme de contraintes de temps ou autres, en utilisant OCL. De plus, Ponder permet de spécifier une délégation négative qui a pour effet de limiter les privilèges par délégation au lieu de les étendre. On reconnaît ici la simplicité d'expression de Ponder qui offre une syntaxe instanciant un type prédéfini pour la délégation tout en le peuplant avec des valeurs d'attribut précisant les éléments d'une délégation qui sont:

- Subject : sujet octroyant des privilèges ;
- Grantee : sujet profitant des privilèges transférés par délégation ;
- Target : Objet visé par le transfert de privilèges par délégation ;

- Action : les privilèges d'accès transférables par délégation ;
- When et Valid : pour restreindre la validité et l'applicabilité de la délégation.

Bref, malgré le fait que Ponder ne prenne pas entièrement en charge le niveau des modèles de contrôle d'accès, on note qu'il jouit de plusieurs points forts qu'on peut adopter dans notre travail. Ces points se résument par :

- Un langage orienté objet avec tout ce qui s'en suit :
  - Définition de types et de structures ;
  - Validation des types lors de la compilation ;
  - Extensibilité et réutilisation par héritage et instanciation ;
  - Spécification de prédicats et contraintes avec OCL ;
- La surveillance et la réponse à des **événements** par des **obligations**;
- L'intégration de spécifications de contraintes et de prédicats dans le langage.

Ainsi, notre travail va profiter de ces points dans une perspective qui vise à compléter la réutilisation et l'instanciation des types définis dans Ponder afin d'atteindre nos objectifs. On doit étudier la possibilité de prendre en charge la définition de nouveaux types pour représenter des structures de concepts de contrôle d'accès avec les comportements qui expliquent leur sémantique.

2.2.2 EPAL

Enterprise Privacy Authorization Language (EPAL)[3] [15] est un langage de spécification de politiques de contrôle d'accès de l'entreprise en vue de limiter l'accès aux données d'un système informatique en termes d'autorisations positives et négatives. EPAL qui est un standard du consortium W3C[4], est conçu dans le but d'assurer un **moyen d'échange** de spécifications de contrôle d'accès entre systèmes hétérogènes. Un exemple de cette interopérabilité est la spécification d'une politique de contrôle d'accès aux informations

[3] http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/
[4] http://www.w3.org/

des clients dans le cas d'une activité de vente supportée parallèlement par téléphone, en ligne sur Internet et en personne devant un guichet opéré par un agent.

EPAL s'inscrit dans la liste des langages de niveau règles. On lui reconnaît tout de même des moyens d'expression convenables de niveau modèles utilisant la hiérarchisation de catégories d'usagers et d'objets, et la spécification du but (*purpose)* de l'accès dans les règles. Son modèle des actions est assez riche pour faire, entre autres, la différence entre une permission de lecture et une divulgation de donnée. La **structure d'une règle** relie cinq éléments : **catégorie d'usagers, catégorie de données, but de l'accès, préconditions, actions et obligations**. L'évaluation des règles respecte leur **ordre**, et la première règle applicable va prévaloir. La Figure 2.2-1 illustre le modèle d'EPAL en donnant une vue d'ensemble au schéma de ce langage utilisant le format XML.

Pour comprendre ce schéma, on va commencer par l'interprétation d'une règle d'EPAL représentée par l'élément *rule*. Une règle est applicable à une requête si les éléments composant cette dernière correspondent aux catégories de sujets et d'objets, aux buts de la demande d'accès et aux modes d'accès spécifiés dans la règle, en plus de satisfaire les conditions attachées à cette règle. L'attribut *ruling* de l'élément *rule* peut prendre l'une des deux valeurs « allow » ou « deny »; sa valeur détermine si l'application d'une règle a pour effet une décision de permission ou de refus d'accès.

La catégorisation des usagers et des objets (*user-category* et *data-category* du modèle EPAL) convient aux ECAE qui, généralement, dépendent du groupement des sujets et des objets selon la structure et les activités de l'entreprise. Les obligations sont des opérations à exécuter en conjonction avec l'application d'une règle à une demande d'accès. Une obligation est identifiable par EPAL, mais ce langage ne prend pas en charge la spécification de la logique d'exécution de l'obligation. L'élément « action » représente les modes d'accès permis selon la règle, et ces modes d'accès ne sont pas prédéfinis dans le langage. L'élément *condition* exprime les conditions d'applicabilité de la règle selon une structure permettant la spécification des tâches et de prédicats.

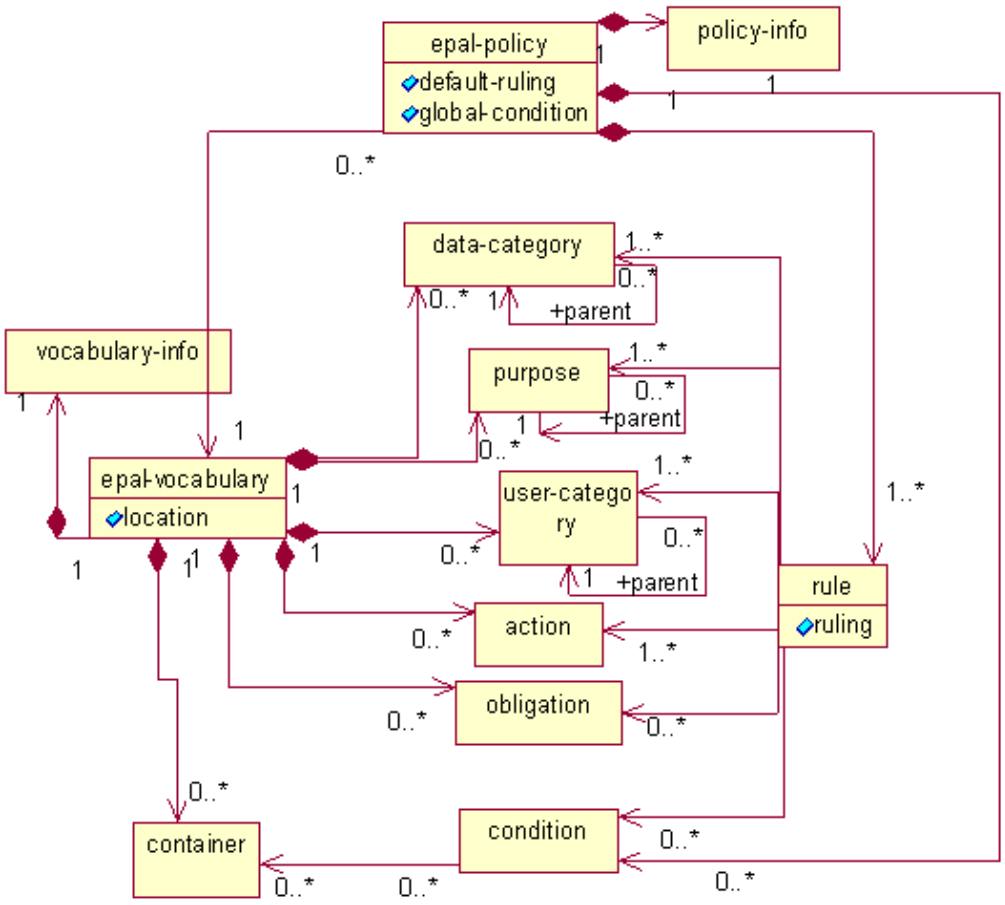

Figure 2.2-1 Modèle du schéma d'EPAL [15]

Epal-vocabulary représente une liste de termes nécessaires pour spécifier une règle comme par exemple les termes définissant l'énumération des actions (modes d'accès). Son attribut optionnel *location* fournit un pointeur (ex. un URI : www.ibm.com) vers l'emplacement où le vocabulaire est consigné. L'élément *vocabulary-info* permet une description structurée de certaines informations sur le vocabulaire comme l'émetteur, la version ou la date.

L'élément *container* fournit une définition abstraite des données de contexte qui peuvent accompagner une requête d'accès. Ces données seront éventuellement utilisées dans l'évaluation de la condition d'une règle. Notons qu'un *container* est un élément étendant *describedObjectType*. Cela implique qu'un container peut représenter une donnée avec son identifiant. Cet élément et l'élément *epal-vocabulary* représentent un moyen

précieux d'adaptation d'EPAL à l'usage du vocabulaire ontologique d'une entreprise. On peut donner à un ***container*** un **identificateur** et un **type** correspondant à ce que ce container représente dans l'ontologie de l'entreprise.

EPAL supporte la combinaison de plusieurs politiques de contrôle d'accès. L'élément *epal-policy* permet de spécifier une condition globale d'applicabilité d'une politique de contrôle d'accès et la décision par défaut si aucune règle de la politique n'est applicable à une requête. L'attribut *globale-condition* est une expression booléenne qui détermine l'applicabilité d'une politique de contrôle d'accès à une requête. La valeur de l'attribut *default-ruling* indique la valeur à émettre par une politique au cas où l'attribut globale-condition de cette politique retourne une valeur booléenne égale à « faux ». Ainsi, un attribut *default-ruling* mis à « not-applicable » permet de passer le traitement d'une requête aux autres politiques quand la politique n'est pas applicable à une requête.

EPAL est parmi les langages de règles de contrôle d'accès qui se distinguent par la prise en considération du but de l'accès (*purpose*) selon une énumération de buts non prédéfinis. Un but ne doit pas correspondre nécessairement à un but de collecte de l'information, mais une règle peut limiter l'accès selon le but. De plus, comme point fort d'EPAL, on peut retenir l'évitement automatique de conflits de non cohérence entre les règles par l'adoption du principe de prévalence de règles selon leur ordre. On peut aussi apprécier la simplicité du mécanisme de combinaison de politiques, tout en étant conscient de ses limitations d'expressivité.

2.2.3 PDL

Policy Description Language (PDL) [16] est un langage qui fait correspondre une série d'événements à une série d'actions à exécuter par une certaine entité écoutant ces événements et assurant leur conformité à une politique. Le paradigme **événement-condition-action** représente l'essentiel de la sémantique de ce langage. Une prise en charge du temps réel et de l'historique d'occurrences de plusieurs événements est un atout caractérisant PDL. Les actions sont exprimées en termes de fonctions à n variables dont les valeurs sont données pour la séquence d'occurrences d'événements considérés.

Les conditions sont des prédicats utilisant des opérateurs de base appliqués à des constantes et à des variables, comme les attributs des événements.

PDL propose aussi une **architecture pour son implémentation**. Ce langage n'est pas dédié au contrôle d'accès. Il ne permet pas la composition de plusieurs politiques et n'offre pas de moyens de groupement ou de structuration de concepts comme celui des rôles.

La prise en charge de l'historique de certains événements est une caractéristique de PDL qui est bien utile pour spécifier, par exemple, la séparation des tâches.

Une vue d'ensemble de PDL nous permet de mettre en relief les caractéristiques suivantes :

- La caractéristique de non limitation à un domaine d'application spécifique ;
- Un **pouvoir expressif muni des types de base et des opérateurs** de base ainsi que certaines structures comme les piles et files ;
- Une sémantique exprimée sous forme de fonctions de transition ;
- Des algorithmes d'implémentation déjà spécifiés et disponibles.

Pour toutes ces caractéristiques, PDL est un candidat potentiel comme base de langage dans différents domaines et en particulier dans le domaine de la spécification de contrôle d'accès. PDL peut jouer le rôle de surveillance et de détection d'événements conflictuels pour répondre par une action de résolution de conflit. On peut envisager une spécification en PDL d'une fonction de transition qui permet d'aboutir à une décision suite à un événement représentant plusieurs décisions de contrôle d'accès résultant de l'application de plusieurs politiques de contrôle d'accès. Un ensemble de telles fonctions de transition implémente une combinaison de plusieurs de ces politiques.

2.2.4 RW

Le langage RW [17] constitue un bon exemple de langage logique qui permet d'exprimer des associations tripartites (un privilège d'accès Read ou Write, une donnée, un ensemble de sujets « agents »). Le niveau `règles' est une caractéristique claire de RW, mais il se

caractérise par l'aptitude à spécifier des politiques de contrôle d'accès qui dépendent des valeurs des données, des permissions déjà utilisées et des rôles des sujets.

RW permet de spécifier une politique de contrôle d'accès, en tenant compte de l'historique d'octroi des permissions. Ainsi, la politique de contrôle d'accès est spécifiable comme un système qui évolue en fonction d'événements d'accès à des données. RW est muni d'un algorithme de model-checking symbolique [18] qui permet d'examiner si un modèle d'une politique de contrôle d'accès peut évoluer pour arriver à un certain état ou vérifier une certaine propriété [18].

L'Outil AcPeg [19] a été développé pour implémenter ce travail de vérification et permet la représentation en modèle d'une politique de contrôle d'accès de RW. AcPeg permet aussi d'examiner si une suite d'actions légitimes d'usage de permissions selon une politique de contrôle d'accès pourra modifier l'état de son modèle pour rendre permissible une certaine action.

Par conséquent, le cadre de travail de RW permet de vérifier les effets des actions suivantes :

- Les interactions entre les règles : une utilisation d'une règle peut modifier l'application d'une autre règle ;
- La coalition entre plusieurs sujets qui peuvent utiliser des règles qui ont des interactions ;
- Les actions multiples d'un sujet utilisant plusieurs règles ayant des interactions.

À noter aussi que RW dispose d'un outil qui convertit une politique de contrôle d'accès écrite en RW, en une politique écrite dans le standard XACML.

On voit ainsi que RW est bien équipé pour supporter la séparation des tâches et d'autres principes de contrôle d'accès basés sur la limitation d'accès dépendamment de l'historique d'accès comme le modèle du Chinese Wall [13]. RW bâtit son formalisme sur la modélisation des politiques de contrôle d'accès. Ces derniers deux points représentent des zones d'intersection avec notre approche qui vise la prise en charge des principes de

contrôle d'accès au niveau des modèles de contrôle d'accès. RW est, comme Ponder, un cadre de travail intéressant; il représente un paradigme dont le point fort est son formalisme basé sur la modélisation et l'exploitation d'un outil de model-checking.

2.2.5 Un Langage avec des contraintes spatio-temporelles

Récemment, un nouveau langage [20] a été proposé avec un modèle étendant le modèle RBAC pour prendre en charge les contraintes liées au temps et à la position dans l'espace. Certains éléments de ce modèle sont associés à des intervalles de temps qui indiquent leur durée de vie ou affectent leur interprétation. D'autres associations à des lieux permettent des spécifications de contraintes sur des privilèges d'accès selon les lieux où se trouvent les sujets et/ou les objets concernés par ces privilèges.

Le modèle proposé fait aussi une distinction entre l'activation d'un rôle et l'usage de ce rôle par un sujet. Cette notion répond à certaines politiques dont l'application dépend de l'historique de l'usage de certaines permissions de rôles. De telles politiques expriment généralement des exigences du niveau modèles. Le formalisme de ce langage est **basé sur les graphes.** Il consolide notre point de vue sur l'importance de la représentation graphique du modèle du langage qu'on veut proposer, et ce comme moyen efficace d'illustration, de formalisation de la sémantique de ce langage et comme base pour son implémentation.

L'apport de ce langage réside dans sa pertinence à enrichir le modèle RBAC avec la considération des contraintes de temps et de lieu sur les privilèges d'accès selon un formalisme de modèle. Toutefois, au-delà de RBAC, ce langage est loin de permettre la spécification des ECAE au niveau des modèles.

2.3 Contrôle d'accès et ontologies

Dans le domaine du contrôle d'accès, il importe de mentionner qu'un autre axe de recherche se base sur des ontologies comme source de vocabulaire et base de modélisation convenable au traitement automatique et au formalisme. Dans [21], on procède à une modélisation des exigences de sécurité en élaborant une ontologie

catégorisant ces exigences, et ce en vue de classifier des systèmes informatiques sur la base de ces catégories d'exigences de sécurité. Un autre ouvrage important mais moins récent, est celui de [22] qui fait la lumière sur l'usage de la logique de premier ordre et de plusieurs ontologies en vue de formaliser un système judiciaire. Cet ouvrage ainsi qu'un autre plus récent [23], couvrent plusieurs approches utilisant la logique et l'intelligence artificielle. Ils peuvent être considérés comme une référence importante dont on pourra s'inspirer lors de l'élaboration d'un langage d'ECAE, car un système PDP agissant en tant que source de décisions de contrôle d'accès présente de nombreuses similarités avec un système judiciaire tel que traité dans les travaux mentionnés.

2.4 Conclusion

Dans ce chapitre, on a vu que les langages normalisés de spécification des exigences de contrôle d'accès sont conçus dans le souci de fournir un moyen d'échange des informations de contrôle d'accès et d'interopérabilité entre des systèmes traitant ces dernières. L'expressivité des ECAE au niveau de modèles et la détection des anomalies ont été visées par des langages qui ne sont pas des standards. Chacun de ces langages a un point fort pour répondre à une problématique particulière, entre autres, Ponder qui profite du paradigme de l'orienté objet pour une meilleure réutilisation et pour la définition de structures, et RW qui tient compte de l'évolution du contrôle d'accès en fonction des évènements d'octroi d'accès.

À noter que dans les chapitres présentant notre langage (Sections 6.4 et 9.8), d'autres travaux récents qui adoptent une approche comparable à la nôtre seront brièvement discutés.

En résumé, nous cherchons à définir un langage qui regroupe le maximum de points forts des langages existants, tout en gardant comme objectif principal l'expressivité des ECAE au niveau de modèles en plus d'un cadre de travail permettant la vérification de propriétés et la détection de spécifications contradictoires.

Chapitre 3 Métamodélisation et modèles de contrôle d'accès

Pour une meilleure compréhension du problème et des perspectives de solutions, on va examiner de près les ECAE. On va voir qu'on peut ranger les ECAE dans des classes, de sorte que toutes les ECAE d'une même classe font référence à un concept commun; chaque ECAE se distinguant toutefois par des valeurs particulières des attributs de ce concept. D'où la nécessité d'explorer ces concepts et leurs attributs car ils représentent un moyen de factorisation des spécifications des ECAE. Cette factorisation convient aux exigences du langage d'expression de contrôle d'accès que l'on veut développer en se basant sur la réutilisation des concepts communs capturés dans des modèles de contrôles d'accès.

3.1 Métamodèle de contrôle d'accès

Pour décrire les ECAE au niveau des modèles de contrôle d'accès, on va suivre une démarche basée sur l'identification des principes et concepts sous-jacents aux ECAE. On va procéder à une classification des ECAE sur la base de ces principes et concepts. Les ECAE qui représentent des applications d'un même principe de contrôle d'accès appartiennent à une même classe; cette classe sera spécifiée par un modèle auquel se conforment les éléments de la dite classe. Ce modèle sera appelé *un métamodèle de*

contrôle d'accès. Il consiste en une structure de concepts reliés entre eux. Il joue le rôle de *template* qui permet de décrire des ECAE par d'autres modèles. Un métamodèle de contrôle d'accès (ACMM) est donc un modèle qui permet de spécifier d'autres modèles (ses instances) d'où son nom de *métamodèle*. La section suivante explique les caractéristiques des métamodèles et des modèles qui en sont des instances.

3.2 Métamodélisation et métamodèles de contrôle d'accès

Pour représenter graphiquement (représenter par modèles) les ACMM, on va utiliser certains éléments du paradigme orienté objet qui sont les types, les attributs des types et les associations entre ces types. Ainsi, un concept sera représenté par un type (dit aussi métaclasse). La possibilité de relation entre deux concepts est représentable par une association entre leur type respectif. Pour décrire un ACMM, la notation UML [24] est suffisante pour représenter des types et leurs associations ainsi que des contraintes sur ces derniers éléments.

Dans la suite, nous allons fournir une vue d'ensemble sur la métamodélisation et comment elle permet la définition de langages pour décrire des systèmes d'une manière conforme aux travaux de normalisation du consortium Object Management Group (OMG)[5].

3.2.1 Métamodélisation

Un modèle représente un système réel d'un certain domaine. Il est basé sur la sémantique et les règles qui conditionnent les concepts du domaine. En d'autres termes, il ne doit en aucun cas briser une structure ou des contraintes choisies que les éléments du système réel respectent. En conséquence, les éléments d'un langage d'expression de modèles doivent satisfaire un ensemble de règles qui, elles-mêmes, sont décrites par un modèle qu'on convient d'appeler métamodèle. Le métamodèle reflète ainsi les règles du domaine considéré et fournit un ensemble d'éléments à instancier pour créer des modèles. Il

[5] www.omg.org

représente, de ce fait, un langage d'expression des systèmes du domaine considéré. Un exemple commenté et illustré dans les figures 3.2-1, 3.2-2 et 3.2-3 complétera l'explication des notions mentionnées dans ce paragraphe.

Selon le même principe déjà présenté, un modèle a besoin d'un langage d'expression dit métamodèle; un métamodèle est écrit dans un langage appelé métalangage et il est une instance d'un méta-métamodèle qui définit les éléments du métalangage. La Figure 3.2-1 [25] illustre le rôle de métamodèle comme une description d'un langage, et le besoin d'un métalangage pour spécifier un métamodèle.

Un langage est défini dans un métalangage, qui doit être lui-même défini dans un méta-métalangage, et ainsi de suite. Pour bien limiter notre étude des niveaux de modèles, on doit choisir un certain niveau élevé où le langage de description d'un langage se réduit à une description informelle mettant ainsi fin à l'accumulation des niveaux de la métamodéliastion. Selon la normalisation de l'OMG, quatre couches de modélisation sont suffisantes pour entièrement décrire des langages de modélisation. Ces quatre couches sont illustrées par un exemple dans la **Figure 3.2-2**, tirée des spécifications de MOF [26], et ce sont les suivantes:

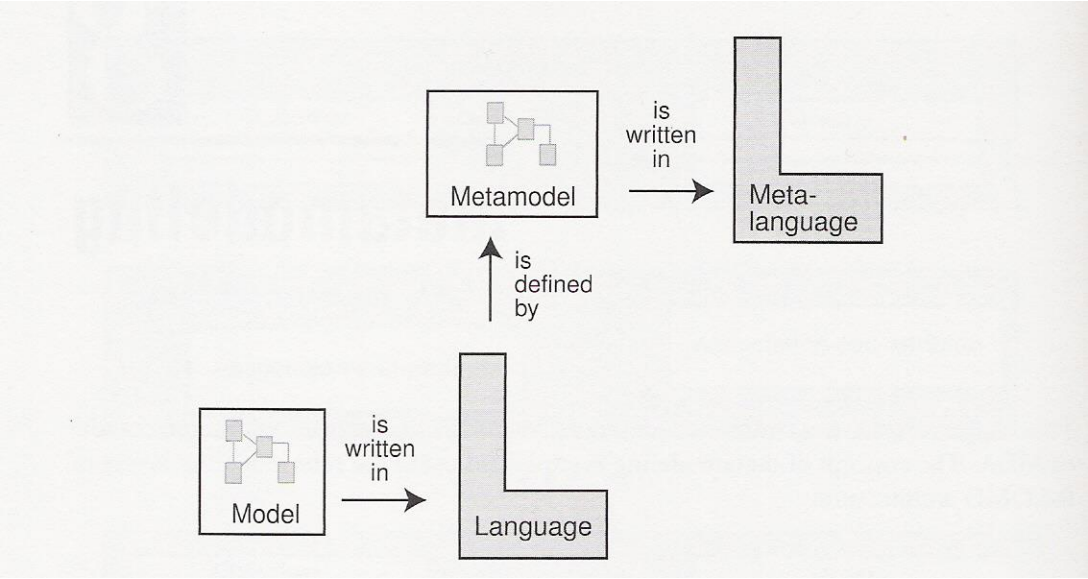

Figure 3.2-1 Modèles, langages, métamodèles et métalangages [25]

M0 c'est la couche de niveau de métamodélisation le plus bas correspondant à l'instance en cours d'exécution d'un système (*Running Application*). Elle est formée des instances d'objets en cours de traitement par le logiciel.

M1 c'est la couche du modèle renfermant la structure et le comportement du système. Par exemple, un modèle en UML d'un système particulier se situerait à ce niveau.

M2 c'est la couche du métamodèle dont les instances sont des modèles de la couche M1. Par exemple, le métamodèle UML (i.e.; le modèle du langage UML) se situe à ce niveau.

M3 c'est le niveau de métamodélisation le plus élevé, où l'instance d'un modèle de cette couche donne un modèle de la couche M2. Un modèle de niveau M3 fournit une syntaxe d'écriture de métamodèles.

L'OMG arrête cette suite au niveau 4 (celui de M3), en définissant les éléments de M3 comme instances de concepts de la même couche M3. C'est à dire que M3 est définie d'une façon auto-descriptive.

On remarque ainsi qu'un métalangage normalisé peut servir de vocabulaire commun pour manipuler des modèles écrits dans divers langages (divers métamodèles). Ceci permet à un même outil qui utilise ce métalangage normalisé d'exprimer divers métamodèles de différents domaines de l'informatique. Par exemple, un métamodèle pour les bases de données relationnelles et un autre pour les flots de données. Si, de plus, plusieurs outils utilisent le même métalangage normalisé, on obtient alors une importante interopérabilité entre ces outils qui peuvent échanger des modèles à différents niveaux.

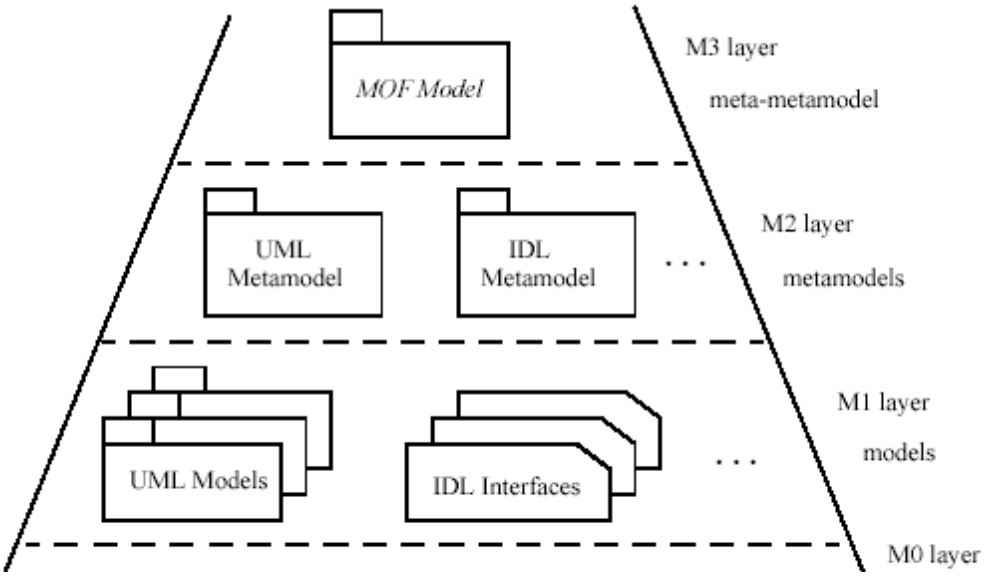

Figure 3.2-2 Le MOF à la base de plusieurs métamodèles

Dans cette optique, l'OMG a défini la spécification d'un métalangage normalisé, appelé *Meta Object Facility* (MOF)[6], qui permet de décrire toute une gamme de métamodèles d'usage courant. La figure 3.2-2, illustre comment le MOF forme un élément commun de spécification de divers métamodèles normalisés par OMG, comme UML et IDL [7]. La figure 3.2-3 illustre cette hiérarchie à travers un exemple relatif au domaine des bases de données relationnelles.

Dans la figure 3.2-3, tout élément d'une couche est une instance d'un élément de la couche directement supérieure, à l'exception des éléments de la couche M3 qui sont définis de façon auto-descriptive. Cette relation d'instance est représentée par une flèche descendante partant d'un élément vers une de ses instances dans la couche directement inférieure.

Ainsi, l'élément *Classe* de la couche M3 représente une abstraction d'un concept. Les éléments *Champ, Table et Instance* sont des concepts qui trouvent leur abstraction dans

[6] http://www.omg.org/docs/formal/02-04-03.pdf

[7] http://www.omg.org/gettingstarted/omg_idl.htm

l'élément *Classe* de M3, et sont par la suite dits des instances de Classe, comme illustré dans la figure 3.2.3, par des flèches sortant de *Classe*.

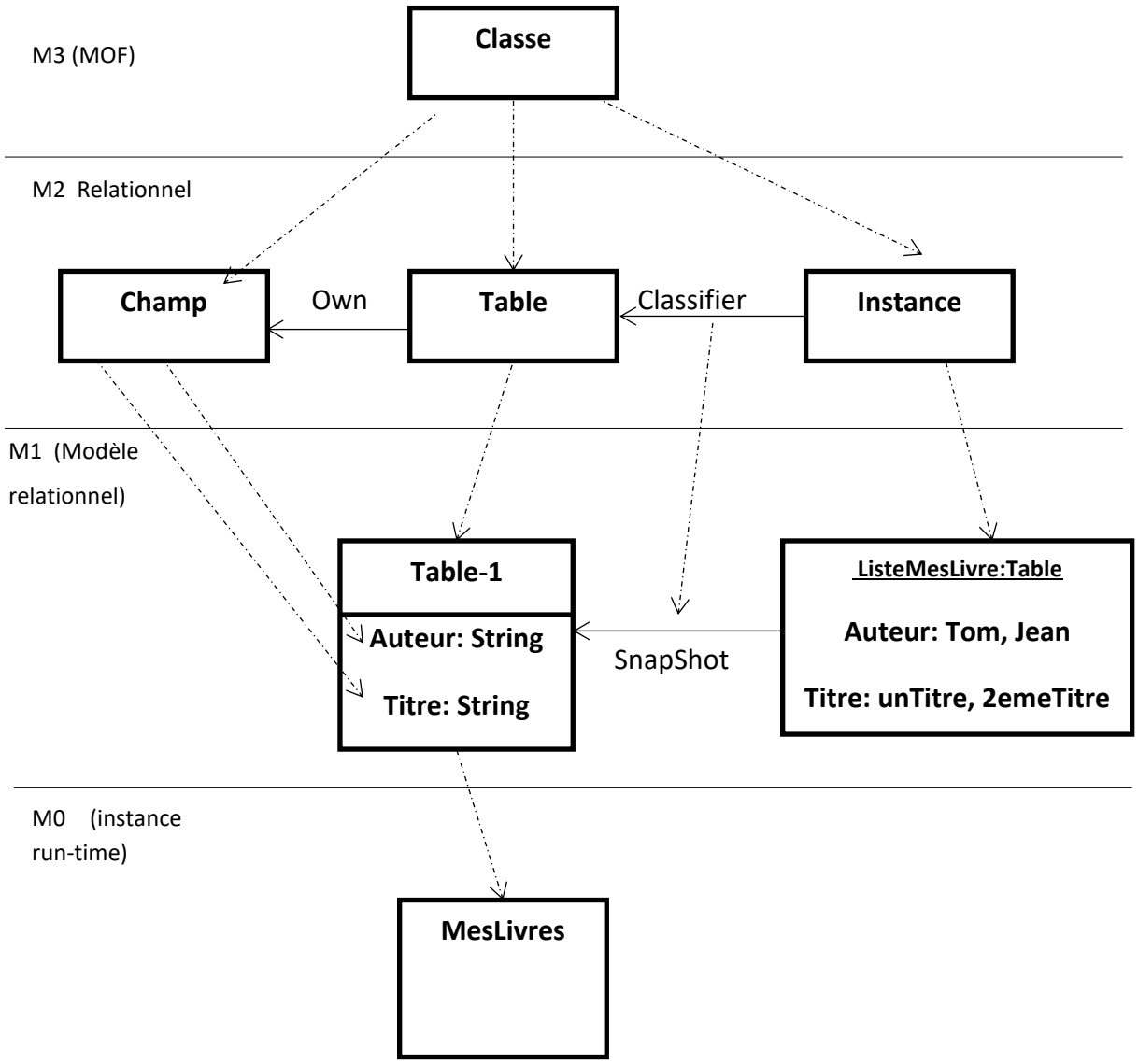

Figure 3.2-3 Exemple d'utilisation des quatre couches de métamodélisation

De même, l'élément *Table-1* représente la structure d'une liste tabulaire de livres, et est une instance de Table. Notons qu'on peut définir une autre instance de *Table* qu'on nommera, par exemple, *Table-2* et qui sera différente de *Table-1; Table-2* peut représenter, par exemple, une liste tabulaire de cours d'une université. *Table-1* et *Table-*

2 sont tous deux des éléments de la couche M1 et sont tous deux instances de Table appartenant à la couche M2.

MesLivres représentent une liste tabulaire de livres, instance de *Table-1*, car elle respecte la structure de *Table-1* comprenant deux champs, un pour les auteurs et un autre pour les titres. Notons aussi qu'on peut avoir une autre instance de *Table-1* dans M0, par exemple LivresPerdus. LivresPerdus et MesLivres appartiennent au même niveau de métamodélisation M0 et sont deux instances de l'élément Table-1 de M1.

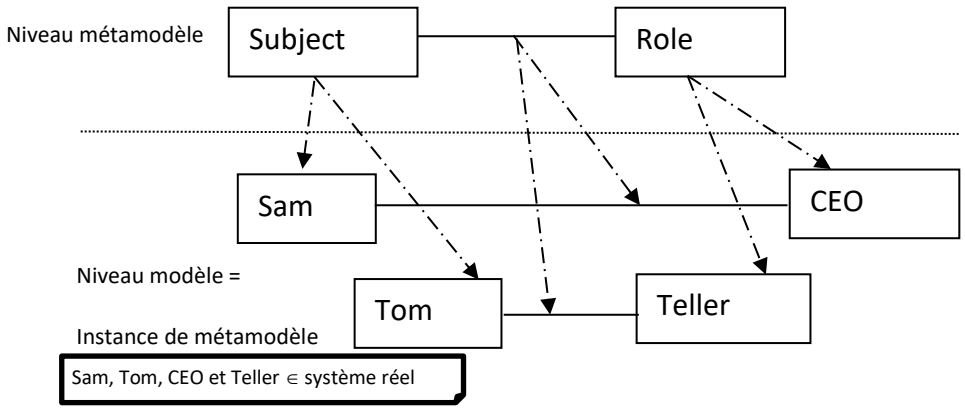

Figure 3.2-4 Les niveaux de métamodélisation

Ainsi, selon le paradigme de la métamodélisation [25], un modèle peut avoir de nombreuses instances à un niveau de métamodélisation inférieur; chacune spécifiant une application particulière (utilisation) de ce modèle. La Figure 3.2-4 fournit une autre illustration des niveaux de métamodélisation, mais cette fois-ci, avec une application au domaine du contrôle d'accès. Dans cette figure, on trouve le métamodèle et de son instance qui est un modèle de niveau de métamodélisation inférieur. Ce métamodèle simple représente le principe qu'un sujet doit être associé à un rôle. Il se compose des éléments de métamodélisation : Subject, Role et leurs associations. À un niveau inférieur, le modèle, instance du métamodèle, est composé de Sam et Tom en tant qu'instances de l'élément Subject, et CEO et Teller en tant qu'instances de l'élément Role. Les flèches en pointillés indiquent l'instanciation; elles pointent vers les instances. En outre, les

associations entre Sam et Teller, et CEO et Tom sont des instances de l'association entre leurs éléments de métamodélisation Subject et Role.

Pour récapituler, on a vu comment un métamodèle joue le rôle d'un langage par ses éléments et leur structure, offrant ainsi un vocabulaire et une grammaire. Une instance d'un métamodèle représente un énoncé du langage défini par le métamodèle. En effet, cette instance utilise le vocabulaire et la grammaire fournis par le métamodèle. Ceci nous permet de considérer un métamodèle comme une description d'un langage, et tout modèle instance de ce métamodèle comme un énoncé bien formé de ce langage. Le modèle concret du langage est implicitement défini dans le métamodèle. Dans la suite de ce document, on ne va plus faire de distinction entre le métamodèle et le langage qu'il définit.

Vu la possibilité d'illustration graphique offerte par la modélisation qui permet de visualiser les concepts et leurs relations et qui convient à la spécification des métamodèles de contrôle d'accès, on a choisi de spécifier notre langage MACL sous forme d'un métamodèle qui se base sur un ensemble de métamodèles de contrôle d'accès.

3.3 Identification des métamodèles de contrôle d'accès

Les ECAE d'une entreprise spécifient quand est-ce qu'un accès doit être autorisé ou refusé. Normalement elles implémentent des principes plus généraux provenant de la loi, de règlements ou de normes. Elles sont généralement décrites dans un langage naturel; elles peuvent faire appel à diverses structures de concepts qu'on va essayer de représenter sous forme de modèles. C'est à travers la révision des documents informels de spécifications de contrôle d'accès qu'on identifie les métamodèles de contrôle d'accès. L'application du principe de contrôle d'accès sous-jacent d'un métamodèle de contrôle d'accès résulte en une instance de ce métamodèle. Cette instance du métamodèle est une spécification de contrôle d'accès permettant d'émettre une décision en réponse à une requête d'accès. Ainsi, les métamodèles de contrôle d'accès qu'on cherche à

identifier, sont celles dont les instances sont des systèmes de décision. Dans cette section on va présenter les métamodèles de contrôle d'accès qu'on a identifiés et qui sont généralement présents dans la littérature du domaine de contrôle d'accès.

3.3.1 Group2Group

L'expression des exigences de contrôle d'accès au niveau des groupes de sujets et d'objets permet d'étendre la spécification des ECAE à plusieurs sujets et objets à la fois (scalabilité). Ainsi, le regroupement de l'ensemble des objets en sous-ensembles d'objets (non nécessairement disjoints) selon les besoins de contrôle d'accès qu'ils partagent, est un moyen de factorisation facilitant l'expression de ces besoins. De même, le regroupement de sujets selon les privilèges qu'ils partagent facilite l'expression de ces privilèges avec moins de répétition.

La spécification de certaines ECAE peut se réduire à donner à un ensemble de sujets des privilèges d'accès sur un ensemble d'objets; ceci se concrétise par une association entre cet ensemble de sujets et leurs privilèges. Plusieurs associations similaires servent à spécifier plusieurs ECAE, toujours reliant des groupes de sujets à des privilèges d'accès sur des groupes d'objets. On exploite tout simplement le fait que pour certaines ECAE, la mise en facteur des spécifications permet une expression plus aisée, et ce en manipulant des groupes plutôt que des individus. C'est pour cela qu'on va définir le métamodèle de contrôle d'accès Group2Group (G2G) comme base pour la construction d'autres métamodèles de contrôle d'accès. Ces autres métamodèles exigeront des regroupements de sujets et d'objets selon certains critères, comme la valeur d'un attribut ou tout autre raison propre à chacun d'eux.

Explication du métamodèle de contrôle d'accès Group2Group :

Dans le diagramme de la Figure 3.3-1, les objets et les sujets sont respectivement représentés par les deux éléments *Object* et *Subject*. Leurs groupes sont respectivement représentés par les éléments *ObjectsGroup* et *SubjectsGroup*.

L'élément *AbstractObject* représente un objet ou un groupe d'objets. De même, *AbstractSubject* représente un sujet ou un groupe de sujets. Un objet individuel peut être ciblé par un privilège d'accès, comme un groupe d'objets peut l'être. De même, un sujet peut avoir un privilège d'accès, comme un groupe peut l'avoir. Nous introduisons les éléments *AbstractObject* et *AbstractSubject* pour fournir un moyen de spécification de concepts qui s'applique de la même façon à un individu ou un groupe.

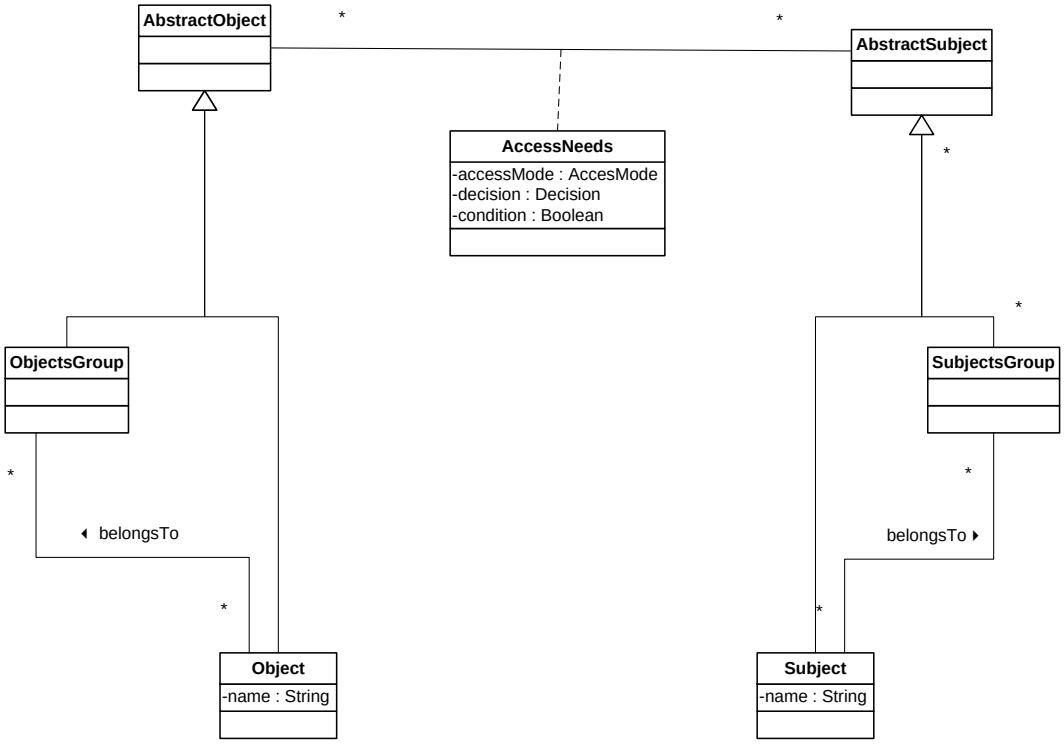

Figure 3.3-1 Métamodèle de contrôle d'accès Group2Group

L'élément *AccessNeeds* relie, indifféremment, un groupe de sujets ou un sujet à un groupe d'objets ou à un objet, et il représente les privilèges d'accès qui sont accordés aux membres du groupe de sujets relativement aux objets du groupe d'objets.

Les attributs d'*AccessNeeds* sont :

- AccessMode : sa valeur est un élément de l'énumération « Read », « Write », « Execute ».

- Decision : sa valeur est un élément de l'énumération « Permit », « Deny », « NotApplicable », et « Indeterminate ».
- Condition : cet attribut exprime la condition à satisfaire pour fournir la décision spécifiée par l'attribut Decision.

À noter que dans le modèle G2G, on n'a pas spécifié les raisons (rôle, confiance, etc.) de la création de groupes de sujets ou d'objets. Ces raisons diffèrent selon les buts et la structure d'une entreprise. Dans la suite de cette section, les métamodèles de contrôle d'accès de rôles et de *Need-to-know* seront expliqués comme étant deux modèles de contrôle d'accès utilisant les concepts de G2G. De plus, G2G est le métamodèle de contrôle d'accès dont la logique est la plus simple; elle consiste en une détermination de la décision de contrôle d'accès en fonction de l'identification d'une éventuelle instance de AccessNeeds reliant les instances de Subject et Object d'une demande d'accès. Cependant, G2G n'est pas le noyau de tout autre métamodèle de contrôle d'accès, mais les autres métamodèles pourront réutiliser des éléments définis dans G2G comme AccessNeeds, Object et Subject.

Contrôle d'accès basé sur les rôles

Le métamodèle de contrôle d'accès le plus connu qui réutilise des concepts de G2G, est le modèle de rôles. En effet, dans certaines entreprises, on groupe les sujets selon leurs fonctions et les postes qu'ils occupent ce qui permet de définir les différents rôles joués par les sujets employés dans l'entreprise. D'où l'idée de spécifier les droits d'accès en fonction de ces rôles, car pour certaines entreprises on peut adopter le principe que les employés accomplissant les mêmes tâches ont besoin des mêmes privilèges d'accès. Ainsi, un employé sera autorisé à accéder à un objet si son rôle lui donne ce droit d'accès. On définit aussi une relation hiérarchique entre les rôles, qui exprime le fait que tout rôle r1 supérieur, selon la hiérarchie des rôles, à un autre rôle r2, aura pour acquis tous les privilèges d'accès de r2; r2 est dit un sous rôle de r1.

Avec le groupement des sujets selon les rôles qu'ils occupent et la structure hiérarchique des rôles, la gestion du contrôle d'accès sera plus simple. On aura moins de répétition en

spécifiant, une fois pour toutes, les droits d'accès des sujets ayant le même rôle. On évite aussi la répétition au niveau de la définition d'un rôle hiérarchiquement supérieur à des rôles déjà définis; il suffit de spécifier les privilèges d'accès qui distinguent le rôle de ceux qui lui sont inférieurs dans la hiérarchie.

Comme exemple de contrôle d'accès basé sur les rôles, on peut considérer dans une université les privilèges d'accès des techniciens de gestion d'un département et de leurs assistants, aux données des programmes de l'université. Le rôle d'assistant donne comme privilège de voir les cours offerts et le nombre d'étudiants inscrits par cours et par programme, alors qu'un technicien doit avoir d'autres privilèges comme la possibilité d'ajouter un cours à un programme ou d'inscrire un étudiant. Pour spécifier ces ECAE, il suffit de spécifier les privilèges du rôle *Assistant et* d'affecter ce rôle à tous les assistants, puis de définir le rôle *Technicien* comme rôle ayant *Assistant* comme sous rôle, et d'ajouter au rôle *Technicien* les privilèges qui lui sont propres. De même, le rôle de directeur de département est supérieur au rôle de technicien; il a donc les privilèges propres à sa fonction et on spécifiera le rôle de *Technicien* comme sous rôle du rôle de directeur de département.

Une définition de plusieurs variantes de modèles représentant le contrôle d'accès basé sur les rôles (RBAC) a été développée en 1996 et a pris le nom de RBAC96 [9]. L'introduction d'un modèle comprenant les rôles administratifs qui permettent de manipuler, définir et gérer les rôles a été ensuite défini sous le nom de ARBAC97 [2]. Tout ce travail a contribué à la définition du standard de l'ANSI-RBAC [1].

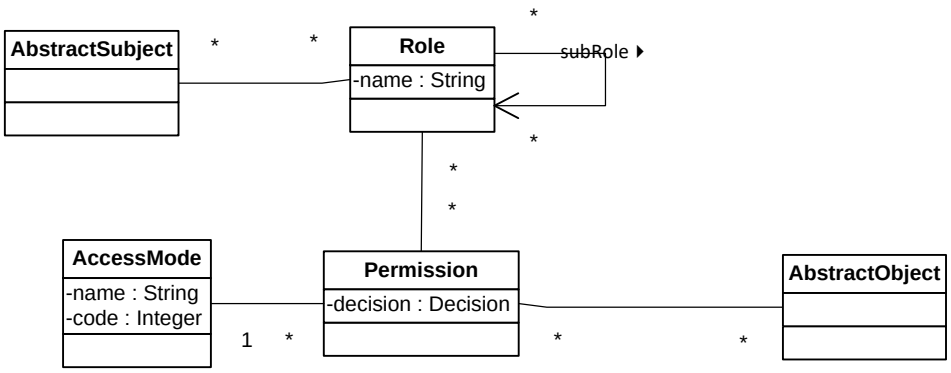

Figure 3.3-2 Métamodèle de contrôle d'accès RBAC

Pour décrire RBAC, nous avons élaboré le métamodèle montré dans les Figure 3.3-2 et Figure 6.2-6. Les éléments spécifiques de cet ACMM sont :

AccessMode : Permet de spécifier les différents modes d'accès. Dans le cadre de notre travail, nous nous limiterons généralement aux instances de AccessMode représentant les actions suivantes : Read, Write et Execute.

Role : cet élément représente le concept de poste dans une organisation.

Permission : Cet élément représente les privilèges d'accès associés à un rôle. Lorsqu'une instance **p** de Permission est associée à une instance **r** de Role, cela signifie que le rôle **r** a la permission **p** d'accéder aux objets liés à **p** avec les modes d'accès indiqués par les instances de AccessMode associées à **p**. Notons ici qu'une instance de Permission reliée à un sujet ne peut pas être associée à plus qu'un mode d'accès quand elle est associée à plusieurs instances de AbstractObject, car ceci porte à confusion. Il faut alors créer, pour chaque couple (sujet, mode d'accès), une nouvelle instance de Permission qui peut être reliée à plusieurs instances de AbstractObject.

Associations de rôle à lui-même et à Subject : l'association réflexive « subRole » de l'élément Role, permet la spécification de la hiérarchie des rôles. L'association entre Subject et Role permet de spécifier qu'un sujet (instance de Subject) qui est dans un rôle particulier jouit des permissions associées à l'instance de Role représentant son rôle.

Compartiments de Need-to-know

Un bon principe de contrôle d'accès c'est de ne donner à chaque sujet que le minimum de privilèges d'accès nécessaires à l'accomplissement de ses fonctions. Ce principe

interdit l'accès à tout objet qui sort de la zone d'intérêt ou des besoins nécessaires au bon fonctionnement d'un sujet. Ce principe sera dit le *principe de Need-to-know*. Pour un sujet donné, les objets formant sa zone d'intérêt ou qui sont nécessaires à son bon fonctionnement forment ce qu'on appelle son *scope*.

Selon le principe de *Need-to-know*, un sujet travaillant dans l'administration d'un département d'une université, ne doit pas accéder aux informations administratives propres aux autres départements. Même s'il jouit d'un rôle au sommet de la hiérarchie des rôles de son département (par exemple directeur), les informations des autres départements sortent de sa zone d'intérêt et du cadre des besoins inhérents à son mandat.

En termes de modélisation, ce métamodèle de contrôle d'accès requiert l'introduction d'un nouvel élément qu'on appelle *compartiment (Compartment)*. Un *compartiment* représente un regroupement d'objets formé pour exprimer l'application du principe de *Need-to-know*. Les compartiments sont définis pour spécifier les *scopes* des sujets. Pour spécifier le scope d'un sujet, on va associer ce sujet aux compartiments dont l'union forme ce scope. Remarquons qu'un groupe d'objets peut intéresser deux sujets qui ont deux mandats différents et par la suite deux *scopes* différents. Ceci veut dire qu'il est possible qu'un *compartiment* contribue à la définition de deux scopes différents, et par la suite ce compartiment sera associé à deux sujets différents.

Notons ici qu'on pourrait exprimer le principe de *Need-to-know* en utilisant le métamodèle de contrôle d'accès Group2Group et en définissant des groupes de sujets et d'objets appropriés. Mais l'introduction de l'élément *compartiment* permet un meilleur rendu du principe de *Need-to-know* qui pourrait être appliqué conjointement avec d'autres principes de contrôle d'accès, comme celui des rôles. Dans ce cas, un accès sera autorisé si le rôle le permet et si tout compartiment associé à l'objet à accéder est associé au sujet demandant l'accès.

3.3.2 Chinese Wall

Une entreprise peut avoir plusieurs clients qui sont en compétition dans leurs activités, telles les compagnies d'assurances clientes d'une même banque. Ces compagnies forment ce qu'on appelle un groupe de conflit. L'accès par un employé aux informations critiques de plus qu'un membre de ce groupe de conflit, lui permettrait de divulguer des informations critiques sur l'état financier d'un membre de ce groupe à un de ses compétiteurs du même groupe.

Dans une telle situation le privacy officer de l'entreprise doit exiger qu'aucun employé n'ait la possibilité de jouer un tel rôle injuste à l'égard des clients en conflit d'intérêt. Il en résulte une ECAE qui est décrite par un métamodèle de contrôle d'accès qu'on appelle Brewer et Nash ou Chinese Wall (CW) [13]. CW a été introduit par ses auteurs comme consistant en deux règles : (1) la règle simple et (2) la règle de la propriété *. Dans cette section, nous procédons de même en introduisant tout d'abord ce métamodèle en considérant seulement la règle simple de CW, pour expliquer ensuite la propriété *.

Chinese Wall : règle simple

Pour spécifier une ECAE de type Chinese Wall, on commence par l'identification des groupes de conflit, ensuite on regroupe les données (objets) de chaque élément d'un groupe de conflit dans un ensemble qu'on appelle classe du groupe de conflit.

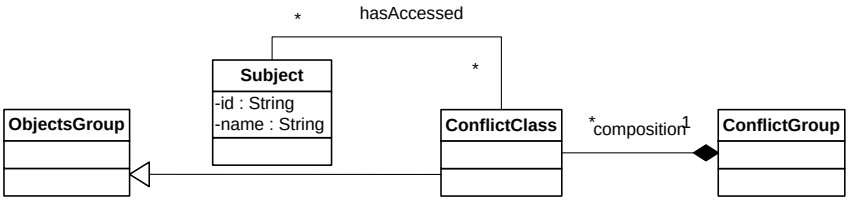

Figure 3.3-3 Métamodèle du Chinese Wall

La Figure 3.3-3 illustre le métamodèle de contrôle d'accès Chinese Wall. Ce métamodèle capture les éléments ConflictGroup et ConflictClass qui représentent les groupes de conflit et les classes des groupes de conflit respectivement. Les éléments Subject et ObjectsGroup sont déjà introduits dans la Section 3.3.1. Les associations montrées et leurs multiplicités représentent les faits notables : plusieurs groupes de conflit peuvent

coexister dans le même modèle CW (instance du même métamodèle). Une classe de conflit appartient à un groupe de conflit unique. L'association hasAccessed permet de représenter l'éventuel fait qu'un sujet (instance de Subject) ait déjà accédé aux éléments d'une classe d'un groupe de conflit (instance de ConflictClass).

Dans notre exemple de compagnies d'assurances clientes d'une banque, l'ensemble des compagnies d'assurances qui sont en compétition constitue un groupe de conflit (ConflictGroup dans la Figure 3.3-3); et pour chaque compagnie de ce groupe de conflit, l'ensemble des objets de cette compagnie constitue une classe du groupe de conflit (ConflictClass dans la Figure 3.3-3). Tout employé de la banque en charge des dossiers de ces compagnies est un sujet (Subject dans la Figure 3.3-3). L'ECAE de type Chinese Wall sera exprimée par l'énoncé suivant : Un sujet ayant déjà accédé à un objet d'une classe d'un groupe de conflit n'est autorisé à accéder à aucun objet appartenant à une autre classe du même groupe de conflit. Ainsi, un sujet ne pourra jamais accéder à des informations appartenant à deux classes distinctes d'un même groupe de conflit.

Considérons, par ailleurs, le cas d'une grande compagnie d'assurance comme Promutuel qui a généralement une activité importante dans le marché immobilier. Elle sera, de ce fait, en conflit d'intérêt avec les autres acteurs du marché immobilier. Mais Promutuel est aussi une compagnie d'assurance. Il en résulte que Promutuel appartiendra à deux groupes de conflit, celui des compagnies d'assurances GCinsurance et celui des acteurs du marché immobilier GCrealEstate. Les objets de Promutuel, comme l'état financier, les comptes des clients et les actifs en biens immobiliers, formeront une classe dans GCinsurance. Ces mêmes objets ou certains d'entre eux formeront une classe d'objets dans le groupe de conflit GCrealEstate. Ce dernier exemple montre clairement qu'un même objet peut appartenir à plusieurs classes dans divers groupes de conflit. Cette vue étendue de Chinese Wall est dite Aggressive Chinese Wall [27].

Selon le modèle de la Figure 3.3-3, l'accès, par un sujet, à un objet d'une classe de conflit doit priver ce sujet de tout accès à tout objet appartenant aux autres classes de conflit du même groupe de conflit. Toutefois, l'accès à un objet d'une classe de conflit, n'empêche

pas l'accès à un autre objet d'une autre classe d'un autre groupe de conflit, distinct du premier. Comme montré sur la Figure 3.3-3, l'élément *Subject* est associé à l'élément *ConflictClass* avec une multiplicité 0 ou plus (représentée par *), pour indiquer qu'un sujet peut accéder à plusieurs objets appartenant à différentes classes de conflit, du moment que ces dernières appartiennent à des groupes de conflits distincts. Cette multiplicité se réduira à zéro ou à 1 dans le cas particulier où il n'y a qu'un seul groupe de conflit.

Bref, la logique de contrôle d'accès de la règle simple de CW peut être formulée comme suit :

Considérons les classes C_1, C_2, …, C_n d'un groupe de conflit CG. Si un sujet s a accédé à n'importe quel objet appartenant à une classe C_i de CG, s ne pourra plus accéder à tout objet appartenant à une autre classe C_k de CG, avec $k \neq i$. En d'autres termes, (AccessedBy-s $\cap$ ($C_1 \cup C_2 \ldots \cup C_n$) $\subseteq C_x$) doit rester *vrai* pour n'importe quel sujet s, où AccessedBy-s désigne l'ensemble des objets accédés par s et Cx désigne une classe du groupe de conflit CG.

Chinese Wall : la propriété *

En plus des exigences de la règle simple de CW, ce métamodèle comprend d'autres exigences qui considèrent le flux d'informations sortant des classes de conflit. Ces exigences additionnelles sont désignées par exigences de la propriété *.

Les exigences de la propriété * visent à empêcher qu'un sujet puisse violer les exigences de la règle simple du CW en deux étapes : (1) en copiant des informations d'une classe de conflit pour ensuite (2) les écrire dans un (ou des) objet(s) accessible(s) par d'autres sujets. Concrètement, considérons les deux dossiers de compagnies d'assurance Promutuel et SunLife comme deux classes de conflits C1 et C2 du même groupe de conflit. Si un sujet Sx accède à C1 et copie son contenu dans un dossier D accessible par un autre sujet Sy qui n'a pas accédé à C1, alors Sy pourrait accéder à C2 selon les exigences simples du CW. Sy pourrait ensuite accéder au dossier D comprenant du contenu propre à C1. En conséquence, Sy détiendrait des informations provenant des deux classes de conflits C1

et C2; il aura accédé à C1 d'une façon indirecte (par l'entremise des actions de Sx) appelée aussi *transitive* dans la littérature sur CW.

Les exigences de la propriété * peuvent être décrites comme suit :

Le flux des informations doit être confiné dans chaque classe de conflit. Ainsi, dès qu'un sujet accède à une classe, il n'aura plus la possibilité d'écrire que dans cette même classe.

3.3.3 Dynamic Chinese Wall

Le Dynamic Chinese Wall (DCW) [28] est une nouvelle variante du CW. Selon DCW, on tient compte du fait qu'un sujet qui accède à des classes de différents groupes de conflit permettrait un certain flot de données entre deux classes C1 et C2 appartenant respectivement aux différents groupes de conflits GC1 et GC2. Ainsi, les deux classes de deux groupes de conflits distincts pourraient contenir certaines informations conflictuelles qui appartenaient à l'un de ces deux groupes de conflit.

En se basant sur l'exemple du CW de la section précédente, on va présenter un cas justifiant le DCW. Ainsi, supposons qu'un employé de la banque accède à des informations critiques de Promutuel contenues dans une classe de conflit ClassPromutuel et les copie ensuite dans un document de la compagnie PetroCanada contenu dans la classe de conflit ClassPetroCanada. ClassPetroCanada est une classe d'un autre groupe de conflit du domaine pétrolier qu'on appelle GCoil. ClassPetroCanada du GCoil contient maintenant des informations critiques sur Promutuel. Il faut alors que ClassPetroCanada appartienne maintenant au groupe de conflit du domaine de l'assurance GCinsurance et au groupe de conflit GCoil en même temps.

Pour spécifier la logique de décision du DCW, on va introduire certains termes et notations. Par abus de langage on dit qu'un sujet S a accédé à un groupe de conflit GC si le sujet S a accédé à un objet d'une classe du groupe de conflit GC. De plus, l'ensemble des groupes de conflit déjà accédés par le sujet S sera noté GC-S. Ceci étant, la logique de décision de contrôle d'accès selon le DCW est la même que celle de CW avec la prise en charge de l'aspect dynamique décrit par :

Chaque fois qu'un sujet S écrit dans un objet d'une classe X appartenant à un groupe de conflit, cette classe X appartiendra à tous les groupes de conflit déjà accédés par le sujet S formant l'ensemble GC-S.

La logique de décision du CW et du DCW sera formellement spécifiée dans la Section 7.2.2.

3.3.4 Sécurité multi-niveaux

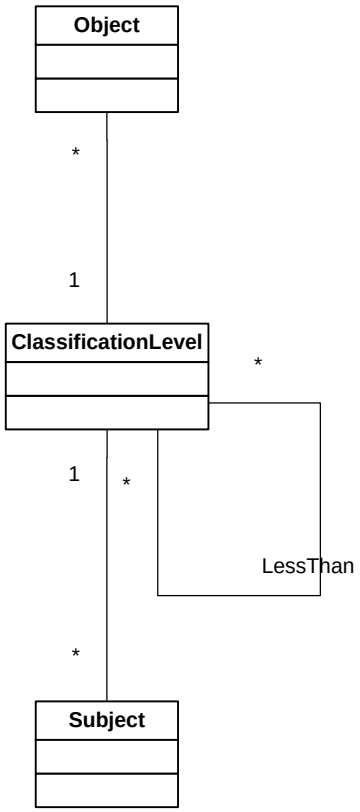

Figure 3.3-4 Métamodèle commun aux métamodèles de contrôle d'accès multi-niveaux

Les métamodèles de contrôle d'accès dits multi-niveaux sont basés sur l'affectation de différents niveaux aux sujets selon certains critères comme la confiance en ces sujets, et de différents niveaux aux objets selon certains critères comme la sensibilité de l'objet. Ces niveaux sont dits *des niveaux de classification (ClassificationLevel).* La comparaison entre le niveau d'un sujet et celui d'un objet détermine ou contribue à déterminer les privilèges d'accès du sujet à l'objet en question. L'aboutissement à une décision de contrôle d'accès à partir de la comparaison des dits niveaux peut suivre différentes

logiques, dépendamment des besoins de contrôle d'accès, telles que la préservation de l'intégrité ou la confidentialité. Chacune de ces logiques peut être décrite par un métamodèle de contrôle d'accès multi-niveaux. Dans le texte qui suit, nous allons examiner différentes variantes du métamodèle de contrôle d'accès multi-niveaux.

Bell-LaPadula

Dans certains organismes, on convient de classifier les objets selon leur sensibilité. Cette sensibilité est à évaluer en fonction de l'importance du préjudice qui résulterait de l'utilisation inappropriée de ces objets. Cette classification permet d'attribuer à tout objet un niveau qu'on convient d'appeler *niveau de sensibilité (NS)* ou « Sensitivity » de l'objet. De leur côté, les sujets sont classifiés en différents niveaux en fonction de la confiance accordée à chaque sujet. Ces niveaux sont dits les *niveaux d'habilitation de sécurité* (NHS) ou «Clearance». Généralement, le NHS d'un sujet est déterminé par une tierce partie, comme une autorité gouvernementale qui émet des cotes de sécurité sur demande pour les sujets souhaitant travailler dans certains ministères. Dans certains cas, le NHS peut être déterminé de façon subjective; par exemple, un directeur de société peut avoir différents niveaux de confiance en ses subalternes et il peut décider de les doter de privilèges d'accès en conséquence. De plus, pour chaque sujet, le contrôle d'accès peut prendre en considération les informations dont le sujet a besoin pour accomplir ses tâches. Un sujet aura la possibilité d'accéder à un objet si son NHS est supérieur ou égal au NS de l'objet en question et si cet objet appartient à certaines catégories d'objets spécifiées accessibles au sujet. Ce métamodèle de contrôle d'accès porte le nom de Bell-LaPadula (BLP) [3]. Il a été adopté initialement dans le domaine militaire, mais il est aussi utile pour représenter des besoins de contrôle d'accès dans d'autres domaines. Dans ce modèle, on utilise, en plus des niveaux de confiance, une catégorisation des objets. Cette catégorisation peut se faire selon l'appartenance des objets à des projets ou à des départements d'une entreprise. Les catégories d'objets accessibles par un sujet seront déterminées selon le principe qui consiste à ne fournir que l'information nécessaire pour remplir une fonction (*Need-to-know*).

Le modèle Bell-LaPadula se base sur les définitions suivantes :

- S : représente l'ensemble des sujets
- O : représente l'ensemble des objets
- Cat : représente l'ensemble de catégories d'objets
- Cl : représente l'ensemble ordonné des niveaux comprenant les NHS et les NS
- m : représente le mode d'accès et peut prendre l'une des valeurs suivantes :
    - r pour read
    - a pour append
    - w pour write
    - x pour execute
- M : représente une matrice de discrétion dont chaque élément M_{ij} prend une valeur m indiquant le mode d'accès permissible pour le sujet i par rapport à l'objet j.

Le niveau de classification d'un objet i noté $L(O_i)$ ou le niveau de classification d'un sujet j noté $L(S_j)$, est défini par un couple (nhs-ns, ss-Cat)[8] où : nhs-ns est un élément de Cl et ss-Cat est un sous-ensemble de Cat; ss-Cat appartient à l'ensemble représentant les parties de Cat et qu'on note *P*(Cat). Ainsi, un niveau de classification est un élément du produit cartésien Cl × *P*(Cat).

On définit une relation **d'ordre partiel**[9] notée ≤ sur l'ensemble Cl × *P*(Cat) ; cette relation est définie par le préordre :

(Cl × P(Cat), ≤)

La relation d'ordre permet de comparer (partiellement) les niveaux de classification des sujets à ceux des objets, et ce en vue de permettre une décision de contrôle d'accès.

Les conditions de contrôle d'accès selon ce modèle se formulent comme suit:

[8] L'ordre dans ce couple n'a pas d'importance, on considère la combinaison d'un niveau de clearance et d'un ensemble de catégories, mais on a opté pour la notation de couple. Remarquons que les deux éléments du couple appartiennent à deux ensembles différents.

[9] Relation d'ordre partiel : Réflexive, antisymétrique et transitive. Possibilité d'éléments non comparables.

1. **No read-write up** : un sujet ne peut effectuer un write ou un read sur un objet que si son niveau est supérieur ou égal à celui de l'objet.
2. **No copy paste downward** : Cette condition interdit tout flux d'information d'un niveau vers un niveau inférieur. Selon cette condition, si un sujet accède simultanément à un objet O_k avec un mode d'accès m1 et à un objet O_l avec un mode d'accès m2, alors :

   si m1 $\in$ {w, a} et m2 $\in${r, w} alors $L(O_l) \leq L(O_k)$.

   Ainsi lors d'un accès en mode « écriture » à un objet d'un certain niveau de classification, on ne peut pas permettre une vision simultanée sur un autre objet d'un niveau de classification plus élevé. Notons que le modèle permet de spécifier un ensemble de sujets de confiance exceptionnelle qui sont exemptés de se conformer à la présente règle.
3. La matrice de discrétion M donne les conditions nécessaires d'autorisation de tout mode d'accès d'un sujet S_i à un objet O_i :

   Pour que S_i puisse accéder à O_i selon un mode m, il faut que m ne contredise pas l'élément M_{ij} de la matrice M.

Le métamodèle de contrôle d'accès de la Figure 3.3-5 comprend les éléments Clearance-Sensitivity, Compartment et ClassificationLevel pour représenter respectivement les NHS et les NS, les catégories d'objets et les niveaux de classification des sujets et des objets. Une instance de ClassificationLevel représente un niveau de classification, et elle est définie pour un sujet (ou un objet) à partir d'un niveau NHS (ou NS) et d'un ensemble de catégories (Compartments). Remarquons qu'un sujet, instance de Subject, participe aux deux associations ; la première le relie à un élément de type Clearance-Sensitivity (son NHS); la deuxième le relie à plusieurs instances de Compartment. Ceci correspond à la spécification du couple (nhs-ns, ss-Cat), déjà mentionné, qui définit le niveau de sécurité d'un sujet.

Toujours selon le métamodèle, les multiplicités des associations montrent bien l'unicité de nhs-ns et la multitude de catégories pour un sujet ou un objet. L'attribut index de

l'élément Clearance-Sensitivity indique que les instances de cet élément forment une liste ordonnée selon la valeur de cet attribut.

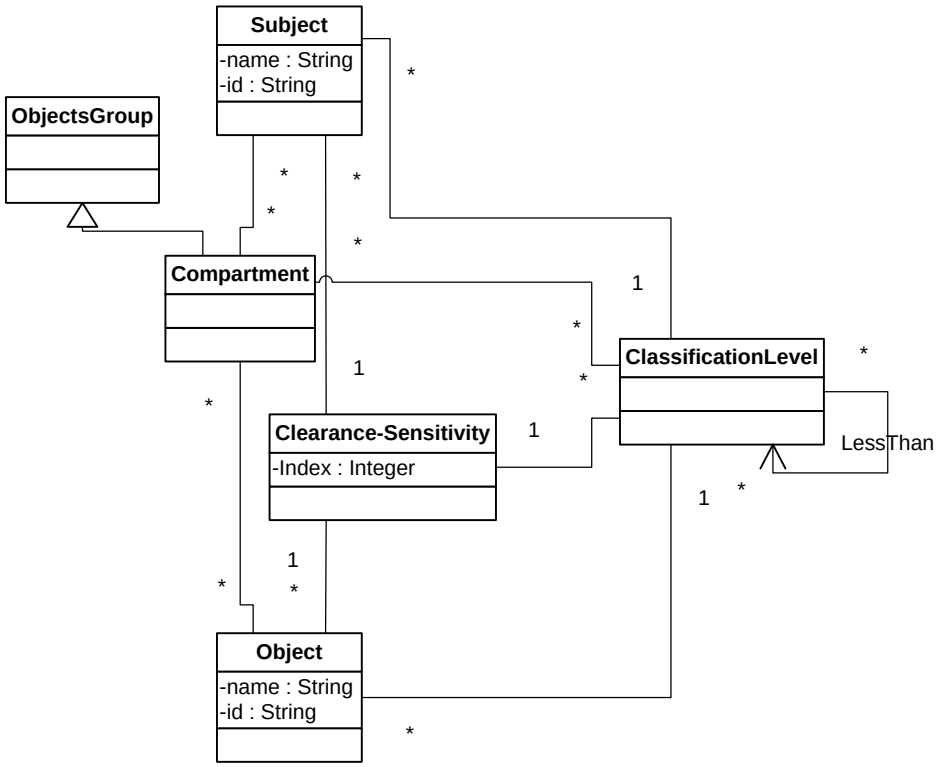

Figure 3.3-5 Métamodèle de contrôle d'accès multi-niveaux avec considération des compartiments et des clearances comme Bell-LaPadula

Le métamodèle de la Figure 3.3-5 décrivant le contrôle d'accès multi-niveaux basés sur les compartiments, les NHS et les NS, comprend tous les éléments du modèle décrivant les métamodèles de contrôle d'accès multi-niveaux, sans le contredire. Mais, de plus, le présent modèle enrichit le précédent avec de nouveaux éléments et associations spécifiques à Bell-LaPadula. Ainsi, l'élément ClassificationLevel est associé aux nouveaux éléments Compartment et Clearance-Sensitivity. Ceci est dû au fait que le niveau de sécurité selon Bell-LaPadula est défini en termes de ces nouveaux éléments. La Figure 3.3-5 représente les éléments de base du métamodèle de contrôle d'accès Bel-Lapadula qui comprend aussi une logique (no read-write up, no copy paste downward) permettant des décisions de contrôle d'accès qui se basent sur la matrice de discrétion M ainsi que les instances des éléments de la figure.

Il importe de noter que nous avons considéré le métamodèle de contrôle d'accès de Bell-LaPadula dans le cas le plus général, alors qu'une variante plus simple et bien connue de ce métamodèle se contente de considérer les niveaux de classification sans définir des compartiments d'objets. Les niveaux de classification sont alors totalement ordonnés et la décision de contrôle d'accès se base seulement sur la comparaison des niveaux de classification des objets à ceux des sujets.

Biba

Le modèle de Biba [29] est basé sur l'attribution de niveaux de classification aux sujets et aux objets d'une façon similaire à celle expliquée dans le modèle précédent. Contrairement au modèle précédent basé sur des niveaux de confidentialité, le principe d'attribution de niveaux selon le modèle Biba se base sur **l'intégrité** des sujets et des objets. Biba se base sur cette classification en niveaux pour établir des règles de contrôle de **flux d'information**. Ces règles sont les suivantes:

- Un sujet ne peut consulter que les objets dont les niveaux de classification sont supérieurs ou égaux à son propre niveau (no read down).
- Un sujet ne peut pas modifier un objet dont le niveau est supérieur à son propre niveau (no write up).

Le modèle de la figure 3.3-4 permet de représenter l'attribution de niveaux selon Biba, mais l'interprétation de ce modèle doit maintenant se conformer aux exigences de « no read down » et « no write up ». Le modèle de Biba est quasiment inverse du modèle de Bell-LaPadula. Ceci s'explique par son but de préserver l'intégrité des informations, ce qui n'est pas le cas de Bell-LaPadula.

Remarquons que les sujets ayant un certain niveau N n'ont pas accès aux objets de niveaux strictement inférieurs à N, mais ils peuvent consulter des objets de niveaux supérieurs. Un exemple d'application de ces règles est comme suit :

On considère un travail d'enquête dans un procès critique mené par trois équipes ayant trois niveaux de classification selon leur fiabilité. On entend par fiabilité, l'intégrité et

l'authenticité des rapports produits. On veut que chaque équipe fasse ses recherches et profite des résultats issus des autres équipes de fiabilité supérieure, et ce dans le but d'avoir trois résultats des trois équipes. La diversité des résultats est demandée et elle est bénéfique pour l'enquête; les résultats ne seront pas traités avec la même confiance. L'équipe 1 est la plus fiable; ses membres ont la tendance minimale à injecter des informations erronées dans leurs rapports. L'équipe 2 est moins fiable que l'équipe 1. L'équipe 3 est la moins fiable.

Pour composer avec cette situation on a intérêt à interdire à tout membre de l'équipe 1 de lire les rapports d'une équipe moins fiable (équipe 2 ou 3), car ceci pourrait affecter négativement la fiabilité de ses travaux qui pourraient être influencés par la lecture des rapports des autres équipes. Pour les mêmes raisons, l'équipe 2 ne devrait pas consulter les rapports de l'équipe 3.

Par contre, rien n'empêche un sujet de profiter des informations provenant d'une équipe plus fiable car cela lui permet d'accéder à des informations au moins à la hauteur de son niveau de classification reflétant sa fiabilité. Ceci ne remet pas en question le niveau de fiabilité des résultats de l'équipe considérée.

L'échange d'information entre les membres d'une même équipe est toujours permis car les sources et les consommateurs de l'information appartiennent au même niveau de classification indiquant leur fiabilité.

Un autre exemple intéressant inspiré de [30], est celui de la classification des applications selon le niveau de certitude comme une métrique d'assurance d'absence de failles de sécurité. On applique le modèle de Biba en exigeant qu'une application, de niveau de certitude n, ne puisse pas compromettre une autre application de niveau de certitude inférieur à n, c'est-à-dire présentant une possibilité de faille supérieure ou égale (moins sûre).

Le modèle de Biba garantit qu'une corruption de l'information ne peut pas passer d'un niveau de certitude à un autre plus haut, ni par consultation ni par écriture. On peut faire

propager de l'information corrompue à un autre sujet de même niveau ou de niveau inférieur. Aucun sujet ne peut induire en erreur un supérieur. Ceci n'empêche pas d'être induit en erreur par un supérieur.

Lattice

Adopter les niveaux NHS et NS, basés sur la confiance accordée aux sujets, comme unique critère de décision s'avère insuffisant dans certains domaines. Le métamodèle de contrôle d'accès Lattice, décrit dans [31] et [32], permet de considérer simultanément plusieurs critères de contrôle d'accès, entre autres, les NHS et NS (notés NHS-NS) et les compartiments, et ce en définissant des *classes de sécurité*. Ce métamodèle se base donc sur la définition d'un ensemble SC de *classes de sécurité*, et associe à tout objet une seule classe de sécurité. La définition d'un ensemble SC du modèle Lattice est telle que :

- SC est muni d'une relation d'ordre partiel ≤
- Tout sous-ensemble de SC a un plus petit majorant et un plus grand minorant
- Le **flux d'information** est permis d'une classe de sécurité c1 vers une autre classe c2, si la classe c2 est supérieure à la classe c1, selon la relation d'ordre partiel ≤

La Figure 3.3-6 tirée de [31], montre un exemple d'application du métamodèle Lattice. Les *classes de sécurité* sont représentées par les nœuds étiquetés. Chaque étiquette est formée d'un préfixe indiquant le niveau NHS-NS basé sur la confiance, et d'un suffixe indiquant un compartiment (au sens mentionné dans le métamodèle de contrôle d'accès Need-to-know). Un trait d'union « - » sépare le préfix du suffixe. Selon cet exemple:

- L'ensemble des NHS-NS est {U, C, S, TS} (respectivement pour Unclassified, Classified, Secret et Top Secret), et ses éléments sont tous comparables avec une relation d'ordre notée « < » telle que U < C < S < TS.
- L'ensemble des compartiments est muni d'une relation d'ordre partiel. Les compartiments sont nommés de sorte qu'un compartiment $Comp_i$ est inférieur à (inclus dans) un $Comp_j$ si les caractères formant le nom de $Comp_i$ sont tous des caractères du nom de $Comp_j$.

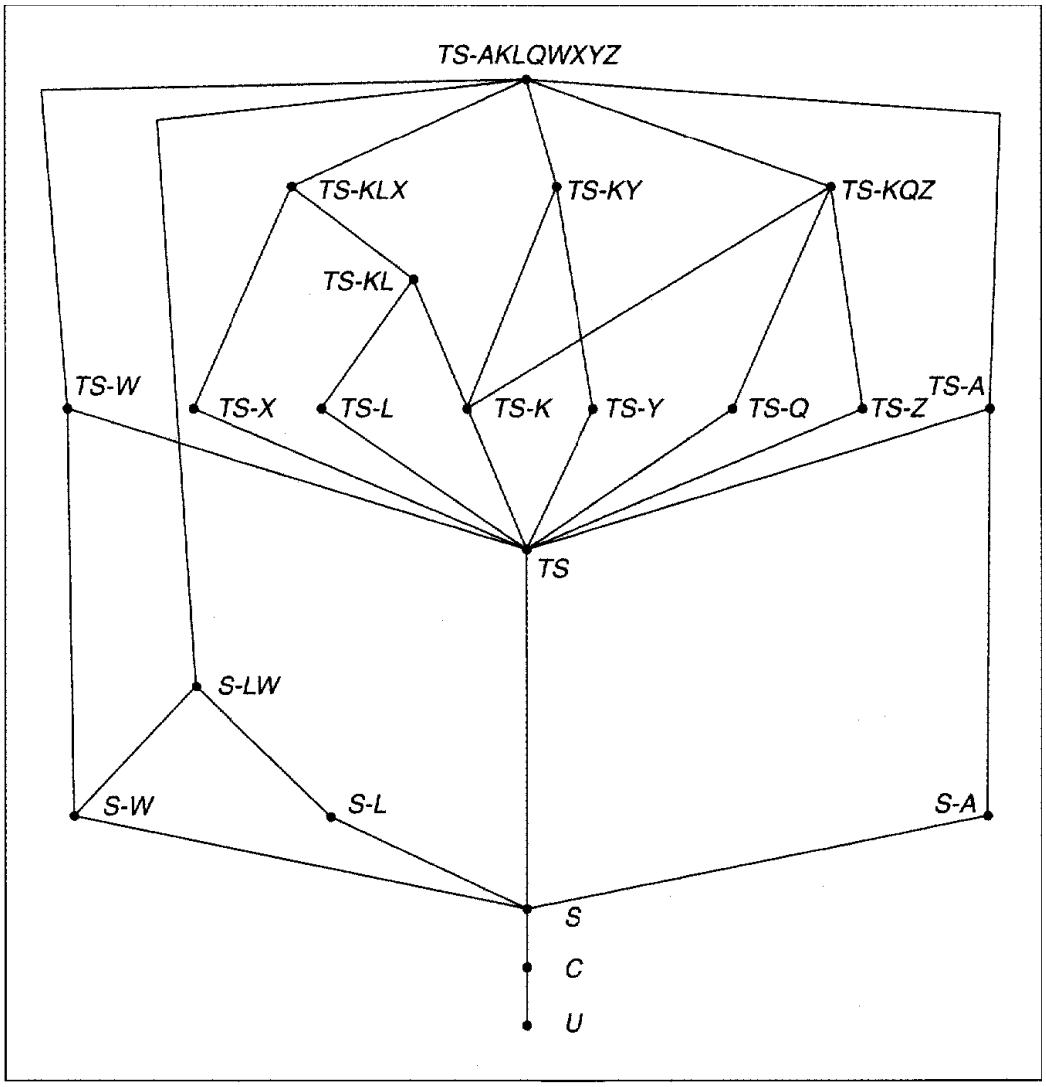

Figure 3.3-6 Un treillis représentant des classes de sécurité selon Lattice [31]

La comparaison de deux *classes de sécurité* consiste en la comparaison des étiquettes des nœuds du graphe. Une *classe de sécurité* représentée par l'étiquette (N1-Comp1) est inférieure à une autre étiquetée (N2-Comp2) si et seulement si N1 est inférieur à N2 et Comp1 est inférieur à Comp2 avec N1, N2 représentant des NHS-NS, et Comp1 et Comp2 des noms de compartiments.

Ainsi, on peut remarquer que le préfixe TS et le suffixe AKLQWXYZ marquent le nœud le plus haut dans le graphe. Ce nœud est un majorant de tous les autres nœuds, car d'une part, il a le plus haut niveau de confiance défini (Top Secret) et d'autre part, il a un suffixe indiquant un compartiment incluant tous les compartiments de l'exemple (A, K, L, Q, W,

X, Y et Z). U marque le nœud le plus bas dans le graphe. Remarquons aussi que les nœuds TS-W, TS-X, TS-L, TS-K, TS-Y, TS-Q, TS-Z et TS-A ont tous le même niveau Top Secret, mais avec des compartiments différents. Ils sont incomparables selon le treillis représentatif, car leurs compartiments sont non comparables (aucun compartiment de ce niveau n'est ni contenu ni conteneur d'un autre compartiment du même niveau). Ils ont, tout de même, un plus grand minorant qui est le nœud TS et un plus petit majorant qui est TS-AKLQWXYZ.

On peut généraliser et affirmer qu'une spécification de *classes de sécurité* basée sur des niveaux NHS-NS totalement ordonnés et sur des compartiments partiellement ordonnés, permet la définition d'une relation d'ordre partiel entre ces *classes de sécurité* selon les exigences du métamodèle de contrôle d'accès Lattice.

Un exemple pratique de l'usage de Lattice existe dans le domaine de la défense nationale où le NS d'un objet n'est pas le seul critère de restriction d'accès. Un autre critère de restriction d'accès est l'appartenance de l'objet à un compartiment distinct de ceux associés au sujet demandant l'accès. Généralement, un sujet est associé à tout compartiment d'objets utiles et reliés à son mandat et à ses missions. Par exemple, un officier des Forces aériennes ayant le NHS le plus haut (*Top Secret*), n'a pas à accéder au compartiment des informations financières sur les salaires du personnel. Cet officier aura un nœud représentatif dans le treillis, et ce nœud sera noté TS-Air (TS pour *Top Secret*, Air pour Forces aériennes). Pour avoir une illustration de Lattice pour le présent exemple, on considère les *classes de sécurité* (nœuds du treillis) suivantes :

- TS-Marine et S-Marine : pour indiquer, respectivement, les classes *Top Secret* et *Secret* dans le compartiment des Forces de la marine.
- TS-Logistique et S-Logistique: pour indiquer, respectivement, les classes *Top Secret* et *Secret* dans le compartiment logistique.
- TS-All et S-All: pour indiquer, respectivement, les classes *Top Secret* et *Secret* dans le compartiment qui contient tous les autres compartiments.

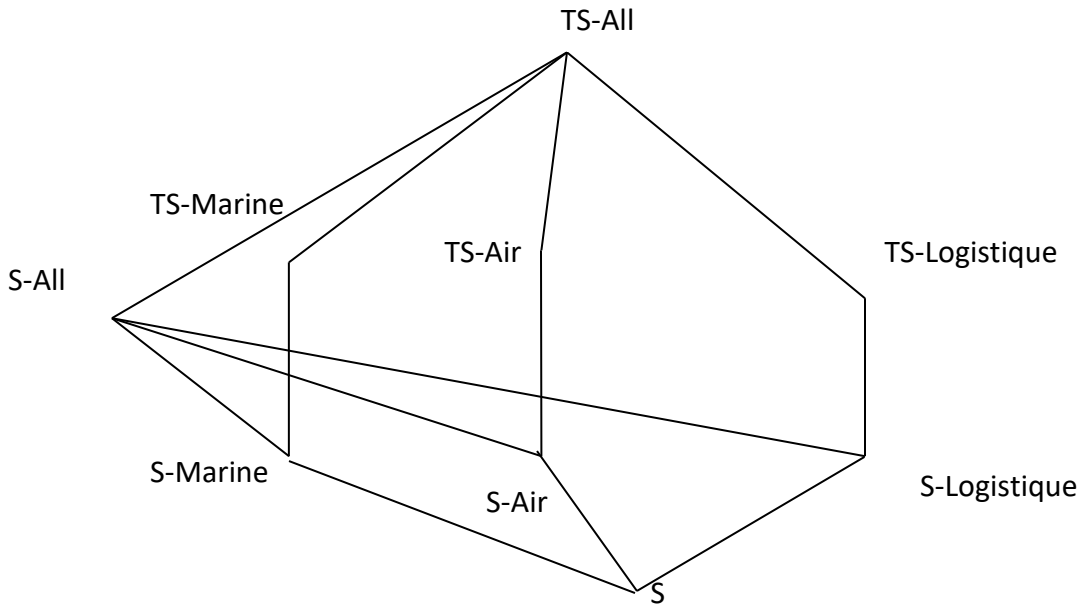

Figure 3.3-7 Exemple de classes de sécurité du domaine militaire

Sur la base de cette classification, on peut limiter les flux d'information pour respecter les deux règles suivantes : (1) un flux peut passer d'un compartiment à un autre compartiment qui le contient; (2) les flux permissibles sont de S vers S ou TS, et de TS vers TS. Il en résulte l'arrangement des nœuds correspondant selon le treillis de la Figure 3.3-7.

Selon ce treillis, les flux d'information de S-Marine par exemple sont permis seulement vers les classes suivantes : S-All, TS-Marine et TS-All. Remarquons aussi qu'à partir de TS-Marine, les informations ne peuvent cheminer que vers TS-All.

3.3.5 Contrôle d'accès basé sur les activités/équipes

Le concept de modèle de contrôle d'accès actif a été évoqué dans [33]. Un tel modèle doit tenir compte des contextes d'implication des sujets et des objets dans certaines activités ou de l'adhésion transitoire des sujets à certains groupes de travail, pour activer ou désactiver un droit d'accès.

Un exemple d'illustration est celui de la permission qu'on ne doit donner à un sujet S pour accéder à un objet O, que quand S est impliqué dans une activité concernant l'objet O. Le sujet S peut être un médecin, l'objet O un dossier de patient, et l'activité un diagnostic.

La durée de vie de la permission dépend de la durée de l'activité et de la durée de l'implication du sujet dans cette activité.

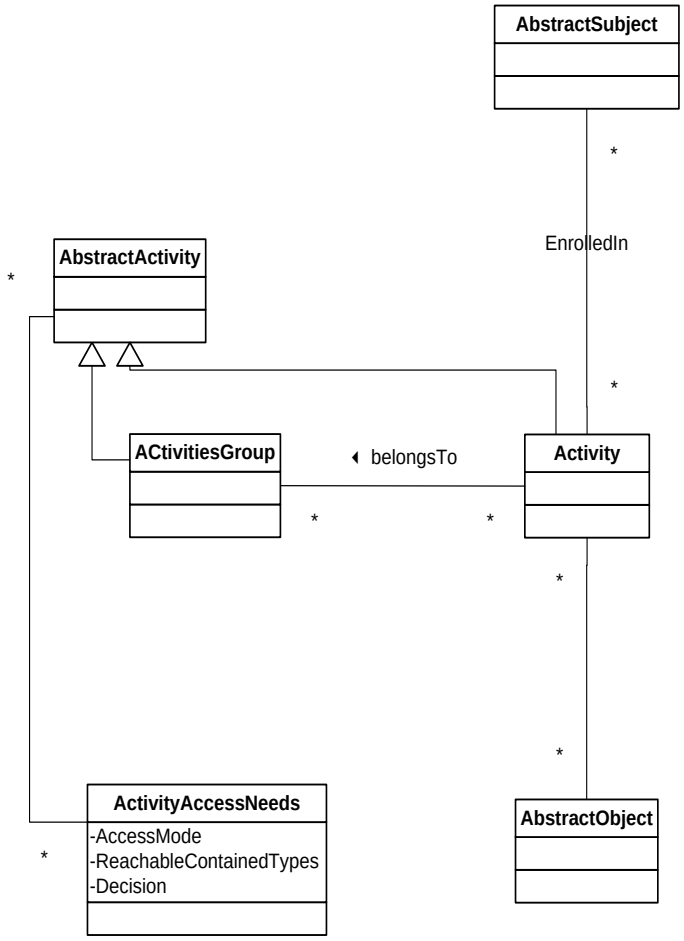

Figure 3.3-8 Métamodèle de contrôle d'accès Activity

Le métamodèle de contrôle d'accès, illustré dans la Figure 3.3-8, montre l'élément *Activity* qui représente le concept d'activité. Il est associé aux sujets et aux objets reliés à une activité (on rappelle que AbstractSubject représente un sujet ou un groupe de sujets; et similairement pour AbstractObjet). Par ailleurs, l'élément AbstractActivity qui représente des activités ou des groupes d'activités, est associé à un élément *ActivityAccessNeeds* définissant les privilèges d'accès reliés à l'accomplissement de ces activités et touchant les objets concernés par ces dernières. Ces privilèges seront octroyés aux sujets impliqués dans l'instance d'AbstractActivity considérée; cette instance est une activité ou un groupe d'activités.

3.3.6 Séparation des tâches

La séparation des tâches ou *Separation of Duty* (SOD) est un principe de contrôle d'accès qui vise à empêcher qu'une seule personne puisse accomplir, successivement ou simultanément, des accès à tous les objets d'un ensemble particulier d'objets. Le but est de prévenir l'accomplissement par une seule personne d'une tâche jugée sensible. Car une telle tâche permettrait à une personne d'abuser de ses privilèges pour atteindre des buts qui ne sont pas dans l'intérêt de l'entreprise. Par exemple, dans le domaine bancaire, un employé qui prépare une demande de crédit pour un client, ne doit pas accéder aux objets nécessaires pour approuver lui-même cette demande.

La modélisation de ce SOD dépend des concepts clés suivants :

- Historique d'accès pour certains objets et sujets. Cet historique peut se limiter à un intervalle de temps dont on spécifie la durée selon le besoin. Dans le cas de l'exemple précédent sur la demande de crédit, si une demande de crédit non approuvée expire dans deux jours, alors l'historique d'accès à surveiller ne doit pas dépasser les deux jours.
- La caractérisation de certains ensembles par le nombre maximal de leurs objets accessibles par un seul sujet.
- La spécification de séquences d'accès à empêcher. Il faut surveiller alors un nombre d'accès qui dépend des longueurs des séquences d'accès à interdire.

3.3.7 Clark-Wilson

Dans le but de préserver l'intégrité des informations, Clark et Wilson [34] ont proposé un modèle de contrôle d'accès selon lequel un objet n'est accessible que par un ensemble d'opérations jugées sans risque d'altération de l'intégrité de cet objet. Ces opérations classifiées sécuritaires vis-à-vis de l'intégrité sont dites certifiées. Il s'en suit un besoin de contrôle d'accès complémentaire limitant l'accès aux tâches de certifications des opérations.

Les détails des règles définissant ce modèle sortent du cadre de notre travail, car ils n'ajoutent pas de concepts autres que ceux déjà vus. Néanmoins, ce modèle mérite d'être

mentionné en raison de sa prise en charge d'un ensemble non prédéfini de modes d'accès possibles. Cela permet de représenter les classes des opérations certifiées et non certifiés par des modes d'accès séparés.

3.4 Concepts de spécification de contrôle d'accès

En plus des métamodèles de contrôle d'accès, notons l'existence de certains concepts qui ne constituent pas des principes de contrôle d'accès mais qui représentent des moyens d'expression de ces derniers. Ces concepts peuvent représenter des critères ou des contraintes qui complètent la spécification des ECAE selon des métamodèles de contrôle d'accès. Comme exemples de ces concepts, on cite : la délégation des privilèges d'accès, la prise en charge de l'aspect temporel ou spatial dans le contrôle d'accès, la permission d'accès dépendante du consentement d'un autre sujet, ou encore la conformité du contrôle d'accès à l'utilisation de l'information uniquement pour servir les raisons spécifiées lors du recueil de l'information.

3.5 Conclusion

Dans ce chapitre, on a vu que le domaine de contrôle d'accès est riche en principes et en concepts. Ces derniers vont servir de base pour l'élaboration d'un langage. On peut maintenant voir qu'une politique de contrôle d'accès peut se réduire à l'application d'un ou de plusieurs métamodèles de contrôle d'accès. Ainsi, si un élément de notre langage indique qu'une politique de contrôle d'accès est une application d'un métamodèle, la quantité d'information spécifiée par cet élément unique (sémantique de cet élément) est bien plus importante que celle généralement véhiculée par une simple règle de contrôle d'accès.

Cela va guider notre approche qui sera le sujet du Chapitre 5 où l'on discutera les critères à adopter pour décider quels métamodèles et quels concepts retenir comme éléments de notre langage et comment les intégrer ensemble.

Chapitre 4 Nos objectifs

Le but principal de notre travail est de réduire le fossé entre le niveau informel d'expression des exigences de contrôle d'accès et leurs spécifications formelles. L'élément clé sera l'élaboration d'un langage de spécification des ECAE. Ce langage qu'on appelle MACL sera élaboré pour servir nos objectifs principaux d'expressivité et de formalisme permettant la vérification de propriétés des ECAE. Ce chapitre explique chacun de ces objectifs ainsi que d'autres accomplissements qui seront préliminaires à nos objectifs.

4.1 Pouvoir expressif adéquat

On voudrait que le nouveau langage MACL ait une expressivité élevée lors de la spécification des ECAE, et ce par le biais des éléments de MACL qui correspondent aux concepts de contrôle d'accès au niveau des modèles de contrôle d'accès. Ces concepts forment normalement une bonne partie des spécifications informelles de contrôle d'accès d'une entreprise. Ainsi, à partir d'un texte écrit en langage naturel, on pourrait produire les expressions correspondantes dans le langage MACL avec moins de possibilité d'erreurs. Ce but primordial est en tête de nos objectifs parce qu'il traite de la problématique principale des niveaux d'expression des ECAE, expliquée précédemment et qui représente notre motivation de recherche.

Nos objectifs

La majeure partie de notre travail de recherche va se concentrer sur la tâche de doter MACL d'un ensemble de caractéristiques favorisant un pouvoir expressif adapté aux politiques de contrôle d'accès. Ainsi, on assurera une migration adéquate de ces politiques exprimées par des ECAE informelles appliquant des modèles de contrôle d'accès vers des spécifications formelles, et ce grâce aux éléments de MACL du niveau des modèles. Toutefois, il importe de préciser que nous ne visons pas l'automatisation de la migration des politiques de contrôle d'accès informelles vers notre langage MACL, mais plutôt la réduction de la complexité et des erreurs dues à cette migration. En effet, considérons l'exemple de la Section 3.3.2 sur le métamodèle de contrôle d'accès Chinese Wall. Cet exemple illustre la spécification d'une exigence de contrôle d'accès visant à empêcher qu'un employé d'une banque puisse divulguer des informations d'un client à un autre client en conflit d'intérêt avec le premier. On veut que MACL permette d'exprimer cette exigence en spécifiant tout simplement :

- Le type de métamodèle de contrôle d'accès : Chinese Wall ;
- Les éléments auxquels s'applique le principe du Chinese Wall :
    - Les groupes de conflit ;
    - Les classes des groupes de conflit.

Avec MACL, on n'aura pas à spécifier ce qu'est le métamodèle de contrôle d'accès Chinese Wall, évitant ainsi des erreurs et des efforts peu productifs. La spécification de la logique du Chinese Wall sera intégrée dans MACL comme il sera détaillé dans les Sections 6.2 et 7.2.2 , alors que dans un langage de règles (voir chap. 1), il faudrait fournir les règles de la logique du métamodèle de contrôle d'accès Chinese Wall, en plus de la spécification des classes et des groupes de conflit.

Cette remarque s'applique aussi aux autres métamodèles de contrôle d'accès vus dans le chapitre précédent, ce qui clarifie comment MACL va rendre la spécification formelle des ECAE plus aisée. MACL doit fournir les moyens de spécifier les instances des métamodèles de contrôle d'accès qu'il prend en charge.

La contribution de MACL va dépendre de sa richesse en éléments du niveau des modèles et de leur intégration dans le langage. Pour créer ces éléments de MACL, on va identifier des agencements de concepts de contrôle d'accès, chaque agencement permettant de décrire un ensemble d'ECAE qu'on appellera une famille d'ECAE. Ces agencements de concepts constituent les métamodèles de contrôle d'accès qui forment la base de notre langage.

Les éléments de MACL sont donc à identifier et à structurer avec les relations qui les relient. Des redondances de sémantique, et même des relations de généralisation entre les éléments de MACL sont à envisager. Une modélisation de ces éléments facilitera leur compréhension et analyse.

4.2 Réutilisation

Comme mentionné dans la section précédente, MACL sera basé sur les métamodèles de contrôle d'accès. De plus, on rappelle que d'après la Section 3.3, une instance d'un métamodèle de contrôle d'accès est une spécification d'une ECAE qui est un système de décision doté d'une logique de décision. Cette logique de décision est déjà définie une fois pour toutes au niveau de chaque métamodèle. Chaque instance de ce métamodèle représente une réutilisation de ce métamodèle et une réutilisation de sa logique de décision. L'instanciation représente une réutilisation qui se caractérise par la spécification des données particulières à l'instance créée, ceci représente une « personnalisation » qui multiplie les possibilités de réutilisation dans le sens de permettre pour un seul élément instanciable une multitude de réutilisations pour différentes applications personnalisées.

Cette *réutilisation* est une caractéristique visée de notre langage qui en représente un atout principal. Elle favorise une meilleure productivité et une réduction des erreurs. En effet, un élément réutilisable est mieux testé et plus fiable, car le coût de son élaboration et de sa vérification peut être distribué sur un bon nombre de réutilisations. La réutilisation en tant que telle est un moyen de test et de vérification. De plus, une

meilleure productivité en résulte du fait que réutiliser est généralement moins couteux qu'élaborer à partir du zéro.

4.3 Validation

La validation d'une spécification est une vérification de certaines propriétés qu'on trouve nécessaires pour qualifier cette spécification de non erronée (valide). Un but très important de notre langage MACL est de fournir un moyen de validation particulier qui est la vérification de la cohérence des spécifications de contrôle d'accès. Cette validation va permettre de détecter des erreurs dans les ECAE informelles, en examinant leur spécification écrite en MACL. Les incohérences sont essentiellement dues à l'existence de règles de contrôle d'accès dont l'application retourne des résultats contradictoires, comme des réponses de permission et de refus d'accès pour une même requête.

Il faut aussi être conscient que les décisions de contrôle d'accès ne dépendent pas uniquement des énoncés de MACL; elles peuvent dépendre de valeurs de variables qui ne figurent pas dans les énoncés de MACL. On reconnaît qu'un bon langage doit tenir compte d'une interaction avec l'environnement d'exécution dans toute sa complexité, et de son caractère dynamique. Ainsi, pour un énoncé en MACL, le PDP est un système dynamique et son état change avec l'évolution des objets environnants desquels il dépend, y compris l'historique des accès. Par exemple, le PDP peut retourner une réponse de permission ou de refus selon la valeur d'une variable représentant le temps, ou même selon les utilisations antérieures du PDP. Un exemple concret indiquant l'évolutivité du PDP a été présenté dans la Section 3.3.2 relativement au métamodèle de contrôle d'accès Chinese Wall : « l'accès à un dossier d'un client empêchera un accès ultérieur au dossier d'un autre client appartenant au même groupe de conflit ». On voit que la décision de contrôle d'accès dépend de l'historique des accès et que le PDP interprétant un énoncé en MACL est un système évoluant dans le temps; et conséquemment toute validation ou vérification de propriété doit en tenir compte.

Ceci étant, la validation comprendra deux volets : le premier sera d'examiner la présence de contradictions dans les énoncés en MACL (voir Section 10.1.3); le deuxième concernera la détection de contradictions en tenant compte de l'évolution des états des variables de l'environnement et de l'historique des accès. Ce deuxième volet représente un objectif ambitieux et sort du cadre de notre travail, tout de même la spécification formelle des exigences de contrôle d'accès en MACL prépare le terrain pour envisager la spécification formelle de l'aspect dynamique.

4.4 Vérification et complétude

L'*incomplétude* exprime l'existence de situations où aucune politique de contrôle d'accès ne peut être appliquée pour retourner une décision en réponse à certaines requêtes. Une vérification de complétude permet de s'assurer qu'un ensemble d'ECAE exprimées en MACL est suffisant pour retourner une décision de réponse à toute requête d'accès. À noter qu'une réponse de dernier recours, consistant en un refus de permission, est adoptée dans certains langages pour assurer la complétude.

La complétude est une propriété, parmi d'autres, qu'on pourra vérifier dans un énoncé en MACL. Dans le Chapitre 8, nous allons explorer le degré possible de vérification automatique de propriétés. Notre approche proposera aussi des moyens de vérification allant au-delà de la complétude, en couvrant des réponses à des questions d'accessibilité. Un exemple illustrant la vérification d'une propriété d'accessibilité serait la réponse à la question : « est-il possible qu'un objet soit absolument inaccessible? ».

4.5 Mécanisme d'interprétation

On voudrait qu'une spécification de contrôle d'accès en MACL puisse être traitée automatiquement pour déterminer une décision en réponse à une demande d'accès. On appelle cette capacité un *mécanisme d'interprétation.* De plus, on appelle *interprétation* d'une spécification de contrôle d'accès toute opération qui traite cette spécification pour

déterminer une décision de contrôle d'accès qui s'y conforme en réponse à une demande d'accès.

Pour se prêter à un traitement automatique appliquant un mécanisme d'interprétation, on va chercher à doter les éléments de notre langage d'un certain mapping vers des éléments qui se prêtent à un traitement automatique. L'atteinte de ce but est soit basée sur un mapping des éléments de notre langage vers des applications entre ensembles, soit par un mapping vers des machines à états finis [35]. La deuxième approche s'avère la plus aisée pour notre langage, vu son interprétation basée sur l'élaboration, ou calcul, de réponses à des demandes d'accès.

Une forme connue de cette approche est celle du mapping vers un langage exécutable. On s'attend à ce que cette forme simplifie le mécanisme d'interprétation de MACL, car elle bâtit sur l'adéquation au traitement automatique du langage exécutable qu'on appelle le **langage cible**.

Le dit mapping doit permettre l'élaboration d'un algorithme d'interprétation de MACL qu'un PDP applique pour retourner ses décisions en réponse à toute requête d'accès. Cet algorithme spécifie une séquence d'opérations qui applique le mapping vers le langage cible et définit comment l'énoncé résultant du mapping peut être traité pour interpréter un énoncé en MACL.

On compte définir MACL de sorte que tout énoncé dans ce langage corresponde à des instances des métamodèles de contrôle d'accès déjà vus dans le Chapitre 3. Cette correspondance servira de base pour définir le mapping vers le langage cible et sera présentée aux Sections 7.2.1 et 9.5.2.

4.6 Combinaison de règles et de politiques

Dans une entreprise, plusieurs politiques de contrôle d'accès sont généralement adoptées pour répondre à divers besoins de contrôle d'accès. Chacune de ces politiques a sa justification par rapport aux buts de l'entreprise. Une politique peut être en faveur

de permettre un accès à une ressource alors qu'une autre peut exiger de le refuser. Dans de telles situations dites de conflit, on doit pouvoir établir un moyen de régler les conflits en veillant à faire prévaloir les décisions qui assurent les intérêts de l'entreprise. En définitive, la décision de contrôle d'accès dépend des décisions provenant des différentes politiques de contrôle d'accès. Cela implique le besoin de spécifier comment aboutir à une décision unique à partir de plusieurs décisions, et ce selon une procédure qu'on a appelée, dans le chapitre 1, *composition de décisions* ou *combining algorithm* (ComAl).

Selon notre vision de MACL, une spécification d'une application d'un modèle de contrôle d'accès en MACL consiste en une spécification d'une instance d'un métamodèle de contrôle d'accès. Elle est de ce fait un système de décision. Ainsi, une spécification d'application de plusieurs modèles de contrôle d'accès peut être écrite en MACL, ce qui résulte en une spécification d'une politique de contrôle d'accès sous forme de plusieurs systèmes de décision. Ceci nous amène à envisager les conflits entre les systèmes de décision dans leur généralité, couvrant aussi bien les règles que les politiques de contrôle d'accès.

Comme application de cette vision, on considère l'exemple des ECAE suivantes pour une banque où :

A. Une première politique, basée sur les rôles, permet à tout agent dans le département des placements d'accéder aux dossiers des clients.
B. Une deuxième politique de Chinese Wall de la banque, empêche qu'un employé puisse accéder aux dossiers de plusieurs clients qui sont en conflit d'intérêt.
C. Une troisième politique stipule qu'un employé peut déléguer ses privilèges de rôle, temporairement, à un autre employé subalterne.

Dans ce contexte supposons que Pamela est une agente du département des placements qui délègue ses privilèges d'accès à un employé subalterne, Sam, pour permettre à ce dernier d'accéder au dossier d'un client X. Supposons également, que Y est un client dont le dossier a été déjà consulté par Sam, et que X et Y sont dans deux classes différentes d'un même groupe de conflit de Chinese Wall. On doit donc se conformer à trois

politiques de contrôle d'accès, ce qui n'est pas évident car on peut envisager plusieurs façons de se conformer à ces politiques. Ainsi, d'après la politique B Sam n'a pas le droit d'accéder à X, car il a déjà accédé à Y qui appartient à une autre classe. Mais, selon la délégation (politiques A et C) qu'il a reçu, il aura le droit d'accès à X, d'où la situation de conflit. En effet, une stratégie possible de règlement de ce conflit pourrait être la suivante :

1. Vu que la délégation de privilège de rôle, pour des requêtes auxquelles s'applique une politique de Chinese Wall, représente un risque de contournement du Chinese Wall, la décision résultante d'une délégation de privilège doit être un *Deny*.
2. Un *Deny* en vertu du Chinese Wall est dominant quel que soit la décision d'accès résultant de l'application d'une autre politique.

En effet, en spécifiant des ECAE d'une façon informelle, on se sert généralement de concepts de contrôle d'accès qui peuvent générer différentes décisions de contrôle d'accès. On utilise souvent un style qu'on qualifie de « multicritères » où l'on exige l'application simultanée de plusieurs critères de contrôle d'accès dont chacun est un système de décision, et qui fournissent plusieurs décisions en réponse à une demande d'accès. Ceci mène à des situations où l'on a à prendre une décision en fonction de plusieurs autres décisions. Le langage MACL qui vise une formalisation des spécifications des ECAE sera élaboré de sorte qu'il prenne en charge le style multicritères qui est utilisé intensément dans les spécifications informelles. On trouve le même style dans les systèmes normatifs où plusieurs articles d'une loi sont généralement pris en considération pour décider de la réponse à une seule action. Par exemple, pour décider de la sanction pour une infraction à la loi consistant en un dépassement de la limite de vitesse en ville, on applique plusieurs règles :

A. Une première règle tient compte de combien de km/h la vitesse limite a été dépassée ;
B. Une deuxième règle examine si le conducteur était en un état d'ébriété ;

C. Une troisième règle examine si le conducteur est un récidiviste de dépassement de limite de vitesse ;
D. Une quatrième règle examine si le conducteur est un récidiviste de conduite en état d'ébriété.

Chacune de ces règles retourne une réponse et la décision de sanction sera le résultat d'un traitement des quatre réponses fournies par les quatre règles. La spécification de ce traitement peut être formalisée en une spécification d'un ComAl qui prend plusieurs décisions en entrée pour fournir une décision unique en sortie.

En conséquence, le langage MACL doit être muni de moyens de spécification d'algorithmes de combinaison de règles et de politiques. Plusieurs solutions existent déjà dans XACML [36], [7], [5] et [37]. Toutefois, aucune de ces solutions ne représente un moyen assez général et assez expressif conforme aux objectifs d'expressivité concise et de cohérence de notre langage. Par exemple, aucune des solutions existantes ne permet d'associer une décision à la politique ou à la règle qui l'a fournie, ainsi une décision *Deny* résultant de l'application d'une politique suivant Chinese Wall aura le même effet que toute autre décision *Deny* quelle que soit la source de cette décision, ce qui n'est pas nécessairement désirable dans une spécification de contrôle d'accès.

L'élaboration d'un langage de spécification de combinaison de systèmes de décision, que l'on nommera ComLang qu'on présente au Chapitre 9, est de ce fait devenue un objectif primordial et une pierre angulaire de notre travail de recherche.

ComLang s'attaque au problème de combinaison de plusieurs systèmes de décisions, dans sa forme générale. Ceci donnera à ComLang une plus grande portée qui dépasse le contrôle d'accès, pour apporter une valeur ajoutée dans le domaine général de la spécification de composition de systèmes de décision. Un exemple assez connu de la composition de systèmes de décisions est celui de la reconnaissance des caractères manuscrits [38] par plusieurs systèmes experts. Chaque système expert est analogue à une politique ou à une règle de contrôle d'accès, car il produit une décision relativement à l'identification d'une lettre manuscrite. Comme les systèmes experts fournissent, en

général, des décisions d'identification divergentes, on a besoin de spécifier comment adopter une identification unique parmi plusieurs.

Le langage ComLang présenté au Chapitre 9 est intégré au langage MACL. Il est simple et concis, et offre une bonne expressivité à travers des spécifications compréhensibles. La validation des spécifications dans ComLang présenté à la Section 10.1.3 est indispensable car le nombre de système de décision spécifiés en MACL peut être, lui-même, considérable de sorte que l'expression de la composition des décisions peut facilement devenir incohérente sans une aide automatisée.

Ainsi le recours à une spécification formelle de combinaison de plusieurs politiques de contrôle d'accès constitue un levier important de MACL. Il permet l'expression de la coordination de l'application de différents modèles de contrôle d'accès représentant plusieurs buts de contrôle d'accès dans une entreprise.

4.7 Vérification de conformité à un règlement

L'examen de conformité à un règlement n'est pas l'objectif visé par notre travail. Néanmoins, les résultats attendus vont servir l'objectif de vérification de conformité à un règlement car MACL vise la spécification des ECAE au niveau de modèles; ce niveau caractérise généralement les exigences des règlements qui requièrent l'adhésion à des principes de contrôle d'accès. Comme travaux futurs, on peut envisager la possibilité de vérification de conformité à des règlements et ce à la lumière des points suivants : (1) la possibilité d'exprimer, en MACL, un règlement sous forme de politiques de contrôle d'accès; (2) la comparaison entre un règlement et des ECAE en se basant sur leurs spécifications respectives en MACL. La cohérence entre les deux spécifications permettra d'évaluer la conformité au règlement. L'examen de la cohérence utilisera les moyens de validation et de combinaison de politiques de contrôle d'accès supportés par MACL.

En effet, si on arrive à représenter un règlement par une politique P écrite en MACL, et si on représente des ECAE par un énoncé E écrit aussi en MACL, on peut alors combiner P et E dans un seul énoncé en MACL. L'examen de la cohérence de l'énoncé résultant

révèlera les éventuels points de divergence entre les ECAE et le règlement. On peut aussi spécifier un algorithme de combinaison de E et P de sorte qu'il fournisse une réponse d'alerte de non-respect du règlement chaque fois qu'il combine deux décisions spécifiées comme non conciliables (par ex. : Deny et Permit). Les moyens de vérification de propriétés de ComLang et de MACL peuvent aussi être utilisés pour détecter la possibilité d'une réponse du type *alerte* indiquant un non-respect du règlement considéré.

4.8 Adhérer à un standard

La conformité à un standard, comme XACML, reste toujours un objectif très souhaitable, surtout que XACML supporte le mécanisme d'extension par des profils. Notons aussi que XACML permet d'associer à un élément, des attributs non prédéfinis dans le standard, ce qui permet d'étendre ce standard sans l'ajout de nouveaux éléments. Ce but de conformité avec XACML n'est pas prioritaire pour le moment et sera reporté à un travail futur pour éviter d'entraver les objectifs principaux de notre initiative de recherche.

4.9 Récapitulation

En résumé et pour bien préciser les objectifs de notre travail, voici une liste énumérant ces objectifs:

1. Expressivité de MACL basée sur le métamodèle présenté au Chapitre 7 avec :
    a. **Éléments du niveau des modèles** : L'expressivité sera basée sur des éléments de MACL représentant des concepts des modèles de contrôle d'accès;
    b. **Éléments du niveau des règles :** L'expressivité utilisera aussi des éléments dee règles simples de contrôle d'accès;
    c. **ComLang** : MACL étendra un langage de spécification d'algorithmes de combinaison de règles et de politiques de contrôle d'accès ;

   d. **Complexité tolérable** : MACL doit représenter un moyen de formalisation des spécifications des ECAE avec un niveau de difficulté justifiable par l'apport de MACL.

2. Réutilisation : Elle est basée sur l'instanciation des métamodèles de contrôle d'accès qui encapsule leur logique de décisions et qui sont spécifiés une fois pour toutes. Chaque instanciation est une réutilisation.
3. Validation et vérification (V&V) des ECAE détaillées aux Chapitre 8 et Chapitre 10, avec :
   a. **Langage cible** : V&V sera basée sur un mapping vers un langage cible permettant son analyse.
   b. **Cohérence** : V&V permettra une validation de cohérence plutôt qu'une simple validation syntaxique.
   c. **Aspect Dynamique** : V&V prendra en charge l'aspect dynamique qui résulte du fait que les exigences de contrôle d'accès sont à interpréter en tenant compte des variations de certaines variables de l'environnement et de l'historique des accès.
   d. **Vérification de la complétude :** elle permet de s'assurer que les spécifications sont suffisantes pour retourner une décision de contrôle d'accès en réponse à toute requête d'accès.
   e. **Vérification de la conformité** utilisant la possibilité de vérification de propriétés exprimables dans le langage cible du mapping. Ceci est exploitable pour vérifier la conformité à des réglementations.

Chapitre 5 Approche

L'expressivité du langage MACL dépendra de l'adéquation de ce langage au domaine de contrôle d'accès au niveau des règles et au niveau des modèles. De plus, ce langage doit être doté des moyens permettant l'atteinte des objectifs de validation et de vérification. Nous considérons les exigences de contrôle d'accès qui ne sont pas simples ou qui consistent en un grand nombre de règles. Ces ECAE se présentent sous forme de centaines d'assertions représentant, informellement, des exigences du niveau des règles et du niveau des modèles (ex. les politiques de contrôle d'accès dans un hôpital). Notre intention finale est d'être en mesure d'aborder ce grand nombre d'assertions de contrôle d'accès avec un souci de réduction de complexité, de validation et de vérification. Notre principal moyen pour atteindre cette finalité est une approche de raffinement et de modularité.

5.1 Introduction

Pour surmonter le niveau de complexité des spécifications des ECAE, nous adopterons une stratégie de diviser pour gagner; et notre technique pour l'appliquer sera la modularité. Chaque module de la spécification des ECAE est un système de décision qui consiste en une instance d'un métamodèle de contrôle d'accès, ou en un système de décision formé d'un groupe de plusieurs systèmes de décision munis d'un algorithme

ComAl. Ce dernier compose leurs décisions et leur permet, de ce fait, de former un seul système de décision. Ainsi, les métamodèles de contrôle d'accès réutilisables par instanciation et leur intégration représentent le fondement de notre langage et expliquent le choix de l'acronyme MACL : Modular Access Control Language.

La métamodélisation, exposée dans le chapitre 3, sera adoptée comme approche de spécification de notre langage. Selon cette approche, un métamodèle est une spécification d'un langage. L'ensemble des métamodèles de contrôle d'accès ainsi qu'un métamodèle spécifiant les règles d'intégration des instances des métamodèles de contrôle d'accès et les systèmes de décision afférents (les modules) forment le métamodèle qui spécifie notre langage.

Comme langage basé sur la métamodélisation, MACL puise son vocabulaire dans les éléments de son métamodèle. Ce dernier fournit aussi les règles de grammaire à suivre. Les énoncés du langage correspondent à des instances du métamodèle du langage. Ces instances ne sont autres qu'une représentation visuelle des énoncés de MACL.

5.2 Identification des concepts de contrôle d'accès

Élaborer des métamodèles commence par l'identification des éléments représentatifs du domaine d'application. Dans cette tâche, nous aurons le souci de bien choisir des éléments de modélisation qui contribueront à l'expressivité de MACL. Un autre souci est d'éviter toute redondance entre des éléments de modélisation qui présentent des recouvrements (*overlap*) de sémantiques. Les sections suivantes présentent notre approche d'identification des éléments de modélisation qui conviennent aux exigences de MACL.

5.2.1 Anatomie des CCA et des métamodèles de contrôle d'accès

Les concepts de contrôle d'accès (CCA) représentent les éléments de construction des métamodèles de contrôle d'accès. Pour les identifier, on va se baser sur les principes de contrôle d'accès déjà vus dans les chapitres précédents. Un CCA doit représenter un

élément de factorisation commun à tout un ensemble d'ECAE qu'on nomme famille d'ECAE. Pour chercher les facteurs communs entre des ECAE, on peut considérer les critères et les principes de contrôle d'accès en répondant à la question : « pourquoi un privilège est donné ou non ». On peut aussi s'inspirer de la structure des informations utilisées pour exprimer la logique de décision propre à un métamodèle de contrôle d'accès. Deux CCA parmi les plus évidents à identifier sont le concept de sujet et celui d'objet. Ces deux CCA formeront deux éléments communs à tous les métamodèles de contrôle d'accès. Pour donner un autre exemple de CCA, considérons les métamodèles de Chinese Wall et de Séparation des tâches. La logique de décision dans chacun de ces deux métamodèles comprend le concept de « contrôle d'accès en fonction de l'historique d'accès ». L'élément qui représentera ce CCA sera une association nommée « hasAccessed » entre un AbstractSubject et un AbstractObject.

Un CCA doit représenter une sémantique commune à un ensemble d'ECAE. Cette sémantique commune prend la forme de structures d'éléments et de contraintes sur ces éléments, le tout pouvant être décrit par un modèle. Ce modèle sera propre à chaque CCA et il se situe au niveau M2 des métamodèles selon la terminologie des couches de modélisation expliquée dans le chapitre 3. De même, une ECAE qui respecte le métamodèle (qui se conforme au métamodèle) d'un CCA peut être spécifiée en une instance de ce métamodèle de contrôle d'accès.

Par ailleurs et dans un souci de mieux exploiter les CCA comme éléments essentiels du métamodèle de MACL, on s'intéresse particulièrement aux CCA qui permettent de représenter des systèmes de décision de contrôle d'accès. En effet, dans notre approche un métamodèle de contrôle d'accès (ACMM) se caractérise par le fait que ses instances sont des systèmes de décision. Ainsi, cette caractéristique guide notre travail d'identification des CCA et des ACMM. On rappelle que le concept de système de décision, déjà introduit dans la sous-section 1.1.1, est défini comme suit :

Un système de décision *de contrôle d'accès est une spécification de contrôle d'accès qui permet d'émettre une réponse à une demande d'accès; la réponse doit appartenir à une énumération de décisions de contrôle d'accès.*

Les éléments d'un métamodèle de contrôle d'accès et leurs associations doivent être suffisants pour spécifier la logique de décision propre à ce métamodèle. La logique de décision fait partie de la spécification de chaque métamodèle de contrôle d'accès et elle est à spécifier en termes des éléments de chaque métamodèle.

Une instance de ACMM peut ne pas être applicable à certaines demandes d'accès. Dans ce cas, la réponse sera « NotApplicable » ou « Indeterminate »; deux possibilités distinctes de réponse qui font partie de l'énumération des décisions qu'un critère de contrôle d'accès peut fournir.

Comme exemples de système de décision, on peut considérer des ECAE composées d'une politique P1 basée sur les rôles et d'une politique P2 basée sur les conflits d'intérêt. P1 est un système de décision car il permet de fournir une réponse (décision préliminaire) à une demande d'accès. Il en est de même pour P2. Un ComAl pourra être utilisé pour combiner les deux décisions éventuelles.

Dans la suite de ce travail on convient de qualifier d'*hybride* toute politique de contrôle d'accès ou ECAE qui applique plus qu'un métamodèle de contrôle d'accès.

5.3 Le métamodèle de MACL

Selon notre approche, un énoncé de MACL est un système de décision. Ce système de décision peut consister en une seule instance d'un métamodèle de contrôle d'accès comme l'exemple de l'application d'un CW de la Section 3.3.2. Dans le cas général, un énoncé de MACL consiste en une composition de plusieurs instances de métamodèles de contrôle d'accès dont les décisions sont composées par des ComAl, et de ce fait, il forme un système de décision. L'application de ComAl(s) à plusieurs systèmes de décision, pour

les transformer en un seul système de décision, est dite une *intégration* de systèmes de décision.

Ainsi, dans le cas général un énoncé de MACL consiste en un système de décision résultant d'une intégration d'un ou de plusieurs systèmes de décision. Notons qu'on peut envisager d'intégrer des systèmes de décision dont certains sont des résultats d'intégrations d'autres systèmes de décision. D'où l'aspect de modularité et de raffinement de MACL.

MACL sera spécifié par un métamodèle qui est formé d'un ensemble de ACMM tous inclus dans un métamodèle d'intégration dit IM. IM comprend les ACMM et définit les règles d'intégration des instances de ACMM pour en faire un seul système de décision. Le chapitre suivant présentera le métamodèle complet de MACL.

5.3.1 Les métamodèles de contrôle d'accès pris en charge par MACL

Notre langage MACL est conçu pour permettre l'expression des exigences de contrôle d'accès sous forme d'instances de ACMM. L'expressivité de MACL dépend des ACMM qu'il prend en charge. Ceci étant, on a choisi d'intégrer dans le métamodèle de MACL cinq ACMM parmi les plus courants dans les spécifications des ECAE, et qui sont : métamodèle de rôles, Chinese Wall (variante étoilée ou non), BLP, Biba et métamodèle des compartiments de Need-to-know.

Ces cinq ACMM offrent une expressivité satisfaisante dans un grand nombre de cas. Toutefois, dans le chapitre suivant nous montrerons que MACL est muni des deux leviers lui permettant d'exprimer des ECAE qui sortent du cadre des cinq ACMM. Ces deux leviers sont :

- Un noyau d'éléments du métamodèle de MACL qui permet d'exprimer des règles simples ;
- Une capacité de déclarer de nouveaux ACMM pour ensuite les réutiliser par instanciation.

5.4 Mapping de MACL et expression de décision

La sémantique d'un énoncé en MACL réside dans l'interprétation de cet énoncé afin de fournir une décision de contrôle d'accès à chaque demande d'accès. Pour permettre une interprétation univoque, on optera pour l'élaboration d'un mapping de MACL.

Comme mentionné dans la Section **Erreur ! Source du renvoi introuvable.**, un mécanisme d'interprétation de MACL est soit basé sur un mapping des éléments du langage vers des applications entre ensembles, soit par un mapping vers des machines à états finis. La deuxième approche s'avère préférable pour notre langage, car un système qui, suite à tout événement de demande d'accès, fournit une réponse, possède un comportement qu'on peut décrire par une machine à états finis. Une forme connue de cette approche est celle du mapping vers un langage exécutable qu'on appelle ***langage cible***.

5.4.1Langage cible

À ce point, nous sommes en mesure de donner plus de précision au sujet du langage cible déjà introduit à la Section 1.2 . Nous avons fait un choix parmi trois candidats pour notre langage cible : (1) la logique de premier ordre (FOL) avec utilisation de sa forme restrictive Constraint Logic Programming (CLP) [8] avec des possibilités d'implémentation utilisant les outils de programmation logique et les outils de Satisfiability Modulo Theory (SMT) [39]; (2) un langage déclaratif comme celui d'Alloy, basé sur un *model checker* [40]; (3) Object Constraint Language (OCL) [14]. Les trois options sont envisageables, mais nous avons opté pour la première option, CLP qui est basée sur FOL en raison de son fondement théorique qui n'est spécifique à aucun domaine d'application et qui ne dépend pas d'outils particuliers.

Pour plus de précision, notre langage cible est basé sur FOL avec les restrictions et les caractéristiques suivantes du CLP :

- Les formules logiques sont réduites aux *definites clauses* [8] comme disjonction de négations de prédicats avec un seul prédicat positif (sans application de l'opérateur de négation logique).

- Des contraintes et des opérations sur certains types d'éléments peuvent être spécifiées grâce à des théories prenant en charges ces types avec des prédicats prédéfinis. Les principaux types pris en charge par le paradigme CLP sont ceux des nombres rationnels et des listes.

Ainsi, d'un côté notre langage est moins expressif que FOL car il est privé d'une libre application de l'opérateur de négation ainsi que des quantificateurs et de toute forme autre que celle de la *définite clause*, mais d'un autre côté notre langage profite de : (1) l'expressivité de contraintes sur les nombres rationnels et les listes et (2) une procédure efficace de preuve de validité d'une proposition propre au CLP et dite SLD résolution [41]. Nous rappelons que dans le cas de domaines finis, qui seraient suffisants pour les spécifications des exigences de contrôle d'accès, tant FOL que CLP sont décidables, mais avec des efficacités algorithmiques très différentes.

Il importe aussi de mentionner que le langage cible partage avec FOL la caractéristique de complétude, mais tous les deux sont indécidables [41]. Ajoutons à ceci, que dans l'éventuelle nécessité de se libérer des contraintes d'expressivité du CLP comme celle indiquée à la Section 10.1, on se permet de spécifier une propriété à vérifier ou un mapping avec un énoncé FOL car ceci permet de fournir une matière qui pourra être utilisée dans un travail futur qui offrirait une implémentation libérée de certaines restrictions du CLP. Bref, le langage cible s'inscrit dans le paradigme CLP basé sur FOL et vise une implémentation moyennant un outil de programmation logique. Cette implémentation présentée au Chapitre 8 montre la faisabilité de notre solution; alors que notre travail conceptuel prépare le terrain pour tout travail futur utilisant une implémentation pouvant profiter de langages logiques plus généraux que CLP.

5.4.2Mapping et instances des éléments des ACMM

Considérons un énoncé en MACL consistant en une intégration d'instances de métamodèles de contrôle d'accès. Le mapping de cet énoncé va résulter en une spécification de ces instances, en plus d'une spécification de la décision de contrôle d'accès de chaque instance, et de la spécification des ComAl permettant l'intégration de

ces instances. Cette spécification contient aussi la logique de décision spécifiée dans la définition de chaque métamodèle de contrôle d'accès. Ainsi, on aura dans le langage cible un ensemble de propositions qui définissent une base de connaissances dite *programme logique*.

Interpréter l'énoncé MACL considéré revient à produire une décision pour chaque requête de contrôle d'accès. En effet, le programme logique et la représentation en CLP d'une requête d'accès doivent permettre de retourner une valeur de vérité (*true* ou *false*) à une décision de contrôle d'accès.

Le langage MACL et son mécanisme d'interprétation se basent sur deux principes : (1) le premier est de définir les éléments de MACL qui permettent de spécifier l'application des ComAI aux systèmes de décision, et ce en plus de leurs mappings; (2) le deuxième est de définir les éléments de MACL qui permettent la spécification des métamodèles de contrôle d'accès et de leurs instances, en plus de leur mapping. Ces deux principes seront développés dans les trois chapitres suivants.

5.5 Conclusion

La carte de route de l'élaboration de MACL est maintenant claire. Cette élaboration va s'étaler sur les trois chapitres suivants dont le premier définit le métamodèle de contrôle d'accès formé des ACMM pris en charge et des moyens d'intégration des instances des ACMM et d'autres systèmes de décision. Le chapitre qui suit dérive la forme textuelle de MACL à partir de son métamodèle. Le troisième, qui est le Chapitre 8, fournit un mapping vers le langage cible couvrant tous les éléments de MACL.

Chapitre 6 Le langage MACL et son métamodèle

Ayant déjà identifié les métamodèles de contrôle d'accès pris en charge par MACL, on va procéder dans ce chapitre à une révision de ces métamodèles avec une considération particulière de l'identification des éléments qui leur sont communs et des éléments encapsulant la logique de décision. On va élaborer le métamodèle d'intégration IM qui est le métamodèle complet de MACL et qui est formé des ACMM et des éléments régissant les règles de leur intégration. Ce métamodèle va fournir une représentation visuelle du langage en illustrant le modèle abstrait de MACL. Selon notre approche de métamodélisation présentée dans le troisième chapitre, ce métamodèle constitue une spécification du langage. On va élaborer un mapping entre les éléments textuels de MACL et les éléments correspondants dans le métamodèle IM. De plus, un mapping logique de ces éléments sera présenté dans ce chapitre.

6.1 Introduction

Un énoncé en MACL consiste en un système de décision résultant d'une intégration d'un ou de plusieurs systèmes de décision. Notons qu'on peut envisager d'intégrer des systèmes de décision dont certains sont des résultats d'intégration d'autres systèmes de décision. Ainsi, MACL se doit d'avoir deux volets de spécification : le premier volet lui permet de spécifier des instances de ACMM comme systèmes de décision, et le deuxième

volet lui permet de spécifier l'intégration de systèmes de décision. Dans ce chapitre, on commence par la présentation du premier volet en identifiant les ACMM pris en charge par MACL et dont les instances seront des systèmes de décision. Ensuite on passe au deuxième volet en présentant l'élaboration de la partie du métamodèle de MACL qui décrit les règles d'intégration de systèmes de décision. Rappelons que le métamodèle de MACL comprenant les ACMM et les règles d'intégration des systèmes de décision se nomme le métamodèle d'intégration (IM).

6.2 Les métamodèles de contrôle d'accès dans MACL

Les concepts capturés dans les ACMM appartiennent à deux groupes: (1) ceux qui ont trait au domaine de contrôle d'accès en général comme « Object », « Subject » ou « AccessMode »; (2) ceux qui sont propres à un ACMM en particulier, comme par exemple le concept de rôle dans RBAC.

Une instance d'un ACMM est elle-même un modèle, à un niveau de métamodélisation inférieur, créé en remplaçant les éléments du ACMM par leurs instances spécifiques correspondantes. Comme exemple, on peut se référer à la Figure 3.2-4 où Sam, CEO, Tom et Teller ainsi que leurs associations respectives représentent un modèle d'une réalité résultant de l'application de RBAC dans une organisation spécifique. Dans cette perspective, RBAC constitue un métamodèle qui définit un langage pour la spécification des modèles d'un niveau de métamodélisation inférieur. Ces modèles sont dits des instances du métamodèle. Bien sûr, les principes de modélisation illustrés dans cet exemple s'appliquent à tous les ACMM; et c'est pourquoi ces derniers seront plus précisément appelés métamodèles plutôt que modèles.

Dans la suite de ce chapitre, on va identifier les ACMM pris en charge par MACL. Chaque ACMM pris en charge subira certaines modifications visant une restructuration de ses éléments pour supporter son intégration dans MACL.

6.2.1 Les métamodèles de contrôle d'accès et leur capacité de décision

En plus des éléments qui représentent les concepts et leurs associations, chaque ACMM encapsule sa logique de décision de contrôle d'accès en réponse aux demandes d'accès. Ainsi, en tant que système de décision, une instance d'un ACMM émet ses décisions de contrôle d'accès en appliquant les règles logiques spécifiées au niveau du ACMM et exprimées en termes d'éléments de ce dernier. Chaque ACMM comprend un élément spécial, nommé DecisionHandler, dont la responsabilité est d'encapsuler la logique de décision permettant de déterminer la décision de contrôle d'accès en réponse à une demande d'accès. Ainsi, une instance de DecisionHandler est elle-même un système de décision, car cette instance dispose de la logique de décision encapsulée déjà dans DecisionHandler et qui lui a été transmise par instanciation. Chaque ACMM spécialise l'élément DecisionHandler selon ses besoins spécifiques de prise de décision, tel qu'illustré à la Figure 6.2-1. L'élément ACmetaModelElement est une généralisation de n'importe quel élément d'un ACMM différent de DecisionHandler.

L'association « uses » de DecisionHandler avec ACmetaModelElement est illustrée pour indiquer que chaque DecisionHandler spécifie sa logique de décision en termes des autres éléments du ACMM. Dans les illustrations suivantes des ACMM et par souci de simplicité et de lisibilité, nous allons omettre de montrer toutes les associations « uses » entre les spécialisations de DecisionHandler et les éléments restants du ACMM.

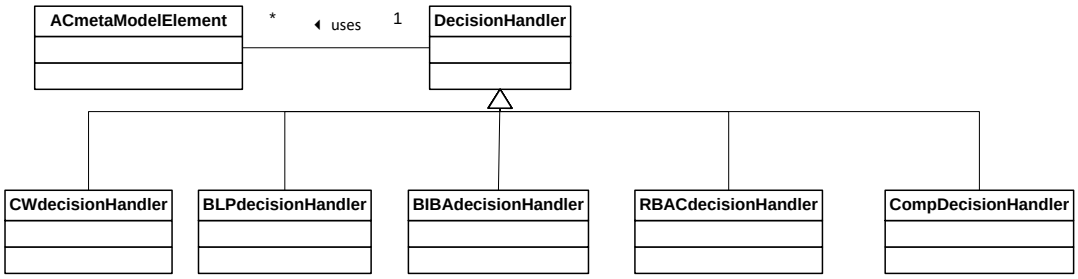

Figure 6.2-1 Les spécialisations de DecisionHandler dans différents ACMM

Dans les sous-sections suivantes, nous présentons les ACMM de base dont sera muni MACL. Cette base comprend: Chinese Wall, Bell LaPadula, Biba, RBAC et Compartment. En plus, nous introduisons un ACMM particulier dont la logique de décision n'est pas

prédéfinie, permettant ainsi une flexibilité de spécification de nouveaux ACMM définissant de nouvelles logiques de décision.

6.2.2 Éléments du noyau de contrôle d'accès

Nous présentons ici des éléments de contrôle d'accès qui représentent des éléments partagés et fondamentaux de nos ACMM. Ces éléments sont appelés éléments du noyau de contrôle d'accès (KernelElements) et ils sont définis ci-après et illustrés à la Figure 6.2-2.

Object, ObjectsGroup, AccessMode, Subject et SubjectsGroup sont des éléments du noyau qui ont déjà été introduits dans la Section 3.3.1.

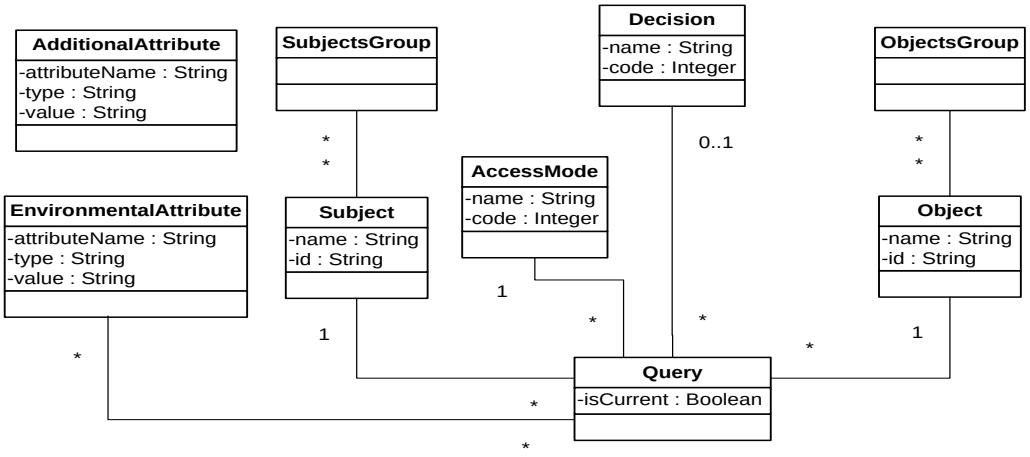

Figure 6.2-2 Éléments de noyau de CA

AdditionalAttribute : représente un élément qui sert à la spécification d'une propriété d'un autre élément du métamodèle. AdditionalAttribute a un nom et un type qui correspond à la spécification d'une propriété.

Query : représente une demande d'accès à un objet par un sujet. L'objet visé par la demande sera dit objet ciblé. Les associations de Query à respectivement Object, Subject et AccessMode (voir Figure 6.2-2) indiquent que nous associons respectivement à une instance de Query : (1) une instance d'Object qui représente l'objet ciblé ; (2) une instance de Subject qui représente le sujet qui demande l'accès ; (3) une instance de AccessMode qui représente l'action demandée par le sujet. L'attribut isCurrent de Query est de type

booléen, il permet d'identifier l'instance actuelle de Query représentant la requête à traiter. Cet attribut est requis lorsque nous devons prendre en considération des requêtes autres que celle à traiter (courante). Ceci est envisageable, quand les requêtes précédentes peuvent influencer une décision de contrôle d'accès.

EnvironmentalAttribute : similaire à AdditionalAttribute, mais utilisé pour décrire une propriété d'une entité pertinente de l'environnement qui n'est pas représentée dans les éléments de ACMM. Cet attribut est utilisé pour contenir des informations liées à l'accès à des événements de demande comme le temps, lieu, niveau d'urgence de contexte, température, etc., d'où l'association entre EnvironmentalAttribute et Query (Voir Figure 6.2-2) parce que la décision de contrôle d'accès dépendrait de l'état de l'environnement.

Decision : représente le concept d'une décision comme une réponse émise suite à une demande d'accès. Dans ce chapitre, les instances suivantes de Decision sont suffisantes pour illustrer les éléments de MACL : Permit, Deny, Indeterminate, NotApplicable. Elles sont les mêmes que celles de XACML.

Les éléments du noyau font partie de chaque ACMM. Ainsi, dans une politique de contrôle d'accès hybride, les sujets, les objets et le contenu de la requête font partie de chacune des instances des ACMM. En outre, notons le rôle clé de l'instance de requête dans la spécification de la logique de décision, indépendamment de l'ACMM considéré.

6.2.3 Le métamodèle de contrôle d'accès Chinese Wall

Le métamodèle de CW a déjà été présenté dans la Section 3.3.2. Il est montré à la Figure 6.2-3 illustrant les concepts déjà introduits du CW et ajoutant l'élément CWdecisionHandler. CWdecisionHandler est une spécialisation de DecisionHandler qui encapsule la logique de décision particulière au CW.

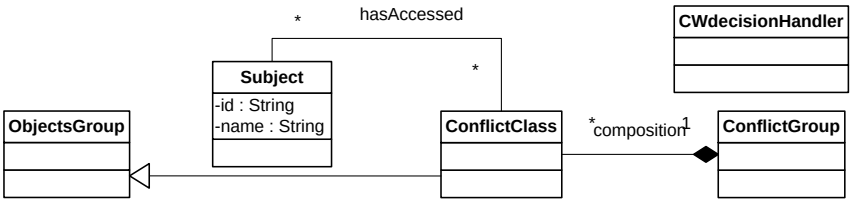

Figure 6.2-3 Métamodèle du Chinese Wall

CW et DCW ont été présentées dans les sous-sections 0.0.0 et 3.3.3. Ces deux variantes seront prises en charge par MACL en spécifiant une logique de décision propre à chacune d'elles. Ces spécifications seront détaillées dans la sous-section 7.2.2.

6.2.4 Le métamodèle de contrôle d'accès Bell LaPadula

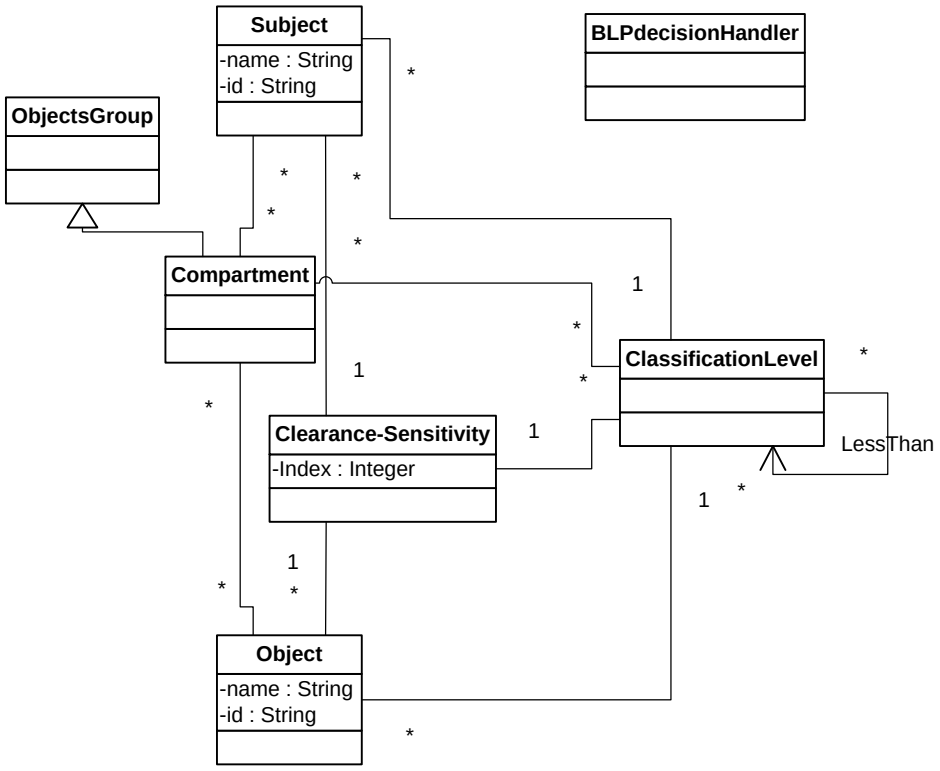

Figure 6.2-4 Métamodèle de contrôle d'accès BLP

Le métamodèle de contrôle d'accès BLP a déjà été introduit dans la Section 3.3.4. Pour satisfaire les exigences des ACMM, on y a ajouté l'élément BLPdecisionHandler comme

illustré à la Figure 6.2-4. BLPdecisionHandler spécialise DecisionHandler en encapsulant la logique de décision propre à l'ACMM BLP.

6.2.5 Le métamodèle de contrôle d'accès Biba

Le métamodèle de Biba a déjà été introduit dans la Section 3.3.4, le seul élément ajouté est BIBAdecisionHandler (Figure 6.2-5). Cet élément spécialise DecisionHandler selon la logique de décision de Biba. Il représente la capacité de décision permettant de présenter Biba comme un ACMM dont les instances sont des systèmes de décision.

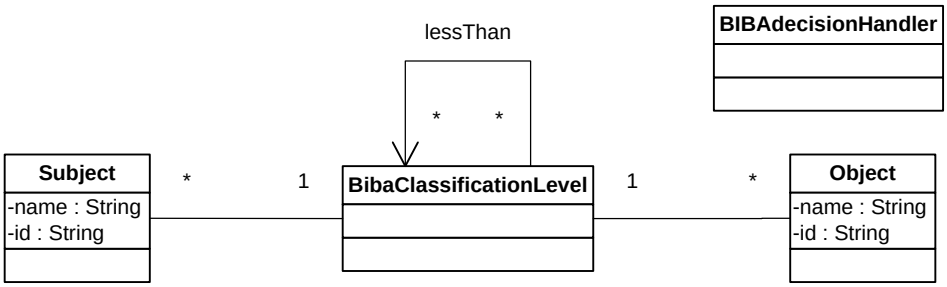

Figure 6.2-5 Métamodèle de contrôle d'accès Biba

6.2.6 Le métamodèle RBAC

Le métamodèle de RBAC a déjà été introduit dans la Section 3.3.1, le seul élément ajouté est RBACdecisionHandler.

RBACdecisionHandler : Cet élément spécialise DecisionHandler selon la logique de décision de RBAC. Il représente la capacité de décision permettant de présenter RBAC comme un ACMM dont les instances sont des systèmes de décision.

D'où son illustration à la Figure 6.2-6.

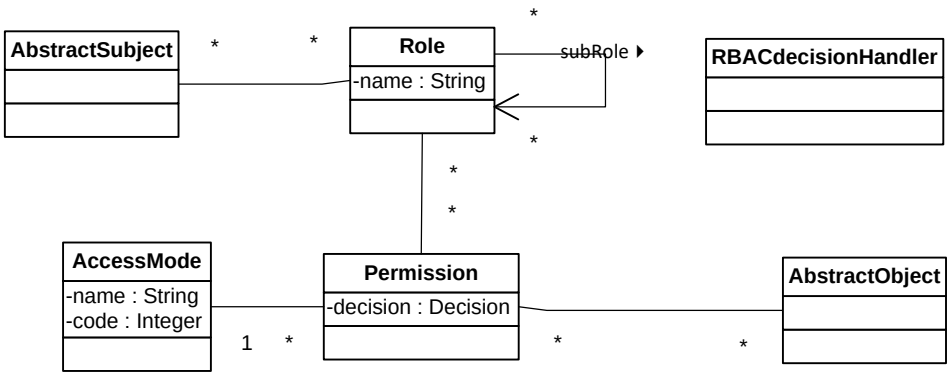

Figure 6.2-6 Métamodèle de contrôle d'accès RBAC

6.2.7 Le métamodèle de compartiments de Need-to-know

Le concept de compartiment satisfait les caractéristiques d'un ACMM dont les instances sont des systèmes de décision.

Ce métamodèle (Figure 6.2-7) comprend l'élément *Compartment* déjà introduit dans la Section 3.3.1. Les instances de cet élément sont associées à des sujets et à des objets. L'élément CompDecisionHandler spécialise l'élément DecisionHandler en encapsulant la logique de décision propre à ce métamodèle. Cette logique exige que toute instance de cet ACMM émette la décision Deny si, selon cette instance, l'objet ciblé est associé à un compartiment qui n'est pas associé au sujet requérant l'accès.

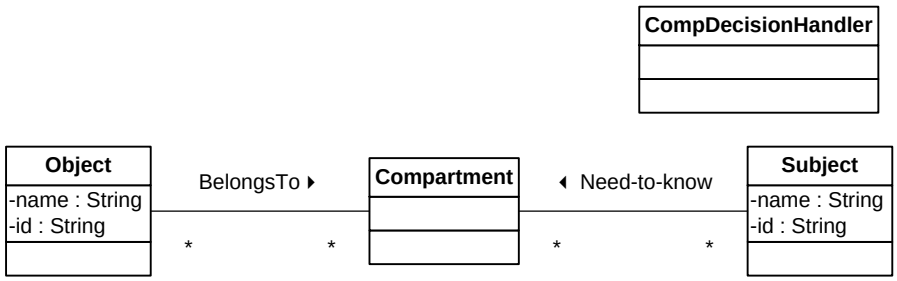

Figure 6.2-7 ACMM de compartiments de Need-to-know

6.2.8 Un métamodèle de contrôle d'accès générique

Rappelons que la Section 5.4 du chapitre présentant notre approche, précise que l'expressivité de la sémantique de notre langage se base sur son pouvoir de spécifier des métamodèles et des instances de métamodèles. Ainsi, pour spécifier une ECAE qui ne correspond à aucun des ACMM prédéfinis, notre langage sera doté des moyens de

spécification de nouveaux métamodèles de contrôle d'accès pour les réutiliser au besoin par instanciation. À cette fin, on a créé un ACMM qui contient déjà la définition des éléments du noyau de contrôle d'accès et une spécialisation de l'élément DecisionHandler. Ce métamodèle sera qualifié de générique et sera désigné par GenACMM. La logique de décision du DecisionHandler du GenACMM n'est pas prédéfinie, et c'est lors de chaque instanciation de cet élément que cette logique sera spécifiée. Cette logique s'exprime en termes des instances des éléments du noyau de contrôle d'accès en suivant le formalisme qu'on présentera dans les Sections 7.2.1 et 7.2.2.

Le chapitre suivant sera consacré à la présentation des éléments de MACL permettant la spécification de métamodèles ACMM, de la logique de décision et de l'instanciation des éléments déclarés.

6.3 Le métamodèle d'intégration IM de MACL

On rappelle qu'on a mentionné dans la Section 5.3 qu'un énoncé de MACL consiste en un système de décision résultant d'une intégration d'un ou de plusieurs systèmes de décision. Notons qu'on peut envisager d'intégrer des systèmes de décision dont certains sont résultants d'intégrations d'autres systèmes de décision. D'où l'aspect de modularité et de raffinement de MACL.

MACL est spécifié par le métamodèle IM qui est formé d'un ensemble de ACMM et d'un métamodèle d'intégration. IM comprend ces ACMM et définit les règles d'intégration des instances de ces ACMM pour en faire un seul système de décision. Les ACMM étant déjà introduits, il reste à spécifier les règles d'intégration des instances des ACMM pour compléter la spécification de l'IM. À noter que les éléments du KernelElements (voir Figure 6.2-2) font partie de l'IM car ce sont des éléments communs à tous les ACMM, qui sont eux-mêmes compris dans l'IM.

Notre approche d'intégration consiste à arranger, selon une structure arborescente, les instances de DecisionHandler des instances des ACMM d'une politique hybride de contrôle d'accès. On aura alors un arbre de systèmes de décision auquel on pourra

appliquer des ComAl, à raison d'un ComAl par groupe d'instances se trouvant à un même niveau de l'arbre. Ainsi, Pour spécifier l'intégration de plusieurs ACMM d'une politique hybride de contrôle d'accès, nous proposons des règles d'intégration basées sur une structure arborescente de systèmes de décision, appelée arbre de décisions ascendant (Ascending Decision Tree) (ADA). Les nœuds de l'ADA sont des instances de DecisionHandler ou des nœuds appliquant ComAl que nous appelons ComAlNode(s). Dans un ADA, chaque nœud terminal (feuille) est une instance de DecisionHandler qui émet sa décision de contrôle d'accès en appliquant la logique de décision de l'instance de son métamodèle. Par contre, dans un ADA un nœud non-feuille (interne) est une instance d'un élément qu'on appelle *ComAlNode* et qui émet sa décision en appliquant un ComAl sur les décisions de ses nœuds fils directs. Ces derniers nœuds enfants pourraient être eux-mêmes des feuilles ou des nœuds non-feuilles.

L'ADA a un nœud racine unique; cette racine est elle-même un système de décision. La décision de la racine représente la décision de l'arbre entier, c.à.d. la décision de la politique hybride de contrôle d'accès que l'arbre modélise. Notez que la décision du nœud racine conclut l'application d'un ou plusieurs ComAl à un ou plusieurs niveaux de l'arbre. Par ailleurs, nous remarquons que n'importe quel sous-arbre de l'ADA est lui-même un ADA, et est aussi un système de décision. Ainsi, l'ADA utilise un raffinement et une approche de diviser pour mieux régner. Cette approche permet de structurer la spécification de contrôle d'accès en plusieurs groupes auxquels s'appliquent plusieurs ComAl afin de toujours aboutir à une seule décision de contrôle d'accès.

Notons que pour un ensemble d'instances de DecisionHandler représentant l'application de plusieurs modèles de contrôle d'accès, la structure de l'ADA intégrant ces instances n'est pas unique. C'est la personne responsable de la spécification des ECAE, généralement désignée par « Privacy Officer » (PO), qui détermine comment diviser cet ensemble en un ou plusieurs groupes et comment concilier les décisions qui résultent de chaque groupe. Ceci se reflète sur le nombre de niveaux de l'ADA et sa structure générale. On rappelle, que la structure de l'ADA doit respecter toujours les exigences de structure

d'un arbre (pas de cycles) en plus de l'exigence que ses feuilles soient des instances de DecisionHandler et ses nœuds internes soient des instances de ComAlNode. Ainsi, tout en adhérant à ces exigences, le PO développe sa stratégie de réconciliation de différentes sources de décisions d'une façon spécifique au contexte, par la suite la conciliation est à « travailler » par le PO et elle n'est pas garantie. Ce qui est réutilisable dans un ADA consiste en les exigences déjà mentionnées; chaque ADA a une structure et des spécifications d'instances de ComAlNode spécifiques qui représentent une solution taillée sur mesure pour appliquer la stratégie de conciliation conçue par le PO.

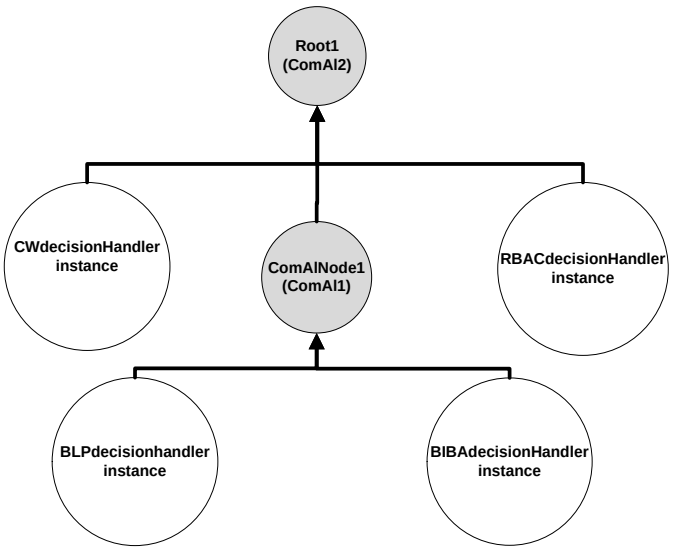

Figure 6.3-1 Un ADA intégrant des systèmes de décision d'une politique hybride de CA

Comme exemple d'application de l'intégration de systèmes de décision selon l'IM, nous considérons une politique hybride de contrôle d'accès illustrée à la Figure 6.3-1 par un ADA. Selon cet ADA, nous commençons par appliquer un ComAl, désigné par ComAl1, à une instance de BLPdecisionHandler et une instance de BIBAdecisionHandler. Puis un autre ComAl au niveau du nœud racine, désigné par ComAl2, s'applique aux décisions des éléments suivants : ComAlNode1, une instance CWdecisionHandler et une instance de RBACdecisionHandler. Ainsi, ce nœud racine de l'ADA produit une seule décision de

contrôle d'accès basée sur quatre décisions produites par quatre instances de DecisionHandler et l'application séquentielle de deux ComAl.

Notre IM est conçu de sorte qu'il englobe les ACMM, y compris leurs éléments DecisionHandler, en plus de l'élément ComAlNode. L'IM est donc suffisant pour exprimer la spécification des instances des ACMM et de leur intégration. IM est présenté sous différents angles : la vue principale qui capture la structure des systèmes de décision, à savoir DecisionHandler et ComAlNode ; et les vues présentées dans les Sections précédentes couvrant les éléments de noyau de contrôle d'accès et les éléments des ACMM. Notre IM est défini d'une manière qui permet la description d'une politique hybride et complexe de contrôle d'accès et ce en structurant des instances de DecisionHandler et de ComAlNode dans un ADA. La Figure 6.3-2 montre la vue principale de l'IM avec ses éléments structurants.

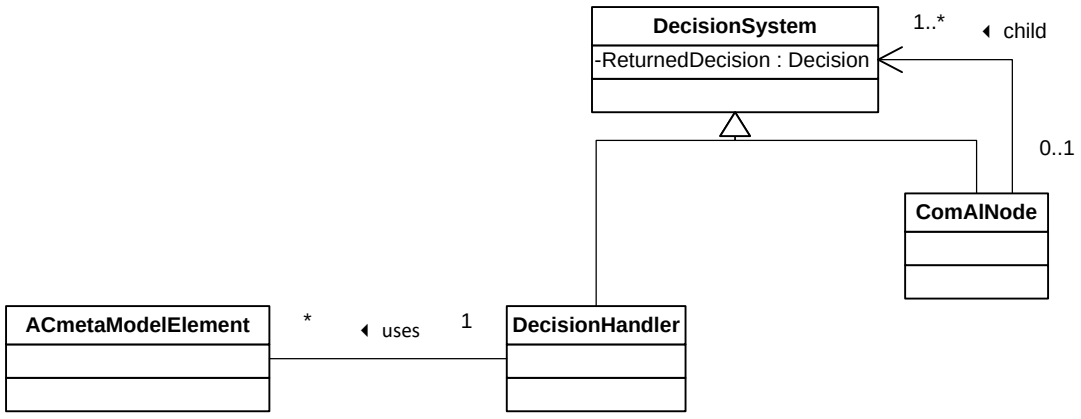

Figure 6.3-2 Vue principale de l'IM

DecisionSystem : cet élément représente le concept d'un système de décision dont les instances sont des systèmes de décision. Cet élément est doté de l'attribut ReturnedDecision servant à consigner une logique qui spécifie le choix d'une décision. Cet attribut sera expliqué dans la Section 7.2.2.

DecisionHandler : Cet élément a déjà été introduit comme élément commun à tous les ACMM dont les instances sont des systèmes de décision.

ComAlNode : cet élément représente un nœud de l'arbre qui n'est pas une feuille et qui applique un ComAl aux décisions émises par ses nœuds enfants. Son association à DecisionSystem, avec une multiplicité d'un ou plusieurs, indique que chaque instance de ComAlNode est associée à une ou plusieurs instances de DecisionSystem. Une instance de ComAlNode est elle-même un système de décision car elle a la capacité d'émettre une décision, et par conséquent c'est une spécialisation de DecisionSystem.

ACmetaModelElement : cet élément a déjà été introduit comme une généralisation de tout élément d'un ACMM. Il est montré dans cette vue pour indiquer que l'IM comprend, en plus de DecisionHandler, les éléments restants des ACMM. Il rappelle également qu'un DecisionHandler dépend, pour la spécification de la logique de décision, d'autres éléments du même AC métamodèle.

Pour résumer, l'IM comporte tous les éléments nécessaires pour spécifier une politique hybride de contrôle d'accès par un arbre de systèmes de décision, avec la capacité de spécifier chaque nœud système de décision avec ses éléments de modélisation spécifiques. Par conséquence, une politique de contrôle d'accès peut être spécifiée visuellement comme un ADA dans lequel on peut développer ou réduire :

- Un nœud (comme un système de décision) pour afficher ou masquer les instances des éléments d'un ACMM ou d'une spécification d'un ComAl ;
- Un sous-arbre pour afficher ou masquer un sous-ensemble de systèmes de décision se trouvant à un niveau inférieur direct.

6.3.1 Le « 123-4-5System » : un exemple de référence d'ADA

Afin de fournir un exemple d'illustration des concepts du métamodèle IM, on va considérer un exemple qu'on appelle 123-4-5System qui consiste en les exigences de contrôle d'accès d'une banque spécifiées par plusieurs instances d'ACMM, à savoir, des instances de Chinese Wall, de Need-to-know, de RBAC et de niveaux de sécurité. Cet exemple est illustré par l'ADA de la Figure 6.3-3. Les cinq feuilles de cet ADA sont des instances de DecisionHandler de cinq instances de ACMM. On convient de désigner les instances de ACMM par ACMM1, …, ACMM5. Leurs instances de DecisionHandler sont désignées, respectivement, par ACMM1-DH, …, ACMM5-DH.

Selon 123-4-5System, le processus d'analyse de toute demande d'accès suit deux étapes. Dans la première étape, on examine si le niveau de classification du sujet lui permet d'ignorer des restrictions d'accès éventuelles du Chinese Wall (définies dans ACMM1 et représentées dans l'ADA par l'instance de DecisionHandler nommée ACMM1-DH); on examine aussi si l'objet visé appartient au compartiment de Need-to-Know du sujet. Dans la deuxième étape, représentée par le deuxième niveau de l'ADA, on examine si l'accès demandé est conforme aux exigences d'un deuxième critère de Chinese Wall (défini dans ACMM4) et si le sujet possède le bon rôle (défini dans ACMM5). Ce deuxième critère de Chinese Wall est qualifié d'incontournable quel que soit le niveau de classification ou le rôle du sujet. Cela veut dire qu'en aucun cas un sujet ne peut accéder à deux classes d'un même groupe de conflit du Chinese Wall défini par l'instance ACMM5.

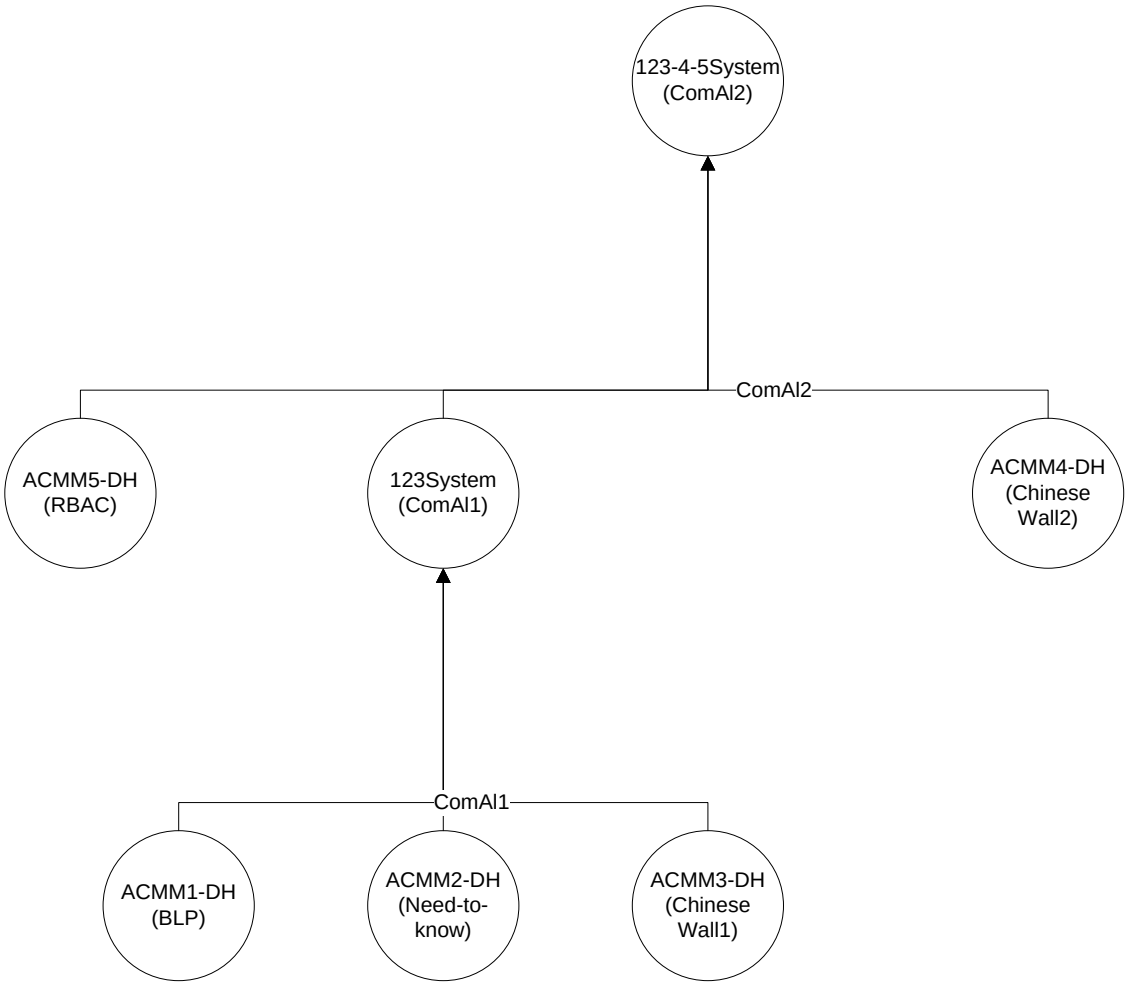

Figure 6.3-3 Un exemple de composition des décisions de plusieurs instances de ACMM

Ainsi, selon l'ADA représentatif de notre exemple, 123-4-5System applique plusieurs politiques de contrôle d'accès en combinant les décisions : (1) un Chinese Wall ACMM3 qu'un sujet peut ignorer si ce sujet a un niveau de classification suffisant et si l'objet auquel on demande l'accès est dans le compartiment du Need-to-Know du sujet; (2) si l'application de ce qui précède aboutit à une réponse différente de *Deny*, alors on applique une instance de RBAC ACMM4 et une instance de Chinese Wall ACMM5 et on compose les décisions résultantes. Selon cette dernière composition une permission d'accès n'est possible que si ACMM4, ACMM5 et le résultat de la partie (1) résultent tous en des décisions autres que Deny. Cette composition de décisions est détaillée dans le point 2 plus loin dans cette section.

À noter que les décisions possibles des systèmes de décision considérés sont les éléments de Σ = {Permit, Deny, NotApplicable}.

Ainsi l'ADA du 123-4-5System comprend deux niveaux de combinaison de décisions détaillés comme suit (Le symbole * désigne une décision quelconque) :

1. À un premier niveau (celui de ComAl1): L'idée principale derrière ComAl1 est de combiner ACMM1, ACMM2 et ACMM3 de sorte qu'un *Permit* nécessite au moins un *Permit* de ACMM1 ou ACMM3, en plus d'un autre *Permit* de ACMM2. ComAl1 permet de créer 123System qui est spécifié par les associations suivantes reliant une ou plusieurs listes de trois décisions d'entrée à une décision de sortie :
    a. [Permit, Permit, Deny] ou [Deny, Permit, Permit] → Permit
    b. [NotApplicable, Permit, Permit] ou [Permit, Permit, NotApplicable] ou [NotApplicable, Permit, NotApplicable] → Permit
    c. [Permit, Permit, Permit] → Permit
    d. [NotApplicable, NotApplicable, NotApplicable] → NotApplicable
    e. Toutes les listes de décision restantes entrainent un *Deny* et elles sont :
        i. [Deny, *, Deny] → Deny
        ii. [*, Deny, *] → Deny

iii. [NotApplicable, *, Deny] → Deny

iv. [*, NotApplicable, *]→ Deny

2. À un deuxième niveau (celui de ComAl2) : On combine 123System avec ACMM4 et ACMM5, de sorte qu'un Permit nécessite une décision autre que Deny de 123System et un couple de (Permit, Permit) ou (Permit, NotApplicable) de ACMM4 et ACMM5. Plus précisément, ComAl2 associe les listes de trois décisions aux décisions de sortie comme suit :
   a. [Permit, Permit, Permit], [Permit, NotApplicable, Permit], [Permit, NotApplicable, NotApplicable] ou [Permit, Permit, NotApplicable] → Permit
   b. [NotApplicable, NotApplicable, NotApplicable] → NotApplicable
   c. Deny appartient à liste des décisions d'entrée → Deny

Dans la suite de ce document, et sauf mention contraire, l'exemple 123-4-5System sera adopté comme exemple de référence. Il sera désigné par 123-4-5System et ses spécifications seront référencées directement par leur numérotation (ex. 1.a, 3.b, ComAl1, ComAl2, etc.) dans cette section et dans l'ADA qui les illustre.

La partie précédente de cette section a permis de fournir un exemple de plusieurs ComAl dans un ADA. Pour compléter cet exemple concrétisant un ADA on va fournir des énumérations de sujets et d'objets, en plus de spécifications inhérentes aux ACMM de l'ADA. Cela est présenté sous forme tabulaire comme suit :

Tableau 6.3-1 Les sujets, leurs rôles ACMM4, leur compartiment et leurs NHS dans ACMM1

Sujet	**Rôle du ACMM4**	**Compartiments du ACMM2**	**Niveau d'habilitation de sécurité du ACMM3**
Tom	Conseiller en placement	Applications de base (AppB) + FinanceAffaires +	TS

		FinanceParticuliers +Investissement	
Sam	Conseiller en prêts-particuliers	AppB +FinanceParticuliers	TS
Karine	Conseiller en prêts-affaires	AppB +FinanceAffaires	S
Paul	Agent au comptoir	FinanceAffaires + FinancesParticuliers	C
Sara	Agent au comptoir	FinanceAffaires + FinanceParticuliers	TS
Bob	Caissier	FinanceAffaires + FinanceParticuliers	S
Pamela	Directrice de succursale	AppB + Investissement + Finance + ressources humaines (HR)	S
Jamal	Gardien	AppB	TS
Fred	Agent de soutien technique en TI	AppB +Technologies de l'information (TI)	TS
Kim	Opérateur ATM	AppB + Technologies de l'information	U

Le tableau ci-dessus fournit une énumération d'un certain nombre de sujets du cas étudié ainsi que leurs rôles selon l'instance ACMM4 du ACMM de rôles, leur compartiment selon une instance ACMM2 du ACMM Need-to-Know et leur niveau d'habilitation de sécurité (NHS) selon une instance du ACMM Bell-LaPadula. TS, S, C et U désignent respectivement les NHS Top Secret, Secret, Confidentiel et Unclassified.

Tableau 6.3-2 Les objets, leurs NS dans ACMM1, leur compartiment, et leurs appartenances aux classes de groupes de conflit

Objet	**Id**	**Niveau de sécurité ACMM 3**	**Compartiment ACMM2**	**Classes des groupes de conflit de ACMM3 Groupe1 Croupe2**		**Classes du groupe de conflit de ACMM5**
Dossier de nos placements dans la banque CIT	O1	TS	Investissement			CW2C1
Dossier de nos placements dans la banque CanadaTrust	O2	TS	Investissement			Cw2C2
Dossier de placement du développeur Vachon	O3	TS	Finance	CW1G1-1		
Dossier hypothèques du développeur AssuranceVie	O4	TS	Finance	CW1G1-2		
Placement AssuranceVie	O5	S	Finance	CW1G1-2		
Payroll	O6	C	HR			
Promotion	O7	TS	HR			
Comptes particuliers	O8	C	FinancePa rticuliers			
Comptes affaires-AssuranceVie	O9	S	FinanceAf faires	CW1G1-2		
Comptes affaires-Bistro	O10	S	FinanceAf faires		CW1G2-1	

Comptes affaires-East Side Mario	O11	S	FinanceAffaires		CW1G2-2	
DBMS	O12	TS	TI			
Outils informatiques	O13	TS	TI			
Navigateur Web	O14	U	AppB			
Éditeur de texte	O15	U	AppB			
Gestion ATM	O16	S	TI			

Le tableau ci-dessus énumère des objets en spécifiant leurs identifiants, les compartiments qui les contiennent, et leurs appartenances à des classes de groupes de conflit pour les deux instances du ACMM Chinese Wall.

Remarquons que pour la première instance ACMM3 de Chinese Wall, on a deux groupes de conflit : (1) Groupe1 reflétant le besoin d'éviter un conflit entre des compagnies œuvrant dans le marché immobilier et contenant les classes CW1G1-1, CW1G1-2; (2) Groupe2 reflétant le besoin d'éviter un conflit entre les sociétés de restauration et comportant les deux classes CW1G2-1 et CW1G2-2.

Tableau 6.3-3 Les privilèges des rôles du ACMM4

Rôle	**Modes d'accès**	**Objet**	**Sous-rôle**
Directrice de succursale	Read	ALL	Rôle de base
Conseiller en placement	Read	ALL	Rôle de base
Conseiller en placement	Write	O1, O2, O3, O5	Rôle de base
Conseiller en prêts-particuliers	Read	O8	Rôle de base

Conseiller en prêts-affaires	Read	O9, O10, O11	Rôle de base
Agent au comptoir	Read, Write	O8, .., O11	--
Caissier	Read	O8, …, O11	--
Gardien	Read, Write	O14	Rôle de base
Agent de soutien TI	Read, Write	O12, O13	Rôle de base, Opérateur ATM
Opérateur ATM	Read, Write	O16	Rôle de base
RoledeBase	Read, Write	O14, O15	--

Le tableau ci-dessus spécifie les privilèges des rôles en associant chaque rôle à des modes d'accès sur des objets. Le tableau spécifie aussi les éventuels sous-rôles de chaque rôle.

On va maintenant supposer que Karine qui joue le rôle de « conseiller en prêts-affaires » prépare une demande de prêt pour l'affaire du restaurant Bistro. Elle a déjà accès aux comptes affaires-Bistro (O10), mais elle voudrait élargir son analyse du dossier en y incluant des informations sur l'état financier du propriétaire du Bistro en tant que client particulier dont les comptes clients sont différents du compte de l'affaire Bistro. Elle émet une demande d'accès en mode lecture aux comptes particuliers (O8). Pour étudier cette demande d'accès, on recueille les décisions préliminaires de chaque système de décision de l'ADA auxquelles on applique ComAl1 et ComAl2 pour obtenir la décision de sortie :

- ACMM1 → NotApplicable (car O8 n'appartient à aucune classe de groupe de conflit)
- ACMM2 → Deny (car O8 n'appartient à aucun compartiment de Karine)
- ACMM3 → Permit (car le niveau de O8 est *Confidential* donc inférieur au NHS de Karine qui est *Secret*)

On applique alors ComAl1 à la liste de décision [NotApplicable, Deny, Permit]. La spécification 1.e.ii de ComAl1 ([*, Deny, *] → Deny), nous permet de conclure que ComAl1 retourne une décision Deny, ceci veut dire que 123System retourne la décision Deny.

Continuons au deuxième niveau de l'ADA avec ACMM4 et ACMM5 :

- ACMM5 → NotApplicable (car le rôle de Karine ne permet pas l'accès à O8)
- 123System → Deny (d'après ComAl1)
- ACMM4 → NotApplicable (car O8 n'appartient à aucune classe de groupe de conflit)

Ainsi, pour trouver la décision du système 123-4-5System, on doit appliquer ComAl2 avec la liste de décisions [NotApplicable, Deny, NotApplicable]. La spécification 2.c de ComAl2 (Deny appartient à la liste des décisions -> Deny) permet de conclure que la décision de sortie est Deny.

Avec cet exemple de demande d'accès simple de Karine, on a vu que l'ADA nous a permis de structurer notre raisonnement pour arriver à une décision de contrôle d'accès. Ceci s'avère intéressant, mais ce que nous visons dans la suite de notre travail est d'étudier la spécification formelle des ADA. Ce formalisme est essentiel pour permettre l'automatisation de notre dernier raisonnement et le traitement de questions dont chacune peut concerner un ensemble de requêtes, des exemples de telles questions sont les suivantes: « quels sont les objets accessibles en mode lecture par Karine? », « quels sont les sujets qui peuvent accéder à O8 ou O10? », etc.

6.4 Discussion et comparaison avec des travaux connexes

Les articles [42, 43] de Barker sont parmi les plus connus sur le thème de la modélisation de contrôle d'accès. Barker propose un métamodèle qui identifie les « primitives » de CA: sujet, ressources et action complétée avec un élément abstrait appelé « Category ».

Category peut représenter n'importe quel concept de contrôle d'accès qui n'est pas prédéfini comme l'une des primitives mentionnées ci-dessus; ce concept devrait être spécifique à un certain métamodèle de contrôle d'accès. Catégorie est un élément générique pouvant représenter les notions de rôle, d'habilitation de sécurité, de sensibilité, ou tout autre concept de contrôle d'accès. Citons également une extension au métamodèle de Barker [44] qui y ajoute un élément qui permet la spécification de contraintes sur les éléments du métamodèle. Notre travail est cohérent avec le métamodèle de Barker ou son extension; en plus il permet la spécification de chaque métamodèle de contrôle d'accès avec un ensemble d'éléments prédéfinis, y compris sa logique sous-jacente de décision. Un autre aspect de la comparaison entre notre métamodèle et le travail de Barker réside dans la capacité d'envisager un recours structuré à des combining algorithms pour considérer des politiques de contrôle d'accès hybrides.

Citons également d'autres œuvres basées sur la logique permettant de spécifier les exigences de contrôle d'accès, comme celles de Craven et al. [45] et Gelfond et Lobo [46]. Ces travaux ont des caractéristiques intéressantes, mais ils ne considèrent pas une approche de modélisation pour illustrer les concepts de contrôle d'accès en reliant ces concepts à des éléments d'un langage de spécification de contrôle d'accès. L'approche de SecPAL [47] propose un cadre déclaratif général pour préciser les exigences de contrôle d'accès avec une syntaxe intuitive semblable à celle de la programmation logique, mais ce travail ne s'appuie pas sur, ni ne propose une approche de modélisation.

En outre, plusieurs tentatives d'étendre UML pour prendre en charge la conception et la mise en œuvre du contrôle d'accès devraient être mentionnées. Epstein et Sandhu [48] utilisent UML pour spécifier les exigences RBAC. Shin et Ahn [49] proposent des techniques utilisant la notation UML pour réaliser la modélisation RBAC. Doan et al. proposent une base pour la spécification du *mandatory AC* dans des diagrammes UML [50] .

Dans son approche UMLsec pour modéliser des systèmes sécurisés avec une extension de UML, Jurjens [51] recourt à des annotations formellement spécifiées attachées à des éléments de UML pour spécifier des exigences de contrôle d'accès. SecureUML est proposé par Basin [52] comme une preuve de concept d'adoption d'une approche dirigée par les modèles pour décrire des systèmes sécurisés; mais SecureUML est limité à la spécification RBAC sans prise en charge d'autres ACMM. Dans leur travail, Pavlich-Mariscal et al. [53] proposent une extension d'UML pour représenter des concepts de contrôle d'accès de RBAC, Bell LaPadula et délégation de privilèges. Le métamodèle écrit en MOF est considéré comme un noyau qui peut être augmenté, dans un travail futur, avec de nouveaux éléments de métamodélisation de contrôle d'accès. Ils proposent des moyens d'exprimer seulement des ComAl simples pour régler les conflits entre des métamodèles de contrôle d'accès. Malgré la ressemblance apparente avec notre travail, Pavlich-Mariscal et al. se concentrent sur la génération de code d'une application, et ce en conformité avec une politique de contrôle d'accès, alors que notre recherche se concentre plus sur l'élaboration d'une approche globale intégrant les ACMM et leur mapping logique pour la vérification de propriétés.

Jajodia et al. [54] proposent un modèle englobant des hiérarchies de rôles, de sujets et d'objets. Le modèle proposé prend en compte l'historique des accès accordés, qui peut être utilisé pour exprimer des métamodèles de contrôle d'accès particuliers comme CW. Ce modèle permet la spécification de conflits entre les exigences de contrôle d'accès, mais avec une expressivité limitée basée sur la priorité du refus ou de l'autorisation, ou sur une priorité absolue d'une autorisation spécifiée.

6.5 Conclusion

L'élaboration du métamodèle de MACL vise la spécification de ce langage en s'appuyant sur une représentation visuelle de ses éléments. Cette représentation visuelle est utile pour réduire la complexité du langage qui comprend un bon nombre d'éléments. Dans ce sens, le métamodèle est aussi utile pour le concepteur du langage que pour un futur

utilisateur. Remarquons qu'avec notre métamodèle, on peut envisager de spécifier des ECAE en créant et en éditant des modèles.

Dans un cadre de métamodélisation tel que présenté dans la Section 3.2.1, le métamodèle d'un langage constitue une spécification de ce langage. Dans le cas de MACL, le métamodèle fournit son modèle abstrait et offre une base pour spécifier un mécanisme de détermination de décision projeté à la Section 4.5. Ceci est l'objet du chapitre suivant où l'on va associer (via un *mapping*) les éléments du métamodèle à leurs éléments textuels correspondants en MACL et à leurs expressions logiques correspondantes [55] [10].

[10] Le métamodèle muni de sa sémantique formelle a fait l'objet d'un article du Journal of Software 55. Abd-Ali, J., K. El Guemhioui, and L. Logrippo, *A Metamodel for Access Control Policies.* The Journal of Software, 2015. **10**(7): p. 784-797.

Chapitre 7 MACL : forme textuelle et logique de décision

Dans ce chapitre on va présenter la forme textuelle de MACL qui est basée sur le métamodèle IM présenté dans le chapitre précédent. On va voir aussi comment spécifier la sémantique des éléments de IM et de ses instances par un mapping vers le langage cible. Ceci permettra la spécification du mapping des énoncés de MACL.

7.1 Introduction

Comme mentionné dans la Section 5.4, notre approche se base sur un mapping vers le langage cible de tout énoncé de MACL. Le premier volet de ce chapitre explique comment produire ce mapping. Pour ce faire, on va spécifier le mapping des éléments de IM et de leurs instances. Cela se fera par un mapping qui relie chaque élément de modélisation d'une part à un énoncé du langage cible et d'autre part à sa forme textuelle dans MACL.

Rappelons ici que le métamodèle IM est une spécification de MACL, et que les énoncés de MACL représentent des instances de IM sous forme textuelle. Les moyens de spécification d'instances de IM par du texte constituent une spécification de la forme textuelle de MACL.

La Section 7.2 fournit le mapping des éléments de modélisation vers le langage cible, alors que la Section 7.33 fournit le mapping des éléments de modélisation vers la forme textuelle de MACL.

7.2 Spécifications formelles des éléments de IM et de leurs instances

Notre représentation des éléments de IM utilise un sous-ensemble d'UML. Une telle représentation par modélisation n'est ni complète ni exempte d'ambiguïté. Chacun des ACMM que nous avons présenté identifie des concepts et leurs associations, mais sans formellement déterminer comment une instance d'un ACMM choisit sa décision de contrôle d'accès en réponse à une demande d'accès. La logique de premier ordre (FOL) qui est la base de notre langage cible est connue pour être adaptée pour relier les individus à leurs types, pour spécifier des relations entre les individus et pour exprimer une logique de décision fondée sur les relations susmentionnées. Par conséquent, nous allons utiliser notre langage cible basé sur FOL, moyennant des prédicats dédiés et réutilisables, pour spécifier formellement nos éléments de méta-modélisation, leurs attributs et leurs associations. Par la suite, nous élaborerons la spécification formelle de la logique de décision en termes de prédicats.

Nos éléments de métamodélisation seront dotés de logique. Cela se fera en attachant des attributs spéciaux consignant des expressions logiques (clauses) aux éléments des ACMM. Ces expressions permettent la définition des règles applicables aux éléments du métamodèle et peuvent être utilisées pour évaluer les valeurs de vérité des autres prédicats. Nous appelons respectivement ces attributs spéciaux et leur type « Clause Attribute » et « Clause ». Une instance de Clause est une clause comme expression logique ou même une conjonction de clauses. Le côté gauche d'une clause est un prédicat (appelé la tête), tandis que son côté droit (appelée corps) est une conjonction de prédicats qui implique la vérité de la tête. Par exemple, considérons la clause suivante :

```
Human(X)  ← Walk(X), Talk(X).   (1)
```

La clause (1) indique que pour un individu X, lorsque les prédicats `Walk(X)` et `Talk(X)` sont évalués à *true*, alors `Human(X)` s'évalue à *true.* `Human(X)` est la tête de la clause, alors que le corps se compose de la conjonction logique de deux prédicats `Walk(X)` et `Talk(X)`. La virgule séparant des prédicats indique l'opérateur logique AND. Lorsqu'un prédicat ou une conjonction logique de prédicats est suivi par un point, et ne forme pas le corps d'une clause, cela signifie qu'il s'évalue à *true*.

Dans la suite de notre travail, on adopte les conventions suivantes pour les variables arguments de prédicats de la tête et du corps d'une clause :

- Toute variable, argument de prédicat du corps d'une clause qui n'est pas argument de la tête de cette même clause, est considéré comme quantifiée par le quantificateur existentiel ∃.
- Toute variable, argument de prédicat de la tête d'une clause, est considérée comme quantifiée par le quantificateur universel ∀.

Ainsi, on peut interpréter la clause (2) suivante par le fait que tout individu X qui marche et parle et qui a une mère et un père comme étant alors un humain :

```
Human(X)  ← Walk(X), Talk(X), Mother (X, M), Father (X, P). (1)

(1) la clause peut être réécrite:

∀ X, Human(X)  OR ¬ ( ∃ M, P  Walk(X), Talk(X), Mother (X, M), Father(X, P) )
```

Un **Clause Attribute** est un attribut dont la valeur est une clause ou une conjonction de clauses (séparées par des virgules qui signifient AND). Le nom d'un Clause Attribute est généralement le même que celui de la tête d'une de ses clauses. Un Clause Attribute indispensable pour notre langage est la clause ReturnedDecision qui renferme la logique de décision propre à l'élément du type DecisionSystem qui le contient. Ceci sera expliqué en détail dans la sous-section 7.2.2.

Selon le langage cible décrit à la Section 5.4.1, le corps d'une clause peut contenir des prédicats prédéfinis pour certains types, comme ceux de la comparaison de nombres rationnels, de vérification de cardinalité de listes, de fusion de listes, de concaténation de

chaînes de caractères, etc. On dit alors que « l'implémentation prend en charge la théorie d'un type » signifiant par-là que cette implémentation prédéfinit des prédicats qui vérifient l'appartenance d'un individu à ce type, et des opérations sur des individus de ce type. Dans la suite de notre travail, on considère que l'implémentation prend en charge au moins la théorie des nombres rationnels et les listes. Ainsi, on s'attend à l'utilisation des symboles <, >, [et] dans les corps des clauses pour comparer des nombres ou délimiter des listes d'éléments.

De plus, notre travail conceptuel part du principe que son implémentation future va utiliser des outils de la théorie *Satisfiability Modulo theory* (SMT) [39] et [56] qui nous permet la prise en charge des théories déjà mentionnées et qui permet d'utiliser l'opérateur de négation dans les corps des clauses logiques.

En outre, notre langage de modélisation est un sous-ensemble d'UML limité aux classes, attributs, associations, généralisations et spécialisations, mais augmenté du type Clause et de l'attribut Clause Attribute dont la sémantique et le type sont prédéfinis. Nous proposons un mapping vers le langage cible à partir de ce sous-ensemble d'UML. C'est souvent un mapping un à un direct que nous présenterons avec des exemples simples.

À noter ici que le mapping des éléments de notre sous-ensemble d'UML ne fournit pas la sémantique complète de chacun de ces éléments, mais il se limite à fournir dans le langage cible une représentation des aspects qui nous servirons dans notre utilisation des énoncés produits en langage cible. Ainsi, notre mapping ne comprend pas de représentation des opérations associées à une classe car on n'utilise pas d'opérations ni dans nos métamodèles ni dans leurs instances. Pour la même raison, notre mapping ignore des notions comme celle de classe abstraite, et des caractéristiques reliées à l'héritage comme le polymorphisme ou la redéfinition d'une méthode, etc.

7.2.1Le mapping

On distingue les clauses CLP avec la police spéciale « Courier new », et lorsque la confusion est possible, nous utilisons "/*" et "*/" pour délimiter les commentaires. Pour une meilleure lisibilité, nous utilisons aussi les caractères gras pour les têtes de clauses.

Tableau 7.2-1 Mapping des éléments de notre sous-ensemble d'UML

Éléments du sous-ensemble d'UML	CLP mapping
Déclaration de classe X a) avec une liste d'attributs att_i et leurs types Y_i b) pour chaque Clause Attribute f, Self désigne l'instance actuelle de la classe X. Lorsque le Clause Attribute est statique, il est désigné par sf dans ce tableau de mapping.	**IsType**(X). a) **DeclaredAttribute**(X, att_i, Y_i). /* Checks whether IsType(Y_i). */ b) /* f et sf sont mappés sans aucun changement à l'exception du remplacement de Self par l'instance actuelle lorsque la classe est instanciée. sf est spécifiée lors de la déclaration de la classe X et ce une seule fois pour toutes les instances de la classe X, mais sf a un mapping différent pour chaque instance de X. On remplace Self par l'instance courante de X. Ainsi, f et sf ont un mapping pour chaque instance de X. /* Nous verrons dans le prochain tableau que pour chaque instance de x de X, nous allons mettre le prédicat suivant à la valeur *true*: **IsOfType**(x, X). */
Association nommée LName entre deux classes X et Z	**DeclaredLink**(X, LName, Z). /* Des règles supplémentaires seront présentées hors de ce tableau pour prendre en considération les associations sans nom et la direction de l'association.*/
Association sans nom entre deux classes X et Z	**DeclaredLink**(X, Z).
Association nommée LName entre deux classes X et Z, mais avec une direction de X vers Z	**DeclaredDLink**(X, LName, Z).
Association sans nom entre deux classes X et Z, mais avec une direction de X vers Z	**DeclaredLink**(X, Z).

La classe X hérite de la classe Z	**IsOfType(O, Z)**←IsOfType(O, X). **DeclaredAttribute(X, Attname, Attype)**← DeclaredAttribute (Z, Attname, Attype). /* De même pour l'héritage des associations avec DeclaredLink */ **DeclaredLink**(X, LName, K) ← DeclaredLink (Z, LName, K). **DeclaredLink**(X, K) ← DeclaredLink (Z, K).

Le tableau ci-dessus montre les mappings des opérations de déclaration d'éléments du sous-ensemble d'UML. À ces mappings, on ajoute des règles qui sont établies une fois pour toutes, c.à.d. qu'elles ne sont pas à reproduire chaque fois qu'on déclare un élément. De telles règles établissent par exemple que : si des éléments X et Y sont associés, alors l'ordre n'est pas important et on peut dire que Y et X sont aussi associés.

Ainsi, les règles suivantes sont incluses dans notre mapping vers CLP :

```
DeclaredLink(Z, LName, X) ← DeclaredLink(X, LName, Z).
DeclaredLink(X, Z)← DeclaredLink(Z, X).

/* Une association avec nom est une association; ceci permet
d'examiner la connectivité indépendamment des noms des associations.
*/

DeclaredLink(Z, X) ← DeclaredLink(X, LName, Z).

/* une association directionnelle est aussi une association. */

DeclaredLink(X, LName, Z) ← DeclaredDlink(X, LName, Z).

/* Sauf précision de direction d'une association avec DeclaredDlink,
elle est considérée navigable dans les deux sens*/
```

En se basant sur le mapping susmentionné, l'étape suivante consiste à mapper les instances de classes et de leurs associations, et ce en prenant en considération les Clause Attribute. D'où le tableau suivant :

Tableau 7.2-2 Le mapping des instances des éléments du sous-ensemble d'UML

Instance d'élément de modélisation	CLP correspondant
X1 est une instance de la classe X a) chaque attribut att_i de X1 est mis à la valeur V_i. un $\leq$ index i $\leq$ nombre des attributs de X. b) Chacun des Clause Attribute est assigné une clause CLP comme valeur.	**IsOfType**(X1, X). a) /* Pour chaque attribut att_i */ **Attribute**(X1, atti , Vi). /* Condition devant être satisfaite */ **DeclaredAttribute**(X, Att_i, Y_i), **IsOfType**(V_i ,Y_i). b) /* Pour chaque Clause Attribute non statique, le mapping génère une nouvelle clause en remplaçant Self par l'instance actuelle de X1. Lorsque le Clause Attribute est statique la génération est faite une fois par type. */
Attribut de X multi-valué dont la valeur est une liste de valeurs de même type. Nommons le multiVatt, avec multiVatt = [v1, v2, .. vn]	**Attribute-list**(X, multiVatt, vi, i) /* pour i allant de 1 jusqu'à n */
Une association reliant deux instances de classe X1 et X2, avec éventuellement un nom d'association LName.	**Link**(X1, LName, X2). /* Condition devant être satisfaite : l'association est déclarée au niveau du métamodèle, en examinant le prédicat DeclaredLink(X1, LName, X2)*/ /* Au cas où l'association est sans nom, on génère: */ **Link**(X1, X2). /* A vérifier toujours si l'association est déclarée dans le métamodèle.*/

Une association avec direction reliant deux instances de classe X1 et X2, avec éventuellement un nom d'association LName.	**DLinK**(X1, LName, X2). /* À vérifier si l'association est déclarée au niveau du métamodèle, en examinant le prédicat DeclaredDLink(X1, LName, X2)*/ /* Au cas où l'association est sans nom on génère: */ **DLink**(X1, X2). /* A vérifier toujours si l'association est déclarée dans le métamodèle avec : DeclaredDlink(X1, X2). */

Règles toujours vraies

Comme dans le cas du mapping des déclarations des associations, certaines règles sont à appliquer pour prendre en charge les associations bidirectionnelles et l'éventuelle absence de nom d'association. Ces règles sont :

```
Link(X2, LName, X1)← Link(X1, LName, X2).
Link(X1, X2)← Link(X1, LName, X2).
Link (X2, X1)← Link(X1, X2).

// Une association directionnelle est une association

Link(X2, LName, X1)← DLink(X1, LName, X2).
Link(X1, X2)← DLink(X1, X2).
```

On ajoute à ces règles, d'autres règles concernant le mapping du langage ComLang mentionnés dans le tableau de mapping de la Section 9.5.2.

Ces règles logiques constituent une base de connaissances qui complète tout mapping de tout énoncé de MACL vers CLP.

Prédicats initialisés à *vrai* pour mots réservés prédéfinies

L'utilité du mapping logique d'une spécification de contrôle d'accès réside principalement dans sa capacité de permettre des décisions de contrôle d'accès ainsi que des vérifications de propriétés de la spécification de contrôle d'accès. Ainsi, il importe de mapper et

d'identifier des instances particulières des éléments du métamodèle qui forme l'entrée nécessaire pour solliciter une décision de contrôle d'accès. Ces instances particulières sont identifiées par des prédicats particuliers et éventuellement des mots réservés, et elles sont dressées dans le Tableau 7.2-3 :

Tableau 7.2-3 Instances particulières

Instances particulières et désignations MACL	Prédicats d'identification	Prédicats d'initialisation
Le système de décision DS racine de l'arbre ADA. Identifiant réservé : **Root**	**Root**(DS)	IsOfType(DS, DecisionSystem) AND (∀ OtherDS, i ¬Attribute-list (OtherDS, DS, i)).
La requête Q courante pour y répondre par une décision de contrôle d'accès. identifiant réservé : **CurrentQuery**	**CurrentQuery**(Q)	IsOfType(Q, Query)AND Attribute(Q,isCurrent, true).
Le sujet S demandant l'accès dans la requête courante. identifiant réservé : **QueryingSubject**	**QueryingSubject**(S)	IsOfType(S,Subject)AND IsOfType(Q,Query)AND Link(S,Q)AND CurrentQuery(Q).
L'objet O ciblé par la requête courante. identifiant réservé : **QueriedObject**	QueriedObject(QueriedObject)	IsOfType(O,Object) AND IsOfType(Q,Query)AND Link(O,Q)AND CurrentQuery(Q).

Le mode d'accès A demandé par le demandeur d'accès sur l'objet ciblé dans la requête courante. identifiant réservé : **QueryAccessMode**	QueryAccessMode(A)	`IsOfType( A, AccessMode) AND Link(A,Q)AND CurrentQuery(Q).`
Les modes d'accès read et write. Identifiants réservés (resp.) : **Read** et **Write**	Vérification du type AccessMode et des valeurs de l'attribut « name » qui doivent être : Read et Write.	`IsOfType (AccessMode, Read) And Attribute(Read, name, "Read") And IsOfType(AccessMode, Write) And Attribute (Write, name, "Write").`
Les décisions avec leurs identifiants réservés : **Permit, Deny, NotApplicable, Indeterminate.**	Vérification de du type Decision et des valeurs de l'attribut « name » qui doivent être : Permit, Deny, NotApplicable, Indeterminate.	`IsOfType(Permit , Decision) AND Attribute(Permit, name, "Permit") AND IsOfType(Deny , Decision) AND Attribute(Permit, name, "Deny")AND IsOfType(NotApplicable , Decision) AND Attribute(Permit, name, "NotApplicable") IsOfType(Indeterminate , Decision) AND Attribute(Permit, name, "Indeterminate").`

Ainsi pour permettre l'utilisation de mots réservés, les prédicats de la troisième colonne de ce tableau sont appliqués aux mots clés correspondant et sont initialisés à *true*. Cette initialisation représente une base de connaissance qui complète tout mapping de tout énoncé de MACL vers CLP. Comme exemple d'initialisation, la conjonction suivante assure la vérité de plusieurs prédicats d'initialisation:

Root(**Root**), **QueryingSubject**(**QueryingSubject**), QueriedObject(QueriedObject), QueryAccessMode(QueryAccessMode).

7.2.2Logique de décision et ReturnedDecision

Un DecisionSystem représente le concept d'un élément qui permet l'émission d'une décision, ainsi on l'a muni d'un Clause Attribute particulier qui consigne la spécification de la logique du choix de la décision. DecisionHandler, étant un DecisionSystem, possède un Clause Attribute ReturnedDecision.

Dans le cas d'un DecisionHandler, la valeur de ReturnedDecision est une clause ou une conjonction de clauses comprenant la clause ayant la forme suivante :

```
ReturnedDecision ← CompositePredicates.
```

CompositePredicates est une composition de prédicats portant sur les individus déjà créés par le mapping des instances appartenant à l'instance de l'IM qu'on crée pour spécifier les exigences de contrôle d'accès. Rappelons ici que la spécification de contrôle d'accès est une instance de IM selon notre approche. C'est pour cette raison que les éléments d'un ACMM sont associés à l'élément DecisionHandler avec l'association « uses »; ceci permet de naviguer d'une instance de DecisionHandler vers les autres éléments dans la même instance de ACMM. Cette navigabilité permet d'inclure dans CompositePredicates des références vers les autres éléments associés à l'instance de DecisionHandler. Cette navigabilité se base sur un mot clé dédié « Self ». Dans notre cas « Self » représente l'instance courante de DecisionHandler.

De plus, comme la logique de décision est à définir une fois pour toute pour chaque ACMM, une spécification de ReturnedDecision est à inclure dans chaque spécification d'un ACMM pris en charge par MACL. La suite de cette section détaille les spécifications des Clause Attribute ReturnedDecision des ACMM en commençant par CW.

ReturnedDecision du ACMM du Chinese Wall

Pour fournir un exemple d'illustration d'une application directe des règles de mapping à des éléments de l'IM, on commence par le mapping de certains éléments du noyau de contrôle d'accès. Ensuite, on enchaîne avec les éléments de l'ACMM de CW en termes de prédicats comme suit :

Un extrait du mapping de déclaration des éléments du noyau de CA

```
/*Déclaration des éléments Subject et Object: */

IsType(Subject)and IsType(Object).

/* Exemple de déclarations d'associations entre les éléments : */

DeclaredLink(Subject, Query),
DeclaredLink(Object, Query),
DeclaredLink(AccessMode, Query).

/* Au cas où une requête d'accès est à considérer on identifie les éléments de la
requête après les avoir instanciés avec : */

QueryingSubject(unSubject).
QueriedObject(unObject).
QueryAccessMode(unAccessMode).
CurrentQuery(laRequeteCourante).
```

Pour spécifier la logique de décision du CW dans le métamodèle de CW, on spécifie le Clause Attribute ReturnedDecision de CWdecisionHandler. On va donner pour chacune des deux variantes de CW et DCW présentées dans la Section 3.3 une spécification différente pour ReturnedDecision.

Un extrait du mapping de la déclaration du ACMM CW :

```
/* Déclaration du type CWdecisionHandler */
IsType(CWdecisionHandler).

/* CWdecisionHandler spécialise DecisionHandler  */
IsOfType(X, DecisionHandler) ←
                        IsOfType(X, CWdecisionHandler).

/* Déclaration de ConflictClass  et ConflictGroup  */
IsType(ConflictClass).
 // Un conlictClass hérite de ObjectsGroup
IsOfType( K, ObjectsGroup) ←  IsOfType( K, ConflictClass).
IsType(ConflictGroup).

/*  Associations declaration */
DeclaredLink(Subject, hasAccessed, ConflictClass).
DeclaredLink(ConflictGroup, composition, ConflictClass).
DeclaredLink(CWdecisionHandler, uses, ConflictGroup).
```

```
/* Le nom de l'association « uses » dans le dernier prédicat, reflète
le fait que les instances des éléments du ACMM de CW sont liées à
l'instance de DecisionHandler.

Logique pour émettre une décision de contrôle d'accès:
ReturnedDecision est un Clause Attribute de DecisionHandler. Il est
hérité par CWdecisionHandler de DecisionHandler, et il permet de
spécifier la décision à émettre par les instances de
CWdecisionHandler. « Self » désigne l'instance actuelle de la classe
CWdecisionHandler dans un attribut de la Clause. */

/*** Prise en charge de la règle simple du CW
Selon la règle simple de CW, la décision sera refusée si le
QueriedObject appartient à un ConflictClass différent d'un autre
ConflictClass déjà accédé par le queryingSubject ; avec les deux
classes associées à la même instance de ConflictGroup. */

ReturnedDecision  =
  ReturnedDecision (Self, Deny) ←

   Link(Self, uses, SomeConflictGroup),
   Link(SomeConflictGroup, composition, Class1),
   Link(SomeConflictGroup, composition, Class2),
   Link(Self, uses,Class1), Link(Self, uses,Class2),
   Link( QueriedObject, belongsTo,Class1),
   Link(queryingSubject, hasAccessed, Class2),
   ¬Link(queriedObject,belongsTo,Class2),
   QueryingSubject(queryingSubject),
   QueriedObject(queriedObject). ;

  /* La clause ci-dessus peut être lue de façon informelle en disant
  que l'instance actuelle de CWdecisionHandler « Self » retournera
  une décision Deny si toutes les assertions suivantes sont vraies :
  - il existe un SomeConflictGroup lié au « Self » ;
  - Ce même SomeConflictGroup a deux classes Class1 et Class2 ;
  - Class1 et Class2 sont associées à «Self» ;
  - l'objet ciblé appartient à Class1 ;
  - le sujet requérant a déjà accédé à un objet de Class2 ;
  - l'objet ciblé n'appartient pas à Class2.

  Cette dernière clause doit être générée pour chaque instance de
  CWdecisionHandler en remplaçant « Self » par l'instance considérée
  de CWdecisionHandler. */

  /*** Prise en charge de la propriété *
  La clause ci-dessus dont la tête est ReturnedDecision est toujours
  bonne, mais il faut ajouter qu'on doit empêcher qu'un sujet ayant
  accédé à une classe de conflit puisse écrire hors de cette classe.
  Ceci se formule par l'ajout de la clause suivante : */

  ReturnedDecision (Self, Deny) ←
   Link(Self, uses, SomeConflictGroup),
   Link(SomeConflictGroup, composition, Class1),
   Link(Self, uses,Class1),
   Link(queryingSubject, hasAccessed, Class1),
   ¬Link( QueriedObject, belongsTo,Class1),
```

```
   Link (CurrentQuery, Write).


/* Deny est à imposer à toute tentative d'écriture hors d'une classe
déjà visitée par un sujet.*/

// ****  Chinese Wall Dynamique  ****

ReturnedDecision  =
/* la clause suivante dont la tête est ReturnedDecision reste
identique à celle du CW, mais on ajoute une partie supplémentaire
pour l'aspect dynamique comme suit :  */

 // Première clause inchangée
  ReturnedDecision (Self, Deny) ←

   Link(Self, uses, SomeConflictGroup),
   Link(SomeConflictGroup, composition, Class1),
   Link(SomeConflictGroup, composition, Class2),
   Link(Self, uses,Class1), Link(Self, uses,Class2),
   Link( queriedObject, belongsTo,Class1),
   Link(queryingSubject, hasAccessed, Class2),
   ¬Link(queriedObject,belongsTo,Class2),
   QueryingSubject(queryingSubject),
   QueriedObject(queriedObject).,

 // Clause supplémentaire
  /* la clause suivante est ajoutée pour spécifier l'exigence
  particulière de l'aspect dynamique du Dynamic Chinese Wall */

 Link(ConflictGroup2, composition, Class1) ←
    Link(SomeSubject, hasAccessed, Class1), Link(SomeSubject,
    hasAccessed, Class2), IsOfType (SomeSubject, Subject),
    Link(ConflictGroup2, composition, Class2). ;

/* cette dernière clause assure qu'une Classe doit appartenir à tout
groupe de conflit accédé par un sujet qui a déjà accédé à cette
classe. */
```

Ayant détaillé l'explication du ReturnedDecision dans le cas du CW, on passe aux autres spécifications des Clause Attribute ReturnedDecision des autres ACMM dans les sous-sections suivantes.

ReturnedDecision de BLP

Pour se rappeler des éléments du ACMM BLP, on fournit son illustration (Figure 7.2-1) :

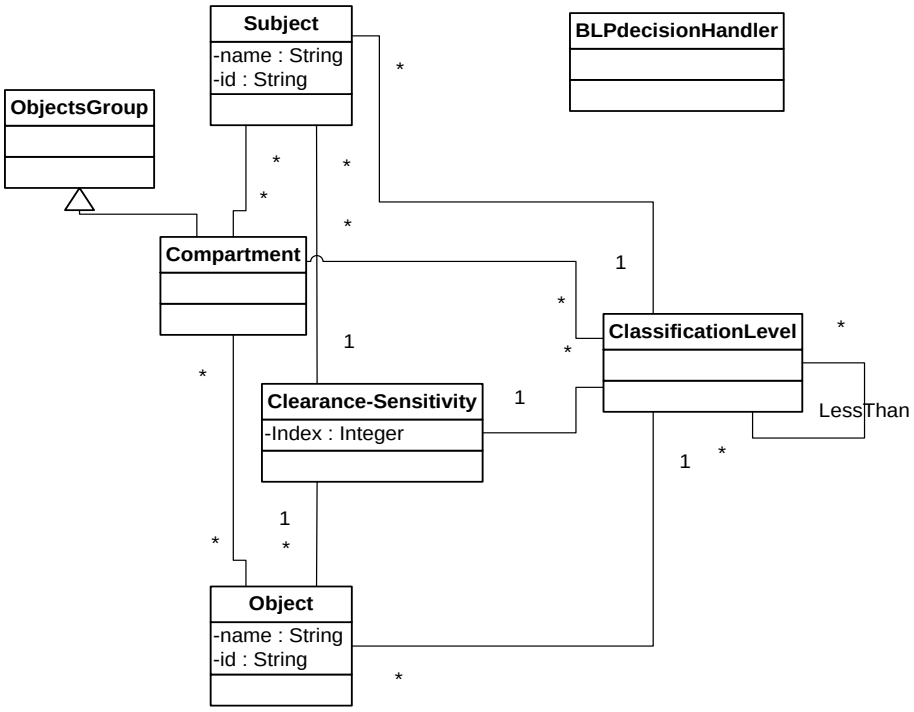

Figure 7.2-1 Métamodèle de contrôle d'accès BLP

Selon BLP l'accès est accepté si :

ClassificationLevel de l'objet est inférieur au ClassificationLevel du sujet.

Rappelons que le ClassificationLevel d'un sujet est défini par une Clearance-Sensitivity et un ensemble de compartiments qui lui sont associés. Il en est de même pour ClassificationLevel d'un objet. La comparaison des instances de ClassificationLevel se base sur la différence de compartiment et la comparaison des Clearance-Sensitivity. Ceci s'écrit en logique :

```
/* Le ClassificationLevel est inférieur à un autre si : (1) le NHS est
inférieur au NS ; ou (2) si la catégorie est inférieure.  On va spécifier
ces deux points avec les deux clauses suivantes :*/

// (1) Première clause spécifiant l'infériorité de niveau:

DLink  ( CL1, lessThan, CL2) ←
 IsOfType(    CL1,    ClassificationLevel)    ,    IsOfType(    CL2,
 ClassificationLevel) ,  IsOfType( ClSens1, Cleareance-Sensitivity),
 IsOfType( ClSens2, Cleareance-Sensitivity),
 Link (CL1, LlSens1),  Link (CL2, LlSens2),
```

```
  //  Condition de comparaison des Clearance-Sensitivity
 Attribute(ClSens1, index, index1),  Attribute(ClSens2, index, index2),
 index1 ≤ index2.

// (2) Deuxième clause d'infériorité de catégorie (compartiments):

DLink  ( CL1, lessThan, CL2) ←
 IsOfType(    CL1,    ClassificationLevel)    ,    IsOfType(    CL2,
 ClassificationLevel) ,
 IsOfType( Comp2, Compartment),  Link (Comp2, CL2) , ¬ Link (Comp2,
 CL1).

// On peut alors spécifier la condition de Deny par:

ReturnedDecision( Self, Deny) ←
 DLink  ( CL1, lessThan, Cl2), Link (QueriedObject, CL2),
 Link(QueryingSubject, CL1), Link (CL1, Self), Link(CL2, Self).

/* Ceci étant on peut spécifier la condition d'acceptation de demande
d'accès comme décision par défaut comme suit */

ReturnedDecision( Self, Permit) ←   ¬ ReturnedDecision( Self, Deny).
```

Le Clause Attribute ReturnedDecision de l'élément BLPdecisionHandler est spécifié par les trois clauses ci-haut. On remarque que les deux premières clauses ne comprennent pas « Self », donc elles représentent des règles dont la spécification est indépendante de l'instance courante, et donc on doit les générer une fois pour toute pour le type BLPdecisionHandler. On a pu envisager de spécifier ces deux premières clauses comme spécification d'un Clause Attribute statique de l'élément BLPdecisionHandler.

Notons ici que pour garder un niveau de lisibilité convenable, on a opté de spécifier la logique de décision de la variante de BLP qui ne considère pas la matrice de discrétion M mentionnée dans la Section 3.3.4. On peut tout de même ajouter au besoin cette spécification qui limite les accès aux modes d'accès Mij qui sont les éléments de la matrice M.

La matrice de discrétion peut être spécifiée comme une relation qui relie des triplets de (sujeti, objeti, Mij). On peut la représenter par le prédicat Dmatrix à trois arguments. On ajoute à ceci une relation d'ordre, notée LT, entre les modes d'accès pour indiquer si un mode d'accès n'implique pas un autre, par exemple Read n'implique pas Write. Ainsi,

`LT(Read, Write)` est *vrai*, alors que `LT (Write, Write)` et `LT(Read, Read)` sont tous les deux faux.

D'où l'on peut fournir une troisième condition suffisante de refus d'accès :

ReturnedDecision(Self, Deny) ← MatrixElement(QueryingSubject, QueriedObject, ma), LT(ma, QueryAccessMode).

ReturnedDecision de Biba

Pour se rappeler des éléments de ce ACMM, on fournit son illustration (Figure 6.2-5)

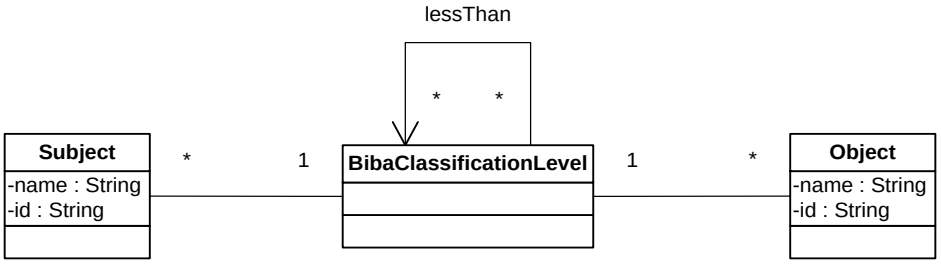

Figure 7.2-2 Métamodèle de contrôle d'accès Biba

La décision dépend de la comparaison des BibaClassificationLevel de l'objet et du sujet de la requête. Le Clause Attribute ReturnedDecision du Biba sera :

```
// La permission de lecture: no read down

ReturnedDecision( Self, Permit) ←

  /* Vérification des types des instances */

  IsOfType (BCLsubj, BibaClassificationLevel), IsOfType (BCLobj,
  BibaClassificationLevel),

  /*Les associations au DecisionHandler de Biba et aux sujets et objets
  de la requête.*/

  Link (BCLsubj, Self), Link (BCLobj, Self),
  Link (BCLsubj, QueryingSubject), Link (BCLobj, QueriedObject),

  // La comparaison des niveaux de Biba

  Attribute(QueryAccessMode, name, read),
  DLink (BCLsubj, lessThan, BCLobj).
```

```
/* La permission d'écriture seulement au même niveau Biba : Idem au
précédent mais avec un QueryAccessMode de Write et égalité de
BibaClassificationLevel.*/

ReturnedDecision( Self, Permit) ←

  // Vérification des types des instances

  IsOfType (BCL, BibaClassificationLevel),

  /*Les associations au DecisionHandler de Biba et aux sujets et objets
  de la requête.*/
  Link (BCL, Self),
  Link (BCL, QueryingSubject), Link (BCL, QueriedObject),

  // Même BCL pour objet et sujet
  // Mode d'accès demandé est Write

  Attribute(QueryAccessMode, name, Write).

// La condition de refus de la demande s'écrit comme réponse par défaut:

ReturnedDecision( Self, Deny) ← ¬ ReturnedDecision( Self, Permit).
```

ReturnedDecision de RBAC

Pour voir les éléments de cet ACMM on rappelle son illustration (Figure 7.2-3).

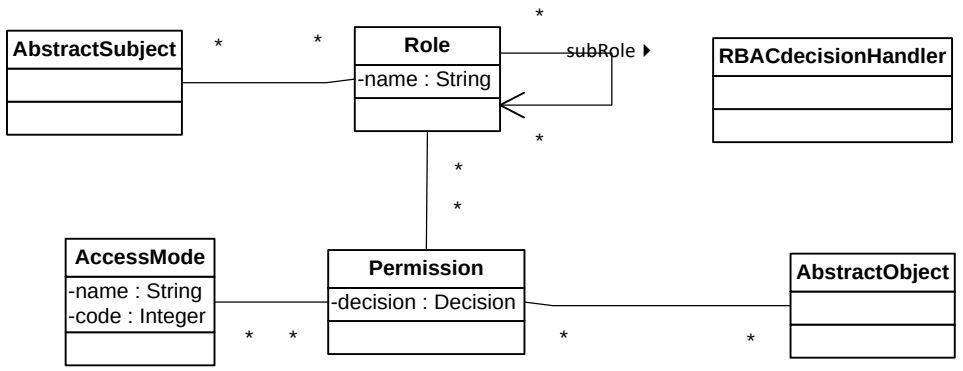

Figure 7.2-3 Métamodèle de contrôle d'accès RBAC

La décision dépend des permissions associées au rôle du sujet demandant l'accès. Le Clause Attribute ReturnedDecision du RBAC sera :

```
ReturnedDecision( Self, aDecision) ←
 // Les instances
  IsOfType(    RoleX,    Role),    IsOfType(    aDecision,    Decision),
  IsOfType(Perm, Permission),  Attribute( Perm, decision, aDecision),
```

```
// Les associations au DecisionHandler
  Link (Self, RoleX)    , Link (Self, Perm),
  // Les associations des éléments selon le métamodèle
  Link (Perm, QueriedObject), Link (Perm, QueryingSubject), Link(Perm,
  QueryAccessMode).
```

De plus, il faut ajouter une clause qui établit que l'association d'une permission à un groupe d'objets (sujets) implique une association de Permission à tout objet (respectivement sujet) de ce groupe :

```
Link (Perm, o) ← IsOfType(Perm, Permission),  IsOfType(  o, Object),
  IsOfType(GroupO,  ObjectGroup),  Link  (  o,  belongsTo,  GroupO),
  Link(Perm, GroupO).

Link (Perm, s) ← IsOfType(Perm, Permission),  IsOfType(  s,Subject),
   IsOfType(GoupS, ObjectGroup), Link ( s, belongsTo, GroupS), Link
   (Perm, GroupS).

// La decision par défaut est NotApplicable

ReturnedDecision( Self, NotApplicable) ←
   ¬(ReturnedDecision( Self, Permit), ¬ ReturnedDecision( Self, Deny).
```

ReturnedDecsion de l'ACMM de compartiments de Need-to-know

Pour voir les éléments de cet ACMM on rappelle son illustration (Figure 7.2-4).

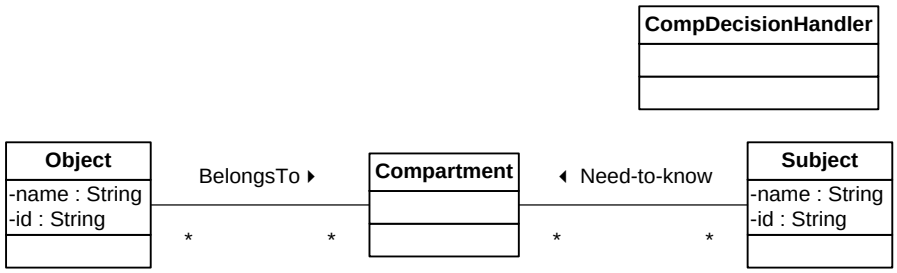

Figure 7.2-4 Metamodèle de contrôle d'accès de compartiments

La décision dépend de l'association du sujet requérant l'accès et de l'objet ciblé aux mêmes instances de Compartiment. Le Clause Attribute ReturnedDecision du CompDecisionHandler sera:

```
ReturnedDecision( Self, Deny) ←
  IsOfType(Comp, Compartment), Link (Comp, Self),
  ¬Link(Comp, QueryingSubject), Link(Comp, QueriedObject).
```

```
// Avec autrement une décision de Permit s'impose

ReturnedDecision( Self, Permit) ← ¬ ReturnedDecision( Self, Deny) .
```

ReturnedDecision du métamodèle de contrôle d'accès générique

Selon sa vocation décrite dans 6.2.8, la spécification de la logique de décision pour un GenACMM est à définir pour spécifier des exigences de contrôle d'accès qui ne cadrent pas dans les ACMM déjà pris en charge. Le Clause Attribute ReturnedDecision du DecisionHandler s'écrit en termes des éléments de noyau de contrôle d'accès décrit dans la Section 6.2.2.

7.3 Forme textuelle de MACL

Jusqu'ici on, on sait que selon notre approche et comme précisé dans la Section 5.3, MACL est un langage dont les énoncés représentent une composition d'instances de ACMM. De plus, on envisage de munir MACL des moyens permettant son extension par la déclaration de nouveaux ACMM. Ainsi, ayant en main le métamodèle IM spécifiant MACL et comprenant les ACMM pris en charge, la spécification de la forme textuelle de MACL consiste à préciser les éléments textuels correspondant aux éléments du métamodèle IM et ses instances. Cela fournit les moyens textuels pour les fins suivantes :

1. Manipulation des instances par :
    a. Création d'une instance de tout élément du métamodèle et initialisation des valeurs des attributs de cette instance;
    b. Ajout d'une association entre deux instances;
2. Déclaration de nouveaux éléments et de nouvelles associations, et ce au niveau des métamodèles (au cas où l'on veut ajouter un nouveau ACMM).
3. Spécification des ComAl.

Les points 1 et 2 seront détaillés dans cette section sous forme tabulaire, alors que le point 3 fera l'objet du Chapitre 9. Comme déjà mentionné dans la Section 7.2, les

éléments de modélisation de notre métamodèle IM appartiennent à un sous-ensemble d'UML. Le tableau ci-dessous établit une correspondance entre les éléments de ce sous-ensemble et leurs instances d'une part, et les éléments textuels de MACL d'autre part.

Table 7.3-1 Mapping: éléments de modélisation - Texte MACL

Élément de métamodélisation	Texte MACL

Déclaration du type X	DeclareType(X); // sans initialisation
	//Ou
	DeclareType X {} // pour initialisation

Déclaration d'attributs	DeclareAttribute(X, atti, typei); DeclareClauseAttribue(X, f,Clause); DeclareStaticClauseAttribute(X,fs,Clause); // Ou {Typei atti; Clause f; f= "f ← predicates expression"; Static Clause fs; fs = "fs ← predicates expression"; }
La classe X hérite de la classe Z	X inherits Z;

Declaration d'une association des classes Y et Z, nommée linkNameYZ ou sans nom	DeclareLink(Y, linkNameYZ, Z)
	// ou
	DeclareLink(Y,Z) // sans nom d'association

Élément de modèles (Instances)	**Texte MACL**
X1 est une instance de la classe X	CreateInstance(X, X1);

Assigner une valeur Vi à l'attribut yi de l'instance de classe X1.	Attribute(X1, yi, Vi) ;
	// Ou
	yi = Vi; /* entre deux accolades suivant directement le mot clé CreateInstance à la place du « ; » */

Assigner une valeur à un Clause Attribute f qui est une conjonction logique de plusieurs clauses.	f = une clause ou une conjonction de clauses séparées par des virgules ; /* N.B. : « Rappelons que lors du mapping vers CLP « Self » dans une spécification de Clause Attribute sera remplacé par le nom de l'instance courante.*/
De même pour l'assignation des valeurs des Static Clause Attribute	/* Idem la cellule adjacente ci-haut. */
Association de deux instances de classes X1 et X2 avec éventuellement un nom d'association Lname.	Link(X1, Lname, X2); /* et s'il n y a pas de nom d'association on génère : */ Link(X1, X2);
Association directionnelle entre deux instances de classes X1 et X2 avec éventuellement un nom d'association Lname.	DLink(X1, Lname, X2); /* et s'il n y a pas de nom d'association on génère: */ DLink(X1, X2);

On peut remarquer que les éléments de modélisation sont à l'origine de tout mapping vers le langage cible CLP et de tout énoncé textuel de MACL. Ainsi, on peut déduire un mapping entre les énoncés MACL et leur sémantique en CLP en passant par les éléments de modélisation comme langage intermédiaire. Plus précisément, pour mapper un énoncé textuel de MACL vers CLP, on le fait en deux étapes : (1) D'après Table 7.3-1 on mappe l'énoncé textuel de MACL vers des éléments de modélisation; (2) on mappe ces derniers éléments de modélisation vers CLP et ce d'après les tableaux 7.2-1 et 7.2-2.

7.3.1 Moyens de simplification au niveau de MACL

De plus, pour alléger et simplifier un énoncé MACL où l'on a généralement un bon nombre d'objets et de sujets, on a muni notre syntaxe d'alias qui permettent de prévenir certaines répétitions de lignes similaires et ce comme suit.

1. On utilise les caractères spéciaux « ![» et «]! » pour délimiter en argument un ensemble d'éléments pour lesquels la ligne de spécification est à répéter et ce selon l'exemple suivant :

CreateInstance(X , ![o1, o2, o3]!);

Remplace:

CreateInstance(X , o1);

CreateInstance(X , o2);

CreateInstance(X , o3);

Et d'une façon générale:

UnCertainPredicat(argument1, …, **![element1, .., elementk]!**, .. argumenti, .. , argumentn);

Remplace :

UnCertainPredicat (argument1, …, **element1**, .. argumenti, .. , argumentn);

….

UnCertainPredicat (argument1, …, **elementk**, .. argumenti, .. , argumentn);

2. On peut utiliser la même technique de l'alias précédent en utilisant plutôt un nom de groupe d'éléments délimités par « ![» et «]! ».
   Ex. si O1, O2 et O3 sont trois éléments formant un groupe d'objets OG de type ObjectsGroup ou SubjetsGroup, alors :
   Link(X, ![OG]!);

Remplace : Link(X, O1); Link(X, O2); Link(X, O2);

7.3.2Exemple de spécification de CW

Pour spécifier en MACL une application du Chinese Wall, il faut créer une instance du ACMM du Chinese Wall, ce qui revient à spécifier des instances des éléments et des associations du dit métamodèle illustré dans la Figure 7.3-1.

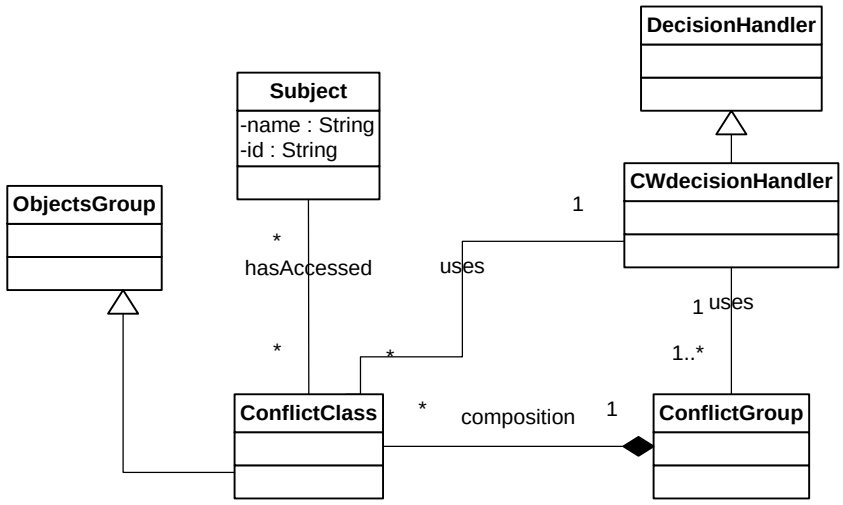

Figure 7.3-1 Le métamodèle du Chinese Wall

On va reprendre l'exemple présenté dans la Section 3.3.2 et qui sera référé par CW1 dans la suite de notre travail. On va fournir la spécification de CW1 en MACL et ce en créant les énoncés MACL correspondant aux créations et associations d'instances d'éléments puisés dans le ACMM du Chinese Wall.

1. Deux ConflictClass : ClassPromutuel et ClassSunLife formées, respectivement, de deux ensembles d'objets {Ob1, …, Ob5} et {Ob6, …, Ob10}.

```
// Création de 10 objets
CreateInstance (Object, Ob1); ….

CreateInstance (Object, Ob10);
/* les lignes précédentes peuvent être réécrite en utilisant les
délimiteurs « ![ » et « ]! » :
CreateInstance (Object, « ![Ob1, …,Ob10 ]! » ); */
// Création d'une classe d'un groupe de conflit
CreateInstance (ConflictClass, ClassPromutuel);
CreateInstance (ConflictClass, ClassSunLife);
CreateInstance (ConflictClass, ClassDesjardins);
```

```
// Appartenance des objets aux classes

Link (Ob1, belongsTo, ClassPromutuel); ….
Link (Ob5, belongsTo, ClassPromutuel);

Link (Ob6, belongsTo, ClassSunLife); ….
Link (Ob8, belongsTo, ClassSunLife);

Link (Ob9, belongsTo, ClassSunLife);
Link (Ob10, belongsTo, ClassSunLife);
```

Rappelons que notre langage MACL permet la spécification d'ensembles d'objets et de sujets en utilisant les éléments ObjectsGroup et SubjectsGroup initialement définis comme éléments du noyau de contrôle d'accès. Par exemple, la spécification de l'ensemble d'objets ClassPromutuel est obtenue en déclarant ClassPromutuel en tant qu'instance de ConflictClass qui est une spécialisation de ObjectsGroup, et en définissant cinq associations de ClassPromutuel aux cinq objets Ob1, …, Ob5. Cette définition répétitive d'associations de même nature est plutôt fastidieuse, et n'a été utilisée que dans un souci de simplification de la présentation du premier exemple de spécification MACL. Une alternative qui permet l'établissement de plusieurs associations en une seule instruction est déjà prévue et s'énonce comme suit :

Link (! [Ob1, Ob2, Ob3, Ob4, Ob5]!, belongsTo, ClassPromutuel) ;

2. Un groupe de conflit, nommé ConflictGr, est formé des classes d'objets ClassPromutuel, ClassSunLife et d'une troisième classe dénotant une troisième compagnie d'assurance ClassDesjardins.

```
CreateInstance (ConflictGroup, ConflictGr);

Link(ConflictGr, composition, ClassPromutuel);

Link(ConflictGr, composition, ClassSunLife);

Link(ConflictGr, composition, ClassDesjardins);
```

3. L'application du Chinese Wall est spécifiée en créant une instance du métamodèle de Chinese Wall, en associant son DecisionHandler au groupe de conflit.

```
CreateInstance (CWdecisionHandler, CW1);
Link(ConflictGr, uses, CW1);
Link(ClassSunLife, uses, CW1);
Link(ClassPromutuel, uses, CW1);
Link(ClassDesjardins, uses, CW1);
```

4. Plusieurs sujets sont définis : Tom, Sam, Pamela.

```
CreateInstance (Subject, Tom){id = 1;}
CreateInstance (Subject, Sam){id = 2;}
CreateInstance (Subject, Pamela){id = 3;}
```

5. L'historique d'accès aux classes indique que Tom a accédé à Ob1 et que Sam a accédé à Ob6 et Ob10.

```
Link( Tom, hasAccessed, Ob1);
Link( Sam, hasAccessed, Ob6);
Link( Sam, hasAccessed, Ob10);
```

La spécification MACL précédente définit les classes de conflit et l'historique des accès à ces classes. Le mapping vers CLP de cet énoncé MACL, comprend la logique de décision du CWdecisionHandler. Ces connaissances logiques ajoutées à celle d'une requête définissant le sujet requérant, l'objet ciblé et le mode d'accès vont permettre d'examiner la valeur de vérité des prédicats ReturnedDecision du CWdecisionHandler. Ainsi on peut fournir des décisions de contrôle d'accès d'après notre spécification d'une instance du Chinese Wall spécifiée en MACL.

Rappelons que la logique de contrôle d'accès adoptée dans l'instance créée de l'élément ChineseWall est prédéfinie une fois pour toutes dans la Section 7.2.2 lors de la déclaration de l'élément CWdecisionHandler du métamodèle de Chinese Wall avec son Clause Attribute ReturnedDecision.

7.4 Conclusion

Notre approche nous a permis d'intégrer dans les éléments d'un métamodèle la sémantique de ce métamodèle et de ses instances ainsi qu'un mapping vers le langage que ce métamodèle définit. Cette approche est applicable à des domaines autres que le contrôle d'accès et représente un accomplissement au niveau de l'élaboration des langages basés sur la métamodélisation [57] [11].

Dans ce chapitre, on a vu comment une application d'un ACMM comme le CW peut être spécifiée dans MACL. De plus, on a vu que cette spécification correspond à des assertions en CLP qui permettent de fournir une décision en réponse à une demande d'accès. Le chapitre suivant présente une implémentation permettant d'éditer les instances d'un ACMM, de les mapper en CLP, et de déterminer les décisions de contrôle d'accès. La spécification de l'intégration de plusieurs instances de ACMM selon un ADA fait l'objet du Chapitre 9.

[11] Cette approche a fait l'objet d'une publication d'un article dans la conférence MODELSWARD 2015 57. Abd-Ali, J., K. El Guemhioui, and L. Logrippo, *Metamodelling with Formal Sematics with Application to Access Control Specification*, in *MODELSWARD*. 2015: Angers, France. p. 354-362.

Chapitre 8 Un prototype d'implémentation pour CW

Nous avons défini dans le chapitre 6 le métamodèle IM dont les instances représentent des spécifications de contrôle d'accès. Ainsi, on peut exprimer les exigences de contrôle d'accès en créant et éditant des modèles instances de IM. Ces modèles doivent se conformer à IM et ce en se limitant aux instances d'éléments et d'associations, définies dans ce métamodèle et en satisfaisant les règles d'instanciation établies dans ce dernier. D'où l'importance de développer un outil qui assiste l'édition de ces modèles conformément au métamodèle IM.

D'autre part, nous avons défini dans le chapitre 7 les mappings des éléments des modèles instances de IM moyennant un mapping. D'où l'importance d'obtenir les mappings des modèles en automatisant l'application du mapping. Ainsi, nous projetons d'ajouter à l'outil d'édition une fonctionnalité qui automatise la génération de code écrit en CLP. Nous rappelons ici que selon notre approche et comme mentionné dans la Section 5.4, le résultant de l'application du mapping fournit une base de connaissances écrite en CLP qui permet de déterminer les décisions de contrôle d'accès en fonction des données des requêtes d'accès. Nous rappelons aussi le deuxième objectif principal de validation et de vérification mentionné dans la Section 4.9. Ainsi, une fonctionnalité de traitement du code généré en CLP, permet d'expérimenter la matérialisation de ces derniers objectifs

de détermination de décisions et de vérification de propriétés des spécifications de contrôle d'accès. Ainsi, nous projetons d'élaborer un prototype implémentant les fonctionnalités mentionnées ci-haut en appliquant et exploitant la sortie en CLP du mapping.

Dans ce chapitre, nous visons à montrer un exemple illustrant l'application de notre approche moyennant un exemple concret de spécifications de contrôle d'accès. L'exemple choisi est celui du Chinese Wall, CW, décrit à la Section 7.3.2. Nous présentons un prototype de CW qui permet une illustration de la spécification des exigences de contrôle d'accès en modèles. L'exemple spécifique utilisé est celui du CW. Ce prototype implémente aussi un éditeur de modèles instances du métamodèle du CW. Il permet l'automatisation de la validation de conformité des instances au métamodèle, la génération des mappings CLP des modèles instances et, conséquemment, la vérification de propriétés et la détermination des décisions de contrôle d'accès.

8.1 Introduction

La représentation visuelle en modèles des spécifications de contrôle d'accès permet de réduire la complexité de cette tâche. Le premier volet d'implémentation pour permettre cette visualisation est de produire un outil (« l'éditeur ») pour l'édition de modèles en conformité avec le métamodèle spécifiant MACL. Le deuxième volet consiste à ajouter à cet outil un composant pour la génération du mapping logique des modèles édités (« le générateur »). Le troisième volet consiste en une adaptation du code généré aux exigences syntaxiques particulières d'un outil de programmation logique. Ceci va permettre d'exploiter la sortie du mapping pour déterminer automatiquement les décisions de contrôle d'accès et pour vérifier des propriétés de politiques de contrôle d'accès représentées sous forme de modèles.

L'implémentation du modèle de CW que nous présenterons ici est basée sur le modèle de CW présenté à la Section 6.2.3.

8.2 Choix d'implémentation

Dans le but de développer l'éditeur doté de fonctionnalités de générateur, on a utilisé la plate-forme de Mia-Studio[12] intégré dans l'environnement de développement Eclipse[13]. Cette plate-forme permet la spécification de métamodèles en utilisant le langage Ecore (Figure 8.2-1) qui représente un noyau de MOF, et qui est le langage de spécification de métamodèles pris en charge par l'outil Eclipse. Ainsi, on a spécifié en Ecore le métamodèle du CW et des éléments du noyau de notre métamodèle. Ce métamodèle spécifié en Ecore est illustré dans la Figure 8.2-2 et la Figure 8.2-3 . Après importation de ce métamodèle dans Mia-Studio, on a ajouté un éditeur d'instances de ce métamodèle et ce en tant que Plug-In d'Eclipse.

Notons que Ecore comprend tous les éléments nécessaires pour spécifier notre métamodèle qui est formé de types ayant des attributs et des associations en plus de son utilisation des types primitifs de Ecore.

On a ensuite développé un générateur de code CLP basé sur le métamodèle importé. Ce générateur de code accepte en entrée une instance du métamodèle importé et produit en sortie son mapping en code CLP. On a conçu le générateur de code de façon qu'il applique les mappings déjà introduits à la Section 7.2.1.

Mentionnons que Mia-Studio offre la possibilité de spécifier les règles de mapping séparément pour chaque élément du métamodèle. On peut par la suite créer plusieurs scénarios pour appliquer les mappings déjà spécifiés en une ou plusieurs itérations. Le langage de spécification de génération de code est Java augmenté d'une librairie permettant la manipulation des instances des métamodèles écrits en Ecore et importés dans Mia-Studio. Ceci nous permet de spécifier les mappings avec une expressivité qui prend avantage des librairies Java, y compris celles offrant des méthodes de traitement des chaines de caractères.

[12] http://www.mia-software.com/produits/filiere-java/mia-studio/
[13] http://www.eclipse.org/

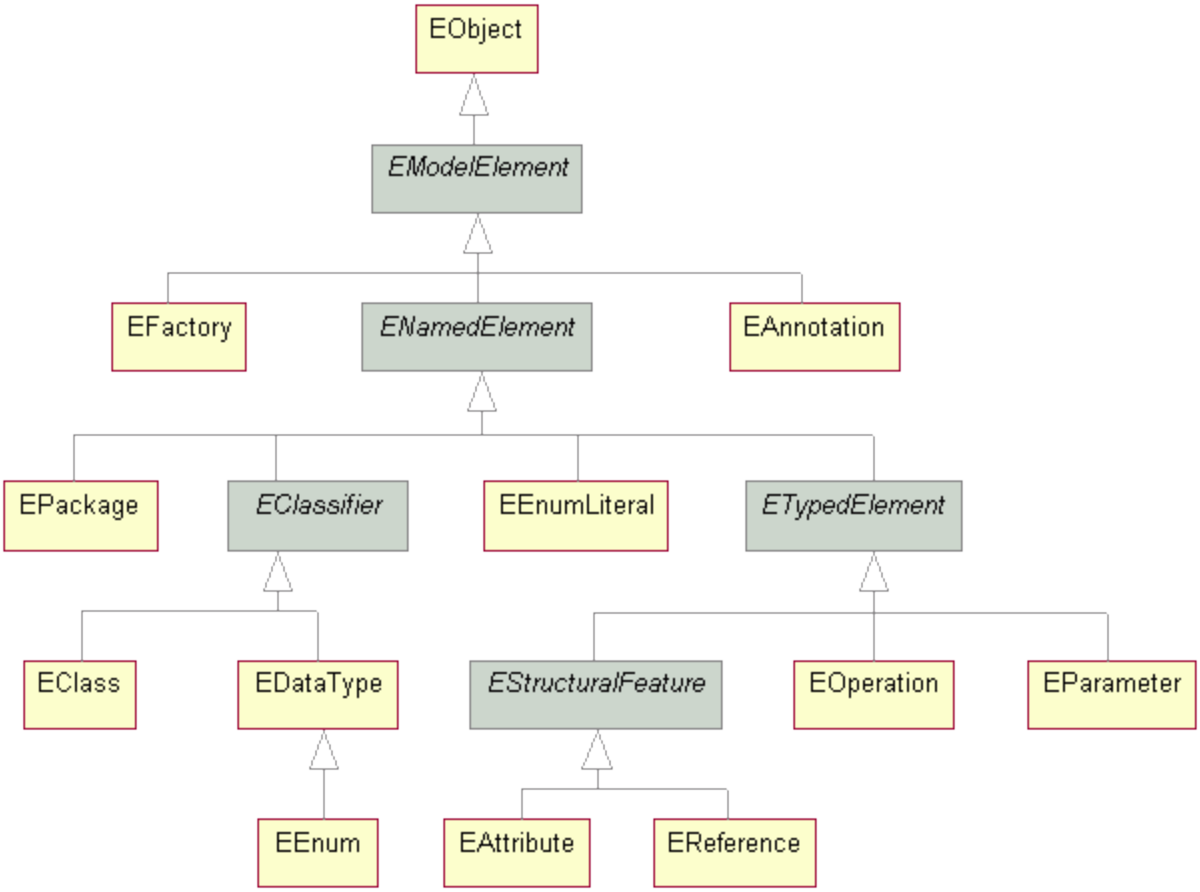

Figure 8.2-1 Le modèle Ecore qui permet de spécifier des métamodèles intégrables dans Eclipse.

La sortie du générateur consiste en un fichier de texte comprenant des clauses logiques et des commentaires. Pour exploiter ce code, on a opté pour l'utilisation de l'outil de programmation logique SWI-Prolog[14]. SWI-Prolog permet, après compilation du fichier texte généré, de vérifier la satisfiabilité de prédicats comprenant des variables. Ces prédicats sont dits goals selon la terminologie de la programmation logique. La vérification d'un goal par SWI-Prolog produit une sortie qui identifie les valeurs possibles des variables pour lesquelles ce goal sera *vrai*. Le choix d'un prédicat à vérifier va permettre, entre autres, de déterminer une décision de contrôle d'accès. Le prédicat choisi pour cette fin est de la forme *returnedDecision(UneInstanceCWDecisionHandler, UneVariableReprésentantUneDecision)*. Ainsi, en choisissant ce prédicat à vérifier comme

14 www.swi-prolog.org

goal, SWI-Prolog va afficher tous les couples d'instances de *CWdecisionHandler* et de décisions associées pour lesquels le goal est *vrai*.

Notons que pour éviter des collisions de noms et pour respecter les conventions des outils utilisés, on a renommé l'élément *Object* de notre métamodèle en *Objectt* avec deux « t » et on a renoncé à capitaliser les noms de certaines classes et instances dans le mapping CLP (par exemple, *jamal* et *tom* représentent deux sujets). Pour les mêmes raisons, on a remplacé *link* par *linkk*.

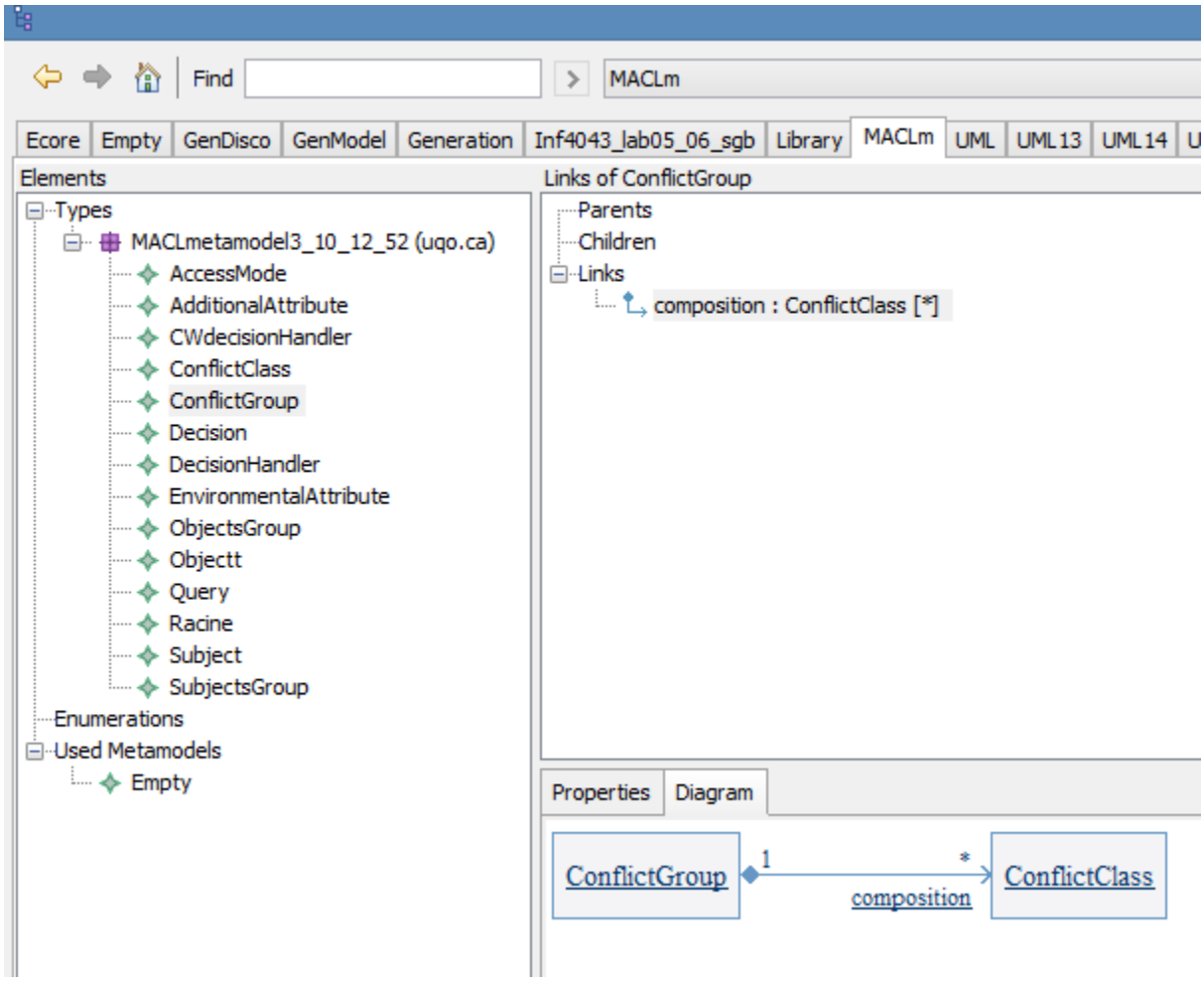

Figure 8.2-2 Notre métamodèle illustré dans la fenêtre Model Browser montrant graphiquement une association sélectionnée.

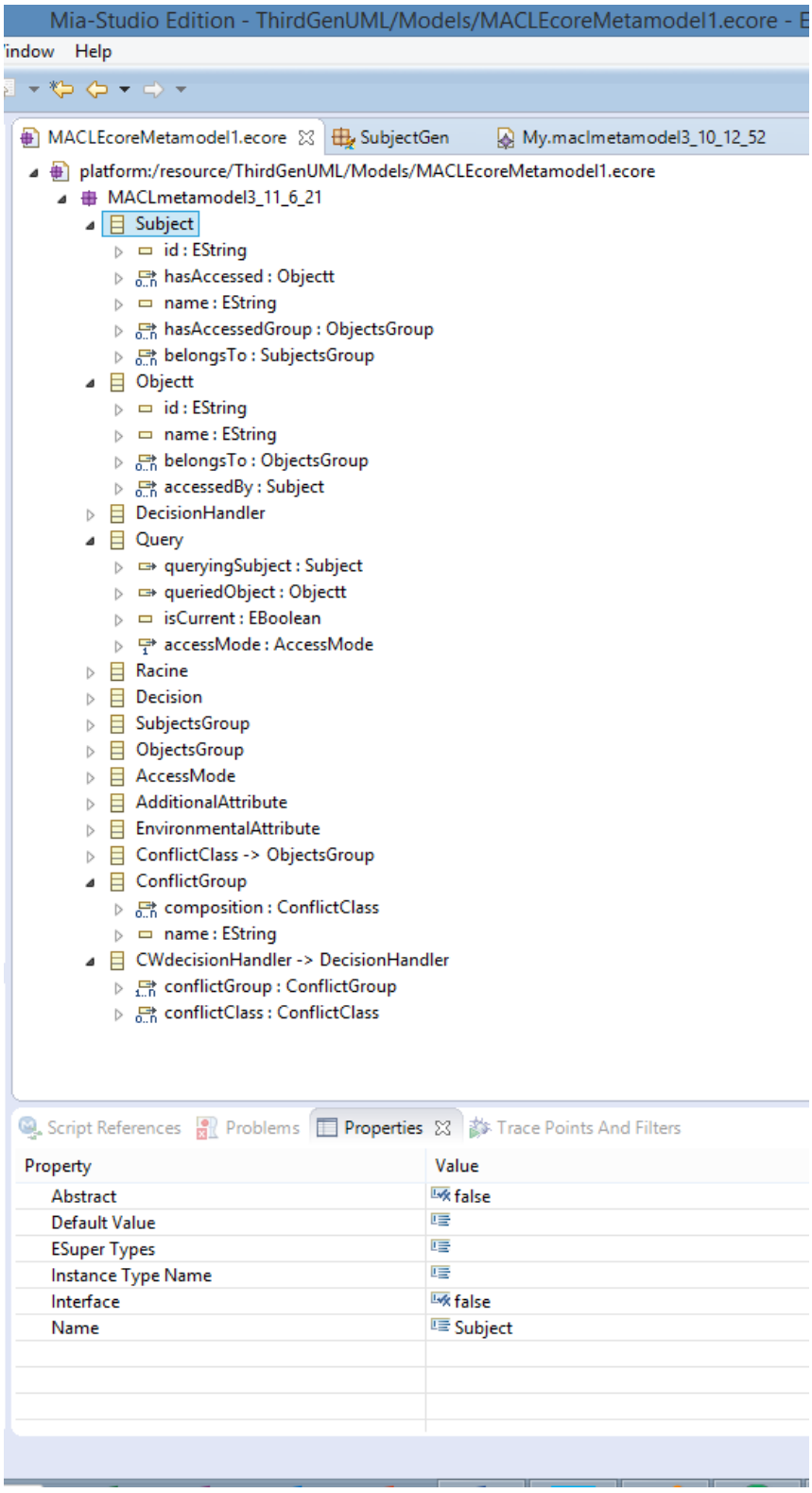

Figure 8.2-3 Notre métamodèle dans une vue Mia-Studio Edition qui permet son édition

8.2.1Limitations de notre implémentation

Remarquons que le métamodèle illustré à la figure ci-dessus ne comprend pas tous les éléments du métamodèle de notre langage. C'est parce que notre implémentation est développée dans une perspective d'expérimentation de l'atteinte certains objectifs offrant aussi une illustration de faisabilité de ces objectifs pour les ACMM non implémentés. Il importe alors de bien définir les limites de notre implémentation et ce comme suit :

- Le métamodèle ACMM de CW est le seul métamodèle pris en charge, en plus des éléments du noyau de contrôle d'accès;
- La prise en charge d'une combinaison de plusieurs instances de ACMM dans un arbre de décision ascendante ne fait pas partie de notre implémentation.

Ajoutons à ceci les limitations qui sont inhérentes aux outils Mia-Studio et SWI-Prolog. Principalement, la version que nous avons utilisée de Mia-Studio nous impose une limite sur le nombre d'éléments du métamodèle pris en charge. SWI-Prolog a aussi ses propres limites comme l'arité des prédicats et celle de l'évitement de collision entre les noms des prédicats de l'utilisateur et ceux de l'implémentation de l'outil lui-même; une explication plus détaillée des limitations de SWI-Prolog est disponible sur le lien offert en pied de cette page[15].

Notre implémentation, montre l'aspect de la spécification visuelle des instances des éléments des ACMM, mais elle permet aussi de remarquer la limitation de l'absence de moyen de spécification visuelle de la logique de décision, car on n'a pas préparé aucun moyen d'édition graphique (par des dessins) d'énoncés CLP ou FOL. La logique décision devrait être spécifiée avec du code CLP lors de l'élaboration de chaque ACMM pour être transmise automatiquement aux instances de DecisionHandler de chaque ACMM. La logique de décision peut être aussi spécifiée avec du code CLP à associer directement à l'instance de DecisionHandler quand on instancie le ACMM dit générique mentionné dans la Section 6.2.8.

D'autre part, il importe de noter que la compagnie Mia-Software nous a gracieusement accordé une licence d'utilisation de son outil pour une durée limitée de trois mois. Pour cette raison et pour des raisons inhérentes à la nécessité de plusieurs étapes manuelles pas simples pour assurer l'interopérabilité entre Mia-Studio et SWI-Prolog lors de chaque exécution, nous n'avons pas pu rendre notre implémentation disponible sur le Web.

[15] http://www.swi-prolog.org/pldoc/man?section=limits

Dans la Section suivante, nous allons expérimenter des fonctionnalités de vérification et de validation et de détermination de décision de contrôle d'accès correspondant à des objectifs déclarés au Chapitre 4 et ceci sera présenté avec un exemple qui respecte les limitations déjà mentionnées.

8.3 Exemple d'utilisation du prototype

Nous reprenons l'exemple de la Section 7.3.2 de la spécification de contrôle d'accès où l'on considère une application du métamodèle du CW. Dans cet exemple, on veut éviter des conflits d'intérêt qui pourraient arriver au cas où un sujet accède à des informations de plus d'une compagnie d'assurance en compétition sur le marché.

Ainsi, la spécification de notre exemple d'illustration, visible sur la Figure 8.3-1, consiste principalement en :

- Une instance de *ConflictGroup* représentant le groupe de conflit des compagnies d'assurance;
- Trois classes de conflit associées à l'instance de *ConflictGroup* : *classDesJardins*, *classSunLife*, *classPromutuel*, *classIntact*;
- Une instance de *CWdecisionHandler* qui est associée aux instances précédentes à travers l'association *uses*.
- Six instances *d'Object* : *obj1Promutuel, obj4Promutuel* et *obj5Promutuel* associés à *classPromutuel*; *obj2DesJardins, obj3Sunlife* et *obj6Intact* associés respectivement à *classDesJardins*, *classSunLife* et *classIntact*.
- Trois instances de *Subject* : *jamal*, *tom* et *sam* ayant accédé respectivement à *obj1Promutuel*, *obj2Desjardins* et *obj3Sunlife*.
- La requête considérée exprime que *tom* veut accéder en mode lecture à *obj1Promutuel*.

L'éditeur permet de créer les différents éléments du modèle en conformité avec le métamodèle. Cette vérification de conformité, qu'on nomme validation du modèle par rapport au métamodèle, a été accomplie moyennant une simple commande du menu principal de l'éditeur. Ainsi, on a une collection de types bien définis à instancier. La multiplicité des instances aux deux bouts d'une association et les types des valeurs des attributs sont contraints à être conformes au métamodèle.

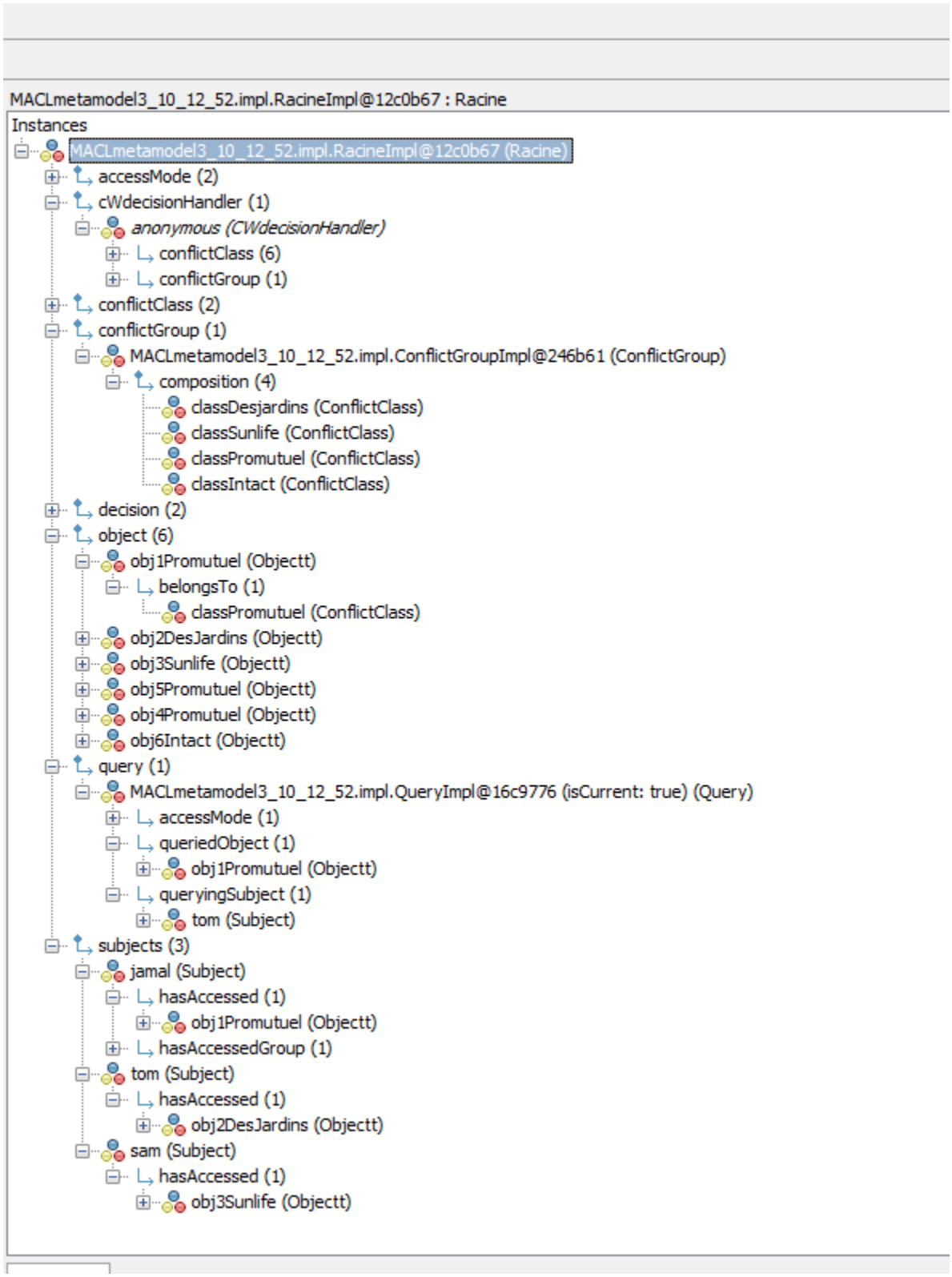

Figure 8.3-1 Modèle représentant une instance du métamodèle CW

Le modèle de notre exemple a été fourni en entrée au générateur de code qui a produit en sortie son mapping logique dans un fichier qu'on a appelé result2.pl et dont un extrait est annexé à ce chapitre.

L'étape suivante consiste à ouvrir ce fichier texte avec l'application SWI-Prolog. Après compilation de ce fichier par un éditeur intégré à SWI-Prolog, l'outil est prêt à prendre en charge la vérification des goals. Pour chaque goal, l'outil va chercher à trouver des affectations de valeurs aux différentes variables qui pourraient assurer la satisfaction du goal.

La Figure 8.3-2 illustre les traitements suivants :

1) La proposition du goal *returnedDecision(QuelDecisionHandler, QuelDecision)* qui a résulté en un couple représentant la seule décision possible qui est « deny » et ce pour le système de décision *cWdhcf8b* [16] qui est une instance du *CWdecisionHandler*. À noter que selon notre implémentation on a convenu que tout identifiant d'une instance de *CWdecisionHandler* doit commencer par le préfix *cWdh* suivi de quatre autres caractères afin de permettre l'éventualité de prise en charge de plusieurs instances de *CWdecisionHandler*.
2) La proposition des *goals queryingSubject(QuelSujet)* et *queriedObject (QuelObjet)* et ce dans le but de chercher les éléments de la demande de contrôle d'accès considérée. La réponse de l'outil assure que c'est tom qui demande l'accès à obj1Promutuel.
3) Les *goals* restants permettent de vérifier que tom a déjà accédé à *obj2DesJardins* appartenant à la classe *classDesJardins*. Mais *obj1Promutuel* ciblé par la requête appartient à *classPromutuel*. Ainsi, on remarque que tom est en train de tenter de franchir la muraille de Chine qui doit être élevé entre *classPromutuel* et *classDesJardins*. Ceci explique bien pourquoi, selon le traitement du *goal* du point 1), la seule valeur de QuelDecision qui vérifie le *goal returnedDecision (QuelDecisionHandler, QuelDecision)* est *deny*.

[16] Ce nom d'identifiant est automatiquement généré par l'outil.

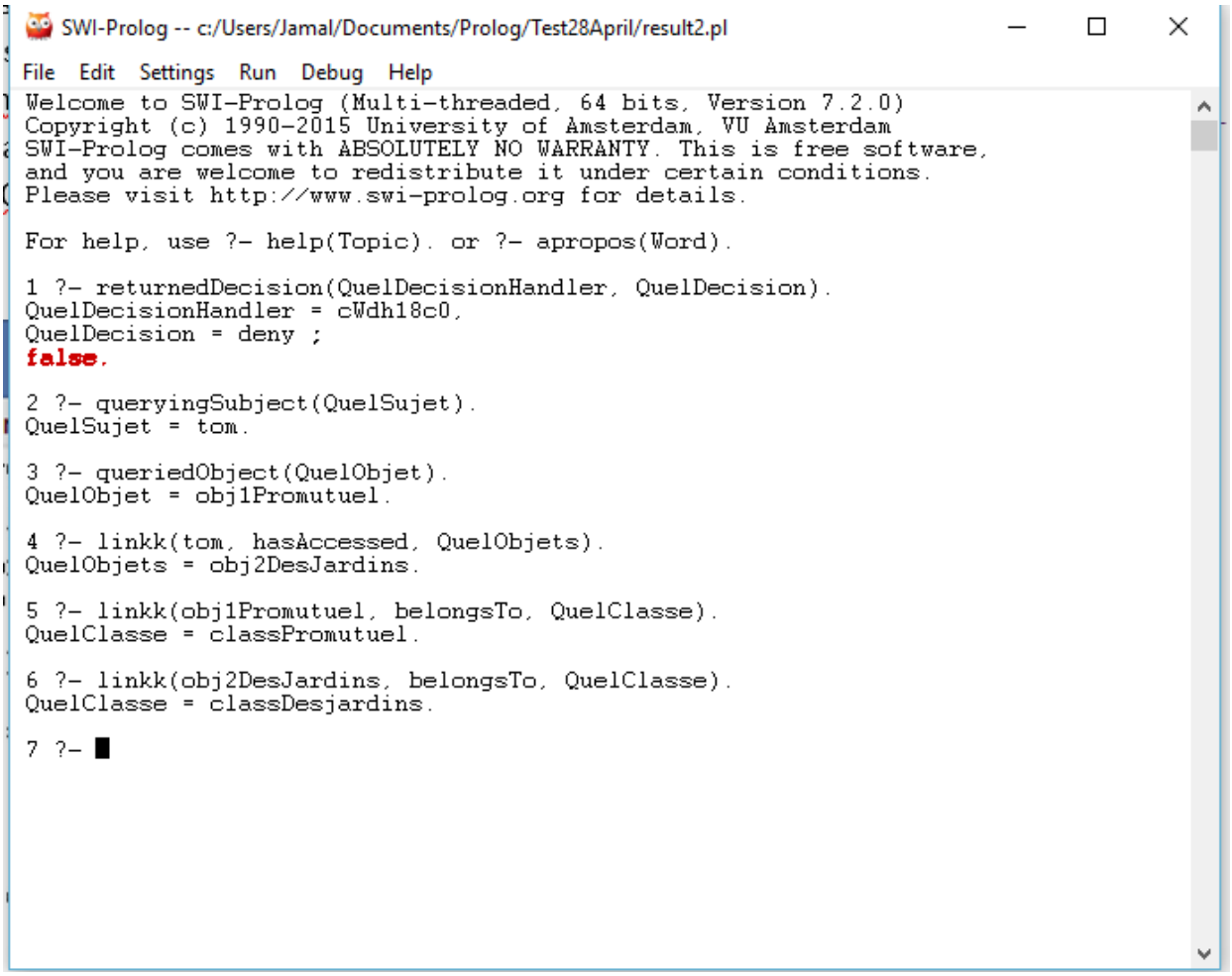

Figure 8.3-2 Illustration de vérification de propriétés.

Pour produire un exemple qui met en valeur la vérification de propriétés, on a ajouté au modèle précédent des accès à des objets. Le modèle représentant l'exemple est illustré à la Figure 8.3-3. On voudrait maintenant s'en servir pour pratiquer la vérification de deux propriétés, et ce en cherchant les réponses aux deux questions suivantes :

1) Est-ce que l'historique des accès représentés dans le modèle comprend une violation du principe de CW ?
2) Est-ce qu'un objet n'est accessible par aucun des sujets représentés dans le modèle ?

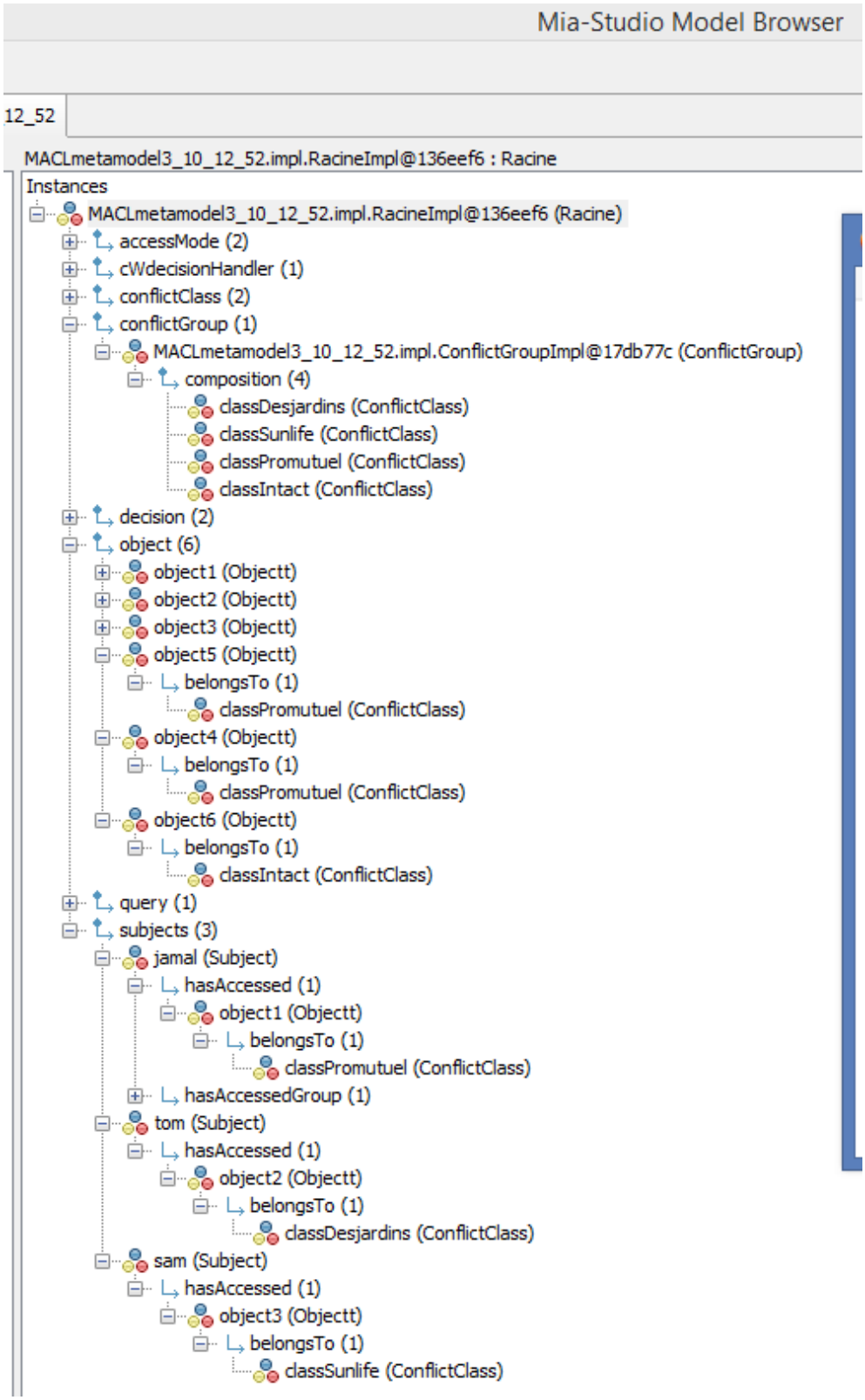

Figure 8.3-3 Modèle des spécifications de contrôle d'accès pour les exemples de vérification de propriétés

8.3.1 Vérification de violation de CW

Pour vérifier la propriété associée à la question 1), on a défini un prédicat qui n'est *vrai* que lorsqu'il existe un sujet qui a déjà accédé à deux objets appartenant à deux classes de conflit associées à l'instance de *CWdecisionHandler* identifiée par *cWdh18c0*. La clause ayant pour tête ce prédicat est montré dans l'encadré suivant. Elle est tout simplement une reproduction modifiée de la clause ayant le prédicat *returnedDecision(instanceDeCWdecisionHandler, deny)* comme tête et qui est générée automatiquement, comme on peut le remarquer en consultant l'avant dernière clause du fichier *result2.pl*. Les modifications apportées pour exprimer la violation du CW comportent un nouveau prédicat de la tête qui est *testCWviolation* et l'ajout, à la fin de son corps, d'un prédicat exprimant qu'un accès a été enregistré (rappelons que, hors outil, les noms de variables commencent par une lettre majuscule) :

```
testCWviolation(cWdh18c0, QueryingSubject, QueriedObject)
:-
linkk(cWdh18c0, uses, SomeConflictGroup),
linkk(SomeConflictGroup, composition, Class1),
linkk(SomeConflictGroup, composition, Class2),
linkk(cWdh18c0, uses, Class1),
linkk(cWdh18c0, uses, Class2 ),
linkk( QueriedObject , belongsTo, Class1),
linkk(QueryingSubject, hasAccessed, Ob),
linkk(Ob, belongsTo, Class2),
linkk(QueriedObject, belongsTo, Class1),
queryingSubject(QueryingSubject),
queriedObject(QueriedObject),
isOfType(QueryingSubject, subject),
isOfType(QueriedObject, object),
not(linkk(QueriedObject, belongsTo, Class2)),
linkk(QueryingSubject, hasAccessed, QueriedObject).
```

Le goal *testCWviolation* retourne *false* car selon la spécification illustrée dans le modèle de la Figure 8.3-3, il n'y a pas eu d'accès en violation du CW. Pour avoir un cas de test avec un résultat différent, on a ajouté au fichier de code généré automatiquement, un prédicat spécifiant que le sujet *jamal* a déjà accédé à *obj2DesJardins*. Selon le modèle de la Figure 8.3-3, *jamal* a accédé à *obj1Promutuel* et *obj2DesJardins* qui sont deux objets appartenant à deux classes distinctes d'un même groupe de conflit. C'est une violation du CW. Pour détecter cette violation, on a délégué à l'outil SWI-Prolog de revérifier le goal, *testCWviolation* et on a eu le résultat suivant qui reproduit le contenu d'une partie de la

capture d'écran de la Figure 8.3-4 (*Sujet* et *Objet* sont deux noms de variables, alors que *cWdh18c0* identifie l'unique instance de *CWdecisionHandler*) :

```
14 ?- testCWviolation(cWdh18c0, QuelSujet, QuelObjet).
QuelSujet = jamal,
QuelObjet = obj2DesJardins ;
Sujet = jamal,
Objet = obj1Promutuel ;
false.
```

On remarque que l'outil détecte deux attributions de valeur différentes aux variables *Sujet* et *Objet* satisfaisant le prédicat *testCWviolation*. Ceci est inhérent au fait que l'aspect temporel des accès n'est pas pris en charge, ainsi la violation peut résulter de l'accès par *jamal* à *obj2DesJardins*, comme elle peut résulter de l'accès par *jamal* à *obj1Promutuel*.

8.3.2 Vérification de l'accessibilité d'un objet

Pour vérifier la propriété associée à la question 2) et exprimant l'existence d'objets qui ne sont accessibles par aucun des sujets, on a défini un prédicat *testPermit* qui exprime qu'une décision permit est à émettre pour un sujet, désigné par la variable QS, demandant l'accès à un objet, désigné par la variable QO, et ce selon l'instance cWdhc18c0 de *CWdecisionHandler*. Le prédicat est défini comme tête de la clause suivante (Rappelons que « %% » indique une ligne de commentaire pour SWI-Prolog) :

```
%% La clause suivante est pour spécifier une condition de
permission %% d'accéder à une classe déjà accédée par le
sujet requérant.

testPermit(cWdh18c0, permit,QS ,QO) :- linkk(QS,
hasAccessed,O2),linkk(O2,belongsTo, ConfClass), linkk(QO,
belongsTo, ConfClass), isOfType(ConfClass, conflictClass).

%%la clause suivante est pour spécifier une condition de
permission %% d'accès pour un sujet qui n'a jamais accédé à
aucune classe de %%conflit.
```

```
testPermit(cWdh18c0, permit,QS ,QO) :- not(linkk(QS,
hasAccessed,O2)),linkk(O2,belongsTo,          ConfClass),
isOfType(ConfClass, conflictClass).
```

Selon notre modèle, on sait que *obj6Intact* appartient à la classe *classIntact* et que chacun des sujets a déjà accédé à une classe de conflit différente de *ClassIntact*. Ceci permet de déduire que *obj6Intact* n'est accessible par aucun des sujets identifiés dans le modèle. Pour vérifier ceci par SWI-Prolog, on a choisi comme goal à vérifier la satisfiabilité du prédicat *testPermit* et on a obtenu le résultat *false* comme suit :

```
12 ?- testPermit(cWdh18c0, permit, QuelSujet, obj6Intact).
false.
```

Ceci veut dire que l'objet *obj6Intact* est toujours inaccessible selon l'instance de *CWdecisionHandler* identifié par *cWdhc18c0*.

Pour les objets restants, on a pu s'assurer de leur accessibilité en vérifiant la satisfiabilité du même prédicat mais en laissant l'identifiant des objets comme une variable :

```
11 ?- testPermit(cWdh18c0, permit, QuelSujet, QuelObjet).
QuelSujet = jamal,
QuelObjet = obj1Promutuel ;
QuelSujet = jamal,
QuelObjet = obj5Promutuel ;
QuelSujet = jamal,
QuelObjet = obj4Promutuel ;
QuelSujet = tom,
QuelObjet = ob2DesJardins ;
QuelSujet = sam,
QuelObjet = obj3Sunlife ;
QuelSujet = jamal,
QuelObjet = obj2DesJardins ;
false.
```

Ainsi cette vérification de satisfiabilité a permis d'identifier tous les accès permissibles à ce point qui représentent un état de l'historique des accès. On remarque bien que *obj6Intact* est absent de cette liste car il n'est accessible par aucun des sujets identifiés dans le modèle.

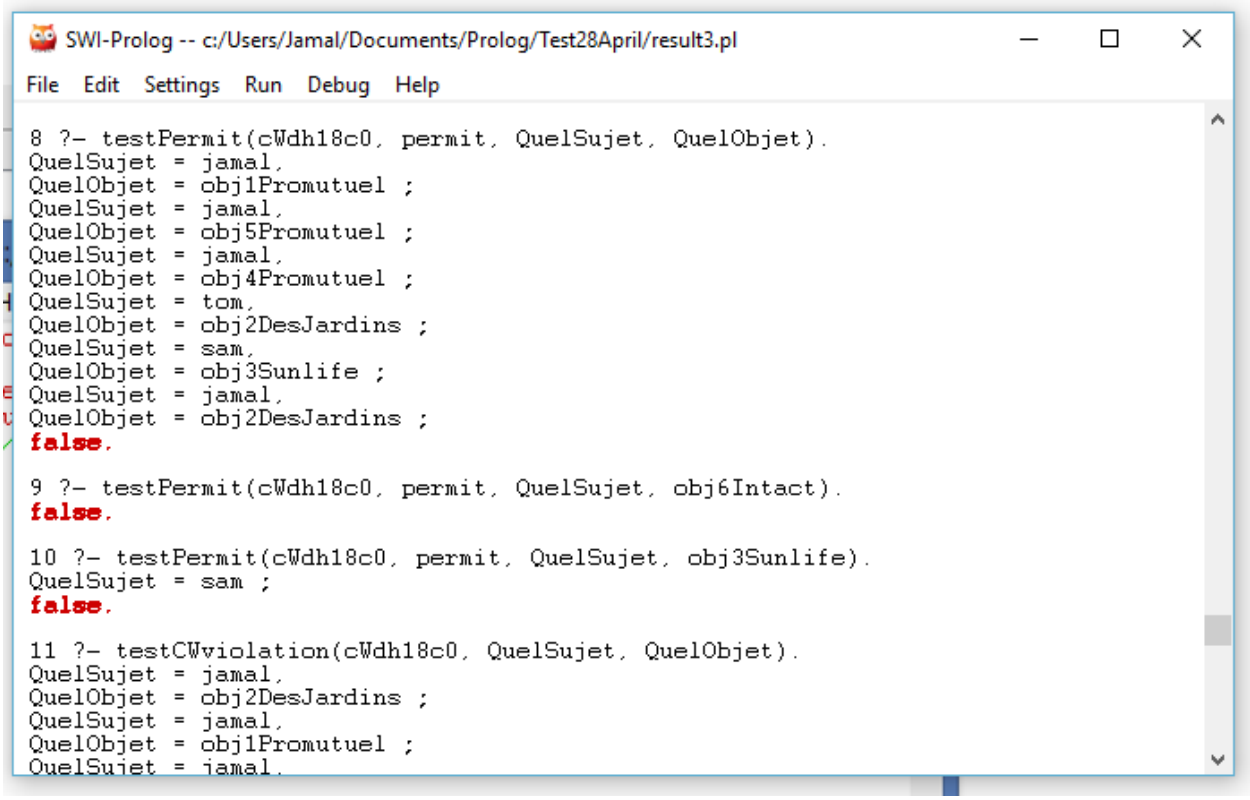

Figure 8.3-4 une capture d'écran montrant les résultats des recherches de satisfiabilité des prédicats utilisés pour les vérifications de propriétés.

8.4 Conclusion

Ce chapitre montre une implémentation qui permet l'édition visuelle et l'analyse de politiques de contrôle d'accès. Cette implémentation aide dans la spécification de politiques de contrôle d'accès et par conséquent, permet de réduire les erreurs dans l'élaboration des spécifications. En effet, créer un modèle en utilisant un outil qui assure la conformité à un métamodèle donné accroit la productivité et réduit la difficulté de maîtriser un langage textuel. À ceci s'ajoute l'automatisation de la génération de code qui

alimente un outil de programmation logique permettant de tester des cas de décisions de contrôle d'accès et ainsi vérifier certaines propriétés. Ceci confirme l'aspect réaliste d'une partie des objectifs de notre travail résumés à la Section 4.9.

Comme illustration de matérialisation d'objectifs, on mentionne que : (1) nous avons démontré comment le mapping vers le langage cible d'une spécification de contrôle d'accès permet la détermination d'une décision; (2) nous avons réalisé une activité de vérification de propriété d'accessibilité d'un objet, (3) ainsi qu'une activité de validation, moyennant la fonctionnalité de validation de modèle de notre prototype.

Annexe

**********Extrait du fichier texte généré automatiquement result2.pl

(« %% » indique une ligne de commentaire pour l'outil SWI-Prolog)

Certains commentaires sont ajoutés/modifiés manuellement pour clarifier certains points

****

```
%% les trois lignes suivantes ont des raisons syntaxiques de SWI-Prolog
:- discontiguous linkk/3.
:- discontiguous isOfType/2.
:- use_module(library(clpfd)).
%% Bloc de test
%% pour tester simplement l'exécution du mapping on a fait que l'outil
%%produit deux lignes de commentaires montrant l'extraction
%%du Id, du nom et d'un identifinat généré d'un subject, comme suit :
%%  Test : generated text  voici le id du sujet  JamalId  et voici son nom jamal  . */
%% Test : un identifiant toString est de longueur 4 :   subject1734 */
%% fin bloc de test.


%% Début des créations structurées des mappings des types
```

```
%%  créer les sujets

%% jamal est un sujet instance de subject.

isOfType( jamal , subject).

%% jamal a déjà accédé à obj1Promutuel

 linkk(jamal, hasAccessed, obj1Promutuel).

%% jamal a accédé au group de sujet class1 contenant obj1Promutuel est

%% une information redondante, mais générée uniquement pour jamal pour

%% des raisons de test.

 linkk(jamal, hasAccessedGroup,class1).

%%Test : generated text voici le id du sujet tomId  et voici son nom tom
. */

%%Test : un identifiant toString est de longueur 4 :   subjectf63d */

%% tom est un sujet instance de subject.

isOfType( tom , subject).

%% tom a déjà accédé à obj2DesJardins

linkk(tom, hasAccessed, obj2DesJardins).

%%  generated text voici le id du sujet  samId  et voici son nom sam . */

%% un identifiant toString est de longueur 4 :   subject1081 */

isOfType( sam , subject).

 linkk(sam, hasAccessed, obj3Sunlife).

%% Créer les objetts

%%  Object generated text

isOfType(obj1Promutuel , object).

 linkk(obj1Promutuel, belongsTo, classPromutuel).

%%  Object generated text

isOfType(obj2DesJardins , object).

 linkk(obj2DesJardins, belongsTo, classDesjardins).

%%  Object generated text

isOfType(obj3Sunlife , object).
```

```
 linkk(obj3Sunlife, belongsTo, classSunlife).
%%  Object generated text
isOfType(obj5Promutuel , object).
 linkk(obj5Promutuel, belongsTo, classPromutuel).
%%  Object generated text
isOfType(obj4Promutuel , object).
 linkk(obj4Promutuel, belongsTo, classPromutuel).
%%  Object generated text
isOfType(obj6Intact , object).
 linkk(obj6Intact, belongsTo, classIntact).

%%  Créer Query  avec ses attributs et associations donne */
%% Query  generated text
isOfType(query1673, query).
%% L'identifiant query1673 de la requête est généré automatiquement
%% le sujet de la requête est tom
linkk( query1673, queryingSubject,tom).
%% L'objet ciblé par la requête est obj1Promutuel
 linkk( query1673, queryingOubject, obj1Promutuel).
%% la requête identifié par query1673 est la requête courante à traiter.
 attribute(query1673, isCurrent, true).
%% Génération de prédicats identifiant le sujet et l'objet de la requête
%% Ces prédicats seront utilisés dans la clause dont la tête est
%% returnedDecision pour déterminer la décision en réponse à la requête.
 queryingSubject(tom).
 queriedObject(obj1Promutuel).

%%  Créer le mapping des ConflictGroup
```

```
%%    L'identifiant généré automatiquement    du  ConflictGroup  est
conflictGroup1400.

isOfType( conflictGroup1400 , conflictGroup).

%% Associer les classes à leur conflictGroup

 linkk(conflictGroup1400, composition, classDesjardins).

isOfType(classDesjardins , conflictClass).

 linkk(conflictGroup1400, composition, classSunlife).

isOfType(classSunlife , conflictClass).

 linkk(conflictGroup1400, composition, classPromutuel).

isOfType(classPromutuel , conflictClass).

 linkk(conflictGroup1400, composition, classIntact).

isOfType(classIntact , conflictClass).


%% Création du ConflictClass class1 créer pour des raisons de test sans

%% association à un conflictGroup

isOfType(class1, conflictClass).


%% mapping de l'appartenance d'objets à class1.

 linkk(class1, object,obj1Promutuel).


%% Création du ConflictClass class2

isOfType(class2, conflictClass).


%%  Créer le mapping du CWdecisionHandler

%%  CWdecisionHander generated text

isOfType( cWdh18c0 , cWdecisionHandler).

%% cWdh18c0 est un identifiant généré automatiquement de l'instance de
cWdecisionHandler
```

```
%% Association des groupes de conflit et classes au cWdecisionHandler
cWdh18c0

 linkk(cWdh18c0, uses , conflictGroup1400).

 linkk(cWdh18c0, uses, classDesjardins).

 linkk(cWdh18c0, uses, classSunLife).

 linkk(cWdh18c0, uses, class1).

 linkk(cWdh18c0, uses, class2).

 linkk(cWdh18c0, uses, classPromutuel).

 linkk(cWdh18c0, uses, classIntact).
```

%% La clause suivante exprime le fait que la décision est *deny* si : le sujet *QueryingSubject* demandant l'accès à un objet *QueriedObject* appartenant à une classe *Class1* a déjà accédé à un autre objet *Ob* appartenant à une classe *Class2*, avec *QueriedObject* n'appartenant pas à *Class2*.

```
returnedDecision(cWdh18c0, deny):- linkk(cWdh18c0, uses,
SomeConflictGroup),linkk(SomeConflictGroup, composition, Class1),
linkk(SomeConflictGroup, composition, Class2), linkk(cWdh18c0, uses,
Class1),linkk(cWdh18c0, uses, Class2 ), linkk( QueriedObject , belongsTo,
Class1), linkk(QueryingSubject, hasAccessed, Ob),linkk(Ob, belongsTo,
Class2), linkk(QueriedObject, belongsTo, Class1),
queryingSubject(QueryingSubject), queriedObject(QueriedObject),
isOfType(QueryingSubject, subject), isOfType(QueriedObject, object),
not(linkk(QueriedObject, belongsTo, Class2)).
```

%% La clause suivante exprime le fait que la décision est *permit* si la décision *deny* n'est pas un résultat de la clause précédente.

```
returnedDecision(cWdh18c0, permit):- not(returnedDecision( cWdh18c0,
deny)).
```

**********Fin result2.pl**********

Chapitre 9 Le langage de spécification des *combining algorithms*

Dans ce chapitre nous compléterions la spécification de MACL avec des éléments de spécification de ComAl. Nous introduisons un langage qu'on appelle ComLang qui permet de spécifier sans ambiguïté comment composer des décisions en provenance de plusieurs systèmes de décision pour aboutir à une seule décision. On abordera aussi la spécification de l'application des ComAl par des ComAlNode aux nœuds fils dans l'ADA. L'aspect formel de ComLang présenté dans ce chapitre se base, comme dans le chapitre précédent, sur un mapping de ComLang vers CLP.

9.1 MACL et ComLang

MACL doit donc fournit des moyens de spécifier des règles de composition de décisions provenant de plusieurs systèmes de décision.

Le processus qui permet de transformer les décisions provenant de plusieurs systèmes de décision en une seule décision, est dit une composition de ces décisions. La composition de systèmes de décision résulte en un nouveau système de décision. La spécification de cette composition est dite un Combining Algorithm (ComAl, voir Section 1.1.3).

On a vu dans la Section 6.3 qu'un énoncé en MACL doit spécifier un ADA. Ce dernier est formé d'instances de DecisionHandler et de ComAlNode. ComLang est le langage qui permet de spécifier les ComAl que les ComAlNode appliquent pour composer les décisions de leurs nœuds fils dans l'ADA. Selon notre approche, MACL et ComLang se complètent pour permettre la spécification des instances des ACMM. Tout énoncé en ComLang est un énoncé en MACL, mais l'inverse n'est pas *vrai*.

ComLang doit être simple, concis et expressif. Vu la complexité provenant de l'explosion combinatoire du nombre de compositions possibles de décisions, un besoin de validation des spécifications de composition de décisions s'impose. La section suivante détaillera les éléments d'un ComAl et d'un ComAlNode.

9.2 Les combining algorithms les plus connus

Les combining algorithms les plus connus dans le domaine du contrôle d'accès sont ceux proposés par la norme XACML [4]. Ci-dessous une explication brève et simplifiée de ces combining algorithms. Pour simplifier, nous supposons que l'évaluation de toute règle (ou politique) se limite à une des valeurs: {Permit, Deny, NotApplicable, Indeterminate}.

Deny-overrides : le résultat de la composition d'un ensemble de règles dans lequel au moins une d'entre elles s'évalue à *Deny* sera un *Deny*. Le résultat de la composition d'un ensemble de règles dans lequel aucune d'entre elles ne s'évalue à *Deny*, sera NotApplicable.

Permit-overrides : le résultat de la composition d'un ensemble de règles dans lequel au moins une d'entre elles s'évalue à *Permit* sera *Permit*. Le résultat de la composition d'un ensemble de règles dans lequel aucune d'entre elles ne s'évalue *Permit*, sera *NotApplicable*.

First-applicable : l'ordre dans lequel les règles sont écrites est significatif ; les règles sont évaluées à partir de la première et une fois qu'une règle applicable est rencontrée, elle est appliquée et les règles restantes ne sont pas évaluées.

Only-one-applicable: si une seule règle s'applique à une demande d'accès, cette règle détermine la décision. Lorsque plus d'une règle est applicable à une demande d'accès, une décision *NotApplicable* est retournée. Lorsqu'aucune règle n'est applicable à une demande, une décision *NotApplicable* est retournée.

D'autres stratégies de composition sont utilisées dans autres domaines d'application, dont certaines sont revues dans [7] et parmi lesquelles on peut mentionner :

Faible Consensus : il retourne *Permit* lorsque certains systèmes de décision adoptent cette décision et aucun autre système de décision ne renvoie *Deny*. Si *Deny* et *Permit* sont simultanément dans la liste des décisions à composer, le combining algorithm retourne une décision *Indeterminate*.

Fort consensus : il reflète une décision unanime. Si des décisions différentes figurent dans la liste de décisions à composer, le résultat est la décision *Indeterminate*.

Majorité de vote : il adopte la décision qui a le plus grand nombre d'occurrences dans la liste des décisions à composer. Une décision *Indeterminate* est retournée lorsque le plus grand nombre d'occurrences est enregistré pour deux ou plusieurs décisions différentes.

Majorité absolue : il adopte la décision *Permit* si plus de la moitié des décisions dans la liste d'entrée sont des décisions *Permit*. Dans le cas contraire, la décision est un Deny.

9.3 Aspects et exigences de ComLang

L'utilisation simultanée de plusieurs ComAl, qui peut être nécessaire dans un ADA, rend leur spécification complexe et sujette à des erreurs. Notez qu'on ne se limite pas à un seul modèle de ComAl : entre autres, on peut mélanger un modèle de vote à un modèle de priorité de décision, en plus de suivre certains modèles conçus spécifiquement pour des situations particulières. D'où la nécessité d'un langage qui permet de spécifier formellement les ComAl afin d'automatiser l'analyse de propriétés et la détection d'erreurs.

Ainsi, notre langage proposé ComLang est élaboré avec plusieurs exigences :

(1) Expressivité, avec :
    a. Capacité d'exprimer les ComAl les plus connus, ce qui signifie que ComLang doit permettre de spécifier au moins les ComAl bien connus de la Section 9.2.
    b. Aucune limitation à un domaine d'application particulier avec ses méthodes de décision spécifiques.
    c. Conscience de la source: c'est la capacité de prendre en compte l'origine de chaque décision dans la liste de décisions à composer. Par exemple, une décision de *Deny* visant à se conformer à une exigence légale de contrôle d'accès peut être traitée différemment d'un *Deny* résultant d'une politique de recommandations générales comme celle du Need-to-know.
    d. Considération des valeurs d'entrée contextuelles qui désignent l'aptitude à prendre comme entrée des valeurs qui ne sont pas des décisions, mais qui peuvent influencer les décisions. Par exemple, on considère un ComAl qui, en spécifiant la décision de sortie, tient compte de l'heure, de l'objet ciblé ou du sujet requérant l'accès.
(2) Dotation de moyens de composition qui permettent de spécifier l'utilisation successive de plusieurs ComAl appartenant aux différents ComAlNode dans un ADA.
(3) Des moyens de validation et vérification : le langage cible permet l'automatisation de la validation de la cohérence et la vérification des propriétés des systèmes de décision.

Ces exigences vont guider l'élaboration de ComLang, et nous y revenons dans la Section 9.7.1 pour voir à quel point ComLang satisfait ces exigences.

9.4 Les éléments d'un *combining algorithm*

Formellement, un ComAl est une fonction qui prend en entrée un n-uplet comprenant des décisions appelées les décisions d'entrée, en plus d'éventuelles informations de contexte, et qui renvoie en sortie une décision unique que nous appelons la décision de sortie (Od pour Output Decision). L'ensemble de toutes les décisions d'entrée possibles est dénoté par $\sum$. L'ensemble de toutes les décisions finales possibles est dénoté par Π. Comme introduction à l'analyse qui suit, nous examinerons un exemple qui considère trois stratégies de contrôle d'accès avec les systèmes de décision associés formés des

politiques P1, P2 et P3. En réponse à une demande d'accès, chaque politique renvoie une décision membre de l'ensemble $\sum$ = {Permit, Deny, NotApplicable}. Les trois décisions qui peuvent être émises par P1, P2 et P3 dans un cas précis sont dénotées d1, d2 et d3, où chaque di a une valeur dans $\sum$. Dans notre exemple, la décision de sortie qui résultera de la composition des décisions de P1, P2 et P3 est un élément de l'ensemble de décisions de sortie $\prod$ = {Permit, Indeterminate}.

La décision résultant de la composition des trois décisions de P1, P2 et P3 peut aussi dépendre des valeurs d'attributs que nous appelons des attributs de contexte. Ces attributs de contexte peuvent être des identificateurs du sujet et/ou de l'objet dans le domaine de contrôle d'accès, des attributs du sujet et/ou de l'objet dont la valeur doit être prise en considération par l'algorithme de combinaison, ou des attributs environnementaux tels que le temps ou le lieu. Dans notre exemple, les attributs de contexte sont l'identificateur de la source de demande, l'objet ciblé et le jour de la semaine. Ces attributs sont désignés par Source, TargetedObject et Day, et leurs valeurs possibles sont respectivement des éléments des ensembles désignés par C1, C2 et C3. Ces ensembles peuvent être définis comme suit :

Source $\in$ C1 = {Tom, Bob, Sam};

TargetedObject $\in$ C2 = {Printer, Scanner, File1, File2};

Day$\in$ C3 = {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}.

Un exemple d'une entrée de n-uplet de décisions et d'attributs de contexte peut être: ((Permit, Deny, Permit), (Sam, Scanner, Sunday)). Ce n-uplet d'entrée du ComAl représente trois décisions rendues par les trois systèmes de décision P1, P2 et P3, suivi par les valeurs de trois attributs de contexte indiquant que Sam souhaite accéder au Scanner dimanche.

Le ComAl va émettre une décision élément de l'ensemble Π = {Permit, Indeterminate}.

Le langage de spécification des combining algorithms

On peut formuler la définition du ComAl comme suit :

$\sum \times \sum \times \sum \times$ C1 × C2 × C3 → Π

Le produit cartésien de m ensembles C1 $\times$ C2 $\times$... $\times$ Cm où chaque Ci est un ensemble de valeurs possibles d'un des attributs de contexte considéré s'appellera CP. Les éléments de $\sum$ peuvent être ordonnés dans une liste et sont dénotés par D_1, D_2, ..., D_{idn} où idn (pour input decision number) est la cardinalité de $\sum$. Les éléments de l'ensemble des décisions finales possibles $\prod$ peuvent être ordonnés dans une liste et sont dénotés par Od_1 et Od_2, ..., Od_{odn} où odn (pour output decision number) est la cardinalité de $\prod$. À noter que $\prod$ et $\sum$ peuvent être deux ensembles disjoints.

Par la suite, X désigne une variable dont la valeur appartient à l'ensemble des n-uplets d'entrée possibles alimentant un ComAl. Cet ensemble sera désigné par A. Par conséquent on peut écrire:

X = < Feed, Cont> $\in$ A, avec

Feed $\in \sum^n$ AND Cont $\in$ CP

Autrement dit, un ComAl est une fonction qui peut être notée comme suit :

ComAl: $\sum^n \times$ CP $\rightarrow \prod$

ou plus brièvement,

ComAl: A $\rightarrow \prod$

Nous remarquons ici que certains ComAl peuvent être définis avec un nombre indéterminé de décisions d'entrée à composer. Par conséquent, pour considérer de tels ComAl nous devons remplacer $\sum^n$ dans la définition ci-dessus par l'ensemble des séquences d'éléments de $\sum$. Dans la discipline des langages formels, cet ensemble est dit la fermeture positive de Kleene du langage $\sum$ et est dénotée par $\sum$ +.

Les éléments de A qui ont la même image Od_i forment un sous-ensemble de A que nous désignons par ClassOd(i) ou ClassOd(Od_i). Une telle classe représente les n-uplets d'entrée pour lesquelles le ComAl retourne la même décision de sortie Od_i. Par conséquent, l'ensemble de toutes les ClassOd(i) pour toutes les valeurs de i allant de 1 jusqu'à odn constitue une partition de A. Les trois propriétés suivantes découlent de cette observation, et nous les énumérons ici parce que chacune d'elle sera sujette à une validation lors de la spécification d'un ComAl :

1. La cohérence d'un ComAl: Toutes les classes distinctes sont disjointes deux à deux. Autrement on aura une entrée X impliquant deux décisions de sortie différentes par un même ComAl.
2. La complétude d'un : ClassOd(1) U ClassOd(2) U…..U ClassOd(odn) = A . Ceci revient au fait que ComAl est une fonction définie sur tout l'ensemble A.
3. La non-redondance d'un ComAl: il n'y a pas de ClassOd(i) vide.

La définition d'un ComAl revient à définir une partition de l'ensemble A en un ensemble de classes ClassOd(i). Notre objectif est d'élaborer un langage dont les éléments peuvent être mappés à des prédicats qui spécifient chaque ClassOd(i). Un prédicat spécifiant une ClassOd(i) est spécifié en termes de composition de prédicats exprimant les contraintes sur les variables qui représentent des éléments de $\sum$, $\prod$ et des attributs de contexte éléments de CP.

Comme exemple d'illustration que nous désignons dans la suite par Exemple (1), on considère un ComAl selon lequel toute occurrence d'une décision *Deny* dans le n-uplet d'entrée entraine une décision de sortie *Deny*. Avec, D_1 = Deny comme élément de $\sum$ et Od_3 = *Deny* comme élément de $\prod$, nous pouvons définir cette propriété par :

ClassOd(3) = ClassOd(Deny) = { X = < Feed, Cont> | $D_1 \in$ Feed }

ClassOd(Deny) représente l'ensemble des éléments de A qui mènent à une décision de sortie *Deny* selon le ComAl considéré.

Conformément aux exigences de ComLang mentionnées dans la Section 9.3, nous devons être capables d'exprimer les correspondances entre les n-uplets de A et les décisions de

sortie qui en résultent. Plus précisément, la définition d'un ComAl se réduit à la spécification des classes ClassOd(i) pour i allant de 1 à odn. Ainsi, on peut remarquer que les ClassOd(i) sont des sous-ensembles de A; et que les sous-ensembles sont spécifiables avec des prédicats sur les éléments de A. La logique du premier ordre [58] est bien connue par son adéquation à la spécification de prédicats dans le génie logiciel [35].

Selon notre approche la spécification du sous-ensemble ClassOd(i) est assurée par la spécification des prédicats sur les éléments de A pour que la décision de sortie soit Od_i et ceci s'écrit selon la clause suivante :

ClassOd(i) ← UneConjonctiondePrédicats.

Cette clause est, comme déjà mentionné dans la Section 7.3, une expression logique dont le membre gauche est un prédicat (appelé la tête), tandis que son membre droit (appelé corps) est une conjonction logique de prédicats qui implique la vérité de la tête. On va donc suivre les mêmes notations des prédicats présentés dans la Section 7.2. On rappelle aussi l'exemple de la clause suivante :

```
Human(X)  ← Walk(X), Talk(X).
```

Cette spécification est basée sur la logique du premier ordre ce qui permet d'en vérifier des propriétés et en particulier sa cohérence. Cela est essentiel pour satisfaire l'exigence numéro 3 de vérifiabilité. La Section 10.1.2 portera sur l'élaboration de moyens de vérification de propriétés.

9.5 Le langage ComLang

À l'instar de notre explication du langage MACL, on va aborder ComLang avec une approche de métamodélisation. Ainsi, la première sous-section sera consacrée à l'explication du métamodèle de ComLang. Ayant les éléments de modélisation bien définis, le mapping vers les éléments textuels de ComLang en dérive en adoptant les règles de mapping déjà établies pour MACL dans Table 7.3-1. La deuxième section portera sur le mapping vers CLP de ComLang.

9.5.1 Métamodèle de ComLang et sa forme textuelle

Étant donné les éléments d'un ComAl comme décrits dans la section précédente, nous allons introduire le métamodèle spécifiant ComLang puis nous expliquerons ses éléments syntaxiques. Le métamodèle de ComLang fait partie du métamodèle d'intégration de MACL illustré partiellement dans la Figure 6.3-2. Dans la Figure 9.5-1 , on montre les éléments qui enrichissent le IM pour représenter des concepts propres à la spécification des ComAlNode.

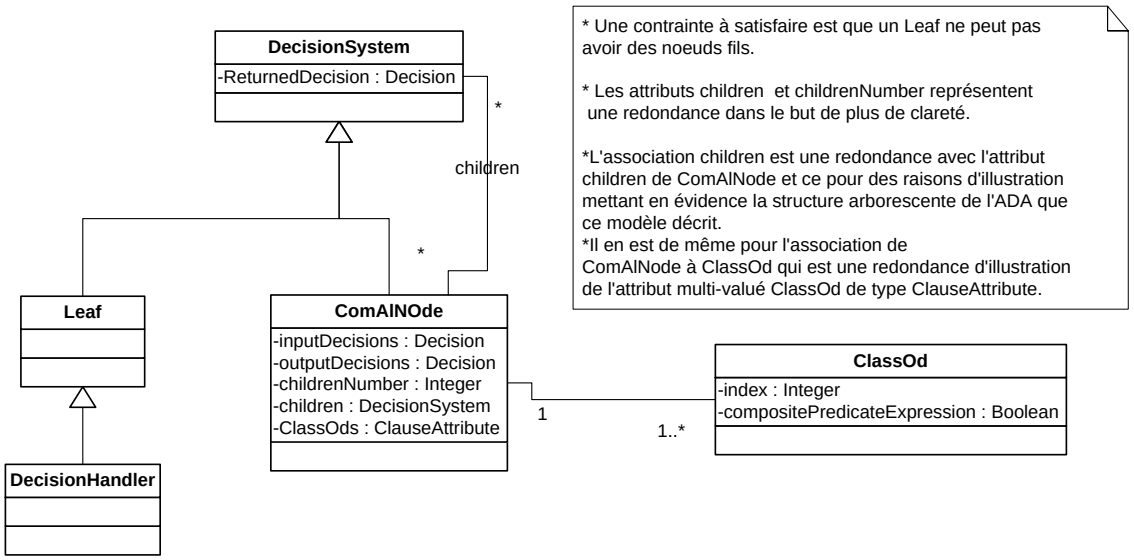

Figure 9.5-1 Vue de l'IM montrant les éléments de composition de systèmes de décision

Ce modèle indique que la composition de systèmes de décision doit suivre la structure de l'ADA où les feuilles sont des instances de DecisionHandler et les autres nœuds sont des ComAlNode.

Pour spécifier un ComAlNode, il faut spécifier ses attributs: (1) *inputDecisions* fournissant la liste des décisions d'entrée, (2) *outputDecisions* fournissant la liste des décisions de sortie, (3) *childrenNumber* et (4) *children* qui représentent, respectivement, le nombre de systèmes de décision fils et leurs identifiants. La relation parent-fils est exprimée en ComLang par le biais de l'attribut du ComAlNode nommé *children* qui consiste en une liste de références pour les instances de feuilles ou de ComAlNode représentant des systèmes de décision. À remarquer ici, qu'à l'exception de *childrenNumber*, tous les attributs de

ComAlNode sont multi-valués, (5) ClassOds qui est l'attribut qui consigne la logique de l'élaboration de la décision de sortie.

La spécification des ClassOd(i) est fournie comme valeur d'un Clause Attribute (voir Section 7.2) ClassOds qui est une conjonction de clauses dont au moins une ayant comme tête ClassOd(i) où i est un entier compris entre 1 et odn. Ces clauses sont séparées par des « , » car chacune d'elles se termine par un point. Chaque clause dont la tête est ClassOd(i) comprend un corps qui est une expression logique formée d'une composition de prédicats sur les éléments d'entrée (désigné par X = <Feed, Cont>) et les éléments instances des éléments du métamodèle de MACL.

De plus pour faire le lien entre les Clause Attribute ClassOds et ReturnedDecision d'un ComAlNode, on adopte la règle logique suivante :

« Si selon l'attribut ClassOds pour une décision d, ClassOd(d) s'évalue à *true* alors le Clause Attribute ReturnedDecision doit avoir ReturnedDecision (d) qui s'évalue à *true.* »

Ceci sera compris dans le complément du mapping vers CLP de l'IM et de ses instances et qui prend en charge des aspects particuliers de ComLang. Ce complément fera l'objet de la section suivante.

ComLang puise son pouvoir expressif dans ses moyens de spécification des instances de ComAlNode et leurs attributs. Pour dériver les éléments syntaxiques du modèle abstrait comprenant ComAlNode, nous allons adopter le même mapping déjà défini dans la Section 7.3 pour passer des éléments d'un sous-ensemble d'UML aux éléments textuels de MACL.

Comme conventions typographiques, nous utiliserons des caractères gras pour les mots clés et les têtes des clauses; les noms d'instance seront préfixés par leur nom de type précédé de « a »; "/" marquera le début d'une ligne de commentaire, tandis que"/ * " et "* / " délimitera le texte du commentaire.

En se basant sur les mappings (entre modèles et textes) déjà adoptés pour MACL, une instance de ComAlNode est créée selon la syntaxe suivante :

```
CreateInstance(ComAlNode, aComAlName)
{ inputDecisions = [Comma separated decision names] ;
  outputDecisions =[Comma separated decision names] ;
  childrenNumber = anInteger;
  children = [Comma separated decision systems instance names ] ;

      ClassOds = Comma separated ClassOd(Odi)← Predicate
expression.;
}
```

Les accolades "**{ }**" délimitent les spécifications des attributs d'une instance. Les crochets "[ ]" délimitent les éléments d'une liste séparés par des virgules. L'expression d'un ClassOd(i) est une fonction logique des variables de décision d'entrée conduisant au choix de la décision finale Od_i. Nous avons réservé, en ComLang et le langage cible, certains noms de variables pour désigner des éléments particuliers tels que le n-uplet de la paire d'entrée X = <Feed, Cont>, les décisions d'entrée D_1, ..., D_{idn}, les décisions de sortie Od1, ..., Od_{odn} et les contexte attributes $Cont_i$. Ces noms de variables réservés indiquent les éléments essentiels qui sont nécessaires à l'expression de ClassOd(i) en tant que prédicats. ComLang spécifie un ClassOd(i) en fournissant une expression booléenne de la prémisse pour le prédicat ClassOd(i), cette expression est désignée par aCompositionOfpredicates dans la syntaxe suivante :

ClassOd(i) ← aCompositionOfpredicates.

Dans la suite de notre travail, les n-uplets seront représentés par des listes, car les listes sont dans un type pris en charge avec des opérations prédéfinies par un bon nombre d'implémentation du CLP. De plus, dans le cadre de notre travail associant ComLang à MACL les attributs de contexte seront spécifiés par des instances du métamodèle IM. En outre, on n'a besoin du n-uplet Cont que lorsqu'on veut utiliser ComLang comme un langage de composition de décision indépendant et qui doit prendre en considération des éléments autres que les décisions d'entrée.

De plus, pour améliorer le pouvoir expressif de spécification des ClassOds, on a prédéfini des prédicats qui seront réutilisables et pris en charge dans ComLang. Le tableau Table 9.5-1 ci-dessous présente ces prédicats où chacun d'eux est une fonction booléenne.

Table 9.5-1 Prédicats prédéfinis

Prédicats prédéfinis	**Arguments**	**Sémantique [17]**
Card(T, N)	Une liste T et un entier naturel N	L'entier N donne la taille de la liste T
IsMemberOf(Y, T)	Un élément Y et une liste T	L'élément Y est membre de T
IsMemberOf(Y, T, i)	Un élément Y une liste T et un entier naturel i	L'élément Y est membre de T et son rang est i : Y= T(i)
Concat (T1, T2, T3)	Trois listes T1, T2 et T3	La liste T3 est une concaténation des deux listes T1 et T2
CountIndexAndValueMatches (T1, T2, N)	Deux listes T1 et T2 et un entier naturel N	L'entier N est égal au nombre de paires d'éléments égaux et ayant les mêmes indices dans les deux listes T1 et T2

[17] Dans la colonne de Sémantique on fournit l'interprétation du prédicat en spécifiant le contexte dans lequel il s'évalue à *vrai*.

CountMatches(T1, T2, N)	Deux listes T1 et T2 et un entier naturel N	Nous comptons le nombre d'occurrences de chaque élément de T1 dans T2 ; Ensuite, nous faisons la somme de ces nombres pour trouver N.
InListRange (Y, J, K, T)	Un élément Y, deux entiers naturels J et K, et une liste T	Au moins un élément de T correspond à Y et l'indice de cet élément est entre J et K inclusivement.

À ce stade-ci, on peut fournir l'expression de l'attribut ClassOds pour notre exemple illustratif de Deny-overrides et ce comme suit :

```
ClassOds = ClassOd(Deny) ←IsMemberOf(Deny, Feed). ,
            ClassOd(NotApplicable) ← CountMatches(Feed, [Deny], 0).;

// Ou encore

ClassOds = ClassOd(Deny) ← CountMatches(Feed, [Deny], Count)
           AND Count>0. ,
            ClassOd(NotApplicable) ← CountMatches(Feed, [Deny], 0).;
```

La spécification complète d'un ComAlNode appliquant Deny-overrides avec des nœuds fils à ajouter ultérieurement dépendamment de son utilisation dans un ADA, sera comme suit :

```
CreateInstance(ComAlNode, D-OComAl)
{inputDecisions = [Permit,NotApplicable,Deny,Indeterminate];
outputDecisions = [NotApplicable,Deny] ;

// childrenNumber attribut à prendre la valeur du nombre de nœuds fils.
// children à spécifier comme liste d'instances de DecisionSystem

ClassOds = ClassOd(NotApplicable)← CountMatches(Feed, [Deny], 0).,
           ClassOd(Deny)← IsMemberOf(Deny, Feed).  ;
}
```

À noter ici qu'on aurait pu remplacer ClassOd(Deny) par ClassOd(2) car *Deny* est la décision de sortie d'indice 2 dans la liste outputDecisions, cela est normal dans la référence à tout élément d'une énumération où un entier peut se substituer à un élément de l'énumération. Cela serait utile dans certains cas où l'indice i est utilisable plus efficacement dans la spécification des ClassOds, par exemple dans le cas où l'on applique la priorité des décisions qui sont ordonnées dans leurs listes selon leurs priorités. Ceci prend des clauses logiques qui initialisent tout mapping vers CLP comme suit :

```
ClassOd(i) ← ClassOd(Odi).
ClassOd(Odi) ← ClassOd(i).
```

Outre les éléments ClassOd(i), ComLang permet d'inclure dans la conjonction des clauses de l'attribut ClassOds des clauses introduisant éventuellement de nouveaux prédicats et ce pour les inclure directement dans le mapping vers le langage cible. De telles clauses seront donc ajoutées au mapping vers CLP de l'attribut ClassOds considéré. Par exemple, on peut spécifier et réutiliser un prédicat Max3(J, K, L, M) qui s'évalue à *true* si M est le plus grand des nombres réels J, K et L.

9.5.2 Le mapping vers CLP

Comme dans le cas des éléments déjà vus de MACL, les énoncés de ComLang peuvent être sujets d'un mapping vers CLP permettant le raisonnement formel. Pour élaborer le mapping, on va se baser sur le métamodèle de ComLang qui représente un enrichissement de l'IM avec des spécifications plus détaillées de l'élément ComAlNode. Le langage cible et les conventions typographiques seront les mêmes que celles adoptées dans la Section 7.2

Tout élément textuel de ComLang correspond à un élément de modélisation dont le métamodèle est présenté dans la Figure 9.5-1. Ainsi, il suffit de définir les mapping vers CLP des éléments de modélisation qu'on appelle mapping model-CLP. Le mapping des éléments textuels de ComLang vers CLP (dit mapping Text-CLP) peut être défini en deux

étapes : (1) on applique le mapping des éléments textuels vers éléments de modélisation (dit mapping Text-model); (2) les éléments résultant vont subir le mapping model-CLP. La Figure 9.5-2 illustre les différents chemins de mapping.

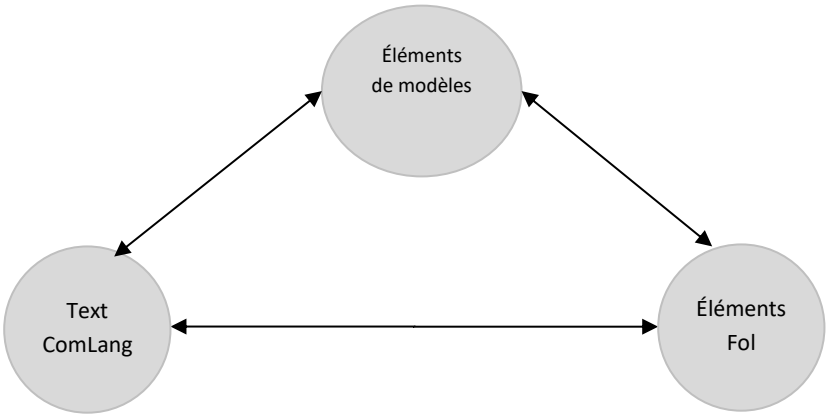

Figure 9.5-2 Les mappings entre les différentes formes de spécification de ComAlNode et ses instances

Les mappings model-CLP et Text-model sont déjà définis dans les Sections 7.2 et 7.3 pour MACL et seront applicables pour ComLang qui fait partie de MACL. On fournit néanmoins dans le tableau suivant ces mappings appliqués particulièrement à une instance de ComAlNode pour bien illustrer les particularités de la traduction logique de la structure arborescente des DecisionSystem dans un ADA, en plus des mapping vers CLP des attributs ClassOds et ReturnedDecision dans une instance de ComAlNode.

Table 9.5-2 Mapping Model-CLP de ComLang

Éléments de modélisation ComLang	**Éléments mappés en CLP**
Création d'une instance de ComAlNode dont le nom est aComAlNode	**IsOfType**(aComAlNode, ComAlNode).

inputDecisions = [$inDecisionName_1$, $inDecisionName_2$, … , $InDecisionName_{idn}$];	**Attribute-list**(aComAlNode, inputDecisions, inDecisionNamei, i). // pour i = 1 à idn
outputDecisions = [$outDecisionName_1$, $outDecisionName_2$, … , $OutDecisionName_{odn}$];	**Attribute-list**(aComAlNode, outputDecisions, $outDecisionName_i$, i). // Pour i = 1 à odn
Children = [childName1, childName2,..childNameN];	**Attribute-list**(aComAlNode,children, childName1, 1). **Attribute-list**(aComAlNode,children, childName2, 2). … **Attribute-list**(aComAlNode,children, childNameN, N).
childrenNumber= N;	**Attribute**(aComAlNode, childrenNumber, N).

ClassOds = ClassOd(U**)**← Predicate expression. , ClassOd(V))← Predicate expression. , … , LastClauseHead ← LastClauseBody.; /* U et V représentant des decisions de sortie. */	/* Les clauses de ClassOds passent aux éléments CLP avec trois modifications mineures qui permettent de les associer à l'instance courante de ComAlNode. **Premièrement** : Les Clauses dont les têtes sont de la forme **ClassOd(aDecision)** sont déjà en CLP, mais le mapping doit les modifier en ajoutant le nom de l'instance courante du ComAlNode. Ce dernier est désigné dans ce tableau par aComAlNode. Ainsi on reproduit les têtes de ces clauses sous la forme **ClassOd(aComAlNode, aDecision)**, comme suit: */ ClassOd(aComAlNode, U)← expression de prédicats. ClassOd(aComAlNode, V)← expression de prédicats. /* **Deuxièmement** : L'élément Feed est utilisé dans la spécification des clauses ClassOd pour désigner la liste des décisions d'entrée, cette liste provient des décisions des nœuds fils de aComAlNode (l'instance courante étant ici nommée aComAlNode). Pour traduire ceci dans le mapping, on remplace « **Feed »** par une concaténation du nom de l'instance courante et de « Feed », ce qui donne un nom comme **aComAlNodeFeed**. */ /* **Troisièmement** : Pour donner à Feed sa sémantique de liste de décisions d'entrée de l'instance courante de ComAlNode, il faut que cette sémantique se reflète dans le mapping. Ainsi le

	mapping des clauses de l'attribut ClassOds de toute instance de ComAlNode, nommée ici aComAlNode, sera augmenté des clauses suivantes : */ `/* pour i allant de 1 à childrenNumber */` **`IsMemberOf`** `(Di,aComAlNodeFeed, i) ← Attribute-list (aComAlNode,children, childi,i) , ReturnedDecision(childi,Di).` `/*` Cette clause indique que les éléments de la liste aComAlNodeFeed est la même que la liste des décisions émises par les nœuds fils de l'instance courante aComAlNode.*/ /* Cette nomenclature du mot réservé Feed, nous évite la collision de nom des différents Feed de plusieurs ComAlNode mappés vers CLP. */

Règles supplémentaires d'initialisation complétant tout mapping	
Justification de l'initialisation	**Éléments CLP**
ReturnedDecision : ce Clause Attribute est hérité de DecisionSystem et il sera implémenté dans ComAlNode pour indiquer que les prédicats ClassOd sont responsables de déterminer la décision retournée par toute instance de ComAlNode. ReturnedDecision est par suite un Clause Attribute dont la valeur est: **ReturnedDecision (i) ← ClassOd(i).** **Pour i allant de 1 jusqu'à odn**	/* Pas besoin de mapping pour chaque instance de ComAlNode, car on se contente d'établir une fois pour toutes la règle suivante : */ **ReturnedDecision**(anyComAlNode, Z) ← ClassOd(anyComAlNode, Z), IsOfType(anyComAlNode, ComAlNode).

Les décisions de sortie prises en charge sont spécifiées en listes, d'où on ne fait pas de distinction entre le rang de la décision dans sa liste et la décision elle-même (pratique généralement appliquée pour les éléments des énumérations).	**/* Règles à générer dans CLP une fois pour toutes les instances de ComAlNode: */** **ClassOd**(anyComAlNode, i) ← ClassOd(anyComAlNode, Odi), Attribute-list(anyComAlNode, outputDecision, odi, i), IsOfType(anyComAlNode, ComAlNode). /* l'inverse est *vrai* aussi : */ **ClassOd**(anyComAlNode, odi) ← ClassOd(anyComAlNode, i), Attribute-list(anyComAlNode, outputDecision, odi, i), IsOfType(anyComAlNode, ComAlNode).

Pour appliquer ou tester une spécification d'une instance de ComAlNode qu'on désigne par aComAlNode, on peut fournir des expressions CLP qui spécifient les valeurs des éléments de Feed comme une liste; puis on les ajoute au mapping CLP de l'instance ComAlNode. Cela peut être fait en donnant pour chaque décision d'entrée particulière au rang i, nommée par exemple ForcedFeed-i, une valeur *vrai* au prédicat :

```
IsMemberOf(ForcedFeed-i, aComAlNodeFeed, i).
```

`ForcedFeed-i` est bien une décision choisie comme Permit ou Deny.

Un ensemble de clauses, qui est un programme logique, résulte de l'application des règles du tableau du mapping y compris les initialisations et les prédicats vrais spécifiant les décisions ForcedFeed-i. Le programme logique résultant permettra de déduire la décision de sortie à l'aide des règles d'inférence CLP. On procède par recherche de satisfiabilité pour trouver la valeur de décision od_i qui satisfait le prédicat ClassOd (aComAlNode, od_i) pour notre choix des décisions d'entrée.

D'autre part, les prédicats prédéfinis et leurs sémantiques doivent faire partie d'une base de connaissance qui complète tout programme logique. On veut dire par sémantique d'un prédicat prédéfini toute clause nécessaire pour spécifier les conditions de vérité du prédicat prédéfini y compris la théorie des types dont il dépend. Ces spécifications dépendent de l'implémentation, et elles dépendent essentiellement des théories prises en charge par l'outil CLP et SMT (entiers, listes, leurs compositions et relations) et des restrictions d'utilisation des connecteurs logiques de l'implémentation. Par exemple, dans certains implémentations du CLP, le prédicat qui vérifie le nombre d'éléments d'une liste est prédéfini, et on aura une implémentation directe du prédicat Card(L, n) qui est *true* si la longueur de la liste L est n.

Comme exemple de mapping logique on reprend l'exemple d'une spécification d'une instance de ComAlNode appliquant le ComAl Deny-overrides de façon à avoir une décision de sortie limitée à Deny ou NotApplicable. Donc si *Deny* est dans Feed alors le résultat est *Deny*, sinon c'est NotApplicable. Dans cet exemple, et dans les exemples des sections suivantes, on omet généralement de spécifier les nœuds fils pour ne pas alourdir le mapping avec des éléments qui n'influencent pas la logique de composition de décisions. Le mapping est présenté dans le tableau ci-dessous :

Spécification dans ComLang	Mapping logique
CreateInstance(ComAlNode, D-OComAl**) {**	IsOfType(D-OComAl, ComAlNode). /* mapping à confirmer en attente de l'accolade de fermeture*/
inputDecisions = [Permit, NotApplicable, Deny, Indeterminate];	**Attribute-list**(D-OComAl, inputDecisions, Permit, 1) **AND** **Attribute-list**(D-OComAl, inputDecisions, Deny, 2) **AND**

	Attribute-list(D-OComAl, inputDecisions, NotApplicable, 3) **AND** **Attribute-list**(D-OComAl, inputDecisions, Indeterminate, 4) = *true*
outputDecisions = [NotApplicable, Deny];	**Attribute-list**(D-OComAl, outputDecisions, Deny, 1) **AND** **Attribute-list**(D-OComAl, outputDecisions, NotApplicable, 2) = *true*
ClassOds = **ClassOd(**1**)**← CountMatches(Feed, [Deny], 0). ,	**ClassOd(**D-OComAl, NotApplicable**)** ← CountMatches(D-OComAlFeed, [Deny], 0).
ClassOd(2**)**← IsMemberOf(Deny, Feed). ;	**ClassOd(**D-OComAl,Deny**)**← IsMemberOf(Deny, D-OComAlFeed).
}	/* l'accolade de fermeture est exigée pour confirmer le mapping produit dans la première ligne de ce tableau */

À ces éléments résultant de l'application du tableau du mapping pour tout élément de modélisation, il faut toujours ajouter les autres parties déjà mentionnées du programme logique. Ces parties représentent des règles d'initialisation, les sémantiques des prédicats prédéfinis et les théories supportées.

À noter que selon notre tableau de mapping, on associe toujours le nom d'instance D-OComAl aux valeurs des attributs et des décisions d'entrée pour éviter tout conflit de

noms dans le cas le plus général où plusieurs instances de ComAlNode sont spécifiées pour peupler un seul ADA.

9.6 Exemples de spécifications de ComAl

Quelques exemples de spécifications de ComAl aideront à illustrer l'expressivité de ComLang. On s'intéresse dans cette section à montrer la capacité de ComLang à spécifier les ComAl en général en considérant les ComAl les plus connus. On omet de spécifier les nœuds fils des instances des ComAlNode pour se concentrer sur les ComAl plutôt que sur la structure arborescente de l'ADA.

Nous commençons par un premier exemple simple, où il faut composer deux politiques P1 et P2. Les décisions de P1 et P2 et la décision de sortie sont toutes dans l'ensemble $\sum$ = $\prod$ = {Deny, Permit, Indeterminate, NotApplicable}. Nous mentionnons ici qu'on convient d'appeler $\sum$ et $\prod$ les ensembles de décisions d'entrée et de sortie, mais ces ensembles sont à spécifier pour chaque ComAl. Le ComAl qu'on considère applique une règle simple qui indique que la décision de sortie sera la plus prévalente des deux décisions émises par P1 et P2, l'ordre de prévalence étant défini comme suit : *Deny* > *Permit* > *Indeterminate* > *NotApplicable*.

Pour spécifier ce ComAl, que nous appelons PriorityComAl, nous commençons par définir les ensembles de décisions d'entrée et de sortie et décisions finales et le nombre de systèmes de décision, puis nous spécifions les ClassOd(i) pour i = 1 à 4 comme suit.

9.6.1 Premier exemple : Priorité de décisions

La spécification du ComAl basé sur la priorité de décisions est simple, et elle prend une clause pour chaque décision.

```
CreateInstance(ComAlNode, PriorityComAl)
 {inputDecisions = [Deny, Permit, Indeterminate, NotApplicable];
  outputDecisions = [Deny, Permit, Indeterminate, NotApplicable];
  childrenNumber = 2;

// La logique du choix de décision.

ClassOds = ClassOd(Deny) ← IsMemberOf(Deny, Feed). ,
```

```
 ClassOd(Permit)← CountMatches(Feed, [Deny], 0)
 AND     IsMemberOf(Permit, Feed). ,
 ClassOd(Indeterminate)  ←  CountMatches(Feed,  [Deny,  Permit],  0),
      IsMemberOf(Indeterminate,Feed). ,
ClassOd(NotApplicable)     ←     CountMatches(Feed,     [Deny,     Permit,
Indeterminate],0),     IsMemberOf(NotApplicable, Feed). ;
}
```

9.6.2Deuxième exemple : majorité absolue

Cet exemple est considéré sans préciser le nombre de nœuds fils, les décisions appartiennent aux mêmes ensemble $\sum$ et $\prod$ de l'exemple précédent. Ici, le ComAl précise que la décision de sortie est déterminée par la majorité absolue des voix ; ainsi, c'est la décision de la majorité (plus de 50 % des voix) qui prévaudra. Une décision de sortie Indeterminate sera adoptée quand il n'y a aucune décision ayant la majorité absolue. La spécification en ComLang de ce ComAl est comme suit :

```
CreateInstance(ComAlNode, StrongMajority)
  {inputDecisions = [Deny, Permit, NotApplicable, Indeterminate];
   outputDecisions = [Deny, Permit, NotApplicable, Indeterminate];

/* Logique de décision fournie par ClassOds  */

  ClassOds =
   ClassOd(od)← card(Feed, Total ) AND
      CountMatches(Feed, [od],Count ) AND  Count*2 > Total. ,

   ClassOd(Indeterminate)← card(Feed, Total ) AND
    CountMatches(Feed, [Deny],Countd ) AND  Countd*2 ≤ Total AND
    CountMatches(Feed, [Permit],CountP ) AND  CountP*2 ≤ Total AND
    CountMatches(Feed, [NoteApplicable],CountNA ) AND  CountNA*2 ≤ Total
    .;
}
```

Remarquons que la spécification de ce ComAl est définie indépendamment d'une valeur particulière de l'attribut childrenNumber et children. Cela permet de réutiliser ce code dans toute instance de ComAlNode appliquant un ComAl de forte majorité à ses nœuds fils dans un ADA.

9.6.3 Troisième exemple : Deny-overrides

La spécification ComLang de Deny-overrides a déjà été fournie à la fin de la Section 9.6.2. On mentionne ce ComAl car il est assez connu dans le domaine contrôle d'accès, seulement pour indiquer que l'expressivité de ComLang est suffisante pour le spécifier.

9.7 Caractéristiques de ComLang

Nous avons élaboré ComLang avec les objectifs exprimés dans l'ensemble des exigences énoncées dans la Section 9.3. Après avoir introduit le métamodèle et la syntaxe de ComLang, nous allons évaluer la satisfaction de ces exigences et ce dans l'ordre selon lequel elles ont été mentionnées dans la Section 9.3. Ainsi, dans ce qui suit nous présenterons les différentes caractéristiques de ComLang.

9.7.1 Satisfaction des exigences projetées

1. Expressivité :
    a. Expression des ComAl les plus utilisés : nous avons vu dans les exemples de la section précédente que les ComAl basés sur le vote et la priorité peuvent être spécifiés dans ComLang. Concernant les ComAl définis dans XACML, nous avons fourni la spécification du ComAl Deny-overrides à titre d'exemple, et nous avons pris le soin de vérifier que tous les autres peuvent être facilement exprimés.
    b. L'expressivité de ComLang est garantie dans le cas où le nombre de systèmes de décision est fini et le nombre des décisions est aussi fini. En effet, le nombre de séquences de décision d'entrée est alors fini, et on peut créer dans ClassOds pour chaque séquence une clause qui en spécifie la décision de sortie selon le ComAl.
    c. Pas de limitation à un domaine d'application particulier: nous notons ici que ComLang peut être utilisé pour spécifier des algorithmes composant des décisions créées selon les besoins du domaine d'application, par exemple appartenant à l'ensemble {Permit, Deny} seul ou également à des ensembles plus étendus de décisions comprenant par exemple des

décisions comme « ApprouverUnDocument » ou « Delete ». Les systèmes de décision impliqués ne doivent pas partager un même ensemble de décisions communes. Pour chaque ComAlNode on peut spécifier les décisions en entrée (inputDecisions) et les décisions en sortie (outputDecisions).

d. Conscience de la source de décision : Il est clair que cette exigence est satisfaite car les décisions à composer sont désignées par une liste dite Feed, où le rang de chaque décision dans Feed indique le numéro du nœud fils source de cette décision.

e. Valeurs d'entrée qui ne sont pas des décisions : l'expression de ClassOds peut contenir des prédicats impliquant des individus représentant des instances de tout élément de l'IM.

2. La Composition de ComAl : Un autre aspect de l'expressivité réside dans la prise en charge de plusieurs ComAl qui sont appliqués au niveau de plusieurs ComAlNode de l'ADA. Ainsi, les résultats de composition de plusieurs décisions peuvent eux-mêmes être traités par un nouveau ComAl d'un ComAlNode de niveau supérieur dans l'arbre ADA. Ceci est le cas lorsque les nœuds fils ont eux-mêmes des nœuds fils, et par conséquent les sorties de certains ComAl sont des entrées d'autres ComAl : c'est ce qu'on appelle la composition de ComAl.
3. La satisfaction des exigences de validation et de vérification de propriétés (V &V) est étroitement liée à et dépend de mapping de MACL vers le langage CLP. Le chapitre suivant sera consacré à l'aspect V&V de MACL comprenant aussi ComLang.

9.7.2 Modularité et composition de ComAl

La composition de ComAl permet de composer des décisions autant de fois qu'il y a de nœuds internes aux différents niveaux dans l'ADA. Toute instance de ComAlNode et de ses nœuds fils peuvent être isolés et analysés comme un système de décision que nous pouvons spécifier séparément pour le tester, l'analyser et éventuellement l'intégrer avec d'autres systèmes de décision dans un nouveau ADA. Ainsi, ComLang se prête bien à la

réutilisation de modules et prend en charge le raffinement et la modularité. Ceci sera convenable pour une meilleure procédure progressive de V & V qui fera l'objet du chapitre suivant.

En pratique, décomposer une politique de contrôle d'accès en plusieurs politiques plus simples, ou définir une politique comme une composition d'autres politiques, permet la modularité dans les spécifications de contrôle d'accès. Cela justifie notre point de vue d'une spécification de contrôle d'accès complexe comme une structure regroupant les politiques à différents niveaux selon un ADA.

Enfin, à la lumière du modèle concret de ComLang et des exemples fournis, on peut voir à quel point ComLang satisfait les exigences projetées dans la Section 9.3.

9.8 Travaux connexes

À la lumière de ce qui précède, on peut évaluer la contribution de ComLang, à savoir les points où ComLang a réussi à surmonter les limitations des langages existants pour la spécification des ComAl.

En ce qui concerne la norme XACML [4], ComLang est suffisant pour exprimer les ComAl XACML existants; Il peut aussi en spécifier de nouvelles variantes. Comme exemple illustrant l'expressivité de ComLang, nous avons fourni la spécification du ComAl Deny-overrides d'XACML. En outre, ComLang offre un moyen de validation, de vérification et de composition de ComAl.

Bauer [5] propose une expressivité plus simple dans son style impératif qui diffère du style déclaratif de ComLang. En revanche, la validation et la vérification ne sont pas considérées du tout dans son travail.

Le langage basé sur un modèle d'automate de [7] et le langage du projet Ponder [59] se limitent à la spécification de composition de deux décisions, et la composition de plus de deux décisions n'est pas réalisable, sauf si elle peut être spécifiée comme l'application successive de composition des deux décisions. La limitation de l'expressivité est facile à

remarquer si on essaye d'exprimer le ComAl de vote que nous avons spécifié dans ComLang dans la Section 9.6.2. En outre, les moyens de validation sont limités à la validation syntaxique.

Les ComAl utilisés dans les systèmes de reconnaissance de caractères qui sont basés sur le classement et le degré de fiabilité ne sont pas abordés dans les langages de spécification de composition de décisions, néanmoins Kam-Ho et al. [38] les discutent sans proposer un langage de spécification formelle. En revanche, ComLang peut prendre en charge le classement et la fiabilité des systèmes de décision en utilisant une pondération des systèmes de décision car une expression ClassOd(i) peut inclure des conditions et des opérations sur les nombres. ComLang peut aussi exprimer la dépendance d'une pondération de systèmes de décision et des attributs de contexte.

9.9 Conclusion

On a vu dans ce chapitre que ComLang est un langage spécifié par un métamodèle et qui est assez expressif et qui a son mapping vers le langage cible. ComLang fait partie de MACL bien sûr, mais il peut aussi être utilisé séparément pour spécifier sans ambiguïté des ComAl et des ADA pour des systèmes de décision d'autres domaines d'application.

L'importance du mapping de ComLang réside dans la possibilité de vérifier des propriétés et de valider les spécifications écrites dans ComLang. Le chapitre suivant sera consacré à l'explication le volet V & V de ComLang et de MACL.

Chapitre 10 Vérification et validation dans MACL

Dans ce chapitre nous présenterons les moyens de validation et de vérification de propriétés (V & V) dans MACL et sa composante ComLang. Ces moyens sont basés sur les le mapping des énoncés de MACL vers CLP. Dans la première section nous introduisons les outils permettant l'automatisation des tâches de V & V, ainsi que la démarche pour spécifier des propriétés à vérifier ou valider avec des exemples. Ces exemples portent sur des spécifications de contrôle d'accès en instances des ACMM qui ne dépendent pas de ComAl. La deuxième section considère notre perception au Chapitre 9 des propriétés à valider d'un ComAl écrites en ComLang, et ce en vue d'y appliquer les moyens de V & V. La troisième section montre comment profiter du mapping des spécifications en ComLang des ComAl pour comparer deux ComAl. Nous concluons ce chapitre en pointant vers de nouvelles pistes d'exploitation de l'aspect formel de nos spécifications.

10.1 Prise en charge de la vérification de propriété et de la validation

Pour augmenter le niveau de confiance qu'un énoncé MACL est exempt d'erreur, on s'intéresse à deux aspects : (1) la vérification qui permet de s'assurer qu'une propriété quelconque est satisfaite par l'énoncé MACL; (2) la validation qui est une vérification de certaines propriétés prédéfinies applicables à un énoncé MACL et qui sont considérées

nécessaires pour qualifier cet énoncé de non erroné (valide). La propriété de cohérence en est un exemple.

À noter que nous sommes conscients du fait que comparé au CLP, FOL est plus prometteur au niveau de la vérification et la validation et ce via ses règles d'inférence et son expressivité libérée de la forme restrictive de *definite clause* du CLP. De plus, notre travail conceptuel prépare le terrain pour un travail futur pouvant exploiter ce potentiel du FOL. Ainsi, dans la suite de ce chapitre nous allons nous dissocier des restrictions du CLP et des limitations de l'implémentation présentée au chapitre 8, pour envisager la V & V dans un cadre du FOL. Dans le reste de ce chapitre, pour des raisons de simplicité et par abus de langage, on désignerait par énoncé FOL les *definite clauses* du CLP produites par les mapping, ceci est justifié par le fait qu'une *definite clause* n'est autre qu'une forme restrictive de FOL.

10.1.1Outils de décision de satisfiabilité et vérification de propriétés

Selon notre approche de V & V, une propriété à vérifier est spécifiée en un énoncé en FOL. Désignons cet énoncé par *P*. La vérification de cette propriété consiste à étudier la satisfiabilité de *P*. *P* est une formule logique formée de prédicats reliés avec des opérateurs logiques (ex. conjonction, disjonction, négation) en plus de quantificateurs universels et existentiels. Certains de ces prédicats sont prédéfinis dans des théories mathématiques comme celles des nombres rationnels, des listes et des vecteurs. Les outils SMT [39] et [56] introduits dans la Section 7.2 prennent en charge les formules logiques comme celles spécifiant une propriété *P*. Chaque outil SMT prend en charge des prédicats et des opérateurs prédéfinis pour certains types spécifiques. La plupart de ces outils prennent en charge la comparaison de nombres rationnels, la vérification de cardinalité de listes, la fusion de listes, la concaténation de chaînes de caractères, etc. Les outils SMT prennent en entrée une base de connaissance formées de formules logiques vraies et une formule logique à considérer pour retourner une réponse spécifiant les valeurs de variables qui rendent cette dernière formule logique vraie. Ceci peut être une preuve de vérité de *P* quand cette propriété a la forme d'une assertion d'existence (utilisant $\exists$) de valeurs de variables qui rendent une composition de prédicats *vrai*. En

effet, considérons un exemple où *P* s'énonce par « ∃ x Q(x) », alors il suffit de prouver la satisfiabilité de Q en trouvant au moins une valeur de *x* qui satisfait *Q* pour avoir une preuve que *P* est *vrai*.

En revanche, un moyen de preuve de vérité d'une propriété *P* qui utilise un quantificateur universel (∀) serait de prouver la satisfiabilité de NOT *P* qui utilisera un quantificateur existentiel (∃). En effet, considérons un exemple où *P* s'énonce par « ∀ y R(y) », alors on peut écrire NOT P comme « ∃ y NOT R(y) ». La satisfiabilité de NOT R(y) serait donc une preuve que NOT P est *vrai*, car la valeur satisfaisant NOT R(y) qu'on appelle *y'*, donne un contre-exemple pour *P* car ∃ y= y' NOT R(y') est *vrai*.

Ajoutons à cela que dans certains cas nous pouvons trouver seulement certaines valeurs de variables qui satisfont *P*, alors nous pouvons assurer que *P* est *vrai* si on se limite à ces valeurs trouvées. Nous pouvons aussi trouver certaines valeurs de variables pour lesquelles Non *P* est *vrai*, alors pour ces valeurs de variables *P* est fausse. Ainsi, moyennant l'examen de satisfiabilité de *P* et de sa négation, nous pouvons déterminer des ensembles de valeurs des variables où nous connaissons la valeur de vérité de *P*.

Une alternative aux outils SMT est celle des outils de Constraint Logic Programming [8] comme l'outil SWI-Prolog que nous avons utilisé dans notre implémentation expliquée au chapitre 8. De tels outils assurent aussi la décision de satisfiabilité d'une formule logique mais avec une restriction sur la forme des formules logiques qui sera celle d'une clause formée d'une tête et d'un corps comme expliquée dans la Section 7.2. Dans la suite de notre travail, nous désignons les outils de Constraint Logic Programming et les outils SMT par *outils de décision de satisfiabilité*.

À noter que chacun des outils exige des règles syntaxiques qui lui sont spécifiques. D'où nous avons opté de spécifier les énoncés FOL en formules logiques formées comme mentionné plus haut de compositions de prédicats avec prise en charge de prédicats et d'opérateurs de certaines théories mathématiques en plus des quantificateurs. De plus, pour des raisons de lisibilité nous utilisons dans certains cas la notation en clauses

formées de têtes et de corps. Ceci est justifié car une clause est une forme particulière de représentation d'une formule logique comme composition de prédicats moyennant les opérateurs de conjonction, de disjonction et de négation.

10.1.2 Vérification de propriétés

Dans la Section 9.5.2 nous avons montré qu'une spécification de contrôle d'accès est un énoncé en MACL mappable vers un énoncé en langage cible qu'on désigne par *E*. La vérification de propriétés des spécifications de contrôle d'accès et la validation des spécifications sont élaborées au niveau de l'énoncé FOL car il est le plus approprié pour ces tâches. En effet, si on considère une propriété *P* portant sur les instances des ACMM et/ou de tout autre élément dérivant par instanciation du métamodèle de MACL (donc mappable vers FOL), alors *P* est exprimable en FOL en respectant les règles de mapping des éléments de modélisation vers FOL. *P* est alors une proposition logique qui s'évalue soit à *vrai* soit à *faux*.

La procédure de vérification d'une propriété P consiste à spécifier *P* en FOL et à prouver ensuite qu'elle est soit une proposition vraie soit une proposition fausse. La preuve utilise les règles d'inférence du FOL et le mapping vers FOL des exigences de contrôle d'accès *E* ainsi que des propositions dont on connaît les valeurs de vérité et qu'on peut appeler les *assertions de contexte*. Cette application des règles d'inférence du FOL sur *E*, *P* et les assertions de contexte peut être appelée une *composition* de ces trois éléments. Pratiquement on se sert d'un outil de décision de satisfiabilité qui accepte en entrée (avec la commande de « compilation » ensuite « consulte » pour certains outils) ces trois éléments sous forme d'un ou de plusieurs fichiers d'énoncé en FOL.

Au-delà de la vérification d'une propriété *P,* on peut être intéressé à répondre à des questions cherchant à trouver une relation logique entre *P* et *E*. On peut vérifier si :

a) *P* est une conclusion de *E*, alors (NOT *E* OR *P*) est *vrai*.

b) *E* est une conclusion de *P*, alors (NOT *P* OR *E*) est *vrai.*

c) *E* et *P* sont équivalentes quand a) et b) sont *vrais.*

d) *P* et *E* donnent une contradiction, alors (*P* AND *E*) non satisfiable.

e) Ni a) ni b) ni c) et ni d).

La vérification de ces cinq points sera appelée *la recherche de lien logique.* À noter que l'application de la recherche de lien logique dans le cas où *P* exprime tout un règlement s'inscrit dans le cadre de vérification de conformité des exigences *E* à ce règlement.

Remarquons ici que le point a) de la recherche de lien logique répond directement à la question suivante: Si on admet que les exigences de contrôle d'accès sont respectées est-ce qu'on peut déduire que la propriété considérée est vraie ? Le point b) nous indique que la propriété *P* est plus forte que les exigences de contrôle d'accès, dans le sens qu'elle est plus exigeante et que *P* remplace *P* et *E*. Le point c) représente le cas où a) et b) sont *vrais,* alors *E* et *P* sont équivalentes et conséquemment *P* est une condition nécessaire et suffisante pour que *E* soit vrai, et inversement. Le point d) indique que si on satisfait aux exigences de contrôle d'accès, il sera impossible d'avoir *P vrai*. Le point e) indique carrément que nos efforts de recherche de lien logique, conduisent à un résultat qu'on peut qualifier de neutre ou d'échec d'assurer l'existence d'une relation entre *P* et *E* exprimées par l'un des points a), b), c) et d). Ce résultat neutre n'est pas exclu en logique, il suffit de considérer les deux propositions « Il fait beau » et « c'est le matin » et d'y appliquer la recherche de lien logique. Il est évident qu'on n'a pas à vérifier c) si a) ou b) n'est pas *vrai*.

Exemple de spécification d'une propriété

Comme exemple de spécification en FOL d'une propriété, considérons celui du CW1 de la Section 7.3.2, et plus précisément la propriété qui s'énonce par « Il existe un sujet qui a déjà accédé à deux objets, un appartenant à la classe ClassPromutuel et l'autre appartenant à la classe ClassSunLife ». Pour vérifier cette propriété, il faut la réécrire dans le même langage cible de MACL, ce qui donne un énoncé qu'on désigne par *Px* (Au début de cette section, nous avons désigné la propriété à vérifier par *P*, mais dans le cas des exemples de propriétés à vérifier nous utilisons *Px, Py, …,* etc. Il en sera de même pour *E* et *Ex, …,* etc.). La valeur de vérité de *Px* est à étudier en tenant compte de l'énoncé résultant du mapping des spécifications en MACL de CW1 vers le langage cible FOL.

Px s'énonce en FOL comme suit :

```
∃ s, x, y

 IsOfType (s, Subject) AND IsOfType (x, Object) AND IsOfType (y,
Object) AND Link (s, has Accessed, x) AND Link (s, has Accessed,
y) AND Link (x, belongsTo, ClassPromutuel) AND Link (y,
belongsTo, ClassSunLife).
```

D'autre part, *Ex* représente la spécification en FOL du CW1, et il est obtenu en appliquant le mapping aux spécifications de CW1 présentées dans la Section 7.3.2. L'essentiel de ce mapping consiste en le Clause Attribute spécifique aux instances de MMCA de Chinese Wall et qui fournit la logique de décision. Ce Clause Attribute pour cette instance dite CW1 ayant un DecisionHandler qu'on appelle CW1DH sera d'après la Section 7.2.2 spécifiant la logique des décisions des MMCA comme suit (on respecte les mêmes conventions de notation décrites dans la Section 7.2 et Nous rappelons que selon ces notations A ← B est l'équivalent de (NOT B OR A) et la virgule « , » représente un AND) :

```
ReturnedDecision (CW1DH, Deny) ←
   Link(CW1DH, uses, SomeConflictGroup),
   Link(SomeConflictGroup, composition, Class1),
   Link(SomeConflictGroup, composition, Class2),
   Link(CW1DH, uses, Class1), Link(CW1DH, uses, Class2),
   ¬Link(queriedObject, belongsTo, Class2),
   Link(queriedObject, belongsTo, Class1),
   Link(queryingSubject, hasAccessed, Class2),
   QueryingSubject(queryingSubject),
   QueriedObject(queriedObject).
```

Le mapping complet des spécifications de contrôle d'accès comprend, entre autres, des clauses qui assurent que ClassPromutuel et ClassSunLife sont de type ConflictClass, et qui spécifient les objets qui y appartiennent.

Ceci étant, vérifions la propriété *Px*. On peut démontrer par l'absurde que *Px* n'est pas *vrai.* Nous supposons donc que *Px* est *vrai* et nous considérons alors les points suivants :

1) On fait l'hypothèse que le sujet *s* mentionné dans l'énoncé de *Px* ne peut pas accéder simultanément aux deux objets *x* et *y*, mais il faut qu'il accède à *x* ensuite à *y* ou inversement. Notons ici que *x* et *y* possèdent des rôles symétriques dans les différents énoncés logiques. Nous allons supposer que l'accès à *x* précède l'accès à *y*.
2) On fait aussi les hypothèses suivantes :
    a. Une décision de *Deny* d'accès d'un sujet quelconque *A* à un objet quelconque *B* permet de nier tout accès ultérieur ou antérieur de *A* à *B*. La décision *Deny* résulte de la vérité des prédicats ReturnedDecision(CW1DH, Deny), QueryingSubject(A) et QueriedObject(B); cette décision implique alors une négation du prédicat Link (A, hasAccessed, B). Cette hypothèse résulte du fait qu'on admet que la violation de *Ex* n'est pas possible avant son application, et du fait qu'une fois le sujet *A* a été refusé l'accès à un objet *B*, il lui sera impossible d'y accéder ultérieurement tant que l'instance du Chinese Wall s'applique.
    b. Si un sujet *A* accédé à un objet *B* on peut assurer qu'il a accédé à la classe contenant cet objet. Ceci peut être exprimé par:
    Link(A, hasAccessed, ConflictClassX) ←Link (A, hasAccessed, B), Link(B, belongsTo, X).
3) D'après 1) et *Px* qui est supposé *vrai* par l'absurde, nous pouvons assurer que *s* a accédé à *x* et ensuite il a demandé l'accès à *y*. Étudions alors la demande d'accès de *s* à l'objet *y*, d'où nous considérons la logique de décision de *Ex* avec :
    a. Remplacement de Class1 et Class2 par ClassSunLife et ClassPromutuel respectivement,
    b. Remplacement de queryingSubject et queriedObject par *s* et *y* respectivement,

c. Remplacement de SomeConflictGroup par ConflictGr (ConflictGr est le groupe de conflit selon l'exemple CW1 présenté dans la Section 7.2.2).
d. Remarquer que d'après *Px,* Link(x, belongsTo, ClassPromutuel) et Link(y, belongsTo, ClassSunLife) s'évaluent à *vrai.*
e. D'après *Px et 2)b.* , Link(s, hasAccessed, x) permet d'assurer la vérité de Link(s, hasAccessed, ClassPromutuel).
f. Nous remarquons que les six premiers prédicats du corps de la clause représentant la logique de décision de *Ex* s'évaluent à *vrai* selon la spécification de l'exemple CW1.

Appliquons maintenant les remplacements et les assertions de 3). Nous avons alors tous les prédicats du corps de la clause représentant la logique de décision de *Ex* qui s'évaluent à vrai, il en résulte que sa tête ReturnedDecision(CW1DH, Deny) s'évalue à *vrai.* La décision d'accès de *s* à *y* est donc *Deny*. En appliquant l'hypothèse 2) a. avec remplacement de de *A* et *B* par *s* et *y* respectivement on aura Link(s, hasAccessed, y) est *faux*. Cela veut dire que *s* ne peut jamais accéder à *y* et assure que *Px* n'est pas *vrai* car dans la conjonction de prédicats de son énoncé, il existe un prédicat qui ne s'évalue pas à *vrai* (c'est Link(s, hasAccessed, y)).

À remarquer que la vérification de la propriété *Px* a bien utilisé ce qu'on a appelé plus haut les assertions de contexte et qui sont représentées par les points 1) et 2).

Dans l'exemple CW1 et dans la suite de ce chapitre traitant d'autres exemples de propriétés, nous nous concentrons sur la vérification d'une propriété plutôt que sur la recherche de lien logique vu que notre thème principal est la V & V.

Exemples de propriétés de complétude et d'accessibilité

Les propriétés à vérifier le plus souvent dans le domaine du contrôle d'accès sont les suivantes :

(1) La complétude : C'est la propriété d'une spécification de contrôle d'accès qui permet de retourner une décision de contrôle d'accès en réponse à toute

demande d'accès. L'incomplétude est la propriété qui représente la négation de la propriété de complétude et qui est aussi mentionnée dans la littérature du domaine de contrôle d'accès. Dans la suite de cette section nous allons vérifier la complétude en vérifiant sa négation « l'incomplétude », car la vérification d'incomplétude se réduit à une recherche de contre-exemple qui correspond à l'étude de satisfiabilité de l'énoncé FOL représentant l'incomplétude.

(2) L'accessibilité à un objet : pour un objet donné, il existe au moins un sujet qui peut y accéder.

Selon notre *procédure de vérification* d'une propriété, le point de départ est de spécifier en FOL la propriété à vérifier. Ainsi, nous allons donner les énoncés logiques correspondant aux propriétés (1) et (2) d'incomplétude et d'accessibilité définies ci-haut.

La formulation logique de l'incomplétude revient à élaborer un énoncé FOL qui stipule qu'il existe une demande d'accès pour laquelle la spécification de contrôle d'accès n'a pas de réponse. On sait que la réponse de contrôle d'accès est celle émise par le nœud racine de l'ADA, d'où l'énoncé suivant dont on a à vérifier la satisfiabilité :

```
∃ q, s, o, a     ∀d

CurrentQuery(q) AND QueryingSubject(s) AND QueriedObject(o) AND
QueryAccessMode(a) AND Root(root) AND
IsOfType(root,DecisionSystem) AND IsOfType(d, Decision) AND NOT
ReturnedDecision(root, d).                              (10.1.2.1)
```

q représente une requête; *s*, *o* et *a* représentent respectivement le sujet, l'objet et le mode d'accès associés à *q*. *root* représente le nœud racine de l'ADA. La proposition ci-haut peut être lue comme une assertion de l'existence d'une requête d'accès représentée par le n-uplet (*q*, *s*, *o*, *a*) pour laquelle aucune décision n'est associable au nœud racine de l'ADA. Dans ce cas, on peut assurer l'incomplétude des spécifications de contrôle d'accès.

Notons ici qu'on a pu inclure dans (10.1.2.1) des prédicats exprimant l'appartenance de chaque individu, représenté par un symbole, à un type spécifique (ex. `IsOfType(s, Subject)`), mais la spécification bien formée de contrôle d'accès (exempte d'erreurs de mapping et de modélisation) permet de déduire le type de tout symbole figurant dans tout prédicat évalué à *vrai* dans la proposition (10.1.2.1). Comme exemple, remarquons que quand (10.1.2.1) est *vrai* on peut en déduire que `QueryingSubject(s)` est *vrai*, d'où l'on peut déduire aussi d'après la troisième ligne du Tableau 7.2-3 du mapping de la spécification de contrôle d'accès que `IsOfType(s, Subject)` est *vrai*. Ainsi, seuls *root* et *d* sont sujets à des vérifications de type incluses dans la proposition (10.1.2.1) car ils sont arguments d'un prédicat sujet d'une négation dans la conjonction formant notre proposition. Cependant, en pratique pour plus de rigueur on pourrait opter d'inclure les prédicats de vérification de type de tous les symboles; ceci pourrait révéler des erreurs commises lors de la spécification des exigences de contrôle d'accès. Dans cette section, et pour des raisons de meilleure lisibilité, on omet d'inclure explicitement tous les prédicats de vérification de type.

Un autre point mérite d'être clarifié, est que la négation d'un prédicat *Pred* avec *NOT Pred* est utilisé dans le sens de négation par l'échec, qui signifie que la dérivation de *NOT Pred* est assurée par l'échec de dérivation de *Pred*. Comme exemple de vérification d'incomplétude, considérons l'exemple de la Section 1.1.3 et rappelons l'énoncé de cet exemple. La décision d'accès à l'achat automatisé de tabac pour un client qui s'est enregistré et identifié, est fonction de certains attributs de ce client, conformément aux règles de contrôle d'accès suivantes :

(1) Un étudiant universitaire peut acheter du tabac: la décision est *Permit.*
(2) Un étudiant au CEGEP ne peut pas acheter du tabac: la décision est *Deny.*
(3) Une personne âgée de plus de 18 ans peut acheter du tabac: la décision est *Permit.*

Cet ensemble de règles n'est pas complet car il ne spécifie pas quelle décision prendre dans le cas où le client n'est ni étudiant, ni âgé de plus de 18 ans.

Cet exemple peut être spécifié comme instance du métamodèle de contrôle d'accès générique décrit dans les Sections 6.2.8 et 7.2.2. La spécification de sa logique de décision est comme suit (on respecte les mêmes conventions de notation décrites dans la Section 7.2 et Nous rappelons que selon ces notations *A* ← *B* est l'équivalent de (NOT *B* OR *A*) et la virgule «,» représente un AND) :

```
/* Pour indiquer que l'ADA est formé d'un seul nœud qui est une
instance de l'élément DecisionHandler faisant partie d'une instance
du métamodèle de de contrôle d'accès générique. On appelle ce seul
nœud de l'ADA root. */

IsOfType (root, DecisionHandler).

Root(root).

// La logique de décision

ReturnedDecision( root, Permit) ← QueryingSubject(s), IsUniv(s).

ReturnedDecision( root, Deny) ← QueryingSubject(s), IsCEGEP(s).

ReturnedDecision( root, Permit) ← QueryingSubject(s), IsOver18(s).

/* s1, …, s8 sont les huit sujets considérés pour couvrir toutes
les combinaisons de valeurs de vérité des prédicats caractérisant
les sujets. */

IsUniv(s1), IsCEGEP(s1), IsOver18(s1).

IsUniv(s2), IsCEGEP(s2), NOT IsOver18(s2).

IsUniv(s3), NOT IsCEGEP(s3), IsOver18(s3).

NOT IsUniv(s4), IsCEGEP(s4), IsOver18(s3).

……

NOT IsUniv(s8), NOT IsCEGEP(s8), NOT IsOver18(s8).

IsOfType(s1, Subject).
```

…..

```
IsOfType(s8, Subject).

/* Le seul objet à considérer est la fonctionnalité d'achat de
tabac qu'on désigne par achatTabac.*/

IsOfType(achatTabac, Object), QueriedObject(achatTabac).

/* Le mode d'accès à considérer est l'exécution de la
fonctionnalité d'achat de tabac et qu'on désigne par exec. */

IsOfType( exec, AccessMode), QueryAccessMode(exec).
```

Pour vérifier l'incomplétude de la spécification de contrôle d'accès de notre exemple, nous allons examiner la satisfiabilité de l'énoncé (10.1.2.1). On remarque que cet énoncé est satisfiable pour une requête *q* qu'on associe au sujet *s8*, à l'objet *achatTabac* et au mode d'accès *exec*. De plus, la spécification de notre exemple, y compris les clauses dont la tête est *ReturnedDecision,* ne permet pas la dérivation de la vérité du prédicat *ReturnedDecision* et ce quelle que soit la décision *d* considérée.

Pour remédier à cette incomplétude on va modifier la spécification de contrôle d'accès en ajoutant que si un sujet n'est ni universitaire, ni au CEGEP et ni âgé de plus de 18 ans, alors la décision de contrôle d'accès est *Deny*. D'où l'ajout de la spécification suivante:

`ReturnedDecision(root, Deny)` ← `QueryingSubject(s), NOT` IsUniv(s), `NOT` IsCEGEP(s), `NOT` IsOver18(s).

D'autre part, la formulation logique de la propriété (2) d'accessibilité doit permettre de vérifier la satisfiabilité d'un énoncé FOL qui stipule l'existence d'une décision *Permit* en réponse à une requête d'accès ciblant l'objet considéré qu'on désigne par *objetAX*. L'énoncé FOL correspondant à cette propriété (2) est le suivant, et son interprétation est assez claire :

```
∃ q, s, a
```

```
CurrentQuery(q) AND QueryingSubject(s) AND QueriedObject(objetAX)
AND QueryAccessMode(a) AND Root(root) AND ReturnedDecision(root,
Permit).                                                (10.1.2.2)
```

Comme exemple de vérification de l'accessibilité selon une spécification de contrôle d'accès, il suffit de considérer le même exemple utilisé pour la vérification de la propriété d'incomplétude, mais en modifiant les sujets de sorte qu'ils soient tous des étudiants au CEGEP. Ainsi, selon la nouvelle spécification de notre exemple sujet de l'étude d'accessibilité, pour n'importe quel sujet *s*, IsCEGEP(*s*) est *vrai*, IsUniv(*s*) est *faux* et IsOver18(s) est *faux*. Pour étudier l'accessibilité de l'objet achatTabac, on vérifie la satisfiabilité de (10.1.2.2) en substituant *achatTabac* à *objetAX*. La vérification de satisfiabilité retournera nécessairement une réponse négative et ceci résulte du fait que pour tout sujet IsCEGEP(s) est *vrai*, et de la clause suivante faisant partie des exigences de contrôle d'accès détaillées deux pages plus haut :

```
ReturnedDecision(root, Deny) ← QueryingSubject(s), IsCEGEP(s).
```

Pour rendre *achatTabac* accessible il suffit d'ajouter un ou plusieurs sujets qui rendent *vrai* le corps de l'une des clauses de la spécification de contrôle d'accès dont la tête est `ReturnedDecision(root, Permit)`. Il suffit donc d'ajouter au moins un sujet qu'on désigne par *sx* et qui satisfait IsUniv(sx) ou IsOver18(sx) et NOT IsCEGEP(sx), et qu'on associe à une requête d'accès. Il en résulte que QueryingSubject(sx) soit *vrai*, et le corps de l'une des deux clauses suivantes des exigences de contrôle d'accès soit *vrai* :

```
ReturnedDecision( root, Permit) ← QueryingSubject(s), IsUniv(s).

ReturnedDecision( root, Permit) ← QueryingSubject(s), IsOver18(s).
```

Ainsi, la tête de l'une de ces deux clauses est *vrai*, en plus des autres prédicats figurant dans (10.1.2.2). D'où nous pouvons assurer la vérité de l'énoncé (10.1.2.2) et par suite la vérité de la propriété d'accessibilité qu'il spécifie.

10.1.3 La validation

La validation est une vérification de certaines propriétés qu'on trouve nécessaires pour qualifier nos spécifications de non erronées (valides). De telles propriétés à vérifier se trouvent au niveau des nœuds de l'ADA et/ou au niveau de la structure complète de l'ADA. On mentionne les validations suivantes:

1. La validation de la structure de l'ADA : la validation de la structure arborescente de l'ADA (la forme) est un test de conformité de l'ADA à son métamodèle. On cherche à détecter les anomalies suivantes :
    - Présence de cycle dans le graphe;
    - Non connexité du graphe;
    - Nœud feuille qui n'est pas instance de DecisionHandler;
    - Nœud qui n'est pas une feuille mais qui n'a pas de ComAl associé.
2. La validation des ComAl (détaillée dans la section suivante) qui consiste à s'assurer de l'existence et de l'unicité d'une réponse pour toute combinaison de décisions.
3. La validation de chaque instance de ACMM: on se base sur les structures et les contraintes fournies par le métamodèle ACMM. Par exemple, selon Bell LaPadula, un sujet ne peut pas avoir plus d'un niveau d'habilitation de sécurité.
4. La validation de cohérence de la logique de décision: on cherche s'il existe une requête pour laquelle une instance de DecisionHandler feuille d'un ADA peut retourner deux décisions différentes. De plus, on examine ce même aspect de non unicité de décision au niveau de la racine de l'ADA. Ceci tient compte du fait que la décision du nœud racine représente celle de l'ADA tout entier. Par exemple, pour une instance de RBACdecisionHandler de l'ACMM de rôles, imaginons qu'un certain rôle permet de fournir pour une requête donnée une décision *Deny*, alors que son sous-rôle permet une autre décision *Permit* pour la même requête.
5. La validation de la cohérence d'un énoncé MACL dans le sens de l'inexistence de contradictions, est abordable en examinant la cohérence de son mapping FOL.

Notons ici que la validation est étroitement liée à la structure de l'ADA qui permet d'examiner soit des nœuds, soit une intégration de nœuds. Par contre, la vérification de propriétés globales de la section précédente considère le mapping complet de l'énoncé MACL sans rentrer dans les détails des structures sous-jacentes. Ce dernier point représente une certaine analogie avec les tests de la boîte noire et de la boîte transparente du génie logiciel.

Exemple de spécification de critère de validation

Comme exemple d'une propriété servant de critère de validation, considérons la validation de cohérence de la logique de décision d'une spécification de contrôle d'accès expliquée au point 4) ci-haut. Nous allons formuler la propriété d'incohérence dans le cas simple d'une spécification avec deux décisions possibles *Permit* et *Deny* au niveau du nœud *root* racine de l'ADA. La propriété d'incohérence peut être formulée comme suit :

```
∃ q, s, a, o

CurrentQuery(q) AND QueryingSubject(s) AND QueriedObject(o) AND
QueryAccessMode(a) AND Root (root) AND ReturnedDecision(root,
Permit) AND ReturnedDecision(root, Deny).                (10.1.3.1)
```

Exemple de validation d'une spécification de contrôle d'accès

Pour fournir un exemple de vérification de cohérence d'une spécification de contrôle d'accès nous allons retenir le même exemple déjà traité en vue d'une vérification d'incomplétude dans la Section 10.1.2. La spécification de cet exemple comprend un sujet désigné par s4 qui est étudiant au CEGEP et qui est en même temps âgé de plus de 18 ans, il en résulte que les deux dernières règles de contrôle d'accès (2) et (3) de cet exemple s'appliquent en fournissant deux décisions contradictoires de « Deny » et de « Permit ». En effet, IsCEGEP(s4) et IsOver18(s4) s'évaluent à *vrai*, et si on considère la requête où s4 demande l'accès à achatTabac, alors `QueryingSubject(s4)` est *vrai*. Appliquons maintenant les deux règles suivantes de la spécification de contrôle d'accès de notre exemple en substituant *s4* à *s* :

```
ReturnedDecision(root, Deny) ← QueryingSubject(s4), IsCEGEP(s4).
```

```
ReturnedDecision(root, Permit) ← QueryingSubject(s4),
IsOver18(s4).
```

Il en résulte que les deux têtes `ReturnedDecision(root, Deny)` et `ReturnedDecision(root, Permit)` des deux clauses ci-haut s'évaluent à *vrai*.

Ceci étant, en plus du raisonnement précédent dévoilant l'incohérence, on peut aussi démontrer que l'exemple considéré est incohérent en se basant sur la proposition (10.1.3.1) spécifiant formellement l'incohérence. (10.1.3.1) commence par le quantificateur existentiel de quatre individus, d'où nous commençons par considérer une requête *q* et nous l'associons au sujet *s4,* à l'objet *achatTabac* et au mode d'accès *exec*. Ceci assure la vérité de la conjonction des prédicats :

```
CurrentQuery(q) AND QueryingSubject(s4) AND
QueriedObject(achatTabac) AND QueryAccessMode(exec).
```

De plus, on a déjà démontré la vérité de `ReturnedDecision(root, Deny)` et `ReturnedDecision(root, Permit)`. La spécification de contrôle d'accès fournie dans la section précédente comprend déjà une clause qui assure la vérité de `Root(root)`. D'où l'on peut déduire la vérité de la proposition suivante comme conjonction de prédicats qui s'évaluent tous à *vrai*:

```
CurrentQuery(q) AND QueryingSubject(s4) AND
QueriedObject(achatTabac) AND QueryAccessMode(exec) AND
Root(root) AND ReturnedDecision(root, Permit) AND
ReturnedDecision(root, Deny).                          (PropExist)
```

Ceci fournit les données d'une requête *q* dont l'existence et la vérité de la proposition (PropExist) ci-haut démontrent bien la vérité de la proposition (10.1.3.1). La vérité de (10.1.3.1) permet d'assurer l'incohérence de notre exemple de spécification de contrôle d'accès à l'achat de tabac.

10.2 Validation dans ComLang

D'après la Section 9.4, la validation d'un ComAl consiste principalement à vérifier trois points : (1) sa cohérence en vérifiant l'unicité de la décision de sortie résultant d'une composition de décisions; (2) sa complétude qui est assurée si toute composition de décisions résulte en une certaine décision (y compris la décision *Indeterminate* ou *NotApplicable* car c'est la complétude du ComAl qui est envisagée et non pas celle de la spécification de CA); (3) La non redondance qui est assurée si pour toute décision de sortie on peut trouver au moins une combinaison de décisions d'entrée qui lui est associée par le ComAl. Pour formuler plus précisément ces points, on adopte les termes définis dans la Section 9.4; Π désigne l'ensemble des décisions de sortie Od_f, avec *f* allant de 1 à *odn*. Rappelons aussi que Σ^n désigne l'ensemble de tous les n-uplets possibles de *n* décisions d'entrée d'un ComAlNode en provenance des *n* systèmes de décision fils.

Notons ici que la complétude d'un ComAl est une propriété bien définie d'après la Section 9.4, car on définit un ComAl comme une application qui doit faire correspondre une décision à toute liste de décisions. Ainsi, pour un ComAl, la complétude est une propriété qui est nécessaire afin qu'on puisse le qualifier de non erroné (valide). Conséquemment, la vérification de complétude d'un ComAl est une validation.

10.2.1 Cohérence de spécification d'un ComAl

Pour une instance de ComAlNode qu'on nomme myComAlNode la condition de cohérence peut être formulée par :

$\forall$ *f, k* entiers naturels inférieurs à *odn* avec $f \neq k$ alors les deux ensembles de combinaison de décisions d'entrée qui rendent respectivement ClassOd(myComAlNode, f) et ClassOd(myComAlNode, k) vrais doivent être deux ensembles disjoints. La validation de la cohérence revient à s'assurer de la non existence de couples de décisions Od_f et Od_k avec $f \neq k$ (décisions distinctes) pour lesquelles les classes de décision définies par les prédicats ClassOd (myComAlNode, f) et ClassOd(myComAlNode, k) ne sont pas disjointes. Ainsi pour détecter une incohérence il suffit de trouver la satisfiabilité de :

∃ f, k, myComAlNodeFeed

f ∈ N, k ∈ N, f ≠ k, myComAlNodeFeed ∈ Σ^n

(ClassOd(myComAlNode, f) AND ClassOd(myComAlNode, k))

Pour comprendre ce que représente myComAlNodeFeed on peut se référer à la dernière ligne du tableau 9-5.2 où on explique que la concaténation du nom du ComAl et du mot « Feed » représente la liste de décision d'entrée considérée pour trouver une décision de sortie. Cette liste remplace le nom de variable « Feed » dans la spécification du ComAl pour trouver la décision de sortie.

La propriété d'incohérence de spécification d'un ComAl qu'on nomme ComAlx peut être formulée par un prédicat qu'on nomme InvalidateConsistency et qui s'applique à ComAlx avec :

InvalidateConsistency(ComAlx) ← ClassOd(ComAlx, f), ClassOd(ComAlx,k), f ≠ k. (10.2.1.a)

Comme exemple d'incohérence d'un ComAl, considérons ComAlv un ComAl de vote qui adopte la décision dont le nombre d'occurrences dans la liste des décisions d'entrée est plus grand que le tiers de la longueur de cette liste. ComAlv n'est pas cohérent, car on peut avoir, dans une liste de décisions d'entrée, deux décisions distinctes Di et Dj dont chacune a un nombre d'occurrences supérieur au tiers de la longueur de la liste. Selon notre explication de ComAlv, on peut formuler ClassOd comme suit:

ClassOd(ComAlv, i) ← Card(Feed, Total), CountMatches(Feed, [Di], Counti), Counti *3 > Total.

Pour montrer que ComAlv est incohérent il suffit de trouver une affectation de valeurs de variables qui font que InvalidateConsistency(ComAlv) soit *vrai*. La clause (10.2.1.a) appliquée à ComAlv s'écrit:

InvalidateConsistency(ComAlv) ← Card(Feed, Total),
CountMatches(Feed, [D_f], Countf), Countf *3 > Total ,
Card(Feed, Total), CountMatches(Feed, [D_k], Countk),
Countk *3 > Total , f ≠ k .

Un outil de décision de satisfiabilité (Section 10.1.1) peut être utilisé pour trouver les valeurs de variables qui satisfont la clause ci-haut. Il peut alors trouver une valeur de *ComAlvFeed* rendant InvalidateConsistency(ComAlv) *vrai*. En effet, InvalidateConsistency(ComAlv) est *vrai* si on choisit une valeur de *Feed* correspondant à la liste [D3, D3, D3, D4, D4, D4]. Il en résulte que pour *f= 3* et *k=4* l'expression suivante s'évalue à *vrai* :

```
Card(Feed, Total ) AND CountMatches(Feed, [D3], Countf) AND
Countf *3 > Total   AND ( Card(Feed, Total ) AND
CountMatches(Feed, [D4], Countk) AND Countk *3 > Total AND 3 ≠ 4
```

On aura donc une seule liste de décision qui permet deux décisions de sortie différentes *D3* et *D4*. On en déduit que ComAlv n'est pas cohérent comme combining algorithm.

10.2.2 Complétude de spécification de ComAl

La complétude d'une spécification d'un ComAl assure que toute composition de décisions d'entrée doit résulter en une décision de sortie. Cette assertion se formule pour un ComAl associé à un ComAlNode qu'on nomme myComAlNode :

$\forall$ myComAlNodeFeed $\in \Sigma^n$ $\exists$ f $\in$ N

$1 \leq f \leq$ odn AND ClassOd(myComAlNode, f)

Comme exemple de complétude, considérons la spécification de ComAl1 de la Section 6.3.1 qui est déjà spécifié par les associations suivantes reliant une ou plusieurs listes de trois décisions d'entrée à une décision de sortie:

a. [Permit, Permit, Deny] OR [Deny, Permit, Permit] → Permit
b. [NotApplicable, Permit, Permit] OR [Permit, Permit, NotApplicable] OR [NotApplicable, Permit, NotApplicable] → Permit
c. [Permit, Permit, Permit] → Permit
d. [NotApplicable , NotApplicable, NotApplicable] → NotApplicable

e. Toutes les listes de décision restantes entrainent un *Deny* et elles sont (un « * » indique une décision quelconque) :
   i. [Deny, *, Deny] → Deny
   ii. [*, Deny, *] → Deny
   iii. [NotApplicable, *, Deny] → Deny
   iv. [*, NotApplicable, *]→ Deny

Ceci étant, si on suppose que la spécification de ComAl1 ne contient pas le point e. ii de cette même section, alors on peut trouver au moins un n-uplet comme (*Deny*, *Deny*, *NotApplicable*) qui n'appartient à aucun ensemble ClassOd(f). C'est bien une incomplétude car le n-uplet des décisions (*Deny*, *Deny*, *NotApplicable)* ne permet pas de fournir une décision de sortie en appliquant les spécifications privées de e. ii.

Les deux conditions de cohérence et de complétude sont satisfaites si et seulement si : les ensembles ClassOd(f), avec *f* allant de 1 à *odn*, forment une partition de Σ^n. Notons que toute partition d'un ensemble permet de définir une relation d'équivalence sur cet ensemble; et cette relation est « appartient au même élément de la partition ». Donc tout revient à définir une relation d'équivalence sur l'ensemble Σ^n, avec association d'une décision distincte à chaque classe d'équivalence.

Il faut noter que dans le cas d'une spécification de contrôle d'accès écrite en MACL et interprétée par un ADA, la complétude de la spécification de contrôle d'accès signifie qu'elle est suffisante pour retourner une décision de contrôle d'accès en réponse à toute demande d'accès.

10.3 Comparaison de deux ComAl

La comparaison des ComAl est possible dans le langage cible FOL. Nous pouvons formuler la comparaison en termes de prédicats qui impliquent les mapping des ClassOd(i) de deux instances de ComAlNode. Le prédicat Divergence (ComAl1, ComAl2) de la clause suivante permet de vérifier l'existence de valeurs de variables qui révèle une situation où ComAl1

et ComAl2 donne deux décisions de sortie différentes pour une même liste de décisions d'entrée.

```
Divergence (ComAl1, ComAl2)  ←
 ClassOd(ComAl1, i),  ClassOd(ComAl2, j),  i ≠j , ComAl1Feed =
ComAl2Feed.
```

En prenant en considération les conventions de quantification des variables d'une clause décrites dans la Section 7.2 la clause ci-haut peut s'écrire comme suit :

```
∀ ComAl1, ComAl2

Divergence (ComAl1, ComAl2) OR

NOT ( ∃ i, j, ComAl1Feed, ComAl2Feed

 ( ClassOd(ComAl1, i) AND ClassOd(ComAl2 , j) AND i ≠j AND ComAl1Feed =
ComAl2Feed ))
```

Notons que la valeur de vérité du ClassOd dépend de la liste des décisions d'entrée qui est désignée selon le mapping par une concaténation du nom du ComAlNode et du mot « Feed ». Un outil de décision de satisfiabilité peut trouver les valeurs qui satisfont le prédicat *Divergence* si elles existent.

10.4 Conclusion

Une spécification MACL est mappable vers le langage cible FOL sans perte d'information. Il en résulte que pour toute question écrite en FOL et portant sur une spécification de contrôle d'accès, on aura des moyens pour y chercher des réponses si ces dernières existent dans la spécification MACL. Cela reste tout de même dans des limites de la complétude de FOL et des capacités de traitement disponibles. Néanmoins, ces moyens offrent une assistance automatisée appréciable surtout quand la spécification de contrôle d'accès comprend un grand nombre d'énoncés qui dépasse les capacités humaines non automatisées d'assimilation et d'analyse.

On a montré, à titre d'exemple, la vérification de certaines propriétés. La vérification d'autres propriétés et d'autres avantages de l'aspect formel des spécifications des

exigences de contrôle d'accès peuvent aussi intéresser un analyste de contrôle d'accès. Par exemple, on peut considérer la réduction d'un ADA à plusieurs niveaux en un ADA à un seul niveau (comprenant des feuilles et un seul nœud de type ComAlNode qui est Root). Ça prend une vérification de non divergence entre une composition de plusieurs ComAl et un seul ComAl. On peut même envisager l'automatisation de la spécification d'un ComAl qui remplacerait une composition de plusieurs ComAl. Cela sera utile pour réduire la taille d'un ADA et optimiser tout traitement automatique de la spécification qu'elle représente.

Chapitre 11 Étude de cas

Dans ce chapitre, on présente le cas d'une entreprise ayant des besoins en contrôle d'accès basés sur plusieurs modèles. L'utilisation de MACL pour spécifier ces besoins va mettre en évidence la réduction de la complexité et la modularité qui représentent les principaux atouts de ce langage. Le cas de l'entreprise considérée est réaliste, sans toutefois embrouiller le lecteur par des détails administratifs ou par un grand nombre d'objets et de sujets similaires.

11.1 Vue d'ensemble de l'entreprise

On considère le cas d'un bureau d'études d'ingénierie qu'on nomme Hautes Études Techniques (HET). Il est spécialisé dans les projets d'infrastructure de haute technicité. Les projets envisagés pouvant être, entre autres, des projets de conception de structures d'ouvrages spéciaux comme les tours, les dômes, et les ponts; ou des projets de conception de centrales électriques hydroliennes, solaires, à combustibles ou nucléaires. Le développement de ces projets fait appel à des expertises multidisciplinaires et se caractérise par la production d'un grand nombre d'artéfacts : études de faisabilité, cahiers des charges, devis, plans de conception, planning d'exécution, etc. La bonne gestion de ces artéfacts, notamment de leur accessibilité pour d'éventuelles modifications et de leur protection contre tout accès inapproprié, constitue un vrai défi.

Puisque nous sommes donc en présence d'un problème de contrôle d'accès, il importe, dans un premier temps, de bien identifier les sujets et les objets concernés. Au niveau des sujets, on identifie les entités suivantes :

- Chief Executive Officer (CEO) et Project Manager interdépartemental (PM-interDep).
- Les départements d'informatique (IT), de mécanique et électricité (MEP), de structure (STR) et d'architecture (ARC).
- Chaque département comprend : un responsable (Head), des ingénieurs (Eng), des Project Managers (PM), un Quality Assurance Officer (QAO), et des assistants (Asst).

Dans la suite de ce chapitre, on désignera les rôles par des noms de postes parlants pour alléger le texte. Ainsi, le rôle MEPHead désignera le rôle du Head du département MEP. De même, le rôle d'un ingénieur du département IT sera ITEng. Il en est de même pour les noms des employés, par exemple, on désignera par ITEng1 et ITEng2 deux ingénieurs du département IT. Ceci nous évitera de mémoriser les associations entre les noms des employés et leur poste ou département d'appartenance. Ainsi, la liste des sujets (principalement des employés de HET) considérés est comme suit :

Au niveau de la haute direction, on identifie l'ensemble de sujets qu'on désigne par le TopManagement = **{CEO-Hercule, PM-interDep1}**

Au département MEP, on identifie l'ensemble des employés qu'on désigne par MEPdep = **{MEPhead, MEPeng1, MEPeng2, MEPeng3, MEPasst1, MEPasst2, MEPasst3, MEPpM1, MEPqAO1}**

De même, pour le département STR, STRdep = **{STRhead, STReng1, STReng2, STReng3, STRasst1, STRasst2, STRasst3, STRpM1, STRqAO1}**

De même, pour le département IT, ITdep = **{IThead, ITeng1, ITeng2, ITasst1, ITasst2, ITpM1, ITqAO1}**

À ceux-ci, on ajoute les intervenants externes suivants qui auront partiellement accès à certaines ressources de la société HET:

- Consultants avec un SubjectsGroup dit Consultants = **{AllDepConsultant1, ITconsultant1, MEPconsultant2, MEPconsultant1, STRconsultant1}**
- Entrepreneurs de sous-traitance qui constituent un SubjectsGroup, Subcontractors = **{GENsub1, ITsub1, MEPsub1, STRsub1}**

Ayant identifié les sujets, on passe à l'identification des objets. Ils sont classés selon :

- Le département d'appartenance de ces objets.
- L'intégrité des objets en deux catégories : les données de recherche et de développement (RDdata); les données de références (REFdata).
- Le type contractuel ou technique (qu'on appelle docType) : les documents produits des activités de conception ont un docType dit Plan, alors que les documents contractuels ont un docType dit Cont.

Ainsi, si on considère le département MEP, on identifie les groupes de documents par des noms parlant qui indiquent leur contenu : MEPrDdata (pour MEP données de recherche), MEPrEFdata, MEPplan, MEPcont. Il en sera de même pour les autres départements.

Les artéfacts produits par chaque département évoluent selon les quatre statuts présentés ci-dessous.

Les Statuts MasterPlan, InProcess, Execution et AsBuilt seront désignés, respectivement, par les nombres 1, 2, 3 et 4. Cette désignation sera utilisée au besoin comme suffixe dans la nomenclature des objets pour indiquer leur état d'avancement.

La liste suivante des projets en cours chez HET est essentielle pour comprendre les besoins de ce bureau d'études en matière de contrôle d'accès:

- **MEP-Nuclear** (Nuc) : un projet de conception des éléments mécaniques et structurels d'une centrale nucléaire.
- **EWpipeLines** (Ppl) : un projet de conception de pipelines pour la compagnie PetroNord spécialisée dans le raffinement du pétrole.
- **Refinery** (Raf) : un projet de conception d'éléments mécaniques pour la compagnie PetroSand spécialisée dans la prospection et le raffinement du pétrole.
- **InovStruct1** et **inovStruct2 (Inov):** un projet de conception d'une structure d'un pont InovStruct caractérisé par un ensemble de défis techniques exigeant des recherches et de l'innovation. Deux équipes y travaillent. La première équipe se base sur les méthodes déjà bien établies et adoptées dans d'autres projets similaires; les données de ces méthodes font partie du groupe d'objets REFdata. La deuxième équipe se base sur des données dont la fiabilité est à prouver, de

telles données font partie du groupe d'objets RDdata. Les deux équipes produiront séparément deux conceptions, **InovStruct1** et **innovStruct2**.

- **HydroGen (H2) :** un projet de conception d'une centrale hydraulique de génération d'électricité.

Comme déjà fait pour nos sujets, les objets considérés dans notre étude auront des noms parlants. Chaque nom d'objet est une concaténation des abréviations des éléments suivants:

Nom du département, nom du projet, DocType, catégorie d'intégrité, numéro de statut.

Ex. MEPnucPlanRD1 est un objet du département MEP, du projet Nuc, formé de données de plans de conception (pas de contractuel); son niveau d'intégrité est celui des données de recherche et développement; et le suffixe « 1 » indique que son statut est le premier de l'énumération des statuts qui est « MasterPlan ».

On va considérer l'ensemble suivant d'objets qu'on juge suffisant pour les fins d'Illustration des spécifications MACL de notre étude de cas, et ce en les présentant par rapport aux projets d'appartenance comme suit.

Les objets du projet Nuc hors IT, c'est-à-dire qui appartiennent au projet Nuc et qui ne sont pas maintenus et contrôlés directement par le département IT :
MEPnucPlanRD1, MEPnucPlanHET1, MEPnucPlanHET2, MEPnucPlanHET3, MEPnucPlanHET4, MEPnucContHET1, MEPnucContHET3.

Les objets du projet Ppl hors IT :
MEPpplPlanRD1, MEPpplPlanHET1, MEPpplPlanHET2, MEPpplPlanHET3, MEPpplPlanHET4, MEPpplContHET1, MEPpplContHET3.

Les objets du projet Raf hors IT :
MEPrafPlanRD1, MEPrafPlanHET1, MEPrafPlanHET2, MEPrafPlanHET3, MEPrafPlanHET4, MEPrafContHET1, MEPrafContHET3.

Les objets du projet Inov hors IT :
STRinovPlanRD1, STRinovPlanHET1, STRinovPlanHET2, STRInovPlanHET3, STRinovPlanHET4, STRinovContHET1, STRinovContHET3.

Les objets IT :

ITallProjContHET3, ITnucContHET2, ITnucPlanHET3, ITpplPlanHET3, ITrafContHET3, ITinovContRD2, ITinovPlanHET2.

De plus, les dirigeants de HET définissent leurs besoins en contrôle d'accès d'une façon informelle comme suit :

- Les projets doivent être classés selon leur niveau de sensibilité, et seuls les employés ayant les cotes de sécurité convenables y auront accès. Une application de **BLP** est de mise car seuls les employés ayant les cotes de sécurité convenables auront accès aux données de certains projets comme MEP-Nuclear.
- Les données techniques à disposition des équipes de conception doivent être classifiées selon leur niveau d'intégrité. Une application de **Biba** est de mise pour séparer les données de différents niveaux d'intégrité des deux conceptions InovStruct1 et InovStruct2.
- Les clients de HET qui œuvrent dans le même domaine, auront la sûreté qu'on applique le modèle de contrôle d'accès **CW** pour protéger leurs données des conflits d'intérêt, ex. PetroNord et PetroSand.
- En plus des besoins de contrôle d'accès déjà mentionnés, **les rôles** des employés sont à prendre en considération pour accorder une permission d'accès. Les permissions de rôles sont définies à la lumière des responsabilités associées à chaque rôle, c'est-à-dire à chaque poste dans la structure organisationnelle de l'entreprise. Ces permissions aboutiront à des décisions de contrôle d'accès qui seront composées avec d'autres décisions provenant de l'application des autres exigences de contrôle d'accès mentionnées dans les points précédents. Ainsi pour aboutir à une décision de contrôle d'accès l'application de ComAl sera nécessaire. Les permissions de chaque rôle seront spécifiées dans la section suivante; en voici quelques exemples sous forme de recommandations des dirigeants de HET:
    - Pour le rôle Eng, l'accès est permis à un objet ayant le statut Execution ou MasterPlan. Alors que le rôle Asst permet l'accès aux objets dont le statut est InProgress. On réserve la modification des objets du statut AsBuilt au rôle Head. Pratiquement, pour créer les documents AsBuilt, les ingénieurs modifient les documents d'exécution pour les soumettre au Head qui les approuvera comme documents AsBuilt.
    - Les privilèges des rôles de consultants et de sous-traitants observent le principe de compartiments (départements, projets, type de document Plan/Cont) avec accessibilité limitée à la lecture des objets ayant les statuts MasterPlan, Execution et AsBuilt. Les sous-traitants n'ont accès ni aux documents contractuels ni aux documents InProgress. À noter que ces intervenants peuvent avoir un accès à certains documents en suivant des procédures administratives de HET de demandes d'information, mais les règles de contrôle d'accès appliquées systématiquement sont comme déjà mentionnées.

- La résolution de conflits entre les exigences de contrôle d'accès est à spécifier par application de la composition de décisions :
    - Certaines données de recherche (appartenant à RDdata) pourraient être utilisées dans le cadre d'un projet particulier et elles seront associées à ce projet (compartiment de ce projet). De telles données sont toujours de niveau d'intégrité RDdata, mais elles sont tout de même considérées vitales pour un certain projet. Ainsi, elles devront être accessibles par l'équipe de conception de ce projet en se basant sur les rôles des membres de cette équipe. On exige qu'un *Deny* de l'application de Biba soit ignoré si les exigences de Role et du Compartment émettent deux décisions *Permit*. Dans les autres cas, tout *Deny* résultant de l'une des exigences de contrôle d'accès de rôles, de Compartment ou de Biba, se traduira par une décision *Deny*. De plus, si les trois décisions sont des NotApplicable, la décision de sortie sera NotApplicable. On commence par considérer la composition des instances des ACMM de **Role, de Compartment et de Biba** selon les règles de composition mentionnées dans ce paragraphe et qui seront implémentées par un nœud de l'ADA dit **ComAlNode1**.
        - À noter que ceci permet d'exempter les Head des départements et les PM des restrictions d'accès provenant de Biba. Ceci revient au fait que les Head et les PM n'ont pas de restriction de compartiments ni de permission de rôle pour les objets de leur département d'appartenance.
    - Deuxième exigence de règlement de conflit (elle sera implémentée par un nœud racine de l'ADA dit **ComAlNode2**): Après la considération des exigences de Role, Compartment et Biba comme mentionné dans le point précédent, les dirigeant de HET recommandent de respecter sans exception toute exigence de déni d'accès provenant de BLP ou CW. Ceci revient à : (1) l'engagement de HET auprès de ses clients de protéger leurs informations contre tout conflit d'intérêt; (2) l'importance de protéger les informations critiques dont la divulgation est très préjudiciable et porterait atteinte à la sécurité publique. Ceci justifie le recours à l'application d'un ComAl de Deny-overrides, aux instances des ACMM de **BLP, CW et ComAlNode1**.

11.2 Spécification MACL du cas HET

11.2.1 Spécifications informelles structurées selon un ADA

Après avoir présenté les intentions et les préoccupations des dirigeants de HET au niveau du contrôle d'accès, on va procéder à l'étape préparant le passage des ECA informelles de HET vers leurs spécifications en MACL. Cette étape consiste à réécrire les exigences

sous forme d'applications de plusieurs ACMM avec plus de clarté et de structuration, comme suit :

1. Compartiments de Need-to-know : pour permettre l'accès seulement aux sujets concernés, on définit pour chaque **projet**, **département** et **docType** (Contract/Plan) un compartiment. Ainsi, un objet peut être associé à un projet, à un département, et à un type de document, de conception ou contractuel. À noter qu'on désigne par « AllProjects » une association à tous les compartiments de projets. Pareillement, on utilisera les termes AllDepartments et AllDocTypes. Les compartiments des sujets sont formés comme suit :
    a. Asst et Eng sont associés aux compartiments des projets assignés, du département d'appartenance, et au compartiment Plan de docType.
    b. Head et QAO sont associés aux compartiments définis par les projets assignés, le département d'appartenance et AllDocTypes.
    c. Les PM sont associés aux compartiments définis par les projets qu'ils gèrent, AllDepartments et AllDocTypes.
    d. Le CEO est associé aux compartiments définis par AllProjects-AllDepartments-AllDocTypes.
2. Biba :
    a. RDdata et REFdata sont considérés comme deux niveaux d'intégrité tels que RDdata est inférieur à REFdata. À noter que chaque Eng ou Asst est assigné à l'un de ces deux niveaux selon ses fonctions qui sont soit en conception basée sur les données fiables (REFdata) soit en RD.
3. BLP : Les sujets ont des niveaux d'habilitation de sécurité et les données des projets ont des niveaux de sensibilité comme suit :
    a. Les ClassificationLevel des sujets seront spécifiés directement en MACL dans la Section 11.2.3.
    b. MEP-Nuclear: Top Secret (TS);
    c. EWpipeLines et RefineryB, HydroGen: Secret (S);
4. CW : Un Chinese Wall est à ériger entre les objets des deux projets EWpipelines et RefineryB.
5. RBAC : Une instance de RBAC est conçue de façon à compléter l'instance de Compartment, ainsi on n'a pas à spécifier des restrictions d'accès qui reflètent le principe du Need-to-know déjà pris en charge par l'instance de Compartment. Pour chaque rôle, on spécifie les permissions en prenant en considération les statuts des documents (MasterPlan, InProgress, Execution et AsBuilt). Les rôles et leurs permissions sont :
    a. **Asst** avec les permissions suivantes:

i. Read pour tous les objets;
ii. Write pour tous les objets dont le statut est InProgress.

b. **Eng** avec les permissions suivantes :
   i. Permissions du rôle Asst, car il l'a comme sous-rôle; il en résulte les privilèges de Read pour tous les objets et de Write pour tous les objets dont le statut est InProgress;
   ii. Write pour les objets dont le statut est Execution ou MasterPlan.
c. **Head** avec les permissions suivantes :
   i. Permissions du rôle Eng, car il l'a comme sous-rôle;
   ii. Read et Write pour tous les objets de tout statut.
d. Le rôle **TopManagement** de CEO, QAO et PM ont les permissions suivantes:
   i. Read pour tous les objets. À noter que comme le CEO est associé à tous les compartiments, il peut lire tous les objets même avec un *Deny* provenant des exigences de Biba, et ceci selon ComAlNode1 assurant la composition des instances des ACMM de Role, de Compartment et de Biba.
e. Role **External** : Consultant et Sub avec les permissions
   i. Read pour tous les objets sauf les InProgress. Il va de soi que les exigences des autres instances de ACMM, entre autres, celles du Compartment, vont limiter les privilèges d'accès des sujets dans le rôle External.

Ainsi, en respectant l'énumération des instances d'ACMM cités ci-dessus, on associe chacun des objets à un compartiment (projet, département, docType), on lui donne éventuellement un niveau d'habilitation de sécurité, et un niveau de sensibilité, et il peut appartenir à une ou plusieurs classes de groupe de conflit. On peut aussi considérer l'association de l'objet aux groupes d'objets : MasterPlan, InProgress, Execution et AsBuilt.

De même, pour la considération du contrôle d'accès pour un sujet, on l'associe à un compartiment (projet, département, docType), et on lui donne éventuellement un niveau d'habilitation de sécurité et un niveau de sensibilité.

La sous-section suivante fournit les spécifications de contrôle d'accès de HET dans le langage MACL et ce selon l'ordre suivant :

a. Définir les instances des éléments du KernelElements : les sujets, les objets, les groupes d'objets et les modes d'accès.
b. Créer les instances de ACMM énumérées ci-haut. On va créer les instances des éléments hors du Kernel ainsi que les associations des éléments entre eux et entre les instances des différents DecisionHandler.
c. Spécifier les ComAlNode et leurs nœuds fils pour avoir une spécification d'un ADA.

11.2.2 Instanciation des sujets, des objets et des modes d'accès

Les sujets et les objets considérés dans notre cas sont déjà énumérés et décrits dans la section précédente, leur création en MACL est comme suit:

```
CreateInstance(Subject,![ CEO-Hercule, PM-interDep1, /*fin
des sujets du TopManagement*/
MEPhead, MEPeng1, MEPeng2, MEPeng3, MEPasst1, MEPasst2,
MEPasst3, MEPpM1, MEPqAO1, /*fin MEP dep.*/

STRhead, STReng1, STReng2, STReng3, STRasst1, STRasst2,
STRasst3, STRpM1, STRqAO1, /*fin STR dep.*/

IThead, ITeng1, ITeng2, ITasst1, ITasst2, ITpM1,  ITqAO1,
/*fin IT dep.*/

GenConsultant1,
ITconsultant1, MEPconsultant2, MEPconsultant1,
STRconsultant1, GENsub1, ITSub1, MEPsub1, STRsub1]!);

/* « CreateInstance(Subject,![x, y, z]!); » remplace
« CreateInstance(Subject,x);
CreateInstance(Subject,y);
CreateInstance(Subject,z); » */

CreateInstance(Object, ![  MEPnucPlanRD1, MEPnucPlanREF1,
MEPnucPlanREF2, MEPnucPlanREF3, MEPnucPlanREF4,
MEPnucContREF1, MEPnucContREF3, // fin MEPnuc

MEPpplPlanRD1, MEPpplPlanREF1, MEPpplPlanREF2,
MEPpplPlanREF3, MEPpplPlanREF4, MEPpplContREF1,
MEPpplContREF3, // fin MEPppl

MEPrafPlanRD1, MEPrafPlanREF1, MEPrafPlanREF2,
MEPrafPlanREF3, MEPrafPlanREF4, MEPrafContREF1,
MEPrafContREF3, // fin MEPraf
```

```
STRinovPlanRD1, STRinovPlanREF1, STRinovPlanREF2,
STRInovPlanREF3, STRinovPlanREF4, STRinovContREF1,
STRinovContREF3, STRh2PlnaREF3, // fin STR

MEPh2PlanREF3, MEPh2PlanREF4, // fin MEPh2

ITallProjContREF3, ITnucContREF2, ITnucPlanREF3,
ITpplPlanREF3, ITrafContREF3, ITinovContRD2, ITinovPlanREF2
/*fin IT */]!);
```

Après la création des objets et des sujets, il faut créer les instances des ObjectsGroup et SubjectsGroup pour une organisation facilitant la spécification des instances des ACMM :

```
/* Création de groupes de sujets et d'objets selon les
départements. */

CreateInstance(SubjectsGroup, MEPsubjectsGroup);
// groupe des sujets du département MEP

Link(![ MEPhead, MEPeng1, MEPeng2, MEPeng3, MEPasst1,
 MEPasst2, MEPasst3, MEPpM1, MEPqAO1, MEPconsultant2,
 MEPconsultant1, MEPSub1 /*fin MEP*/,  beongsTo, belongsTo ,
 MEPsubjectsGroup);

CreateInstance(ObjectsGroup, MEPobjectsGroup);
// groupe des objets du département MEP

Link  (![MEPnucPlanRD1, MEPnucPlanREF1, MEPnucPlanREF2,
 MEPnucPlanREF3, MEPnucPlanREF4, MEPnucContREF1,
 MEPnucContREF3, MEPh2PlanREF3, MEPh2PlanREF4 ]!, belongsTo,
 MEPobjectsGroup) ;

// Idem pour les groupes du département STR

CreateInstance(SubjectsGroup, STRsubjectsGroup);

Link ( ![STRhead, STReng1, STReng2, STReng3, STRasst1,
 STRasst2, STRasst3, STRpM1, STRqAO1, SRTconsultant1,
 STRSub1]!, belongsTo, STRsubjectsGroup );

CreateInstance( ObjectsGroup, STRobjectsGroup);

Link ( ![STRinovPlanRD1, STRinovPlanREF1, STRinovPlanREF2,
```

```
 STRInovPlanREF3, STRinovPlanREF4, STRinovContREF1,
 STRinovContREF3, STRh2PlnaREF3]!, belongsTo, STRobjectsGroup);

// Idem pour les groupes du département IT

CreateInstance(SubjectsGroup, ITsubjectsGroup);

Link ( ![ IThead, ITeng1, ITeng2, ITasst1, ITasst2, ITpM1,
ITqAO1, ITConsultant1, ITSub1]!, belongsTo, ITSubjectsGroup);

CreateInstance(ObjectsGroup, ITobjectsGroup);

Link  (  ![ITallProjContREF3,  ITnucContREF2,  ITnucPlanREF3,
ITpplPlanREF3, ITrafContREF3, ITinovContRD2, ITinovPlanREF2
]!  , belongsTo, ITobjectsGroup);

// Sujets associés à tous les départements.

Link (![ CEO-Hercule, PM-interDep1, GenConsultant1, GenSub1 ]!,
belongsTo, ![ MEPobjectsGroup, STRobjectsGroup, ITobjectsGroup]!);

/* Création des groupes d'objets selon les projets. Les
sujets seront associés plus tard aux projets comme
compartiments dans la spécification d'instances de
Compartment. */

CreateInstance (ObjectsGroup, NucObjectsGroup;
// pour le projet Nuc
Link (![ ITallProjContREF3, ITnucContREF2, ITnucPlanREF3,
 MEPnucPlanRD1, MEPnucPlanREF1, MEPnucPlanREF2,
 MEPnucPlanREF3, MEPnucPlanREF4, MEPnucContREF1,
 MEPnucContREF3]!, belongsTo, NucObjectsGroup) ;

/*Les objets des projets Ppl et Raf appartiennent à deux
classes de conflit. Notons que ConflictClass est une
spécialisation de ObjectsGroup. On définit les groupes
d'objets de ces deux projets directement comme instances de
ConflictClass. */

// On commence par les objets du projet Ppl
CreateInstance(ConflictClass, PplObjectsGroup);

Link (![MEPpplPlanRD1, MEPpplPlanREF1, MEPpplPlanREF2,
 MEPpplPlanREF3, MEPpplPlanREF4, MEPpplContREF1,
 MEPpplContREF3, /* fin MEPppl */ ITpplPlanREF3,
 ITallProjContREF3 ]!, belongsTo, PplObjectsGroup) ;
```

```
// On enchaîne de même avec les objets du projet Raf
CreateInstance(ConflictClass, RafObjectsGroup);

Link ( ![  ITrafContREF3,ITallProjContREF3, MEPrafPlanRD1,
 MEPrafPlanREF1, MEPrafPlanREF2, MEPrafPlanREF3,
 MEPrafPlanREF4, MEPrafContREF1, MEPrafContREF3 // fin Raf
 ]!, belongsTo, RafObjectsGroup) ;

// Création du groupe des objets du projet Inov
CreateInstance(ObjectsGroup, InovObjectsGroup);

Link ( ![  ITinovContRD2, ITinovPlanREF2, ITallProjContREF3,
 STRinovPlanRD1,      STRinovPlanREF1,      STRinovPlanREF2,
 STRInovPlanREF3,      STRinovPlanREF4,      STRinovContREF1,
 STRinovContREF3 ]!, belongsTo, InovObjectsGroup);

// Création du groupe des objets du projet H2
CreateInstance(ObjectsGroup, H2ObjectsGroup);

Link ( ![ITallProjContREF3, STRh2PlnaREF3,
 MEPh2PlanREF3, MEPh2PlanREF4 ]!, belongsTo, H2ObjectsGroup);

/* Création des groupes d'objets selon les statuts:
MasterPlan, InProgress, Execution, AsBuilt*/

CreateInstance (ObjectsGroupe, ![ MasterPlanObjectsGroup,
InProgressObjectsGroup, ExecutionObjectsGroup,
AsBuiltObjectsGroup ]!);

Link(  ![ /*Suffixe1: avant projet*/
STRinovContREF1,STRinovPlanRD1, STRinovPlanREF1,
MEPpplContREF1, MEPrafContREF1, MEPnucPlanRD1,
MEPnucContREF1, MEPnucPlanREF1, MEPpplPlanRD1,
MEPpplPlanREF1, MEPrafPlanRD1, MEPrafPlanREF1 ]!,
belongsTo,   MasterPlanObjectsGroup );


Link(   ![  /*Suffixe2:  in  progress*/      ITnucContREF2,
ITinovContRD2,       ITinovPlanREF2,       STRinovPlanREF2,
MEPrafPlanREF2,  MEPpplPlanREF2,  MEPnucPlanREF2  ]!,  belongsTo,
InProgressObjectsGroup );
```

```
Link(  ![ /*Suffixe3 : execution */      ITrafContREF3,
ITpplPlanREF3, ITnucPlanREF3, ITallProjContREF3,
MEPh2PlanREF3, STRinovContREF3, STRh2PlnaREF3,
STRinovPlanREF3, MEPnucContREF3,MEPrafContREF3,
EPrafPlanREF3, MEPpplContREF3, MEPpplPlanREF3,
MEPnucPlanREF3]!,  belongsTo,  ExecutionObjectsGroup );


Link(  /*Suffixe4: AsBuilt*/ ![ MEPnucPlanREF4,
MEPrafPlanREF4, STRinovPlanREF4, MEPh2PlanREF4,
MEPpplPlanREF4 ]!, belongsTo, AsBuiltObjectsGroup );


// *** Création des modes d'accès

CreateInstance (AccessMode, AMread)
   {Name = Read;}

CreateInstance (AccessMode, AMwrite)
   {Name = Write;}
```

11.2.3 Spécification des applications des ACMM

Dans cette section, on va spécifier en MACL les applications des ACMM déjà décrits dans la Section 11.2.1. Chacune des sections suivantes donne les spécifications en MACL de ces applications des ACMM et ce dans le même ordre.

Instance du ACMM de Compartiments de Need-to-know

```
/*Création du DecisionHandler du ACMM : instance de
CompDecisionHandler. */

CreateInstance (CompDecisionHandler, CompDH) ;

// Création des instances de Compartment

CreateInstance (Compartment, ![ / * les départements*/ MEP,
STR, IT, /* les projets  */ Nuc, Ppl, Raf, Inov, H2,

  /* les types documents */ Plan, Cont]!);

/* on associe les compartiments à l'instance CompDH créée
plus haut */

Link(CompDH, uses,  ![ / * les départements*/ MEP, STR, IT,

/* les projets */ Nuc, Ppl, Raf, Inov, H2,
```

```
 /* les types documents */ Plan, Cont]!);

// Associations des objets et sujets aux compartiments
```

// Compartiments des départements

```
 /* Association des sujets et des objets du département MEP
au Compartment MEP. */

Link (MEP, ![ MEPsubjectGroup ]!);

Link (MEP, ![ MEPobjectsGroup ]!);

 /* Association des sujets et des objets du département MEP
au Compartment STR. */

Link ( STR, ![STRsubjectsGroup]!);

Link ( STR, ![STRobjectsGroup]!);

 /* Association des sujets et des objets du département MEP
au Compartment IT. */

Link (IT, ![ ITsubjectsGroup]!);

Link (IT, ![ ITobjectsGroup ]!);

 // Sujets associés à tous les départements

Link ( ![ MEP, STR, IT ]! , ![ CEO-Hercule, PM-interDep1, GenConsultant1,
GenSub1 ]!);
```

// Compartiments des projets

```
/* Quand des noms des sujets ont des suffixes numériques, ces
derniers indiquent les projets auxquels ils sont associés;
sujet à suffixe 1 => projets Nuc et H2, alors que les sujets
à suffixe 2 ou 3  => Ppl, Raf et Inov.  Sans numéro en suffixe
=> tous les projets.

Exemples de noms de sujets utilisés ci-dessous: ITqAO1 est un
Quality Assurance Officer du département IT affecté aux
projets Nuc et H2. STReng2 et STReng3 sont deux ingénieurs du
département de structure et chacun d'eux est affecté aux
projets Ppl, Raf et Inov */
```

Link (![Nuc, H2]!, ![CEO-Hercule, PM-interDep1, MEPhead, MEPeng1, MEPasst1, MEPpM1, MEPqAO1, STRhead, STReng1, STRasst1, STRpM1, STRqAO1, IThead, ITeng1, ITasst1, ITpM1,ITqAO1, GenConsultant1, ITconsultant1, MEPconsultant1, STRconsultant1, GENsub1, ITSub1, MEPsub1, STRsub1]!);

Link (![Raf, Ppl, Inov]!, ![CEO-Hercule, PM-interDep1, MEPeng2, MEPeng3, MEPasst1, MEPasst2, MEPasst3, STReng2, STReng3, STRasst2, STRasst3, ITeng2, ITasst2, MEPconsultant2]!);

/* Association des objets aux compartiments des projets. On réutilise les groupes d'objets déjà spécifiés dans la partie du KernelElements. */

Link (Nuc, ![NucObjectsGroup]!);

Link (Ppl, ![PplObjectsGroup]!);

Link (Raf, ![RafObjectsGroup]!);

Link (Inov, ![InovObjectsGroup]!);

Link (H2, ![H2ObjectsGroup]!);

// Compartiments de type de document

/*Association des sujets aux compartiments de type de documents : Cont et Plan. */

Link (![Cont, Plan]! , ![CEO-Hercule, PM-interDep1, MEPhead, MEPpM1, MEPqAO1, STRhead, STRpM1, STRqAO1, IThead, ITpM1, ITqAO1, GenConsultant1]!);

Link (Plan , ![MEPeng1, MEPeng2, MEPeng3, MEPasst1, MEPasst2, MEPasst3, STReng1, STReng2, STReng3, STRasst1, STRasst2, STRasst3, ITeng1, ITeng2, ITasst1, ITasst2, ITconsultant1, MEPconsultant2, MEPconsultant1, STRconsultant1, GENsub1, ITSub1, MEPsub1, STRsub1]!);

/*Association des objets aux compartiments de type de documents : Cont et Plan. */

Link (Plan , ![MEPnucPlanRD1, MEPnucPlanREF1, MEPnucPlanREF2,

```
 MEPnucPlanREF3, MEPnucPlanREF4, // fin MEPnuc

 MEPpplPlanRD1, MEPpplPlanREF1, MEPpplPlanREF2,
 MEPpplPlanREF3, MEPpplPlanREF4, // fin MEPppl

 MEPrafPlanRD1, MEPrafPlanREF1, MEPrafPlanREF2,
 MEPrafPlanREF3, MEPrafPlanREF4, // fin MEPraf

 STRinovPlanRD1, STRinovPlanREF1, STRinovPlanREF2,
 STRinovPlanREF3, STRinovPlanREF4, // fin STR

 MEPh2PlanREF3, MEPh2PlanREF4,

 ITpplPlanREF3, ITinovPlanREF2 /*fin IT */ ]! );
```

```
Link ( Cont, ![  MEPnucContREF1, MEPnucContREF3, // fin MEPnuc

 MEPpplContREF1, MEPpplContREF3, // fin MEPppl

 MEPrafContREF1, MEPrafContREF3, // fin MEPraf

 STRinovContREF1, STRinovContREF3, // fin STR

 ITallProjContREF3, ITnucContREF2, ITrafContREF3,
 ITinovContRD2 /*fin IT */]!);
```

Instance de Biba

```
CreateInstance( BIBAdecsionHandler, BIBAdH);

// Définir les niveaux d'intégrité

CreateInstance(BIBAclassificationLevel,
![ REFdata, RDdata]!);

/* Associer les niveaux d'intégrité à l'instance BIBAdH de
BIBAdecisionHandler */

Link (BIBAdH, uses, ![ REFdata, RDdata]!);

// Définir l'ordre des niveaux d'integrité

DLink(RDdata, lessThan, REFdata) ;
```

```
/* Spécifier les niveaux d'intégrité des objets en associant
ces objets à RDdata ou à REFdata */

Link(RDdata, ![ MEPnucPlanRD1, MEPpplPlanRD1,
MEPrafPlanRD1, STRinovPlanRD1, ITinovContRD2  ]!) ;

Link (REFdata,![  MEPnucPlanREF1, MEPnucPlanREF2,
 MEPnucPlanREF3, MEPnucPlanREF4, MEPnucContREF1,
 MEPnucContREF3, MEPpplPlanREF1, MEPpplPlanREF2,
 MEPpplPlanREF3, MEPpplPlanREF4, MEPpplContREF1,
 MEPpplContREF3, /* fin MEPppl */
 MEPrafPlanREF1, MEPrafPlanREF2, MEPrafPlanREF3,
 MEPrafPlanREF4, MEPrafContREF1, MEPrafContREF3, /* fin
 MEPraf */
 STRinovPlanREF1, STRinovPlanREF2, STRinovPlanREF3,
 STRinovPlanREF4, STRinovContREF1, STRinovContREF3,
 STRh2PlnaREF3, /* fin STR */
 MEPh2PlanREF3, MEPh2PlanREF4,  /* fin MEPh2 */
 ITallProjContREF3, ITnucContREF2, ITnucPlanREF3,
 ITpplPlanREF3, ITrafContREF3, ITinovPlanREF2 /*fin IT */
 ]!) ;

/* Spécification des niveaux d'intégrité des sujets en
associant les sujets à l'un ou l'autre des deux niveaux
REFdata et  RDdata */

Link ( REFdata , ![ CEO-Hercule, PM-interDep1, MEPhead, MEPeng1,
 MEPeng2, MEPasst1, MEPasst2, MEPpM1, MEPqAO1,
 STRhead, STReng1, STReng2, STRasst1, STRasst2, STRasst3,
 STRpM1, STRqAO1, IThead, ITeng1, ITasst1, ITasst2, ITpM1,
 ITqAO1, GenConsultant1, ITconsultant1, MEPconsultant2,
 MEPconsultant1, STRconsultant1, GENsub1, ITSub1, MEPsub1,
 STRsub1]!);

Link(RDdata, ![  MEPeng3, MEPasst3,STReng3, ITeng2]!);
```

Instance Bell LaPadula

```
// Création d'une instance de BLPdecisionHandler
CreateInstance( BLPdecisionHandler, BLPdH);

/* Création des instances de ClassificationLevel et leur
association à l'instance BLPdH de BLPdecisionHandler */
CreateInstance( ClassificationLevel, ![TS-CL , S-CL]!);
Link(BLPdH, uses, ![S-CL, TS-CL]!);
```

```
/* spécification de la relation d'ordre entre les instances
TS-CL et S-CL de ClassificationLevel */
Dlink (S-CL, lessThan, TS-CL);

/* Association des objets à leur ClassificationLevel (selon
les projets) */

Link (TS-CL,![ NucObjectsGroup ]!);
Link (S-CL,![ PplObjectsGroup, RafObjectsGroup,
InovObjectsGroup, H2ObjectsGroup ]!);

/* Association des sujets aux deux ClassificationLevel. TS-
CL et S-CL */
Link (TS-CL, ![ CEO-Hercule, PM-interDep1, MEPhead, MEPeng1,
 MEPasst1, MEPpM1, MEPqAO1, /*Fin dep. MEP  */ STRhead,
 STReng1, STRasst1, STRpM1, STRqAO1, /*Fin dep. STR  */
 IThead, ITeng1, ITasst1, ITpM1,ITqAO1, /*Fin dep.  IT */
 GenConsultant1, ITconsultant1, ITSub1, MEPsub1, STRsub1]!);

Link ( S-CL, ![ MEPeng2, MEPeng3, MEPasst2,
 MEPasst3, STReng2, STReng3, STRasst2, STRasst3 , ITeng2,
 ITasst2, MEPconsultant2,
 MEPconsultant1, STRconsultant1, GENsub1
 ]!);

/* Les trois derniers sujets MEPconsultant1,
STRconsultant1, GENsub1 sont en principe attribués des
tâches reliées au projet MEP-Nuclear, mais cela doit être
reconsidéré par les dirigeants de HET, car leur niveau S-CL
ne les qualifie pas pour accéder aux données de MEP-
Nuclear. */
```

Instance Chinese Wall

```
// Création d'une instance de CWdecisionHandler
CreateInstance( CWdecisionHandler, CWdH) ;

/* création du groupe de conflit des projets pétroliers et
son association à l'instance de CWdecisionHandler */

CreateInstance(ConflictGroup, Petrol);
Link(CWdH, uses, Petrol);

/* Création et association des groupes d'objets
représentant des classes de conflit au groupe de conflit
Petrol.*/
```

```
CreateInstance (ConflictClass, RafObjectsGroup);
CreateInstance (ConflictClass, PplObjectsGroup);

Link (RafObjectsGroup, Petrol);
Link (PplObjectsGroup, Petrol);

/* Association des instances de ConflictClass à l'instance
de CWdecisionHandler */
Link (CWdH, uses, RafObjectsGroup);
Link (CWdH, uses, PplObjectsGroup);
```

Instance de RBAC

```
/* Le code spécifiant les rôles suit le métamodèle de RBAC.
Ceci implique qu'on doit créer une instance du
DecisionHandler spécialisé de RBAC, ensuite on crée : les
rôles, les permissions avec leurs modes d'accès.
Il faut associer les permissions aux modes d'accès et
objets concernés.
Il faut associer chaque rôle aux sujets qui sont dans ce
rôle.
Il faut associer les instances des éléments créés à
l'instance de RBACdecisionHandler. */

// Création du RBACdecisionHandler
CreateInstance(RBACdecisionHandler, RBACdH);

// Création des rôles
CreateInstance( Role, ![ Asst, Eng, Head, External,
   TopManagement]!);

// Association des rôles créés au DecisionHandler
Link (RBACdH, uses, ![Asst, Eng, Head, External,
  TopManagement]!);

/**  Permission de lire à associer au Rôle Asst */

CreateInstance( Permission, AsstPermR)
 {Decision  = Permit;}

/* Rappelons que AMread et AMwrite sont déjà créés comme
instances d'éléments du KernelElements */
// Asst permet de lire tout objet
Link(AsstPermR, ![AMread, AllObjectsGroup ]!);

/* Permission d'écriture dans les objets InProgress */
/**  Permission d'écrire à associer au Rôle Asst */
```

```
CreateInstance( Permission, AsstPermW)
  {Decision = Permit;}

Link(AsstPermW, ![AMwrite, InProgressObjectsGroup, Asst
]!);

/* notons que la dernière ligne de code associe, entre
autres, la permission AssistnatPermW au rôle Asst. */

/* On rappelle que l'usage de l'Alias avec « ! » dans la
ligne ci-dessus fait qu'elle est équivalente à :
Link (AsstPermW, AMwrite); Link (AsstPermW,
InProgressObjectsGroup); Link (AsstPermW, Asst); */

/* On rappelle aussi que selon notre spécification du
prédicat ReturnedDecision, on peut spécifier une permission
de rôle pour un groupe d'objets directement, sans besoin de
créer une nouvelle instance de Permission pour chaque objet
accessible par un rôle.*/

/* Il faut associer les éléments de l'instance du
métamodèle de RBAC à l'instance RBACdH de
RBACdecisionHandler */
Link (RBACdH, uses, ![ Assitant, AsstPermR, AMread,
AsstPermW, AMwrite   ]!) ;

/** Rôle Eng : Selon les spécifications informelles, Eng a
Asst comme sous-rôle et un Eng peut écrire sur les objets
dont le statut est Execution.  */

DLink (Asst, subRole, Eng);

// Création de l'instance EngPermW de Permission
CreateInstance( Permission, EngPermW)
  {   decision = Permit; }
/* Association de EngPermW au bon mode d'accès et objets
concernés et au rôle Eng. */
Link( EngPermW, ![AMwrite, ExecutionObjectsGroup,
MasterPlanObjectsGroup, Eng ]!);

/* on n'oublie pas d'associer les éléments de l'instance du
métamodèle de RBAC à l'instance de RBACdecisionHandler */
Link (RBACdH, uses,![ EngRole, EngPermW ]!) ;


/***** Role Head. Il a Eng comme sous-rôle, et il peut
écrire dans les objets de tout statut. */
```

```
DLink( Eng , subrole, Head);

CreateInstance( Permission, HeadPermW)
  {decision = Permit;}

Link( HeadPermW, ![AMwrite, MasterPlanObjectsGroup,
   AsBuiltObjectsGroup, Head ]!);

// Associer à l'instance RBACdH de RBACdecisionHandler

Link(RBACdH, uses, ![ AssitantRole, AsstPermR,
   AsstPermW]! ,) ;

/**** Rôle External des consultants et Subcontractors: il
peut lire tout sans rien écrire directement.  **/

CreateInstance(Permission, ExternalPermR)
   {decision = Permit;}

Link (ExternalPermR, ![AMread, MasterPlanObjectsGroup,
 ExecutionObjectsGroup,  AsBuiltObjectsGroup,   External
 ]!); /* pas d'accès à InProgressObjectsGroup */

// Associer à l'instance RBACdH de RBACdecisionHandler
Link (RBACdH, uses, ![ External, ExternalPermR  ]!) ;



/******* Rôle TopManagement des CEO, PM, PM-interDep, QAO
lisant tous et ne pouvant rien écrire directement */


DLink (External, subRole, TopManagement);
CreateInstance (Permission, TopManagementPermR)
   {decision = Permit;}
// Ajouter la permission de lire les objets Inprogress.
Link (TopManagementPermR, ![TopManagement, AMread,
InprogressObjectsGroup ]!);

// Associer à l'instance RBACdH de RBACdecisionHandler
Link (RBACdH, ![TopManagementPermR, TopManagement]!) ;


// ***** Association des sujets à leurs rôles
```

```
Link( Asst, ![     MEPasst1, MEPasst2,
 MEPasst3,      STRasst1, STRasst2, STRasst3, ITasst1,
 ITasst2 ]!);

Link( Eng, ![  ITeng1, ITeng2,MEPeng1, MEPeng2, MEPeng3,
     STReng1, STReng2, STReng3]!);

Link( Head, ![  IThead, STRhead, MEPhead,]!);

Link( External, ![   GenConsultant1, ITconsultant1,
 MEPconsultant2, MEPconsultant1, STRconsultant1, GENsub1,
 ITSub1, MEPsub1, STRsub1 ]!);

Link( TopManagement, ![ ITpM1,  ITqAO1, STRpM1, STRqAO1,
   CEO-Hercule, PM-interDep1, MEPpM1, MEPqAO1,]!);


/* Remarquons que le sous-traitant STRSub1 peut lire
presque tout selon son rôle de External, mais son
compartiment l'empêche d'accéder à des données hors du
compartiment des projets Nuc et H2. */
```

Spécification de la structure de l'ADA

D'après les spécifications informelles mentionnées au début de la Section 11.2, on doit créer deux instances de ComAlNode, qu'on appelle ComAlNode1 et ComAlNode2. L'ADA est illustré dans la figure ci-dessous :

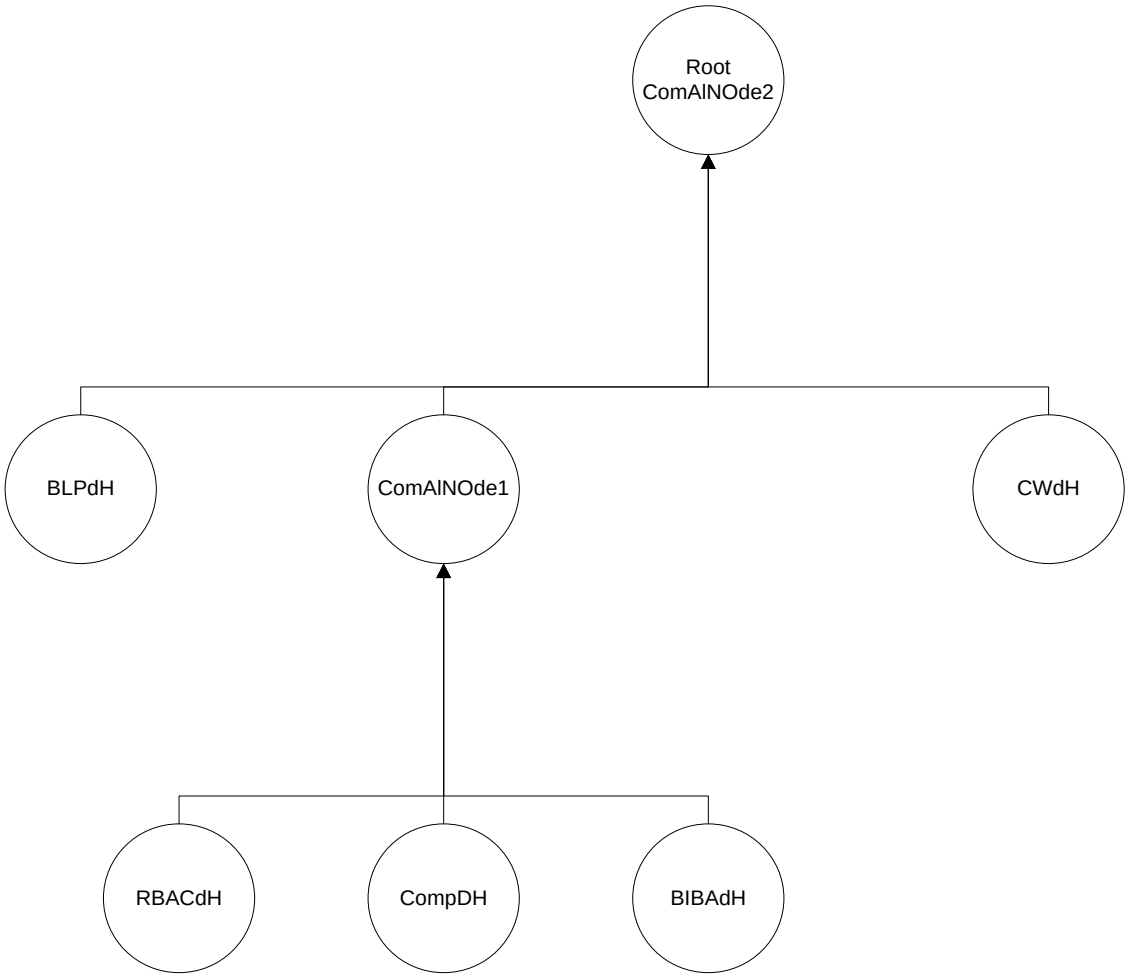

Figure 11.2-1 L'ADA des instances des ACMM spécifiant les exigences de contrôle d'accès de HET.

MACL doit permettre de spécifier les deux ComAlNode et les associer aux nœuds fils pour spécifier l'ADA. Ainsi, on aura l'énoncé suivant :

```
// Création du ComAlNode1
CreateInstance(ComAlNode, ComAlNode1) {
  inputDecsionList= [Permit, Deny, NoteApplicable];
  outputDecsionList = [Permit, Deny, NoteApplicable];
  children = [RBACdH, CompDH, BIBAdH];
  ClassOds =
```

```
/* Deux Permit de rôle et de compartiment suffisent pour
résulter en Permit */

   ClassOd( Permit) ←

   CountIndexAndValueMatches (Feed, [Permit, Permit] , 2). ,


/* Deux Deny sont suffisants pour résulter en Deny */

 ClassOd (Deny) ← CountIndexAndValueMatches (Feed,[Deny,
 Deny, Deny], m), m ≥ 2.,

/* Un Deny de rôle ou de compartiment suffit pour résulter en
Deny. */

 ClassOd (Deny) ← CountIndexAndValueMatches (Feed,[Deny,
 Deny], n) , n ≥ 1. ;

}

// Création du ComAlNode2

CreateInstance(ComAlNode, ComAlNode2) {

  inputDecsionList= [Permit, Deny, NoteApplicable];

  outputDecsionList = [Permit, Deny, NoteApplicable];

  children = [BLPfH, ComAlNode1, CWdh];

  ClassOds =

   /* Une seule occurrence de Deny suffit pour prévaloir sur
   toutes les autres décisions. */

   ClassOd(Deny) ← CountIMatches (Feed,[ Deny] , m) AND

   m ≥ 1. ,

   /* (Les décisions d'entrée comprennent au moins un Permit
   sans aucune occurrence de Deny) => Sortie de Permit. */

  ClassOd(Permit) ← CountIMatches (Feed,[ Permit] , m) AND
  m ≥ 1 AND CountIMatches (Feed,[ Deny] , 0). ,
```

```
 /* (Si toutes les décisions d'entrée sont des NotApplicable)
 => Sortie NotApplicable. */

 ClassOd (NotApplicable) ←

   CountMatches (Feed,[NotApplicable], 3). ;

}
```

11.3 Conclusion

Le cas présenté nous a permis de voir comment on peut spécifier séparément les différentes applications des modèles de contrôle d'accès. On a vu que pour spécifier une application du principe de CW, nous nous sommes contenté d'instancier le ACMM de CW sans besoin de se soucier de la spécification de la logique de décision du CW. Ainsi, nous ne sommes pas partis de zéro, au contraire nous avons réutilisé le ACMM de CW en l'instanciant et en fournissant les données propres à notre cas. Ces données fournies lors de la création d'une instance de groupe de conflit et des classes de conflit qui la composent, et lors de la spécification de l'appartenance des objets aux différentes classes de conflit. Ceci est accompli par des instanciations des éléments ConflictGroup, ConflictClass et de l'association « belongsTo », qui sont des éléments réutilisables du ACMM de CW. Ceci illustre bien la réutilisation et la personnalisation qui représente un objectif principal de notre travail annoncé à la Section 4.2. La même réutilisation avec personnalisation est claire avec l'instanciation des métamodèles de compartiment, Biba, Bell Lapadula et RBAC dans notre étude cas. L'expressivité de notre langage est adéquate pour l'expression des ECAE qui comprennent généralement des exigences d'applications de modèles de contrôle d'accès, car ces dernières sont spécifiables en MACL par des instances des ACMM.

De plus, on a vu comment la spécification de plusieurs ComAlNode à plusieurs niveaux permet une intégration graduelle des différentes applications des modèles de contrôle d'accès représentées par des instances de plusieurs ACMM.

Cette modularité réduit énormément la complexité des spécifications de contrôle d'accès et leur exposition aux erreurs. Dans notre étude de cas, la réduction de la complexité n'a pas été un résultat d'une simple réduction du nombre de lignes de code. Elle a résulté de la modularité de la spécification qui permet de la diviser en modules plus abordables et plus faciles à contrôler par l'humain. Il suffit de voir que chaque exigence de contrôle d'accès de notre étude de cas est associable à un seul nœud de l'ADA. Ainsi, on peut voir à quel point la compréhension et la modification de la spécification sont rendues plus aisées pour bien apprécier l'apport de MACL en réduction de complexité.

Chapitre 12 Conclusion

Dans ce chapitre nous allons examiner avec un certain recul notre travail comme un tout selon une perspective d'évaluation des accomplissements en fonction des objectifs projetés à la Section 4.9. Cela permettra de souligner les succès ainsi que les tentatives qui laissent à désirer. On passera ensuite à une exploration des travaux futurs de recherche qui peuvent étendre notre travail, ou le reproduire avec une adaptation à un autre domaine d'application.

12.1 Objectifs atteints

Dans les chapitres décrivant notre solution, nous avons présenté le langage MACL et sa composante ComLang, nous avons fait aussi la correspondance entre les aspects visés de notre langage et les éléments du langage qui les réalisent. En ajoutant une étude de cas et plusieurs exemples d'illustration, nous sommes en mesure de récapituler en mettant en relief les atouts de MACL permettant l'atteinte des objectifs annoncés au Chapitre 4 et ce comme suit :

1. Grâce à notre approche visant l'expressivité présentée à la Section5.4, MACL permet un pouvoir expressif adéquat basé sur une réduction de la complexité des spécifications de contrôle d'accès avec des instances de ACMM et les moyens de les intégrer ensemble. Ainsi, on peut exprimer aisément des exigences de contrôle

d'accès visant à satisfaire plusieurs principes de contrôle d'accès tout en spécifiant des règles claires de prévalence ou de règlement de conflit.

2. La réutilisation est un résultat direct de la spécification des ECAE par instanciation des métamodèles de contrôle d'accès; les éléments des métamodèles sont réutilisés chaque fois qu'on instancie un ACMM, et la logique de décision encapsulée dans un ACMM est réutilisée chaque fois qu'on instancie un ACMM.
3. La validation : Comme expliquée dans le Chapitre 10, la validation de la cohérence et de la conformité avec des éléments de métamodélisation est rendue possible grâce à un mapping couvrant les spécifications MACL et les éléments de modélisation.
4. La vérification de complétude et l'accessibilité présentées dans la Section 10.1.2 sont faisables au niveau des instances de ACMM et au niveau des ComAlNode comme expliqué dans le Chapitre 10. Notons ici que le Chapitre 8 présente une vérification d'une propriété dans la Section 8.3.1 et une vérification d'accessibilité d'un objet dans la Section 8.3.2.
5. Le mécanisme d'interprétation de MACL présenté à la Section 4.5 est bien atteint car tout énoncé de MACL est un ADA représentant un système de décision mappable vers le langage cible. Ce mapping permet d'émettre une décision de contrôle d'accès en réponse à toute demande d'accès. Un exemple de détermination automatisée d'une décision de contrôle d'accès est présenté dans la Section 8.3.
6. Combinaison de politiques traitées au Chapitre 9 : Le langage ComLang faisant partie de MACL permet l'atteinte de cet objectif avec toute la rigueur désirée. Cet objectif n'est pas implémenté au Chapitre 8, mais les exemples d'implémentation de ce chapitre sont des indicateurs de la faisabilité des autres implémentations.
7. La vérification de conformité à un règlement ou à un standard est un objectif annoncé dès le début comme objectif d'une éventuelle extension future de notre travail. Ce travail futur s'avère maintenant plus réaliste à la lumière des points précédents.

12.2 Contribution

La contribution de MACL sera mieux mise en évidence quand on examine, pour chacun des objectifs atteints énumérés à la section précédente, ce que les autres langages mentionnés au Chapitre 2 offrent. Ceci prend en considération, quand c'est pertinent, l'exemple de l'étude de cas du Chapitre 11 et l'implémentation du Chapitre 8. On détaille ceci en adoptant la même numérotation des objectifs de la section précédente, et ce comme suit :

1. L'expressivité au niveau des modèles: Dans notre étude de cas, la spécification de l'application des principes de contrôle d'accès de compartiment, Biba, Bell LaPadula, de CW et de RBAC a été directement accomplie par simple instanciation des ACMM correspondant. En revanche, Ponder, un profil de XACML et RBAC spatio-temporelle sont les seuls qui permettent seulement une expression directe du principe de contrôle d'accès de RBAC. Concernant les autres ACMM, aucun des langages ne possède des moyens de spécifications directes d'application des principes de contrôle d'accès sous-jacents à ces ACMM.
2. La réutilisation : La logique de décision et les éléments indispensables de spécification des applications des cinq principes de contrôle d'accès sous-jacents aux cinq ACMM de l'étude de cas, ont été une simple réutilisation des ces cinq ACMM en les instanciant. Comme détaillé dans la Section 11.3, l'instanciation d'un ACMM réutilise le métamodèle et permet en même temps de spécifier les caractéristiques particulières à l'instance qu'on crée. Ceci représente une « personnalisation » qui permet une opportunité de réutilisation pour chaque cas selon ses données particulières. De plus, nous ne sommes pas partis de zéro en spécifiant la logique de décision et les concepts inhérents aux principes de contrôle d'accès sous-jacents aux cinq ACMM. Ceci n'est offert dans aucun des langages existants. Même Ponder qui permet de réutiliser des construits réservés à la spécification d'application du principe de RBAC n'offre pas de logique de décision prête à appliquer pour cette spécification. Ponder permet néanmoins la

création de types et l'instanciation ce qui lui confère un avantage de capacité de réutilisation par rapport aux autres langages.

3. 4, et 5. Les objectifs de validation, de vérification de propriété et d'interprétation automatique ne sont atteints que partiellement par certains des langages considérés et détaillés dans le tableau ci-dessous. On rappelle que MACL a atteint ces derniers objectifs comme mentionné dans les points 3, 4 et 5 de la Section précédente. De plus, concernant l'étude de cas, vu que la spécification MACL permet un mapping vers CLP, cela va permettre la validation et la vérification de propriétés spécifiées en termes de prédicats comme détaillé dans les Sections 10.1 et 10.2.
6. La spécification de ComAl : MACL permet la spécification de ComAl, comme illustré dans notre étude de cas où l'on avait à concilier cinq systèmes de décisions qui sont les cinq instances de différents ACMM. Comme illustré dans le tableau ci-dessous, la prise en charge de spécifications de ComAL est très limitée par les autres langages.
7. La vérification de conformité à un règlement ou à un standard n'est pas atteint ni par notre langage ni par les autres langages.

Pour offrir une vue d'ensemble plus facile à consulter, le tableau suivant récapitule cette section.

Objectifs atteints par MACL	XACML	SAML	Ponder	EPAL	PDL	RW	Spatio-Temporal RBAC
Expressivité au niveau de modèles	Seulement pour le modèle de rôles	Non	Seulement pour les modèles de rôles, délégation.	Permet des spécifications de groupes d'objets et de sujets; en plus des buts de l'accès	Non	Oui pour les modèles de rôles, CW, la séparation des tâches, l'historique des accès.	Seulement pour le modèle de rôles avec prise en charge de contraintes de

							temps et de lieu.
Réutilisation	Non	Non	Oui	Non	Non	Non	Non
Validation	Non	Non	Non	Non	Non	Oui	Non
Vérification de complétude et d'accessibilité	Non	Non	Non	Non	Non	Oui	Non
Mécanisme d'interprétation	Oui	Non	Non	Non	Oui	Oui	Oui
Combinaison de politiques	Seulement avec un nombre limité de ComAl prédéfinis	Non	Non	Oui	Oui	Non	Non

12.3 Limitations

Ayant détaillé les contributions et les points forts de notre travail, il importe aussi d'en reconnaitre les principales limitations :

1. Les ACMM sont comme les principes de contrôle d'accès sujets à une évolution et à une émergence continue de nouveautés. En conséquence, on doit mettre à jour les spécifications des logiques de décision au niveau des ACMM existant en plus d'élaborer de nouveaux ACMM quand on en aura besoin.
2. Notre travail ne permet pas la spécification de la logique de décision d'une façon visuelle; cette logique doit être codée textuellement comme mentionné dans la Section 8.2.1.
3. L'aspect temporel n'est pas pris en charge d'une façon explicite, dans le sens que notre langage n'est pas conçu pour répondre à une requête qui prend en considération les différents états possibles des ECAE en fonction des possibilités d'occurrences d'évènements après l'écoulement d'un certain temps.

4. L'indécidabilité, mentionnée dans la Section 5.4.1, caractérise FOL et notre langage cible et fait que la réussite de toute tâche de vérification, de validation ou de détermination de décision automatisée n'est pas toujours garantie.
5. L'efficience des tâches automatisées par le moyen de toute implémentation de notre travail sera toujours dépendante de l'état de l'art des algorithmes sous-jacents aux outils de satisfiabilité utilisés.

Il importe aussi de mentionner que les limitations de notre implémentation ont été identifiées dans la Section 8.2.1.

12.4 Applications potentielles

Pour concevoir notre langage MACL, nous avons développé des moyens préliminaires qui pourraient être d'une grande utilité pour d'autres chercheurs œuvrant dans différents domaines. Ainsi, notre travail a abouti à des accomplissements qui auront des applications au-delà du domaine du contrôle d'accès et qu'on cite dans cette section.

12.4.1 Pont Modèles-CLP

En construisant MACL en éléments faciles à comprendre et à modifier, nous avons eu recours à la métamodélisation. Ensuite, nous avons défini un mapping entre les éléments de modélisation et des énoncés du langage cible. Ceci fournit un pont permettant de passer d'un sous-ensemble d'éléments de UML vers leurs spécifications écrites dans CLP tout en étant conscient de la limitation de notre mapping aux aspects qu'on utilise des éléments du sous-ensemble de UML, comme clarifié au début de la Section 7.2.

Le mapping étant assez simple et efficace, il se prête très bien à une réutilisation et une amélioration continue. Le pont ainsi défini peut intéresser les parties œuvrant dans le domaine de la conception de langages en général et en particulier dans celui de la métamodélisation avec une spécification formelle des éléments des langages.

12.4.2 Spécification formelle de composition de décisions

ComLang a le potentiel d'une contribution importante dans le domaine des systèmes de décision qui est un domaine d'importance avec maintes applications en détection d'intrusion en sécurité informatique [5] ; en reconnaissance de caractères manuscrits [6] et [38]; en intelligence artificielle basée sur plusieurs systèmes experts et dans le

domaine juridique basé sur des spécification de prise de décision. Pour de telles applications, une composition de plusieurs décisions est indispensable, d'où l'importance de ComLang qui permet la spécification formelle des différentes stratégies de composition (conciliation) de décisions. En plus de permettre des spécifications non-ambiguës, ComLang offre à ces domaines d'application des moyens de vérification et de validation. Ces moyens sont indispensables quand il s'agit de plusieurs stratégies de composition de décisions et d'un grand nombre de sources de décision.

12.4.3 Un arbre de décisions ascendantes

La structuration de plusieurs systèmes de décision dans un arbre de décisions ascendantes représente un modèle visuel de structuration d'une collaboration de plusieurs systèmes de décision. Un ADA est plus compliqué à spécifier qu'un simple arbre de décision, mais il permet une meilleure expressivité qui utilise la structure de l'arbre et y ajoute des spécifications de décision dans chaque nœud. Dans un ADA chaque nœud comprend sa propre logique de décision qui considère les données d'une requête et les décisions de ses nœuds fils s'il en a. Selon sa logique de décision chaque nœud produit une décision et contribue ainsi à fournir les données nécessaires à son nœud père pour produire sa décision. Par contre, un arbre de décision permet de partir avec une seule entrée au sommet spécifiant les valeurs de vérité d'un ensemble de conditions, cette entrée détermine le routage à travers l'arbre pour arriver à une de ses feuilles. Chacune des feuilles de l'arbre de décision est marquée par une certaine décision.

12.5 Travaux futurs

Le présent travail fait partie d'un projet plus large. Le travail futur complétant ce projet consiste en le développement d'un outil offrant : (1) un environnement d'édition de MACL avec ses deux formes l'une textuelle l'autre graphique (visuelle) synchronisées; (2) une automatisation du mapping des énoncés MACL vers le langage cible; (3) des moyens d'assistance de spécification des Clause Attribute qui sont définies en termes d'individus et de prédicats logiques représentant les éléments spécifiés en MACL; (4) des moyens de spécification de propriétés et de leur vérification automatique, incluant la validation de

la cohérence et la vérification de la complétude. Cet outil exige un travail d'implémentation important, et pour cette raison il n'a pas pu être complété dans le cadre de ce travail de thèse.

Un autre travail de recherche futur serait l'élaboration d'un mapping permettant la réécriture des énoncés MACL dans d'autres langages comme XACML. Il en résulterait une meilleure et plus large exploitation de notre métamodèle sans qu'on ait à adapter les infrastructures implémentant le contrôle d'accès, comme les systèmes d'exploitation ou les systèmes de gestion des bases de données.

D'autre part, comme mentionné dans la Section 12.4.1, notre travail représente un pas vers l'avant dans le domaine des langages de domaines spécifiques basés sur la métamodélisation et dotés d'un mapping vers un langage cible basé sur FOL. Ainsi, ce qu'on a désigné par Pont Modèles-CLP dans la Section 12.4.1 représenterait une bonne piste de recherche pour d'autres chercheurs visant des spécifications dotées de moyens de V & V pour divers langages de modélisation et pour divers domaines d'application.

12.6 Conclusion

Passer des spécifications informelles des ECAE vers des spécifications formelles, représente depuis longtemps un défi d'envergure du génie logiciel. Ce défi est plus contraignant dans le domaine du contrôle d'accès à cause des aspects critiques de la sécurité des informations où la tolérance aux erreurs est nulle.

Le langage MACL proposé permet la spécification formelle des exigences de contrôle d'accès tout en restant au niveau des modèles de contrôle d'accès, ce qui entraine une réduction significative de la complexité et conséquemment des erreurs. De plus, un survol des travaux connexe au langage ComLang de la Section 9.8 fait constater que ComLang constitue une dimension à part de la contribution de notre travail car ce langage comble un vide dans le domaine de la spécification d'algorithmes de combinaison de politiques et de composition de systèmes de décision. La structure d'arbre de décisions ascendantes, propose un nouveau modèle pour tout domaine d'application où la

spécification des exigences dépend d'un ensemble de décisions qu'on peut diviser et grouper en sous-arbres formant un seul ADA.

Nous croyons que notre travail constitue une contribution concrète, exploitable sur le terrain dans le domaine du contrôle d'accès et dans tout autre domaine axé sur des spécifications de prise de décision. On s'attend à ce que notre travail ait des répercussions sur d'autres travaux de recherche s'inscrivant dans le domaine de la modélisation et des spécifications formelles.

Références

1. *ANSI INCITS 359-2004 for Role Based Access Control*. 2004, American National Standards Institute.
2. Sandhu, R., V. Bhamidipati, and Q. Munawer, *The ARBAC97 model for role-based administration of roles.* ACM Trans. Information and system security, 1999. **2**(1): p. 105-135.
3. Bell, D. and L. LaPadula, *Secure Computer Systems: Unified Exposition and Multics Interpretation*. March 1976, Mitre Corporation: Bedford, MA.
4. Moses, T. and S. Godik, *OASIS eXtensible Access Control Markup Language (XACML), OASIS Standard, Version 2.0*. 2005.
5. Bauer, L., J. Ligatti, and D. Walker, *Composing security policies with polymer*, in *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*. 2005, ACM: Chicago, IL, USA. p. 305-314.
6. Kam-Ho, T., J.J. Hull, and S.N. Srihari, *Decision Combination in Multiple Classifier Systems.* IEEE Trans. Pattern Anal. Mach. Intell., 1994. **16**(1): p. 66-75.
7. Li, N., et al., *A Formal Language for Specifying Policy Combining Algorithms in Access Control*, in *Computer science*. 2008, Purdue University.
8. Cohen, J., *Constraint logic programming languages.* Commun. ACM, 1990. **33**(7): p. 52-68.
9. Sandhu, R., et al., *Role-based access control models*, in *IEEE Computer*. Feb 1996. p. 29 (2): 38-47.
10. Microsystems, S. *Sun's XACML implementation*. sourceforge.net, 2003.
11. Cantor, S. *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML} V2.0*. Available from: http://www.oasis-open.org/committees/documents.php?wg_abbrev=security.
12. Damianou, N., et al., *The Ponder specification language*, in *Workshop on Policies for Distributed Systems and Networks*. Jan 2001, Springer-Verlag. p. 18-38.
13. Brewer, D.F. and M.J. Nash, *The chinese wall security policy*, in *1989 IEEE Symposium on Security and Privacy*. 1989, IEEE. p. 206-214.
14. OMG *Object Managment Group, Unified Modeling Language :OCL. Version 2.0* 2003.
15. Calvin Powers, I.T.S., cspowers at us.ibm.com, Matthias Schunter, IBM Research, mts at zurich.ibm.com, *Enterprise Privacy Authorization Language (EPAL 1.2)*, P. Ashley, et al., Editors. 2003.
16. Lobo, J., R. Bhatia, and S. Naqvi, *A Policy Description Language*, in *AAAI Conf. on Artificial Intelligence. American Association for Artificial Intelligence*. July, 1999: Orlando, Florida, USA. p. 291-298.
17. Zhang, N., M. Ryan, and D.P. Guelev, *Synthesising verified access control systems through model checking.* J. Comput. Secur., 2008. **16**(1): p. 1-61.
18. McMillan, K.L., *Symbolic Model Checking. PhD thesis*. 1993.

19. Zhang, N. *AcPeg tool, the access control policy evaluator and generator*. 2006; Available from: www.cs.bham.ac.uk/ ~ nxz or www.cs.bham.ac.uk/ ~ mdr/research/projects/05-AccessControl.
20. Chen, L. and J. Crampton, *On spatio-temporal constraints and inheritance in role-based access control*, in *Proceedings of the 2008 ACM Symposium on information, Computer and Communications Security. ASIACCS '08. AC*. March 18 - 20, 2008: Tokyo. p. 205-216.
21. Lee, S.w., et al., *Building problem domain ontology from security requirements in regulatory documents*, in *Proceedings of the 2006 international Workshop on Software Engineering For Secure System. SESS '06*. 2006, ACM, New York, NY, . DOI= http://doi.acm.org/10.1145/1137627.1137635. p. 43-50.
22. Valente, A., *Legal Knowledge Engineering - A Modelling Approach*. 1995: IOS Press.
23. Otto, P.N. and A. Antón, *Addressing Legal Requirements in Requirements Engineering*, in *15th IEEE International Requirements Engineering Conference*. 15-19 October 2007: Delhi, India. p. 5 - 14.
24. OMG *Object Managment Group, UML 2.0 Superstructure FTF Rose model containing the UML 2 metamodel*. 2004.
25. Kleppe, A., J. Warmer, and W. Bast, *MDA Explained, The Model Driven Architecture: Practice And Promise*. 2002: Addison-Wesley.
26. OMG, *Object Management Group, MetaObjectFacility(MOF) Specification*. 2003.
27. Lin, T., *Chinese Wall Security Policy - An Aggressive Model*, in *Fifth Annual Computer Security Applications Conference*. 1989. p. 282-289.
28. logrippo, L., *Logical Method for Reasoning about Access Control and Data Flow Control Models*, in *the 7th International Symposium on Foundations and Practice of Security (FSP 2014)*. 2014. p. 205-220.
29. Biba, K., *Integrity Considerartions for Secure Compouter Systems*. April 1977, The Mitre Corporation.
30. Arthur, K., M. Olivier, and H. Venter, *Applying the Biba integrity model within a forensic evidence management system*, in *Shenoi, Philip Craiger and Sujeet (eds), Advances in Digital Forensics III*. 2007, Springer. p. 317-327.
31. Sandhu, R.S., *Lattice-based access control models.* Computer, 1993. **26**(11): p. 9-19.
32. Denning, D.E., *A lattice model of secure information flow.* Commun. ACM, 1976. **19**(5): p. 236-243.
33. Thomas, R.K., *Team-based access control (TMAC): A primitive for applying role-based access controls in collaborative environments*, in *proceedings of the Second ACM Workshop on Role-Based Access Control*. 1997. p. 13-19.
34. Clarck, D.D. and D.R. Wilson, *A Comparison of Commercial and Military Computer Security Policies*, in *1987 IEEE Symposium on Security and Privacy*. 1987, IEEE Computer Society Press. p. 184-194.
35. Bjørner, D., *Specification of Systems and Languages* in *Software Engineering 2*. 2006.
36. Lupu, E.C. and M. Sloman. *Conflicts in Policy-Based Distributed Systems Management.* in *IEEE Trans. on Software Engineering, 25(6): 852-869*. Nov.1999.

37. Backes, M., M. Dürmuth, and R. Steinwandt, *An algebra for Composing Enterprise Privacy policies* in *Research Report 3557, IBM Research*. 2004, Springer. p. 33-52.
38. Kam-Ho, T., J.J. Hull, and S.N. Sargur, *Combination of decisions by multiple classifiers*, in *Structured Document Image Analysis*. 1999, Springer-Verlag. p. 188-202.
39. Malik, S. and L. Zhang, *Boolean satisfiability from theoretical hardness to practical success.* Commun. ACM, 2009. **52**(8): p. 76-82.
40. Jackson, D., ed. *Micromodels of Software: Lightweight Modelling and Analysis with Alloy.* 2002.
41. Gallier, J.H., *Logic for computer science: foundations of automatic theorem proving*. 2015, Courier Dover Publications.
42. Barker, S. and P. Stuckey, *Flexible access control policy specification with constraint logic programming.* 2003. **6**(4): p. 501-546.
43. Artikis, A., et al., *Logical Approaches to Authorization Policies*, in *Logic Programs, Norms and Action*. Springer Berlin Heidelberg. p. 349-373.
44. Slimani, N., et al., *UACML: Unified access control modelling language*, in *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*. 2011, IEEE. p. 1-8.
45. Craven, R., et al., *Expressive policy analysis with enhanced system dynamicity*, in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*. 2009, ACM. p. 239-250.
46. Gelfond, M. and J. Lobo, *Authorization and Obligation Policies in Dynamic Systems. In: Garcia de la Banda, M., Pontelli, E. (eds.)* ICLP LNCS, 2008(5366): p. 22-36.
47. Becker, M.Y., C. Fournet, and A.D. Gordon, *Design and semantics of a decentralized authorization language.* CSF, 2007: p. 3-15.
48. Epstein, P. and R. Sandhu, *Towards a UML based approach to role engineering*, in *Proceedings of the fourth ACM workshop on Role-based access control*. 1999, ACM: Fairfax, Virginia, USA. p. 135-143.
49. Shin, M. and G. Ahn, *UML-based representation of role-based access control*, in *Proceedings of 9th IEEE International Workshops on Enabling Technologies (WETICE'00)*. 2000. p. 195-200.
50. Doan, T., et al., *MAC and UML for secure software design*, in *Proceedings of 2004 ACM workshop on Formal methods in security engineering (FMSE'04)*. 2004. p. 75-85.
51. Jürjens, J., *Towards development of secure systems using UMLsec*, in *International Conference on Fundamental Approaches to Software Engineering*. 2001, Springer. p. 187-200.
52. Basin, D., J. Doser, and T. Lodderstedt, *Model driven security: From UML models to access control infrastructures.* ACM Transactions on Software Engineering and Methodology, 2006. **15**(1): p. 39-91.
53. Pavlich-Mariscal, J.A., S.A. Demurjian, and L.D. Michel, *A framework of composable access control features: Preserving separation of access control concerns from models to code.* Computers & Security, 2010. **29**(3): p. 350-379.

54. Jajodia, S., et al., *Flexible support for multiple access control policies.* ACM Transactions on Database Systems (TODS), 2001. **26**(2): p. 214-260.
55. Abd-Ali, J., K. El Guemhioui, and L. Logrippo, *A Metamodel for Access Control Policies.* The Journal of Software, 2015. **10**(7): p. 784-797.
56. Barrett, C., A. Stump, and C. Tinelli, *The Satisfiability Modulo Theories Library (SMT-LIB)*. 2010.
57. Abd-Ali, J., K. El Guemhioui, and L. Logrippo, *Metamodelling with Formal Sematics with Application to Access Control Specification*, in *MODELSWARD*. 2015: Angers, France. p. 354-362.
58. Halpern, J.Y., et al., *On the Unusual Effectiveness of Logic in Computer Science.* Bulletin of Symbolic Logic, 2001. **7**(2): p. 213-236.
59. Zhao, H., J. Lobo, and S.M. Bellovin, *An Algebra for Integration and Analysis of Ponder2 Policies*, in *Policies for Distributed Systems and Networks. POLICY 2008 IEEE workshop*. 2008. p. 74-77.