Université du Québec en Outaouais

Département d'informatique et d'ingénierie

Un environnement pour la modélisation des systèmes de contrôle d'accès

MEMOIRE

PRESENTE

COMME EXIGENCE PARTIELLE

DE LA MAITRISE EN INFORMATIQUE

PAR

Bone Maboudou

Hiver 2015

Mémoire

présenté par : Bone Maboudou

pour l'obtention du titre de maître ès sciences (M.sc)

évalué par un jury composé de :

Dr. Kamel Adi.....Directeur de recherche

Dr. Luigi Logrippo.....Co-directeur de recherche

Dr. Michal Iglewski.....Président du jury

Dr. Jurek Czyzowicz.....Membre du jury

Table des matières

Table des figures

Liste des tableaux

Remerciements

Je voudrais remercier mon directeur de recherche Dr. Kamel Adi et mon Co-directeur de recherche Dr. Luigi Logrippo pour leur disponibilité, leurs conseils, leur soutien et pour toutes les formes d'aides qu'ils m'ont apporté dans ce travail.

Je voudrais également remercier mes parents et tous les membres de ma famille pour leur soutien.

Je voudrais enfin remercier toutes les personnes qui m'ont aidé tout au long de ce travail.

Résumé

Le contrôle d'accès est une composante essentielle pour la sécurisation de tout système informatique et plusieurs modèles de contrôle d'accès ont été proposés dans la littérature. L'objectif étant d'exprimer de manière simple et naturelle les exigences de sécurité en termes de contrôle d'accès des utilisateurs aux ressources critiques d'un système. Cependant, de nos jours, ces modèles dits universels ont montré leurs limites pour capturer les besoins des grandes organisations et il n'est pas rare qu'une organisation élabore son propre modèle comme une hybridation de différentes composantes des modèles standards. Dans ce mémoire, nous nous proposons d'élaborer un cadre formel et pratique pour la spécification et la validation de systèmes de contrôle d'accès hybrides. En particulier, nous nous proposons d'élaborer une méthodologie pour spécifier des modèles de contrôle d'accès, par raffinement successif, à l'aide d'une notation proche de UML. Les spécifications sont ensuite automatiquement traduites en notation formelle B-événementiel afin de prouver formellement des propriétés de modèles.

Chapitre 1

Introduction

Les systèmes de contrôle d'accès sont des mécanismes qui autorisent ou interdisent à un sujet (entité capable d'initier des requêtes : personne, logiciel, etc.) l'accès à des objets (entités à protéger : fichiers, dossiers, etc.). Ces systèmes de contrôle d'accès sont créés sur la base de modèles de contrôle d'accès. Les modèles de contrôle d'accès sont des abstractions des systèmes de contrôle d'accès. Ils permettent de présenter des politiques de sécurité de façon à atteindre des objectifs particuliers. Plusieurs types de modèles de contrôle d'accès existent dans la littérature. Quelques-uns de ces modèles sont : les modèles MAC (Mandatory Access Control), les modèles DAC (Discretionary Access Control), le modèle RBAC (Role-Based Access Control) etc.

Dans notre mémoire, nous nous interressons à la modélisation des systèmes de contrôle d'accès «*hybrides*». Ces systèmes sont obtenus à partir des modèles de contrôle d'accès hybrides. Les modèles de contrôle d'accès hybrides sont le résultat de la combinaison de plusieurs concepts extraits des modèles de contrôle d'accès tels que les modèles MAC, DAC, RBAC, etc. Leur existence découle du fait que les besoins en sécurité sont diversifiés aujourd'hui.

En effet, on rencontre de plus en plus de situations dans lesquelles, il ne suffit plus d'un modèle de contrôle d'accès pour répondre aux exigences de

sécurité d'un système. Dans ce genre de situation, on a besoin de combiner deux, voir plusieurs concepts (rôles, groupes, niveaux de sécurité, etc.) de différents modèles de contrôle d'accès. Par exemple, avec l'utilisation de l'infonuagique (Cloud Computing), on peut avoir besoin de créer un modèle de contrôle d'accès hybride. L'infonuagique est une technologie de plus en plus répandue qui propose l'hébergement des informations à distance de façon à y accéder en tout temps. Ainsi, si une entreprise décide de stocker ses informations avec le Cloud, ce dernier doit implémenter un modèle de contrôle d'accès qui s'adapte non seulement à l'organisation de cette entreprise, mais aussi au besoin d'accès aux informations depuis l'extérieur de l'entreprise. Par exemple si une entreprise accorde à son personnel des permissions en fonction du rôle que joue chaque employé (par exemple : *tous les employés ayant* ***le rôle de directeur*** *peuvent consulter les fiches de présence des employés de leur département*), le Cloud peut implémenter un modèle RBAC pour l'entreprise, car RBAC est basé sur les rôles. Toutefois si l'entreprise a plusieurs succursales divisées en groupes (par exemple selon la région ou la succursale est établie), le Cloud doit aussi tenir compte du fait que pour accéder à des informations depuis l'extérieur de l'entreprise, un employé d'une succursale sera identifié par le groupe auquel sa succursale appartient et le rôle qu'il joue au sein de cette succursale. Il en résulte qu'en plus du modèle RBAC ce Cloud doit aussi implémenter un modèle qui tient compte des groupes.

Une autre situation pouvant nécessiter la création d'un modèle de contrôle d'accès hybride est la fusion de deux entreprises dont les systèmes d'informations implémentent des modèles de contrôle d'accès différents. La nouvelle entreprise issue de cette fusion doit tenir compte des modèles de contrôle d'accès des deux entreprises pour produire un contrôle d'accès éfficace. Si la première entreprise a un modèle de contrôle d'accès basé sur les rôles (RBAC) et la seconde un modèle de contrôle d'accès basé sur les groupes, la nouvelle entreprise fusionnée doit tenir compte du fait que les employés peuvent continuer à accéder aux informations qui leurs sont autorisées à partir de leurs rôles (RBAC), de leurs groupes, et à la fois à partir de leurs rôles

et de leurs groupes.

De ces exemples, il ressort que la création des modèles de contrôle d'accès hybrides peut aboutir à des modèles qui ne sont pas explicitement définis dans la littérature. Pour ces modèles hybrides, il faut être en mesure de concerver les exigences de sécurité des différents modèles ayant contribué à leur création. Nous allons par conséquent nous interresser dans ce mémoire, aux procédures de *créations* et de *modélisations* des systèmes basés sur les modèles hybrides. Ces systemes peuvent devenir complexes lorsqu'ils sont de taille importante. Nous utiliserons par conséquent le « *raffinement* » afin de produire une succession de modèles en nous intéressant à chaque fois et dans chaque modèle à des propriétés précises.

Le raffinement est un concept qui permet de créer des modèles en considérant un petit nombre de propriétés à la fois. Ce concept est intéressant lorsqu'on doit considérer des modèles complexes. Pour notre mémoire la modélisation par raffinement va consister à créer une succession de modèles dont chacun répond à des objectifs précis. Nous aurons ainsi une meilleure vision des problèmes potentiels que le modèle final pourrait générer.

1.1 Motivations

La modélisation d'un système informatique est une tâche faisant partie du processus de développement global du système. Elle consiste à exprimer dans un langage semi-formel ou formel, les exigences de ce système. Dans des systèmes complexes où les besoins en sécurité sont importants, des procédés doivent être mis au point pour guider le processus de modélisation afin d'assurer que toutes les exigences de sécurité sont atteintes.

Aujourd'hui, UML (Unifed Modeling Language) est le langage de modélisation semi-formel le plus utilisé. Il fournit des notations qui permettent de présenter graphiquement les exigences du système à développer. Le but recherché est de faciliter la compréhension des modèles pour les différentes personnes qui interviennent dans le développement du système. L'utilisation

d'UML peut donc être indispensable pour un système complexe. Toutefois pour certaines exigences, il peut être nécéssaire d'utiliser des techniques de modélisations formelles afin d'exprimer et montrer les effets de ces exigences sur le reste des éléments du modèle. A titre d'exemple, il est simple de modéliser par un diagramme de classes UML, le fait qu'un étudiant soit inscrit dans un cours. Par contre, si une contrainte impose que tous les étudiants soient inscrits à un maximum de cinq cours, alors il devient plus difficile de modéliser les effets de cette contrainte sur toutes les opérations qui pourraient être disponibles dans le système. Par exemple cette contrainte pourrait avoir effet sur l'opération qui permet à un étudiant d'ajouter son nom dans un nouveau cours. Cependant elle ne devrait pas être violée par une autre opération qui permet d'ajouter un groupe d'étudiants dans un cours. Dans un système complexe, il faudra alors repérer toutes les opérations concernées, afin d'éviter d'éventuelles contradictions.

Il existe dans la littérature des méthodes de spécification semi-formelle de modèles de contrôle d'accès et des méthodes pour la spécification de modèles hybrides. Parmi ces méthodes nous nous intéressons particulièrement à la méthode semi-formelle CatBAC (Category Based Access Control). Cette méthode permet grâce à des concepts assez généraux et grâce au raffinement (concept qui permet de produire un modèle final à partir d'une succession de modèles se rapprochant de plus en plus des objectifs finaux (chapitres 3 et 4)) de produire des modèles de contrôle d'accès hybrides basés sur UML. Cependant CatBAC à elle seule ne peut garantir que l'implémentation des modèles créés à partir de notations UML puisse satisfaire aux exigences de sécurité d'un système. Il est alors nécessaire d'introduire de nouvelles techniques pour faire en sorte que les implémentations des modèles CatBAC soient correctes par rapport à des propriétés de sécurité. Nous proposons par conséquent dans ce mémoire, de spécifier par raffinement des systèmes de contrôle d'accès sûrs par construction. Pour y arriver nous utilisons la méthode CatBAC combinée à une méthode de modélisation formelle appelée B-événementiel.

B-événementiel est une méthode de spécification formelle de système par raffinement. Cette méthode dispose de mécanismes pour vérifier et prouver des modèles par rapport aux propriétés concernées. Contrairement aux vérifications des méthodes classiques, les vérifications dans B-événementiel ne reposent pas sur l'exécution de codes. Elles sont plutôt basées sur l'élaboration de preuves mathématiques, à mesure que la modélisation se poursuit.

En effet, une modélisation avec B-événementiel aboutit à un ensemble de spécifications formelles sur lequel on peut prouver mathématiquement des propriétés. Ces preuves permettent d'assurer que le modèle en construction préserve les propriétés qu'on lui impose. Il est donc possible de repérer à un niveau de raffinement donné, les erreurs potentielles que le futur système pourra générer face à certaines propriétés de sécurité. Dans l'exemple précédent qui concerne l'inscription d'un étudiant à un maximum de cinq cours, une modélisation avec B-événementiel nous permettrait de repérer facilement les opérations qui pourraient contredire cette propriété. Appelons P1 la propriété suivante : *tous les étudiants doivent s'inscrire à un maximum de cinq cours.* Pour modéliser par raffinement le système avec P1, il suffit de créer pour ce cas particulier, deux modèles successifs dont le second (M2) raffine le premier (M1). Dans le premier modèle M1, nous pouvons modéliser P1 et une première opération qui permet à un étudiant d'ajouter son nom à un cours. Nous pouvons ensuite prouver dans un raisonnement mathématique que cette opération ne pourra jamais violer la propriété P1. Puisque le modèle M2 raffine le modèle M1, B-événementiel concerve dans M2 toutes les spécifications et les preuves déja effectuées dans M1. Il suffit donc de modéliser dans M2 l'oprération qui permet d'ajouter un groupe d'étudiant dans un cours et de prouver que cette opération ne violera pas elle aussi la propriété P1. On peut contunier ce shéma en ajoutant le reste des opérations que le système doit implémenter. Dans le processus de modélisation avec B événementiel le nombre d'opérations à spécifier par modèle n'est pas liminité à un. Plusieurs opérations et propriétés peuvent être implémentées à un niveau de raffinement donné.

Grace à la méthode B-événementiel nous allons également modéliser par

raffinement des propriétés de sécurité qui bien souvent ne peuvent pas être exprimées adéquatement dans des modèles graphiques.

1.2 Objectifs du mémoire

Notre objectif global dans ce mémoire est de construire un cadre pour la spécification et la vérification des systèmes de contrôle d'accès hybrides. Pour ce qui est de la spécification, nous utiliserons la méthode CatBAC associée à la méthode B-événementiel. Nous proposons,

- une description formelle du raffinement des modèles CatBAC. Il s'agit d'indiquer la façon dont les composants UML des modèles CatBAC doivent être interprétées formellement après chaque niveau de raffinement. Par exemple que devient l'association en deux classes d'un modèle si l'une de ces classes a été raffinée ?

- un ensemble de règles permettant de simplifier le passage de la description semi-formelle des exigences de sécurité avec CatBAC vers des modèles formels avec B-événementiel. En effet, pour pouvoir être formellement vérifiés, les modèles CatBAC doivent être traduits en B-événementiel. Nous montrerons au chapitre 3 que l'interprétation des modèles CatBAC peut donner lieu à de nombreuses possibilités sur la façon dont les éléments des modèles doivent être compris. Il est donc important de s'assurer que cette interprétation soit proche de celle de la majorité des modèles de contrôle d'accès qui, à un sujet, associent une permission.

- un outil permettant de modéliser des systèmes de contrôle d'accès sûrs par construction avec CatBAC en B-événementiel. Cet outil expérimental permettra également d'exprimer des propriétés de modèles que le système final doit garrantir. A ce niveau nous formalisons une approche de modélisation par décomposition (Chapitre 6) qui permet d'obtenir des modèles qui reflètent les exigences des systèmes.

Pour ce qui est de la vérification, l'outil Rodin conçu pour la modélisation avec B-événementiel offre plusieurs plugins permettant d'assurer une bonne vérification des modèles. Il s'agit de plugins comme *Theory* qui permet d'étendre le langage mathématique associé à B-événementiel, *Proof Purger* qui permet d'éliminer les preuves non indispensables à la modélisation etc. Pour bénéficier de tous ces outils nous allons intégrer notre outil de création et de traduction de modèles à l'environnement Rodin.

La structure du présent document est comme suit ; au chapitre 2, quelques modèles de contrôle d'accès existants dans la littérature sont présentés. Trois types de modèles nous intéressent particulièrement : les modèles discretionnaires, les modèles obligatoires, et les modèles basés sur les rôles. Dans le chapitre 3, nous discutons de quelques méthodes de modélisations semi-formelles de politiques de sécurité et de la méthode CatBAC. La méthode B-événementiel est présentée au chapitre 4. Au chapitre 5, nous discutons des différentes techniques de traductions de UML vers B-événementiel. Nous discutons ensuite de la façon dont nous interprétons le raffinement dans les modèles CatBAC. Dans le chapitre 6, nous proposons une approche de modélisation par décomposition des systèmes avec CatBAC et B-événementiel. Au chapitre 7, nous présentons notre outil de modélisation avec quelques exemples. Enfin les conclusions générales sont exposées au chapitre 8.

Chapitre 2

Les modèles de contrôle d'accès

Afin d'avoir une idée des structures que peuvent avoir les modèles de contrôle d'accès hybrides, nous présentons dans ce chapitre quelques modèles de contrôles d'accès existants dans la littérature.

2.1 Introduction

Un système de contrôle d'accès est un mécanisme qui autorise ou interdit à un sujet (entité capable d'initier des requêtes : personne, logiciel, etc.) l'accès à des objets (entités à protéger : fichiers, dossiers, etc.). Les décisions concernant ces autorisations et interdictions sont prises par le système en conformité avec des règles de contrôle d'accès. Les systèmes de contrôle d'accès sont conçus sur la base de modèles de contrôle d'accès. Ces derniers sont des abstractions des systèmes de contrôle d'accès. Ils permettent de présenter des ensembles de règles de contrôle d'accès (politique de sécurité) de façon à atteindre des objectifs particuliers de sécurité. Ces objectifs de sécurité dépendent généralement de la structure organisationnelle ou du modèle de gestion interne et externe des compagnies ou entreprises qui implémentent ces modèles de contrôle d'accès.

Pour atteindre différents objectifs de sécurité, plusieurs modèles de contrôle d'accès ont été proposés. Dans ce chapitre nous présenterons quelques mo-

dèles de contrôle d'accès qui seront utilisés plus tard dans ce mémoire. Il s'agit des modèles DAC, MAC et RBAC.

2.2 Les contrôles d'accès discrétionnaires

Dans les modèles de contrôles d'accès discrétionnaires (*DAC : Discretionary Access Control*) des sujets peuvent attribuer des droits d'accès sur des objets à d'autres sujets. Les attributions de droits d'accès sont ainsi laissées à la discrétion des sujets.

2.2.1 Modèle de Lampson

Le modèle de Lampson est proposé dans [16]. Dans ce modèle les entités qui ont accès aux objets sont appelées domaines. Par soucis d'uniformité nous utiliserons le terme sujets. Supposons un ensemble S de sujets et un ensemble O d'objets. Le modèle de contrôle d'accès proposé par Lampson introduit une notion de matrice d'accès. Cette matrice contient l'ensemble des droits d'accès associés aux sujets et aux objets. Soit M_{so} une matrice d'accès qui associe pour des couples (s, o) des ensembles de droits d'accès. Le modèle peut alors être présenté avec le triplet (S, O, M_{so}).

	o_1	o_2	o_3	o_4
s_1	écrire	lire		
s_2			lire	lire, écrire
s_3		lire, écrire	lire	
s_4	écrire			lire

FIGURE 2.1 – Matrice de contrôle d'accès

Dans la Figure 2.1 les droits d'accès «lire» et «écrire» sont associés au couple (s_3, o_2) qui correspond à l'association d'un sujet s_3 et d'un objet o_2. s_3 peut donc accéder en lecture et en écriture à l'objet o_2. Les politiques de sécurité exprimées sur la base du modèle de Lampson sont couteuses en mises à jour. En effet si de nouveaux sujets, objets ou droits d'accès sont

introduits ou supprimés dans la politique, il est nécessaire de produire des modifications sur chaque enregistrement concerné.

2.2.2 Modèle de Harrison-Ruzzo-Ullman (HRU)

Dans [17], Harrison, Ruzzo et Ullman présentent un modèle qui généralise celui de Lampson. Soit S un ensemble de sujets, O un ensemble d'objets et M_{so} une matrice correspondant à l'ensemble des droits d'accès associé à un sujet s et un objet o. Comme pour le modèle de Lampson, un sujet peut donner des droits d'accès sur des objets à d'autres sujets. Par exemple si un droit *own* est associé à un couple (s, o), alors le sujet s est propriétaire de l'objet o et peut attribuer des droits d'accès à d'autre sujets sur l'objet o.

Dans le modèle HRU, un ensemble de commandes correspondant à des opérations élémentaires est fourni pour permettre aux sujets d'attribuer des droits d'accès sur les objets dont ils sont propriétaires, ou suivant les droits qui leurs ont été concédés. Ces commandes permettent également de créer de nouvelles opérations. Les opérations élémentaires sont données ci-dessous :

- entrer un droit r dans M_{so}
- supprimer r de M_{so}
- créer un sujet s
- supprimer un sujet s
- créer un objet o
- supprimer un objet o

A titre d'exemple supposons que la création d'un nouveau fichier est suivie de l'affectation d'un certain sujet à ce fichier en tant que propriétaire, avec le droit *own*. Une nouvelle commande $create_f$ peut être définie pour exécuter cette opération avec la séquence suivante :

- créer un objet (le $fichier$)
- entrer le droit *own* dans M_{sf} ; s étant un sujet et f le fichier nouvellement créé.

2.2.3 Conclusion sur les modèles discrétionnaires

Les modèles discrétionnaires sont assez flexibles. Nous pouvons retrouver leurs implémentations dans les sytèmes Linux. Ces modèles sont utilisés en pratique dans les situations où le propriétaire d'une information peut décider de son utilisation. Notre méthode de modélisation s'applique à ces modèles. Néanmoins les modèles discrétionnaires posent quelques problèmes de sécurité dans certaines circonstances. En effet du fait que l'attribution des droits d'accès est laissée aux sujets, il peut arriver qu'un sujet accorde des droits "sensibles" à d'autres sujets malveillants. Dans ce cas un sujet malveillant peut compromettre des informations importantes ou accéder à du contenu qu'il ne devrait pas. Les chevaux de troie (Section 2.3) sont un exemple de ce type de problème.

2.3 Le contrôle d'accès obligatoire

Le contrôle d'accès obligatoire (*MAC : Mandatory Access Control*) introduit des notions différentes au contrôle d'accès discrétionnaire. Contrairement aux modèles DAC, dans les MAC, l'attribution des droits d'accès n'est pas laissée aux sujets de la même façon. Elle est plutôt réglée par des principes obligatoires.

2.3.1 Modèle de Bell LaPadula

Le modèle de Bell et LaPadula a été proposé dans [18] pour le Département de la Défense Américaine en 1975. L'objectif principal de ce modèle est de prévenir la divulgation d'informations au sein d'un système informatisé. Dans ce modèle les sujets et les objets sont regroupés par niveaux de sécurité. Cela consiste à les classer en fonction de leurs importances dans le système de sécurité. Par exemple les sujets et les objets peuvent être classés dans les niveaux de sécurité suivants : *confidentiel, secret, top secret.*

L'importance des niveaux de sécurité est représentée par la relation d'ordre *inférieure ou égale* ($\leq$). Nous pouvons par exemple ordonner les niveaux de

sécurité précédents comme suit : $confidentiel \leq secret \leq top\ secret$. Lorsqu'un sujet reçoit un niveau de sécurité, on parle de niveau d'habilitation. La fonction d'habilitation est comme suit : $f_S : S \rightarrow L$. S désigne l'ensemble des sujets et L l'ensemble des niveaux de sécurité.

De façon similaire lorsqu'un objet reçoit un niveau de sécurité, on parle de niveau de classification. La fonction de classification est : $f_O : O \rightarrow L$, avec O l'ensemble des objets.

Dans Bell LaPadula, les autorisations d'accès des sujets aux objets sont décidées sur la base de deux propriétés qui doivent être satisfaites. Il s'agit de la propriété simple et de la propriété étoile.

Propriété simple

Cette propriété indique qu'un sujet s peut **lire** un objet o, si son niveau d'habilitation est supérieur ou égal au niveau de classification de l'objet. Cela peut être formalisée comme ceci : $\forall s \in S, \forall o \in O, (s, o, read) \in M \Rightarrow f_S(s) \geq f_O(o)$ avec M un ensemble d'accès courant. Cette propriété formalise la notion de confidentialité de l'information.

Propriété étoile

La propriété étoile indique qu'un sujet s peut **écrire** dans un objet o, si son niveau d'habilitation est inférieur ou égal au niveau de classification de l'objet. Formellement il s'agit d'appliquer la propriété suivante : $\forall s \in S, \forall o \in O, (s, o, write) \in M \Rightarrow f_S(s) \leq f_O(o)$.

Les deux propriétés peuvent être respectivement traduites de façon simple par «pas de lecture au niveau supérieur», «pas d'écriture au niveau inférieur». Un des objectif de la propriété étoile est de prévenir les attaques sous forme de chevaux de Troie. Un cheval de Troie est une application qui exécute certaines actions à l'insu de l'utilisateur qui a exécuté cette application. Il utilise ainsi le droit de l'utilisateur pour détourner des informations. Pour éviter que dans une telle situation le cheval de Troie ne diffuse des informations clas-

sées top secret dans un document confidentiel ($confidentiel \leq secret \leq top\ secret$), on lui interdit l'écriture de ces informations. Tout le modèle préserve ainsi la confidentialité de l'information.

2.3.2 Modèle d'intégrité de Biba

L'intégrité est une propriété de sécurité qui consiste à s'assurer que des informations n'ont pas été modifiées au cours d'une opération à l'insu des utilisateurs.

En 1977 Kenneth J. Biba a proposé dans [19] un modèle de contrôle d'accès en mettant l'accent sur l'intégrité des données. Tandis que le modèle Bell LaPadula prend en considération la divulgation des informations, le modèle Biba prend en considération l'intégrité. Dans ce modèle on parlera donc de niveau d'intégrité. Tout comme pour le modèle Bell LaPadulla, les accès des sujets aux objets dépendent respectivement de leurs habilitations et de leurs classifications. Pour garantir cela, les propriétés simple et étoile sont redéfinies.

Propriété simple

Cette propriété indique qu'un sujet s peut lire un objet o si l'habilitation de s est inférieur ou égal à la classification de o : $\forall s \in S, \forall o \in O, (s, o, read) \in M \Rightarrow f_S(s) \leq f_O(o)$ avec M un ensemble d'accès courant, S un ensemble de sujets, O un ensemble d'objets, f_S et f_O les fonctions d'habilitation et de classification.

Propriété étoile

Cette propriété indique qu'un sujet s peut écrire dans un objet o si l'habilitation de s est supérieur ou égal à la classification de o : $\forall s \in S, \forall o \in O, (s, o, read) \in M \Rightarrow f_S(s) \geq f_O(o)$ avec M un ensemble d'accès courant, S un ensemble de sujets et O un ensemble d'objets,f_S et f_O les fonctions d'habilitation et de classification.

Ces règles peuvent être reprises comme suit : «pas de lecture au niveau

inférieur», « pas d'écriture au niveau supérieur». En considérant la classification des niveaux d'intégrité $confidentiel \leq secret \leq top\ secret$, il faut empêcher qu'un sujet qui à l'habilitation *secret* ne modifie des objets de classification *top secret*. Ainsi en cas d'attaque, l'intégrité d'un document *top secret* sera maintenu en empêchant des sujets malveillants d'y introduire de mauvaises informations. Les sujets peuvent toutefois lire les objets de classification *top secret* mais pas ceux de classification *confidentiel*.

2.3.3 Modèle de Brewer et Nash

Le modèle de Brewer et Nash dit de la Muraille de Chine a été introduite dans [20] pour résoudre les problèmes de conflits d'intérêts dans les organisations commerciales.

L'objectif du modèle est de fournir un contrôle du flux d'informations qui pourrait provenir d'un sujet se trouvant dans une situation de conflit d'intérêts. On parle de conflit d'intérêts lorsqu'un sujet à accès aux informations confidentielles de deux organisations en compétitions.

Les éléments du modèle sont : les *objets* (éléments appartenant à une organisation), l'organisation (disposant de plusieurs objets), les *classes de conflits d'intérêts* (regroupant les organisations en concurrence). Ces éléments sont présentés hiérarchiquement de façon à ce que, les objets correspondent au niveau inférieur (bas niveau), les organisations au niveau intermédiaire et les classes de conflits d'intérêts au niveau supérieur (haut niveau).

La Figure 2.2 montre une illustration du modèle de la Muraille de Chine. Comme énoncé précédemment les classes de conflits d'intérêts $C1$ et $C2$ sont représentées au niveau supérieur. Au niveau intermédiaire, on distingue les organisations $Org1$ et $Org2$ qui sont dans la classe $C1$ et $Org3$ qui est dans la classe $C2$. Au niveau inférieur on distingue les objets $inf1$ appartenant à $Org1$, $inf2$ et $inf3$ appartenant à $Org2$, $inf4$ appartenant à $Org3$.

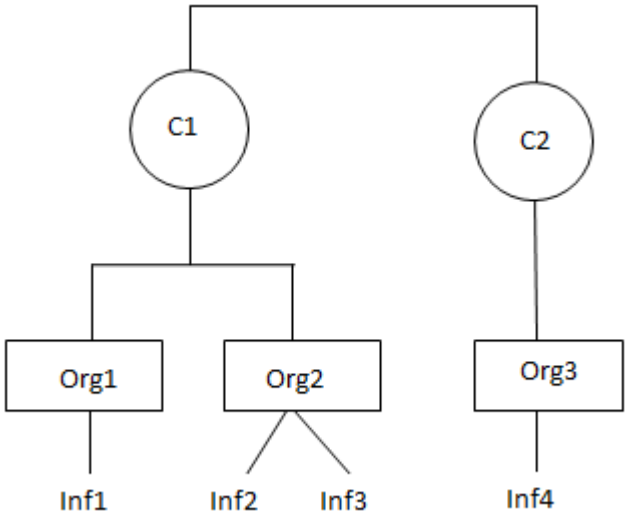

FIGURE 2.2 – Modèle de la Muraille de Chine

Dans le modèle de la Muraille de Chine, un sujet se place dans une classe de conflit d'intérêts lorsqu'il accède la première fois aux informations d'une organisation de la même classe. Les propriétés simple et étoile sont définies pour ce modèle comme suit :

Propriété simple

Cette propriété interdit l'accès en lecture d'un sujet à un objet si cet objet appartient à une organisation de la même classe de conflit d'intérêts que les organisations propriétaires d'objets auxquels ce sujet a déjà accédé en lecture.

Dans la Figure 2.2, si un sujet accède en lecture à $inf1$, il ne peut pas accéder aux objets $inf2$ et $inf3$ car les organisations $Org1$ et $Org2$ se trouvent dans la même classe de conflit d'intérêts $C1$.

Propriété étoile

Cette propriété interdit à un sujet d'écrire dans un objet d'une organisation si ce sujet a déjà accédé en lecture aux objets d'une autre organisation de la même classe de conflits d'intérêts. Si un sujet accède en lecture à $inf1$, il ne peut pas accéder en écriture à $inf2$ ou $inf3$.

Ces deux propriétés doivent être respectées afin d'empêcher un sujet d'écrire dans $Org2$ des informations qu'il a lu dans $Org1$.

2.3.4 Conclusion sur le contrôle d'accès obligatoire

Nous avons présenté dans cette section des modèles de contrôle d'accès multi-niveaux liés à l'organisation militaire (Bell Lapadula et Biba) et un modèle lié à l'organisation commerciale (Muraille de Chine). La littérature propose d'autres types de modèles MAC. Nous montrerons plus tard dans notre mémoire comment des modèles MAC peuvent être modélisés avec Cat-BAC.

2.4 Le modèle RBAC

RBAC : *Role-Based Access Control* (contrôle d'accès basé sur les rôles) [21] est un modèle de contrôle d'accès organisé différemment des modèles DAC et MAC. Les éléments de ce modèle sont : les *utilisateurs* (*users*), les *sujets* (*subjects*), les *rôles* (*roles*), les objets (*objects*), les *opérations* (*operations*) et les *permissions* (*permissions*).

Un utilisateur est une personne qui interagit avec le système informatique. Par exemple dans les systèmes d'exploitation, un utilisateur se connecte à partir de son compte. Une instance du dialogue entre un utilisateur et le système avec lequel il interagit est appelée *session*. Normalement, le rôle est la fonction que l'utilisateur occupe au sein d'une entreprise.

Un sujet est un processus qui agit dans une sessoin au nom d'un utilisateur. Il s'agit par exemple d'un programme lancé dans une session de cet utilisateur. Un utilisateur peut donc avoir plusieurs sujets.

Un objet représente une entité passive du système, Il s'agit de fichiers, dossiers, imprimantes etc. Une opération est un processus actif invoqué par un sujet sur un objet (accéder à un fichier, modifier des données, etc).

L'autorisation d'effectuer une action dans le système est une permission. Dans RBAC une permission combine un objet et une opération.

Les permissions dans RBAC sont accordées aux rôles. Les utilisateurs et les sujets héritent ensuite de ces permissions en fonction des rôles qu'ils jouent dans une entreprise. Dans le standard NIST (National Institute of Standards and Technology), le modèle RBAC est constitué de trois niveaux :

- Le noyau RBAC (*Core* RBAC) qui contient les fonctionnalités de base du modèle
- RBAC hiérarchique (*Hierarchical RBAC*) qui introduit une hiérarchie de rôles
- RBAC avec contraintes (*Constrained* RBAC) qui permet de spécifier des contraintes pour l'application des permissions.

2.4.1 Le noyau RBAC

Le noyau RBAC définit la structure de base du modèle RBAC. Les rôles sont affectés aux utilisateurs et les permissions aux rôles. La Figure 2.3 présente les relations entre les éléments du modèle. UA (*User Assignment*) désigne la relation qui existe entre les utilisateurs et les rôles. PA (*Permission Assignment*) relie les permissions aux rôles.

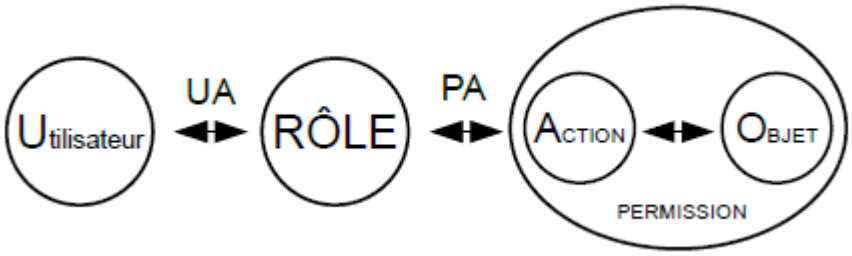

Figure 2.3 – Core RBAC (extrait de [6])

2.4.2 RBAC hiérarchique

RBAC hiérarchique ajoute en plus des composantes du noyau RBAC une hiérarchie entre les rôles. C'est un moyen de représenter les niveaux de responsabilité dans une entreprise. Dans la Figure 2.4, la relation *RH* (*Role Hierarchy*) représente l'hiérarchie entre les rôles.

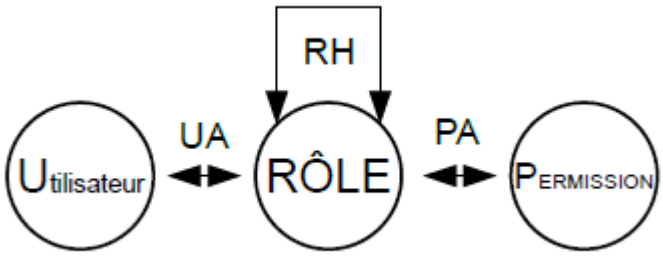

FIGURE 2.4 – RBAC hiérarchique (extrait de [6])

Si un rôle r_2 hérite de r_1 alors l'utilisateur qui joue le rôle r_2 aura en plus des permissions qui lui sont accordées, les permissions de r_1. Par exemple si on considère que le *chef du département informatique* est avant tout un simple employé d'entreprise, alors il a en plus des permissions accordées aux simples employés, les permissions liées au rôle *chef du département informatique*.

2.4.3 RBAC avec contraintes

Ce modèle permet de définir des contraintes afin d'éviter d'éventuels conflits dans l'affectation des rôles. On parle de *séparation de pouvoirs (SoD : Separtion of duty)*.

Séparation statique de pouvoirs

La séparation statique de pouvoirs (*SSoD : Static Separation of Duty*) permet d'éviter les conflits que peuvent engendrer l'affectation des rôles aux sujets. Il s'agit d'empêcher à un sujet de jouer des rôles en conflits. La nécéssité de gérer les conflits est également liée à la hiérarchie qui peut exister entre

les rôles. En effet pour respecter une certainne cohérence, les contraintes de séparation doivent être établies de façon à éviter que des rôles en hiérachie soient également en conflits. Avec la *SSoD* si le rôle *chef du département informatique* est en conflit avec le rôle *gestionnaire des équipements informatiques*, il faut vérifier qu'un utilisateur n'a pas un de ces rôles, avant de lui attribuer le second.

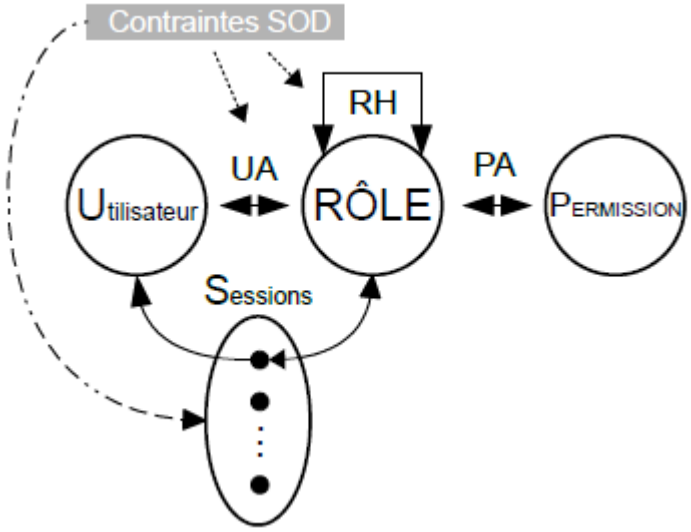

Figure 2.5 – RBAC avec contraintes de séparation de pouvoirs (extrait de [6])

Séparation dynamique de pouvoirs

La séparation dynamique de pouvoirs (*DSoD : Dynamic Separation of Duty*) permet d'empêcher à un utilisateur de jouer des rôles en conflit au même moment. En effet il existe des situations où différents rôles en conflit doivent être attribués à un utilisateur. Pour résoudre ce genre de problèmes, des rôles en conflits peuvent être affectés à un utilisateur, mais dans des sessions différentes. Un même sujet peut donc jouer les rôles *chef du département d'informatique* et *gestionnaire des équipements informatiques* dans des sessions différentes.

La Figure 2.5 présente le modèle RBAC avec contraintes. On remarque

dans cette figure que les contraintes SOD sont appliquées à la hiérarchie des rôles (RH) et à l'attribution d'un rôle à un utilisateur (UA).

2.4.4 Conclusion sur le modèle RBAC

Le modèle RBAC a été largement discuté dans la littérature. Il permet d'avoir une relation stable entre la sécurité des données, la structure d'une organisation et les tâches dans une organisation. Plusieurs exemples que nous donnerons dans notre travail seront liés à ce modèle. Nous présenterons dans le chapitre suivant une méthode de modélisation de politiques de sécurité qui se base également sur ce modèle.

2.5 Conclusion

Dans ce chapitre nous avons discuté de quelques modèles de contrôle d'accès. Chacun de ces modèles permet de présenter des règles de contrôle d'accès suivant un objectif particulier. Les modèles de contrôle d'accès hybrides sont le résultat de la combinaison de modèles de contrôle d'accès comme ceux présentés dans ce chapitre. Il est donc important de fournir des méthodes pour spécifier ces modèles. On retrouve par conséquent dans la littérature, des méthodes de modélisation formelles (avec des langages logiques) et des méthodes de modélisation semi-formelles (avec des aspects graphiques). Le but de ces méthodes est de permettre des modélisations sans ambiguïtés pour les méthodes formelles et de faciliter la compréhension et l'interprétation des modèles, pour les méthodes semi-formelles. Nous présenterons donc dans le chapitre suivant quelques méthodes de modélisation semi-formelles de règles de contrôle d'accès.

Chapitre 3

Modélisation semi-formelle de politiques de sécurité

Notre objectif dans ce mémoire est de modéliser des systèmes de contrôle d'accès hybrides sûrs par construction. Ces sytèmes hybrides sont basés sur des modèles de contrôle d'accès hybrides. Dans le chapitre 2 nous avions présenté quelques modèles de contrôle d'accès qui ont été étudiés dans la littérature. Dans le présent chapitre, nous allons discuter de quelques méthodes semi-formelles qui permettent de modéliser des politiques de sécurité basées sur ces modèles de contrôle d'accès et sur les modèles de contrôle d'accès hybrides.

3.1 Introduction

Afin de présenter des politiques de sécurité dans des cadres bien définis, plusieurs méthodes de modélisations semi-formelles de règles de contrôle d'accès ont été proposées. Ces méthodes ont essentiellement pour objectif d'aider les intervenants dans des projets à comprendre les grandes lignes des exigences de ces projets. Elles facilitent également l'implémentation des systèmes. Ainsi les méthodes que nous présentons dans ce chapitre, permettent d'exprimer à l'aide de langages semi-formels comme UML (*Unified Modeling Language*), des politiques de sécurité de façon à respecter les exigences des

modèles de contrôle d'accès.

Au chapitre 1 nous avions montré comment des technologies comme le *Cloud Computing* ou encore des fusions d'entreprises pouvaient obliger les ingénieurs à modéliser des systèmes basés sur des modèles de contrôle d'accès hybrides. Nous montrons dans ce chapitre comment des langages semi-formels nous permettent d'exprimer ces modèles de contrôle d'accès hybrides. Nous mettrons particulièrement l'accent sur la méthode CatBAC.

3.2 Modélisation de règles de contrôle d'accès

3.2.1 SecureUML

SecureUML [7] est un langage de modélisation graphique, qui permet de spécifier sous forme d'un diagramme de classes UML, les informations relatives à la sécurité d'un système. Son méta-modèle (**syntaxe abstraite**) est une extension du méta-modèle du diagramme de classe d'UML.

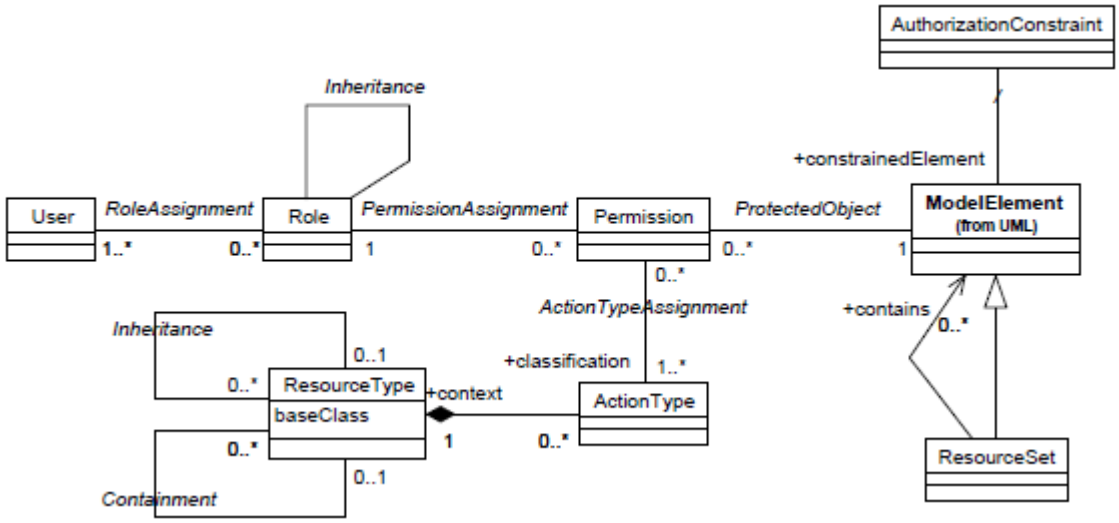

FIGURE 3.1 – Méta-modèle de SecureUML (extrait de [7])

Pour modéliser des informations de sécurité, SecureUML intègre les éléments de base du modèle RBAC. La Figure 3.1 présente le méta-modèle du langage. Dans cette figure, les éléments *User*, *Role* et *Permission* sont introduits pour les éléments de mêmes noms dans le modèle RBAC. L'héritage de l'élément *ResourceSet* indique que des classes du modèle UML peuvent jouer le rôle de ressource c'est à dire un objet.

L'élément *Permission* désigne un ensemble de permissions et est représenté dans un modèle par une classe d'association entre des éléments *Role* et *ModelElement* (une classe UML). Pour chaque permission, il peut exister plusieurs classes d'opérations de type *ActionType*. Chaque élément *ActionType* représente une action à exécuter sur une ressource. *ResourceType* indique au besoin l'ensemble des éléments de type *ActionType* défini pour un modèle particulier. Enfin *AuthorizationConstraint* permet d'exprimer des contraintes d'accès dans un modèle.

Supposons la règles de contrôle d'accès suivante : «*un individu ayant le rôle étudiant peut s'inscrire à un cours s'il en obtient l'autorisation*». Nous nommons cette règle R1 dans le reste de notre document. R1 indique que dans le système à modéliser nous devons identifier toutes les personnes qui ont le rôle *Etudiant* et nous assurer qu'elles ont bien l'autorisation de s'inscrire avant de leur permettre d'exécuter l'action *Inscrire* qui conduit à leur inscription effective à un cours. Pour cet exemple, nous supposerons qu'il existe une unique autorisation *inscrireCours* qui permet de s'inscrire à un cours.

Un modèle SecureUML de la règle R1 est présenté dans la Figure 3.2. Dans cette figure, la classe *Personne* correspond à l'élément *User* du méta-modèle de SecureUML. Cette classe désigne un ensemble de personnes. Lorsqu'une personne est enregistrée dans le système comme étudiant, on lui associe le rôle *Etudiant*. La classe *Cours* désigne l'ensemble des cours auxquels les étudiants peuvent s'inscrire. Il s'agit ici d'une classe de type *ResourceSet*.

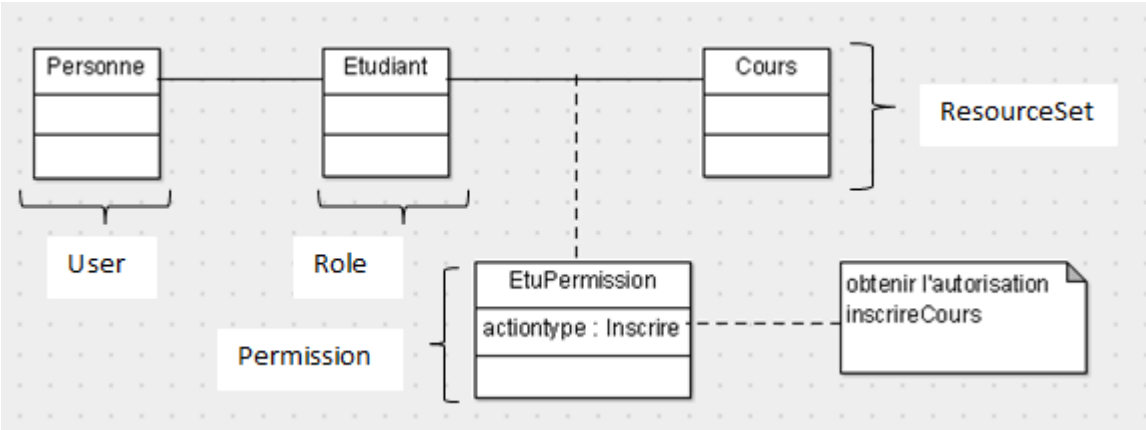

FIGURE 3.2 – Modélisation SecureUML de R1

Pour exprimer l'action *Inscrire*, une classe d'association *EtuPermission* (représentant les permissions (Figure 3.1)) relie le rôle *Etudiant* et la ressource *Cours*. Cette classe contient un élément *ActionType* qui est *Inscrire*. L'expression de l'autorisation *inscrireCours* qui donne le droit d'inscription à un cours est indiquée ici sous forme de commentaire. Dans un diagramme SecureUML complet, il s'agirait de contraintes OCL (Object Constraint Language). D'autres exemples de modélisation avec SecureUML sont données dans [7, 8].

SecureUML est approprié pour la modélisation de règles construites suivant le modèle RBAC. Il apparait que la modélisation d'autres types de règles, dans lesquelles on noterait l'absence de l'élément Role ne serait pas appropriée. Le langage UACML va pallier à cette faiblesse.

3.2.2 UACML : Unified Access Control Modeling Language

UACML [9] est un méta-modèle de contrôle d'accès basé sur UML et conçu pour modéliser des politiques de sécurité pouvant être hybrides. Contrairement à SecureUML qui est basé sur le modèle RBAC, UACML est indépendant d'un modèle de contrôle d'accès particulier. Cette neutralité lui permet de supporter des règles de contrôle d'accès basées sur plusieurs modèles (RBAC, MAC, etc.).

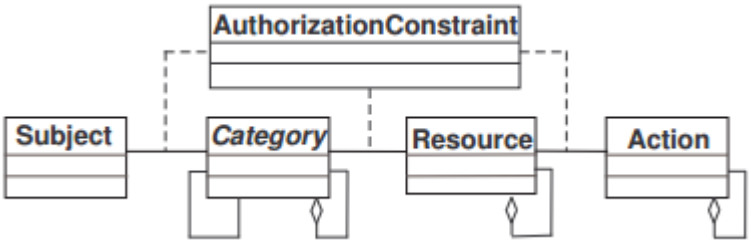

FIGURE 3.3 – Méta-modèle de UACML (extrait de [9])

Le méta-modèle UACML est présenté à la Figure 3.3. L'élément *Subject* représente des sujets ; *Resource* correspond aux objets. *Action* représente les actions qu'un sujet peut exécuter sur une ressource. *Category* est l'élément central du méta-modèle UACML. Il permet de spécifier des concepts propres à chaque modèle de contrôle d'accès. Par exemple, on peut instancier l'élément *Category* vers les éléments rôles pour RBAC, vers des éléments niveaux de sécurité pour MAC, etc. Enfin *AuthorizationConstraint* permet de spécifier des contraintes sur les différentes associations présentes entre les éléments d'un modèle UACML. La Figure 3.4, extraite de [9], montre des exemples de modèles UACML. Nous remarquons au niveau (d) de cette figure un modèle hybride contenant un groupe, un rôle, un niveau de sécurité. Les niveaux (a, b et c) présentent respectivement un modèle basé sur les groupes, un modèle basé sur les niveaux de sécurité, et le modèle RBAC.

Un modèle UACML pour la règle R1 est présenté à la Figure 3.5. Comme pour la Figure 3.2 nous avons mis à la place de contraintes OCL un commentaire qui indique à quelle condition l'opération Inscrire peut être exécutée sur la ressource *Cours*. La classe *Etudiant* qui désigne un rôle, provient de l'élément *Category*.

UACML fournit donc assez de généricité pour contenir la plupart des modèles de contrôle d'accès et pour construire des modèles hybrides. Toutefois son méta-modèle impose la notion de Category dont l'ensemble des instances doit toujours relier un sujet à une ressource. On peut donc difficilement créer un modèle dans lequel des instances de *Category* seraient

destinées à être directement reliées à l'élément *Action*. Par exemple il sera difficile de modéliser des groupes d'actions dans un modèle UACML.

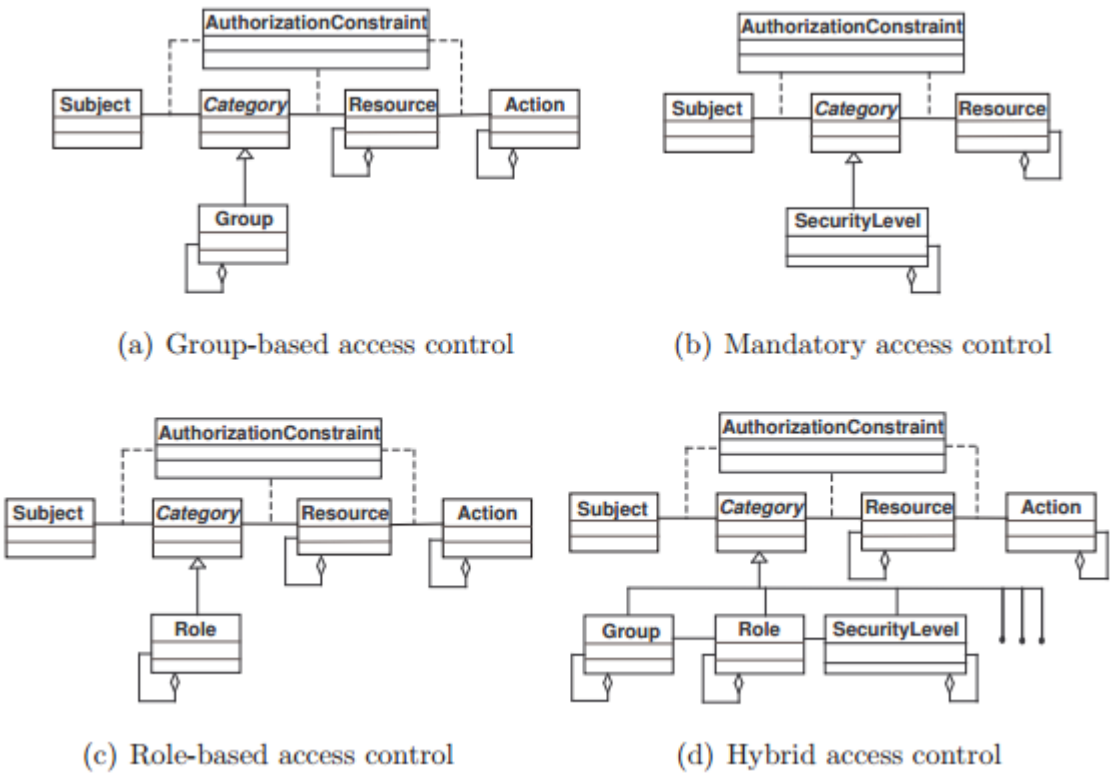

FIGURE 3.4 – Exemple de modèles UACML

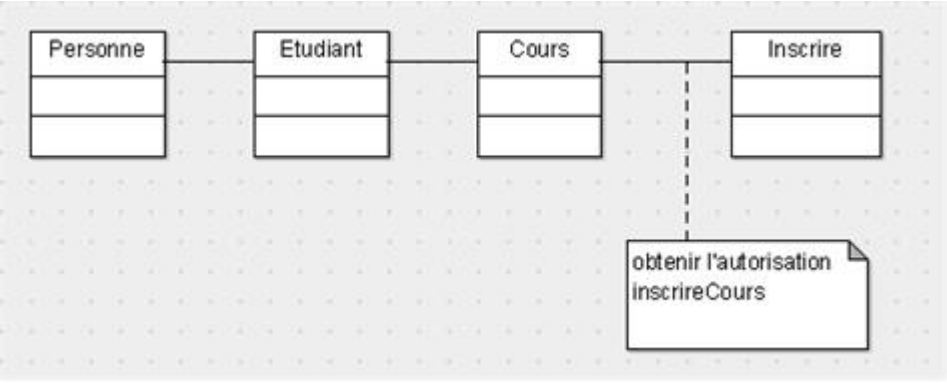

FIGURE 3.5 – Modélisation UACML de R1

3.2.3 CatBAC : Category Based Access Control

CatBAC [24] est un méta-modèle introduit pour la modélisation de politiques de sécurité hybrides. Ce méta-modèle utilise une notion de catégorie différente de celle d'UACML. Nous utiliserons cette méthode pour nos travaux.

La figure 3.6 présente le méta-modèle CatBAC. Ce méta-modèle constitue également le modèle le plus abstrait de la méthode. Nous utilisons le terme méta-modèle pour marquer cette différence. On distingue quatre composantes dans le méta-modèle CatBAC qui sont *SCat* (catégorie des sujets), *ACat* (catégorie des actions), *RCat* (Catégorie des ressources), *CCat* (catégorie des contextes).

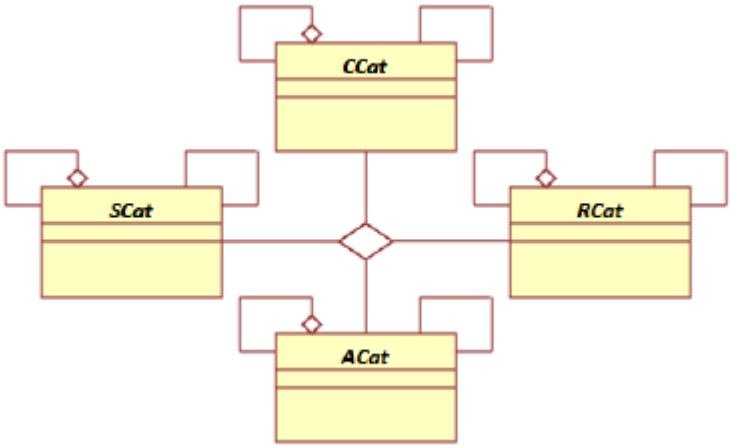

FIGURE 3.6 – Méta-modèle CatBAC (extrait de [24])

Un contexte dans CatBAC correspond à une forme de contrainte. Par exemple si une règle de contrôle d'accès impose une opération sur un fichier dans un intervalle de temps donné, alors on peut considérer que le contexte d'exécution de cette règle est l'intervalle de temps en question.

Les différentes catégories *SCat*, *ACat*, *RCat* et *CCat* sont reliées par une association de type *n-aire* (définie dans UML). La symétrie dans cette association permet de choisir le sens de lecture du modèle suivant l'objectif

recherché. On peut donc avoir des modèles formant le tuple (*SCat*, *ACat*, *RCat*, *CCat*), pour exprimer qu'une catégorie de sujet est reliée à une catégorie d'action et à une catégorie de ressource, dans une catégorie de contexte donné. On peut aussi avoir le tuple (*CCat*, *SCat*, *RCat*, *ACat*), pour exprimer le fait que dans une catégorie de contexte donnée, une catégorie de sujet est reliée à une catégorie de ressource et à une catégorie d'action. Ces formulations peuvent sembler identiques mais ne le sont pas, car formellement, les ensembles dont dérive respectivement chacun des tuples précédents sont différents. Il s'agit de l'ensemble $SCat \times ACat \times RCat \times CCat$ pour le tuple (*SCat*, *ACat*, *RCat*, *CCat*), et de l'ensemble $CCat \times SCat \times RCat \times ACat$ pour le tuple (*CCat*, *SCat*, *RCat*, *ACat*). Nous y reviendrons dans les sections suivantes.

Dans la méthode CatBAC, une catégorie correspond à un élément qu'on peut utiliser pour regrouper une ou plusieurs composantes de sécurité. Par exemple, la catégorie des sujets *SCat* correspond à tous les éléments possibles d'un modèle de contrôle d'accès par lesquels on peut regrouper des sujets. Dans RBAC par exemple, on regroupe les sujets par rôles ; dans MAC, on les regroupe par niveaux de sécurité, etc. La même définition est appliquée pour les autres catégories *ACat*, *RCat* et *CCat*. Par exemple, dans MAC on regroupe les ressources (*RCat*) par niveaux de sécurité. La relation d'agrégation qui se termine par un petit losange (Figure 3.6) et qui est présente sur chaque catégorie permet de spécifier à quelle catégorie une nouvelle composante appartient. Elle nous permet aussi d'établir au besoin, l'hiérarchie qui peut exister pour une catégorie donnée. La relation réflexive sur chaque catégorie indique qu'une catégorie peut être reliée à une autre.

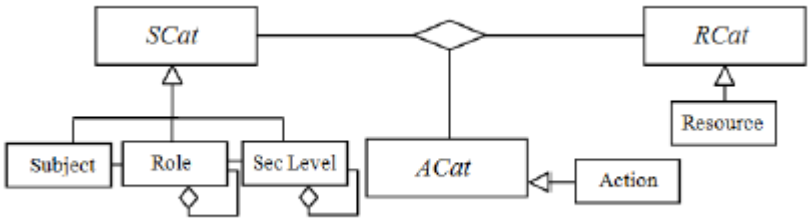

FIGURE 3.7 – Exemple de construction de modèle (extrait de [24])

La figure 3.7 montre un exemple de construction de modèle de contrôle d'accès avec CatBAC. Dans ce modèle, les sujets sont regroupés par rôles (Role) et par niveaux de sécurité (SecLevel). Il n'y a pas de regroupement spécifique sur les actions et les ressources.

Le raffinement dans CatBAC

Raffiner un modèle CatBAC consiste à lui ajouter des détails. Il s'agit d'une technique qui permet de dériver le méta-modèle (qui peut être vu comme un modèle abstrait) vers une succession de modèles avec à chaque fois l'ajout de nouvelles informations. Cette manière de modéliser est souhaitable pour des systèmes complexes. Les objectifs généraux du raffinement dans CatBAC sont :

- déterminer les ressources nécessaires pour satisfaire le modèle d'une politique abstraite ; il s'agit de déterminer quelle catégorie intervient dans la politique (par exemple pour *SCat*, on peut avoir besoin de sujets, rôles, groupes etc.). Une politique abstraite pourrait être : *des sujets accèdent aux permissions à partir de leurs rôles.* Une instance de cette politique abstrait donnerait la politique concrète suivante : *Bob accède à la permission P78 à partir de son rôle de consultant.*
- instancier le modèle d'une politique abstraite vers des modèles représentant des règles concrètes : par exemple *SCat* = {sujets, rôles} ; rôle = consultant etc.
- vérifier que le modèle d'une politique concrète respecte son équivalent abstrait.

Dans la Figure 3.8 un exemple de raffinement de modèle est présenté. Le modèle initial (modèle abstrait) de cette figure correspond au méta-modèle CatBAC. Le second modèle (modèle raffiné ou concret) est relié au modèle abstrait par une relation *refine*. Dans ce modèle concret, la catégorie des sujets *SCat* est raffinée en deux classes *Subject* et *Role.Radialogist*. La catégorie de ressources *RCat* est raffinée en une classe niveau de sécurité *Sec_ level.Private*, une classe groupe *Group.EMR* et une classe ressource *Ressource*. La catégorie des contextes est raffinée en deux classes *Location* (un

contexte de lieu) et *Time* (un contexte de temps).

La Figrue 3.9 montre un nouveau raffinement du modèle concret de la Figure 3.8. Dans le nouveau modèle concret, des nouveaux détails sont ajoutés au contexte de temps pour indiquer qu'une connexion est active uniquement entre 8 et 17 heures.

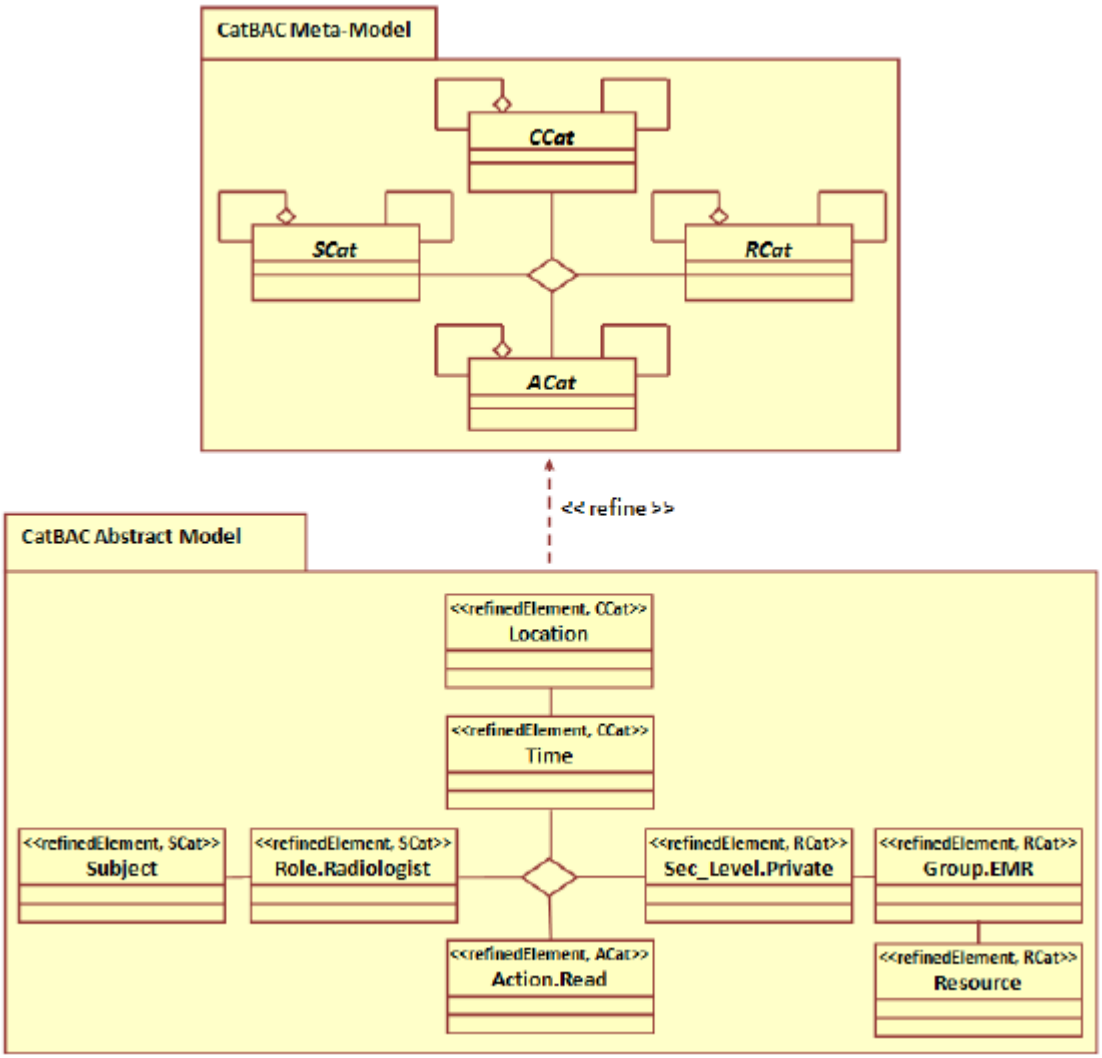

FIGURE 3.8 – Premier niveau de raffinement d'un modèle (extrait de [24])

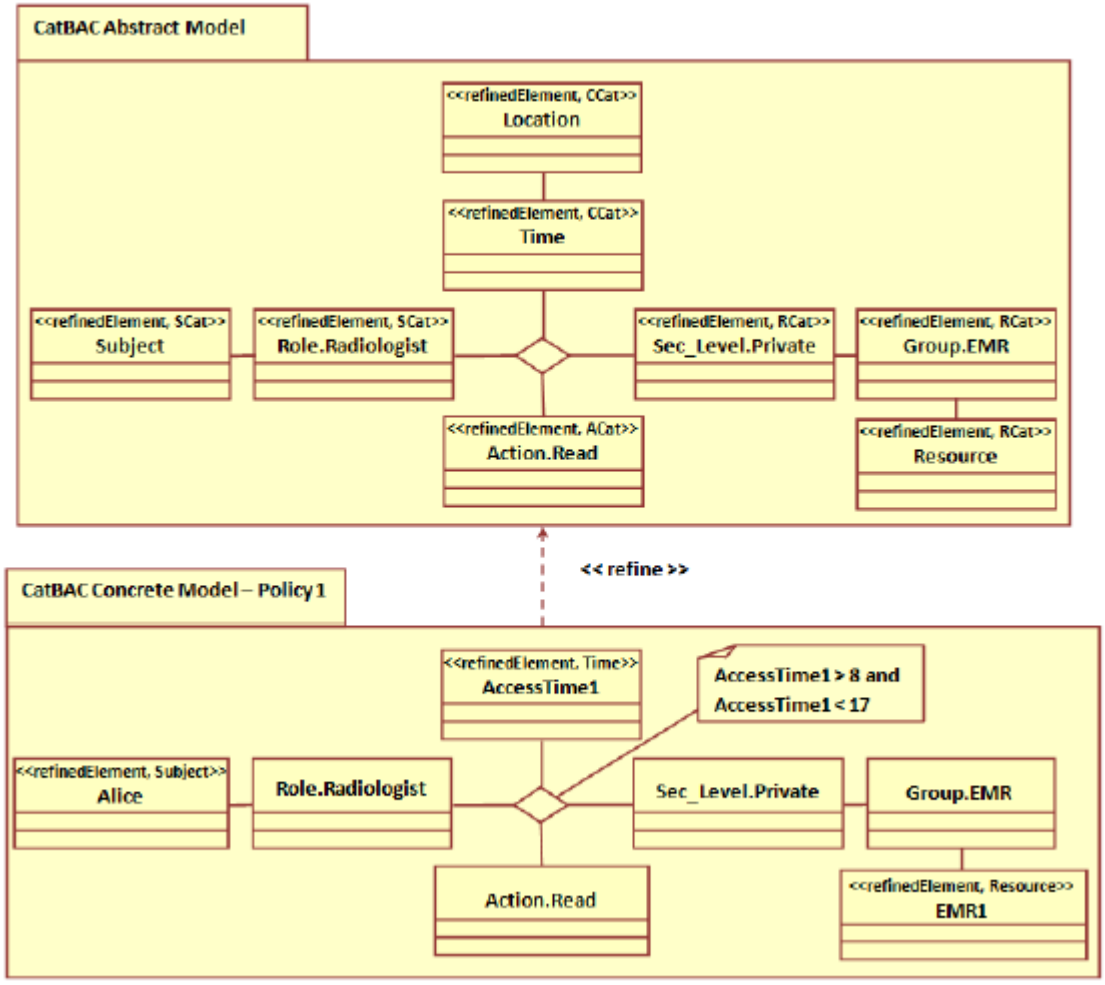

FIGURE 3.9 – Second niveau de raffinement d'un modèle (extrait de [24])

Pour raffiner des modèles CatBAC il faut procéder comme suit :

1. Extraire les concepts utilisés dans la structure de l'entreprise (Consultant, administrateur, privé, etc.)

2. Définir pour chaque concept, la catégorie à laquelle il fait référence (consultant et administrateur sont des rôles (Role), privé est un niveau

de sécurité (Security Level))

3. Déterminer l'agencement des catégories identifiées précédemment (les sujets (Subjects) sont regroupés par rôles, les ressources (Resources) par niveaux de sécurité)

4. Construire l'ensemble des catégories H-Ref tel que :
   H-Ref : Basic Entity $\longrightarrow$ Categories.
   H-Ref(Subjects) = {Role}
   H-Ref(Resources) = {Security Level}
   A ce niveau, l'ensemble H-Ref contient les catégories qui seront utilisées pour regrouper les sujets, les objets, les actions et les contextes.

5. Ajouter à chaque ensemble H-Ref(X) la composante à regrouper (X)
   H-Ref(X) = H-Ref(X) $\cup$ {X}
   H-Ref(Subjects) = {Role, Subjects}
   H-Ref(Resources) = {Security Level, Resources}
   Ici, on complète H-Ref avec la catégorie à regrouper. Par exemple, H-Ref(Subjects) = {Role, Subjects} signifie que les catégories qui seront utilisées pour modéliser la structure d'accès des sujets sont Subjects et Role.

6. Spécialiser les super-classes iCat vers l'ensemble des classes correspondant aux éléments de H-Ref(X).
   SCat devient (Role, Subjects)

Appliquons cette technique à la règle R1 :« *un individu ayant le rôle étudiant peut s'inscrire à un cours s'il en obtient l'autorisation* » :

1. Le concept de regroupement utilisé pour notre système est *étudiant*. Nous le réécrivons Etudiant

2. Etudiant est un rôle (Roles)

3. Les sujets sont regroupés par rôles (*un individu ayant le rôle étudiant...*)

4. H-Ref(Subjects) = {Roles} (les sujets sont regroupés par rôles)
   H-Ref(Resources) = {} (pas de regroupement pour les ressources)
   H-Ref(Action) = {} (pas de regroupement d'actions)
   Nous n'aurons pas besoin de la catégorie des contextes

5. Nous ajoutons à H-Ref la catégorie à regrouper
   H-Ref(Subjects) = {Roles, Subjects} : la structure d'accès qui modélise les sujets contient les éléments Subjects et Roles
   H-Ref(Resources) = {Resources} : la structure d'accès qui modélise les ressources ne contient que l'élément Resources
   H-Ref(Actions) = {Actions}

6. Nous spécialisons SCat = {Roles, Subjects}
   RCat = {Resources}
   ACat = {Actions}

Les Figures 3.10 et 3.11 montrent les modèles correspondants à ce raffinement. La Figure 3.10 correspond au premier niveau de raffinement. Nous obtenons un modèle à partir duquel la règle R1 sera formalisée. Une instance de ce modèle (Figure 3.11) nous donne cette règle. Nous pouvons également y ajouter une contrainte OCL pour indiquer à quelle condition un étudiant peut s'inscrire à un cours.

La technique de raffinement présentée pour CatBAC peut être améliorée. En effet la démarche de construction des modèles doit préserver l'association n-aire. Comme indiqué précédemment, cette association peut être interprétée de plusieurs manières. Il faut donc éviter que son interprétation ne change d'un raffinement à l'autre.

Par exemple, pour le premier raffinement de R1 (Figure 3.10) on peut considérer que le modèle raffiné nous indique qu'un sujet est relié à un

élément (rôle, action, ressource). Ainsi l'association n-aire entre les classes *Roles*, *Actions* et *Resources*, permettrait de créer cet élément d'une part. D'autre part l'association entre les classes *Subjetcs* et *Roles* permettrait de relier l'élément (rôle, action, ressource) à un sujet.

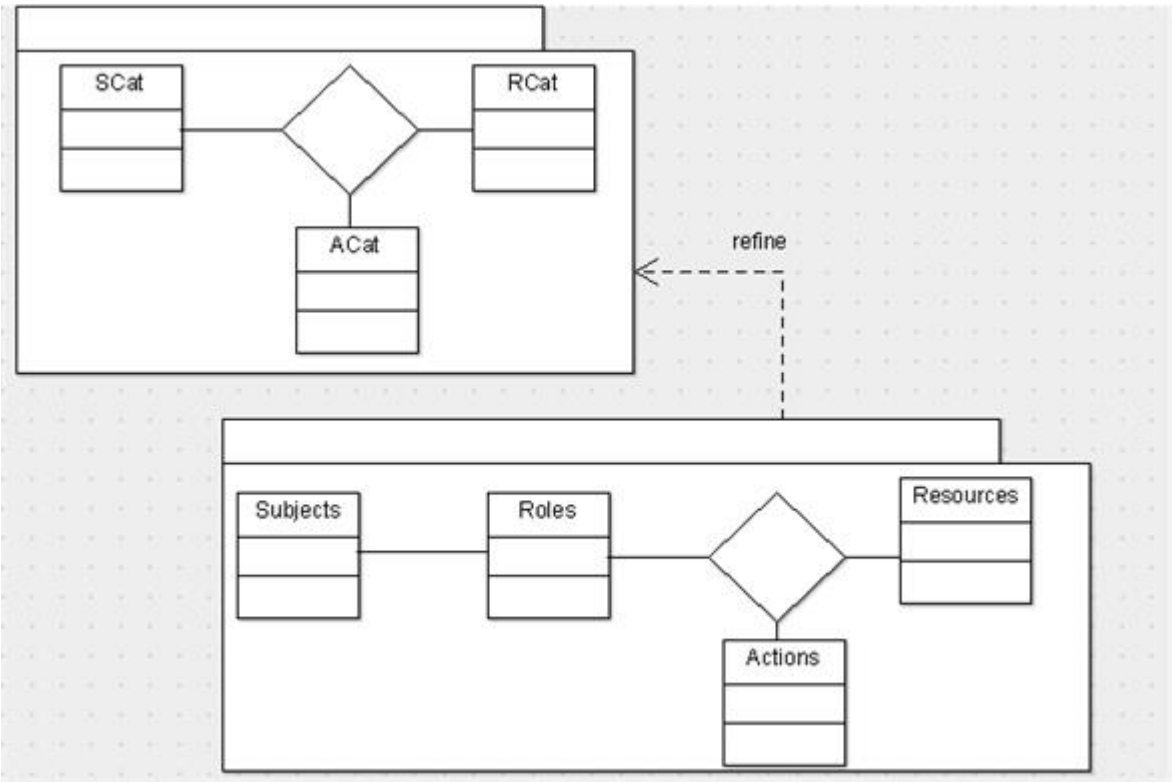

FIGURE 3.10 – Modélisation CatBAC de R1 : premier niveau

On peut également considérer pour le même modèle de la Figure 3.10, qu'un sujet est associé à un rôle dans un premier temps, puis dans un second temps que le rôle est associé à travers l'association n-aire, à un couple (action, ressource). Nous pouvons ainsi arriver à plusieurs possibilités d'interprétations. Cela entraînerait la production de modèles formels différents pour une même succession de raffinement et rendrait les vérifications des modèles difficiles.

Une autre remarque que nous pouvons faire est que le raffinement tel-qu'il est proposé ne nous indique pas le comportement de chaque élément (catégorie et association) au moment du passage d'un modèle abstrait vers un modèle plus concret.

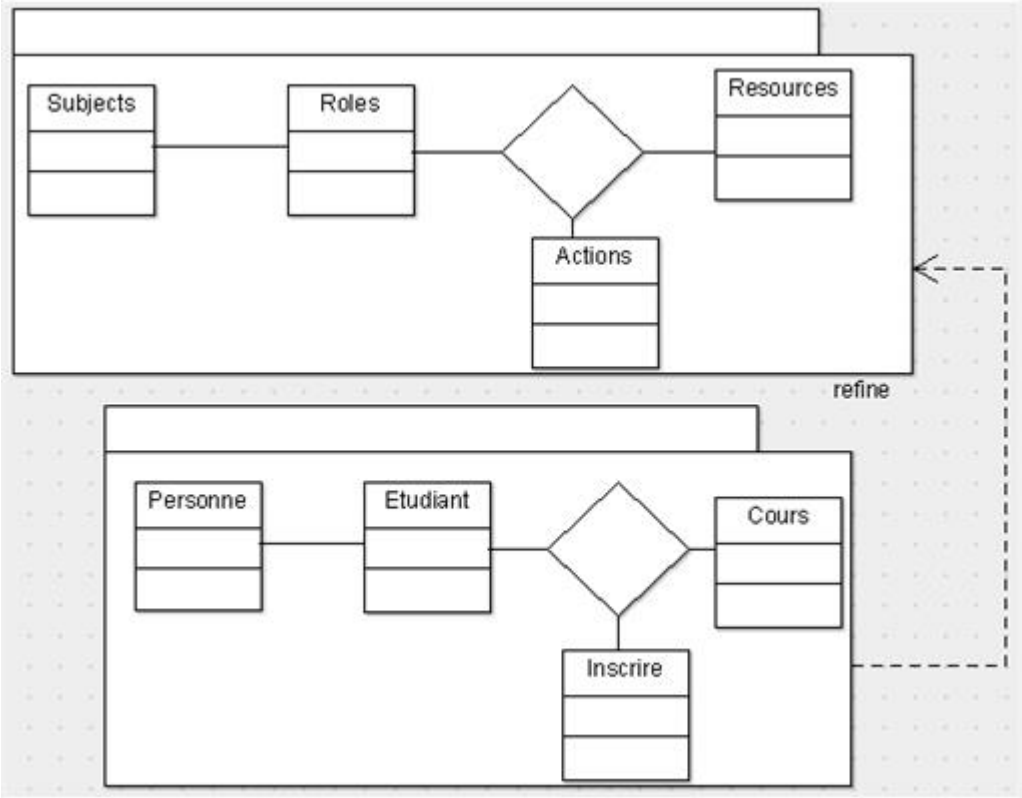

FIGURE 3.11 – Modélisation CatBAC de R1 : second niveau

Par exemple dans la Figure 3.10, nous savons que *SCat* est raffiné en *Subjects* et *Roles*. D'après le méta-modèle CatBAC (Figure 3.6), cela signifie que *Subjects* et *Roles* sont contenus par aggrégation dans *SCat*. Par conséquent si une association avait pour origine *SCat*, elle ne devrait plus avoir la même signification si sa nouvelle origine est *Subjects* ou *Roles*. Dans notre exemple, si on considère que l'association n-aire a pour origine *SCat*, cette

assocition n'a plus la même signification si sa nouvelle origine est *Roles*. Au chapitre 5, nous allons définir des règles qui nous aideront à contrôler le raffinement. Ces règles nous permettrons par exemple d'affirmer pour la Figure 3.10 que l'association n-aire ayant pour origine *SCat* et l'association n-aire ayant pour origine *Roles* jouent la même fonction, mais à des niveaux d'abstraction différents. Ces règles seront utiles, car elles nous aideront à produire les expressions formelles, à partir desquelles nous traduirons les modèles CatBAC vers B-événementiel afin de les vérifier.

Si nous considérons le modèle de la figure 3.12, les composants minimaux qu'on peut avoir dans un contrôle d'accès sont : les sujets, les actions, les ressources et les contextes. Lorsque les contextes sont absents, on suppose qu'ils sont associés à une contrainte toujours vraie.

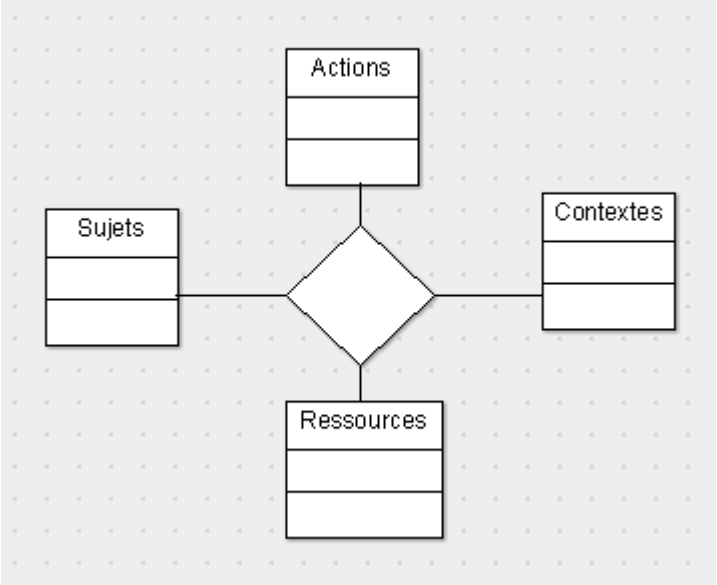

FIGURE 3.12 – Modèle abstrait

Ce modèle relie à travers une association n-aire un sujet, une action, une ressource et un contexte. Dans chaque modèle qu'on raffinera avec CatBAC on devrait être en mesure de conserver cette notion.

De façon générale, les modélisations de la règle R1 que nous avons présentées dans ce chapitre ne montrent pas les modélisations des opérations qui contrôlent la mise en application de R1. Il s'agit par exemple de modéliser explicitement les conditions qui conduisent à l'attribution de la permission *inscrireCours* à un étudiant ou encore les conditions qui doivent précéder l'exécution effective de l'opération *Inscrire*. Vu qu'il est important de faire attention à cela en sécurité informatique, nous ajouterons aux modèles CatBAC traduits en B-événementiel les opérations de ce type.

3.3 Conclusion

Dans ce chapitre nous avons présenté des méthodes semi-formelles de modélisation de politiques de sécurité. Nous avons porté notre attention sur les modèles basés sur des diagrammes de classes UML et sur les modèles CatBAC en particulier, car CatBAC est suffisamment générique pour modéliser un nombre important de modèles de contrôle d'accès. Cependant il existe d'autres méthodes de modélisation qui ne sont pas basées sur les diagrammes de classes UML. Dans [10, 8] une méthode appelée ASTD est présentée. Elle combine des notations sous forme d'automates et un algèbre de processus associé à une méthode appelé EB^3 (*Entity-Based Black Box*) [11]. ASTD donne ainsi la possibilité de présenter les règles de contrôle d'accès dynamiques. Dans ce mémoire, notre travail consiste à créer des modèles de contrôle d'accès hybrides sûrs par construction. Avec les enjeux que représente la sécurité de l'information, une modélisation semi-formelle avec CatBAC ne serait être suffisante pour exprimer adéquatement une politique dans son ensemble. Nous avons donc choisi de compléter les modélisations CatBAC avec la méthode formelle B-événementiel. Nous présentons cette méthode au prochain chapitre.

Chapitre 4

La méthode B événementielle

L'objectif de notre mémoire est de modéliser des sytèmes de contrôle d'accès sûrs par construction. Pour y arriver nous allons associer à CatBAC, une méthode de modélisation formelle appelée B-événementiel (*Event-B*). Dans ce chapitre nous présentons cette méthode avec ses concepts généraux. Nous utilisons cette méthode pour compléter et vérifier les modèles CatBAC.

4.1 Introduction

La méthode B [1,3] à été introduite par Jean-Raymond Abrial dans les années 80. Initialement elle fut conçue pour le développement des logiciels. Elle sera peu à peu étendue pour la modélisation des systèmes comprenant à la fois des logiciels et d'autres composants tels que des humains.

B est une méthode formelle associée à des mécanismes de preuves. Ces preuves permettent de s'assurer que le logiciel en cours de développement est sûr par construction. Ainsi, modéliser un logiciel avec B consiste d'abord à définir les propriétés que ce dernier doit vérifier et ensuite **prouver** que les opérations qui seront définies dans le modèle du logiciel préserveront ces propriétés.

Un concept qui constitue une des richesses de la méthode B est le raffinement. Tout comme pour CatBAC, cela consite à faire une modélisation en

créant des successions de modèles. Dans B, les nouveaux modèles construits doivent être en mesure de conserver les propriétés des modèles précédents et d'inclure de nouvelles propriétés de plus en plus proches des objectifs finaux de développement. B n'est pas une méthode qui doit forcément conduire tout le processus de développement d'un logiciel. Il peut seulement intervenir à une étape précise du développement. Plusieurs travaux ont d'ailleur porté sur la combinaison de B et UML (Chapitre 5). Pour notre mémoire nous allons nous intéresser particulièrement à la méthode B-événementiel qui est une extension de la méthode B.

La méthode B événementielle [2] a également été introduite par Abrial. Cette méthode prend en compte la modélisation de systèmes dynamiques pris dans leurs ensembles. Ces systèmes peuvent comprendre plusieurs composantes comme des logiciels, des composantes matérielles ou humaines, les interactions (communications) entre différents composants etc. Dans [3, 5] les auteurs mentionnent les principales différences entre B et B-événementiel. Dans les sections qui suivent nous donnons une description des différents concepts qui interviennent dans B-événementiel.

4.2 Les éléments de spécification dans B-événementiel

B-événementiel utilise des composants appelés machines et contextes pour spécifier formellement des systèmes. Les machines représentent la partie dynamique de la spécification du système à modéliser. Elles regroupent les propriétés comportementales d'un système. Les contextes contiennent les éléments statiques de la spécification d'un système. Une machine peut être raffinée par une autre machine et un contexte peut être étendu par un autre contexte. le raffinement entre machines est traduit par une relation *refines* ; l'extention entre contextes, par une relation *extends*. Une machine est reliée à son contexte par la relation *sees*. Lorsqu'une machine raffine une autre, elle «voit» implicitement le contexte de la machine raffinée. La Figure 4.1 présente les relations entre machines et contextes.

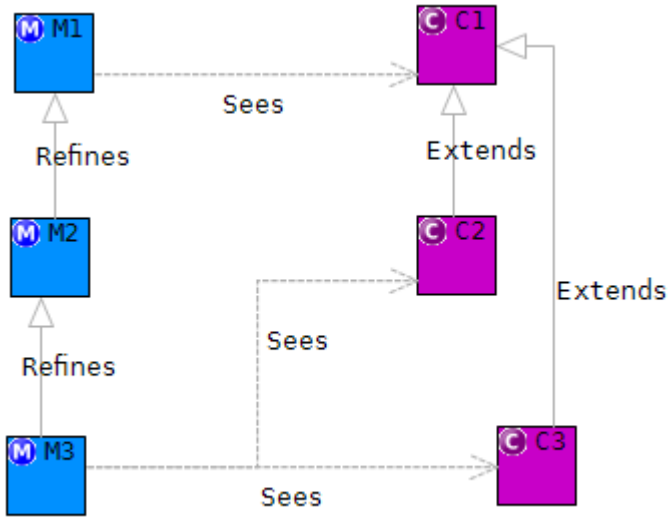

FIGURE 4.1 – Relation entre machine et contextes

Dans cette figure, les éléments (Mi) représentent des machines et les éléments (Ci), des contextes. La machine M1 «voit» le contexte C1. La machine M3 «raffine» la machine M2 qui «raffine» à son tour la machine M1. Les contextes C2 et C3 «étendent» le contexte C1. La machine M2 «voit» indirectement le contexte C1 et la machine M3 «voit» les contextes C2 et C3. Si on considère les machines M1 et M2, M1 est une machine abstraite et M2 une machine concrète ou raffinée.

4.2.1 Les contextes

Un *contexte* dans B-événementiel correspond à la partie statique de la spécification d'un système. Il contient des ensembles, des constantes et des propriétés. Dans le chapitre précédent (Chapitre 3), nous avons défini la catégorie des contextes de CatBAC. Un contexte dans CatBAC est une forme de condition liée à l'application d'un contrôle d'accès. Dans B-événementiel il s'agit d'un élément qui englobe la spécification statique d'un système.

Dans la spécification d'un contexte, le nom de ce dernier est mentionné sous la clause CONTEXT et les noms des contextes étendus, sous la clause

EXTENDS. Les ensembles définis dans les contextes sont regroupés sous la clause SETS. Ils constituent les types de base de la spécification. Les constantes sont regroupées dans la clause CONSTANTS. Certaines propriétés définies sur ces constantes, notamment l'expression de leur type, et d'éventuels axiomes sont définis sous la clause AXIOMS. Enfin les théorèmes sont regroupés sous la clause THEOREMS. Ils peuvent être formés à partir de propriétés spécifiées dans les axiomes. Les théorèmes doivent être prouvés alors que les axiomes sont considérés comme vrais. La Figure 4.2 présente la structure d'un contexte dans B-événementiel.

Exemple

Prenons la règle R1 du Chapitre 3 : «*Un individu ayant le rôle étudiant peut s'inscrire à un cours s'il en obtient l'autorisation*». Cette règle indique que dans le système à modéliser nous devons identifier toutes les personnes qui auraient le rôle *etudiant* et nous assurer qu'elles ont bien l'autorisation de s'inscrire avant de leur permettre d'exécuter l'action *Inscrire* qui conduit à leur inscription éffective à un cours. Un modèle possible de contexte pour cette règle est comme suit :

CONTEXT Ecole
SETS

$Personnes$

$Roles$

$Permissions$

$Cours$
CONSTANTS

$Privileges$
AXIOMS

axm1 : $Privileges \in \mathbb{P}(Personnes \times Roles \times Permissions)$
END

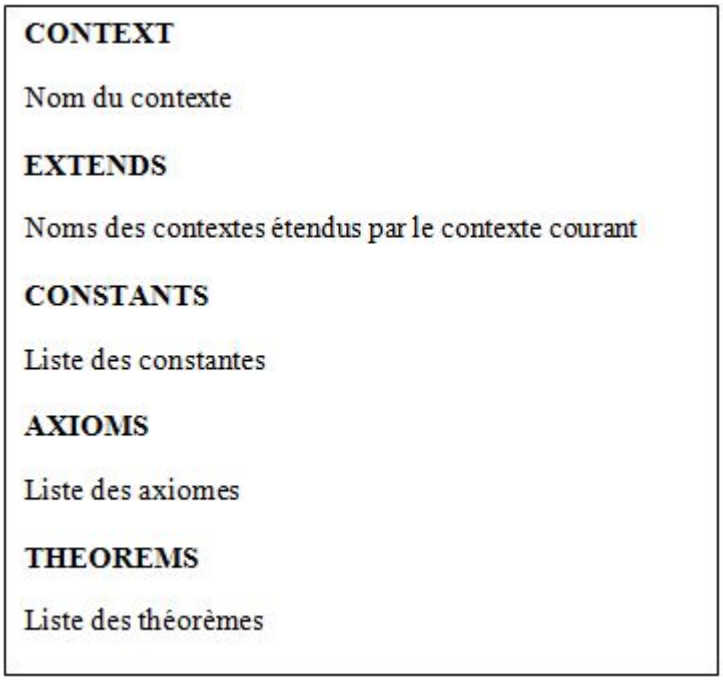

FIGURE 4.2 – Structure d'un contexte B évènementiel

Sous la clause SETS, *Personnes* désigne un ensemble de personnes. Elles peuvent etre étudiantes, enseignantes ou autres. *Roles* correspond à l'ensemble des rôles qui seront enrégistrés dans le système. *Permissions* comme son nom l'indique est un ensemble de permissions telles que *s'inscrire, suivre un cours,* etc. *Cours* correspond à l'ensemble des cours à dispenser. Sous la clause CONSTANTS, *Privileges* désigne un ensemble de privilèges construit à partir de tuples de la forme (*p, r, perm*), correspondant à un élément d'une partie du produit cartésien entre les ensembles *Personnes*, *Roles* et *Permissions*. Cette construction de l'ensemble *Privileges* est indiquée sous la clause AXIOMS.

Lorsque nous procédons à l'extension du contexte précedent pour y ajouter le rôle *etudiant* et la permission *inscrireCours*, un nouveau modèle Ecole1 peut être alors créé comme suit :

CONTEXT Ecole1
EXTENDS Ecole
CONSTANTS

$etudiant$

$inscrireCours$

AXIOMS

axm1 : $etudiant \in Roles$

axm2 : $inscrireCours \in Permissions$

END

Ecole1 conserve toutes les propriétés de Ecole. Sous la clause AXIOMS, nous ajoutons deux nouvelles propriétés qui indiquent l'ensemble d'appartenance de chaque nouvelle constante définie. Ces propriétés doivent être toujours vraies.

4.2.2 Les machines

Une machine représente la partie dynamique d'une spécification B événementielle. Elle regroupe des variables et les propriétés qui y sont associées. Les variables dans B-événementiel sont appelées variables d'états. Elles sont regroupées sous une clause appelée VARIABLES. Le changement de valeurs d'une variable traduit l'état dans lequel le système se trouve. Une clause INVARIANTS contient les invariants du système. Les invariants permettent de spécifier les propriétés que le système doit toujours respecter quelque soit son état. On peut donc y définir le type des variables (entier naturel, ensemble de personnes, etc.). La clause THEOREMS contient des propriétés à prouver. Les événements (Section 4.4) sont regroupés sous la clause EVENTS. Ils permettent de spécifier l'évolution des variables d'états définies dans la spécification du système. Ils peuvent être compris comme des transitions d'un automate. Toutes les variables doivent être initialisées dans une machine B événementielle. L'opération d'initialisation est réalisée par un événement particulier appelé INITIALISATION. Aussi le nom de la machine courante

est défini sous l'entête MACHINE et les contextes associés à cette machine sont définis sous la clause SEES. La Figure 4.3 montre la structure d'une machine dans B-événementiel.

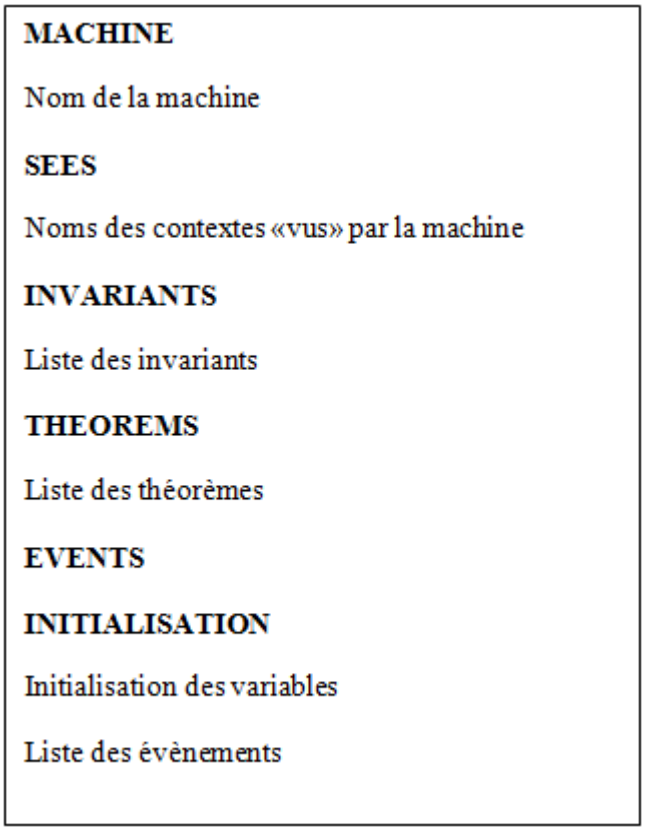

FIGURE 4.3 – Structure d'une machine B événementielle

Exemple

Complétons l'exemple précédent en considérant une variable *inscription* et un événement *Inscrire*. La variable *inscription* est un ensemble correspondant à une partie du produit cartésien de deux ensembles *Etudiant* et *cours*. Ainsi pour savoir si un étudiant *etu* est éffectivement inscrit à un cours *c*, il

suffit de vérifier dans la variable *inscription*, si il existe un élément correspondant au couple (etu, c). Cette caractéristique est mentionée dans l'invaraint 4 (inv4) de la machine Ecole_Machine définie ci-après. L'opérateur $\leftrightarrow$ appelé « Relation » (Section 4.3) désigne l'ensemble des sous-ensembles du produit cartésien des deux ensembles entre lesquels il est placé.

L'évenement *Inscrire* permet d'ajouter un nouvel élément à la variable *inscription*. Cela aura pour effet d'indiquer qu'un nouvel étudiant *etu*, s'est incrit à un cours c. Pour que l'événement *Inscrire* se déclenche, un certain nombre de conditions doivent être vérifiées. Ces conditions appelées *gardes* sont indiquées sous une clause propre aux évènements appelée WHERE (Nous reviendrons plus tard sur la structure des événements). Trois gardes interviennent pour l'événement *Inscrire*. La plus importante dans le cadre de notre exemple est de vérifier que l'étudiant qui s'inscrit à un cours a la permission *incrireCours*. La machine Ecole_Machine est comme suit :

MACHINE Ecole_Machine
SEES Ecole1
VARIABLES

$Etudiants$

$roles$

$cours$

$inscription$

INVARIANTS

inv1 : $Etudiants \in \mathbb{P}(Personnes)$

inv2 : $roles \in \mathbb{P}(Roles)$

inv3 : $cours \in \mathbb{P}(Cours)$

inv4 : $inscription \in Etudiant \leftrightarrow cours$

EVENTS
Initialisation

begin

act1 : $Etudiants := \varnothing$

act2 : $roles := \varnothing$

act3 : $cours := \varnothing$

act4 : $inscription := \varnothing$

end

Event *Inscrire* $\widehat{=}$ (Modélise l'inscription d'un étudiant à un cours)

any

etu

c

where

grd1 : $etu \in Etudiants$

grd2 : $c \in cours$

grd3 : $etu \mapsto etudiant \mapsto inscrireCours \in Privileges$

then

act1 : $inscription := inscription \cup \{etu \mapsto c\}$

end

END

Sous la clause SEES nous avons indiqué que la machine Ecole_Machine *voit* le contexte Ecole1. Trois variables *Etudiants*, *roles* et *cours* sont définies en plus de la variables *inscription*. La variable *Etudiants* représente l'ensemble des étudiants déclarés dans le système. Cet ensemble (*Etudiants*), est un sous-ensemble de l'ensemble des *Personnes* précédement défini dans le contexte Ecole. L'invariant 1 (inv1) de la clause INVARIANTS indique cette propriété. La variable *roles* représente l'ensemble des rôles effectivement créés dans le système. La variable *cours* correspond à l'ensemble des cours effectivement créés.

L'événement INITIALISATION affecte à chaque variable un ensembles vide. L'événement **Inscrire** contient un paramètre *etu* correspondant à l'étudiant qui s'incrit à un cours. Ce cours est désigné par un second paramètre *c*. Les deux paramètres sont déclarés sous la clause ANY. La clause THEN contient l'action à exécuter par l'événement Inscrire. Ici, on ajoute à la variable *inscription* un nouveau couple (etu, c) indiqué avec la notation $etu \mapsto c$. Notons que dans cet exemple nous ne modélisons pas les évènements qui modifient les autres variables.

4.3 Quelques notations du langage

B événementiel est associé à un langage mathématique riche qui permet de faire des spécifications intéressantes. Nous présentons quelques notations dans les tableau ci-après. Des informations complètes sur les notations et leurs significations peuvent être retrouvées dans [1, 2].

S, T et U représentent des ensembles ; r_i est un élément d'une relation (voir tableau). $x \mapsto y$ désigne un couple (x,y).

TABLE 4.1 – Quelques notations (1)

Nom	**Syntaxe**	**Définition**
Relation	$S \leftrightarrow T$	$\mathbb{P}(S \times T)$: Ensemble des parties de $S \times T$
Composition	$r_1 ; r_2$	$r_1 \in S \leftrightarrow T \wedge r_2 \in T \leftrightarrow U \Rightarrow r_1 ; r_2 \in S \leftrightarrow U$
Identité	id(S)	$\{x \mapsto x \mid x \in S\}$
Domaine	dom(r)	$\forall r.r \in S \leftrightarrow T$, dom(r) $= \{x \mid x \in S \wedge \exists y \in T.x \mapsto y \in r\}$
Codomaine	ran(r)	$\forall r.r \in S \leftrightarrow T$, ran(r) $= \{y \mid y \in T \wedge \exists x \in S.x \mapsto y \in r\}$
Inverse	r^{-1}	$\{x \mapsto y \mid y \mapsto x \in r\}$

Supposons $r \in S \leftrightarrow T$, $U \subseteq S$ et $V \subseteq T$.

TABLE 4.2 – Quelques notations (2)

Nom	**Syntaxe**	**Définition**
Restriction	$U \lhd r$	$\{x \mapsto y \mid x \mapsto y \in r \wedge x \in U\}$
Corestriction	$r \rhd V$	$\{x \mapsto y \mid x \mapsto y \in r \wedge y \in V\}$
Anti-restriction	$U \lhd\!\!\!\!- r$	$\{x \mapsto y \mid x \mapsto y \in r \wedge x \notin U\}$
Anti-corestriction	$r \rhd\!\!\!\!- V$	$\{x \mapsto y \mid x \mapsto y \in r \wedge y \notin V\}$

Supposons $r \in S \leftrightarrow T$ et $U \subseteq S$

TABLE 4.3 – Quelques notations (3)

Nom	**Syntaxe**	**Définition**
Image	$r[U]$	$\{y \mid y \in T \wedge \exists x \in U.x \mapsto y \in r\}$
Fonction partielle	$S \nrightarrow T$	$\{f \mid f \in S \leftrightarrow T \wedge \forall(x,y,z).(x,y,z \in S \times T \times T \wedge (x \mapsto y) \in f \wedge (x \mapsto z) \in f \Rightarrow y = z)\}$
Fonction totale	$S \rightarrow T$	$\{f \mid f \in S \nrightarrow T \wedge dom(f) = S\}$
Fonction injective partielle	$S \rightarrowtail T$	$\{f \mid f \in S \nrightarrow T \wedge f^{-1} \in T \nrightarrow S\}$
Fonction injective totale	$S \rightarrowtail T$	$S \rightarrowtail T \cap S \rightarrow T$
Fonction surjective partielle	$S \twoheadrightarrow T$	$\{f \mid f \in S \nrightarrow T \wedge ran(f) = T\}$

4.4 Les événements dans B-événementiel

Les événements permettent de faire évoluer l'état du système à modéliser à travers ses variables. Les événements sont organisés en clauses suivant l'objectif visé et dépendamment du fait que l'on ait affaire à un événement raffiné ou non. Dans la structure d'un événement, les clauses ANY, WHERE, et THEN regroupent respectivement les paramètres, les gardes, et les actions de l'événement. Tant que les gardes d'un événement ne sont pas satisfaites, ce dernier n'est pas déclenché.

Lors du raffinement d'une machine, plusieurs événements peuvent également être raffinés. Dans ces événements, on trouve en plus des clauses précédentes, les clauses REFINES et WITH. La clause REFINES indique l'événement en cours de raffinement (événement abstrait). La clause WITH introduite pour *witnesses* (en anglais) est optionnelle suivant les circonstances. Cette clause intervient lorsqu'un paramètre de l'événement abstrait n'existe plus dans l'événement concret. Il faut dans ce cas indiquer comment l'absence de ce paramètre abstrait est interpretée dans l'événement concret. La clause WITH intervient également pour la disparition des variables d'états abstraites.

Supposons un ensemble $(A \times B)$ et un événement evt qui prend en paramètre un élément p de cet ensemble. Si nous raffinons evt de façon à mieux définir le paramètre p, le nouvel événement raffiné $evt1$ aura deux nouveaux paramètres a et b tels que $a \in A$ et $b \in B$. Le paramètre p est donc remplacé par les deux paramètres a et b. Nous devons alors indiquer dans $evt1$, sous une clause WITH que p correspond au couple $a \mapsto b$ $(p = a \mapsto b)$. La figure 4.4 montre la structure d'un événement.

4.4.1 Les actions dans B événementiel

Dans un événement, les actions à exécuter peuvent être non déterministes ou détermisites. Les actions déterministes sont de la forme «*variable* := *expression*». L'opérateur (:=) caractérise ce type d'action. Dans l'événement *Inscrire* de la machine Ecole_Machine précédente, l'action act1 est déterministe :

`act1` : $inscription := inscription \cup \{etu \mapsto c\}$.

Les actions non déterministes sont associées à l'opérateur (: |). Ces actions sont de la forme «$liste\_de\_variables$: | $predicat\_avant\_apres$» . Les prédicats avant après, *Before After Predicate*(BAP) sont des expressions associées à des actions. Elles dénotent les relations qui existent entre les variables intervenant dans une action, avant et après son exécution.

Pour les illustrer, supposons une action déterministe de la forme $r := r + 1$, r étant une variable. Le prédicat avant après correspondant à cette action est $r' = r + 1$. La variable r' contient la nouvelle valeur de la variable r, après l'exécution de l'action. La variable r représente la valeur avant l'exécution de l'action. La notation r' est une convention. Dans l'expression du BAP, le symbole d'affectation (:=) est remplacé par une simple égalité (=).

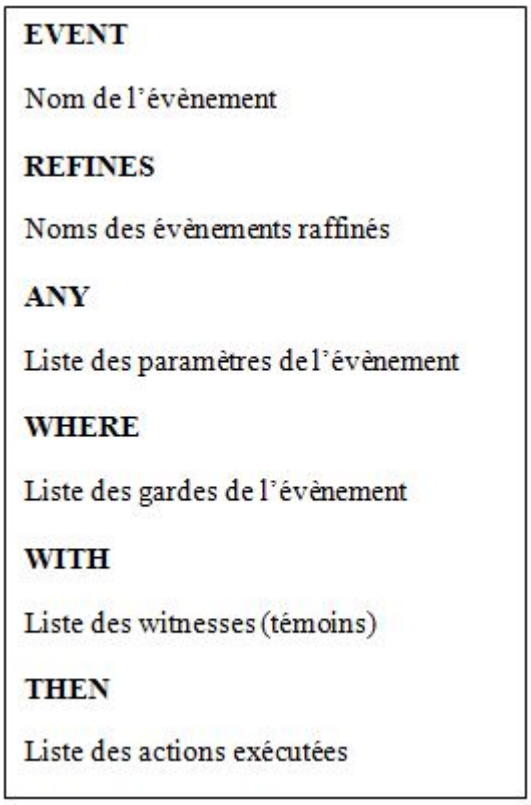

FIGURE 4.4 – Structure générale d'un événement

Le prédicat avant après de l'action act1 de l'événement $Inscrire$ est :

$inscription' = inscription \cup \{etu \mapsto c\} \wedge Etudiants' = Etudiants \wedge roles' = roles \wedge cours' = cours.$

Nous remarquons que les autres variables du modèle ont également été introduites dans ce prédicat. Elles permettent de montrer que seule la variable *inscription* est modifiée dans l'événement. Cela se traduit par le fait qu'aucune transformation n'est apportée à ces variables.

Un exemple d'action non deterministe est comme suit : $a, b : |a' > b \wedge b' > a' + c$. Les éléments a ,b et c sont des variables. a' et b' contiennent les nouvelles valeurs de a et b. Lorsque l'action est exécutée, a prend la valeur a' et devient superieure à b. b est aussi modifiée et devient superieure à a' additionnée à la valeur de la variable c qui ne change pas.

Une autre forme d'action non déterministe associe l'opérateur ($:\in$). Ce type d'action est comme suit : «*variable* $:\in$ *ensemble*». Une action $x :\in A$ (avec x une variable et A un ensemble) signifie que x prend une valeur quelconque dans l'ensemble A. On peut initialiser les variables d'une machine avec ce type d'action.

4.5 Le raffinement

Le raffinement est un moyen permettant d'ajouter des détails à la spécification d'une machine abstraite. Dans B-événementiel, la structure d'une machine de raffinée est presqu'identique à celle d'une machine abstraite. En plus des clauses citées pour une machine abstraite, une machine raffinée inclut une clause REFINES sous laquelle est définie la machine abstraite qu'elle raffine.

Le raffinement conserve toutes les propriétés déjà prouvées dans une machine abstraite. Des nouveaux événements et des nouvelles variables peuvent être ajoutés à une machine raffinée. Un nouvel événement ajouté raffine un événement qui exécute une action *skip* (action qui ne fait rien).

Aussi dans un raffinement, des invariants appelés *invariants de collage* permettent de créer des liens entre les états de la machine abstraite et ceux

de la machine concrète. Ces invariants sont déclarés sous la clause INVARIANTS. La structure d'une machine raffinée est présentée à la Figure 4.5. Dans cette figure, on distingue une clause VARIANT qui sert à définir des variantes. Ces dernières permettent de définir des propriétés qui seraient utiles lorsqu'un évènement se déclenche de façon infinie sans rendre la main à d'autres évènements abstraits.

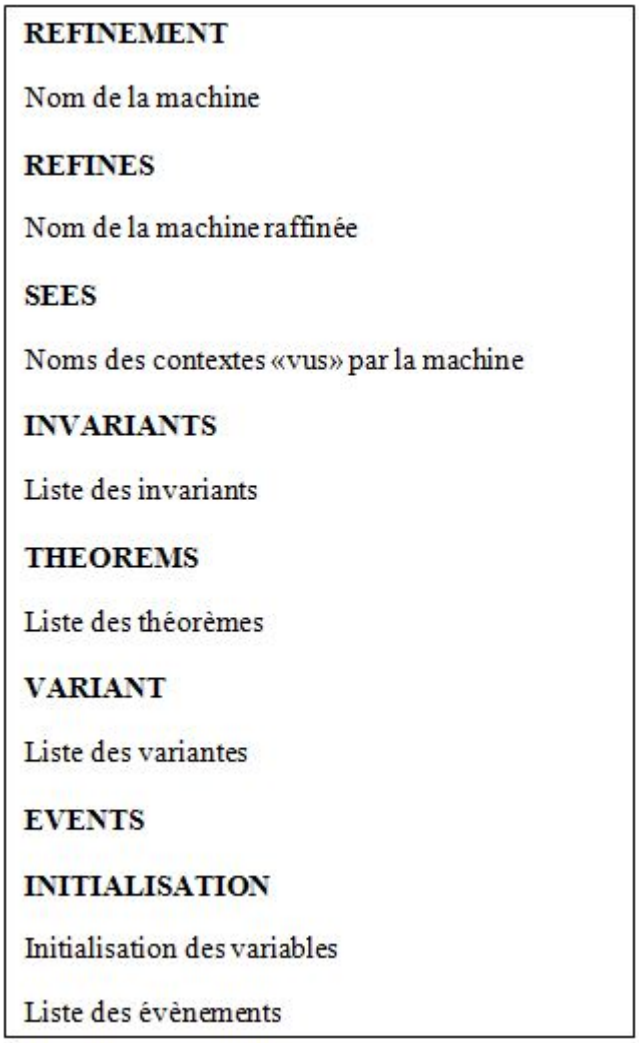

FIGURE 4.5 – Structure d'une machine raffinée

Exemple de raffinement

Supposons que notre règle R1 précédente est renforcée comme suit : «*Un individu ayant le rôle étudiant peut s'inscrire à un cours s'il en obtient l'autorisation et s'il a réussi à tous les cours prérequis à ce cours*». Le contexte Ecole1 peut être modifié comme suit :

CONTEXT Ecole1
EXTENDS Ecole
CONSTANTS

$etudiant$
$inscrireCours$
Pre_requis

AXIOMS

axm1 : $etudiant \in Roles$
axm2 : $inscrireCours \in Permissions$
axm3 : $Pre\_requis \in (Cours \leftrightarrow Cours) \setminus id(Cours \leftrightarrow Cours)$

END

Nous ajoutons à ce contexte, un ensemble Pre_requis qui désigne l'ensemble des cours prérequis à un autre. Dans sa construction dans l'axiome 3 (axm3), il est privé de tous les éléments réflexifs ($id(Cours \leftrightarrow Cours)$). On ne peut donc pas avoir un cours d'anglais_1 prérequis du même cours anglais_1. Rappelons que l'opérateur $\leftrightarrow$ désigne une relation.
La machine Ecole_Machine est raffinée en une nouvelle machine Ecole_Machine1 comme suit :

MACHINE Ecole_Machine1
REFINES Ecole_Machine
SEES Ecole0
VARIABLES

$Etudiants$

roles

cours

inscription

Valider

INVARIANTS

inv1 : $Valider \in Etudiants \leftrightarrow cours$

EVENTS

Initialisation

extended

begin

act1 : $Etudiants := \varnothing$

act2 : $roles := \varnothing$

act3 : $cours := \varnothing$

act4 : $inscription := \varnothing$

act5 : $Valider := \varnothing$

end

Event *Inscrire* $\widehat{=}$

extends *Inscrire*

any

etu

c

where

grd1 : $etu \in Etudiants$

grd2 : $c \in cours$

grd3 : $etu \mapsto etudiant \mapsto inscrireCours \in Privilleges$

grd4 : $\{etu\} \times Pre\_requis[\{c\}] \subseteq Valider$

then

act1 : $inscription := inscription \cup \{etu \mapsto c\}$

end

END

Pour cette nouvelle machine, le raffinement consiste en l'ajout d'une nouvelle variable ($Valider$) qui contient tous les cours réussis par un étudiant. Cet ensemble pourrait être de la forme $\{bob \mapsto anglais_1, bob \mapsto math_1, bob \mapsto math_2, etc...\}$.

L'expression $Pre\_requis[\{c\}]$ dans la garde 4 de l'événement *Inscrire* renvoit l'ensemble des cours en relation avec le cours c. Ainsi, $\{etu\} \times Pre\_requis[\{c\}]$ cré des couples $(etu \mapsto cours\_pre\_requis)$ pour un même étudiant. On vérifie ensuite que cet ensemble de cours est inclus dans l'ensemble des cours réussis par l'étudiant avant de permettre son inscription.

Concrètement, si on considère qu'un cours $math2$ est pré-requis à un cours $math3$, alors un élément de l'ensemble Pre_requis est $math3 \mapsto math2$. $Pre\_requis[\{math3\}]$ retourne alors un singleton$\{math2\}$ avec l'instruction de la garde 4 (grd4). Pour le reste, il suffira de vérifier que le singleton $\{bob \mapsto math2\}$ est bien inclus dans l'ensemble $Valider$ (avec $etu = bob$). Pour cet exemple également nous ne modélisons pas les modifications des autres variables déclarées dans la machine.

4.6 Les obligations de preuves

Un élément très important dans la méthode B est la preuve. Les obligations de preuves définissent les propriétés qui doivent être vérifiées (prouvées) dans le modèle B-événementiel. Il s'agit par exemple de prouver des théorèmes, ou prouver que l'exécution d'un événement préserve toujours les propriétés définies dans le modèle. Les obligations de preuves ont la forme suivante : «$hypothses \vdash conclusion$» (sous les hypothèses prouver la conclusion). Il existe plusieurs types d'obligations de preuves. En pratique elle sont générées par des outils. Nous présentons quelques-unes.

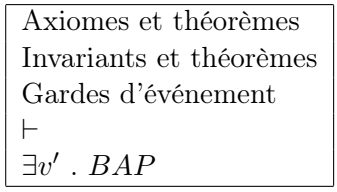

FIGURE 4.6 – Faisabilité dans événement

4.6.1 Faisabilité d'un événement (*Feasibility proof*)

Cette obligation de preuve permet de garantir qu'une action non déterministe est faisable. Sa forme est présentée à la Figure 4.6. Dans cette figure, BAP désigne un prédicat avant après. Si on considère une action non déterministe $v : |BAP(v, v')$ (avec $BAP(v, v')$ un prédicat avant après définit sur la variable v), alors il faut montrer qu'il existe une nouvelle valeur possible v' pour v.

4.6.2 Préservation de l'invariant (*Invariant preservation*)

L'obligation de preuve de préservation de l'invariant d'un modèle permet de s'assurer que les propriétés d'invariance définies dans le modèle sont respectées quelque soit les changements qui s'opèrent sur les variables du système. La forme de cette obligation est présentée dans la Figure 4.7. Pour les événements raffinés une structure du même genre est adoptée à Figure 4.8.

Avec la modélisation faite pour la règle R1, nous devons prouver que la modification de la variable *inscription* à travers l'évenement *Inscrire* respecte toujours l'invariant 4 (inv4) de la machine Ecole_Machine. Cet invariant définit le type de la variable *inscription* : «$inscription \in Etudiant \leftrightarrow cours$».

La Figure 4.9 présente l'obligation correspondant à cet événement.

Axiomes et théorèmes Invariants et théorèmes Gardes d'évènement Prédicat avant après de l'évènement ⊢ Nouvelle expression de l'action de l'événement

FIGURE 4.7 – Préservation de l'invariant

Axiomes et théorèmes Invariants et théorèmes abstraits Invariants et théorèmes concrets Gardes de l'évènement concret Prédicats witness Prédicat avant après de l'évènement concret ⊢ Nouvelle expression de l'action de l'événement en fonction de l'invariant

FIGURE 4.8 – Préservation de l'invariant pour les événements raffinés

$inscription \in Etudiants \leftrightarrow cours$ $etu \in Etudiants$ $c \in cours$ $etu \longmapsto etudiant \mapsto inscrireCours \in Privileges$ ⊢ $inscription \cup \{etu \mapsto c\} \in Etudiants \leftrightarrow cours$

FIGURE 4.9 – Préservation de l'invariant pour l'évènement *Inscription*

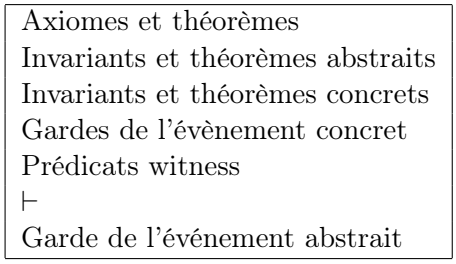

FIGURE 4.10 – Force de garde

4.6.3 Renforcement de la garde d'un événement (*Guard strengthening proof*)

Dans un raffinement, les obligations de preuves de renforcement de gardes, permettent de s'assurer que les gardes des événements concrets sont plus «fortes» que celles des événements abstraits. Ainsi, l'exécution d'un événement concret peut entrainer l'exécution de sa spécification abstraite. La Figure 4.10 montre la forme de cette obligation.

Les détails sur les obligations de preuves et des exemples sont mentionnés dans [2].

4.7 Conclusion

B événementiel est une méthode de spécification formelle. Elle se révèle utile lorsqu'on doit modéliser des systèmes pour lesquels la sûreté n'est pas négligeable. La méthode B qui a précédé B événementiel à été utilisée dans le cadre du projet METEOR (métro sans conducteur) réalisé par Matra Transport International. Son importance a donc été démontrée. Cette méthode a conduit également au développement d'outils de support comme Atelier B, Click'n'proove ou encore Rodin que nous utilisons dans nos travaux. Pour notre mémoire, nous utiliserons la méthode B événementiel pour spécifier et prouver des propriétés dans les modèles CatBAC.

Dans les chapitres précédents, nous avons présenté nos objectifs de recherche (Chapitre 1), des modèles de contrôle d'accès (Chapitre 2) et les outils que nous utiliserons pour atteindre ces objectifs (CatBAC Chapitre

3, B-événementiel Chapitre 4). Dans le prochain chapitre, nous montrons la façon dont nous utilisons CatBAC et B-événementiel pour atteindre notre objectif global : construire un cadre pour la spécification et la vérification des systèmes de contrôle d'accès hybrides.

Chapitre 5

Modélisation des systèmes de contrôle d'accès avec CatBAC (contributions)

Dans ce chapitre, nous présentons la façon dont nous interprétons formellement les modèles CatBAC et leurs raffinements.

5.1 Introduction

Notre objectif dans ce mémoire est de modéliser par raffinements des systèmes de contrôle d'accès hybrides. De ce fait, dans les chapitres précédents nous avions présenté quelques modèles de contrôles d'accès (chapitre 2) et des techniques pour modéliser les systèmes qui implémentent ces modèles (chapitre 3 et 4). Nous avions montré au chapitre 3, que CatBAC est la technique de modélisation qui répond bien, aux besoins de la construction des systèmes de contrôle d'accès hybrides. Elle s'inspire de la modélisation avec des diagrammes de classes UML, pour fournir des représentations graphiques des systèmes. Aujourd'hui l'importance de UML dans la modélisation des systèmes n'est plus à démontrer. CatBAC constitue donc pour nous, un élément essentiel pour la modélisation des systèmes hybrides.

Nous avions également montré qu'une modélisation avec CatBAC uni-

quement n'est pas suffisante pour garantir que le futur système à implémenter sera sûr par rapport aux propriétés de sécurité qu'il doit satisfaire. Nous avons par conséquent introduit au chapitre 4, la méthode formelle B-événementiel que nous allons associer à CatBAC pour modéliser des systèmes sûrs par constructions. B-événementiel est une méthode formelle qui permet de modéliser par raffinements, des systèmes. Notre choix s'est porté sur cette méthode pour deux principales raisons. La première est qu'elle utilise tout comme CatBAC, la notion de raffinement. La seconde raison est qu'elle permet de vérifier grâce à un mécanisme de preuves, que les modèles ne contredisent pas les propriétés qu'ils doivent satisfaire.

Dans le présent chapitre, nous montrons comment nous pouvons utiliser CatBAC et B-événementiel pour modéliser des systèmes de contrôle d'accès hybrides. Notre idée est de modéliser la structure des systèmes avec CatBAC et de compléter et vérifier ces modélisations avec B-événementiel. Nous devons donc produire les équivalents formels des modèles CatBAC et de leurs raffinements, et les traduire ensuite vers B-événementiel.

Par conséquent, dans les sections suivantes, nous présentons les interpretations (équivalents) formelles que nous faisons des modèles CatBAC et de leurs raffinements. Notons que dans ce chapitre, nous présentons uniquement les interpretations formelles des modèles CatBAC. La traduction de ces modèles vers B-événementiel sera abordée au chapitre 6. Toutefois pour comprendre les éléments qui guident nos choix d'interprétations, nous présentons à la section suivante des travaux sur la traduction de UML vers B et B-événementiel.

5.2 Traduction de UML vers B et B événementiel

L'objectif de cette section est de montrer l'intuition que nous utilisons pour interpréter formellement les modèles CatBAC. Nous présentons quelques travaux qui traitent de la traduction de modèles UML vers B-événementiel.

Ainsi, partant du fait que les spécifications semi-formelles sont plus compréhensibles en général que les spécifications formelles, quelques travaux se

sont penchés sur des techniques de traduction de UML vers B.

Dans [12], R. Laleau et A. Mammar, présentent une méthode de génération de spécifications B à partir de notations UML. Nous reprenons ici quelques éléments présentés dans ce travail.

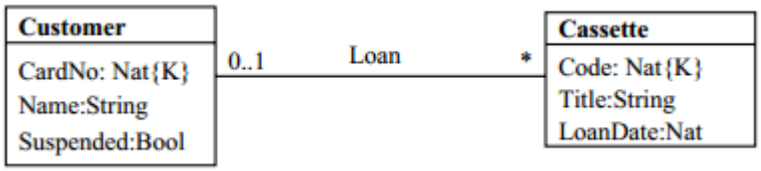

FIGURE 5.1 – Modèle UML (*Figure extraite de* [12])

Dans le modèle UML ci-dessus, nous avons deux classes (*Customer* et *Cassette*). Une association appelée *Loan* relie ces deux classes. Le modèle peut être interprété de la façon suivante : *un client (Customer) peut emprunter plusieurs cassettes. Une cassette est empruntée par 0 ou 1 client à la fois*. La notation {K} indique que l'attribut devant lequel il est placé est une clé primaire. La traduction de la classe *Cassette* vers **B** est alors réalisée comme suit :

SETS

$CASSETTE$

VARIABLES

$Cassette$

INVARIANTS

$Cassette \subseteq CASSETTE$

Un ensemble de cassettes appelé *CASSETTE* est introduit (clause SETS). La classe *Cassette* est déclarée comme une variable. Cette variable représente un sous-ensemble de l'ensemble *CASSETTE*. La définition des attributs est comme suit :

VARIABLES

$Code, Title, LoanDate$

INVARIANTS

$Code \in Cassette \rightarrowtail NAT \wedge Title \in Cassette \rightarrow STRING \wedge LoanDate \in Cassete \rightarrow NAT$

Ici, ($\rightarrowtail$) désigne une fonction injective totale (Chapitre 4). ($\rightarrow$) est une fonction totale. Chaque attribut étant déclaré sous forme de fonction, $Title[\{Cassette\}]$ renvoit l'ensemble des titres des cassettes. Ces titres sont de type STRING (clause INVARIANTS). La traduction de la classe *Customer* est réalisée de façon similaire.

Les associations entre les classes sont représentées par des relations ($\leftrightarrow$). Suivant les cardinalités, ces associations deviennent des fonctions (bijectives, injectives...). Ici l'association Loan est une fonction partielle ($\nrightarrow$).

VARIABLES

$Loan$

INVARIANTS

$Loan \in Cassette \nrightarrow Customer$

D'autres travaux de traductions de diagrammes UML vers B sont présentés dans [13]. I. Akram y propose un couplage de spécification B et des descriptions UML. Il fournit des schémas définissant les liens de structures et de sémantiques entre les méta-modèles B et UML. La Figure 5.2 présente quelques correspondances entre ces méta-modèles. Dans cette figure, le niveau supérieur montre un méta-modèle partiel de B. Le niveau inférieur montre celui du diagramme de classes de UML.

Les correspondances entre les deux méta-modèles sont affichées par les flèches en pointillées. Par exemple, une correspondance est établie entre *BSet* (les ensembles dans B) et *Class* (les classes en UML). Cette correspondance est identique à celle présentée précédemment dans [12]. L'objectif général de

ces transformations est de produire une documentation dans les projets B en utilisant des diagrammes UML.

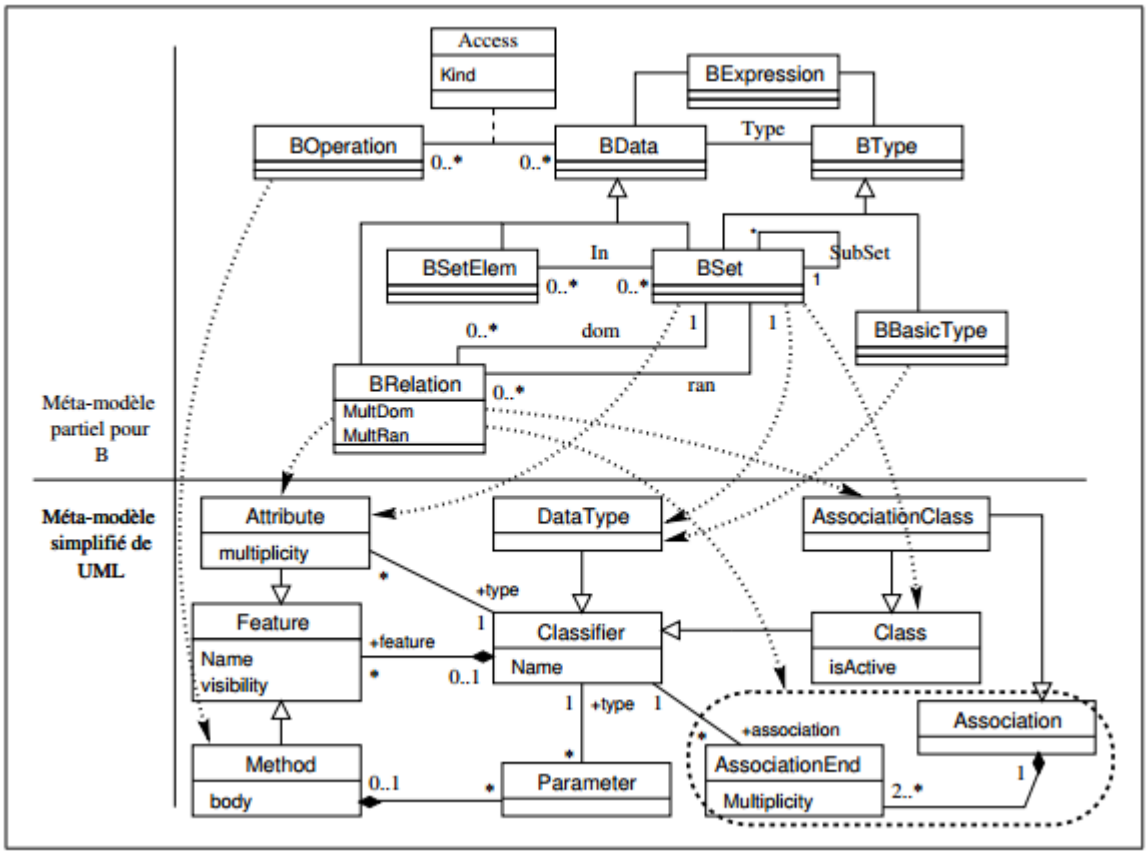

FIGURE 5.2 – Correspondances entre méta-modèles B et UML (*Figure extraite de* [13])

L'environnement de modélisation Rodin, que nous utilisons pour nos travaux dispose également d'un plugin appelé UML-B [23], permettant une traduction vers B événementiel, des modèles construits dans un raisonnement UML. Le tableau 5.1 fournit quelques transformations de UML-B vers B événementiel.

TABLE 5.1 – Correspondance UML-B et B événementiel

UML-B	**B événementiel**
Classe (Class)	Ensemble (SET)
Classe : instances de variables	Variable $\subseteq$ Ensemble (SET)
Classe : instances de variables qui a une super classe	Variable $\subseteq$ Super Classe
Classe : instances fixes qui a une super classe	Constante $\subseteq$ Super Classe
Attribut (card *..1,1)	Variable $\in$ Classe $\rightarrow$ Type
Attribut (card *..0,1)	Variable $\in$ Classe $\nrightarrow$ Type
Attribut (card *..*)	Variable $\in$ Classe $\leftrightarrow$ Type
Associations	Se comporte comme les attributs suivant les cardinalités
Association (card *..*)	Variable ou constante $\in$ Classe $\leftrightarrow$ Classe
Association (card 0,1..*)	Variable ou constante $\in$ Classe $\nrightarrow$ Classe
Constructeur d'une classe	Evénement

Ce tableau contient une première colonne (UML-B) pour désigner les éléments de modélisation de UML-B et une seconde colonne (B événementiel) qui correspond à l'interprétation que l'on peut faire de ces éléments. Ainsi, dans une spécification de contexte B-événementiel, une classe UML-B peut correspondre à un ensemble d'éléments (SET). Une classe peut aussi être définie comme une constante qui possède une super-classe. Dans ce cas il s'agit d'un héritage de classes.

Dans une spécification de machine, les classes sont définies comme des variables. Les attributs des classes sont définis comme des fonctions associant ces classes à des types. Les associations entre classes sont définies de façon similaire que les attributs ; elles associent une classe à une autre classe. Ce sont par défaut des associations unidirectionnelles. Nous remarquerons

par exemple que l'association *Loan* présentée précédemment respecte cette structure. Enfin les opérations d'une classe sont définies comme des événements.

Il existe également des traductions d'autres langages semi-formels vers B. Par exemple une traduction de ASTD (*Algebraic State Transition Diagram*) vers B est fournie dans [8].

Conclusion

Dans cette section, nous avons présenté des travaux qui nous permettrons d'orienter l'interpretation formelle des modèles CatBAC, afin de faciliter leurs traductions en B événementiel. Dans les sections qui suivent, nous présentons notre premier objectif de recherche qui est d'élaborer une description formelle du raffinement des modèles CatBAC. Cela consiste à indiquer la façon dont les composants UML des modèles CatBAC doivent être interprétés formellement à chaque passage vers un modèle plus raffiné.

5.3 Interprétation formelle des modèles CatBAC

Pour faciliter la compréhension de notre travail, nous utilisons l'exemple de politique de sécurité ci-dessous.

5.3.1 Exemple de politique de sécurité : Pol_1

Dans un laboratoire de recherche informatique installé au sein d'une université, les accès des membres sont gérés à partir de leurs rôles et de leurs groupes. Dans ce laboratoire, les membres ont deux types de rôles : le rôle *Etudiant* et le rôle *Responsable de laboratoire* que nous noterons *Resp_lab*. **Tous les membres du laboratoire sont divisés en deux groupes qui sont : le groupe de recherche en sécurité informatique (*GRSI*) et le groupe de recherche en traitements d'images (*GRTI*). Lorsqu'un nouveau menbre qui n'est ni étudiant ni responsable de laboratoire intègre le laboratoire, il est directement affecté à un groupe et obtient ses accès à partir de ce groupe.** C'est souvent le cas des stagiaires.

Les permissions des menbres du laboratoire sont déterminées par les règles de contrôle d'accès suivantes :

1. Tous les membres du laboratoire, peuvent accéder à tous les imprimantes, téléphones et scanners du laboratoire. Afin de faciliter l'accès à ces ressources de base, le laboratoire a décidé de les inclure dans un groupe appelé *Base*. Ainsi, toutes les ressources de base seront directement affectées à ce groupe de ressources, qui est **le seul** défini par la politique.
2. Tous les membres du laboratoire peuvent accéder à un ou plusieurs ordinateurs de travail, dépendamment des ordinateurs qui leurs sont attribués.
3. Seuls les étudiants et les responsables du laboratoire, ont le droit d'accéder au serveur *Biblio_1*, qui donne accès à des articles scientifiques, dont la consultation régulière est payante (l'université subventionne la consultation de ces articles pour ses étudiants et son personnel régulier).
4. Tout membre d'un groupe a accès à toutes les archives de recherche du laboratoire.

Modélisation

Pour modéliser cette politique de sécurité, nous devons d'abord identifier tous les *mécanismes d'accès* des sujets à leurs permissions. Nous appelons mécanisme d'accès, l'architecture qui présente la façon dont les permissions sont attribuées aux sujets et la façon dont les sujets accèdent à ces permissions. Un exemple de mécanisme d'accès est le suivant : les sujets sont affectés aux rôles et les rôles sont affectés aux permissions. Par conséquent, les sujets accèdent à leurs permissions à partir des rôles qui leur sont attribués.

Dans le reste de notre document, nous appelons notre exemple de politique de sécurité, Pol_1. Dans cette politique, nous pouvons identifier deux mécanismes d'accès. La Figure 5.3 présente le premier mécanisme d'accès.

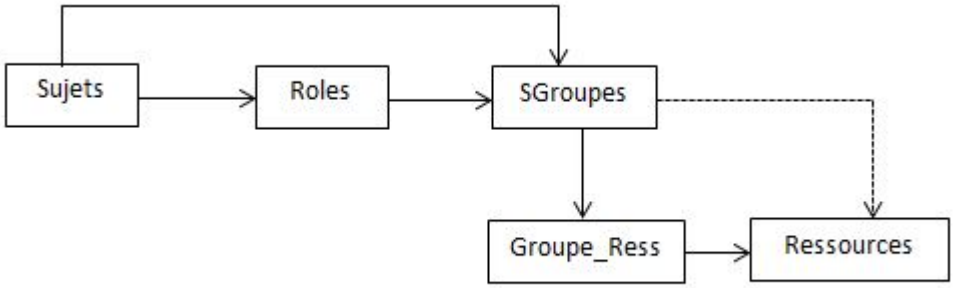

FIGURE 5.3 – Premier mécanisme d'accès

Dans cette figure, les flêches en traits pleins permettent de déterminer le mécanisme d'accès valide. D'après ce mécanisme, des sujets (*Sujets*) peuvent :

- accéder à des ressources (*Ressources*) qui sont affectées à un groupe de ressources (*Groupe_ Ress*), en utilisant à la fois leurs rôles (*Roles*) et leurs groupes de sujets (*SGroupes*). Le groupe de ressources (*Groupe_ Ress*) sera ensuite instancié pour produire le groupe de ressources concret (*Base*).
- accéder à des ressources qui sont affectées à un groupe de ressources (*Groupe_ Ress*) en utilisant uniquement leurs groupes de sujets (*SGroupes*).

Ce mécanisme nous est suggéré par la règle de contrôle d'accès (1) de la politique Pol_1 énumérée précédemment. Cette règle indique que les ressources de base sont regroupées dans un élément appelé *Base*. Par conséquent il existe une relation entre cet objet qui est le groupe de base et certaines ressources du système.

Pour le même mécanisme, les exigences de sécurité marquées en gras dans le texte de la politique Pol_1 s'appliquent. D'après ces exigences, tous les membres du laboratoire sont affectés à un groupe de recherche. Cela signifie que tous les membres qui sont étudiants ou responsables de laboratoire (le laboratoire ne définit que deux rôles) sont affectés à un groupe. On peut

en déduire qu'il existe une relation entre des sujets et des rôles (*Etudiant* ou *Resp_lab*), une autre relation entre des rôles et des groupes de sujets (*GRSI* ou *GRTI*), puis une dernière relation entre des groupes de sujets et des groupes des ressources (ici, il n'y en a qu'un seul appelé *Base*).

D'apres les mêmes exigences de sécurité, tous les membres du laboratoire qui ne sont ni étudiants, ni responsables de laboratoire sont directement affectés à un groupe de sujets (*GRSI* ou *GRTI*). Cela suggère aussi qu'il existe une relation directe entre des sujets (*Sujets*) et les groupes de sujets ((*SGroupes*)).

Le second mécanisme d'accès de la politique Pol_1 est présenté à la Figure 5.4. Ce mécanisme est suggéré par les règles de contrôle d'accès (2,3, et 4) énumérées dans la politique. Dans ce mécanisme, l'accès aux ressources ne se fait pas par l'intermédiaire du groupe de ressources *Base*. Les ressources sont directement affectées aux groupes des sujets *GRSI* et *GRTI*. Dans cette figure, le mécanisme valide est également identifié par les flêches en traits pleins.

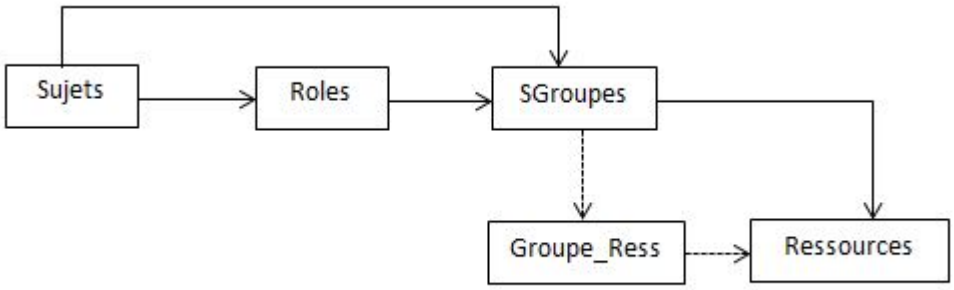

FIGURE 5.4 – Second mécanisme d'accès

Les deux mécanismes des Figures 5.3 et 5.4, peuvent être regroupés, comme présenté à la Figure 5.5. Dans cette figure, toutes les flêches sont en traits pleins. Par conséquent chaque chemin représente un accès valide par rapport à une ressource.

Une fois le mécanisme d'accès de la politique Pol_1 identifié (Figure 5.5), nous pouvons modéliser avec CatBAC le système qui lui correspond. Nous définirons pour chaque niveau de raffinement, les interpretations formelles à élaborer.

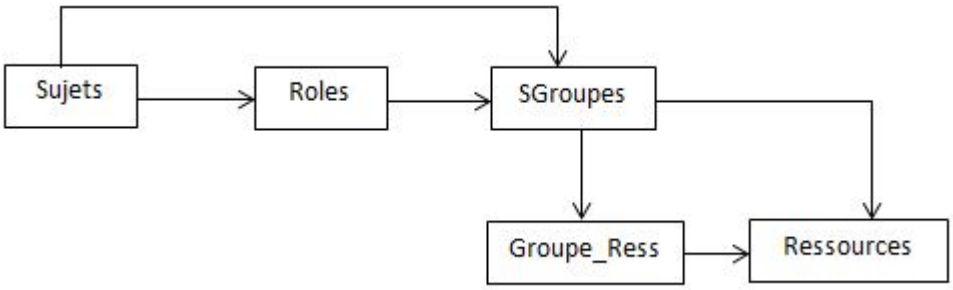

FIGURE 5.5 – Mécanisme d'accès global

5.3.2 Modèle initial (CatBAC)

Pour les raisons énoncées au chapitre 3, nous basons notre travail sur une interprétation précise du méta-modèle CatBAC comme suit : **un sujet est associé à une permission qui peut être donnée dans un certain contexte**. Le contexte peut ne pas être modélisé. Dans ce cas, il est considéré toujours vrai. En prenant les différents types de catégories dans CatBAC, l'interprétation précédente consiste à relier la catégorie des sujets (SCat) au tuple (catégorie des actions ACat, des ressources RCat et des contextes CCat). Cette interprétation est indiquée à la Figure 5.6. A partir de cette interprétation, nous allons définir les différentes opérations formelles qui interviennent dans le processus de raffinement.

Pour la modélisation de la politique Pol_1, la Figure 5.6 correspond également au modèle le plus abstrait avec lequel nous démarrons tout travail de modélisation. Dans la suite de nos travaux, l'association n-aire est nommée ACCESS. Cette association est une relation entre une catégorie de sujets

SCat et un triplet (catégorie d'actions ACat, catégorie de ressources RCat, catégorie de contextes CCat).

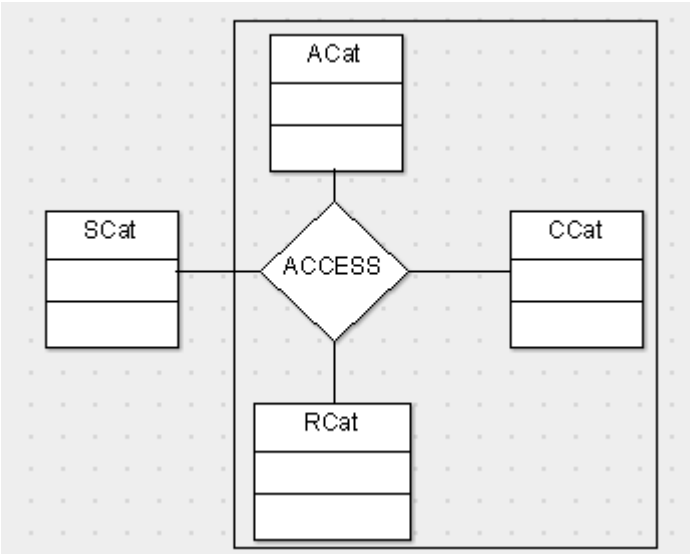

FIGURE 5.6 – Modèle abstrait

5.3.3 Raffinement de classes

D'après le mécanisme d'accès que nous avons identifié pour la politique Pol_1 (Figure 5.5) nous pouvons définir les éléments suivants :

- les sujets : *Sujets*
- les regroupements des sujets : rôles (*Roles*), groupes (*SGroupes*)
- les ressources : *Ressources*
- les regroupements des ressources : *Groupe_Ress*
- les actions : *ACat* (pas de regroupement)
- les contextes : *CCat* (nous avons décidé de les modéliser dans ce cas particulier).

Chacun de ces éléments correspond à une catégorie que nous modéliserons à des niveaux de raffinement différents. Pour commencer nous allons raffiner la catégorie de sujets *SCat* du modèle initial (Figure 5.6).

Le raffinement d'une classe est dans la syntaxe graphique, un héritage entre les nouvelles classes et la classe à raffiner. Ici nous allons raffiner la catégorie des sujets vers trois nouvelles catégories : une catégorie de rôles (*Roles*), une catégorie de groupes (*SGroupes*) et une catégorie de sujets (*Sujets*) qui correspond aux sujets concrets du modèle. Ces trois catégories désignent les éléments qui servent à regrouper les sujets dans la politique Pol_1. Ce premier raffinement est illustré à la Figure 5.7.

Pour éviter de surcharger la modélisation on notera le raffinement comme décrit à la Figure 5.8. A ce niveau nous avons conservé la notation présentée dans [24].

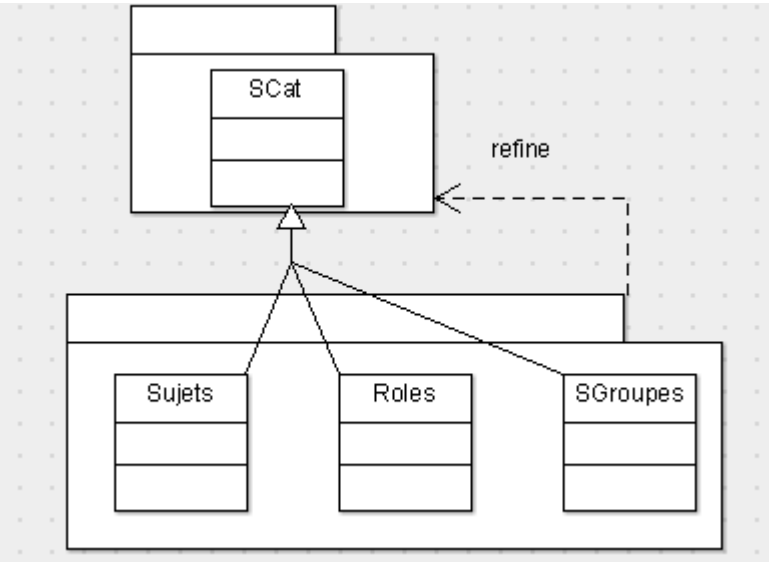

FIGURE 5.7 – Raffinement de classes (1)

D'après le mécanisme d'accès de Pol_1 (Figure 5.5), les sujets sont reliés aux rôles et aux groupes et les rôles, sont reliés aux groupes. Supposons donc le modèle de la Figure 5.9. Une association unidirectionnelle SR (mise pour Sujets_Roles) relie la catégorie *Sujets* à la catégorie *Roles*. Une seconde association RSG (mise pour Roles_SGroupe) relie la catégorie *Roles* à la catégorie *SGroupes*, et une troisième association SSG (Sujets_SGroupes) relie

la catégorie *Sujets* à la catégorie *SGroupes*. Ces associations proviennent de l'association réflexive présente sur chaque catégorie du méta-modèle CatBAC (chapitre 3).

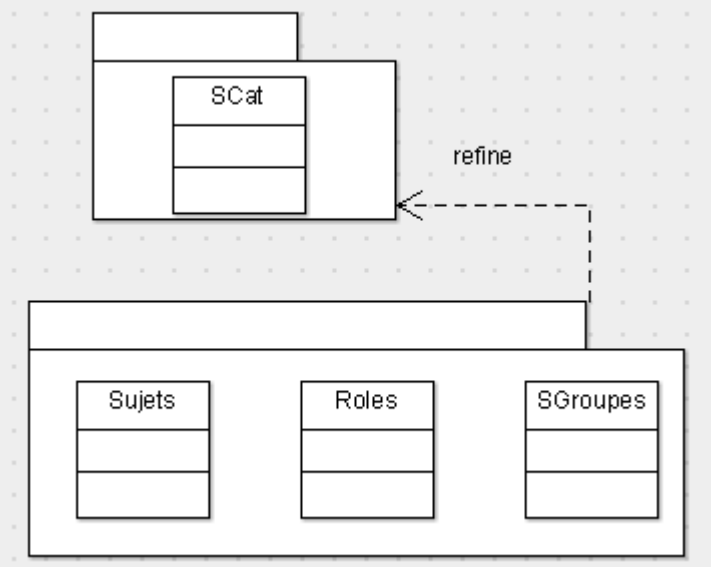

FIGURE 5.8 – Raffinement de classes (2)

La création de ce type d'association n'est pas automatique au raffinement d'une classe. Nous y reviendrons en détails dans la sous-section suivante. Nous interprétons chaque association comme une relation entre les catégories au milieu desquelles elle est placée.

Interprétation formelle

Soient les ensembles SCat (dans le modèle abstrait), Sujets, Roles, SGroupes (dans le modèle raffiné) et une relation $\Re$. Cette relation correspond à l'ensemble des parties formé par le produit cartésien des deux ensembles entre lesquels elle est placée. Elle correspond à la notation $\leftrightarrow$ dans B. Pour mentionner que les catégories Sujets, Roles et SGroupes raffinent la catégorie SCat, il faut indiquer les propriétés suivantes :

- $Sujets \subseteq SCat \wedge Roles \subseteq SCat \wedge SGroupes \subseteq SCat$
- $Sujets \neq Roles \wedge Sujets \neq SGroupes \wedge Roles \neq SGroupes$

A partir de ces propriétés, nous indiquons que les nouvelles catégories du modèle concret sont des parties disjointes de la catégorie SCat du modèle abstrait. Cette interpretation est inspirée des travaux que nous avons présenté à la section 5.2 sur la traduction des modèles UML vers B-événementiel. Une fois les catégories concrètes définies, nous devons mentionner celle qui simule le comportement de la catégorie abstraite.

En effet, la catégorie SCat du modèle abstrait est reliée au tuple (ACat, RCat, CCat) (Figure 5.6). On peut en déduire que c'est à cette catégorie que les permissions (action (ACat), ressource (RCat), contexte (CCat)) sont affectées. Nous devons par conséquent indiquer la catégorie concrète qui joue la même fonction. D'après le mécanisme d'accès de Pol_1, il s'agit de la catégorie des groupes de sujets ($SGroupes$). Nous établissons alors la propriété suivante :

- $var(SGroupes) = var(SCat)$

Cette propriété signifie que la variable du modèle abstrait qui se comporte comme une catégorie de sujets (SCat), doit maintenant se comporter comme une catégorie de groupe de sujets (SGroupes). Par conséquent on ne fera plus appel à la variable SCat dans le reste du raffinement. Nous utilisons la notion de variable car lorsque notre modèle final sera produit et implémenté, toutes les opérations liées aux groupes des sujets (SGroupes) s'éffectueront par appel d'une variable devant représenter le groupe $GRSI$ ou le groupe $GRTI$. Nous verrons l'utilité concrète de ce type d'expression au moment de traduire les modèles vers B-événementiel.

Les nouvelles associations SR, RSG et SSG du modèle concret sont définies ci-dessous :

- $SR \in (Sujets)\ \Re\ (Roles)$
- $SR \in (Sujets)\ \Re\ (SGroupes)$
- $RSG \in (Roles)\ \Re\ (SGroupes)$

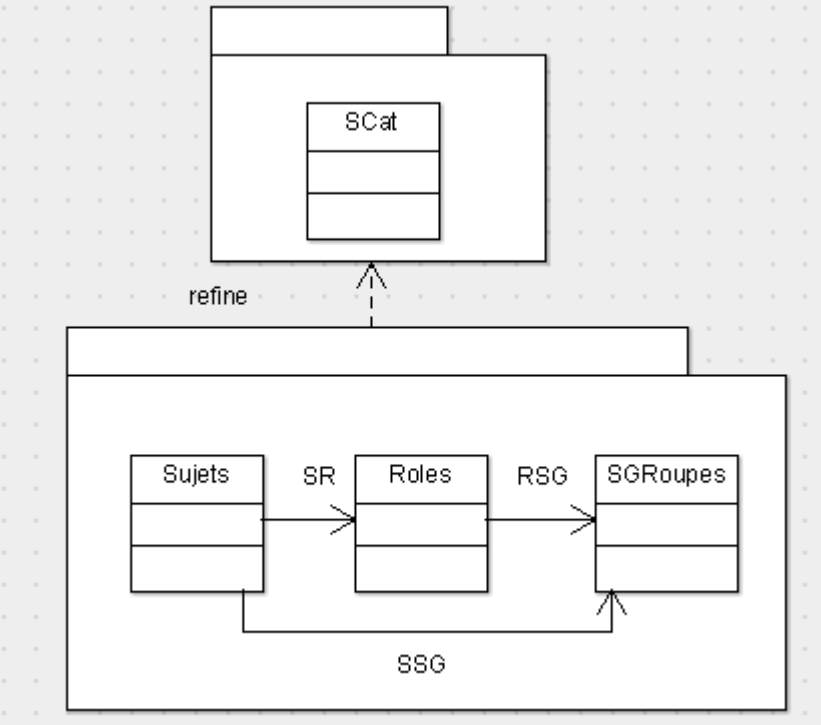

FIGURE 5.9 – Raffinement de classes (3)

Ces expressions indiquent que les associations d'un modèle CatBAC sont par défaut des relations constituées des catégories au milieu desquelles elles sont placées. Si nous prenons la relation SR par exemple, elle permet d'affecter un sujet à un rôle. Le domaine de cette relation est $Sujets$ et son co-domaine est $Roles$. Par conséquent l'opération $SR(s)$ avec $s \in Sujets$ nous renvoit l'ensemble des rôles affectés au sujet s.

Aussi, d'après le mécanisme d'accès de la politique Pol_1, certaines ressources sont regroupées dans un groupe appelé *Base*. Nous devons par conséquent raffiner la catégorie des ressources $RCat$ du modèle inital (modèle abstrait) en deux nouvelles catégories : $Groupe\_Ress$ et $Ressources$ (Figure 5.11). Ce procédé est le même que pour le raffinement de SCat. Nous le présenterons à un autre niveau de raffinement (Figure 5.11).

5.3.4 Raffinement d'associations

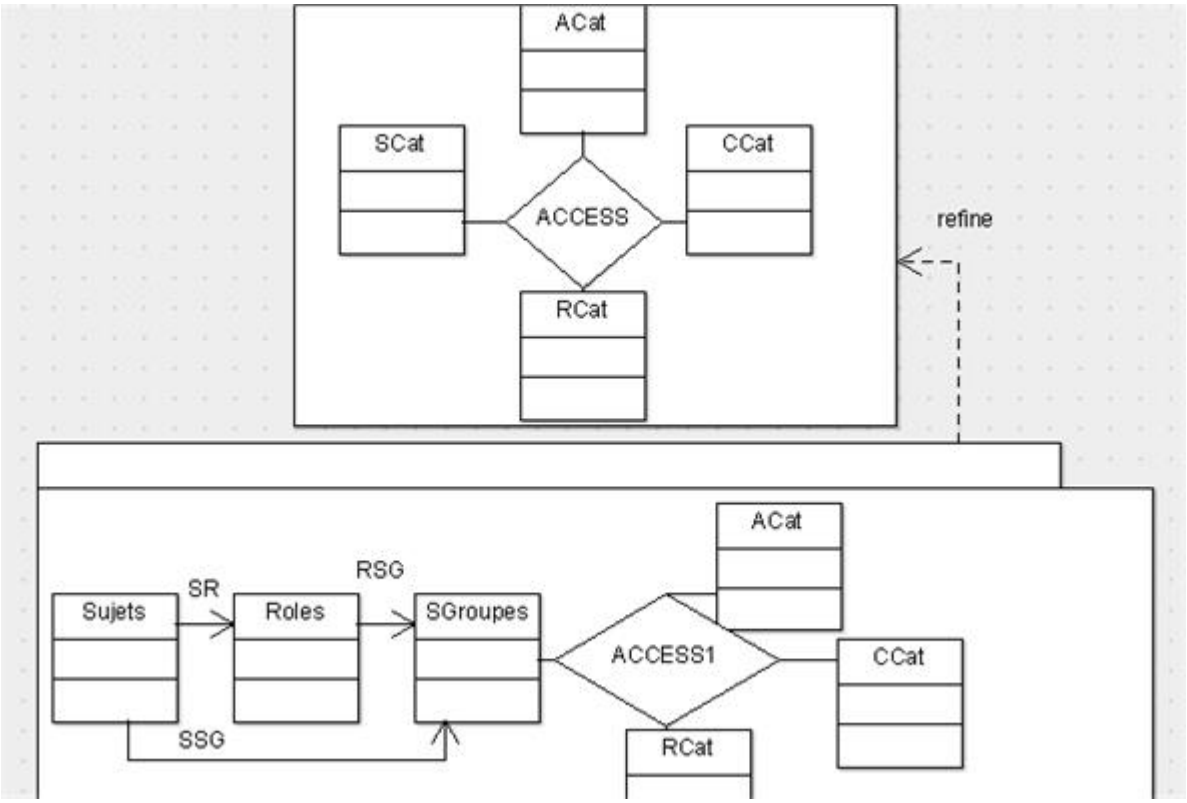

FIGURE 5.10 – Premier raffinement de modèle

Soit la Figure 5.10 qui présente les modèles complets correspondant à la Figure 5.9. Dans le second modèle de la Figure 5.10, on remarque que l'association $ACCESS$ du modèle abstrait à un nouveau nom qui est $ACCESS1$. Ici, $ACCESS1$ est une nouvelle association qui raffine $ACCESS$. Cela est dû au fait que la catégorie $SCat$ du modèle abstrait a été raffinée de façon à ce que son rôle soit joué par la catégorie $SGroupes$. Par conséquent l'association abstraite $ACCESS$ n'a plus La même signification dans le modèle concret car le domaine de son raffinement ($ACCESS1$) est $SGroupes$.

Une autre façon de voir ce raffinement est comme suit. Dans le modèle abstrait, $ACCESS$ permettait de creer des tuples de la forme (SCat, ACat,

RCat, CCat). Vu que le comportement de $SCat$ est joué par $SGroupes$ dans le modèle concret, une nouvelle association de la même forme que $ACCESS$ doit permettre de créer des tuples de la forme (SGroupes, ACat, RCat, CCat). Cette nouvelle association $ACCESS1$ est simplement un raffinement de l'association abstraite $ACCESS$.

Interpretation formelle

Soient les ensembles SCat, ACat, RCat, CCat (pour le modèle abstrait **MA**) et Sujets, Roles, SGroupes (pour le modèle raffiné **MR**). Puisque les catégories ACat, RCat et CCat n'ont pas été raffinées, alors les ensembles de mêmes noms sont conservés dans le modèle concret.

En plus des interprétations formelles faites à la section 5.3.2, nous ajoutons l'interpretation de l'association abstraite $ACCESS$. Il s'agit par défaut, d'une relation entre la catégorie des sujets $SCat$ et l'ensemble formé par le produit cartésien des catégories des actions ($ACat$), des ressources ($RCat$) et des contextes ($CCat$).

– **(MA)** : $ACCESS \in SCat \Re (ACat \times RCat \times CCat)$

L'association ACCESS1, du modèle raffiné est interprétée comme suit :

– **(MR)** : $ACCESS1 \in SGroupes \Re (ACat \times RCat \times CCat)$

ACCESS1 est une relation entre les ensembles $SGroupes$ et l'ensemble constitué du produit cartésien des catégories $ACat$, $RCat$ et $CCat$. Pour indiquer concrètement que ACCESS1 est un raffinement de ACCESS nous définissons la propriété suivante :

– **(MR)** : $ACCESS1 \subseteq ACCESS$

Cette propriété est à l'image à celles utilisées pour rafiner la catégorie $SCat$ en $Sujets$, $Roles$ et $SGroupes$. Ici, du fait qu'il s'agit d'un raffinement s'association, notre objectif est d'indiquer que la création d'un nouvel élément avec $ACCESS1$ ne contredit pas la création d'un élément

avec $ACCESS$. Ainsi un élément créé avec $ACCESS1$ peut l'être avec $ACCESS$. ACCESS1 simule donc ACCESS. Cette forme d'interpretation est valable pour tous types d'associations dans CatBAC.

A cette étape de la modéliation de Pol_1, nous avons éffectué un premier raffinement afin de créer les catégories concrètes qui raffinent la catégories des sujets SCat, et toutes les nouvelles associations qui en découlent (Figure 5.10).

La Figure 5.11 montre un second niveau de raffinement visant à créer les catégories concrètes utilisées pour raffiner la catégorie des ressources RCat. Nous pouvons remarquer dans le second modèle de cette figure, que deux nouvelles catégories $Groupe\_Ress$ (groupe de ressources) et $Ressources$ ont fait leur apparition. Ces deux catégories raffinent la catégorie abstraite $RCat$. Cela conduit à l'interpretation formelle qui suit :

- $Groupe\_Ress \subseteq RCat \wedge Ressources \subseteq RCat$
- $Groupe\_Ress \neq Ressources$
- $var(Groupe\_Ress) = var(RCat)$

Une nouvelle association GRR (mise pour Groupe_Ress_Ressources) est également créée. Cette association permet d'affecter toutes les ressources concernées, au groupe de base ($Base$), comme cela est indiqué dans la politique Pol_1. La définition de l'association GRR est comme suit :

- $GRR \in (Groupe\_Ress)\ \Re\ (Ressources)$

Dans le même modèle, nous remarquons également que l'association n-aire $ACCESS1$ a laissé place à une nouvelle association $ACCESS2$. Cela est dû au fait que la catégorie $RCat$ est raffinée. Tout comme pour le premier raffinement avec $SCat$, le comportement de $RCat$ est maintenant simulé par $Groupe\_Ress$. Par conséquent $ACCESS1$ n'a plus la même signification dans le modèle actuel. Nous interpretons $ACCESS2$ comme suit :

- **(MR)** : $ACCESS2 \in SGroupes\ \Re\ (ACat \times Groupe\_Ress \times CCat)$
- $ACCESS2 \subseteq ACCESS1$

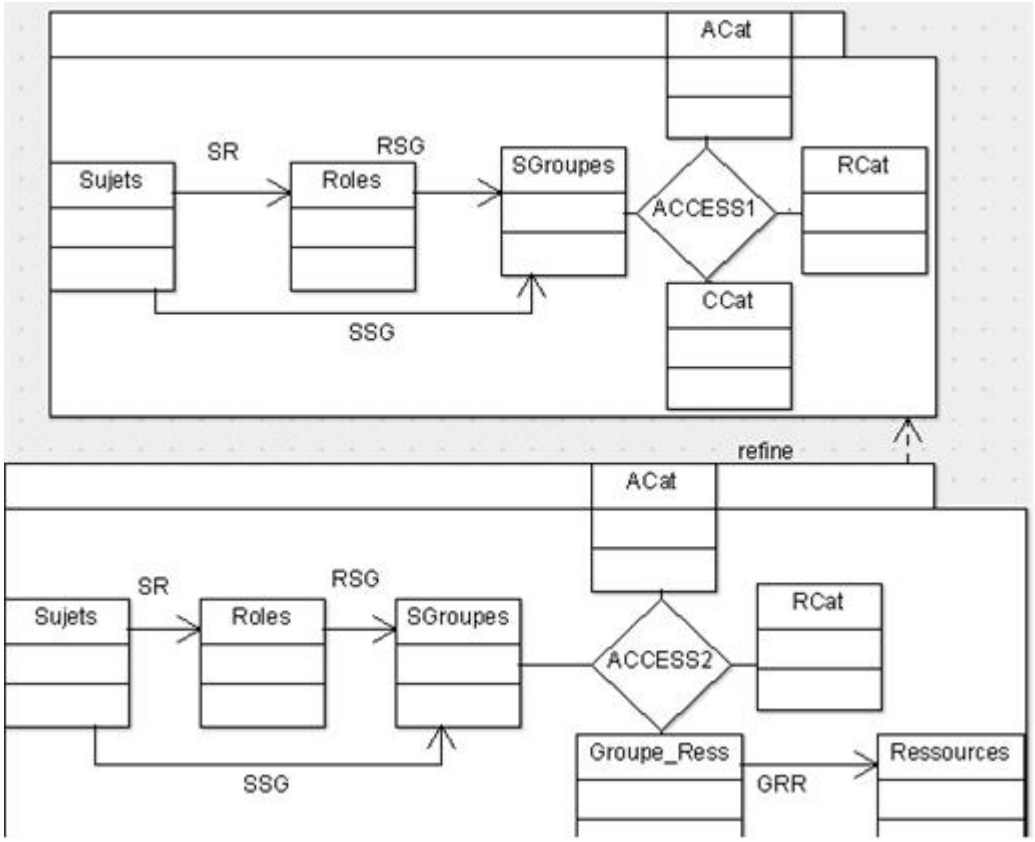

FIGURE 5.11 – Second raffinement de modèle

Conclusion

A cette nouvelle étape, nous venons de creer le modèle CatBAC (second modèle de la Figure 5.11) qui répond au mécanisme d'accès de la politique Pol_1 (Figure 5.5). Nous avons également accompagné le processus de raffinement de ce modèle, d'une interpretation formelle qui nous permettra plus tard de construire le modèle B-événementiel équivalent. Les interpretations que nous avons énoncé sont inspirées des travaux de la section 5.2. Toutefois pour utiliser ce modèle nous devons prendre en compte certaines exigences

qui ne sont pas mentionnées dans la politique de sécurité Pol_1. Nous y reviendrons à la section 5.4.

5.3.5 Proposition de règles de raffinement

A partir des modélisations et interprétations que nous avons fait de la politique Pol_1, nous pouvons élaborer trois règles générales pour obtenir les équivalents formels des modèles à raffiner. La première règle est déduite du raffinement des catégories. Les deux autres proviennent du raffinement des associations.

Première règle de raffinement

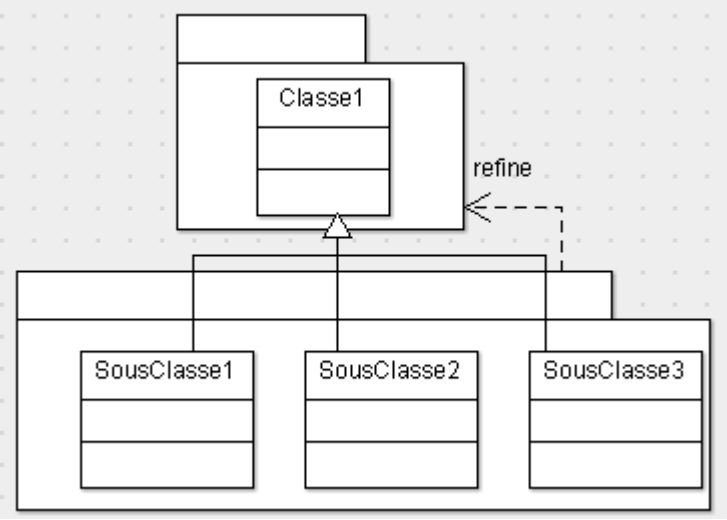

FIGURE 5.12 – Première règle de raffinement

Cette règle est présentée à la Figure 5.12. Elle s'applique lorsque l'on veut raffiner une catégorie en une ou plusieurs nouvelles catégories. Dans la Figure 5.12, La catégorie $Classe1$ est raffinée en trois nouvelles catégories $SousClasse1$, $SousClasse2$ et $SousClasse3$.

L'interprétation de ce raffinement est ci-après. Nous désignons par **MA** un modèle abstrait et par **MR** un modèle raffiné.

– **(MA)** : $Classe1$
– **(MR)** : $SousClasse1 \subseteq Classe1 \wedge SousClasse2 \subseteq Classe1$
– **(MR)** : $\wedge\ SousClasse3 \subseteq Classe1$
– **(MR)** : $SousClasse1 \neq SousClasse2$
– **(MR)** : $SousClasse1 \neq SousClasse3$
– **(MR)** : $SousClasse2 \neq SousClasse3$

Cette interprétation indique que les nouvelles catégories sont des parties distinctes de la catégorie abstraite.

Il faut ensuite indiquer la catégorie raffinée qui a le même comportement que la catégorie abstraite ; ici nous avons choisi $SousClasse3$:

– **(MR)** : $var(SousClasse3) = var(Classe1)$

Cette règle a été utilisée pour raffiner les catégories $SCat$ et $RCat$ de la politique Pol_1 (Figures 5.10 et 5.11).

Seconde règle de raffinement

Cette règle est présentée à la Figure 5.13. Elle concerne le raffinement des associations. On y distingue deux modèles dont le second raffine le premier. L'objectif de cette règle est de raffiner l'association $rel1$ qui relie les deux classes $Classe1$ (dont le raffinement produit la classe $SousClasse1$) et $Classe2$, sachant que la nouvelle association raffinée devra relier les classes $SousClasse1$ et $Classe2$.

Les conditions d'application de cette règle sont :
– $rel1 \in Classe1\ \Re\ Classe2$
– $SousClasse1 \subseteq Classe1$

Ces conditions correspondent à la description formelle de la situation qui se présente dans le premier modèle de la Figure 5.13.

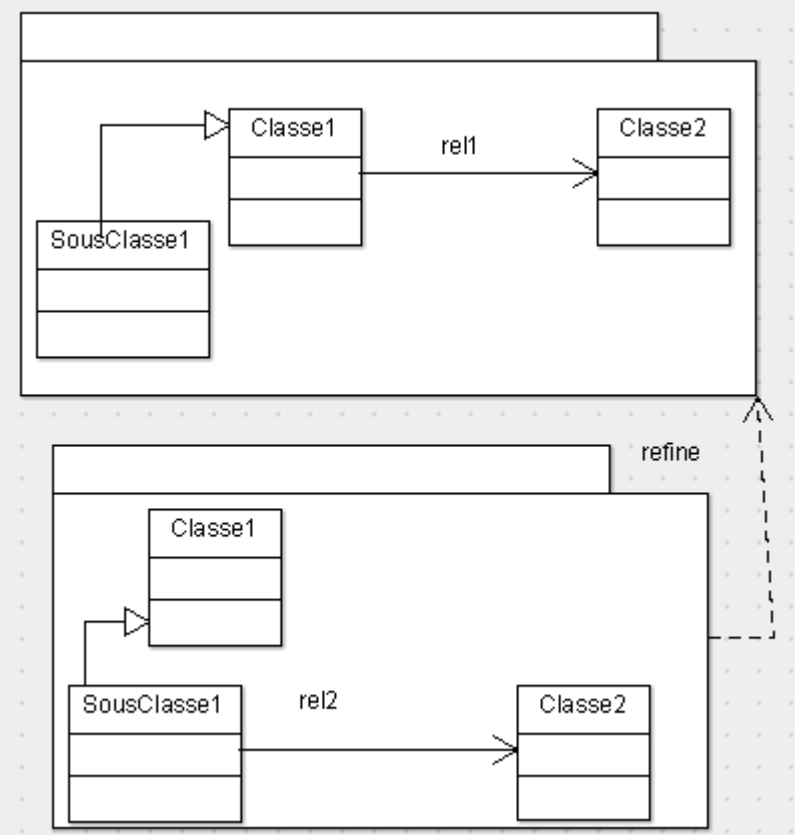

FIGURE 5.13 – Seconde règle de raffinement

Les propriétés du raffinement de cette règle sont :

– $rel2 \in SousClasse1 \Re Classe2$
– $rel2 \subseteq rel1$

L'expression $rel2 \subseteq rel1$ indique que $rel1$ à été raffinée en $rel2$. Cette règle a été appliquée aux Figures 5.10 et 5.11 pour raffiner les associations $ACCESS$ et $ACCESS1$ (Confère les interpretations formelles correspondant à ces deux figures). Pour l'illustrer il suffit de remplacer $ACCESS$ par $Rel1$ et $ACCESS1$ par $Rel2$ pour la Figure 5.10.

Troisième règle de raffinement

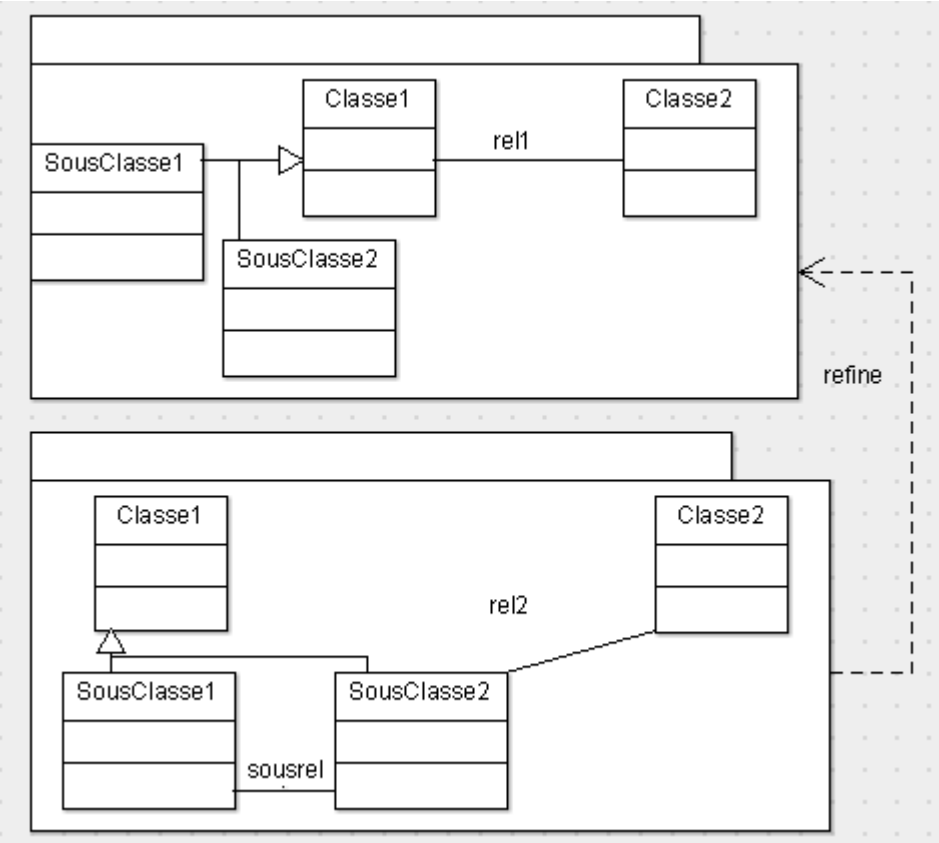

FIGURE 5.14 – Troisième règle de raffinement

Cette règle de raffinement est présentée à la Figure 5.14. Cette règle est une extension de la seconde règle de raffinement (Figure 5.13). Les modèles qui y sont décrits sont similaires à ceux de la seconde règle sauf que, cette fois, la classe $Classe1$ est raffinée en deux sous-classes $SousClasse1$ et $SousClasse2$. Cette règle peut s'appliquer pour un plus grand nombre de sous-classes. L'objectif est de montrer la façon dont le raffinement doit être traité si une classe abstraite est raffinée en plusieurs sous-classes dont certaines sont reliées par de nouvelles associations.

Les conditions d'application de la règle sont :

- $rel1 \in Classe1 \Re Classe2$
- $(SousClasse1) \subseteq (Classe1)$
- $(SousClasse2) \subseteq (Classe1)$
- $(SousClasse1) \neq (SousClasse2)$

Ces conditions correspondent à la description formelle du premier modèle de la figure.

Les propriétés du raffinement sont :

- $var(Classe1) = var(SousClasse2)$
- $rel2 \in (SousClasse2) \Re (Classe2)$
- $rel2 \subseteq rel1$
- $sousrel \in (SousClasse1) \Re (SousClasse2)$

Les deux nouvelles classes $SousClasse1$ et $SousClasse2$ créées par raffinement entraînent dans ce cas particulier, la création d'une association *sousrel* qui les relie. Ici la création de la relation *sousrel* est voulue par l'utilisateur. Dans la mesure ou ce n'est pas le cas, on n'a pas besoin de spécifier cette relation.

Application des règles de raffinement

Pour visualiser l'application de chaque règle de raffinement, il suffit de remplacer chaque élément du modèle de la Figure 5.15 par son correspondant dans chaque règle de raffinement. Pour chaque règle, le nom SCi devant les catégories est mis pour $SousClassei$.

Si nous nous référons à l'interprétation formelle de ce raffinement (Section 5.3.4, Figure 5.10), nous remarquerons que toutes les propriétés obtenues pour ce modèle sont fournies par les trois règles. Pour les obtenir, il suffit de remplacer chaque valeur de catégorie utilisée par son équivalent dans les interprétations formelles de chaque règle. Par exemple pour la première règle il suffit de remplacer SCat par Classe1, Sujets par SousClasse1, Roles par SousClasse2 et SGroupes par SousClasse3.

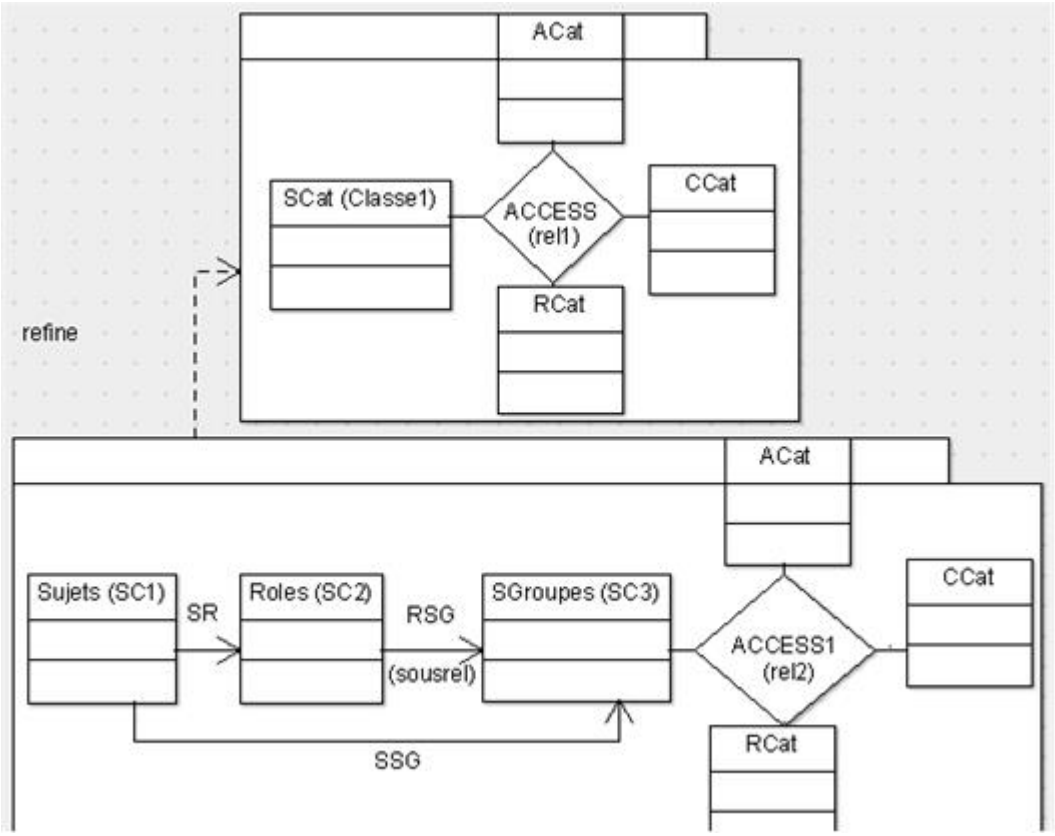

FIGURE 5.15 – Application des règles de raffinements

5.3.6 Conclusion préliminaire

Dans cette section, nous avons interprété formellement les modèles Cat-BAC et élaboré trois règles pouvant aider à cette fin. Néanmoins, avant d'étudier la traduction vers B-événementiel, nous allons voir comment le modèle final de la politique Pol_1 peut être raffiné pour modéliser des règles de contrôle d'accès. Nous verrons que toutes les règles de contrôle d'accès de la politique Pol_1 ne sont pas forcément représentatives du modèle que nous avons construit pour la politique. Des solutions à cette difficulté de modélisation seront fournies au chapitre suivant.

5.4 Modélisation des règles de contrôle d'accès

Considérons les règles de contrôle d'accès (1) et (3) de la politique Pol_1 :

- (1) Tous les membres du laboratoire, peuvent accéder à tous les imprimantes, téléphones et scanners du laboratoire. Afin de faciliter l'accès à ces ressources de base, le laboratoire a décidé de les inclure dans un groupe appelé *Base*. Ainsi, toutes les ressources de base seront directement affectées à ce groupe de ressources. Le groupe de ressources *Base* est le seul autorisé par la politque.
- (3) Seuls les étudiants et responsables de laboratoire ont le droit d'accéder au serveur *Biblio_1*, qui donne accès à des articles scientifiques dont la consultation régulière est payante (l'université subventionne la consultation de ces articles par ses étudiants et son personnel régulier).

En utilisant la règle (1), modélisons le fait qu'un sujet qui est étudiant ou non et qui appartient au groupe *GRSI* a la droit d'accéder à la ressource de base téléphone1. La Figure 5.16 présente le raffinement de cette règle. Un point (.) devant un nom de catégorie, indique que cette catégorie ne peut plus être raffinée. On remarque sur cette figure que le raffinement de la règle à consisté à instancier toutes les catégories concernées du modèle de la politique Pol_1. On remarque aussi que la règle de contrôle d'accès modélisée est exactement représentative du modèle construit pour la politique.

Utilisons maintenant la règle (3) pour modéliser le fait qu'un sujet qui est étudiant et qui appartient au groupe *GRSI* a le droit d'accéder à la ressource *Biblio_1*. D'apres la politique Pol_1, la ressource *Biblio_1* n'est pas une ressource de base. Par conséquent, elle n'est affectée à aucun groupe de ressources. A partir du modèle construit pour la politique pol_1, nous ne pouvons pas automatiquement raffiner cette règle.

Pour que cette règle soit représentative du modèle construit pour Pol_1, nous sommes obligés de la modéliser comme indiqué à la Figure 5.17. Vu que le modèle construit pour la politique Pol_1, indique que les accès aux ressources se font par l'intermédiaire des groupes de ressources (*Groupe_Ress*),

nous avons créé un groupe virtuel de ressources appelé (*.Autre*). Le rôle de ce groupe est de rassembler toutes les ressources qui ne doivent pas être affectées au groupe de base (*Base*).

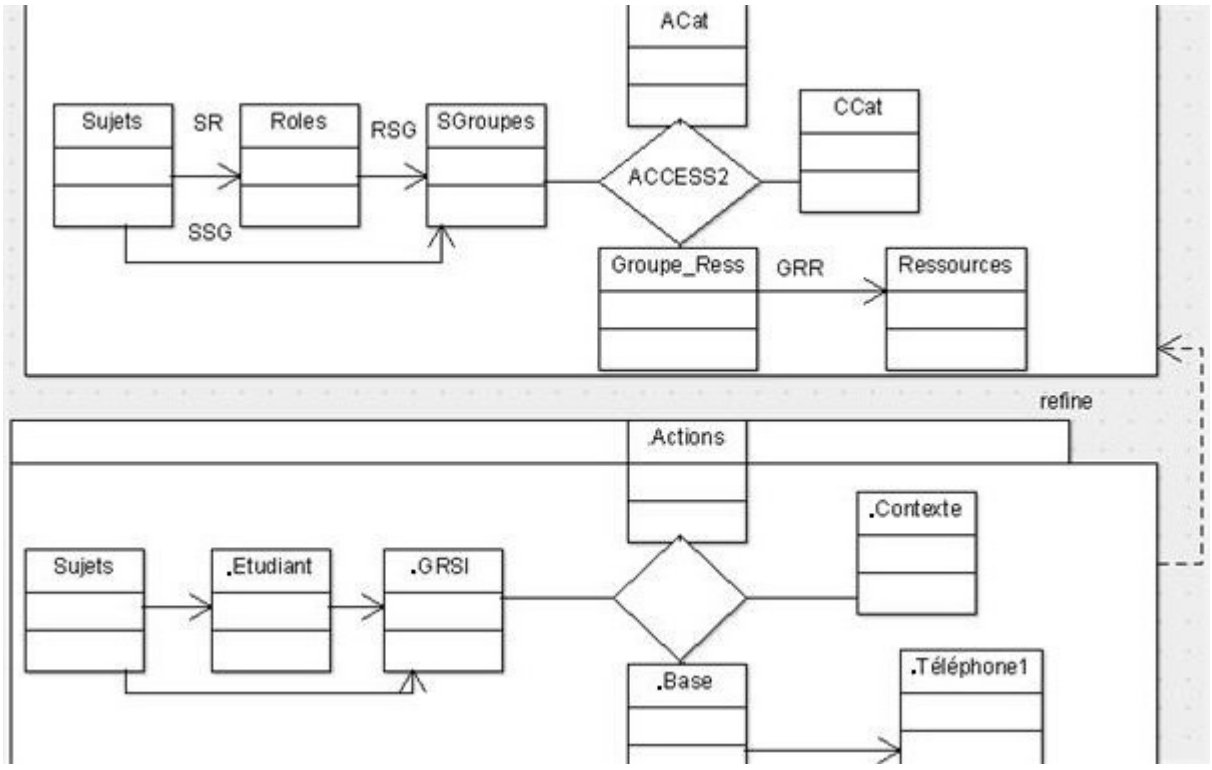

FIGURE 5.16 – Un raffinement de la règle (1)

Cepandant la politque Pol_1 exprime clairement le fait que seul le groupe *Base* est autorisé. Par conséquent le fait de créer le groupe *Autre* ne permet pas d'exprimer fidèlement les besoins de la politique. Toutefois cette solution peut être acceptable jusqu'a un certain point.

En effet, en analysant la politique, on peut considérer que si certaines ressources devaient être affectées à des groupes, alors toutes celles qui ne le seraient pas, pourront être vues comme appartenant à un groupe par défaut différent des autres groupes. Ainsi dès qu'une nouvelle ressource est

créée dans le système, elle appartient automatiquement au groupe par défaut jusqu'a ce qu'on lui affecte un groupe explicitement défini par la politique.

Ce type de raisonnement peut être utilisé pour justifier l'utilisation des catégories virtuelles afin de modéliser des règles de contrôle d'accès qui ne sont pas représentatives du modèle d'une politique construit avec CatBAC.

Toutefois si le nombre de catégories virtuelles augmente (Figure 5.18), la modélisation des règles de contrôle d'accès n'est plus très adéquate car on s'éloignerait de plus en plus de la spécification originale de la politique.

En effet, en gardant à l'esprit que l'objectif d'une modélisation graphique est de faciliter la lecture est la compréhension des modèles, l'introduction de plusieurs catégories virtuelles pourrait rendre un modèle difficile à comprendre et par conséquent à utiliser par différents groupes de personnes ayant connaissance des exigences de la politique de sécurité. Par exemple une personne qui sait que l'utilisation du groupe *Autre* n'est pas complètement conforme aux besoins de la politique de sécurité, pourait avoir des difficultés avec notre modélisation.

Notre objectif étant de construire des modèles façiles à comprendre et à utiliser, nous allons éviter l'utilisation des catégories virtuelles. De cette façon, toutes les règles de contrôle d'accès seront représentatives du modèle de la politque. De même, toutes les opérations d'affectations de catégories se feront uniquement entre les catégories déclarées dans la politique.

Nous proposons donc de formaliser une technique de modélisation par décomposition des mécanismes d'accès. De cette façon, le modèle de la politique de sécurité sera constitué d'un ensemble de sous-modèles à partir desquels chaque règle de contrôle d'accès sera explicitement formalisée.

Pour la vérification globale des exigences de la politique, chaque sous-modèle sera traduit en B-événementiel en respectant les règles d'interprétations formelles que nous avons défini dans ce chapitre. Lorsque les sous-modèles B-événementiel seront obtenus, ils pourront au besoin être composés en un modèles unique à l'aide d'outils existants (chapitre 7). Notre approche de modélisation sera expliquée au chapitre 6.

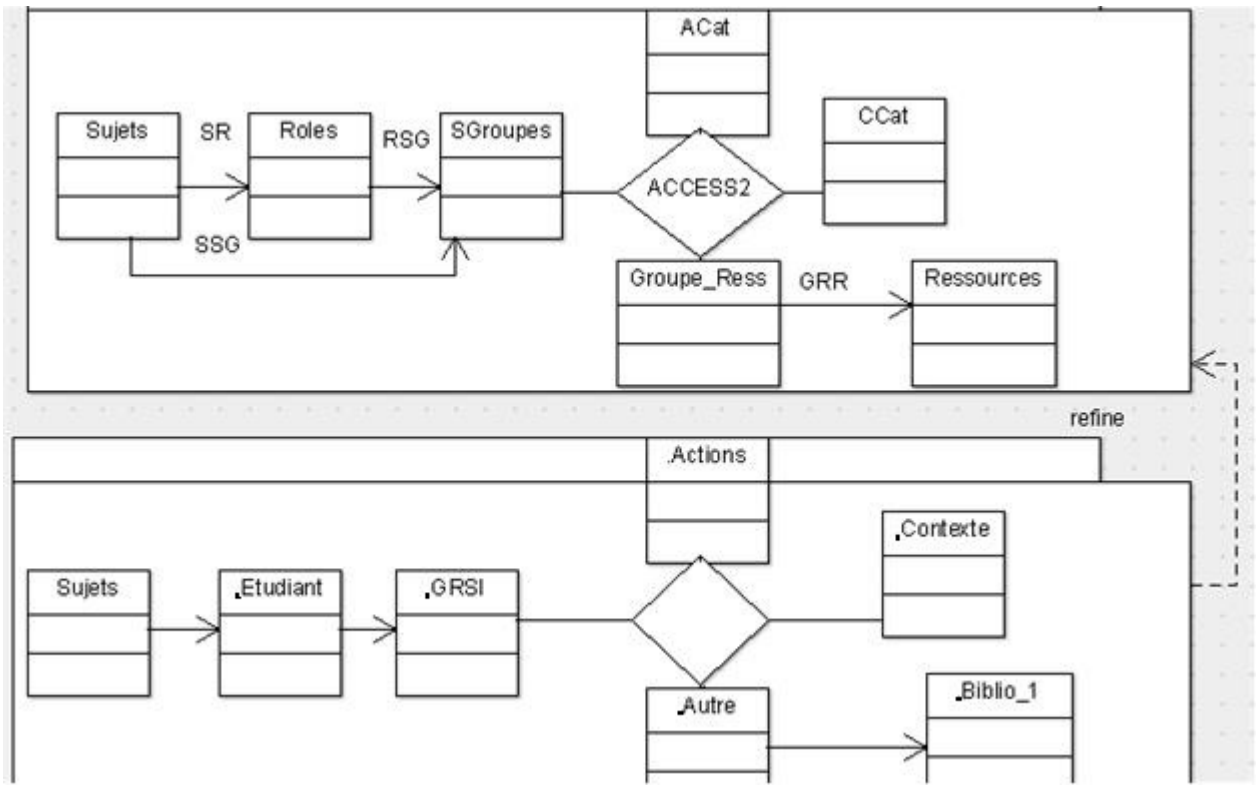

FIGURE 5.17 – Un raffinement de la règle (3)

5.5 Conclusion

Dans ce chapitre, nous avons présenté une première partie de notre contribution de recherche. Nous avons montré comment le raffinement des modèles CatBAC peut être formellement interprété, et nous avons proposé trois règles permettant d'obtenir les expressions formelles de ces raffinements de modèles. Nous avons ensuite montrer à travers l'exemple de la politique Pol_1, que le modèle final construit pour cette politique peut causer des difficultés d'utilisations. Pour y remédier, nous proposons de formaliser au chapitre suivant, une technique de modélisation par décompositions des mécanismes d'accès des politiques, avec CatBAC et B-événementiel.

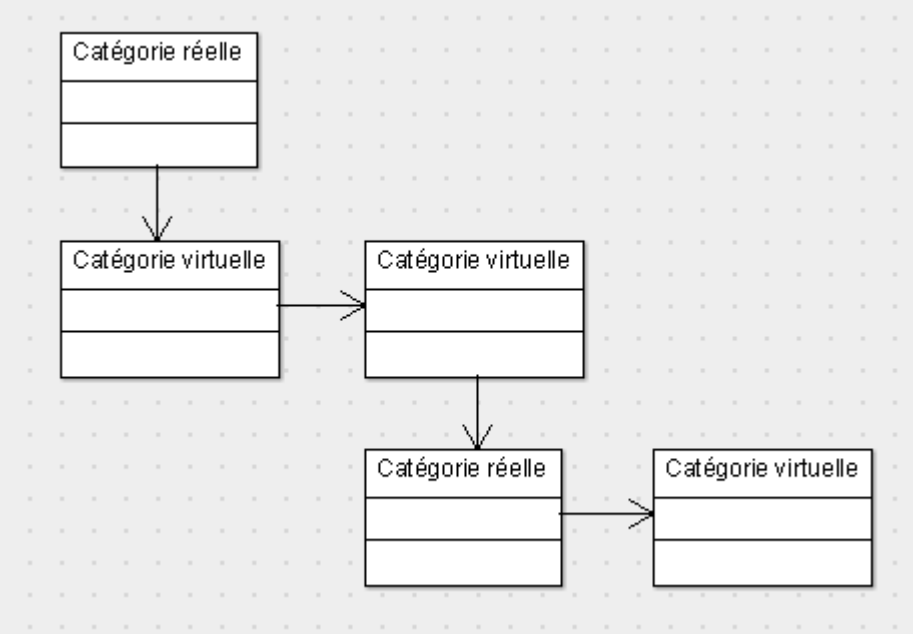

FIGURE 5.18 – Utilisation de plusieurs catégories virtuelles

Chapitre 6

Modélisation par décomposition des systèmes de contrôle d'accès avec CatBAC et B-événementiel (contributions)

6.1 Introduction

Dans le chapitre précédent, nous avons montré que la modélisation d'un système à partir de son mécanisme d'accès global, peut entraîner la création de catégories virtuelles. Cette situation est due au fait que dans une politique de sécurité, toutes les catégories d'éléments ne sont pas forcément sollicitées pour donner un accès. Par conséquent, pour être conforme au mécanisme d'accès le plus large (mécanisme qui sollicite toutes les catégories), on a besoin de définir des catégories virtuelles pour remplacer les catégories manquantes. Dans cette situation, il faut expliquer aux utilisateurs le sens de chaque catégorie virtuelle afin que ces derniers puissent analyser la politique autrement que dans son sens direct (chapitre 5). Par conséquent si le

nombre de catégories virtuelles augmente, l'utilisation des modèles CatBAC par des personnes ayant connaissance des exigences de la politique concernée pourrait être problématique. Pour ces personnes, cela implique que plusieurs éléments qui n'ont pas été définis dans les spécifications de la politique seront introduits dans le système final. Dans cette situation, nos modèles pouraient être considérés comme non conformes aux besoins de la politique.

Pour éviter cela, nous formalisons une méthode de modélisation par décompositions des mécanismes d'accès. A partir de cette méthode nous décomposons en sous-mécanismes, le mécanisme global d'une politique de sécurité avant de modéliser ces sous-mécanismes avec CatBAC. De cette façon, des opérations spécifiques pouront être modélisées pour prendre en charge l'absence de certaines catégories et éviter l'utilisation des catégories virtuelles.

Dans ce chapitre, nous montrons la façon dont la décomposition des mécanismes peut être éffectuée. La notion de décomposition des systèmes a déja été introduite dans différents travaux de modélisations. Dans la modélistion avec B-événementiel, la décomposition [14, 15, 6] a été introduite pour réduire la taille des systèmes à raffiner et par conséquent faciliter le processus de raffinement. Pour nous, l'objectif de la décomposition est de réduire la taille des systèmes à raffiner (comme dans B-événementiel), de guider la création des systèmes de contrôle d'accès de façon à ce que chaque règle de contrôle d'accès soit représentative du modèle d'accès et surtout, d'éviter l'utilisation des catégories virtuelles. Une fois les sous-mécanismes modélisés avec CatBAC, ils seront traduits en B-événementiel puis recomposés si nécéssaire. Dans les sections qui suivent nous présentons notre approche.

6.2 Présentation de notre approche

Considérons la politique Pol_1 présentée dans le chapitre précédent. Cette politique est composée des deux mécanismes d'accès présentés dans les Figures 6.1 et 6.2. Dans chacune de ces figures, la succession de flêches en traits pleins désigne le mécanisme d'accès valide. Nous remarquons pour ces deux mécanismes, que toutes les catégories ne sont pas sollicitées. Le mécanisme de la Figure 6.1 indique que des sujets sont affectés à des rôles (Roles)

ou à des groupes de sujets (SGroupes). Des rôles sont ensuite affectés à des groupes de sujets (SGoupes).

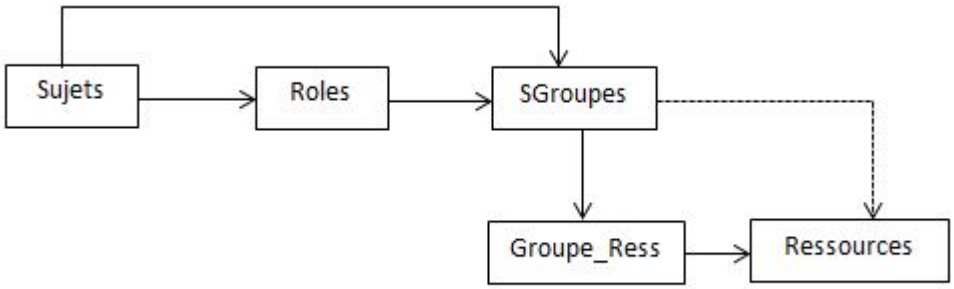

FIGURE 6.1 – Premier mécanisme d'accès

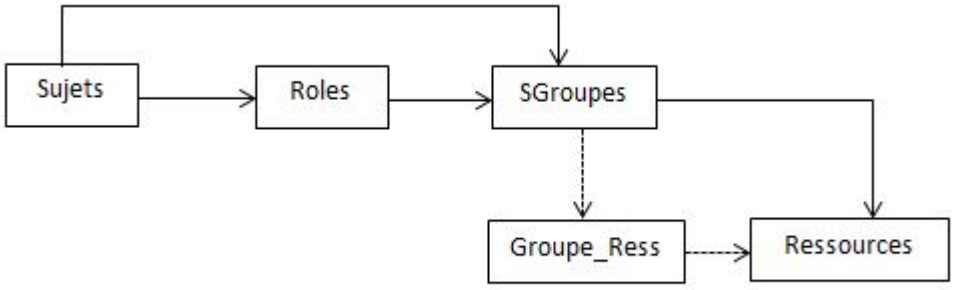

FIGURE 6.2 – Second mécanisme d'accès

Certaines ressources (Ressources) du système sont rassemblées par groupes de ressources (Groupe_Ress). Les groupes de ressources (Groupe_Ress) sont ensuite affectés aux groupes de sujets (SGroupes). Ainsi les sujets qui sont affectés à des rôles ou à des groupes de sujets peuvent accéder aux ressources rassemblées dans des groupes de ressources.

Le mécanisme de la Figure 6.2 est similaire sauf que dans ce cas, les ressources sont directement accessibles à partir des groupes de sujets.

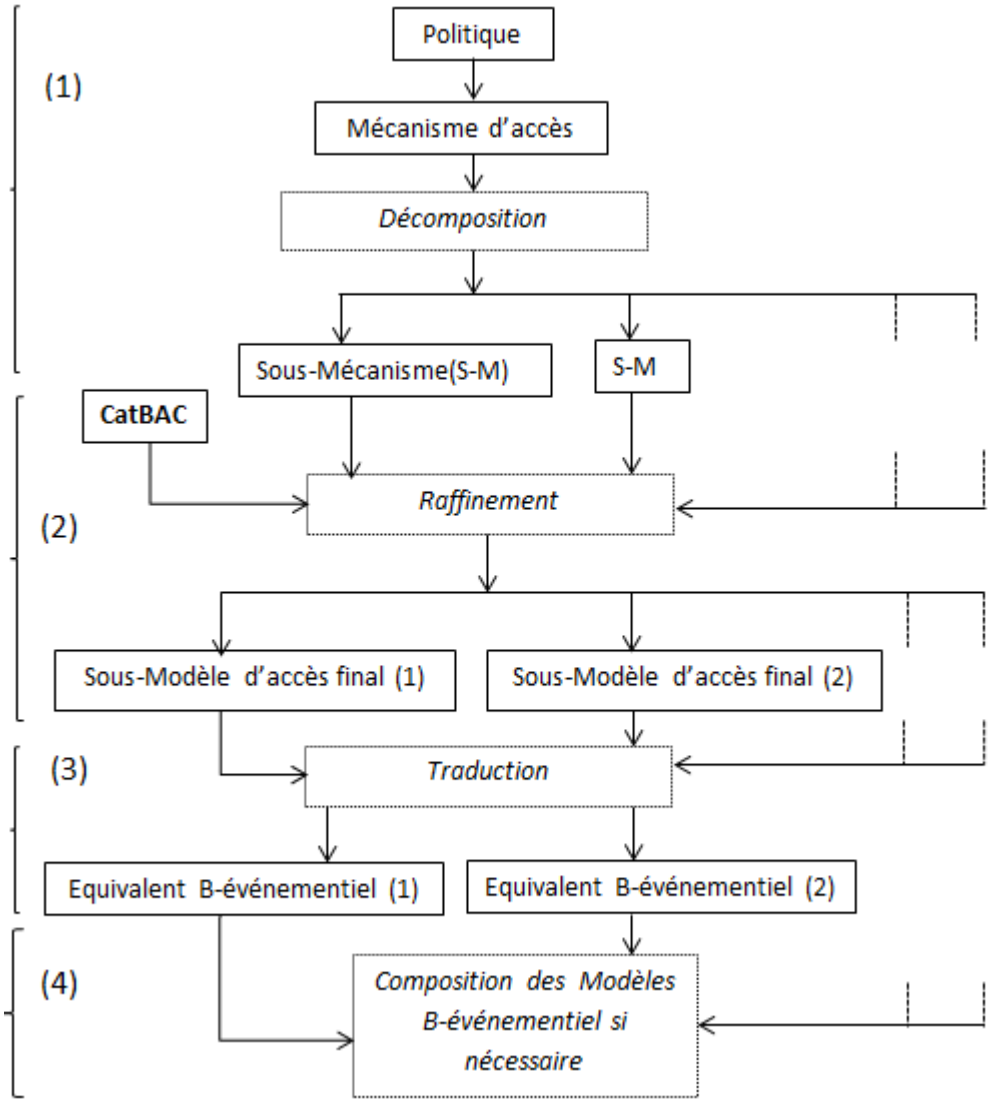

FIGURE 6.3 – Modélisation par décomposition avec CatBAC et B-événementiel

Pour modéliser par décomposition, nous considérons qu'une politique de sécurité qui a pour objectif un contrôle d'accès est constituée d'un mécanisme d'accès qui determine la façon dont les sujets accèdent à leurs permissions. Suivant les objectifs recherchés, le mécanisme d'accès de la politique peut être décomposé en plusieurs sous-mécanismes. Chaque sous-mécanisme sera modélisé par raffinement avec CatBAC et traduit en B-événementiel. Si besoin est, les modèles B-événementiel correspondant à chaque sous-mécanisme, peuvent àpres vérifications et corrections, être recomposés dans un modèle B-événementiel unique à condition de respecter certaines contraintes. La Figure 6.3 présente la procédure à suivre. Dans cette figure, chaque numéro placé devant chaque accolade (à gauche de la figure) correspond à un processus précis.

Etape 1

Cette étape correspond à l'accolade (1). Il s'agit de l'extraction et de la décomposition des mécanismes d'accès d'une politique. Pour ce faire, on commence par considérer une politique de sécurité dans sa globalité. A partir de cette politique, on extrait un mécanisme d'accès particulier. Ce mécanisme d'accès est ensuite décomposé suivant des objectifs précis, ou en fonction des règles de contrôle d'accès qu'on voudrait modéliser avec précision. A la fin de cette étape on obtient les sous-mécanismes d'accès (Sous-Mécanisme(S-M)) à modéliser. Cette étape de la modélisation se fait manuellement. Notons qu'il n'est pas nécéssaire d'extraire le mécanisme global de la politique avant de le décomposer. On peut extraire directement chaque sous-mécanisme à partir des règles de contrôle d'accès.

Etape 2

Elle correspond à la seconde accolade (2) de la Figure 6.3. Cette étape présente la modélistion par raffinement de chaque sous-mécanisme d'accès avec CatBAC. A cette étape, toutes les définitions du raffinement que nous avons défini au chapitre 5 s'appliquent. A la fin, on obtient des modèles

CatBAC correspondant à chaque sous-mécanisme de l'étape (1). Pour cette étape, nous avons développé un prototype d'outil permettant de guider l'utilisateur dans ces modélisations.

Etape 3

A l'étape (3) (accolade (3)), les modèles CatBAC produits à l'étape (2) sont traduits automatiquement vers B-événementiel grâce au prototype d'outil que nous avons développé. L'objectif de cette partie est de permettre aux utilisateurs de modéliser des propriétés de sécurité dans les modèles et de vérifier que ces propriétés ne sont pas violées par les modèles générés.

Etape 4

Dépendamment des objectifs, l'étape (4) pourrait être sollicitée. Elle vise à s'assurer que si tous les sous-modèles B-événementiel sont regroupés dans un modèle unique, alors il n'existe pas de contradictions entre les divers propriétés de chaque sous-modèle. La plateforme Rodin pour laquelle nous avons développé notre outil de modélisation propose des outils [14, 15] (Chapitre 7) qui permettent de composer différents modèles B-événementiel sous certaines conditions.

6.2.1 Exemple de décomposition

Appliquons la procédure mentionnée précédemment, pour modéliser la politique Pol_1 définie au chapitre 5. Le mécanisme d'accès global de cette politique est présenté à la Figure 6.4. Pour cette politique, notre objectif de décomposition est de regrouper les sujets et les ressources du système. Nous allons par conséquent produire les sous-mécanismes suivants :

1. Des sujets (Sujets) sont affectés à des rôles (Roles) ; les rôles sont affetés à des groupes de sujets (SGroupes) ; des groupes de sujets sont affectés à des groupes de ressources (Groupe_Ress) ; des ressources

(Ressources) sont affectées à des groupes de ressources (Groupe_Ress) (Figure 6.5).

2. Des sujets sont affectés à des groupes de sujets ; des ressources sont affectées à des groupes de ressources ; des groupes de ressources sont affectés à des groupe de sujets (Figure 6.6).
3. Figure 6.7 : des sujets sont affectés à des rôles, les rôles sont affectés aux groupes de sujets, et les ressources sont directement affectées aux groupes de sujets.
4. Figure 6.8 : des sujets sont affectés à des groupes de sujets et des ressources sont directement affectées à ces groupes de sujets.

Nous pouvons remarquer que les sous-mécanismes des Figures 6.5 et 6.6 sont issus de la décomposition du sous-mécanisme de la Figure 6.1. De même les sous-mécanismes des Figures 6.7 et 6.8 sont issus de la décomposition du sous-mécanisme de la Figure 6.2. Rappelons que suivant les objectifs visés, les décompositions auraient pu être différentes. Par exemple on aurrait pu conserver les sous-mécanismes des Figures 6.1 et 6.2 comme décomposition finale du mécanisme de la politique Pol_1. Ici nous avons décidé d'exprimer explicitement chaque sous-mécanisme.

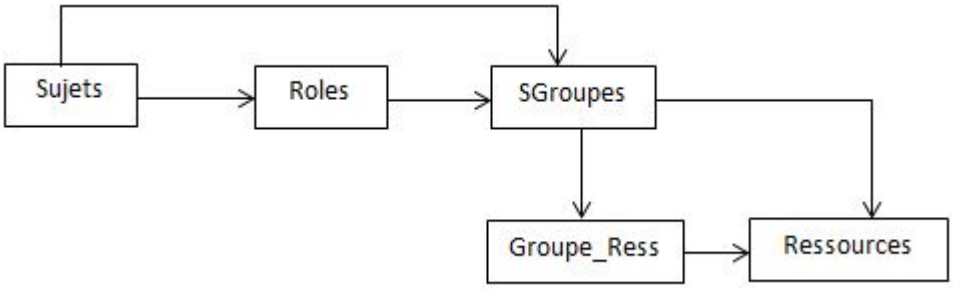

FIGURE 6.4 – Mécanisme d'accès global

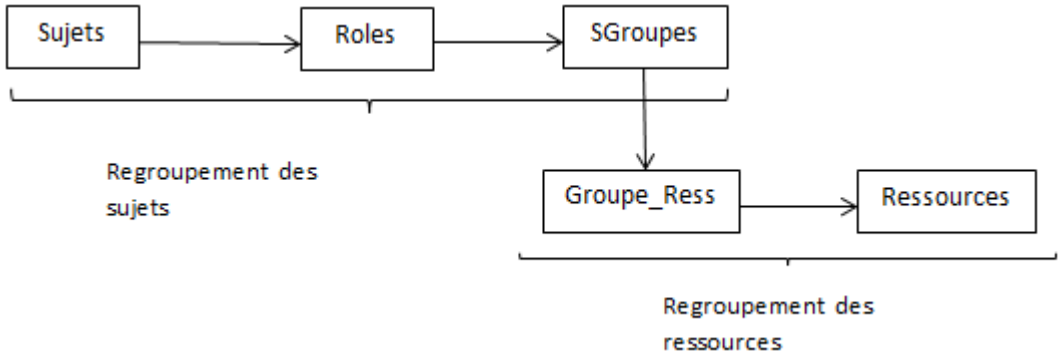

FIGURE 6.5 – Sous-mécanisme d'accès 1

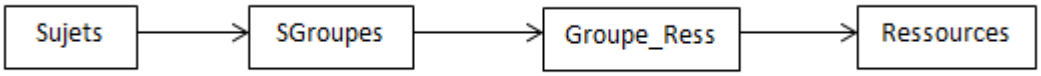

FIGURE 6.6 – Sous-mécanisme d'accès 2

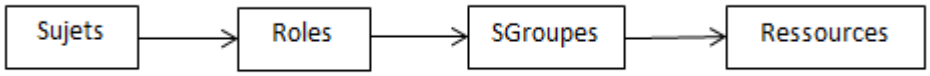

FIGURE 6.7 – Sous-mécanisme d'accès 3

FIGURE 6.8 – Sous-mécanisme d'accès 4

Lors de la décomposition d'un mécanisme d'accès, il faut également faire attention à ne pas générer trop de sous-mécanismes, en évitant les décompositions inutiles. De cette façon on évite d'avoir trop de sous-modèles CatBAC. Une fois la décomposition du mécanisme d'accès de la politique terminée, Nous passons à la seconde étape qui est la modélisation de chaque sous-mécanisme avec CatBAC. Cette partie à déja été traitée dans le chapitre 5. Ici nous allons ajouter des contraintes qui auront éssentiellement pour but de faciliter la recomposition (fusion) des modèles B-événementiel correspondant à chaque sous-mécanisme.

6.2.2 Contraintes de modélisation avec CatBAC

Le mécanisme d'accès de la politique Pol_1 met en évidence 5 catégories d'éléments CatBAC : la catégorie des Sujets (Sujets), la catégorie des rôles (Roles), la catégorie des groupes de sujets (SGroupes), la catégorie des groupes de ressources (Groupe_Ress) et la catégorie des ressources (Ressources). Nous ajoutons la catégorie des contextes pour nos illustrations. Ces catégories ne diffèrent pas d'un sous-mécanisme à l'autre. En d'autres termes ce sont les mêmes catégories qui sont partiellement ou entièrement représentées dans chaque sous-mécanisme.

Pour faciliter la composition future des modèles B-événementiel, nous préconisons d'utiliser les mêmes identifiants (ici mêmes noms) dans chaque sous-modèle CatBAC, pour désigner les catégories qui sont identiques. Par conséquent si nous utilisons le nom "Sujets", pour désigner la catégorie des sujets d'un sous-modèle (1), nous ne pouvons pas utiliser un nom différent pour désigner le même ensemble de sujets dans un autre sous-modèle. Cela éviterait de créer deux ensembles de sujets différents qui sont en réalité les mêmes pour le système.

De même que pour les catégories, les identifiants des associations qui relient les mêmes éléments d'un sous-modèle doivent être conservés. Si dans un sous-modèle une association "SR" relie la catégorie des sujets à la catégorie des rôles, ce nom doit être conservé pour une association qui joue la même fonction dans un autre sous-modèle. Néanmoins des exceptions sont

possibles. Si par exemple dans le futur système à implémenter, on voudrait modéliser deux opérations différentes pour affecter des sujets, à un rôle *Etudiant* dans la première opération, puis un sujet à un rôle *Resp_lab* dans la seconde opération (voir Pol_1 chapitre 5), on doit utiliser des identifiants d'associations différents, car ces deux opérations ont des objectifs différents. Nous verrons l'usage de ces contraintes plus loin dans ce chapitre.

Les Figures 6.9, 6.10, 6.11 et 6.12 montrent les modèles CatBAC qu'on peut produire pour chaque sous-mécanisme obtenu à la Section 6.2.1. Dans les modèles de ces figures, nous remarquons que nous n'avons pas besoin d'introduire des catégories virtuelles. De ce fait chaque règle de contrôle d'accès est directement déduite du sous-modèle d'accès CatBAC correspondant.

Le Figure 6.9 présente le sous-modèle final (Premier modèle de la figure) correspondant à la modélisation par raffinement du sous-mécanisme de la Figure 6.5. Une fois ce modèle obtenu, nous avons ajouté un niveau de raffinement supplémentaire pour extraire une règle R2 de la politique Pol_1 comme suit : *Tous les membres du laboratoire peuvent accéder à la ressource de base Téléphone1* (voir Pol_1 chapitre 5).

La Figure 6.10 montre le modèle final du sous-mécanisme correspondant à la Figure 6.6 avec une représentation de la règle R2.

Nous remarquons dans les Figures 6.9 et 6.10 que l'association ACCESS2 a la même signification. Elle associe des groupes de sujets (SGroupes) à l'ensemble formé par le produit cartésien des ensembles ACat, Groupe_Ress et CCat. Par conséquent nous avons conservé le même nom pour cette association. Le fait d'utiliser cette convention facilitera la fusion des sous-modèles B-événementiel que nous allons générer plus tard.

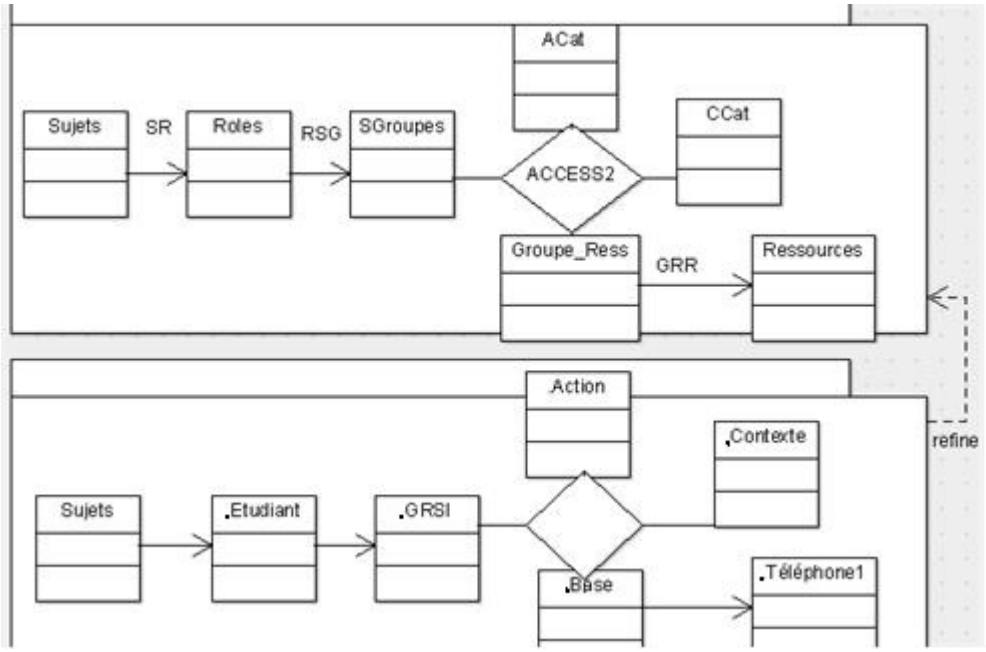

FIGURE 6.9 – Premier sous-modèle d'accès avec la règle R2 (Figure 6.5)

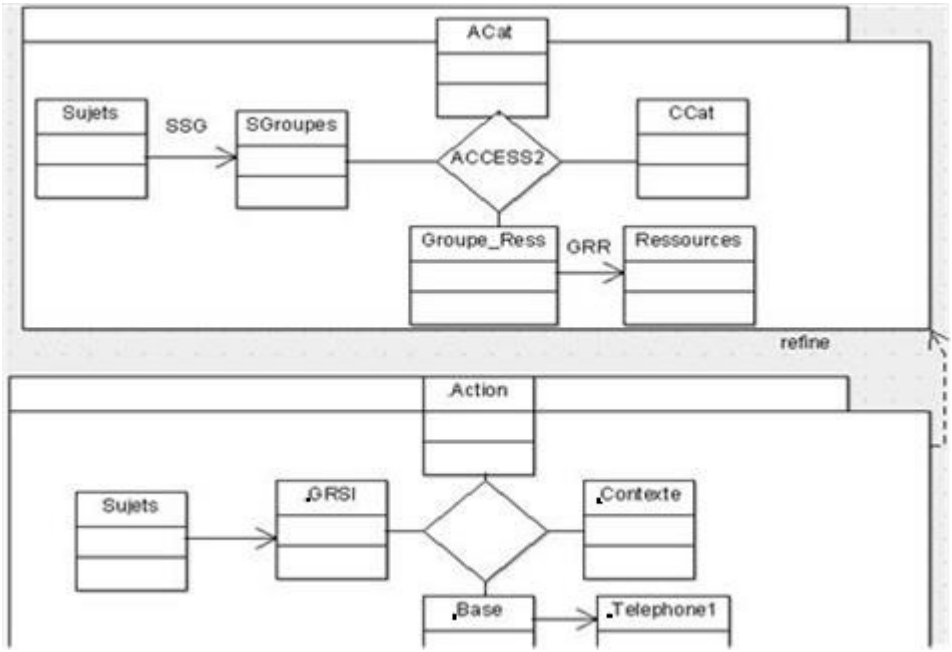

FIGURE 6.10 – Second sous-modèle d'accès avec la règle R2 (Figure 6.6)

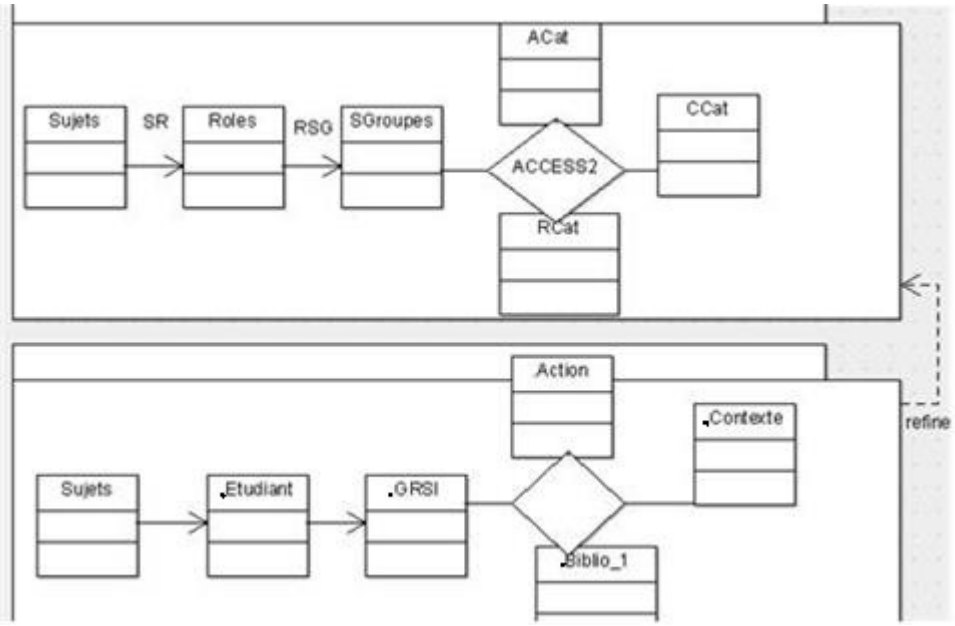

FIGURE 6.11 – Troisième sous-modèle d'accès avec la règle R3 (Figure 6.7)

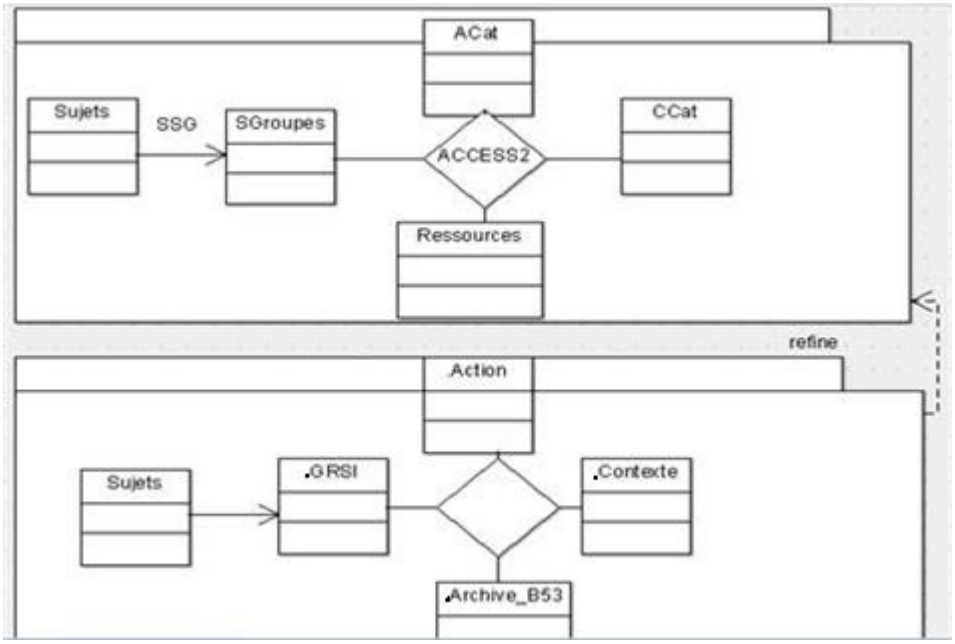

FIGURE 6.12 – Quatrième sous-modèle d'accès avec la règle R4 (Figure 6.8)

La Figure 6.11 présente la modélisation par raffinement du sous-mécanisme présenté à la Figure 6.7. Nous avons modélisé dans cette même figure la règle de contrôle d'accès R3 : *seuls les étudiants et responsables de laboratoire ont le droit d'accéder au serveur Biblio_ 1*. Nous pourrons remarquer que contrairement à la modélisation proposée au chapitre 5, nous n'avons pas affecté la ressource *Biblio_ 1* à un groupe de ressources virtuel.

La Figure 6.12 montre les modèles finaux correspondant à la modélisation par raffinement du sous-mécanisme de la Figure 6.8, et de la règle R4 : *Tout membre d'un groupe a accès à toutes les archives de recherche du laboratoire.*

De façon générale, nous remarquons qu'a partir de la décomposition, nous rendons la modélisation plus simple et plus explicite pour toutes les personnes concernées. Une fois la modélisation CatBAC de chaque sous-mécanisme terminée, l'étape suivante est de convertir les sous-modèles obtenus vers B-événementiel. Pour éviter de surcharger ce chapitre nous ne traitons que la traduction du raffinement conduisant au sous-modèle de la Figure 6.9. Ces traductions peuvent être obtenues automatiquement grâce à l'outil que nous avons dévellopé dans le cadre de notre mémoire. Cet outil sera présenté au Chapitre 7.

6.3 Les modèles B-événementiel

En nous référant aux étapes (2) et (3) de la Figure 6.3, qui décrit notre procéure de modélisation globale, tous les sous-modèles CatBAC doivent être traduits en B-événementiel pour être vérifiés et corrigés si nécéssaire. Dans cette section, nous montrons comment élaborer cette traduction.

Dans le chapitre 4, nous avons présenté la méthode B-événementiel. Nous avons mentionné qu'un modèle B-événementiel est constitué de deux composants principaux : un contexte pour les spécifications des éléments statiques des systèmes et une machine pour les spécifications des éléments dynamiques.

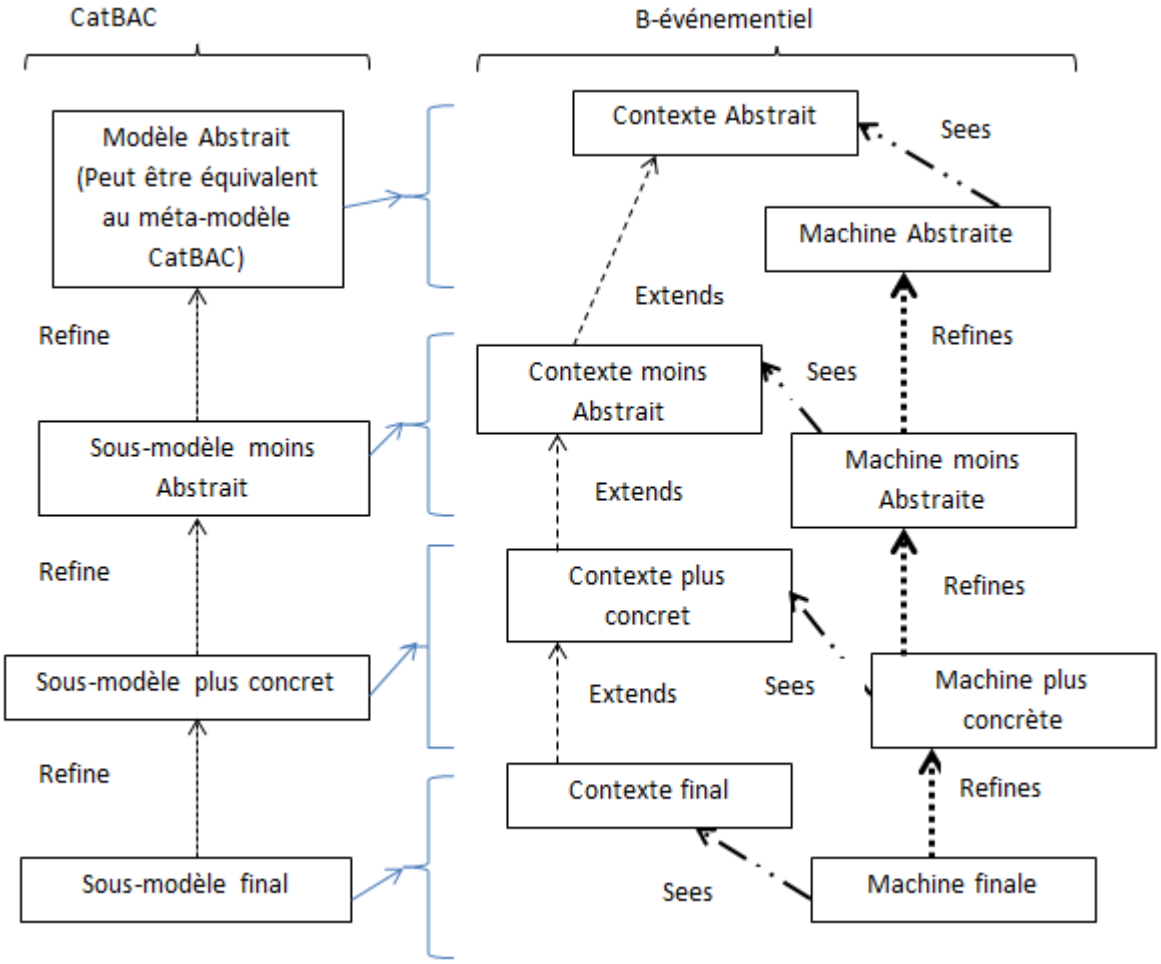

FIGURE 6.13 – Traduction du raffinement d'un sous-modèle CatBAC

Nous allons donc traduire chaque modèle CatBAC vers des composants machines et contextes. Pour ce faire, nous nous servirons des expressions formelles du raffinement que nous avons établi au Chapitre 5. La Figure 6.13 montre la démarche que nous suivons, pour traduire un raffinement de modèles CatBAC. Dans la partie gauche de cette figure, nous avons une succession de raffinements de modèles CatBAC. Dans la partie droite, on distingue une succession de raffinements de machines et de contextes B-

événementiel correspondant à chaque niveau de raffinement dans CatBAC. Pour chaque raffinement, une machine *voit* un contexte. A chaque passage vers un niveau de raffinement plus concret, le nouveau contexte *étend* le contexte du niveau abstrait et la nouvelle machine raffine la machine du niveau abstrait. Pour un rappel sur les définitions des relations *Sees, Extends, Refines* et sur le sens des notations B-événementiel, voir le Chapitre 4.

Les Figures 6.14, 6.15 et 6.16 correspondent à la succession du raffinement du sous-mécanisme de la Figure 6.5. Nous nous sommes arrêtés au second niveau de raffinement. Pour la traduction en B-événementiel des modèles de ces figures, nous noterons par convention toutes les propriétés à inclure dans les contextes en lettres majuscules. De cette façon, il sera facile de faire la différence entre des propriétés provenant des contextes et des propriétés provenant des machines.

6.3.1 Modèle abstrait (Figure 6.14)

Le contexte correspondant au modèle de la Figure 6.14 est mentionné ci-dessous. Nous appelons ce contexte ABSTRACT_C.

CONTEXT ABSTRACT_C
SETS

$SCAT$:Ensemble correspondant à la catégorie sCat
$ACAT$:Ensemble correspondant à la catégorie aCat
$RCAT$:Ensemble correspondant à la catégorie rCat
$CCAT$:Ensemble correspondant à la catégorie cCat

CONSTANTS

$ACCESS$:Ensemble contenant les permissions de la forme (s, a, r, c)

AXIOMS

axm1 : $ACCESS \in SCAT \leftrightarrow (ACAT \times RCAT \times CCAT)$

END

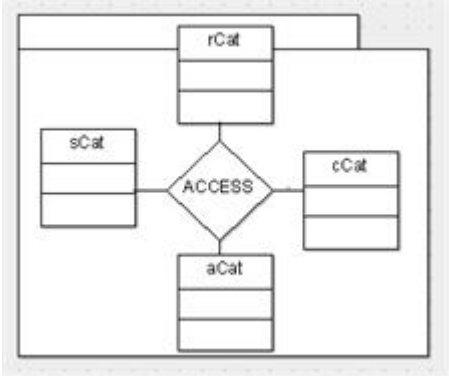

FIGURE 6.14 – Modèle abstrait

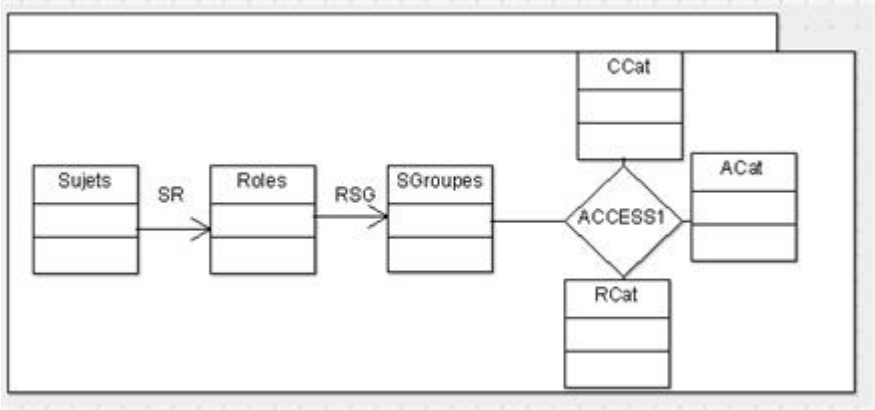

FIGURE 6.15 – Premier niveau de raffinement

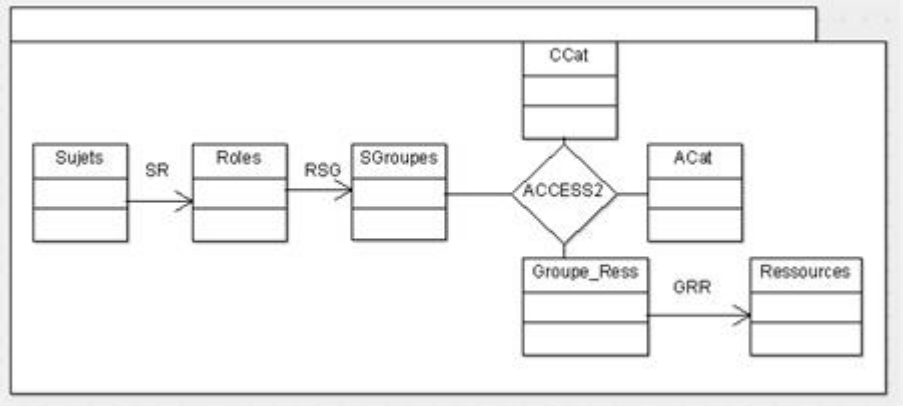

FIGURE 6.16 – Second niveau de raffinement

Dans ce contexte B-événementiel, nous avons défini les ensembles correspondant à chaque catégorie, sous la clause SETS. Conformément à notre convention, nous les avons noté en majuscules. Sous la clause CONSTANTS nous avons déclaré l'association ACCESS. Cette association est définie sous la clause AXIOMS, comme une relation entre l'ensemble des catégories des sujets et le produit cartésien des ensembles catégories d'actions, de ressources et de contextes.

La machine correspondant au modèle est décrite ci-dessous. Dans la traduction des machines, nous modélisons trois types d'événements (les opérations B-événementiel) :

- les événements qui permettent d'ajouter ou de supprimer du système des catégories
- les événements qui permettent d'ajouter ou de supprimer du système des éléments d'association
- les événements qui permettent d'attribuer ou de supprimer des permissions.

Nous ne modélisons pas les événements qui vérifient qu'un sujet a le droit ou non d'accéder à une ressource s'il en fait la demande. Ce choix est dû au fait que la structure de ce types d'événements diffère d'un modèle de contrôle d'accès à l'autre. Il revient donc à l'utilisateur de modéliser l'événement correspondant à ses besoins dans les sous-modèles finaux.

MACHINE ABSTRACT_M
SEES ABSTRACT_C
VARIABLES

$sCat$	: variable d'état correspondant à la catégorie des sujets
$sCat$	: variable d'état correspondant à la catégorie des actions
$rCat$	: variable d'état correspondant à la catégorie des ressources
$cCat$	: variable d'état correspondant à la catégorie des contextes
$access$	: variable d'état correspondant à l'association ACCESS

INVARIANTS

inv0 : $sCat \subseteq SCAT$: définition des variables d'états

inv2 : $aCat \subseteq ACAT$

inv4 : $rCat \subseteq RCAT$

inv6 : $cCat \subseteq CCAT$

inv8 : $access \subseteq ACCESS$

EVENTS

Initialisation

begin

act0 : $sCat :\in \mathbb{P}(SCAT)$: chaque variable est initialisée avec une partie de l'ensemble qui lui correspond dans le contexte associé à la machine

act2 : $aCat :\in \mathbb{P}(ACAT)$

act4 : $rCat :\in \mathbb{P}(RCAT)$

act6 : $cCat :\in \mathbb{P}(CCAT)$

act10 : $access :\in \mathbb{P}(ACCESS)$

end

END

Le nom de cette machine est ABSTRACT_M. Cette machine voit le contexte ABSTRACT_C que nous avons décrit précédement. Dans cette machine, nous définissons les variables d'états (Chapitre 4) qui correspondent à tous les éléments qui ont été introduits dans le modèle. Il s'agit de toutes les catégories d'éléments et de l'association ACCESS.

Nous définissons ensuite pour chaque variable d'état, son type dans la clause INVARIANTS, puis nous l'initialisons dans l'événement INITIALIZATION de la clause EVENTS. Chaque variable est initialisée avec une partie quelconque de l'ensemble des catégories qui lui correspond. Ces ensembles ont été définis dans le contexte ABSTRACT_C. Pour cette machine, nous présentons uniquement l'événement *access_evt* correspondant à la création

des permissions avec l'association n-aire. Les spécifications complètes des machines sont fournies dans les annexes. *access_evt* est comme suit :

Event *access_evt* $\widehat{=}$: Evénement qui simule la création d'une permission. Par abus nous utiliserons le terme permission pour désigner dépendamment des situations, les tuples (s, a, r, c) ou (a, r, c)

any

$scat\_p$: paramètre simulant un sujet

$acat\_p$: paramètre simulant une action

$rcat\_p$: paramètre simulant une ressource

$ccat\_p$: paramètre simulant une variable de contexte

where

grd1 : $scat\_p \in sCat$: chaque sujet doit être enregistré dans le système avant qu'on lui donne une permission (a, r, c)

grd2 : $acat\_p \in aCat$

grd3 : $rcat\_p \in rCat$

grd4 : $ccat\_p \in cCat$

grd5 : $scat\_p \mapsto (acat\_p \mapsto rcat\_p \mapsto ccat\_p) \in ACCESS \setminus access$:la permission (s, a, r, c) ne doit pas préalablement exister dans le système

then

act10 : $access := access \cup \{scat\_p \mapsto (acat\_p \mapsto rcat\_p \mapsto ccat\_p)\}$: on ajoute la nouvelle permission au système

END

L'événement *access_evt* prend quatre paramètres : un sujet, une action, une ressource et un contexte (Clause *any*). Sous la clause *where* il vérifie que chaque paramètre satisfait une condition particulière. D'après ces conditions, chaque sujet, action, ressource et contexte doit être préalablement enrégistré

dans le système avant le déclenchement de l'événement. Aussi, la permission à créer doit être une nouvelle permission. Le symbole ($\mapsto$) est utilisé pour former des tuples d'éléments (scat_p, (acat_p, rcat_p, ccat_p)). Sous la clause *then*, une nouvelle permission est ajoutée à l'ensemble *access* qui correspond à la variable dynamique qui stocke les permissions.

6.3.2 Premier niveau de raffinement (Figure 6.15)

Ce niveau de raffinement correspond au modèle de la Figure 6.15. le contexte de ce modèle est comme suit :

CONTEXT Less_ABSTRACT_C
EXTENDS ABSTRACT_C
CONSTANTS

$SGROUPES$ ..... : ensemble des groupes des sujets
$ROLES$ ..... : ensemble des rôles
$SUJETS$ ..... : ensemble des sujets
SR ..... : association entre sujets et rôles
RGR ..... : association entre rôles et groupes de sujets
$ACCESS1$ ..... : nouvelle association n-aire

AXIOMS

axm1 : $SGROUPES \subseteq SCAT$

axm2 : $ROLES \subseteq SCAT$

axm3 : $SUJETS \subseteq SCAT$

axm4 : $SGROUPES \neq ROLES$

axm5 : $SGROUPES \neq SUJETS$

axm6 : $ROLES \neq SUJETS$

axm7 : $SR \in SUJETS \leftrightarrow ROLES$

axm8 : $RSG \in ROLES \leftrightarrow SGROUPES$

axm10 : $ACCESS1 \subseteq ACCESS$

END

Le nom de ce contexte est Less_ABSTRACT_C. Ce contexte est une extension du contexte ABSTRACT_C. Chaque nouvel ensemble est ajouté sous la clause CONSTANTS. Les définitions de ces ensembles sont sous la clause AXIOMS. Nous pouvons remarquer dans les axiomes (*axm1, axm2* et *axm3*), la définition du raffinement de la catégorie des sujets sCat. Les associations SR et RSG sont définies dans les axiomes *axm7 et axm8*. Dans l'axiome *axm10* nous indiquons que l'association ACCESS1 raffine l'association abstraite ACCESS.

La machine correspondant au modèle est définie ci-après.

MACHINE Less_ABSTRACT_M
REFINES ABSTRACT_M
SEES Less_ABSTRACT_C
VARIABLES

$SGroupes$: variable d'état correspondant aux groupes des sujets
$aCat$ ..... : variable d'état correspondant aux actions
$rCat$ ..... : variable d'état correspondant aux ressources
$cCat$ ..... : variable d'état correspondant aux contextes
$Roles$ ..... : variable d'état correspondant aux rôles
$Sujets$ ..... : variable d'état correspondant aux sujets
$access1$

sr .. : variable d'état correspondant à l'association entre sujets et rôles

rsg .. : variable d'état correspondant à l'association entre rôles et groupes de sujets

INVARIANTS

inv0 : $SGroupes \subseteq SGROUPES$

inv1 : $SGroupes = sCat$: invariant de collage

inv2 : $aCat \subseteq ACAT$

inv4 : $rCat \subseteq RCAT$

inv6 : $cCat \subseteq CCAT$

inv8 : $Roles \subseteq ROLES$

inv10 : $Sujets \subseteq SUJETS$

inv12 : $access1 \subseteq ACCESS1$

inv13 : $access1 = access$: invariant de collage

inv15 : $sr \subseteq SR$

inv17 : $rsg \subseteq RSG$

Le nom de la machine précédente est Less_ABSTRACT_M. Elle raffine la machine abstraite ABSTRACT_M et voit le contexte Less_ABSTRACT_C. Comme pour le premier raffinement, la variable correspondant à chaque catégorie du modèle est définie sous la clause VARIABLES. Les expressions de typages des variables sont mentionées sous la clause INVARIANTS. Deux invariants de collage (Chapitre 4) sont ajoutés (*inv1 et inv13*).

L'invariant *inv1* indique que la variable *SGroupes* qui correspond aux groupes des sujets doit se comporter comme la variable abstraite $sCat$, car cette dernière n'existe plus dans le modèle concret. Nous pouvons remarquer son absence dans la clause VARIABLES. Cet invariant de collage fait référence à l'interpretation formelle $var(SGroupes) = var(sCat)$ que nous avons introduit dans le chapitre 5. L'invariant de collage *inv13* joue la même fonction pour la variable *access* qui disparait pour laisser place à la variable $access1$.

Pour la même machine, puisque les variables $sCat$ et $access$ ont disparu, nous devons indiquer une clause WITH pour modéliser le comportement du prédicat-avant-après (BAP) correspondant aux invariants *inv1 et inv13*.

begin

with

`sCat'` : $sCat' = SGroupes'$

`access'` : $access' = access1'$

Par exemple, la première expression de cette clause indique que la nouvelle valeur de la variable $sCat$ est $SGroupes$.

L'événement correspondant à la création des permissions avec l'association $ACCESS1$ est ci-après :

Event $access1\_evt \mathrel{\widehat{=}}$
refines $access\_evt$

any

$sgroupes\_p$ .. : paramètre de groupe de sujets

$acat\_p$ .... : paramètre action

$rcat\_p$ ... . : paramètre ressource

$ccat\_p$ .... : paramètre contexte

where

`grd1` : $sgroupes\_p \in SGroupes$

`grd2` : $acat\_p \in aCat$

`grd3` : $rcat\_p \in rCat$

`grd4` : $ccat\_p \in cCat$

`grd5` : $sgroupes\_p \mapsto (acat\_p \mapsto rcat\_p \mapsto ccat\_p) \in ACCESS1 \setminus access1$

with

`scat_p` : $sgroupes\_p = scat\_p$: le paramètre abstrait $scat\_p$ a disparu

then

act14 : $access1 := access1 \cup \{sgroupes\_p \mapsto (acat\_p \mapsto rcat\_p \mapsto ccat\_p)\}$

end

Cet événement est appelé $access1\_evt$. Il raffine l'événement abstrait $access\_evt$. L'événement $access1\_evt$ prend en paramètres les mêmes éléments que $access\_evt$ sauf qu'à la place du paramètre $scat\_p$ qui représentait un sujet, $access1\_evt$ prend un paramètre $sgroupes\_p$ qui représente un groupe de sujets. Dans cet événement, il s'agit d'affecter une permission à un groupe de sujets. Ainsi lorsqu'un groupe de sujets sera à son tour affecté à un rôle, ce rôle aura toutes les permissions du groupe de sujets affecté.

Pour ce modèle, nous présentons également les événements qui permettent d'affecter un rôle à un groupe de sujets et un sujet à un rôle.

Pour affecter un rôle à un groupe de sujets, l'événement $rsgcr\_evt$ ci-dessous doit être déclenché.

Event $rsgcr\_evt \mathrel{\widehat{=}}$ (Roles_SGroupe_creation_evt)

any

$roles\_p$ ..... : un rôle

$sgroupes\_p$: un groupe de sujets

where

grd4 : $roles\_p \in Roles$

grd5 : $sgroupes\_p \in SGroupes$

grd6 : $roles\_p \mapsto sgroupes\_p \in RSG \setminus rgr$

then

act1 : $rsg := rsg \cup \{roles\_p \mapsto (sgroupes\_p)\}$

end

A l'image des autres événements présentés dans le modèles, $rsgcr\_evt$ prend en paramètres un rôle et un groupe de sujets. Il vérifie ensuite dans la clause *where* ($grd6$), que l'affectation des deux paramètres n'est pas encore faite, puis il la réalise dans la clause *then*.

De la même façon, l'événement $srcr\_evt$ (Sujets_Roles_creation_evt) prend en paramètre un sujet et un rôle, vérifie que l'affectation de ce sujet à ce rôle n'a pas encore été faite dans le système puis la réalise.

Event $srcr\_evt \mathrel{\widehat{=}}$

any

$sujets\_p$

$roles\_p$

where

grd1 : $sujets\_p \in Sujets$

grd2 : $roles\_p \in Roles$

grd3 : $sujets\_p \mapsto roles\_p \in SR \setminus sr$

then

act1 : $sr := sr \cup \{sujets\_p \mapsto (roles\_p)\}$

end

6.3.3 Second niveau de raffinement (Figure 6.16)

Ce niveau correspond à la figure 6.16. Il s'agit de raffiner le modèle précédent pour y ajouter la catégorie des groupes de sujets et la catégorie des ressources. Ces nouvelles catégories raffinent la catégorie abstraite $rCat$ du modèle présenté au premier raffinement. Le contexte de ce modèle est ci-après :

CONTEXT CONCRETE_C
EXTENDS Less_ABSTRACT_C

CONSTANTS

$GROUPE\_RESS$

$RESSOURCES$

GRR

$ACCESS2$

AXIOMS

axm1 : $GROUPE\_RESS \subseteq RCAT$

axm2 : $RESSOURCES \subseteq RCAT$

axm3 : $GROUPE\_RESS \neq RESSOURCES$

axm4 : $GRR \in GROUPE\_RESS \leftrightarrow RESSOURCES$

axm6 : $ACCESS2 \subseteq ACCESS1$

END

Ce contexte est à l'image des contextes précédents. Nous y avons ajouté les ensembles $GROUPE\_RESS$, $RESSOURCES$, GRR, $ACCESS2$. Ces ensembles ont déja été présentés. Ils correspondent respectivement aux groupes de ressources, aux ressources, aux éléments de l'association GRR (qui relie une ressources à un groupe de ressources), et à l'association $ACCESS2$. Les descriptions du raffinements sont présentées sous la clause $AXIOMS$. Elles sont conformes aux interprétations faites dans le Chapitre 5. La machine correspondant à ce contexte est ci-dessous.

MACHINE CONCRETE_M
REFINES Less_ABSTRACT_M
SEES CONCRETE_C

VARIABLES

$SGroupes$ ..... : groupes de sujets

$aCat$ ..... : actions

$Groupe\_Ress$: groupes de ressources

$cCat$ ..... : contextes

$Roles$ ..... : rôles

$Sujets$ ..... : sujets

$Ressources$ ... : ressources

$access2$

sr .. : relation entre sujets et rôles

rsg : relation entre rôles et groupes de sujets

grr : relation entre groupes de ressources et ressources

INVARIANTS

inv0 : $SGroupes \subseteq SGROUPES$

inv2 : $aCat \subseteq ACAT$

inv4 : $Groupe\_Ress \subseteq GROUPE\_RESS$

inv5 : $Groupe\_Ress = rCat$: invariant de collage

inv6 : $cCat \subseteq CCAT$

inv8 : $Roles \subseteq ROLES$

inv10 : $Sujets \subseteq SUJETS$

inv12 : $Ressources \subseteq RESSOURCES$

inv14 : $access2 \subseteq ACCESS2$

inv15 : $access2 = access1$ ... : invariant de collage

inv17 : $sr \subseteq SR$

inv19 : $rsg \subseteq RGR$

inv21 : $grr \subseteq GRR$

END

Dans cette machine, nous avons ajouté en plus des variables d'états déjà déclarées dans les machines Less_ABSTRACT_M et ABSTRACT_M, les

variables $Groupe\_Ress$, $Ressources$, $access2$, grr. Les types de ces variables d'états sont définis dans la clause INVARIANTS. Dans cette clause nous avons également ajouté l'expression $inv5 : Groupe\_Ress = rCat$ correspondant à l'invariant de collage qui justifie la disparition de la variable $rCat$.

L'initialisation de cette machine est ci-après.

Initialisation

begin

with

`rCat'` : $rCat' = Groupe\_Ress'$

`access1'` : $access1' = access2'$

`act0` : $SGroupes :\in \mathbb{P}(SGROUPES)$
`act2` : $aCat :\in \mathbb{P}(ACAT)$
`act4` : $Groupe\_Ress :\in \mathbb{P}(GROUPE\_RESS)$
`act6` : $cCat :\in \mathbb{P}(CCAT)$
`act8` : $Roles :\in \mathbb{P}(ROLES)$
`act10` : $Sujets :\in \mathbb{P}(SUJETS)$
`act12` : $Ressources :\in \mathbb{P}(RESSOURCES)$
`act16` : $access2 :\in \mathbb{P}(ACCESS2)$
`act18` : $sr :\in \mathbb{P}(SR)$
`act20` : $rgr :\in \mathbb{P}(RGR)$
`act22` : $grr :\in \mathbb{P}(GRR)$

end

Chaque variable est initialisée avec une partie quelconque de l'ensemble qui lui correspond. Sous la clause *with* nous avons spécifié deux expressions. La première indique que dans la machine actuelle, toutes les valeures de la variable abstraite $rCat$ correspondent à la variable $Groupe\_Ress$. La seconde expression indique la même chose pour les variables $access2$ du modèle concret et $access1$ du modèle abstrait.

L'événement d'attribution des permissions est ci-après :

Event $access2\_evt \widehat{=}$
refines $access1\_evt$

any

$sgroupes\_p$ ..... : un groupe de sujets

$acat\_p$ ..... : une action

$groupe\_ress\_p$: un groupe de ressources

$ccat\_p$ ..... : un contexte

where

grd1 : $sgroupes\_p \in SGroupes$

grd2 : $acat\_p \in aCat$

grd3 : $groupe\_ress\_p \in Groupe\_Ress$

grd4 : $ccat\_p \in cCat$

grd5 : $sgroupes\_p \mapsto (acat\_p \mapsto groupe\_ress\_p \mapsto ccat\_p) \in ACCESS2 \setminus access2$: vérifie que l'action en cours ne l'a pas encore été

with

rcat_p : $groupe\_ress\_p = rcat\_p$: le paramètre $rcat\_p$ a disparu

then

act16 : $access2 := access2 \cup \{sgroupes\_p \mapsto (acat\_p \mapsto groupe\_ress\_p \mapsto ccat\_p)\}$

end

La logique de cet événement est la même que les précédentes. Cet événement raffine l'événement abstrait $access1\_evt$. Il prend en paramètres un groupe de sujets, une action, un groupe de ressources, un contexte, vérifie ensuite que chaque élément existe dans le système et que la permission en cours de création n'a pas encore été créée. Une fois les vérifications faites, il ajoute la nouvelle permission à l'ensemble des permissions contenus dans $access2$ (clause *then*). L'expression sous la clause *with* fait référence à la disparition du paramètre $rcat\_p$ qui existait dans l'événement abstrait $access1\_evt$.

Les spécifications complètes des machines sont fournies en annexes. Ici nous avons présenté uniquement la traduction de la modélisation du sous-mécanisme de la Figure 6.4. Chaque sous-mécanisme devra être traduit de la même façon. Dans la traduction des modèles, nous avons spécifié essentiellement les événements qui sont sous forme de constructeurs. D'autres événements comme les événements d'historisations des accès peuvent être ajoutés si les utilisateurs le jugent nécéssaire. Le rôle de ces événements est d'archiver les accès de chaque sujet. Dans la version actuelle de notre outil, ce type d'événement n'a pas été ajouté.

Si besoin est, tous les sous-modèles B-événementiel correspondant aux sous-mécanismes d'une politique peuvent être recomposés. Nous donnons dans la section suivante une idée sur cette recomposition. Un objectif possible de la recomposition des sous-modèles serait de pouvoir exprimer des propriétés de sécurité qui font référence à tous les sous-mécanismes d'accès. Un autre objectif pourrait être de vérifier que les sous-modèles ne se contredisent pas.

6.4 Composition des modèles B-événemnetiel

Dans notre mémoire cette étape correspond à l'accolade (4) de la Figure 6.3. Elle est facultative. Ici nous donnons une idée sur la façon dont la recomposition des modèles B-événementiel pourrait se faire. Notons que la composition des modèles B-événementiel a largement été discutée dans [14,15]. Pour composer des modèles B-événementiel, il faut prendre en compte plusieurs éléments, parmi lesquels, les variables d'états, les propriétés et les événements de chaque sous-modèle. La Figure 6.17 présente un exemple de composition de sous-modèle en considérant uniquement les événements de chaque sous-modèle.

Dans cette figure, nous avons deux sous-modèles B-événementiel à composer. Il s'agit de deux sous-machines (S-M B-événementiel (1)) et (S-M B-événementiel (2)). Dans la sous-machine S-M B-événementiel (1), nous avons trois événements comme suit :

– Evt1(S, R) est un événement qui prend en paramètre un sujet et rôle puis en fait l'association dans une variable d'état SR.
– Evt2(R, G) est un événement qui prend en paramètre un rôle et groupe puis en fait l'association dans une variable d'état RG.
– Evt3(G, P) prend en paramètre un groupe et une permission puis en fait l'association dans une variable d'état GP.

Dans la sous-machine S-M B-événementiel (2) on distingue les deux événements ci-après :

– Evt1(S, R) prend en paramètre un sujet et un rôle puis en fait l'association dans une variable d'état SR.
– Evt3(R, P) prend en paramètre un rôle et une permission puis en fait l'association dans une variable d'état RP.

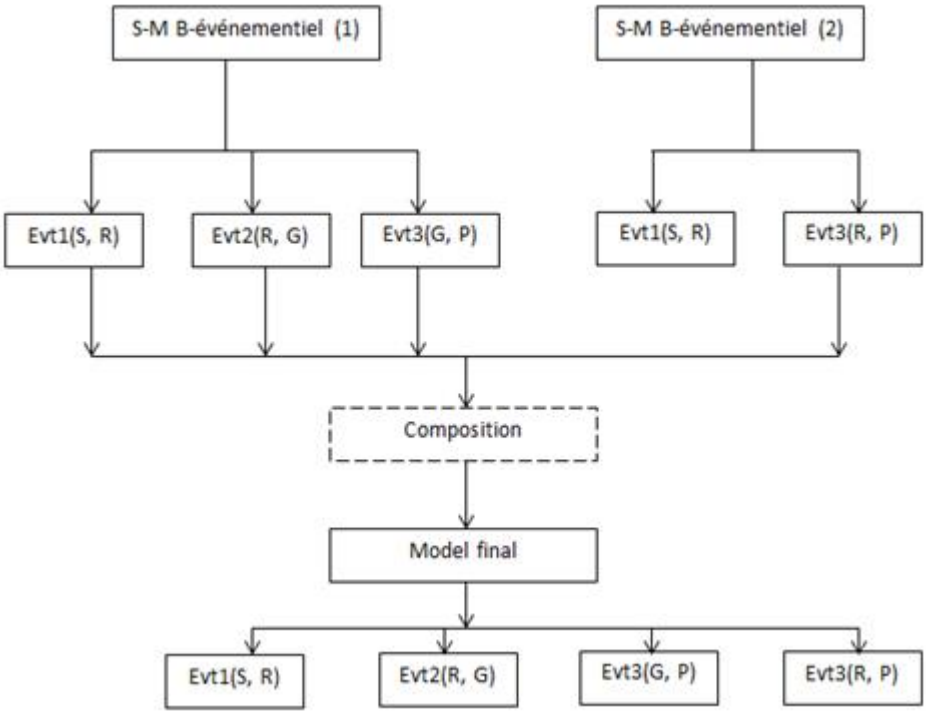

FIGURE 6.17 – Composition de machines en considérant les événements

La composition de ces deux machines produit un modèle final (ici une machine B-événementiel) qui contient les événements mentionnés dans la Figure 6.17. Nous remarquons que tous les événements qui n'exisitent pas encore dans d'autres sous-machines ont tous été pris en considération. C'est le cas des événements Evt2(R, G), Evt3(G, P) et Evt3(R, P). Pour les événements communs aux sous-machines, il faut les modéliser une seule fois dans la machine finale. C'est le cas de l'événement Evt1(S, R) (nous considérons que cet événement a les mêmes gardes et fait les mêmes actions dans les deux sous-modèles).

6.4.1 Cas des modèles correspondants à des règles de contrôle d'accès

Pour les modèles correspondant à des règles de contrôle d'accès (exemple second modèle de la Figure 6.9), certains événements communs à plusieurs sous-machines doivent être traités différemment.

Supposons un événement E1(S, R) dans une sous-machine A et un événement E1(S, R) dans une sous-machine B. L'objectif global de cet événement est d'affecter un sujet à un rôle. Dans la sous-machine A, E1 prend en paramètre un sujet et un rôle qui doit être *Etudiant*. Dans la sous-machine B, E1 prend en paramètre un sujet et un rôle qui cette fois-ci doit être $Resp\_lab$ (responsable de laboratoire).

Pour composer ce type d'événement, Il faut indiquer dans la machine finale que l'événement E1(S, R) prend en paramètre un sujet, et un rôle qui peut être étudiant ou responsable de laboratoire. Une disjonction sera donc introduite pour modéliser la garde B-événementiel du paramètre rôle (R).

Ainsi lorsque nous modélisons des règles de contrôle d'accès, nous exprimons en réalité une partie ou la totalité d'une même garde pour un paramètre. Dans le cas où les événements contiennent des gardes partielles, nous devons composer ces gardes en appliquant des disjonctions pour obtenir la garde totale. Un exemple est fourni ci-dessous. Dans ces événements la variable sr et l'ensemble SR sont les mêmes que ceux définis à la section précédente.

Machine A

Event $srcr\_evt \mathrel{\widehat{=}}$

Evénement qui définit un sujet comme étudiant

any

$sujets\_p$:paramètre sujet

$roles\_p$: paramètre rôle

where

grd1 : $sujets\_p \in Sujets$: garde de sujet

grd2 : $roles\_p \in Roles$ ... : garde de rôle

grd3 : $roles\_p = Etudiant$: rôle comme Etudiant

grd4 : $sujets\_p \mapsto roles\_p \in SR \setminus sr$

then

act1 : $sr := sr \cup \{sujets\_p \mapsto (roles\_p)\}$

end

Machine B

Event $srcr\_evt \mathrel{\widehat{=}}$

Evénement qui définit un sujet comme responsable de lab

any

$sujets\_p$:paramètre sujet

$roles\_p$: paramètre rôle

where

grd1 : $sujets\_p \in Sujets$: garde de sujet

grd2 : $roles\_p \in Roles$ ... : garde de rôle

grd3 : $roles\_p = Resp\_lab$: rôle comme responsable de laboratoire

grd4 : $sujets\_p \mapsto roles\_p \in SR \setminus sr$

then

act1 : $sr := sr \cup \{sujets\_p \mapsto (roles\_p)\}$

end

Evénement Composé :

Event $srcr\_evt \mathrel{\widehat{=}}$

Composition des événements précédents

any

$sujets\_p$: paramètre sujet

$roles\_p$: paramètre rôle

where

grd1 : $sujets\_p \in Sujets$: garde de sujet

grd2 : $roles\_p \in Roles$: garde de rôle

grd3 : $roles\_p = Etudiant \vee roles\_p = Resp\_lab$ **: rôle comme étudiant ou responsable de laboratoire**

grd4 : $sujets\_p \mapsto roles\_p \in SR \setminus sr$

then

act1 : $sr := sr \cup \{sujets\_p \mapsto (roles\_p)\}$

end

Nous pouvons remarquer la disjonction utilisée dans la garde $grd3$ de l'événement composé.

6.5 Conclusion

L'objectif de ce mémoire est de construire un cadre de spécifications et de vérifications des modèles de contrôle d'accès hybrides. Pour y arriver nous avons combiné la méthode de spécification semi-formelle CatBAC et la méthode de spécification formelle B-événementiel afin de produire des modèles de contrôles d'accès hybrides qui sont à la fois simples et corrects par

construction. Dans le Chapitre 5, nous avons montré comment interpréter formellement le raffinement des modèles CatBAC. Dans le présent chapitre, nous avons présenté une approche de modélisation globale visant à alléger la compréhension des modèles et leurs implémentations. Nous allons maintenant présenter au chapitre suivant un outil qui permet de créer des modèles de contrôle d'accès hybrides en appliquant les méthodes discutées dans ce mémoire.

Chapitre 7

Un outil pour la modélisation des systèmes de contrôle d'accès avec CatBAC et B-événementiel (contributions)

Dans ce chapitre, nous présentons l'outil CatBAC_tl que nous avons dévellopé pour modéliser des systèmes de contrôle d'accès hybrides. Les diagrammes que nous avons réalisé dans ce chapitre sont ceux correspondant à la politique Pol_1. Ces diagrammes ont déja été discutés dans les chapitres 5 et 6. Il en est de même pour les spécifications B-événemmentiel.

7.1 Introduction

Pour répondre au dernier objectif de notre mémoire, nous avons développé un outil pour modéliser des systèmes de contrôle d'accès hybrides. Cet outil implémente l'approche de modélisation par décomposition présentée au Chapitre 6. Dans sa version actuelle, notre outil de modélisation est expérimental. Il s'agit d'un plugin pour l'application Rodin. Rodin [26] est une plateforme qui permet de construire des modèles avec B-événementiel.

Cette plateforme est conçue par extension de l'environement de dévellopement Eclipse [27]. Cela lui permet d'avoir la plupart des fonctionnalités de Eclipse, dont celui de l'ajout de plugins. Eclipse est l'une des applications plus utilisées en développement informatique. Grâce à son mode de fonctionnement, des modules d'extensions (plugins) destinés à lui apporter de nouvelles fonctionnalités peuvent être ajoutés. C'est ce que nous avons fait pour la plateforme Rodin.

Dans ce chapitre, nous présentons les fonctionnalités que nous avons dévellopé pour notre plugin. Nous montrons éssentiellement comment ce plugin que nous avons appelé CatBAC_tl (CatBAC tool) nous permet de modéliser des systèmes de contrôles d'accès hybrides.

7.2 Présentation de CatBAC_tl

Pour concevoir notre outil, nous avons choisi de l'implémenter sous forme de plugin pour la platforme Rodin. Ainsi, nous pouvons bénéficier des nombreux outils de traitements de modèles B-événementiel existants pour cette platforme.

Notre plugin est un éditeur graphique qui permet de modéliser des sytèmes de contrôle d'accès avec CatBAC et de les traduire ensuite vers B-événementiel. Pour concevoir ce plugin, nous avons utilisé un autre plugin de Eclipse appelé GMF (Graphical Modeling Framework) [28]. Ce plugin facillite la création des éditeurs graphiques pour les applications qui sont basées sur Eclipse (dans notre cas il s'agit de Rodin).

Dans ce chapitre, les figures que nous présentons sont issues des tests réalisés avec Eclipse (version Indigo Service Release 2). Notre plugin aura le même comportement dans Rodin.

7.2.1 Creation d'un nouveau projet pour la politique Pol_1

Lorsque notre plugin CatBAC_tl est intégré à Eclipse, l'environnement partiel de tavail obtenu est présenté à la Figure 7.1. Nous pouvons y remar-

quer le menu CatBAC à partir duquel nous ferons la plupart des manipulations.

Pour commencer une nouvelle modélisation, nous devons créer un projet de type *General*, comme présenté dans la Figure 7.2. Dans ce projet, nous créons ensuite un nouveau diagramme CatBAC (Figure 7.3). Le projet créé pour notre démonstration est Pol_1. Nous allons y modéliser les exigences de la politique Pol_1 présentée dans les chapitres précédents.

Dans le projet Pol_1 que nous venons de créer, on distingue deux types de fichiers : un premier avec l'extension *.catbac* et un second avec l'extension *.catbac_diagram*. Le fichier avec l'extension *.catbac* (Figure 7.4) fournit un premier éditeur de modèle qui nous permet surtout de visualiser un modèle en déroulant chaque noeud (nous y reviendrons). Le second fichier avec l'extension *.catbac_diagram* fournit l'éditeur graphique (Figure 7.5) dans lequel nous allons construire des modèles CatBAC. Dans cet éditeur nous remarquons un noeud appelé $CatBAC$. Ce noeud est créé par défaut à chaque fois qu'un nouveau fichier de type *.catbac_diagram* est créé. Ce noeud correspond au méta-modèle CatBAC.

7.2.2 Création d'un nouveau modèle

Un double-clique avec la souris sur le noeud CatBAC (Figure 7.6), permet d'initialiser un diagramme correspondant au modèle le plus abstrait dans CatBAC. La Figure 7.7 présente ce diagramme. Il s'agit d'un modèle dont chaque noeud représente l'élément du même nom dans le méta-modèle CatBAC. On distingue donc la catégorie des sujets $sCat$, la catégorie des ressources $rCat$, la catégorie des actions $aCat$, et la catégorie des contextes $cCat$. L'association $ACCESS$ est notée en minuscules.

Chaque noeud d'un diagramme est composé de quatre compartiments. Sur le noeud $sCat$ (Figure 7.8), on distingue par exemple un compartiment pour exprimer le nom du noeud (ici le nom du neoud est $sCat$), un compartiment $Axioms$ pour exprimer des propriétés que nous considérons toujours vraies, un compartiment $Theorems$ pour les propriétés à démontrer et un

compartiment *SuperType* pour indiquer la classe abstraite dont la classe courante est le raffinement.

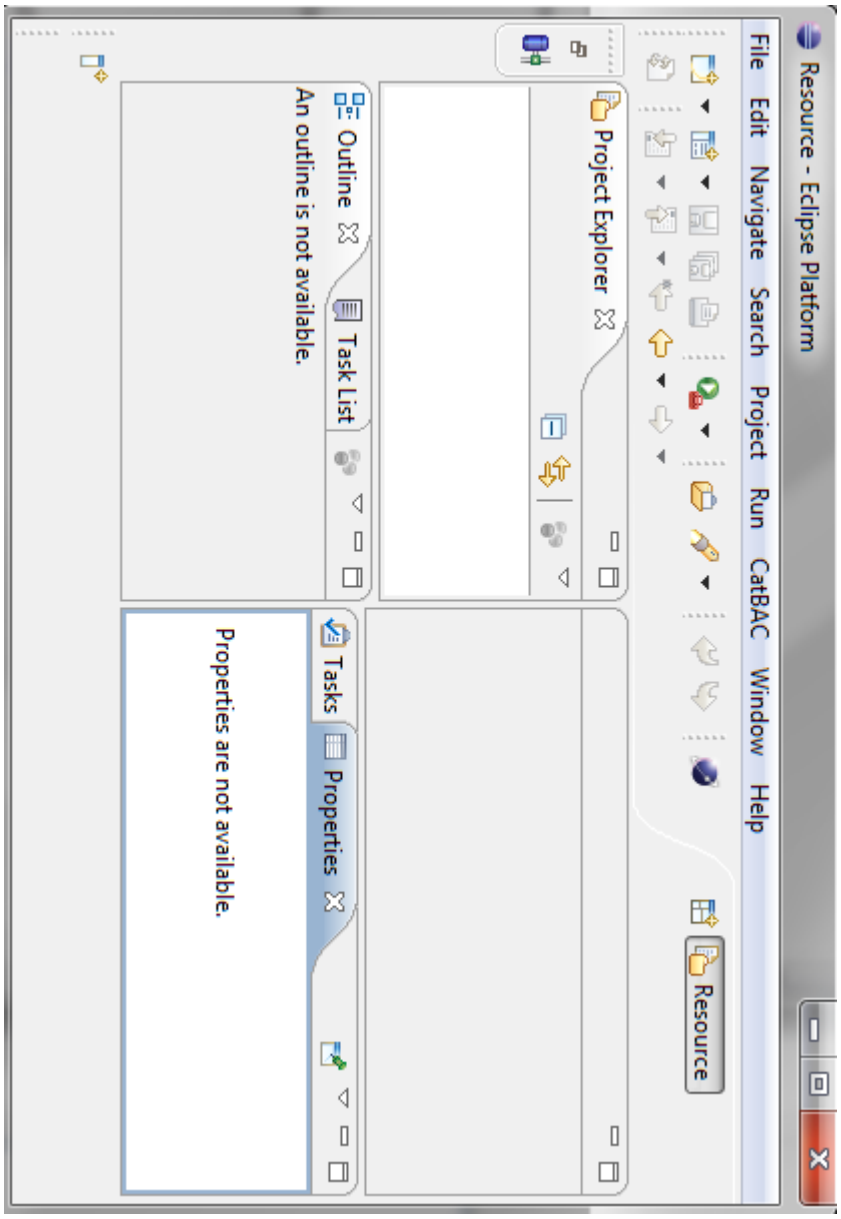

FIGURE 7.1 – Environnement de travail

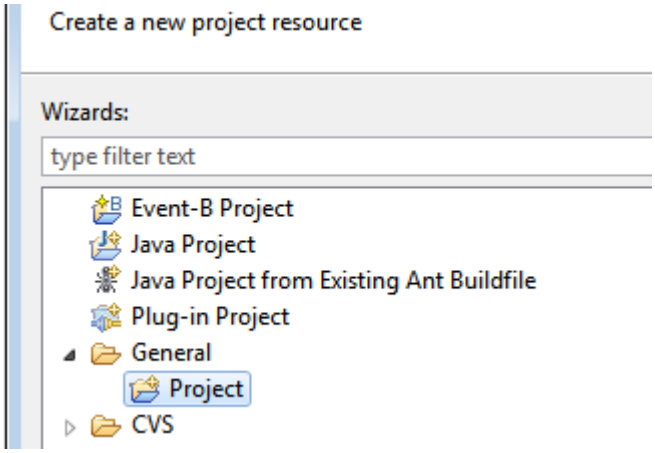

FIGURE 7.2 – Creation d'un nouveau projet

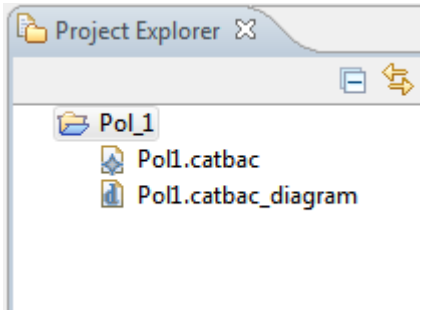

FIGURE 7.3 – Un nouveau projet

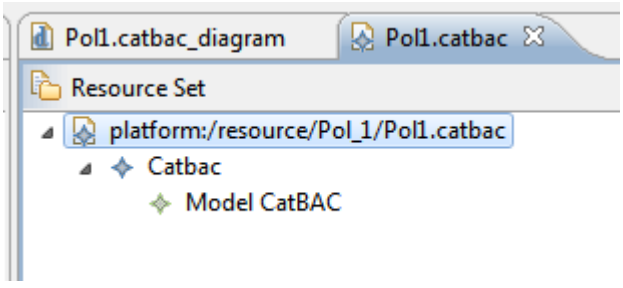

FIGURE 7.4 – Fichier avec etension .catbac

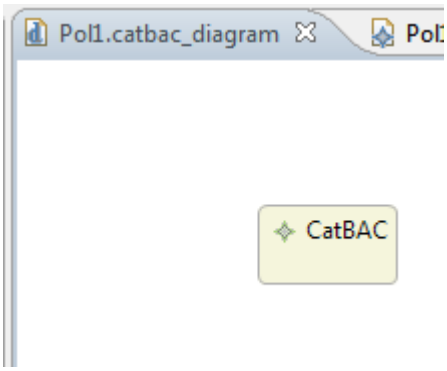

FIGURE 7.5 – Fichier avec etension .catbac_diagram

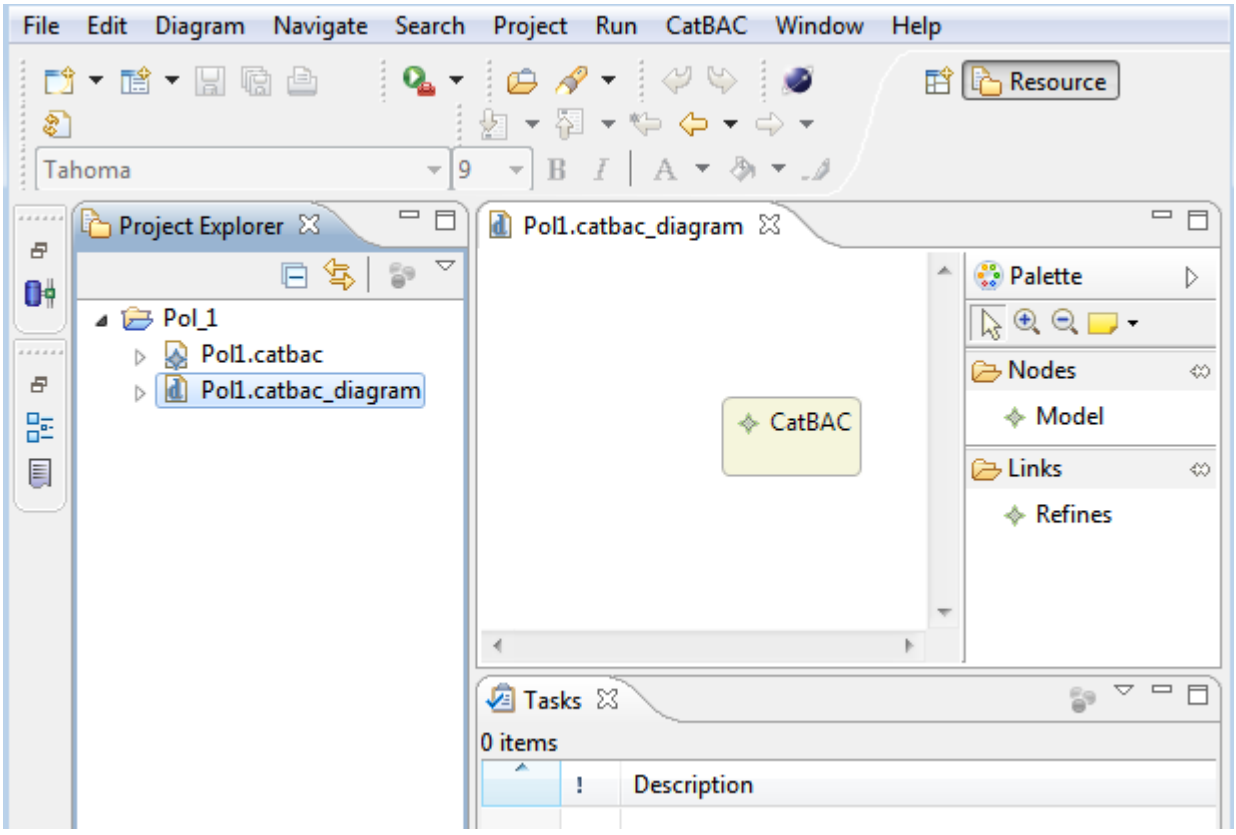

FIGURE 7.6 – Editeur de modèles

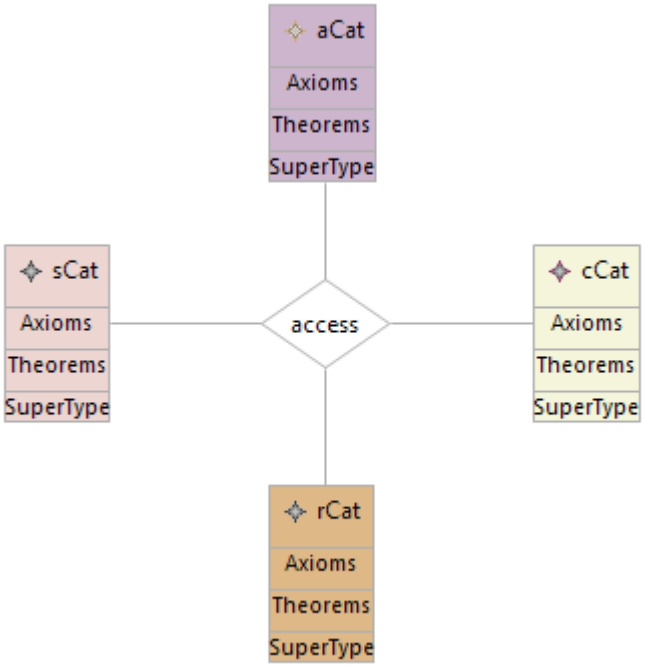

FIGURE 7.7 – Modèle abstrait

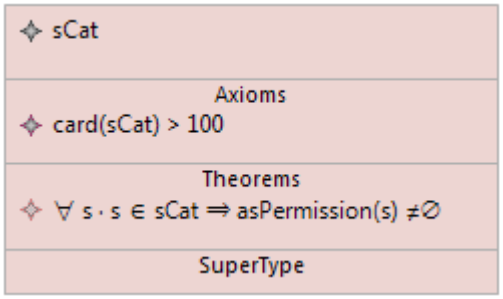

FIGURE 7.8 – Exemple de catégorie

La Figure 7.9 montre les composants qui nous permettent de construire des modèles. On y distingue des noeuds, des associations, et des composants pour définir des propriétés de modèles. Parmi ces composants, *Cl_ Axiom* et *Cl_ Theorem* permettent de définir des propriétés propres à des noeuds ; tandis que *Axiom* et *Theorem* permettent de définir des propriétés plus globales sur un modèle.

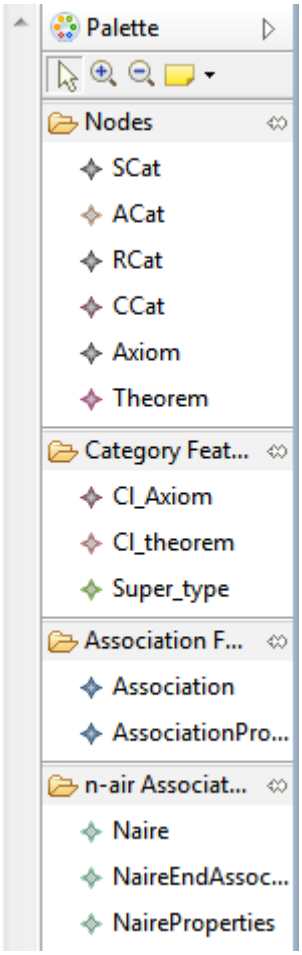

FIGURE 7.9 – Palette d'éléments pour construire les modèles

Premier raffinement de modèle

Ce raffinement correspond à la modélisation du sous-mécanisme de la Figure 6.5 du Chapitre 6. Dans ce sous-mécanisme, des sujets ($Sujet$s) sont affectés à des rôles ($Roles$). Les rôles sont ensuite affectés à des groupes de sujets ($SGroupes$). Les ressources ($Ressources$) sont affectées à des groupes de ressources ($Groupe\_Ress$) et les groupes de ressources sont affectés aux groupes de sujets ($SGroupes$).

Nous allons donc raffiner le modèle abstrait (Figure 7.7) pour construire ce mécanisme. Dans un premier temps, nous allons raffiner la catégorie des sujets $sCat$ pour obtenir les trois nouvelles catégories $Roles$, $SGroupes$, $Sujets$ (nous suivons les mêmes étapes qu'aux chapitres 5 et 6). Pour cela, dans le plugin CatBAC_tl, nous double-cliquons sur le noeud $sCat$ pour initialiser un éditeur dédié uniquement à la création des catégories de sujets. La Figure 7.10 présente le diagramme obtenu. Nous remarquons que toutes les nouvelles catégories ajoutées ont un supertype du nom de $sCat$. Ce supertype fait le lien entre la catégorie abstraite et les catégories raffinées. Le diagramme produit a déja été expliqué au chapitre 5 et 6. Dans le noeud $SGroupes$ du diagramme, l'expression modélisée dans le compartiment $Axioms$ est générée automatiquement. Cette expression est identique à celle discutée dans le chapitre 5. Elle indique que la catégorie $SGroupes$ joue le rôle de la catégorie abstraite $sCat$.

Toutes les catégories du diagramme de la Figure 7.10 ont été générées automatiquement à l'exception des nouvelles catégories $Roles$, $Sujets$ et des associations sr et rsg que l'utilisateur doit ajouter lui même. Il doit également indiquer le nom de la catégorie qui joue le rôle de la catégorie abstraite.

Second raffinement de modèle

Le second raffinement de notre sous-modèle consiste à spécifier la catégorie des groupes de ressources ($Groupe\_Ress$) et la catégorie des ressources ($Ressources$) dans le modèle.

Tout comme pour le premier raffinement, il suffit de double-cliquer sur la catégorie des ressources *rCat* afin d'initialiser un éditeur dédié à la création des catégories de ressources uniquement. La Figure 7.11 présente le modèle obtenu. Toutes les catégories y sont créées automatiquement sauf la catégorie *Ressources*.

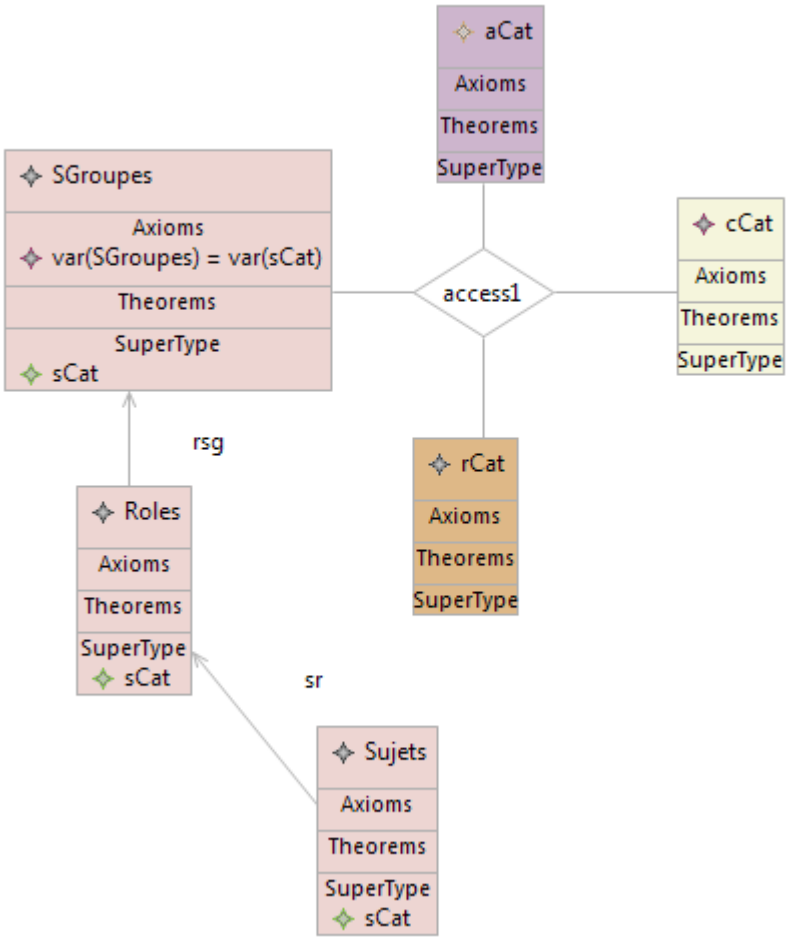

FIGURE 7.10 – Premier raffinement

Conformément à la modélisation des chapitres précédents, nous nous arrêtons à ce niveau de raffinement. La construction des règles de contrôle

d'accès est simillaire. Il faudra toutefois prendre soin d'écrire les noms des catégories qui ne doivent plus être raffinées en commençant par un point (.).

Nous pouvons visualiser la démarche de création des modèles à l'aide du fichier $Pol\_1.catbac$ (Figure 7.6). La Figure 7.12 presente le contenu de ce fichier pour le premier raffinement. Dans le coté gauche de cette figure nous remarquons que nous avons créé un modèle appelé CatBAC dans lequel nous avons ajouté quatre catégories et une association n-aire. Si nous déroulons le noeud qui présente la catégorie SCat (coté droit de la figure), nous remarquons que de nouveaux éléments ont été créés sous ce noeud. Cela signifie que dans notre processus de modélisation nous avons raffiné cette catégorie pour ajouter les éléments mentionnés sous ce noeud.

7.2.3 Création du raffinement d'un sous-mécanisme d'accès

Pour obtenir la représentation graphique du raffinement que nous avons éffectué à la section précédente, nous devons utiliser le menu CatBAC (Figure 7.6). Sous ce menu nous avons deux sous-menus (Figure 7.13). Pour activer le sous-menu *Create refinements*, il suffit de sélectionner le noeud CatBAC comme indiqué à la Figure 7.13. Une fois le noeud CatBAC sélectionné, nous cliquons sur le sous-menu *Create refinements* et obtenons la succession du raffinement du premier sous-mécanisme (Figure 7.14). Deux nouveaux éléments sont alors générés avec les noms $CatBAC\_sCatRef$ et $CatBAC\_sCat\_rCatRef$. Ces noms peuvent être modifiés dans la vue des propriétés (Figure 7.15).

La nouvelle succession du raffinement est présentée à la Figure 7.16. Un double-clique sur chaque noeud nous permet de visualiser respectivement le diagramme de la Figure 7.7 pour le noeud $CatBAC$, le diagramme de la Figure 7.10 pour le noeud $SousMec1$ et le diagramme de la Figure 7.11 pour le noeud $SousMecC1$.

Nous venons donc de modéliser par raffinement le sous-mécanisme de la Figure 6.5 du chapitre 6. Une fois la succession du raffinement créée avec le sous-menu *Create refinements*, nous pouvons commencer la modélisation

d'un autre sous-mécanisme. A la fin nous obtenons les successions de raffinements présentées à la Figure 7.17.

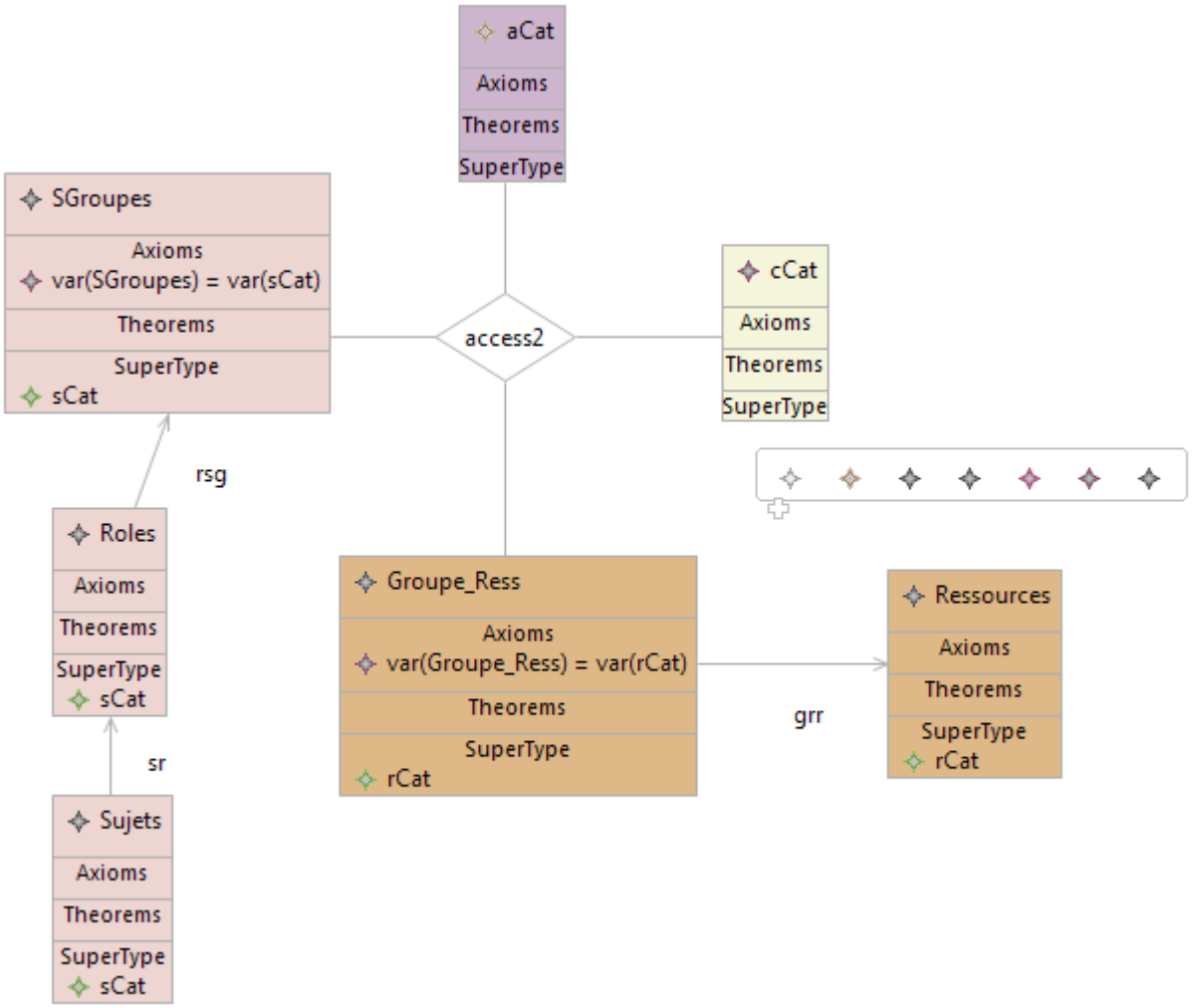

FIGURE 7.11 – Second raffinement

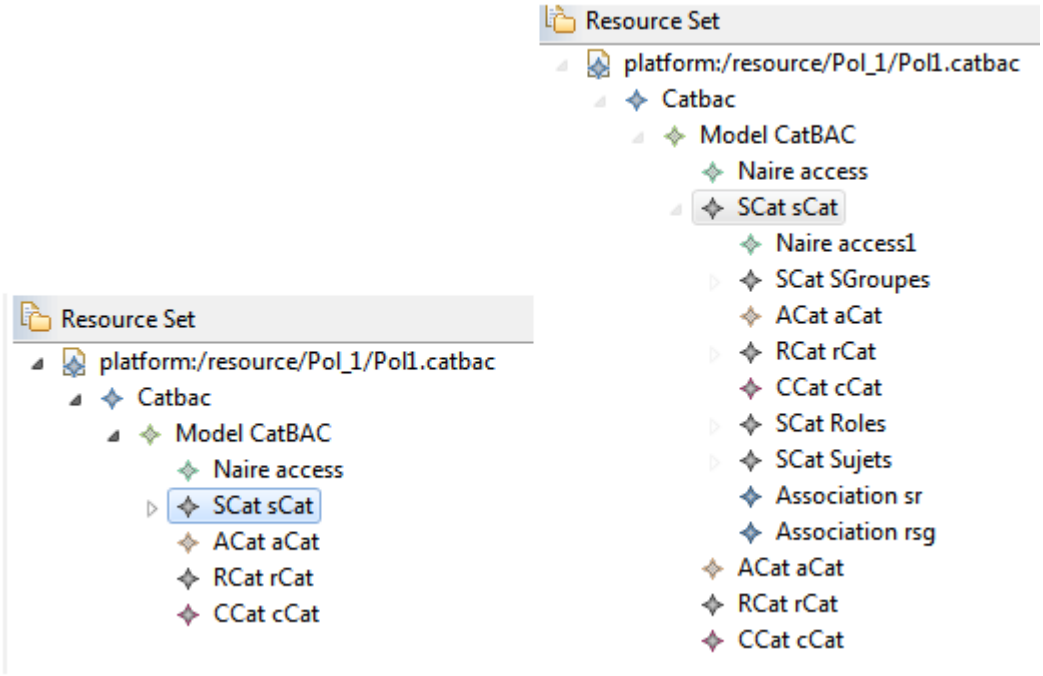

FIGURE 7.12 – Visualisation du raffinement

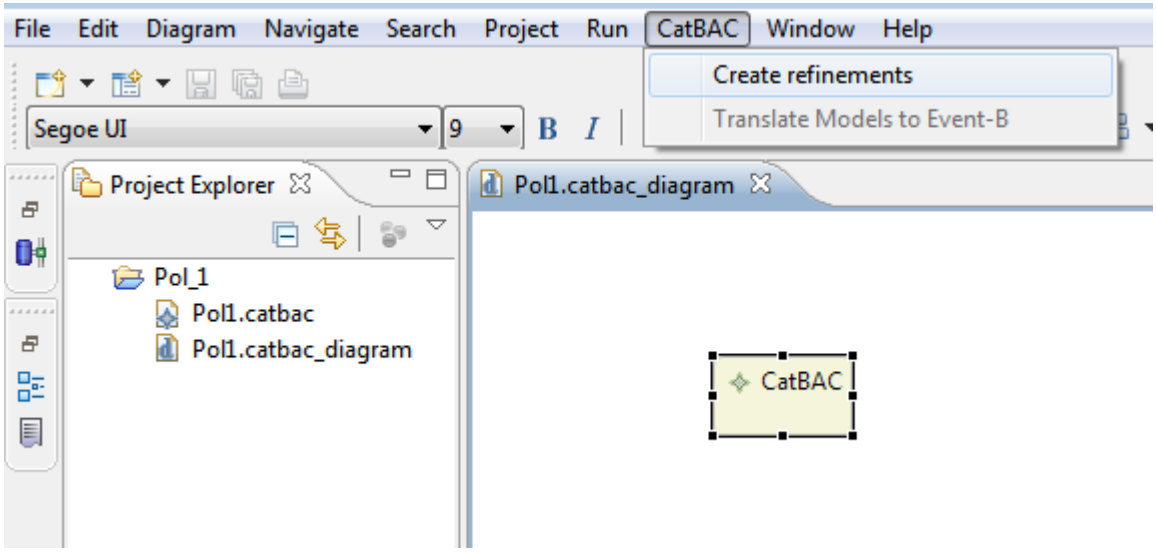

FIGURE 7.13 – Génération du raffinement

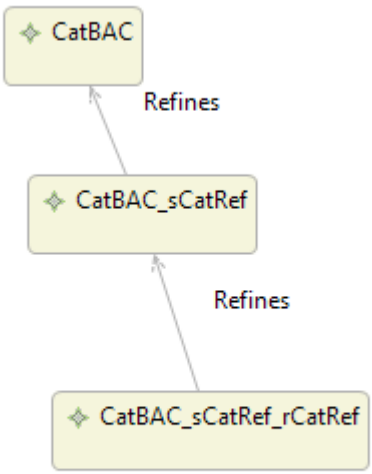

FIGURE 7.14 – Génération du raffinement

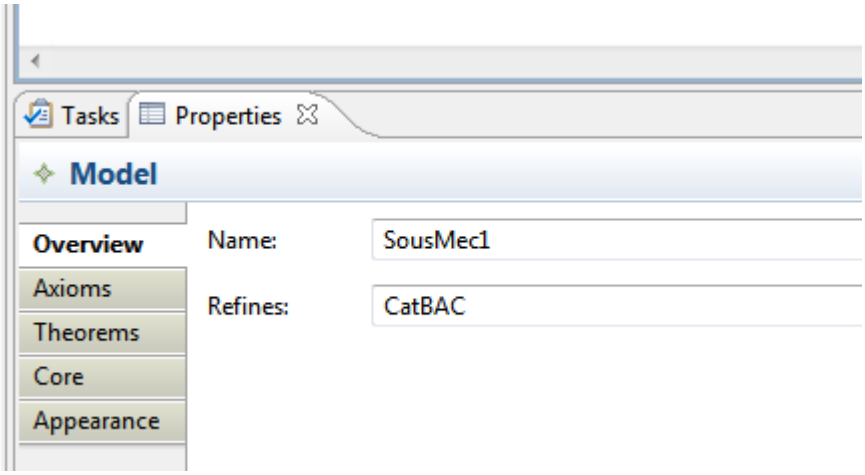

FIGURE 7.15 – Vue des propriétés de CatBAC_Tl

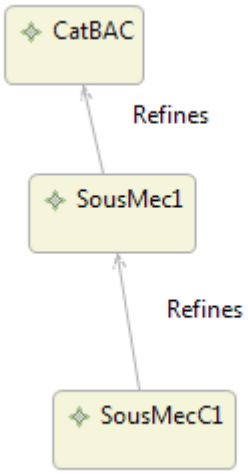

FIGURE 7.16 – Nouvelle succession de raffinement

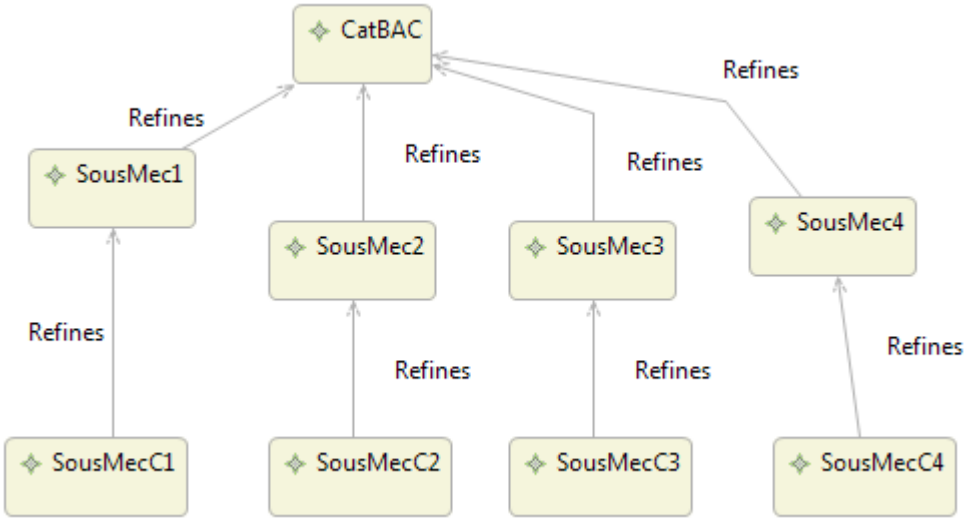

FIGURE 7.17 – Raffinements final

Dans cette figure (Figure 7.17), nous avons éffectué deux raffinements pour chaque sous-mécanisme. Les noeuds *SousMecC2, SousMecC3* et $SousMecC4$

correspondent respectivement aux seconds raffinements de la modélisation des sous-mécanismes des Figures 6.6, 6.7 et 6.8 du chapitre 6.

Les Figures 7.19, 7.20 et 7.21 présentent respectivement les diagrammes modélisés pour ces sous-mécanismes avec *CatBAC_tl*. On peut remarquer dans ces figures, que nous avons utilisé des notations identiques pour désigner les éléments qui ont les mêmes significations.

Si nous analysons chaque sous-mécanisme de la politique Pol_1 (Figures 6.5, 6.6, 6.7, 6.8 du chapitre 6), nous pouvons remarquer qu'au **premier niveau de raffinement**, les sous-modèles de ces sous-mécanismes sont deux à deux identiques. Cela est dû au fait qu'à ce niveau de raffinement, nous traitons uniquement le raffinement de la catégorie des sujets. Par conséquent on obtient un premier sous-modèle avec une spécification des sujets reliés aux rôles et des rôles reliés aux groupes de sujets, et un second sous-modèle avec une spécification des sujets reliés aux groupes de sujets uniquement. On peut faire cette remarque en observant uniquement la modélisation des catégories des sujets des Figures 7.11, 7.19, 7.20, 7.21. Ainsi, le diagramme de modélisation final aurrait été celui de la (Figure 7.18), car les sous-modèles des noeuds *SousMec1* et *SousMec3* de la Figure 7.17 sont identiques. De même les sous-modèles des noeuds *SousMec2* et *SousMec4* sont identiques. Une modélisation comme celle de la Figure 7.18 permet donc d'éliminer des spécifications inutiles et d'alléger le modèle. L'utilisateur doit donc faire attemtion à ce type de situation. Pour nos illustrations, nous avons fait le choix de modéliser comme présenté à la Figure 7.17.

7.2.4 Traduction des modèles vers B-événementiel

Une fois la modélisation des sous-mécanismes terminée (Figure 7.17), nous devons conformément à notre approche, traduire les modèles obtenus vers B-événementiel pour des fins de vérifications. Pour traduire vers B-événemnetiel les sous-modèles CatBAC que nous venons de créer, nous devons sélectionner le sous-menu *Translate Models to Event-B* (Figure 7.13). Ce sous-menu s'active après un clique dans l'espace de travail du diagramme *Pol_1.catbac_diagram*. Une fois ce sous-menu sélectioné, les modèles sont

automatiquement traduits vers B-événementiel comme présenté à la Figure 7.22 et en suivant les opérations que nous avons défini aux chapitres 5 et 6.

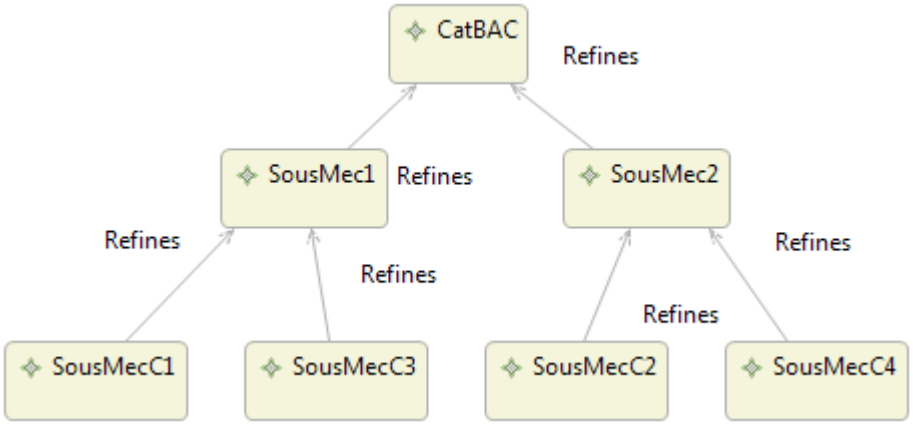

FIGURE 7.18 – Autre raffinements final

Dans la Figure 7.22, nous remarquons qu'un nouveau projet *pol_ 1.EventB* a été créé. Lorsque nous déroulons ce projet, nous avons les contextes et les machines correspondants à chaque noeud du diagramme. Ces contextes et machines ne sont pas encore tous prouvés (Figure7.23) (comme en témoigne le point d'intérrogation devant les noms des machines). Les preuves peuvent être opérées par les outils de preuves présents sur la plateforme Rodin. Dans la Figure 7.24 on peut automatiquement réaliser les preuves en séléctionnant l'option *Launch NewPP (with lasso)*. Cette option permet de prouver les propriétés concernées en utilisant toutes les hypothèses qui y sont associées. Pour des proriétés relativement complexes, les preuves devront se faire manuellement.

La Figure 7.25 montre toutes les obligations de preuves faites pour la machine *CatBAC_ M* (on pourra remarquer que le point d'intérrogation devant le nom de la machine a disparu). Les explications des obligations de preuves sont au chapitre 4 et dans [1, 2]. Nous ne présentons pas tous les contenus des contextes et machines générés afin de ne pas surcharger notre document. Toutefois les contextes *CatBAC_ C, SousMec1_ C, SousMecC1_ C* et les machines *CatBAC_ M, SousMec1_ M, SousMecC1_ M* sont ceux qui ont

été présentés au chapitre 6. Ils correspondent respectivement aux contextes ABSTRACT_C, Less_ABSTRACT_C, CONCRETE_C, et aux machines ABSTRACT_M, Less_ABSTRACT_M, CONCRETE_M.

7.2.5 Conclusion partielle

Tous les objectifs que nous nous sommes fixés dans notre mémoire sont atteints à cette nouvelle étape. Nous avons créé un cadre pour la modélisation des systèmes de contrôle d'accès hybrides. Cependant notre outil dans la version actuelle est expérimental. Certains éléments comme l'expression d'une hiérarchie n'est pas pris en compte. Aussi dans la traduction vers B-événementiel nous avons ajouté uniquement les événements qui sont sous forme de constructeurs.

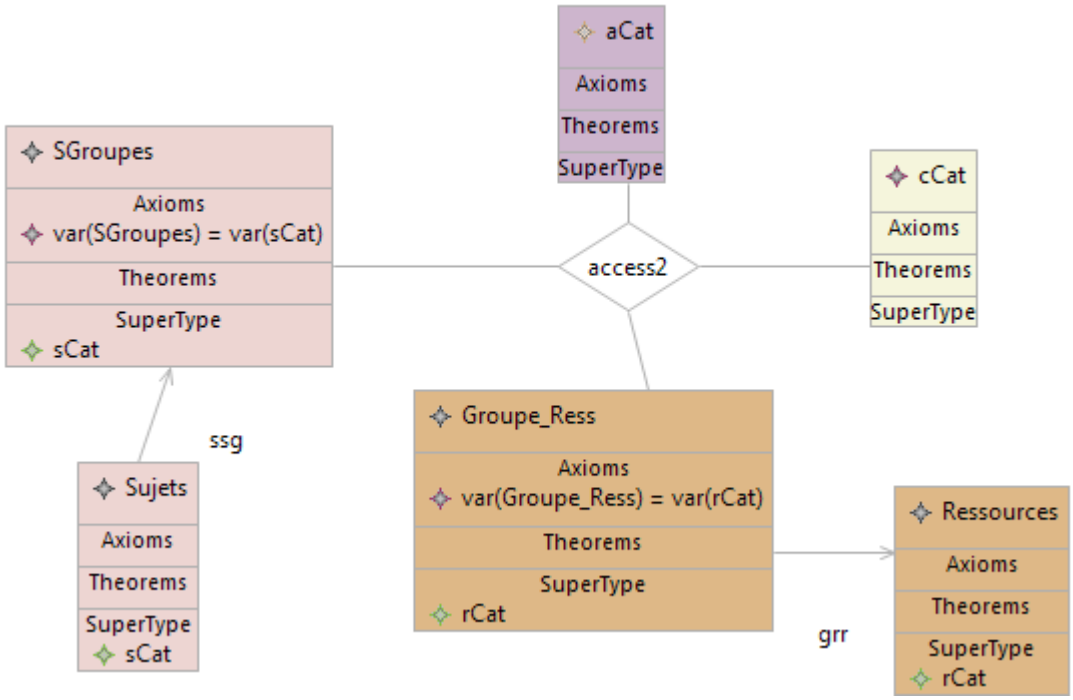

FIGURE 7.19 – Second sous-mecanisme

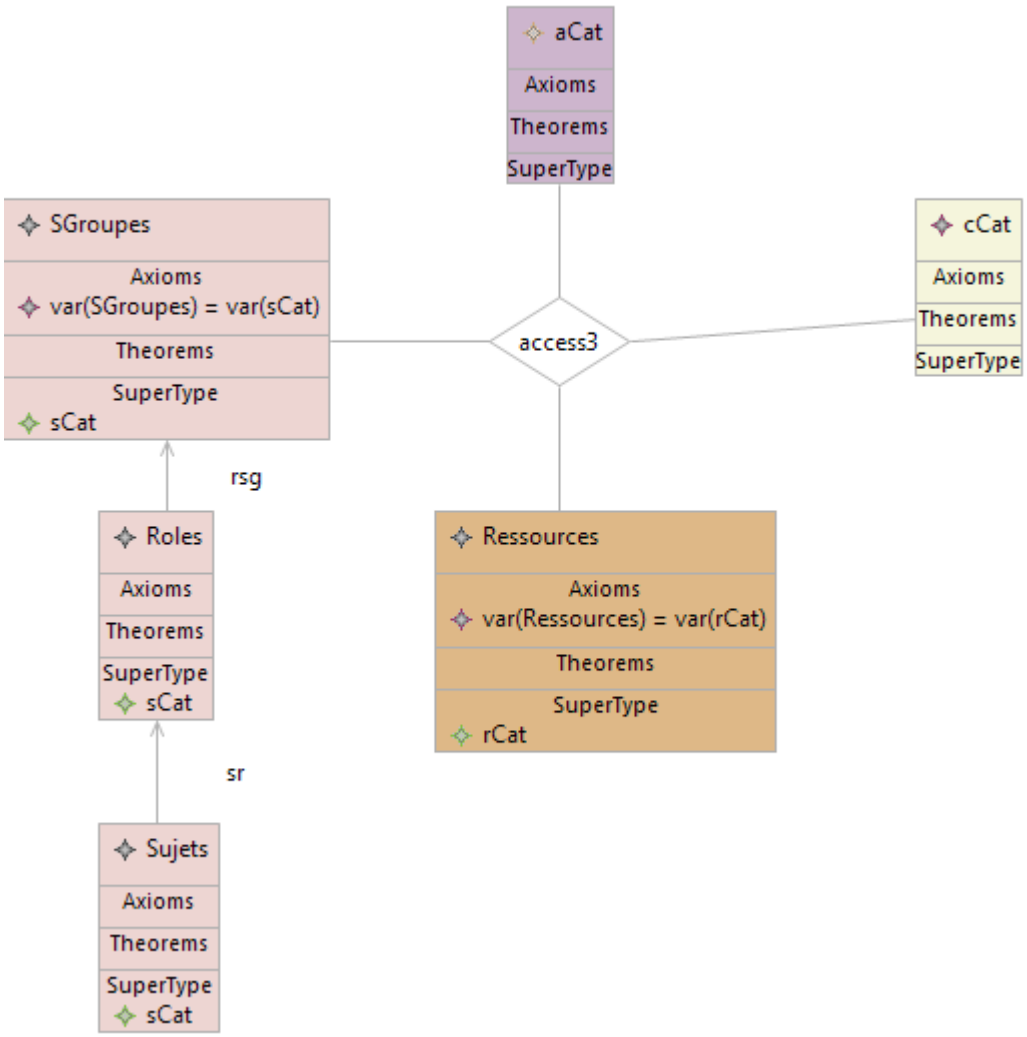

FIGURE 7.20 – Troisième sous-mecanisme

7.3 Composition des sous-modèles B-événementiel

Dans notre approche, nous pouvons composer les sous-modèles B-événementiel à conditions de faire certains changements (Chapitre 6). Ici, nous allons uti-

liser le plugin *Feature Composition* pour éffectuer cette tâche. Ce plugin est conçu pour la plateforme Rodin. Il permet de composer plusieurs machines ou plusieurs contextes. Le manuel d'utilisation de ce plugin peut être retrouvé sur le site web officiel de B-événementiel [29].

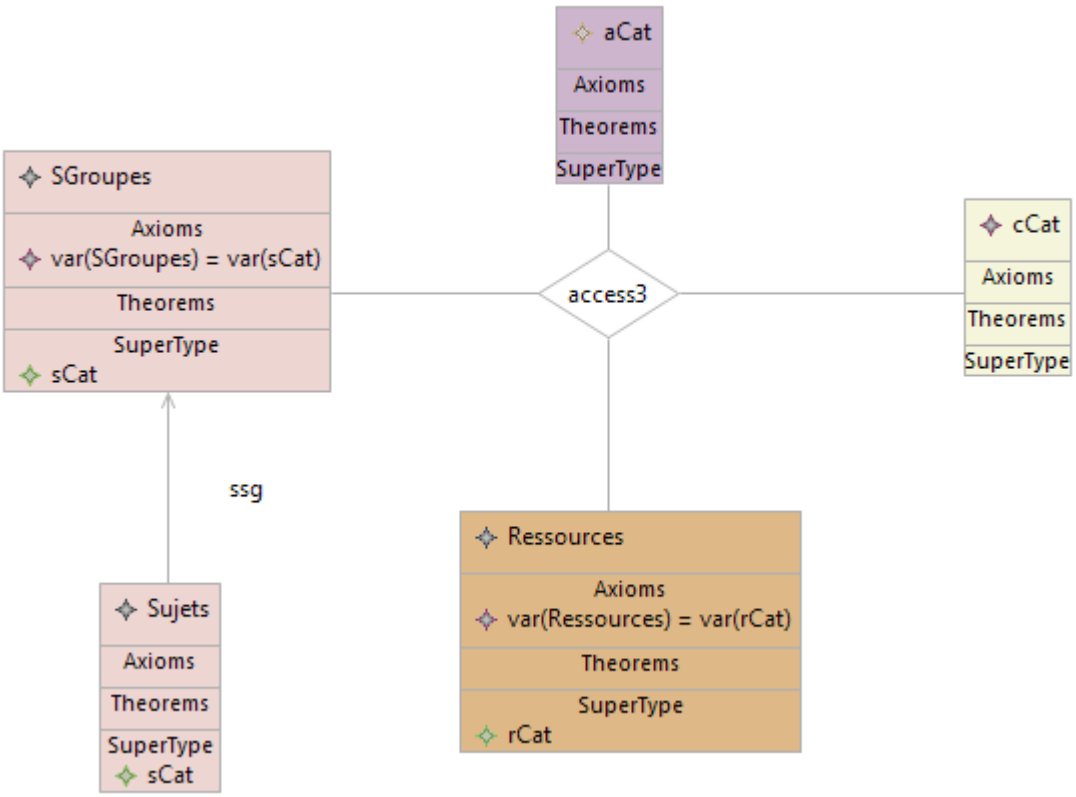

FIGURE 7.21 – Quatrième sous-mecanisme

L'objectif de la composition pour la modélisation de la politique Pol_1 est de montrer que si toutes les spécifications des sous-mécanismes sont mises ensemble, alors elle ne se contredisent pas.

Pour réaliser la composition de nos sous-modèles B-événementiel, nous allons nous référer à la Figure 7.22. Nous allons conformément à cette figure, créer deux niveaux de raffinement en partant du contexte abstrait

$CatBAC\_C$ et de la machine abstraite $CatBAC\_M$ obtenus avec la traduction du modèle abstrait $CatBAC$.

Nous allons ensuite créer un premier niveau de raffinement en composant les contextes *SousMec1_C, SousMec2_C, SousMec3_C, SousMec4_C* et les machines *SousMec1_M, SousMec2_M, SousMec3_M* et $SousMec4\_M$ (Figure 7.22).

Le contexte obtenu sera appelé $Global1\_C$. Ce contexte doit étendre le contexte abstrait $CatBAC\_C$. La machine composée sera appelé $Global1\_M$. Elle doit raffiner la machine abstraite $CatBAC\_M$.

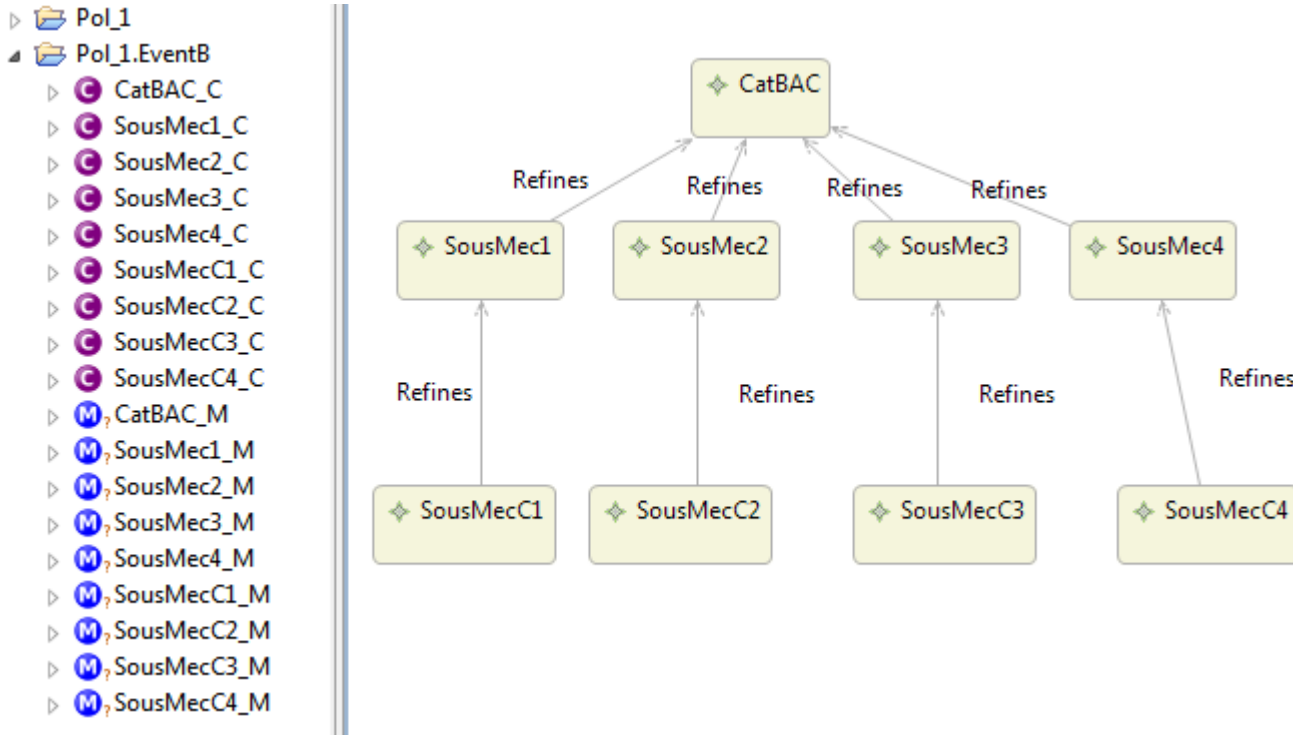

FIGURE 7.22 – Traduction B-événementiel

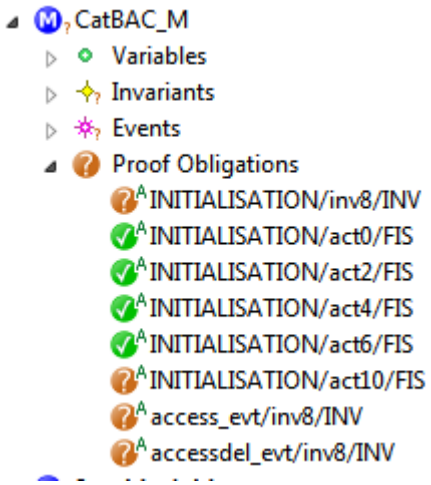

FIGURE 7.23 – Obligations de preuves

Pour créer le second niveau de raffinement, nous allons composer les contextes *SousMecC1_C, SousMecC2_C, SousMecC3_C* et $SousMecC4\_C$ dans un nouveau contexte $GlobalFinal\_C$ qui étend le contexte $Global1\_C$. Ensuite, nous allons composer les machines *SousMecC1_M, SousMecC2_M, SousMecC3_M* et $SousMecC4\_M$ dans une nouvelle machine $GlobalFinal\_M$ qui raffine la machine $Global1\_M$.

La Figure 7.26 montre l'interface de composition pour le contexte $Global1\_C$. Pour réaliser cette composition, nous avons besoin uniquement des contextes sélectionnés dans cette figure, car au premier niveau de raffinement, les modèles sont deux à deux identiques (Section 7.2.3).

La Figure 7.27 montre une partie des éléments sélectionnés pour créer le contexte $Global1\_C$. Une fois les sélections faites et le bouton *Compose* pressé (Figures 7.26 et 7.28), on obtient un nouveau contexte (Figure 7.29). Nous remarquerons que ce contexte étend le contexte abstrait $CatBAC\_C$. Les éléments contenus dans ce contexte ont déja été discutés dans les chapitres précédents. Les labels *not theorem* sont mis pour indiquer les expressions qui ne sont pas à démonter.

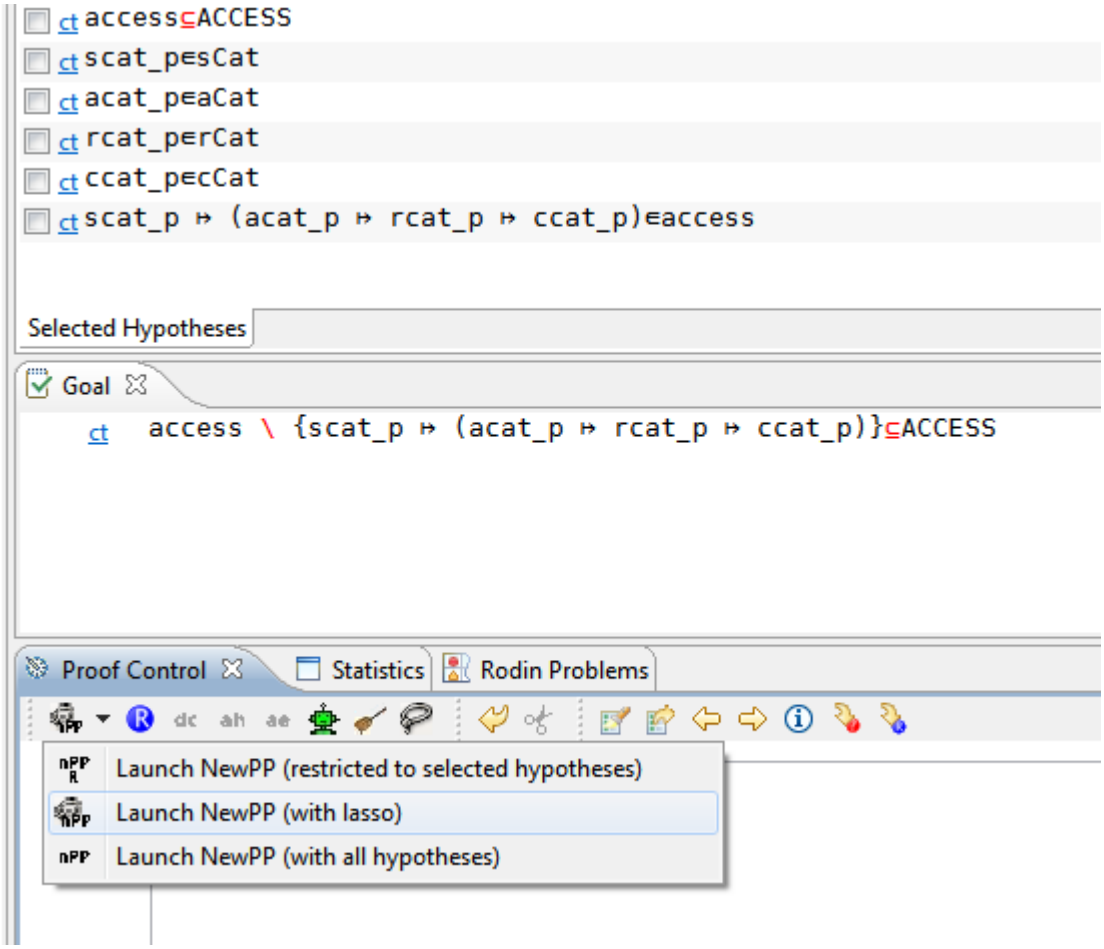

FIGURE 7.24 – Outil de preuves

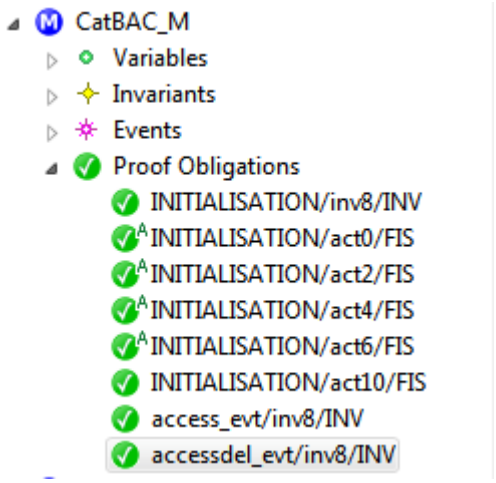

FIGURE 7.25 – Obligations de preuves (2)

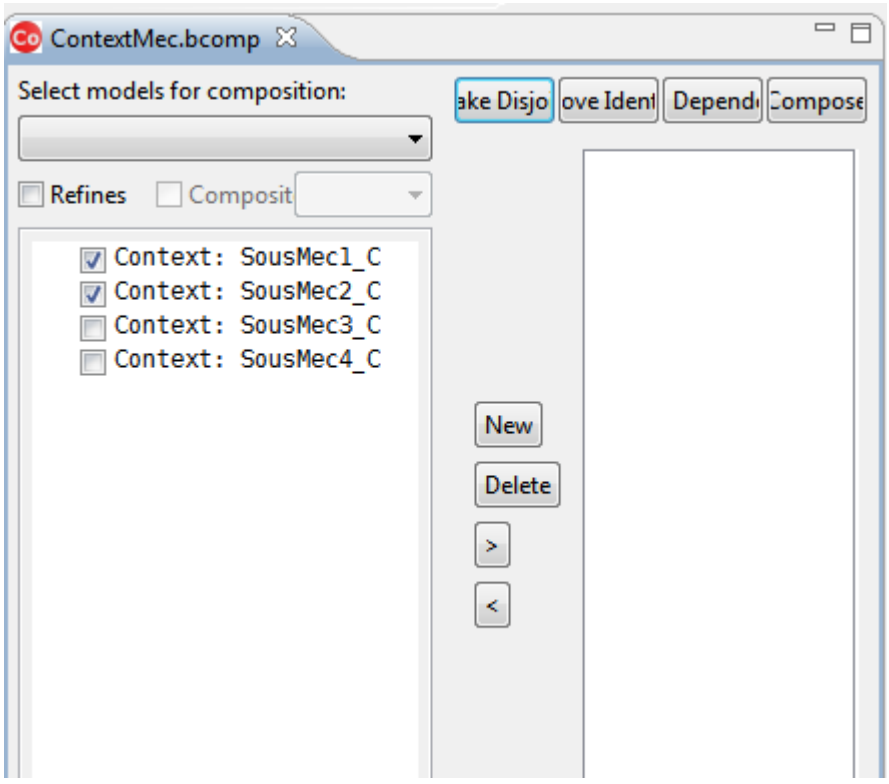

FIGURE 7.26 – Comnposition des contextes du premier raffinement

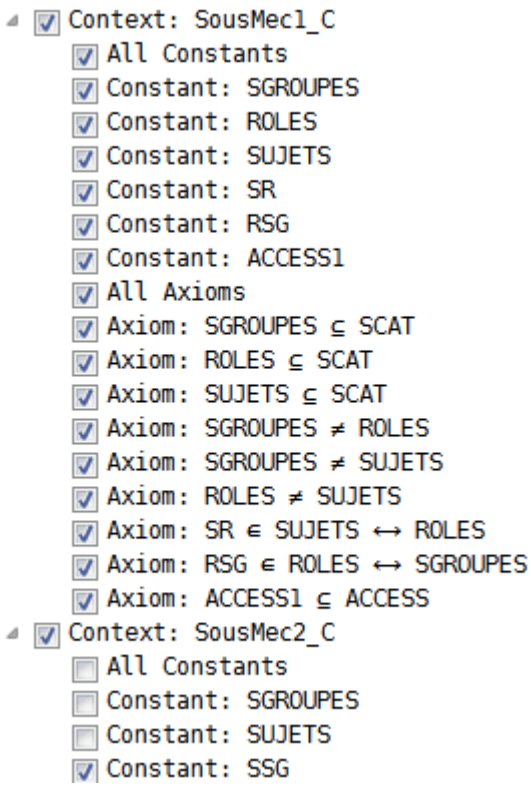

FIGURE 7.27 – Sélection des éléments pour la composition

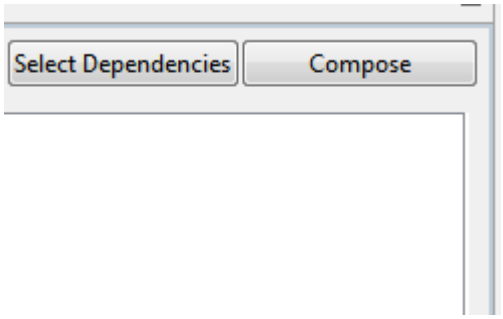

FIGURE 7.28 – Bouton de Composition

```
CONTEXT
    Global1_C    ›
EXTENDS
     CatBAC_C
CONSTANTS
    SGROUPES      ›
    ROLES     ›
    SUJETS    ›
    SR    ›
    RSG   ›
    ACCESS1   ›
    SSG   ›
AXIOMS
    SousMec1_C.axm1:    SGROUPES ⊆ SCAT not theorem ›
    SousMec1_C.axm2:    ROLES ⊆ SCAT not theorem ›
    SousMec1_C.axm3:    SUJETS ⊆ SCAT not theorem ›
    SousMec1_C.axm4:    SGROUPES ≠ ROLES not theorem ›
    SousMec1_C.axm5:    SGROUPES ≠ SUJETS not theorem ›
    SousMec1_C.axm6:    ROLES ≠ SUJETS not theorem ›
    SousMec1_C.axm7:    SR ∈ SUJETS ↔ ROLES not theorem ›
    SousMec1_C.axm8:    RSG ∈ ROLES ↔ SGROUPES not theorem ›
    SousMec1_C.axm10:   ACCESS1 ⊆ ACCESS not theorem ›
    SousMec2_C.axm4:    SSG ∈ SUJETS ↔ SGROUPES not theorem ›
END
```

FIGURE 7.29 – Résultat de la composition des contextes du premier raffinement

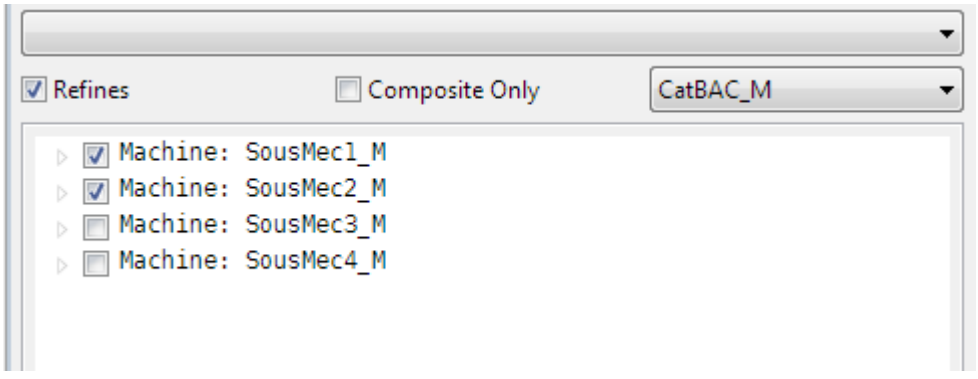

FIGURE 7.30 – Composition des machines du premier raffinement

```
MACHINE
    GlobalFinal_M   ›
REFINES
     Global1_M
SEES
     GlobalFinal_C
VARIABLES
    SGroupes     private ›
    aCat     private ›
    Groupe_Ress  private ›
    cCat     private ›
    Roles    private ›
    Sujets   private ›
    Ressources   private ›
    access2  private ›
    sr   private ›
    rsg  private ›
    grr  private ›
    ssg  private ›
    access3  private ›
INVARIANTS
    SousMecC1_M.inv0:   SGroupes ⊆ SGROUPES not theorem ›
    SousMecC1_M.inv2:   aCat ⊆ ACAT not theorem ›
    SousMecC1_M.inv4:   Groupe_Ress ⊆ GROUPE_RESS not theorem
    SousMecC1_M.inv5:   Groupe_Ress = rCat not theorem ›
    SousMecC1_M.inv6:   cCat ⊆ CCAT not theorem ›
    SousMecC1_M.inv8:   Roles ⊆ ROLES not theorem ›
    SousMecC1_M.inv10:  Sujets ⊆ SUJETS not theorem ›
    SousMecC1_M.inv12:  Ressources ⊆ RESSOURCES not theorem ›
    SousMecC1_M.inv14:  access2 ⊆ ACCESS2 not theorem ›
    SousMecC1_M.inv15:  access2 = access1 not theorem ›
    SousMecC1_M.inv17:  sr ⊆ SR not theorem ›
    SousMecC1_M.inv19:  rsg ⊆ RSG not theorem ›
    SousMecC1_M.inv21:  grr ⊆ GRR not theorem ›
    SousMecC2_M.inv15:  ssg ⊆ SSG not theorem ›
    SousMecC3_M.inv12:  access3 ⊆ ACCESS3 not theorem ›
```

FIGURE 7.31 – Résultat de la composition des machines du second raffinement

La composition des machines du premier niveau de raffinement est similaire (Figure 7.30). La même procédure est appliquée pour composer les contextes et les machines du second niveau de raffinement. Toutefois dans ce cas, la machine finale $GlobalFinal\_M$ (Figure 7.31) devra être modifiée au niveau

de la spécification de certains événements, car des incohérences ont été détectées. Ces incohérences sont dûes aux choix de modélisation que nous avons fait. Nous les expliquons dans la sous-section qui suit. Les spécifications complètes des machines que nous discutons sont fournies en annexes.

7.3.1 Modification des événements dans les machines composées

Considérons l'événement de création des permissions dans la sous-machine $SousMecC4\_M$. Cet événement $access3\_evt$ est ci-dessous :

Event $access3\_evt \widehat{=}$
refines $access1\_evt$

any

$sgroupes\_p$ ... :paramètre groupe de sujets
$acat\_p$ ..... :paramètre action
$ressources\_p$:paramètre ressource
$ccat\_p$ ..... :paramètre contexte

where

grd1 : $sgroupes\_p \in SGroupes$
grd2 : $acat\_p \in aCat$
grd3 : $ressources\_p \in Ressources$
grd4 : $ccat\_p \in cCat$
grd5 : $sgroupes\_p \mapsto (acat\_p \mapsto ressources\_p \mapsto ccat\_p) \in ACCESS3 \setminus access3$

with

rcat_p : $ressources\_p = rcat\_p$

then

act12 : $access3 := access3 \cup \{sgroupes\_p \mapsto (acat\_p \mapsto ressources\_p \mapsto ccat\_p)\}$

end

$access3\_evt$ raffine l'événement abstrait $access1\_evt$. De même l'événement $access2\_evt$ de la sous-machine $SousMecC2\_M$ raffine l'événement abstrait $access1\_evt$. La spécification de $access2\_evt$ est ci-dessous :

Event $access2\_evt \widehat{=}$
refines $access1\_evt$

any

$sgroupes\_p$ ..... :paramètre groupe de sujets
$acat\_p$ ..... :paramètre action
$groupe\_ress\_p$: paramètre ressource
$ccat\_p$ ..... :paramètre contexte

where

grd1 : $sgroupes\_p \in SGroupes$
grd2 : $acat\_p \in aCat$
grd3 : $groupe\_ress\_p \in Groupe\_Ress$
grd4 : $ccat\_p \in cCat$
grd5 : $sgroupes\_p \mapsto (acat\_p \mapsto groupe\_ress\_p \mapsto ccat\_p) \in ACCESS2 \setminus access2$

with

rcat_p : $groupe\_ress\_p = rcat\_p$

then

act14 : $access2 := access2 \cup \{sgroupes\_p \mapsto (acat\_p \mapsto groupe\_ress\_p \mapsto ccat\_p)\}$

end

L'architecture de ce raffinement d'événements est présentée dans la Figure 7.32. Dans cette architecture l'événement $access\_evt$ de la machine $CatBAC\_M$ est raffiné dans les deux machines $SousMec2\_M$ et $SousMec4\_M$

pour produire respectivement l'événement $access1\_evt$. L'événement $access1\_evt$ est ensuite raffiné pour produire les événements $access2\_evt$ de la machine $SousMecC2\_M$ et $access3\_evt$ de la machine $SousMecC4\_M$.

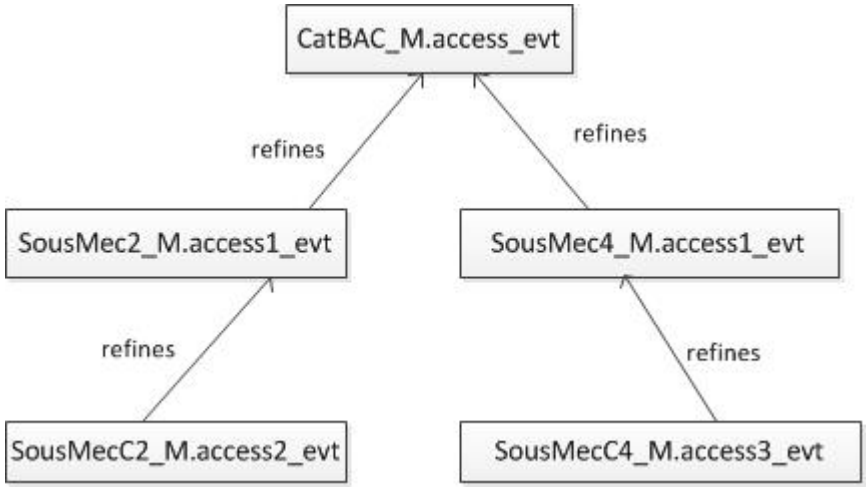

FIGURE 7.32 – Architecture du raffinement des événements avant la composition

Pour les deux événements $access2\_evt$ et $access3\_evt$, si nous nous référons aux interprétations formelles que nous avons élaboré au chapitre 5, alors les variables $access2$ et $access3$ (variables modifiées dans chacun de ces événements) simulent chacun la variable abstraite $access1$ qui est modifiée dans l'événement $access1\_evt$. Cela implique que dans les deux machines contenant ces événements, nous avons modélisé les propriétés suivantes : $access2 = access1$, $access1 = access3$.

Par conséquent, dans la machine de composition finale $GlobalFinal\_M$, nous ne pouvons pas modéliser ces deux propriétés ensemble. Cela reviendrait à écrire que $access2 = access3$ ce qui n'est pas le cas, du fait que les associations leurs correspondants n'ont pas les mêmes significations formelles.

Pour éviter cette situation, nous allons modifier l'architecture de ce raffinement comme présentée à la Figure 7.33. Dans cette nouvelle architecture, l'événement $access2\_evt$ de la machine composée $GlobalFinal\_M$ raffine

l'événement $access1\_evt$ de la machine $Global1\_M$ (issue de la composition des machines du premier niveau de raffinent) qui raffine à son tour l'évenement $access\_evt$ de la machine abstraite $CatBAC\_M$.

L'évenemnet $access3\_evt$ qui raffinait l'événement $access1\_evt$ dans l'architecture du raffinement de la Figure 7.32 raffine maintenant l'événement *skip* (événement qui ne fait rien. Chapitre 4). Par conséquent dans la composition des machines nous considérons que l'événement $access3\_evt$ est un nouvel événement créé au second niveau de raffinement. Ainsi nous pouvons supprimer l'invariant de collage $access1 = access3$ de la machine $GlobalFinal\_M$ mais conserver cependant la structure n-aire de cet événement.

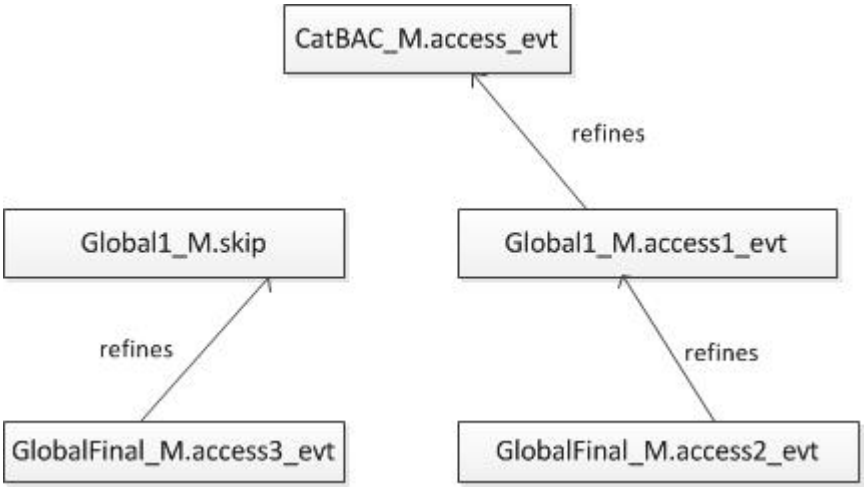

FIGURE 7.33 – Architecture du raffinement des événements après la composition

Pour obtenir cette nouvelle architecture, il suffit de modifier l'événement $access3\_evt$ de la machine $GlobalFinal\_M$ comme mentionné ci-dessous. Nous allons également supprimer de la machine $GlobalFinal\_M$, la propriété $access1 = access3$. Par cette suppression, nous indiquons que seulle la variable $access\_2$ peut simuler la variable $access1\_$ et nous évitons ainsi les conflits entre ces varables.

Event $access3\_evt \mathrel{\widehat{=}}$

any

$sgroupes\_p$

$acat\_p$

$ressources\_p$

$ccat\_p$

where

SousMecC3_M.access3_evt.grd1 : $sgroupes\_p \in SGroupes$

SousMecC3_M.access3_evt.grd2 : $acat\_p \in aCat$

SousMecC3_M.access3_evt.grd5 : $sgroupes\_p \mapsto (acat\_p \mapsto ressources\_p \mapsto ccat\_p) \in ACCESS3 \setminus access3$

SousMecC3_M.access3_evt.grd3 : $ressources\_p \in Ressources$

SousMecC3_M.access3_evt.grd4 : $ccat\_p \in cCat$

then

SousMecC3_M.access3_evt.act14 : $access3 := access3 \cup \{sgroupes\_p \mapsto (acat\_p \mapsto ressources\_p \mapsto ccat\_p)\}$

end

Dans la nouvelle spécification de $access3\_evt$, nous avons retiré la clause $refine$ et la clause $with$, car $access3\_evt$ est considéré comme un nouvel événement qui raffine $skip$. Le choix de modifier $access3\_evt$ est aléatoire.

Il suffit donc d'appliquer ce procédé pour obtenir une composition de machines bien formée. De futurs travaux pourront étendre ce concept afin de l'intégrer directement au précésus de traduction des modèles CatBAC vers B-événementiel. Pour notre mémoire nous avions comme objectif de traduire chaque raffinement de sous-modèle indépendament (sans tenir compte immédiatement des autres sous-modèles). Nous n'avons donc pas pris en compte ce procédé de façon automatique.

La Figure 7.34 montre une partie des obligations de preuves réalisées pour la machine finale $GlobalFinal\_M$.

Les spécifications complètes des machines B-événementiel que nous avons traité dans ce mémoire sont fournies en annexes. Lorsque les modèles B-événementiel sont obtenus, il peuvent à l'aide d'autres plugins, être traduits dans un langage de programmation (Java, C , C++, etc.).

7.4 Conclusion

Dans ce chapitre, nous avons présenté les fonctionnalités que nous avons développé pour le plugin CatBAC_tl. Ce plugin est un éditeur graphique qui permet de modéliser des systèmes de contrôle d'accès avec CatBAC et de les traduire vers B-événementiel. Nous avons ensuite montré comment des modèles CatBAC construits grâce à notre plugin, peuvent être composés. La version du plugin CatBAC_tl que nous avons présenté dans ce mémoire est expérimentale. D'autres versions plus élaborées pourront être proposées.

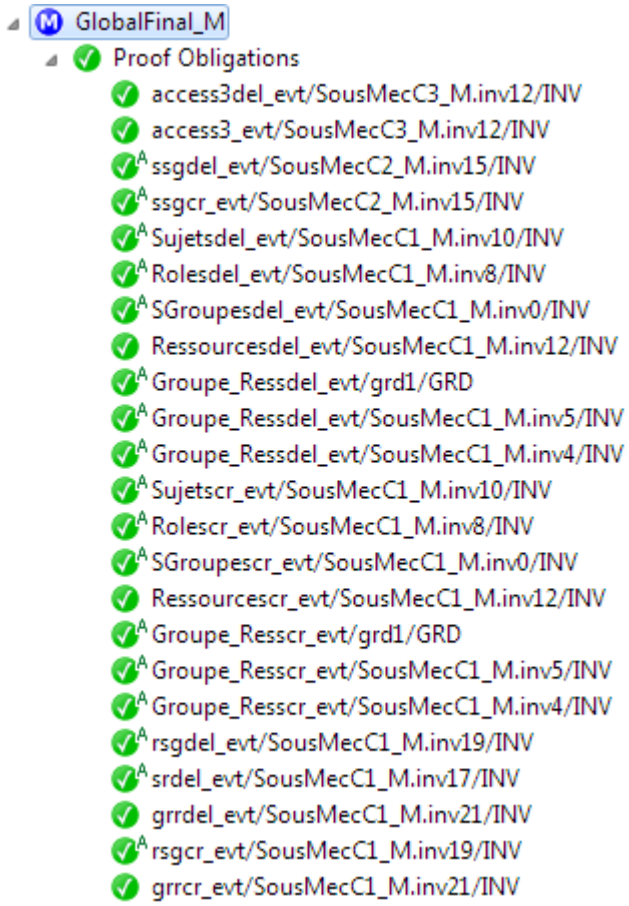

FIGURE 7.34 – Obligations de preuves de la machine finale

Chapitre 8

Conclusion

L'objectif de notre mémoire est de créer un cadre pour la spécification des systèmes de contrôle d'accès hybrides. Pour atteindre cet objectif, nous avons utilisé une méthode de spécification semi-formelle appelée CatBAC et une méthode de spécification formelle appelée B-événementiel. CatBAC est une méthode qui nous permet de spécifier sous forme de diagrammes UML, les besoins en contrôle d'accès d'un système. B-événementiel permet quant à lui de spécifier formellement les exigences d'un système et de prouver que ces exigences répondent bien aux attentes.

Pour élaborer notre processus de modélisation, nous avons combiné ces deux méthodes de modélisation afin de créer des modèles de contrôle d'accès à la fois simples et sûrs par construction. CatBAC nous a permis de créer des modèles simples et facilement compréhensibles grâce à sa notation graphique. B-événementiel nous a permis de spécifier des opérations d'accès et de prouver que ces opérations ne contredisent pas l'ensemble des propriétés établies dans les modèles CatBAC.

La nécessité de trouver des techniques pour modéliser des systèmes de contrôle d'accès hybrides découle du fait que de plus en plus d'organisations ont besoin de mettre en oeuvre des modèles de contrôle d'accès basés sur la combinaison des concepts de différents modèles de contrôle d'accès. La politique de sécurité Pol_1, que nous avons traité dans les chapitres précédents

en est un exemple. Pour mettre en oeuvre cette politique, nous avons fait appel aux concepts *rôle*, *groupe de sujets* et *groupe de ressources*.

Puisque ces différents concepts n'existent pas tous dans un même modèle de contrôle d'accès connu, nous sommes obligés de créer un nouveau modèle qui regroupe ces concepts et qui répond aux besoins de la politique à implémenter.

Des situations comme celle qui a conduit à la création de la politique Pol_1 peuvent découler de l'utilisation de l'infonuagique, d'une fusion d'entreprise, du changement du mode de fonctionnement d'une entreprise, etc.

8.1 Contributions

Nous avons fait trois contributions dans ce mémoire.

- La première contribution a consisté à fournir une description formelle du raffinement des modèles CatBAC. Cette description est nécéssaire car pour prouver formellement les modèles CatBAC, nous avons besoin de les traduire dans le langage formel utilisé par B-événementiel. Aussi, puisque CatBAC utilise une association n-aire (UML), nous devons définir une interpretation de cette association de façon à ce que la construction des modèles se fasse dans un même raisonnement.

- La seconde contribution à consisté à fournir un ensemble de règles pour obtenir les interprétations formelles des modèles CatBAC. L'objectif de cette contribution est de simplifier l'obtention de l'équivalent formel d'un modèle afin de le traduire dans un langage formel.

- La troisième contribution est la création d'un outil de modélisation et de traduction des modèles CatBAC vers B-événementiel. A ce niveau nous avons formalisé une approche de modélisation par décomposition afin d'obtenir des modèles CatBAC qui reflètent le plus possible les exigences d'une politique de sécurité. Nous avons implémenté notre

outil en nous basant sur cette approche et sur les contributions précédemment mentionnées. Pour ce mémoire, notre outil a été développé comme un outil expérimental.

8.2 Limites de notre approche

Notre approche est liée à la méthode B-événemntiel. Elle nécéssite des connaissances de base sur cette méthode. Néanmoins grâce aux interprétations formelles que nous avons établi au Chapitre 5, d'autres méthodes formelles pourront être utilisées. Les propriétés que nous générons dans les modèles CatBAC au moment de leurs traductions vers B-événementiel sont façilement prouvables ; ce n'est pas forcement le cas pour d'autres propriétés qui seront ajoutées aux modèles. Par conséquent, l'utilisateur sera amené à modifier certaines modélisations d'opérations générées (ce fut le cas au Chapitre 7).

Une autre limite de notre approche est qu'elle suggère la décomposition des mécanismes d'accès afin que les modélisations puissent réfléter le plus possible les exigences d'une politique. Dans ce genre de situation l'utilisateur doit faire de son mieux, afin de ne pas créer trop de sous-modèles d'accès, en évitant les décompositions inutiles (Chapitre 6). Il doit modéliser adéquatement afin de repérer les niveaux de raffinements qui sont similaires pour plusieurs sous-modèles.

8.3 Travaux futurs

A partir de nos travaux, plusieurs extensions peuvent être envisagées. Si nous considérons la décomposition des modèles, des moyens peuvent être étudiés afin de réduire le nombre de sous-modèles CatBAC créés pour une politique. Par exemple, toutes les catégorisations des sujets peuvent être représentées dans un seul modèle si on introduit une nouvelle catégorie qui joue le rôle des permissions (Figure 8.1). Sur cette figure, nous voyons que

toutes les possibilités d'associations entre les catégories de sujets peuvent être représentées. On n'a donc pas besoin de substituer l'absence d'une catégorie de sujet par des catégories virtuelles, car chaque catégorie définie dans une politique pourra directement être reliée à la catégorie *Permissions*.

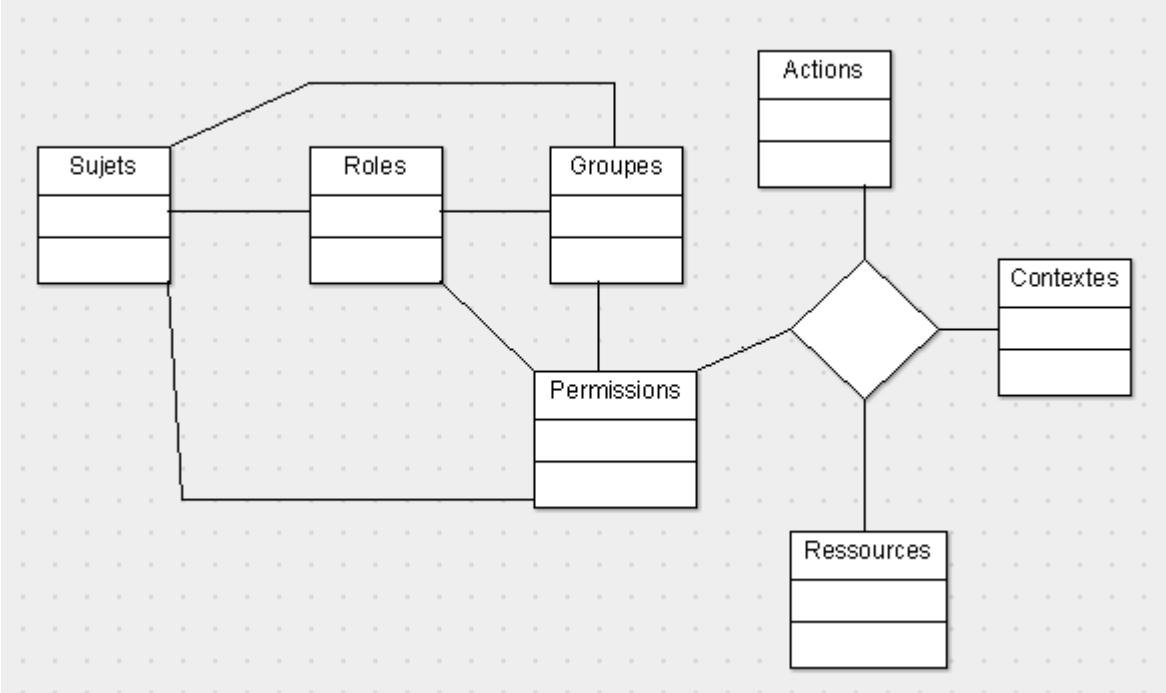

FIGURE 8.1 – Catégorisation des sujets

La modélisation de la Figure 8.1 constitue donc une première façon de réduire le nombre de décompositions d'un modèle en se basant uniquement sur les catégories des sujets. Une extension de notre travail pourait donc être de trouver des composants adéquats qui puissent jouer des fonctions similaires pour les catégories de ressources, d'actions et de contextes.

Une autre extension de notre travail pourrait être d'étudier la fusion des sous-modèles CatBAC avant leurs traduction vers B-événementiel. Dans ce cas, il faudrait définir des opérations qui prennent en compte les gardes

partielles (Chapitre 6) des événements modélisés pour des règles de contrôle d'accès.

Un troisième élément intéressant pour étendre nos travaux serait d'élaborer un langage textuel avec une sémantique formelle, qui permettrait de générer le raffinement graphique que nous avons étudié dans ce mémoire. L'objectif serait de permettre à l'utilisateur d'exprimer dans ce langage textuel des politiques qui soient des interprétations exactes du texte initial de la politque. Dans ce cas, l'utilisateur n'a plus à interpréter la politique avant de la modéliser. Il réécrit simplement cette politique dans un autre langage textuel et obtient le raffinement demandé.

Appendices

Annexe A

Machine B-événementiel CatBAC_M

MACHINE CatBAC_M
SEES CatBAC_C
VARIABLES

$sCat$
$aCat$
$rCat$
$cCat$
$access$

INVARIANTS

inv0 : $sCat \subseteq SCAT$
inv2 : $aCat \subseteq ACAT$
inv4 : $rCat \subseteq RCAT$
inv6 : $cCat \subseteq CCAT$
inv8 : $access \subseteq ACCESS$

EVENTS
Initialisation

begin

act0 : $sCat :\in \mathbb{P}(SCAT)$
act2 : $aCat :\in \mathbb{P}(ACAT)$
act4 : $rCat :\in \mathbb{P}(RCAT)$
act6 : $cCat :\in \mathbb{P}(CCAT)$
act10 : $access :\in \mathbb{P}(ACCESS)$

end

Evénement de création des permissions

Event $access\_evt \mathrel{\widehat{=}}$

any

$scat\_p$ paramètre sCat
$acat\_p$ paramètre aCat
$rcat\_p$ paramètre rCat
$ccat\_p$ paramètre cCat

where

grd1 : $scat\_p \in sCat$

grd2 : $acat\_p \in aCat$

grd3 : $rcat\_p \in rCat$

grd4 : $ccat\_p \in cCat$

grd5 : $scat\_p \mapsto (acat\_p \mapsto rcat\_p \mapsto ccat\_p) \in ACCESS \setminus access$

then

act10 : $access := access \cup \{scat\_p \mapsto (acat\_p \mapsto rcat\_p \mapsto ccat\_p)\}$

end

Evénement de suppression des permissions

Event $accessdel\_evt \widehat{=}$

any

$scat\_p$ paramètre sCat

$acat\_p$ paramètre aCat

$rcat\_p$ paramètre rCat

$ccat\_p$ paramètre cCat

where

grd1 : $scat\_p \in sCat$

grd2 : $acat\_p \in aCat$

grd3 : $rcat\_p \in rCat$

grd4 : $ccat\_p \in cCat$

grd5 : $scat\_p \mapsto (acat\_p \mapsto rcat\_p \mapsto ccat\_p) \in access$

then

act10 : $access := access \setminus \{scat\_p \mapsto (acat\_p \mapsto rcat\_p \mapsto ccat\_p)\}$

end

Evénement de création des sujets

Event $sCatcr\_evt \widehat{=}$

any

$scat\_p$ paramètre sCat

where

grd1 : $scat\_p \in SCAT \setminus sCat$

then

act1 : $sCat := sCat \cup \{scat\_p\}$

end

Evénement de création des actions

Event $aCatcr\_evt \widehat{=}$

any

$acat\_p$ paramètre aCat

where

grd1 : $acat\_p \in ACAT \setminus aCat$

then

act1 : $aCat := aCat \cup \{acat\_p\}$

end

Evénement de création des ressources

Event $rCatcr\_evt \widehat{=}$

any

$rcat\_p$ paramètre rCat

where

grd1 : $rcat\_p \in RCAT \setminus rCat$

then

act1 : $rCat := rCat \cup \{rcat\_p\}$

end

Evénement de création des contextes

Event $cCatcr\_evt \widehat{=}$

any

$ccat\_p$ paramètre cCat

where

grd1 : $ccat\_p \in CCAT \setminus cCat$

then

act1 : $cCat := cCat \cup \{ccat\_p\}$

end

Evénement de suppression des sujets

Event $sCatdel\_evt \widehat{=}$

any

$scat\_p$

where

grd1 : $scat\_p \in sCat$

then

act1 : $sCat := sCat \setminus \{scat\_p\}$

end

Evénement de suppression des actions

Event $aCatdel\_evt \widehat{=}$

any

$acat\_p$

where

grd1 : $acat\_p \in aCat$

then

act1 : $aCat := aCat \setminus \{acat\_p\}$

end

Evénement de suppression des ressources

Event $rCatdel\_evt \widehat{=}$

any

$rcat\_p$

where

grd1 : $rcat\_p \in rCat$

then

act1 : $rCat := rCat \setminus \{rcat\_p\}$

end

Evénement de suppression des contextes

Event $cCatdel\_evt \mathrel{\widehat{=}}$

any

$ccat\_p$

where

grd1 : $ccat\_p \in cCat$

then

act1 : $cCat := cCat \setminus \{ccat\_p\}$

end

END

Annexe B

Machine B-événementiel SousMec1_M

MACHINE SousMec1_M
REFINES CatBAC_M
SEES SousMec1_C
VARIABLES

$SGroupes$: groupes de sujets

$aCat$: actions
$rCat$: ressources
$cCat$: contextes
$Roles$: rôles
$Sujets$: sujets
$access1$: permissions
sr : couple sujet-rôle
rsg : couple rôle-groupe de sujet

INVARIANTS

définition des variables d'état

inv0 : $SGroupes \subseteq SGROUPES$
inv1 : $SGroupes = sCat$
inv2 : $aCat \subseteq ACAT$
inv4 : $rCat \subseteq RCAT$
inv6 : $cCat \subseteq CCAT$
inv8 : $Roles \subseteq ROLES$
inv10 : $Sujets \subseteq SUJETS$
inv12 : $access1 \subseteq ACCESS1$
inv13 : $access1 = access$
inv15 : $sr \subseteq SR$
inv17 : $rsg \subseteq RSG$

EVENTS
Initialisation
begin
with

sCat' : $sCat' = SGroupes'$

access' : $access' = access1'$

act0 : $SGroupes :\in \mathbb{P}(SGROUPES)$

act2 : $aCat :\in \mathbb{P}(ACAT)$

act4 : $rCat :\in \mathbb{P}(RCAT)$

act6 : $cCat :\in \mathbb{P}(CCAT)$

act8 : $Roles :\in \mathbb{P}(ROLES)$

act10 : $Sujets :\in \mathbb{P}(SUJETS)$

act14 : $access1 :\in \mathbb{P}(ACCESS1)$

act16 : $sr :\in \mathbb{P}(SR)$

act18 : $rsg :\in \mathbb{P}(RSG)$

end

Evénement de création des permissions

Event *access1_evt* $\widehat{=}$
refines *access_evt*

any

$sgroupes\_p$: paramètre SGroupes

$acat\_p$: paramètre aCat

$rcat\_p$: paramètre rCat

$ccat\_p$: paramètre cCat

where

grd1 : $sgroupes\_p \in SGroupes$

grd2 : $acat\_p \in aCat$

grd3 : $rcat\_p \in rCat$

grd4 : $ccat\_p \in cCat$

grd5 : $sgroupes\_p \mapsto (acat\_p \mapsto rcat\_p \mapsto ccat\_p) \in ACCESS1 \setminus access1$

with

scat_p : $sgroupes\_p = scat\_p$

then

act14 : $access1 := access1 \cup \{sgroupes\_p \mapsto (acat\_p \mapsto rcat\_p \mapsto ccat\_p)\}$

end

Evénement de suppression des permissions

Event *access1del_evt* $\widehat{=}$
refines *accessdel_evt*

any

$sgroupes\_p$

$acat\_p$

$rcat\_p$

$ccat\_p$

where

grd1 : $sgroupes\_p \in SGroupes$

grd2 : $acat\_p \in aCat$

grd3 : $rcat\_p \in rCat$

grd4 : $ccat\_p \in cCat$

grd5 : $sgroupes\_p \mapsto (acat\_p \mapsto rcat\_p \mapsto ccat\_p) \in access1$

with

scat_p : $sgroupes\_p = scat\_p$

then

act14 : $access1 := access1 \setminus \{sgroupes\_p \mapsto (acat\_p \mapsto rcat\_p \mapsto ccat\_p)\}$

end

Evénement de création des couples sujets - rôles

Event $srcr\_evt \widehat{=}$

any

$sujets\_p$:un sujet

$roles\_p$:un rôle

where

grd1 : $sujets\_p \in Sujets$

grd2 : $roles\_p \in Roles$

grd3 : $sujets\_p \mapsto roles\_p \in SR \setminus sr$

then

act1 : $sr := sr \cup \{sujets\_p \mapsto (roles\_p)\}$

end

Evénement de création des couples rôles - groupe de sujets

Event $rsgcr\_evt \widehat{=}$

any

$roles\_p$: un rôle

$sgroupes\_p$: un groupe de sujet

where

grd4 : $roles\_p \in Roles$

grd5 : $sgroupes\_p \in SGroupes$

grd6 : $roles\_p \mapsto sgroupes\_p \in RSG \setminus rsg$

then

act1 : $rsg := rsg \cup \{roles\_p \mapsto (sgroupes\_p)\}$

end

Evénement de suppression des couples sujets - rôles

Event $srdel\_evt \widehat{=}$

any

$sujets\_p$

$roles\_p$

where

grd1 : $sujets\_p \in Sujets$

grd2 : $roles\_p \in Roles$

grd3 : $sujets\_p \mapsto roles\_p \in sr$

then

act1 : $sr := sr \setminus \{sujets\_p \mapsto (roles\_p)\}$

end

Evénement de suppression des couples rôles - groupe de sujets

Event $rsgdel\_evt \mathrel{\widehat{=}}$

any

$roles\_p$: un rôle

$sgroupes\_p$: un groupe de sujets

where

grd4 : $roles\_p \in Roles$

grd5 : $sgroupes\_p \in SGroupes$

grd6 : $roles\_p \mapsto sgroupes\_p \in rsg$

then

act1 : $rsg := rsg \setminus \{roles\_p \mapsto (sgroupes\_p)\}$

end

Evénement de création des groupe de sujets

Event $SGroupescr\_evt \mathrel{\widehat{=}}$ C

refines $sCatcr\_evt$

any

$sgroupes\_p$

where

grd1 : $sgroupes\_p \in SGROUPES \setminus SGroupes$

with

scat_p : $scat\_p = sgroupes\_p$

then

act1 : $SGroupes := SGroupes \cup \{sgroupes\_p\}$

end

Evénement de création des rôles

Event $Rolescr\_evt \mathrel{\widehat{=}}$

any

$roles\_p$

where

grd1 : $roles\_p \in ROLES \setminus Roles$

then

act1 : $Roles := Roles \cup \{roles\_p\}$

end

Evénement de création des sujets

Event $Sujetscr\_evt \widehat{=}$

Create New Element of type Sujets

any

$sujets\_p$: un sujet

where

grd1 : $sujets\_p \in SUJETS \setminus Sujets$

then

act1 : $Sujets := Sujets \cup \{sujets\_p\}$

end

Evénement de création des actions

Event $aCatcr\_evt \widehat{=}$

extends $aCatcr\_evt$

any

$acat\_p$

where

grd1 : $acat\_p \in ACAT \setminus aCat$

then

act1 : $aCat := aCat \cup \{acat\_p\}$

end

Evénement de création des ressources

Event $rCatcr\_evt \widehat{=}$

extends $rCatcr\_evt$

any

$rcat\_p$

where

grd1 : $rcat\_p \in RCAT \setminus rCat$

then

act1 : $rCat := rCat \cup \{rcat\_p\}$

end

Evénement de création des contextes

Event $cCatcr\_evt \widehat{=}$

extends $cCatcr\_evt$

any

$ccat\_p$

where

grd1 : $ccat\_p \in CCAT \setminus cCat$

then

act1 : $cCat := cCat \cup \{ccat\_p\}$

end

Evénement de suppression des groupe de sujets

Event $SGroupesdel\_evt \widehat{=}$

refines $sCatdel\_evt$

any

$sgroupes\_p$

where

grd1 : $sgroupes\_p \in SGroupes$

with

scat_p : $scat\_p = sgroupes\_p$

then

act1 : $SGroupes := SGroupes \setminus \{sgroupes\_p\}$

end

Evénement de suppression des rôles

Event $Rolesdel\_evt \widehat{=}$

any

$roles\_p$

where

grd1 : $roles\_p \in Roles$

then

act1 : $Roles := Roles \setminus \{roles\_p\}$

end

Evénement de suppression des sujets

Event $Sujetsdel\_evt \widehat{=}$

any

$sujets\_p$

where

grd1 : $sujets\_p \in Sujets$

then

act1 : $Sujets := Sujets \setminus \{sujets\_p\}$

end

Evénement de suppression des actions

Event $aCatdel\_evt \widehat{=}$

extends $aCatdel\_evt$

any

$acat\_p$

where

grd1 : $acat\_p \in aCat$

then

act1 : $aCat := aCat \setminus \{acat\_p\}$

end

Evénement de suppression des ressources

Event $rCatdel\_evt \widehat{=}$

extends $rCatdel\_evt$

any

$rcat\_p$

where

grd1 : $rcat\_p \in rCat$

then

act1 : $rCat := rCat \setminus \{rcat\_p\}$

end

Evénement de suppression des contextes

Event $cCatdel\_evt \widehat{=}$

extends $cCatdel\_evt$

any

$ccat\_p$

where

grd1 : $ccat\_p \in cCat$

then

act1 : $cCat := cCat \setminus \{ccat\_p\}$

end

END

Annexe C

Machine B-événementiel SousMecC1_M

MACHINE SousMecC1_M
REFINES SousMec1_M
SEES SousMecC1_C
VARIABLES

$SGroupes$: groupes de sujets

$aCat$: actions

$Groupe\_Ress$: groupe de ressources

$cCat$: contextes

$Roles$: rôles

$Sujets$: sujets

$Ressources$:ressources

$access2$: permissions

sr : couples sujet - rôle

rsg : couples rôle - groupe de sujet

grr : couples groupe de ressources - ressources

INVARIANTS

inv0 : $SGroupes \subseteq SGROUPES$

inv2 : $aCat \subseteq ACAT$

inv4 : $Groupe\_Ress \subseteq GROUPE\_RESS$

inv5 : $Groupe\_Ress = rCat$

inv6 : $cCat \subseteq CCAT$

inv8 : $Roles \subseteq ROLES$

inv10 : $Sujets \subseteq SUJETS$

inv12 : $Ressources \subseteq RESSOURCES$

inv14 : $access2 \subseteq ACCESS2$

inv15 : $access2 = access1$

inv17 : $sr \subseteq SR$

inv19 : $rsg \subseteq RSG$

inv21 : $grr \subseteq GRR$

EVENTS

Initialisation

begin
with

rCat' : $rCat' = Groupe\_Ress'$

access1' : $access1' = access2'$

act0 : $SGroupes :\in \mathbb{P}(SGROUPES)$

act2 : $aCat :\in \mathbb{P}(ACAT)$
act4 : $Groupe\_Ress :\in \mathbb{P}(GROUPE\_RESS)$

act6 : $cCat :\in \mathbb{P}(CCAT)$
act8 : $Roles :\in \mathbb{P}(ROLES)$
act10 : $Sujets :\in \mathbb{P}(SUJETS)$

act12 : $Ressources :\in \mathbb{P}(RESSOURCES)$

act16 : $access2 :\in \mathbb{P}(ACCESS2)$

act18 : $sr :\in \mathbb{P}(SR)$
act20 : $rsg :\in \mathbb{P}(RSG)$
act22 : $grr :\in \mathbb{P}(GRR)$

end

Evénement de création des permissions

Event *access2_evt* $\widehat{=}$

refines *access1_evt*

any

$sgroupes\_p$: un groupe de sujets
$acat\_p$: une action
$groupe\_ress\_p$:un groupe de ressources
$ccat\_p$: un contexte

where

grd1 : $sgroupes\_p \in SGroupes$
grd2 : $acat\_p \in aCat$
grd3 : $groupe\_ress\_p \in Groupe\_Ress$
grd4 : $ccat\_p \in cCat$
grd5 : $sgroupes\_p \mapsto (acat\_p \mapsto groupe\_ress\_p \mapsto ccat\_p) \in ACCESS2 \setminus access2$

with

rcat_p : $groupe\_ress\_p = rcat\_p$

then

act16 : $access2 := access2 \cup \{sgroupes\_p \mapsto (acat\_p \mapsto groupe\_ress\_p \mapsto ccat\_p)\}$

end

Evénement de suppression des permissions

Event *access2del_evt* $\widehat{=}$
refines *access1del_evt*

any

$sgroupes\_p$

$acat\_p$

$groupe\_ress\_p$

$ccat\_p$

where

grd1 : $sgroupes\_p \in SGroupes$

grd2 : $acat\_p \in aCat$

grd3 : $groupe\_ress\_p \in Groupe\_Ress$

grd4 : $ccat\_p \in cCat$

grd5 : $sgroupes\_p \mapsto (acat\_p \mapsto groupe\_ress\_p \mapsto ccat\_p) \in access2$

with

rcat_p : $groupe\_ress\_p = rcat\_p$

then

act16 : $access2 := access2 \setminus \{sgroupes\_p \mapsto (acat\_p \mapsto groupe\_ress\_p \mapsto ccat\_p)\}$

end

Evénement de création des couples groupe de ressources - ressources

Event $grrcr\_evt \widehat{=}$

any

$groupe\_ress\_p$: un groupe de ressources

$ressources\_p$: une ressource

where

grd1 : $groupe\_ress\_p \in Groupe\_Ress$

grd2 : $ressources\_p \in Ressources$

grd3 : $groupe\_ress\_p \mapsto ressources\_p \in GRR \setminus grr$

then

act1 : $grr := grr \cup \{groupe\_ress\_p \mapsto (ressources\_p)\}$

end

Evénement de création des couples sujet - rôle

Event $srcr\_evt \widehat{=}$

extends $srcr\_evt$

any

$sujets\_p$

$roles\_p$

where

grd1 : $sujets\_p \in Sujets$

grd2 : $roles\_p \in Roles$

grd3 : $sujets\_p \mapsto roles\_p \in SR \setminus sr$

then

act1 : $sr := sr \cup \{sujets\_p \mapsto (roles\_p)\}$

end

Evénement de création des couples rôle - groupe de sujets

Event $rsgcr\_evt \widehat{=}$

extends $rsgcr\_evt$

any

$roles\_p$:un rôle
$sgroupes\_p$:un groupe de sujets

where

grd4 : $roles\_p \in Roles$
grd5 : $sgroupes\_p \in SGroupes$
grd6 : $roles\_p \mapsto sgroupes\_p \in RSG \setminus rsg$

then

act1 : $rsg := rsg \cup \{roles\_p \mapsto (sgroupes\_p)\}$

end

Evénement de suppression des couples groupe de ressources - ressources

Event $grrdel\_evt \widehat{=}$

any

$groupe\_ress\_p$
$ressources\_p$

where

grd1 : $groupe\_ress\_p \in Groupe\_Ress$
grd2 : $ressources\_p \in Ressources$
grd3 : $groupe\_ress\_p \mapsto ressources\_p \in grr$

then

act1 : $grr := grr \setminus \{groupe\_ress\_p \mapsto (ressources\_p)\}$

end

Evénement de suppression des couples sujets - rôles

Event $srdel\_evt \widehat{=}$

extends $srdel\_evt$

any

$sujets\_p$
$roles\_p$

where

grd1 : $sujets\_p \in Sujets$
grd2 : $roles\_p \in Roles$
grd3 : $sujets\_p \mapsto roles\_p \in sr$

then

act1 : $sr := sr \setminus \{sujets\_p \mapsto (roles\_p)\}$

end

Evénement de suppression des couples rôle - groupe de sujets

Event $rsgdel\_evt \widehat{=}$

extends $rsgdel\_evt$

any

$roles\_p$
$sgroupes\_p$

where

grd4 : $roles\_p \in Roles$
grd5 : $sgroupes\_p \in SGroupes$
grd6 : $roles\_p \mapsto sgroupes\_p \in rsg$

then

act1 : $rsg := rsg \setminus \{roles\_p \mapsto (sgroupes\_p)\}$

end

Evénement de création d'un groupe de ressources

Event $Groupe\_Resscr\_evt \widehat{=}$

refines $rCatcr\_evt$

any

$groupe\_ress\_p$

where

grd1 : $groupe\_ress\_p \in GROUPE\_RESS \setminus Groupe\_Ress$

with

rcat_p : $rcat\_p = groupe\_ress\_p$

then

act1 : $Groupe\_Ress := Groupe\_Ress \cup \{groupe\_ress\_p\}$

end

Evénement de création des ressources

Event $Ressourcescr\_evt \widehat{=}$

any

$ressources\_p$

where

grd1 : $ressources\_p \in RESSOURCES \setminus Ressources$

then

act1 : $Ressources := Ressources \cup \{ressources\_p\}$

end

Evénement de création des groupes de sujets

Event $SGroupescr\_evt \widehat{=}$

extends $SGroupescr\_evt$

any

$sgroupes\_p$

where

grd1 : $sgroupes\_p \in SGROUPES \setminus SGroupes$

then

act1 : $SGroupes := SGroupes \cup \{sgroupes\_p\}$

end

Evénement de création des actions

Event $aCatcr\_evt \widehat{=}$

extends $aCatcr\_evt$

any

$acat\_p$

where

grd1 : $acat\_p \in ACAT \setminus aCat$

then

act1 : $aCat := aCat \cup \{acat\_p\}$

end

Evénement de création des contextes

Event $cCatcr\_evt \widehat{=}$

extends $cCatcr\_evt$

any

$ccat\_p$ cCat parmater

where

grd1 : $ccat\_p \in CCAT \setminus cCat$

then

act1 : $cCat := cCat \cup \{ccat\_p\}$

end

Evénement de création des rôles

Event $Rolescr\_evt \widehat{=}$

extends $Rolescr\_evt$

any

$roles\_p$

where

grd1 : $roles\_p \in ROLES \setminus Roles$

then

act1 : $Roles := Roles \cup \{roles\_p\}$

end

Evénement de création de sujets

Event $Sujetscr\_evt \widehat{=}$

extends $Sujetscr\_evt$

any

$sujets\_p$

where

grd1 : $sujets\_p \in SUJETS \setminus Sujets$

then

act1 : $Sujets := Sujets \cup \{sujets\_p\}$

end

Evénement de suppression des groupes de ressources

Event $Groupe\_Ressdel\_evt \widehat{=}$

refines $rCatdel\_evt$

any

$groupe\_ress\_p$

where

grd1 : $groupe\_ress\_p \in Groupe\_Ress$

with

rcat_p : $rcat\_p = groupe\_ress\_p$

then

act1 : $Groupe\_Ress := Groupe\_Ress \setminus \{groupe\_ress\_p\}$

end

Evénement de suppression des ressources

Event $Ressourcesdel\_evt \widehat{=}$

any

$ressources\_p$

where

grd1 : $ressources\_p \in Ressources$

then

act1 : $Ressources := Ressources \setminus \{ressources\_p\}$

end

Evénement de suppression des groupes de sujets

Event $SGroupesdel\_evt \widehat{=}$

extends $SGroupesdel\_evt$

any

$sgroupes\_p$

where

grd1 : $sgroupes\_p \in SGroupes$

then

act1 : $SGroupes := SGroupes \setminus \{sgroupes\_p\}$

end

Evénement de suppression des actions

Event $aCatdel\_evt \mathrel{\widehat{=}}$

extends $aCatdel\_evt$

any

$acat\_p$

where

grd1 : $acat\_p \in aCat$

then

act1 : $aCat := aCat \setminus \{acat\_p\}$

end

Evénement de suppression des contextes

Event $cCatdel\_evt \mathrel{\widehat{=}}$

extends $cCatdel\_evt$

any

$ccat\_p$

where

grd1 : $ccat\_p \in cCat$

then

act1 : $cCat := cCat \setminus \{ccat\_p\}$

end

Evénement de suppression des rôles

Event $Rolesdel\_evt \mathrel{\widehat{=}}$

extends $Rolesdel\_evt$

any

$roles\_p$

where

grd1 : $roles\_p \in Roles$

then

act1 : $Roles := Roles \setminus \{roles\_p\}$

end

Evénement de suppression des sujets

Event $Sujetsdel\_evt \mathrel{\widehat{=}}$

extends $Sujetsdel\_evt$

any

$sujets\_p$ Sujets parmater

where

grd1 : $sujets\_p \in Sujets$

then

act1 : $Sujets := Sujets \setminus \{sujets\_p\}$

end

END

Annexe D

Machine B-événementiel GlobalFinal_M

MACHINE GlobalFinal_M
REFINES Global1_M
SEES GlobalFinal_C
VARIABLES

SGroupes
groupes de sujets

aCat
actions

Groupe_Ress
groupe de ressources

cCat
contextes

Roles
rôles

Sujets
sujets

Ressources
ressources

*access*2
permissions

sr
couples sujet-rôle

rsg
couples rôle - groupe de sujets

grr
couples groupes de ressources - ressources

ssg
couples sujet - groupe de sujets

*access*3
permission

INVARIANTS

SousMecC1_M.inv0 : $SGroupes \subseteq SGROUPES$

SousMecC1_M.inv2 : $aCat \subseteq ACAT$

SousMecC1_M.inv4 : $Groupe\_Ress \subseteq GROUPE\_RESS$

SousMecC1_M.inv5 : $Groupe\_Ress = rCat$

SousMecC1_M.inv6 : $cCat \subseteq CCAT$

SousMecC1_M.inv8 : $Roles \subseteq ROLES$

SousMecC1_M.inv10 : $Sujets$ $subseteqSUJETS$

SousMecC1_M.inv12 : $Res - sources \subseteq RESSOURCES$

SousMecC1_M.inv14 : $access2 \subseteq ACCESS2$

SousMecC1_M.inv15 : $access2 = access1$

SousMecC1_M.inv17 : $sr \subseteq SR$

SousMecC1_M.inv19 : $rsg \subseteq RSG$

SousMecC1_M.inv21 : $grr \subseteq GRR$

SousMecC2_M.inv15 : $ssg \subseteq SSG$

SousMecC3_M.inv12 : $access3 \subseteq ACCESS3$

EVENTS

Initialisation

begin

with

access1' : $access1' = access2'$

rCat' : $rCat' = Groupe\_Ress'$

SousMecC1_M.INITIALISATION.act4 : $Groupe\_Ress :\in \mathbb{P}(GROUPE\_RESS)$

SousMecC1_M.INITIALISATION.act16 : $access2 :\in \mathbb{P}(ACCESS2)$

SousMecC1_M.INITIALISATION.act6 : $cCat :\in \mathbb{P}(CCAT)$

SousMecC1_M.INITIALISATION.act18 : $sr :\in \mathbb{P}(SR)$

SousMecC1_M.INITIALISATION.act8 : $Roles :\in \mathbb{P}(ROLES)$

SousMecC1_M.INITIALISATION.act10 : $Sujets :\in \mathbb{P}(SUJETS)$

SousMecC1_M.INITIALISATION.act22 : $grr :\in \mathbb{P}(GRR)$

SousMecC1_M.INITIALISATION.act12 : $Res - sources :\in \mathbb{P}(RESSOURCES)$

SousMecC1_M.INITIALISATION.act0 : $SGroupes :\in \mathbb{P}(SGROUPES)$

SousMecC1_M.INITIALISATION.act20 : $rsg :\in \mathbb{P}(RSG)$

SousMecC1_M.INITIALISATION.act2 : $aCat :\in \mathbb{P}(ACAT)$

act1 : $ssg :\in \mathbb{P}(SSG)$

act2 : $access3 :\in \mathbb{P}(ACCESS3)$

end

Evénement de création de permissions de la forme (groupe de sujet, action, groupe de ressource, contexte)

Event $access2\_evt \widehat{=}$
refines $access1\_evt$

any

$sgroupes\_p$

$acat\_p$

$groupe\_ress\_p$

$ccat\_p$

where

grd5 : $sgroupes\_p \mapsto (acat\_p \mapsto groupe\_ress\_p \mapsto ccat\_p) \in ACCESS2 \setminus access2$

grd4 : $ccat\_p \in cCat$

grd3 : $groupe\_ress\_p \in Groupe\_Ress$

grd2 : $acat\_p \in aCat$

grd1 : $sgroupes\_p \in SGroupes$

with

rcat_p : $groupe\_ress\_p = rcat\_p$

then

act16 : $access2 := access2 \cup \{sgroupes\_p \mapsto (acat\_p \mapsto groupe\_ress\_p \mapsto ccat\_p)\}$

end

Evénement de supression de permissions de la forme (groupe de sujet, action, groupe de ressource, contexte)

Event $access2del\_evt \widehat{=}$
refines $access1del\_evt$

any

$sgroupes\_p$

$acat\_p$

$groupe\_ress\_p$

$ccat\_p$

where

grd3 : $groupe\_ress\_p \in Groupe\_Ress$

grd4 : $ccat\_p \in cCat$

grd1 : $sgroupes\_p \in SGroupes$

grd2 : $acat\_p \in aCat$

grd5 : $sgroupes\_p \mapsto (acat\_p \mapsto groupe\_ress\_p \mapsto ccat\_p) \in access2$

with

rcat_p : $groupe\_ress\_p = rcat\_p$

then

act16 : $access2 := access2 \setminus \{sgroupes\_p \mapsto (acat\_p \mapsto groupe\_ress\_p \mapsto ccat\_p)\}$

end

Evénement de création des couples groupe de ressources - ressource

Event $grrcr\_evt \widehat{=}$

any

$groupe\_ress\_p$

$ressources\_p$

where

grd1 : $groupe\_ress\_p \in Groupe\_Ress$

grd3 : $groupe\_ress\_p \mapsto ressources\_p \in GRR \setminus grr$

grd2 : $ressources\_p \in Ressources$

then

act1 : $grr := grr \cup \{groupe\_ress\_p \mapsto (ressources\_p)\}$

end

Evénement de création des couples rôle - groupe de sujet

Event $rsgcr\_evt \widehat{=}$
extends $rsgcr\_evt$

any

$roles\_p$

$sgroupes\_p$

where

grd5 : $sgroupes\_p \in SGroupes$

grd4 : $roles\_p \in Roles$

grd6 : $roles\_p \mapsto sgroupes\_p \in RSG \setminus rsg$

then

act1 : $rsg := rsg \cup \{roles\_p \mapsto (sgroupes\_p)\}$

end

Evénement de suppression des couples groupe de ressources - ressource

Event $grrdel\_evt \widehat{=}$

any

$groupe\_ress\_p$

$ressources\_p$

where

grd2 : $ressources\_p \in Ressources$

grd3 : $groupe\_ress\_p \mapsto ressources\_p \in grr$

grd1 : $groupe\_ress\_p \in Groupe\_Ress$

then

act1 : $grr := grr \setminus \{groupe\_ress\_p \mapsto (ressources\_p)\}$

end

Evénement de suppression des couples sujet - rôle

Event $srdel\_evt \widehat{=}$

extends *srdel_evt*

any

sujets_p

roles_p

where

grd3 : $sujets\_p \mapsto roles\_p \in sr$

grd1 : $sujets\_p \in Sujets$

grd2 : $roles\_p \in Roles$

then

act1 : $sr := sr \setminus \{sujets\_p \mapsto (roles\_p)\}$

end

Evénement de suppression des couples rôle - groupe de sujet

Event *rsgdel_evt* $\widehat{=}$
extends *rsgdel_evt*

any

roles_p

sgroupes_p

where

grd6 : $roles\_p \mapsto sgroupes\_p \in rsg$

grd5 : $sgroupes\_p \in SGroupes$

grd4 : $roles\_p \in Roles$

then

act1 : $rsg := rsg \setminus \{roles\_p \mapsto (sgroupes\_p)\}$

end

Evénement de création des groupes de ressources

Event *Groupe_Resscr_evt* $\widehat{=}$
refines *rCatcr_evt*

any

groupe_ress_p

where

grd1 : $groupe\_ress\_p \in GROUPE\_RESS \setminus Groupe\_Ress$

with

rcat_p : $rcat\_p = groupe\_ress\_p$

then

act1 : $Groupe\_Ress := Groupe\_Ress \cup \{groupe\_ress\_p\}$

end

Evénement de création des ressources

Event *Ressourcescr_evt* $\widehat{=}$

any

ressources_p

where

grd1 : $ressources\_p \in RESSOURCES \setminus Ressources$

then

act1 : $Ressources := Ressources \cup \{ressources\_p\}$

end

Evénement de création des groupe de sujet

Event $SGroupescr\_evt \widehat{=}$
extends $SGroupescr\_evt$

any

$sgroupes\_p$

where

grd1 : $sgroupes\_p \in SGROUPES \setminus SGroupes$

then

act1 : $SGroupes := SGroupes \cup \{sgroupes\_p\}$

end

Evénement de création des actions

Event $aCatcr\_evt \widehat{=}$
extends $aCatcr\_evt$

any

$acat\_p$

where

grd1 : $acat\_p \in ACAT \setminus aCat$

then

act1 : $aCat := aCat \cup \{acat\_p\}$

end

Evénement de création des contextes

Event $cCatcr\_evt \widehat{=}$
extends $cCatcr\_evt$

any

$ccat\_p$

where

grd1 : $ccat\_p \in CCAT \setminus cCat$

then

act1 : $cCat := cCat \cup \{ccat\_p\}$

end

Evénement de création des roles

Event $Rolescr\_evt \widehat{=}$
extends $Rolescr\_evt$

any

$roles\_p$

where

grd1 : $roles\_p \in ROLES \setminus Roles$

then

act1 : $Roles := Roles \cup \{roles\_p\}$

end

Evénement de création des sujets

Event $Sujetscr\_evt \widehat{=}$
extends $Sujetscr\_evt$

any

$sujets\_p$

where

grd1 : $sujets\_p \in SUJETS \setminus Sujets$

then

act1 : $Sujets := Sujets \cup \{sujets\_p\}$

end

Evénement de suppression des groupes de ressources

Event $Groupe\_Ressdel\_evt \widehat{=}$
refines $rCatdel\_evt$

any

$groupe\_ress\_p$

where

grd1 : $groupe\_ress\_p \in Groupe\_Ress$

with

rcat_p : $rcat\_p = groupe\_ress\_p$

then

act1 : $Groupe\_Ress := Groupe\_Ress \setminus \{groupe\_ress\_p\}$

end

Evénement de suppression des ressources

Event $Ressourcesdel\_evt \widehat{=}$

any

$ressources\_p$

where

grd1 : $ressources\_p \in Ressources$

then

act1 : $Ressources := Ressources \setminus \{ressources\_p\}$

end

Evénement de suppression des groupe de sujets

Event $SGroupesdel\_evt \widehat{=}$
extends $SGroupesdel\_evt$

any

$sgroupes\_p$

where

grd1 : $sgroupes\_p \in SGroupes$

then

act1 : $SGroupes := SGroupes \setminus \{sgroupes\_p\}$

end

Evénement de suppression des actions

Event $aCatdel\_evt \widehat{=}$
extends $aCatdel\_evt$

any

$acat\_p$ aCat parmater

where

grd1 : $acat\_p \in aCat$

then

act1 : $aCat := aCat \setminus \{acat\_p\}$

end

Evénement de suppression des contextes

Event $cCatdel\_evt \widehat{=}$
extends $cCatdel\_evt$

any

$ccat\_p$

where

grd1 : $ccat\_p \in cCat$

then

act1 : $cCat := cCat \setminus \{ccat\_p\}$

end

Evénement de suppression des rôles

Event $Rolesdel\_evt \widehat{=}$
extends $Rolesdel\_evt$

any

$roles\_p$

where

grd1 : $roles\_p \in Roles$

then

act1 : $Roles := Roles \setminus \{roles\_p\}$

end

Evénement de suppression des sujets

Event $Sujetsdel\_evt \widehat{=}$
extends $Sujetsdel\_evt$

any

$sujets\_p$

where

grd1 : $sujets\_p \in Sujets$

then

act1 : $Sujets := Sujets \setminus \{sujets\_p\}$

end

Evénement de création des couples sujet - groupe de

sujets

Event $ssgcr\_evt \widehat{=}$
extends $ssgcr\_evt$

any

$sujets\_p$
$sgroupes\_p$

where

grd3 : $sujets\_p \mapsto sgroupes\_p \in SSG \setminus ssg$
grd2 : $sgroupes\_p \in SGroupes$
grd1 : $sujets\_p \in Sujets$

then

act1 : $ssg := ssg \cup \{sujets\_p \mapsto (sgroupes\_p)\}$

end

Evénement de suppression des couples sujet - groupe de sujets

Event $ssgdel\_evt \widehat{=}$
extends $ssgdel\_evt$

any

$sujets\_p$
$sgroupes\_p$

where

grd1 : $sujets\_p \in Sujets$
grd3 : $sujets\_p \mapsto sgroupes\_p \in ssg$
grd2 : $sgroupes\_p \in SGroupes$

then

act1 : $ssg := ssg \setminus \{sujets\_p \mapsto (sgroupes\_p)\}$

end

Evénement de création des permissions de la forme (groupe de sujet, action, ressource, contexte)

Event $access3\_evt \widehat{=}$

any

$sgroupes\_p$
$acat\_p$
$ressources\_p$
$ccat\_p$

where

grd1 : $sgroupes\_p \in SGroupes$
grd2 : $acat\_p \in aCat$
grd5 : $sgroupes\_p \mapsto (acat\_p \mapsto ressources\_p \mapsto ccat\_p) \in ACCESS3 \setminus access3$
grd3 : $ressources\_p \in Ressources$
grd4 : $ccat\_p \in cCat$

then

act14 : $access3 := access3 \cup \{sgroupes\_p \mapsto (acat\_p \mapsto ressources\_p \mapsto ccat\_p)\}$

end

Evénement de suppression des permissions de la forme (groupe de sujet, action, ressource, contexte)

Event $access3del\_evt \mathrel{\widehat{=}}$

any

$sgroupes\_p$

$acat\_p$

$ressources\_p$

$ccat\_p$

where

grd4 : $ccat\_p \in cCat$

grd5 : $sgroupes\_p \mapsto (acat\_p \mapsto ressources\_p \mapsto ccat\_p) \in access3$

grd2 : $acat\_p \in aCat$

grd3 : $ressources\_p \in Ressources$

grd1 : $sgroupes\_p \in SGroupes$

then

act14 : $access3 := access3 \setminus \{sgroupes\_p \mapsto (acat\_p \mapsto ressources\_p \mapsto ccat\_p)\}$

end

END

Bibliographie

[1] J-R. Abrial : The B-Book : Assigning programs to meanings. Cambridge University Press, 1996.

[2] J-R. Abrial : Modeling in Event-B : System and Software Engineering. Cambridge University Press, 2010.

[3] J-R. Abrial : B : passé, présent, futur. Technique et Science Informatiques, vol. 22, 89-118, 2003.

[4] T. S. Hoang, D. Basin, J-R. Abrial : Specifying Acces control in Event-B. Departement of Computer Science, (ETH Zurich), 2009.

[5] J. Bendisposto, M. Leuschel, O. Ligot, M. Samia : La validation de modèles Event-B avec le plug-in ProB pour RODIN. Technique et Science Informatiques, vol. 27, 1065-1084, 2008.

[6] N. Benaissa : La composition de protocole de sécurité avec la méthode B événementielle. THESE, Université Henri Poincaré - Nancy I, 2010.

[7] T. Lodderstedt , D. Basin, J. Doser : SecureUML : A UML-Based Modeling Language for Model-Driven Security. Springer, vol. 2460,426-441, 2002.

[8] J. Milhau, A. Idani, R. Laleau, M-A. Labiadh, Y. Ledru, M. Frappier : Combining UML, ASTD and B for the formal specification of an access control filter. ISSE, Num. 4, 303-313, 2011.

[9] N. Slimani, H. Khambhammettu, K. Adi, L. Logrippo :UACML : Unified Access Control Modeling Language. IEEE, 2011.

[10] M. Frappier, F. Gervais, R. Laleau, B. Fraikin, R. St-Denis : Extending statecharts with process algebra operators. ISSE, vol. 4, 285–292, 2008.

[11] M. Frappier, R. St-Denis, EB3 : an entity-based black-box specification method for information systems . Software and Systems Modeling, vol. 2, 134–149, 2003.

[12] R. Laleau, A. Mammar : An Overview of a Method and Its Support Tool for Generating B Specifications from UML Notations. ASE, 269-272, 2000.

[13] A. Idani : Couplage de spécifications B et de descriptions UML pour l'aide aux développements formels des Systèmes d'Information. INFORSID, 577-593, 2006.

[14] M. Butler : Decomposition Structures for Event-B. IFM, 20-38, 2009.

[15] J-R. Abrial, S. Hallerstede : Refinement, Decomposition, and Instantiation of Discrete Models : Application to Event-B. Fundam. Inform, vol. 77, 1-28, 2007.

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

[17] M. A. Harrison, W. L. Ruzzo, J. D. Ullman : Protection in Operating Systems. Communication of the ACM, 461-471, 1976.

[18] D. E. Bell, L. j. LaPadula : Secure computer system : Unified exposition and multics interpretation. IEEE Computer Society Symposium on Research in Security and Privacy, 215-228, 1976.

[19] K. Biba : Integrity Consideration for Secure Computer Systems. The MITRE Corporation, 31-53, 1977.

[20] D. Brewer, M. Nash : The chinese wall security policy. IEEE Computer Society Symposium on Research in Security and Privacy, 215-228, 1989

[21] D. F. Ferraiolo, D. R. Kuhn, R. Chandramouli : Role-Based Access Control (RBAC) : Features and Motivations. 11th Annual Computer Security Applications Conference (ACSAC), New Orleans, Louisianna, USA. 1995.

[22] N. Correa, R. Giandini : A uml extention to specify model refinements. CLEI, 2006.

[23] C. Snook, M. Butler UML-B and Event-B : an integration of languages and tools. The IASTED International Conference on Software Engineering SE2008, 12-14, 2008.

[24] S. Khamadja, K. Adi, L. Logrippo : Designing Flexible Access Control Models for the Cloud. Proceeding SIN '13 Proceedings of the 6th International Conference on Security of Information and Networks, 225-232, 2013.

[25] M. Y. Said, M. Butler, C. Snook : A method of refinement in UML-B. Software & Systems Modeling, 2013.

[26] J-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, L. Voisin : Rodin : an open toolset for modelling and reasoning in Event-B. STTT, vol. 12, 447-466, 2010.

[27] The Eclipse Foundation : eclipse. http ://www.eclipse.org (consulté le 20/11/2014).

[28] The Eclipse Foundation : Graphical Modeling Project (GMP). http ://www.eclipse.org/modeling/gmp/ (consulté le 20/11/2014).

[29] ETH - Team : Event-B and the Rodin Platform. http ://www.event-b.org (consulté le 20/11/2014).