UNIVERSITE JOSEPH FOURIER
SCIENCES.TECHNOLOGIE.SANTÉ

MODELISATION ET VERIFICATION DE POLITIQUES DE SECURITE

Amal HADDAD

Master2 Recherche Systèmes d'Information
Université Joseph Fourier
Formation Européenne de 3ème Cycle en Systèmes d'Information: MATIS
Université de Genève

Directeurs:
Mme Marie-Laure POTET
M. Nicolas STOULS

Stage effectué au laboratoire LSR, équipe VASCO
du 8 novembre 2004 au 7 Septembre 2005

Remerciements

Je tiens à remercier Mme Marie-Laure POTET pour ses conseils judicieux, son soutien, sa patience et l'aide qu'elle a prodigué pour ce travail.

Je suis profondément reconnaissante à Nicolas STOULS pour sa gentillesse, son soutien et ses conseils qui ont largement contribué aux avancements du projet.

J'adresse aussi mes remerciements à Monsieur Didier BERT, Chargé de recherches au CNRS dans l'équipe VASCO, pour l'aide qu'il m'a apporté.

Mes remerciements vont aussi au personnel sympathique du LSR. Je note particulièrement Mme Pascale POULET, Mme Christiane PLUMERE et Monsieur Gilles THIEBLEMONT.

Je remercie vivement Mme Marie-Christine FAUVET et Monsieur Michel LEONARD, les responsables de la formation Systèmes d'information que j'ai suivi.

Enfin, je ne peux oublier toute ma famille, maman, papa, mes sœurs (Rabab et Farah) et mes frères (Omar et Ibrahim) pour leur appui continu et inconditionnel. Vous êtes toujours présents dans mon coeur malgré la grande distance géographique qui nous sépare.

Je remercie également tous mes amis en France et ailleurs pour leur aimable présence et tous les bons moments que nous avons partagés ensemble.

Amal

Table des matières

1 Introduction

La sécurité des informations est le processus de protection des données contre toute mauvaise utilisation, accidentelle ou intentionnelle, par des personnes à l'intérieur ou à l'extérieur d'une organisation, dont les employés, les consultants et, les pirates. La sécurité se caractérise par la trilogie traditionnelle de propriétés:

- La confidentialité définie par la prévention d'une divulgation non autorisée de l'information.
- L'intégrité ou la prévention d'une modification non autorisée de l'information.
- La disponibilité ou l'accessibilité des ressources pour les utilisateurs légitimes.

Les problèmes de sécurité au sein des grandes sociétés causent des dommages se chiffrant en millions d'euros. Les virus et vers comme Code Red entraînent régulièrement d'importantes périodes de mise à l'arrêt des systèmes ainsi que des pertes commerciales et l'endommagement de données et d'ordinateurs. Carlsbad, une société de recherche en économie informatique basée en Californie, estime que les codes malveillants ont causé plus de 12 millions d'euros de dommages uniquement en 2001 **[Hp]**. Ajoutons à cela les pertes indirectes liées à la confiance des clients et des actionnaires envers le bon fonctionnement de l'entreprise attaquée. Nous donnons par la suite quelques exemples réels de dommages **[Hp]**.

Au début de l'année 2002, Associated Press rapporta qu'une banque de New York a versé à un pirate russe la somme de 10 000 euros pour qu'il ne révèle pas des informations clients sensibles ; les dommages causés se chiffraient quant à eux à 250 000 euros. Dans un autre cas rapporté par le New York Times, un cadre informatique mécontent a généré jusqu'à 20 millions d'euros de dommages lorsqu'il a saboté les systèmes informatiques d'une société de produits chimiques du New Jersey qui l'avait licencié. A la suite d'une attaque de refus de service ayant causé des dommages spécialement importants, un fournisseur d'accès du Royaume-Uni a, paraît-il, arrêté ses opérations et mis ses actifs en vente. Et, d'après certaines sources, l'éditeur de logiciels Egghead ne s'est jamais vraiment remis d'un incident qui a fait beaucoup de bruit et au cours duquel il annonça que plus de trois millions de numéros de carte de crédit avaient peut-être été volés par un pirate. D'autres statistiques effectuées par Yankee Group proclament que eBay, Yahoo et Amazon ont perdu 1.2 Milliard $ pour des attaques contre la disponibilité de leurs sites en 2000. Le FBI annonce que 85% des compagnies américaines et canadiennes ont trouvé un virus dans leur système et que 80% de pourcentage des attaques correspondent aux intrusions internes **[Cup02]**.

Ces exemples réels de dommages mettent en exergue l'importance de la protection des ressources sensibles. Pour assurer la protection, un système de contrôle d'accès doit intercepter toutes les tentatives d'accès aux ressources et informations critiques.
L'élaboration d'un tel système s'effectue par une approche multi phase traitant différents niveaux de conception. Elle est basée sur les concepts suivants:

- Les politiques de sécurité : elles définissent les règles de haut niveau qui régissent les accès et décident lesquels sont autorisés.
- Les modèles de sécurité : ces modèles dressent une représentation formelle des politiques de sécurité et de leur fonctionnement. Ils permettent de prouver des propriétés sur la sécurité du système.

- Les mécanismes de sécurité : ceux-ci définissent les fonctions bas niveau (logiciels et matériel) permettant d'implémenter les contrôles imposés par la politique de sécurité.

Par ailleurs, pour avoir confiance en la sécurité du système informatique, les utilisateurs ont besoin d'éléments de comparaison leur permettant de choisir les systèmes qui répondent le mieux à leurs besoins. C'est pour cette raison que des critères d'évaluation ont été développés dans plusieurs pays comme les ITSEC ou les critères communs **[CC05]** (un ensemble de normes de sécurité internationalement approuvé). Pour les niveaux d'assurance d'évaluation élevés (EAL 5, 6 et 7), ces critères imposent l'utilisation de méthodes formelles dans la spécification des systèmes ainsi que la démonstration par des preuves que la spécification formelle assure les propriétés de sécurité.
Ces contraintes ont favorisé les recherches dans le cadre de la formalisation du concept de sécurité. En fait, le domaine de sécurité est un monde foisonnant, les recherches ont abouti à des résultats importants, plusieurs modèles de sécurité ont été développés, se rapportant à divers politiques de sécurité (discrétionnaires, obligatoires et basées sur les rôles) nous citons par exemple le modèle HRU **[HRU76]**, le modèle Bell et Lapadula **[BL73a, BL73b, BL74a, BL74b, BL75]**, le modèle Biba **[BIB77]**, le modèle RBAC **[SAN96, AS00, FKS$^+$01]**. Les vérifications associées à ces politiques concernent surtout leur cohérence.

Notre travail s'inscrit dans ce cadre puisque nous nous intéressons à la vérification " du respect " d'une politique par une application. Une application est composée d'un ensemble de données sensibles, les variables manipulées par des opérations. Il est important de s'assurer que la valeur d'une variable est lu et modifié en toute sécurité selon la réglementation spécifiée.

Pour ce faire, nous essayons de prendre en compte la démarche complète de l'élaboration du système de contrôle d'accès, en commençant par la phase formalisation d'une politique de sécurité jusqu'à aboutir à sa mise en œuvre par l'outil réalisé et sa vérification. Nous proposons de raffiner l'application initiale en transformant tout accès direct en lecture ou en écriture en un appel à des services sécurisés. Ces services sont conçus dans un noyau de sécurité respectant la politique attendue. Nous nous basons sur la technique de vérification statique qui garantit une découverte des failles de sécurité à priori et avant l'exécution. Une originalité de notre travail réside dans le fait de concevoir un outil générique capable de traiter plusieurs modèles de sécurité : discrétionnaires, obligatoires et basés rôles.

Afin de réaliser notre objectif, nous utilisons la méthode formelle B qui est une méthode de spécification, de conception et d'implémentation des systèmes logiciels. Sa caractéristique fondamentale est qu'elle permet de développer des logiciels corrects par construction. Ainsi, elle s'avère comme étant une approche adaptée à la maîtrise des développements de logiciels critiques de sécurité. Elle a été appliquée avec succès dans de grands projets industriels. Citons par exemple le développement des automatismes sécuritaires du métro automatique METEOR, grâce à un ensemble d'outils logiciels d'assistance : l'Atelier B.

Ajoutons à cela que l'Atelier B génère des formules (appelées obligations de preuve) qui permettent de vérifier des propriétés. Dans notre cas, ces formules permettent de garantir que les appels des services de lecture et d'écriture s'effectuent par les opérations habilitées. Lorsque des failles de sécurité sont présentes, les obligations de preuve ne peuvent pas être démontrées et leur examen permet de déceler les erreurs.

Ce mémoire est organisé en trois parties :

- Un chapitre sur le processus de contrôle d'accès. Dans ce chapitre nous présentons les politiques de sécurité et les modèles que nous allons traiter dans notre outil. Nous évoquons aussi des mécanismes d'implémentation de ces politiques et des approches pour vérifier le respect de ces politiques.
- Un deuxième chapitre fait le point sur la méthode B, ses principes de base, les outils qui lui sont associés ainsi que son apport dans notre travail.
- Après avoir présenté l'état de l'art à travers les deux premiers chapitres, nous abordons l'approche et l'outil réalisé, son fonctionnement ainsi que son utilisation à travers une étude de cas.

Nous terminons par une conclusion générale du travail et des perspectives que nous comptons réalisées.

2 Le contrôle d'accès

2.1 Introduction

La sécurité d'un système informatique a pour but la protection des ressources (incluant les données et les programmes) contre la révélation, la modification ou la destruction accidentelle ou malintentionnée, tout en garantissant l'accès pour les utilisateurs légitimes. Pour assurer la sécurité, plusieurs techniques sont employées dont le contrôle d'accès, le chiffrement et la cryptographie. Dans le cadre de notre travail, nous nous intéressons au contrôle d'accès. C'est un processus permettant de contrôler les tentatives d'accès afin de garantir les propriétés de sécurité suivantes : la confidentialité, l'intégrité et la disponibilité de service. Brièvement, la confidentialité permet de protéger les ressources contre les révélations non autorisées, l'intégrité permet de prohiber les modifications non appropriées des données et la disponibilité de service permet de garantir l'accès aux informations et services.

2.2 Le contrôle d'accès, les sujets et les objets

Les techniques de contrôle d'accès sont utilisées dans des systèmes dont on veut protéger les ressources. Plusieurs entités entrent alors en jeu :

Les objets sont des entités passives qui contiennent des informations. Ils représentent les éléments du système qui doivent être protégés. Par exemple, dans le domaine de la protection des systèmes d'exploitation, il s'agit des fichiers, des répertoires ou des programmes. Dans le domaine des systèmes de base de données, les objets peuvent désigner des procédures, des relations ou des vues. Il peut s'agir aussi des variables d'une application referençant des données confidentielles.

Les entités actives qui manipulent les objets sont appelés sujets. Ces derniers possédent des autorisations (droits d'accès) sur les objets et demandent d'y accéder. Un sujet peut être considéré en tant qu'objet puisqu'il est suceptible d'être manipulé par un autre sujet. Par exemple, les procédures sont des sujets qui manipulent des objets qui sont les variables. Mais elles sont aussi des objets auxquels accèdent d'autres sujets comme les utilisateurs. D'où la relation : $S \subseteq O$, où S représente l'ensemble des sujets et O l'ensemble des objets.

Par la suite, il conviendra de présenter les phases d'élaboration d'un système de contrôle d'accès.

2.3 Les phases d'élaboration d'un système de contrôle d'accès

L'élaboration d'un système de contrôle d'accès s'effectue par une approche multi phase (ou à phase multiple) basée sur les concepts suivants:

- Les politiques de sécurité : elles définissent les règles de haut niveau qui régissent les accès et décident lesquels sont autorisés pour garantir un système sûr. Autrement dit, la politique de sécurité d'un système est l'ensemble des lois, règles et pratiques qui régissent la façon dont l'information sensible et les autres ressources sont gérées, protégées et distribuées à l'intérieur d'un système spécifique **[ITSEC91]** . Nous nous intéressons dans le cadre de notre étude aux politiques de sécurité logiques qui sont réalisées par le système informatique lui-même et particulièrement aux politiques

d'autorisation qui déterminent les droits que les sujets ont sur les objets et comment ces droits peuvent être modifiés.

- Les modèles de sécurité : ces modèles décrivent une représentation formelle des politiques de sécurité et de leur fonctionnement. Ils permettent de faciliter les preuves sur la sécurité d'un programme, c'est la raison pourlaquelle les efforts se sont focalisés autour de la construction des modèles formels pour la sécurité **[Mcl90]**.

- Les mécanismes de sécurité : ceux-ci définissent les fonctions bas niveau (logiciels et matériels) permettant d'implémenter les contrôles imposés par la politique de sécurité.

Ces trois concepts se rapportent à différents niveaux d'abstraction liés au développement d'un système de contrôle d'accès, ce qui offre plusieurs avantages évoqués dans **[SD01]**. D'abord, ceci permet d'aborder les exigences de sécurité indépendamment de leur implémentation et rend possible la comparaison entre différentes politiques de sécurité. Enfin, cela permet de mettre en œuvre des mécanismes implémentant des politiques de sécurité multiples. Ce dernier avantage apporte plus de flexibilité au système de contrôle d'accès. En effet, le mécanisme n'étant pas lié à une politique de sécurité spécifique, le changement de cette dernière n'impose pas le changement de tout le système de contrôle d'accès.

Nous allons examiner maintenant les politiques et modèles de sécurité représentés dans la littérature.

2.4 Les politiques et modèles de sécurité

Une politique de sécurité doit prendre en considération les règlements qui doivent être appliqués ainsi que les menaces éventuelles dues à l'utilisation du système informatique.

Les politiques du contrôle d'accès peuvent être groupées en trois principales classes :

- Les politiques discrétionnaires (DAC) : les politiques discrétionnaires accordent au propriétaire de l'information, généralement le créateur, tous les droits d'accès ainsi que la possibilité de les propager aux autres selon sa discrétion.
- Les politiques obligatoires (MAC) : ces politiques décrètent des règles incontournables qui régissent les droits des sujets et des objets. Elles permettent de restreindre les privilèges que possèdent les sujets sur les objets qui leur appartiennent.
- Les politiques basées sur les rôles (RBAC) : ces politiques sont les plus récentes, elle servent à décrire d'une manière plus expressive et plus puissante les fonctionnalités dans les organisations. Les droits d'accès sont accordés aux rôles, tâches dans l'organisation, ils sont attribués aux sujets en fonction des rôles qu'ils jouent.

Nous allons exposer les trois grandes familles des politiques de sécurité ainsi que les modèles qui les représentent.

2.4.1 Politiques et modèles d'autorisation discrétionnaires (DAC)

Une politique est dite discrétionnaire si les sujets qui possèdent les objets disposent du droit de passer librement (à leur discrétion) les autorisations qu'ils détiennent sur ces derniers. L'accord ou la révocation des privilèges est régulé par une politique

administrative. La gestion des accès aux fichiers du système d'exploitation UNIX constitue un exemple classique de mécanisme de contrôle d'accès basé sur une politique discrétionnaire. Nous exposons par la suite le modèle discrétionnaire le plus connu, il s'agit du modèle de la matrice d'accès.

2.4.1.1 Aperçu Historique

Historiquement, ce modèle est proposé par Lampson **[LAM71].** Il a été ensuite redéfini par Graham et Dening **[GD72]** qui ont précisé des règles pour créer, supprimer des objets, transférer et donner des permissions d'accès afin de mettre à jour la matrice d'accès. Ensuite, ce modèle fut formalisé par Harrison, Ruzzo, et Ullmann, le modèle HRU **[HRU76]**, où les auteurs ont abordé le problème de la protection d'un système.

2.4.1.2 Le modèle Lampson

Dans le cadre de ce modèle, l'état courant d'un système est défini par une matrice d'accès et un triplet (S, O, M) représentant les autorisations. Le terme S désigne l'ensemble des sujets, O l'ensemble des objets et M la matrice d'accès.

Chaque cellule M [s,o] de la matrice contient les droits d'accès que possède le sujet s sur l'objet o. Chaque sujet occupe une ligne de la matrice tandis qu'une colonne est assignée à un sujet ou un objet. Les droits d'accès sont lire (read), écrire (write), exécuter (execute), contrôler (control) et posséder (own).Une bannière * (flag) contrôle le transfert des droits d'accès effectué entre les sujets, elle signifie que l'autorisation concernée peut être transférée à d'autres sujets. Un exemple de matrice d'accès est donné dans la figure 2.1.

La matrice n'étant pas figée celle-ci peut être mise à jour par la création de nouveaux objets ou sujets, par la destruction de ces derniers ainsi que par l'ajout ou la suppression des droits d'accès.

	Fichier 1	Fichier 2	Sujet 3	Processus1
Ann Bob	*own read write read	*read write *contol		execute
Sujet3		read	own control	execute read

Figure 2.1 : la matrice d'accès du modèle Lampson

2.4.1.3 Le modèle Graham et Dening

Graham et Dening ont précisé huit commandes qui permettent de mettre à jour la matrice d'accès du modèle Lampson. Ces commandes traitent la création et la destruction des objets par les sujets ainsi que le transfert des autorisations entre les sujets. Ces commandes

ainsi que la matrice d'accès forment la pierre angulaire du système de protection établi en définissant les évolutions futures de l'état de la matrice **[LAN81]**. Nous ne détaillons pas ces commandes dans notre mémoire, le lecteur intéressé peut se référer à **[GD72]**.

2.4.1.4 Le modèle HRU

Dans ce modèle, les auteurs se sont intéressés aux propriétés vérifiées par un système de contrôle d'accès lorsque son état change. Ce changement s'effectue par l'intermédiaire des commandes exécutant des opérations primitives sur les autorisations sous des conditions spécifiées.

La figure 2.2 **[SD01]** offre une présentation ces opérations primitives avec leurs conditions d'application ainsi que le nouvel état du système résultant après l'exécution de ces opérations :

OPERATION (op)	CONDITIONS	NEW STATE $(Q \vdash_{op} Q')$
enter r into $A[s,o]$	$s \in S$ $o \in O$	$S' = S$ $O' = O$ $A'[s,o] = A[s,o] \cup \{r\}$ $A'[s_i,o_j] = A[s_i,o_j] \quad \forall (s_i,o_j) \neq (s,o)$
delete r from $A[s,o]$	$s \in S$ $o \in O$	$S' = S$ $O' = O$ $A'[s,o] = A[s,o] \setminus \{r\}$ $A'[s_i,o_j] = A[s_i,o_j] \quad \forall (s_i,o_j) \neq (s,o)$
create subject s'	$s' \notin S$	$S' = S \cup \{s'\}$ $O' = O \cup \{s'\}$ $A'[s,o] = A[s,o] \quad \forall s \in S, o \in O$ $A'[s',o] = \emptyset \quad \forall o \in O'$ $A'[s,s'] = \emptyset \quad \forall s \in S'$
create object o'	$o' \notin O$	$S' = S$ $O' = O \cup \{o'\}$ $A'[s,o] = A[s,o] \quad \forall s \in S, o \in O$ $A'[s,o'] = \emptyset \quad \forall s \in S'$
destroy subject s'	$s' \in S$	$S' = S \setminus \{s'\}$ $O' = O \setminus \{s'\}$ $A'[s,o] = A[s,o] \quad \forall s \in S', o \in O'$
destroy object o'	$o' \in O$ $o' \notin S$	$S' = S$ $O' = O \setminus \{o'\}$ $A'[s,o] = A[s,o] \quad \forall s \in S', o \in O'$

Figure 2.2 : les opérations primitives du modèle HRU

Une commande HRU possède une partie conditionnelle facultative ainsi qu'un corps, elle a le format suivant :

command c(x_1,…,x_k)

if a_1 in M [s_1,o_1]

a_2 in M [s_2,o_2]

.

.

a_n in M [s_m,o_m]

then op_1

op_2

.

.

op_n

end

Avec n>0, m≥0, a_1, a_m sont des autorisations, op_1… op_n sont des opérations primitives. Une commande HRU peut ne pas avoir de condition. Le cas échéant m vaut 0.

Nous allons donner un exemple d'une commande HRU dans laquelle le possesseur d'un fichier accorde l'autorisation read à un friend :

command $CONFER_{read}$(owner,friend,file)

if own in M [owner,file]

then enter read into M [friend,file]

end

La flexibilité dans la définition des commandes fait apparaître un problème se rapportant à la protection dans un système de contrôle d'accès ou « safety problem ». Etant donné un système avec une configuration initiale Q, le problème de protection est concerné par le fait de déterminer si un sujet « s » peut obtenir une autorisation « a » sur un objet « o » suite à l'exécution d'une séquence de commandes. Les auteurs montrent **[HRU76]** que le problème de protection est indécidable. Cependant, il devient décidable dans le cas des systèmes mono opérationnel, où le corps de la commande possède une opération et le nombre des sujets et des objets est fini. Cependant, les systèmes mono opérationnels ont été critiqués dans **[SAN92]** au motif qu'ils réduisent l'utilité et l'efficacité des commandes. Par exemple, une commande create a pour effet d'ajouter une ligne ou une colonne vide dans la matrice d'accès sans pour autant entrer un attribut « owner » ou « control ». En conséquence, il sera impossible de supporter les autorisations « owner » ou « control » entre les sujets.

2.4.1.5 Les vulnérabilités des politiques discrétionnaires

Dans le cadre de ces politiques, les demandes d'accès aux objets effectués par les utilisateurs ont été vérifiées selon les autorisations de ces derniers. Il est incontestable que toute demande d'accès est due aux actions des utilisateurs. Cependant une analyse plus précise du problème de contrôle d'accès fait surgir l'utilité d'établir la distinction entre sujets et utilisateurs. Le terme utilisateurs désigne des entités passives possédant des autorisations et qui se connectent au système. Un utilisateur connecté au système génère un sujet ou un processus qui effectue à son compte les demandes au système.

Les politiques discrétionnaires ignorent la distinction entre utilisateurs et sujets. Ajoutons à cela le fait qu'elles ne contrôlent pas les flots d'information acquis par un sujet, ce qui peut causer une fuite d'informations pour des sujets non autorisés. Précisément, les politiques discrétionnaires sont vulnérables vis-à-vis des chevaux de Troie qui sont des processus exécutant des fonctions illicites cachées dans des fonctions légitimes.

Pour expliquer le fonctionnement d'un cheval de Troie, nous allons donner un exemple tiré de **[SD01]** illustré par la figure 2.3 :

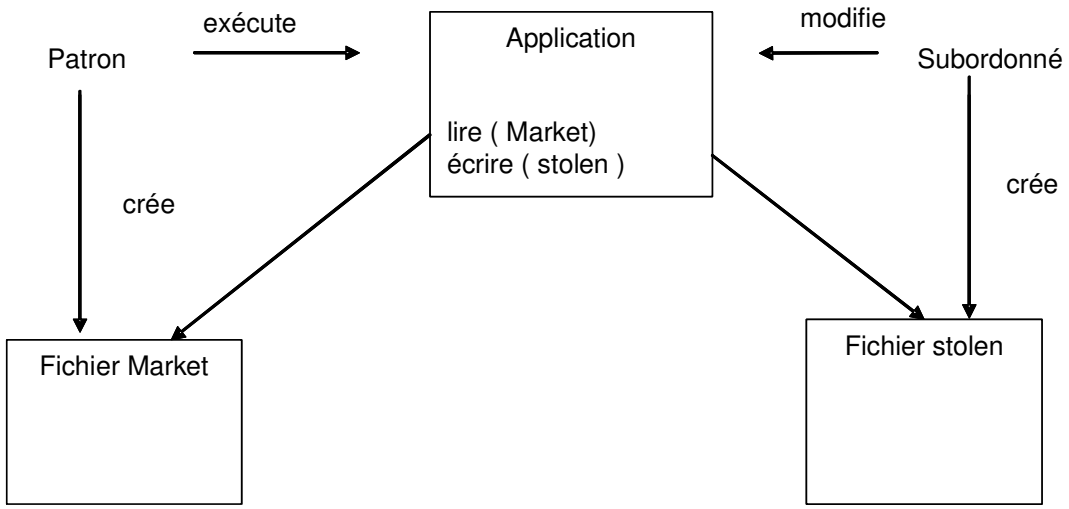

Figure 2.3 : un exemple de cheval de Troie

Supposons par exemple, que Vicky une manager de l'entreprise, crée un fichier secret 'Market' contenant des informations importantes sur le lancement de nouveaux produits sur le marché. Selon les règlements de l'entreprise, ces informations délicates ne doivent pas être divilguées à une personne autre que Vicky.

Considérons maintenant John, un des subordonnés de Vicky, qui veut s'emparer de ces informations pour les vendre à une entreprise concurrente. Pour ce but, John crée un fichier 'Stolen' et accorde à Vicky le droit d'écriture dans ce fichier, celle-ci n'est même pas censée connaître l'existence de ce fichier. John modifie aussi une application généralement utilisée par Vicky pour y inclure deux opérations cachées, une opération lire sur le fichier Market et écrire dans Stolen, il donne ensuite cette nouvelle application à sa manager Vicky. Lorsque Vicky exécute cette application, les droits d'accès sont vérifiés et en particulier les deux opérations lire et écrire sont autorisées. Par conséquence, durant l'exécution, les informations délicates seront transmises du fichier Market vers le fichier Stolen que John possède.

Nous pouvons bien nous rendre compte de l'importance d'établir la distinction entre sujets et utilisateurs : tandis que Vicky est chargée de préserver la confidentialité des informations qu'elle connaît, le processus exécuté à son compte ne peut pas avoir le même degré de confiance.

2.4.1.6 Conclusion

Pour pallier ce problème de fuite d'informations, des restrictions doivent être imposées sur les opérations exécutées par les processus. D'une manière plus précise, un processus de

contrôle de flots est requis afin de défaire les chevaux de Troie et les fuites d'informations qu'ils causent.

Les politiques obligatoires ont pour objectif d'empêcher les fuites en contrôlant les flots d'information. Nous allons les présenter ci-dessous.

2.4.2 Les politiques obligatoires (MAC)

Comme énoncé précédemment les politiques obligatoires décrètent des règles pour contrôler les flots d'informations, afin de contrer le transfert illégal et les fuites. Ces politiques sont généralement des politiques multi-niveaux basées sur une classification des sujets et des objets imposée par une autorité centrale.

2.4.2.1 Les politiques multi-niveaux

Les politiques multi-niveaux effectuent un classement des sujets et des objets. Elles introduisent la notion de classe d'accès. Une classe d'accès est affectée à chaque sujet et objet. Une relation d'ordre partiel est définie sur l'ensemble des classes d'accès, il s'agit de la relation de dominance symbolisée par $\geq$.

Une classe d'accès est formée de deux composants :

- Un niveau de sécurité
- Un ensemble de catégories

Le niveau de sécurité est un élément d'un ensemble totalement ordonné, par exemple top secret (TS), secret (S), confidentiel (C) et non classifié (N) où $TS \geq S \geq C \geq N$. L'ensemble des catégories décrit les divers domaines des systèmes en étude. Par exemple dans les systèmes militaires cet ensemble peut contenir les catégories : nucléaire et armée, dans les systèmes commerciaux les catégories sont plutôt financier, administratif, mercatique …

Etant données deux classes d'accès ac1 et ac2, la relation de dominance $\geq$ se définit ainsi :

ac_1 domine ac_2 ou $ac_1 \geq ac_2$ si :

1. le niveau de sécurité de ac_1 est plus grand ou égal à celui de ac_2
2. les catégories de ac_1 incluent celles de ac_2

Formellement cette relation de dominance est définie de la façon suivante :

Soit L, l'ensemble des niveaux de sécurité, muni de la relation d'ordre partiel $\geq$ et C, un ensemble de catégories, muni de la relation d'ordre partiel $\supseteq$. Soit l_1, l_2 deux niveaux et c_1, c_2 deux catégories tels que $l_1 \in L$, $l_2 \in L$, $c_1 \in C$ et $c_2 \in C$.

$\forall\ ac_1 = (l_1,c_1), ac_2 = (l_2,c_2) : ac_1 \geq ac_2 \Leftrightarrow l_1 \geq l_2 \wedge c_1 \supseteq c_2$.

Deux classes d'accès ac_1 et ac_2 sont dites incomparables si on ne peut ni vérifier $ac_1 \geq ac_2$, ni $ac_2 \geq ac_1$.

L'ensemble de classes d'accès muni de la relation d'ordre partiel ($\geq$) possède une borne inférieure et une borne supérieure. La structure de cet ensemble forme un treillis **[DEN76]**. La figure 2.4 illustre un exemple de treillis de sécurité pour le système militaire **[SD01]**.

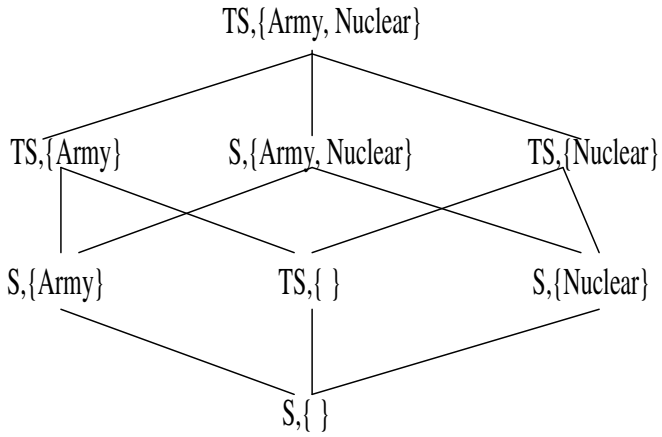

Figure 2.4 : un exemple d'un treillis de sécurité

L'utilisation ainsi que la sémantique de la classification dépendent de l'objectif visé : préserver la confidentialité ou l'intégrité des données. Par la suite nous présentons les politiques multi-niveaux basées confidentialité ainsi que celles basées intégrité et les modèles basés dessus.

2.4.2.2 Les politiques basées confidentialité et le modèle Bell-Lapadula

Ces politiques visent à préserver la confidentialité des données c'est-à-dire que celles-ci ne sont accessibles que pour les utilisateurs autorisés.

Pour maintenir la confidentialité, des restrictions sont à imposer sur les opérations de lecture et d'écriture, qui permettent de transférer des informations. Ces restrictrions se traduisent par les deux principes suivants :

- No read up : Un sujet ne peut lire un objet que si la classe d'accès du sujet domine celle de l'objet (un sujet ne doit pas apprendre des informations qui ne lui sont pas autorisées).
- No write down : Un sujet ne peut écrire dans un objet que si la classe d'accès de l'objet domine celle du sujet (un sujet ne doit pas révéler des secrets).

Dans le cadre de ces politiques, le niveau de sécurité ou classification associé à un objet reflète l'importance des informations qu'il contient ainsi que le danger causé par la divulgation de celles-ci. Le niveau de sécurité associé à un sujet, appelé aussi autorisation, est un moyen de représenter la confiance qui lui est accordée.

Les utilisateurs se connectent au système à une classe d'accès quelconque dominée par leurs autorisations et génèrent des sujets appartenant à cette classe d'accès. Par exemple (cf. **Figure 2.4**), un utilisateur possédant l'autorisation (TS,{Nuclear}) peut se connecter au système en tant que (TS,{Nuclear}), (S,{Nuclear}), (TS,∅) ou (S,∅). Les demandes d'accès d'un sujet à un objet sont contrôlées selon les relations de dominance entre les classes d'accès correspondantes.

La satisfaction des deux principes (No read up, No write down) empêche les flots d'informations de se propager d'un niveau de sécurité haut vers d'autres niveaux de sécurité plus bas.

Montrons comment la tâche du cheval de Troie du paragraphe 2.4.1.5 sera bloquée. Considérons les niveaux de sécurité suivants : Secret pour Vicky et son fichier Market et non classifié pour John et son fichier Stolen. Si Vicky se connecte au système au niveau Secret ou Confidentiel, le sujet ou l'application va être exécutée à un niveau Secret ou Confidentiel et l'opération d'écriture dans Stolen sera bloquée selon le prinipe No write down. Si Vicky se connecte au système en tant que sujet non classifié et exécute l'application à ce niveau, l'opération de lecture dans le fichier Market sera bloquée selon le principe No read up.

2.4.2.2.1 Le modèle Bell-Lapadula

Nous présentons maintenant le modèle Bell-Lapadula qui décrit formellement les contrôles des accès pour préserver la confidentialité.

Ce modèle a été proposé pour la première fois par David Bell et Leonard Lapadula en 1973 **[BL73a]**. Par la suite, différentes variantes ont été élaborées pour la prise en compte de nouvelles propriétés **[BL73b, BL74a, BL74b]** et aussi pour adapter ce modèle à d'autres environnements spécifiques **[BL75]**. Le modèle Bell-Lapadula a été développé dans la société Mitre, dans le cadre de la réalisation d'un système informatique de sécurité pour les forces aériennes américaines du département de défense. L'objectif est d'assurer la confidentialité des données militaires.

A travers ce modèle, les auteurs tracent une définition formelle des concepts d'une sécurité militaire applicable dans les systèmes informatiques. Le modèle est fondé sur la notion de treillis, il utilise aussi une matrice d'accès identique à celle du modèle HRU pour présenter les autorisations auxquelles s'ajoutent les niveaux de sécurité d'un système militaire.

Selon Bell et Lapadula, le système est composé d'un ensemble de sujets S, d'un ensemble d'objets O, d'un ensemble d'autorisations A (lire et écrire) et d'un ensemble de classification I (des classes d'accès). A un moment donné, l'état d'un système est exprimé par un ensemble formé de trois composants qui sont :

- L'ensemble des accès courants d'un sujet sur un objet noté b. Un accès courant est représenté par un triplet (s, o, a) signifiant que le sujet « s » possède l'accès courant « a » sur l'objet « o » dans un état du système.
- La matrice d'accès M (la même que celle du modèle HRU) formée de i lignes et j colones. Chaque cellule M_{ij} enregistre les modes d'accès que possède le sujet s_i sur l'objet o_j. Ainsi, les entrées de M sont des sous-ensembles de l'ensemble des accès possibles A.
- La fonction $\lambda : S \cup O \rightarrow I$ qui appliquée aux sujets et aux objets retournent la classification de ces derniers.

L'état du système est ainsi représenté par un triplet (b, M, λ).

La description formelle des deux principes No read up et No write down se traduit par les axiomes suivants :

- Propriété simple : Un sujet s ne peut lire un objet o que si son autorisation domine la classification de l'objet : $\forall s \in S, o \in O : (s, o, \text{lire}) \in b \Rightarrow \lambda(s) \geq \lambda(o)$.

- Propriété étoile : Un sujet s ne peut écrire (écrire est utilisée pour signifier écriture seule ou ajout) dans un objet o que si la classification de l'objet domine l'autorisation du sujet : $\forall s \in S, o \in O : (s, o, \text{écrire}) \in b \Rightarrow \lambda(o) \geq \lambda(s)$.

Les auteurs précisent cependant que la propriété étoile ne s'applique pas aux sujets de confiance « trusted subjects », ceux ci étant des sujets qui, pour des raisons de bon fonctionnement de l'organisation, sont autorisés à violer les restrictions imposées.

Une autre propriété doit être vérifiée, c'est la propriété discrétionnaire dénontée ds-property qui stipulent que tous les accès courants des sujets aux objets sont des éléments de la matrice d'accès Mij : $\forall (s_i, o_j, a) \in b \Rightarrow a \in Mij$ (avec $a \in A$).

Les auteurs formalisent un théorème qui annonce que la sécurité d'un système peut être garantie systématiquement si pour tout ajout d'un triplet (sujet, objet, accès) dans l'ensemble b, les trois propriétés évoquées ci dessus sont vérifiées. Ce théorème est appelé le théorème basique de sécurité (BST : Basic Security Theorem).

Un état du système symbolisé par $v \in V$, où V est l'ensemble des états, est défini par un triplet (b, M, λ). L'état initial du système est noté v_0. L'état du système change suite à l'ensemble des demandes. Celles ci peuvent référer à des acquisitions ou des libérations des accès, des ajouts ou des supressions des autorisations ainsi qu'aux changements de classification des sujets et des objets. La fonction de transition T fait parvenir le système d'un état vers un autre, elle est définie de la manière suivante $T : V \times R \rightarrow V$ où R représente l'ensemble des demandes effectuées par les sujets.

Les auteurs proposent aussi le principe de tranquilité (tranquility principle) qui stipule que la classification des objets ne doit pas changer.

2.4.2.2.2 Critiques du modèle Bell-Lapadula

Bien que les utilisateurs puissent se connecter au système à n'importe quel niveau dominé par leur autorisation, l'application des deux principes (No read up et No write down) est très restrictive. Le sujet ne pouvant écrire des informations que dans un objet de niveau plus haut, ceci aboutit à une surclassification des informations qui doivent être déclassifiées après un certain temps par des sujets de confiance. Par exemple, si une information non classifiée est lûe par un sujet de niveau secret, tout objet modifié par ce sujet avec cette information est classifié à un niveau supérieur ou égal à secret.

Le théorème basique de sécurité proposée par Bell et Lapadula a été critiqué par John Mclean **[MCL90]**, celui-ci a démontré que ce théorème ne garantit pas la sécurité du système puisqu'il n'impose aucune restriction sur les transitions. Mclean considère un système Z (à l'origine présenté en 1986 dans Computer Security Forum, issue 14, june22, 1986) vérifiant le BST théorème mais qui n'est logiquement pas sûr. Le système Z est un système qui a un état initial sûr et possède une transition matérialisée de la façon suivante : dès lors qu'un sujet de niveau minimal demande l'accès à un objet quelconque, il met les niveaux de tous les objets et sujets au niveau minimal et l'accès est autorisé. Ceci peut être possible si les niveaux des sujets et des objets sont sauvegardés dans un fichier. System Z satisfait le théorème BST mais n'est certainement pas sûr. Mclean préconise de contrôler les transitions, il propose une extention du modèle avec une fonction C définie de la façon suivante $C : S \cup O \rightarrow S'$ ($S' \subset S$). Cette fonction retourne l'ensemble des sujets ayant le droit de changer les niveaux d'autres sujets et objets, ces sujets sont des sujets de confiance (trusted subject) capables de déclassifier les autres objets et sujets. (la notion de sujets de confiance (trusted subject) a été développé par Bell et Lapadula mais leur modèle ne donne pas de précisions sur les sujets qui peuvent être considérés comme tels **[LAN81]**).

Selon Mclean une transition est sûre si le changement de niveaux des sujets et objets ne peut être effectué que par les sujets appartenant à S' (les sujets de confiance).

Un système (v_0, R, T) est sûr si :

- v_0 est sûr
- chaque nouvel état obtenu à partir de v_0 après l'exécution d'une séquence finie de demandes est sûr
- T est une transition sûre

Une autre critique du modèle Bell et Lapadula est qu'il ne propose pas des axiomes permettant de gérer les objets multi-niveaux. Bien que la hiérarchisation des objets aie été introduite dans **[BL74a],** la représentation qui en découle demeure restrictive **[LAN81]**.

D'autres politiques multi-niveaux ont été proposées afin de préserver l'intégrité des données, nous les représentons dans la partie suivante.

2.4.2.3 Les politiques basées intégrité et le modèle Biba

Maintenir l'intégrité des données est nécessaire afin de garantir qu'aucun sujet de niveau inférieur ne puisse modifier indirectement des objets appartenant à un niveau supérieur. Ken Biba **[Bib77]** propose une politique duale de celle de Bell et Lapadula pour assurer l'intégrité.

Chaque sujet et objet possède une classification d'intégrité. Le niveau d'intégrité est un élément d'un ensemble totalement ordonné ; par exemple Crucial (C), Important (I) et Non classifié (N). Le niveau d'intégrité associé à un sujet reflète la confiance qui lui est accordée concernant la modification, l'ajout ou la supression des données. Pour un objet cette classification d'intégrité est un moyen de représenter le danger que peut constituer la modification de l'information contenue dans ce dernier.

Le contrôle d'accès est imposé par deux principes :

- No read down : un sujet peut lire un objet si la classe d'accès de l'objet domine celle du sujet.
- No write up : un sujet peut écrire dans un objet si la classe d'accès du sujet domine celle de l'objet.

La satisfaction de ces deux principes empêche la circulation des informations des sujets ou objets de niveau d'intégrité inférieur (moins fiable) vers des sujets ou objets de niveau d'intégrité plus élevé. Ces deux principes étant trop restrictifs, Biba propose un alternative plus flexible pour préserver l'intégrité :

- Low water mark pour les sujets : Ce principe vise à contraindre l'opération d'écriture selon le principe No write up. Un sujet peut lire des objets appartenant à un niveau inférieur au sien, mais par conséquence, son niveau d'intégrité diminue vers la borne inférieure de son niveau d'intégrité et de celui de l'objet : $\lambda'(s) = \inf(\lambda(s), \lambda(o))$.
- Low water mark pour les objets : Ce principe vise à contraindre l'opération de lecture selon le principe No read down. Un sujet peut écrire dans un objet de niveau inférieur, mais par conséquence, le niveau d'intégrité de l'objet modifié diminue vers la borne inférieure de son niveau d'intégrité et de celui de l'objet : $\lambda'(o) = \inf(\lambda(s), \lambda(o))$.

2.4.2.3.1 Critiques du modèle Biba

Le modèle Biba présente l'inconvénient dual du modèle Bell et Lapadula c'est celui de la sous-classification ou la dégradation des niveaux d'intégrité des sujets et des objets. Selon le principe low water mark pour les sujets, un sujet peut être empêché d'exécuter une procédure à cause des opérations de lecture qu'il a effectuées précédamment et qui ont abouti à une dégradation de son niveau d'intégrité.

Il faut alors remonter artificiellement les niveaux d'intégrité par des sujets de confiance (trusted subjects) autorisés à violer les restrictions imposées.

2.4.2.4 Les vulnérabilités des politiques obligatoires

Les politiques obligatoires peuvent être contrées (ou détournées) par les canaux cachés. Ces derniers sont des canaux non désignés pour la communication normale mais qui peuvent être exploités pour transférer des informations.

D'une manière générale, on distingue deux types de canaux cachés : les canaux de mémoire et les canaux temporels. Les canaux de mémoire sont des moyens de mémorisation partagés entre des utilisateurs d'habilitations différentes tels que la mémoire partagée, le partage des fichiers…. Les canaux temporels sont par contre difficiles à identifier et analyser, en effet le seule existence d'une ressource matérielle (unité centrale, mémoire…) ou logicielle (verrou, sémaphore…) partagée entre des utilisateurs d'habilitations différentes et d'une horloge commune, suffit pour créer un canal temporel par la modulation de l'usage de la ressource.

Pour illustrer comment les canaux cachés peuvent causer des fuites d'informations, nous considérons la demande effectuée par un sujet d'un niveau spécifique pour écrire dans un fichier non existant d'un niveau plus élevé. L'opération est légitime, selon le principe No Write down. Dès lors, deux comportements systèmes sont à traiter : si le système retourne une erreur, alors, il risque d'être exploité par des fuites malicieuses dûes aux processus haut niveau créant et détruisant des fichiers pour signaler des informations aux processus de niveau plus bas. Si le système n'informe pas le sujet de bas niveau de l'erreur, celui-ci ne saura pas les erreurs possibles dûes à des essais légitimes de tentatives d'écriture (exemple : les erreurs de frappe).

L'analyse des canaux cachés est souvent reportée à la phase d'implémentation. Plusieurs techniques sont utilisées, étudier les traces des flux d'informations dans les programmes **[DEN76]**, vérifier les ressources communes dans un programme susceptibles d'être utilisées pour le transfert d'informations, vérifier l'horloge du système pour les canaux temporels. Par ailleurs, les politiques de contrôle d'interfaces spécifient des restrictions sur les entrées et les sorties du système qui doivent être satisfaites pour se préserver contre les canaux cachés, nous ne les traitons pas dans le cadre de notre étude. Le concept de base pour ce type de politiques est le principe de non-interférence proposé par Goguen et Meseguer **[GM84]** qui stipule que les entrées d'un niveau haut ne doivent pas interférer avec les sorties d'un niveau plus bas.

Enfin, nous signalons que les politiques obligatoires et notamment les politiques multiniveaux, sont utilisées pour des systèmes non militaires. Nous citons par exemple, la sécurité dans les cartes multi-applicative **[GIR99]**, une nouvelle génération de cartes à puces avec des caractéristiques nouvelles : des applications peuvent être chargées après la délivrance de la carte et plusieurs applications différentes peuvent s'exécuter sur une même carte.

2.4.3 Les politiques et modèles basés rôles

Le modèle RBAC est principalement issu d'Internet afin de prendre en compte des applications déployées sur de vastes organisations ou des applications inter-organisations (Extranet par exemple) **[Had 95]**. Cependant, ce n'est que dans les années 1990 que RBAC fait son apparition en tant qu'un modèle assez mûr à l'instar des traditionnels MAC et DAC. Précisement, RBAC a été proposé pour la première fois en 1992 par David Ferrailo et Richard Kuhn **[FK92]**, attachés au département de commerce des Etats-Unis. Ils visaient l'élaboration d'un modèle mieux adapté que ses prédécesseurs dans les domaines industriels et commerciaux.

En effet, les politiques obligatoires et notamment les politiques multi-niveaux sont trop contraignantes pour les systèmes informatiques des organisations, et leur utilisation est généralement limitée au domaine militaire. Les politiques discrétionnaires, quant à elles, confient les autorisations sur les objets à la discrétion de leurs possesseurs. Cependant, dans la plupart des organisations, les utilisateurs finaux d'une information n'en sont pas généralement les propriétaires et le contrôle d'accès est plutôt relatif aux fonctions et aux responsabilités des employés (leurs rôles). D'où l'avènement de l'idée de base de RBAC : il s'agit d'intercaler des rôles entre les utilisateurs et les permissions. D'un côté des permissions sont accordées aux rôles, de l'autre, les utilisateurs se voient affecter un ou plusieurs rôles. Ils obtiennent ainsi les permissions accordées aux rôles qu'ils jouent **[AEB+03]**.

Au fil des années, différentes variantes du modèle RBAC ont été proposées afin d'introduire de nouveaux concepts, comme les sessions, les hiérarchies de rôles et les contraintes sur les rôles **[SAN96, AS00, FKS+01]**. Toutefois l'idée de base est toujours la même, le rôle est le concept central de la politique de sécurité.

La figure ci-dessous illustre le modèle RBAC tel qu'il est proposé par NIST **[FSG+01]**.

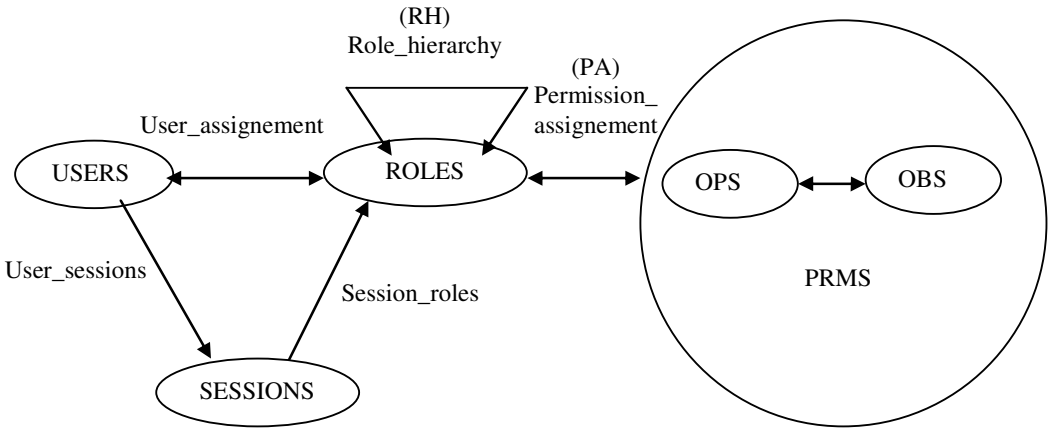

Figure 2.5 : Le modèle RBAC

Par la suite, il conviendra de détailler les concepts utilisés dans le modèle RBAC : Les utilisateurs (USERS) : Ce terme fait en général référence à un être humain mais il peut aussi désigner des machines, networks ou des agents intelligents. Les rôles (ROLES) : Un rôle est une fonction d'un emploi dans une organisation faisant allusion aux autorités et aux responsabilités accordés aux utilisateurs. Les opérations (OPS) : une opération représente l'image exécutable d'un programme. L'appel de cette dernière exécute des fonctions pour l'utilisateur. Par exemple, dans la protection d'un système de fichiers les opérations peuvent être lire, écrire, exécuter ; dans un système de gestion de bases de données, les opérations sont : insérer, supprimer, ajouter et mettre à jour. Les permissions symbolisées par PRMS sont matérialisées par des opérations effectuées sur les objets. PRMS $\subseteq$ OPS $\times$OBS. Comme le montre la figure, les utilisateurs sont assignés à des rôles via la relation UA / UA $\subseteq$ USERS $\times$ ROLES.

Par ailleurs, le côté dynamique du modèle est matérialisé par la notion de session. Une session est attribuée à un utilisateur qui, en l'activant, exerce un ou un ensemble de rôles qu'il assume. Un utilisateur peut ouvrir plusieurs sessions en même temps, cependant à un instant donné, un utilisateur exerce un seul rôle. C'est la notion de rôle actif. Le contrôle d'accès se déroule au cours d'une session. Un utilisateur a le droit d'exécuter sur les objets les seules opérations que son rôle activé lui autorise.

Nous évoquerons ci-dessous le concept de la hiérarchie de rôles employée dans le modèle RBAC.

2.4.3.1 La hiérarchie des rôles

Le concept de hiérarchie est introduit entre les rôles. Il reflète la hiérarchie naturelle qui existe dans l'entreprise. Cette hiérarchie a été décrite dans **[NO99]** en terme de permissions : un rôle r1 hérite du rôle r_2 si toutes les permissions de r_2 sont aussi des permissions pour r_1. Les auteurs **[FSG+01]** mettent en exergue l'existence de deux types de hiérarchie, la hiérarchie génerale des rôles et la hiérarchie limitée.

- La hiérarchie génerale : RH $\subseteq$ ROLES $\times$ ROLES définit un ordre partiel sur l'ensemble des rôles, symbolisé par $\geq$. Elle intègre aussi le concept de l'héritage multiple. Etant donnés deux rôles r_1 et r_2 : $r_1 \geq r_2 \Rightarrow$ authorized-permissions $(r_2) \subseteq$ authorized-permissions $(r_1) \wedge$ authorized-users $(r_1) \subseteq$ authorized-users (r_2), avec authorized-users (r) = { u $\in$ USERS , (u,r) $\in$ UA} (r étant un rôle) et authorized-permissions (r) = { p $\in$ PRMS, (p,r) $\in$ PA} (p étant une permission) .
- La hiérarchie limitée se base sur les mêmes principes que l'hiérarchie génerale, cependant elle ne supporte pas l'héritage multiple, ceci est traduit formellement par la contrainte suivante :

  $\forall$ r, r_1, $r_3 \in$ ROLES, $r \geq r_1 \wedge r \geq r_2 \Rightarrow r_1 = r_2$.

  Il est ainsi prohibé d'avoir un descendant de deux rôles différents.

2.4.3.2 Les contraintes dans RBAC

Ultérieurement, la notion de séparation de tâches a été ajoutée dans le modèle RBAC. Celle ci stipule qu'aucun utilisateur ne possède assez de privilèges pour abuser seul du système, afin d'assurer que les fraudes et les erreurs majeures ne peuvent avoir lieu que par une collaboration préméditée de plusieurs utilisateurs. Au sein des organisations, ceci se traduit par le fait d'empêcher l'utilisateur de jouer des rôles conflictuels ou aussi de restreindre le nombre de rôles associés aux utilisateurs d'une façon statique ou dynamique.

Les auteurs dans **[FSG+01]** distinguent deux types de séparation de tâches, la séparation statique (SSD) et la séparation dynamique (DSD), la première définit les rôles conflictuels ne pouvant pas être attribués à un même utilisateur, la seconde impose ce contrôle dynamiquement par une contrainte sur les rôles lors de l'activation d'une session donnée.

Par ailleurs, les auteurs **[FSG+01]** définissent aussi des fonctions administratives qui assurent la création et la maintenance des divers éléments et relations du modèle proposé.

2.4.3.3 Les atouts du RBAC

L'approche d'un contrôle d'accès basé sur les rôles présente des avantages incontestables sur différents plans :

- L'administration des autorisations : celle-ci s'effectue en deux étapes :
  - L'attribution des rôles aux utilisateurs
  - L'allocation des autorisations aux rôles pour permettre l'accès aux objets.

  Ceci facilite l'administration de la politique de sécurité dans l'entreprise et la rend plus adaptée aux fluctuations occurentes. Par exemple, quand un nouvel utilisateur rejoint l'entreprise, l'administrateur doit uniquement lui affecter les rôles correspondants à son emploi ; quand la fonction d'un utilisateur change, l'administrateur doit modifier les rôles qui lui sont associés ; lorsqu'un nouvel emploi est crée dans l'organisation, l'administrateur doit uniquement décider quels rôles sont aptes à l'exécuter.
- La hiérarchie des rôles pour mieux s'adapter à l'organisation hiérarchique des entreprises.
- Le concept du moindre privilège : ce concept stipule que les utilisateurs sont assignés aux seuls rôles nécessairs pour assumer leurs tâches. Dans RBAC ceci se manifeste de la façon suivante : les utilisateurs peuvent choisir les rôles qu'ils souhaitent activer. Ainsi, ceux qui détiennent des rôles puissants ne les activent que dans les cas de nécessités ; ceci minimise les dégâts causés par les chevaux de Troie et les intrusions.
- La séparation des tâches et la spécification des contraintes qui permettent d'affermir la sécurité des systèmes.
- RBAC peut englober les traditionnels DAC et MAC ceci a été prouvé dans **[OSM00]** par une construction systématique.

Il reste à noter que les atouts de RBAC attirent vers lui les géants de l'informatique tels que IBM et NSA. Il a été appliqué dans des domaines complexes et distribués. Citons par exemple Dresdner Bank **[SMJ01]**, une banque européenne internationale comportant 50 659 employés et 1 459 branches dans le monde dont la branche principale est située en Allmagne.

2.4.3.4 Critiques du RBAC

RBAC a été critiqué dans **[AEB+03]** pour les raisons suivantes :

- L'absence de structure générique de permissions. Celles-ci sont considérées comme dépendantes de l'application concrète du modèle.
- Le concept de hiérarchie de rôle est quelque peu ambigu. En général, la hiérarchie des rôles ne correspond pas tout à fait à la hiérarchie organisationnelle. Par exemple, le

directeur de l'hôpital a un rôle administratif supérieur au rôle de médecin. Pour autant, un directeur de l'hôpital n'est pas nécessairement un médecin, ainsi il n'est pas faisable d'accorder au directeur les permissions du médecin.

- La distinction entre le concept de rôle et celui de groupe est floue.
- L'impossibilité d'exprimer des permissions et notamment des permissions qui dépendent du contexte. Par conséquent, il serait difficile de spécifier qu'un médecin n'a la permission d'accéder au dossier médical d'un patient que si ce dernier est son patient.

Pour faire face aux limites de RBAC, un modèle ORBAC (Organisation Based Access Control) a été proposé **[AEB+03]**. Ce modèle étend le modèle RBAC et attribut les permissions, obligations, recommandations et interdictions pour réaliser des activités à un rôle dans une organisation, sur un groupe d'objets dans un contexte donné **[AD03]**. La motivation était de sécuriser les systèmes d'information et de communication dans le domaine de santé.

Sachant que le but ultime de la spécification des politiques de sécurité est que celles-ci soient vérifiées, nous présentons, dans la section suivante, les mécanismes d'implémentation des politiques de sécurité où nous abordons les deux techniques de véification statique et dynamique.

2.5 Les méchanismes d'implémentation

2.5.1 L'approche security kernel et le concept reference monitor

Pour avoir un maximum de protection pour les données sensibles, une stratégie rigoureuse de développement de l'architecture d'un système est requise. L'approche noyau de sécurité « security Kernel » a été développée en 1972 par Roger Schell, attaché au département de la défense américaine, et redéfini en 1983 **[AGS83]**. Elle permet d'offrir une méthodologie de construction des systèmes sécurisés. Cette approche est basée sur le concept de moniteur de référence « reference monitor » qui est une combinaison de matériels et de logiciels pour imposer la politique de sécurité du système **[Gass88]**.

L'idée principale de cette approche est que dans un système d'exploitation large, une petite quantité de logiciels est responsable de la sécurité. Ainsi, en modifiant la structure du système de manière à isoler les logiciels attachés à la sécurité dans un noyau sûr « trusted kernel », le contrôle de sécurité devient plus performant puisqu'il se limite à un noyau du système au lieu de l'ensemble entier.

Les accès autorisés que possèdent les sujets sur les objets, selon la politique de sécurité adaptée, sont stockés dans une sorte de base de données. Le moniteur de référence est un médiateur incontournable dans toutes les relations entre sujets et objets, il a pour fonction de prohiber tous les accès non autorisés selon la base de données. La figure 2.6 illustre ce qui est énoncé.

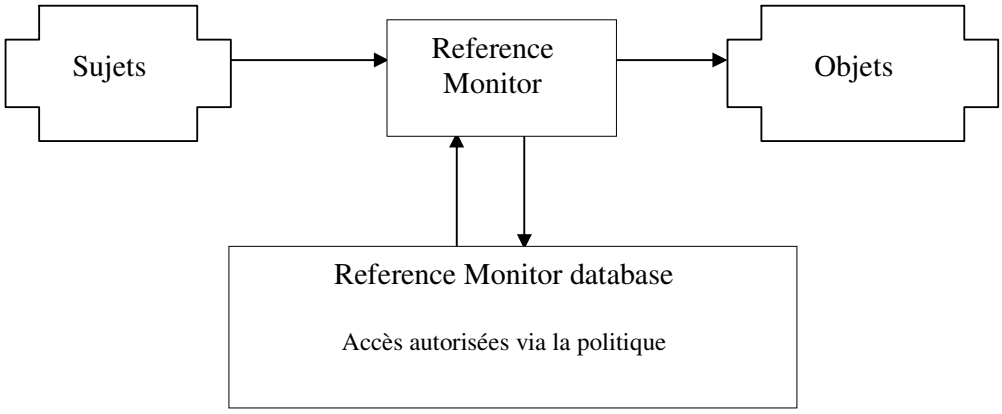

Figure 2.6 : La tâche du reference monitor

Le noyau doit être suffisament sécurisé (tamperproof). Il doit être inévitable (nonbypassable), ce qui signifie que toutes les demandes d'accès doivent passer sous son contrôle. Sa taille doit être suffisamment réduite afin de faciliter les tâches de vérification. Ces restrictions correspondent respectivement aux trois principes de base: isolation (isolation), complétude (completeness) et vérification (verifiability).

Le noyau de sécurité est formé d'une couche de sécurité matérielle et logicielle introduite entre le matériel et le système d'exploitation. Les matériels et les logiciels du noyau de sécurité sûrs, se situent dans le périmètre de sécurité du système, un périmètre qui contient les entités qui sont fondamentales à la sécurité et qui doivent être bien contrôlées. Par contre, le système d'exploitation se trouve à l'extérieur du périmètre de sécurité comme les applications. Le noyau de sécurité « security kernel » assure la sécurité en contôlant les actions du système d'exploitation, tandis que le système d'exploitation assure des services en contrôlant les actions des applications. Pour illustrer ceci, la figure 2.7 trace la structure d'un système sans security kernel et la figure 3.3 présente la structure d'un système renferment un noyau de sécurité, elle est tirée de **[Gass88]**.

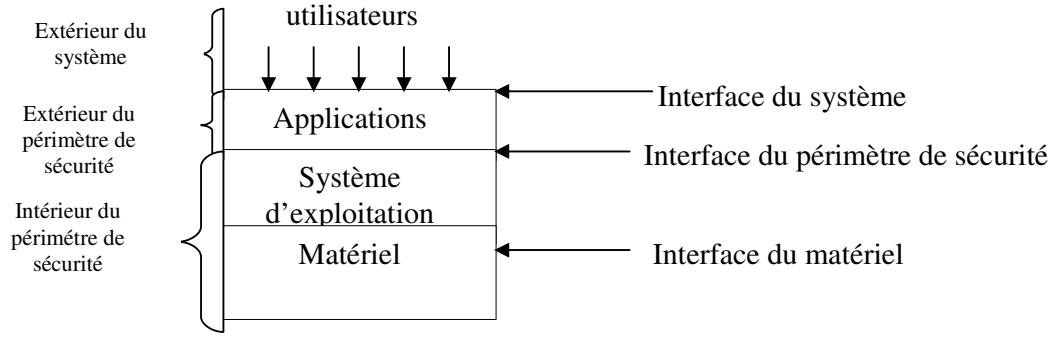

Figure 2.7 : la structure générique d'un système informatique

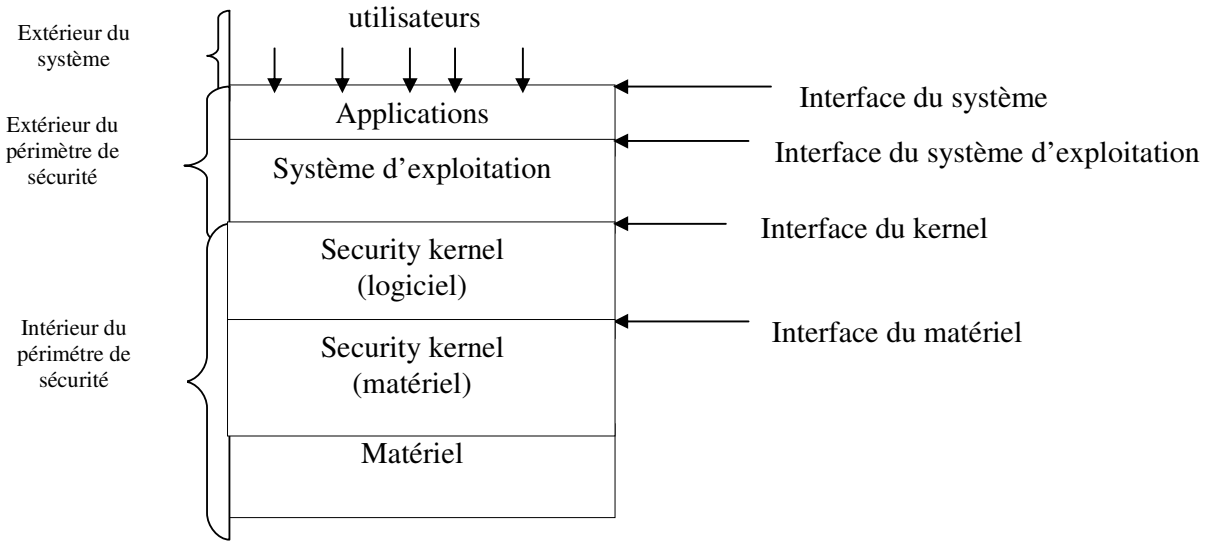

Figure 2.8 : le kernel de sécurité dans un système informatique

En effet, le noyau de sécurité peut être considéré comme étant un système d'exploitation primitif qui effectue des services pour le système d'exploitation qui à son tour offre des services pour les applications. Le noyau de sécurité implémente les fonctions relatives à la sécurité du système d'exploitation, ce dernier est tenu d'assurer le fonctionnement demandé.

Enfin, nous signalons que le moniteur de référence permet d'effectuer la vérification de la sécurité à l'exécution, ceci est désigné aussi par vérification dynamique. Cette approche a pour effet de ralentir l'exécution des applications, aussi se heurte-t-elle à quelques obstacles : l'impossibilité de traiter tous les comportements possibles des utilisateurs des applications. Une autre approche existe, la vérification statique, visant à accélerer l'exécution des applications. Elle agit indépendamment de toute application. Dans le cadre de notre approche, nous avons employé la technique de vérification statique. Dans la section suivante, nous parlerons de la sécurité dans Java où les deux techniques de vérification sont employées.

2.5.2 La Sécurité dans Java

Depuis son apparition en 1995, la popularité de Java ne cesse d'accroître. Java est également utilisé pour le développement des systèmes à base de composants, dans les systèmes embarqués, les smart card et le domaine de e-commerce. Un des principaux atouts de Java est sa portabilité résumée dans la phrase Write once, run anywhere™ (Ecrivez une fois, faites tourner n'importe où). Java est aussi voué à une facilité de programmation. Il possède beaucoup de fonctionnalités qui permettent de créer des programmes robustes dans un laps de temps plus court qu'avec d'autres langages de programmation.

Comme énoncé précédemment, l'architecture de Java permet aux utilisateurs de charger dynamiquement des programmes (code mobile) sur le réseau et de les exécuter localement. Les navigateurs web pouvant interpréter le byte code Java, tels que Netscape et internet

explorer installent et exécutent localement des applets via le web. Cette tâche s'effectue automatiquement et sans préavis pour l'utilisateur. Comme les machines d'exécution du code ne sont pas les mêmes que la machine de développement, il est essentiel que des pirates ne puissent pas se servir des applets pour transporter des virus ou des chevaux de Troie. La plate-forme Java apporte une protection efficace contre de telles menaces. La première barrière introduite par Java est celle de la protection des machines clientes contre des attaques potentielles effectuées illégalement par du code de provenance externe, tout simplement en contraignant ce code à ne s'exécuter que dans les limites rigides de protection d'un bac à sable (sandbox), nous exposons ce modèle par la suite.

2.5.2.1 Le modèle bac à sable et ses améliorations

L'idée primordiale de ce modèle est de confiner les codes de provenance externe, considéré à priori « non digne de confiance » dans un bac pour limiter leurs accès aux ressources du système. Ce modèle a été ensuite amélioré pour couvrir une granularité plus fine de sécurité. Cependant les composants principaux du modèle demeurennt les mêmes. Nous les présentons dans la figure 2.9 :

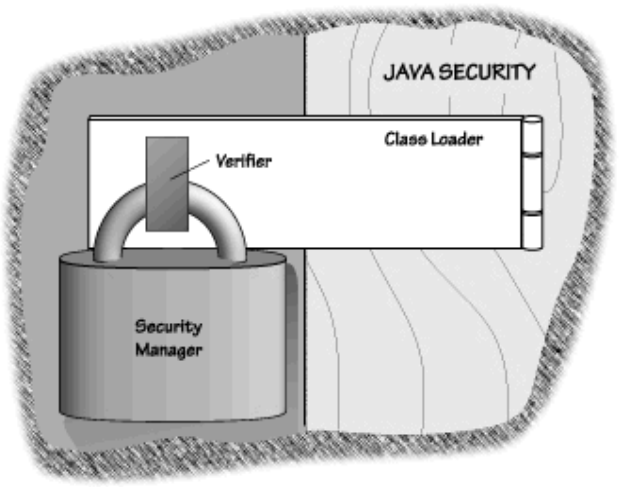

Figure 2.9 : Les composants du bac à sable **[Sec]**

Il s'agit du vérifieur qui assure la vérification du typage, le class loader responsable du chargement et du déchargement dynamique des classes de l'environnement Java runtime, et le security manager qui fonctionne en tant qu'un garde ou reference monitor qui renferme la politique de sécurité et accorde les permissions. Nous détaillons davantage ces composants

- *Le chargeur de classe « class loader »:* Le class loader est responsable du chargement dynamique des classes à partir du disque (chargement interne) ou à travers le réseau (chargement externe) quand la machine virtuelle a besoin du byte code localisé dans une classe spécifique. Il gère l'attribution des espaces de noms uniques pour les classes internes et externes et les sites pour éliminer les risques d'interférence des classes internes et externes. Il est tenu aussi d'empêcher les codes externes malhonnêtes de remplacer les classes internes de l'environnement d'exécution Java. (Par exemple, le remplacement « spoofing » du security manager)

- *Le vérifieur de type :* Avant l'exécution du code sur la machine virtuel Java, le byte code java, produit par une compilation, doit être vérifié pour satisfaire les règles de typage imposé. Ainsi, quand le code externe est chargé dans une classe via le classe loader, le vérifieur l'examine automatiquement. Des exceptions sont générées pour empêcher l'exécution du code ne vérifiant pas les règlements de typage. Cependant, la vérification ne se limite pas au moment de la compilation. Une autre tâche du vérifieur est la vérification dynamique effectuée au moment de l'exécution. Après l'examen du code par le vérifieur, les propriétés suivantes sont garanties :

    - Les classes possèdent des formats corrects.
    - Pas de débordement dans les piles
    - Tous les paramètres des instructions ont un type correct
    - Pas de casting ou de conversions illégales
    - Les accès privés, publiques, protégés et par défaut sont légaux
    - Tous les enregistrements sont valides

- *Le security Manager :* le security manager impose des restrictions sur les activités des applets invoquant les appels de l'API. C'est un objet Java qui effectue les vérifications dynamiques quand un appel critique veut s'effectuer. En effet, lorsque la librarie Java reçoie un appel qui peut être considéré comme tel, elle envoie une requête au security Manager. Celui-ci répond positivement, par l'exécution de la requête, ou négativement, en levant une exception. Des mécanismes de signature et de certification de code sont employés pour signer le byte code. Ajoutons que depuis Java2, le security Manager agit en fonction d'une politique de sécurité que chaque développeur peut configurer pour restreindre les accès des codes signés et répondre à des exigences de sécurité plus fortes. Le security Manager est le reference Monitor qui contrôle, à l'exécution, les accès aux ressources critiques et stoppent les accès non autorisés.

2.5.3 Conclusion

L'état de l'art que nous avons effectué sur les politiques de sécurité ainsi que sur les modèles associés ne couvre pas l'intégralité de ces derniers, cependant elle offre une idée claire sur ce domaine.

Les besoins de sécurité étant diversifiés, ces politiques variées trouvent toujours leur domaine d'application approprié, et la construction systématique d'un système de contrôle d'accès qui traitera ces divers politiques et modèles semble être à la fois originale et utile. Nous avons présenté aussi les techniques de verifications existentes, qui motivent notre travail de spécification de politiques de sécurité.

Dans notre approche, nous spécifions à l'aide d'un langage formel, le B, des modèles basés sur des politiques discrétionnaires, obligatoires et basées rôles. Pour préserver les ressources critiques d'une application contre les tentatives de lecture et d'écriture, nous y interdisons tout accès direct. Il faut passer par un fournisseur de service « un noyau de sécurité » pour obtenir des opérations lire et écrire. A travers une vérification statique que nous avons employée, la méthode B nous garantit par des preuves que les services de lecture et d'écriture ne sont assurées que pour les sujets légitimes.

Nous présentons la méthode B dans le chapitre suivant. L'approche proposée ainsi que l'outil développé font l'objet du quatrième chapitre.

3 La méthode B

3.1 Introduction

Les méthodes formelles permettent de vérifier par la preuve, à l'aide d'outils de preuve, qu'un système vérifie certaines propriétés exprimées avec des notations mathématiques. Les méthodes formelles sont appliquées depuis de nombreuses années dans le domaine du transport ferroviaire, dans le domaine de la sécurité, pour le développement de certains logiciels critiques, et pour le matériel. L'utilisation des méthodes formelles pour la sécurité des systèmes s'est développée ces dernières années de manière très importante, grâce notamment aux nouveaux problèmes de sécurité posés par le développement de l'Internet et du commerce électronique. Nous allons par la suite parler de la méthode B qui a été utilisée avec succès dans le domaine de la sécurité des systèmes critiques.

La méthode B a été élaborée par Jean Raymond Abrial (un des principaux concepteurs de Z dans les années 80) pour spécifier, concevoir et coder des systèmes logiciels. Le manuel de référence principal de cette méthode est The B-Book édité en 1996 **[Abr96]**. En effet, dans cette période, B avait déjà quelques années, a été utilisé dans plusieurs projets industriels et dispose d'outils associés. Sur le plan industriel, le B a été utilisée pour la première fois par Alstom Transport qui a livré dès 1993 aux métros de Calcutta et du Caire, à la SNCF et à la RATP les premiers systèmes de contrôle de vitesse de trains intégrant des logiciels développés avec B. B a aussi intéressé Matra Transport pour la réalisation du métro sans chauffeur Météor, cette dernière a élaboré des outils de preuve et a participé aux discussions sur l'évolution du langage B. Pour pérenniser la méthode et les outils ainsi que pour les besoins de Météor, la RATP s'est associé avec le concours de J. R. Abrial avec un industriel Clearsy (anciennement Steria) à qui Altsom a transféré en 1994 son atelier de développement et qui le développe et le commercialise sous le nom de l'atelier B **[Hab01]**.

Après ce petit aperçu historique, que nous avons évoqué pour mettre en exergue l'importance du B dans le domaine industriel, nous abordons par la suite ses fondements de base. Comme l'énonce le fondateur **[Abr96],** la source principale d'influence se trouve dans les idées portées par C. A. Hoare **[Hoa69]** et E. W. Djkstra **[Dij75]**. Ces derniers, qui perçoivent le programme comme un objet mathématique, ont introduit les concepts de pré-condition et post-condition et de plus faible précondition.

La méthode B utilise un langage pour la spécification fondé sur la logique du premier ordre, appelé aussi langage de substitutions généralisées. Elle s'appuie aussi sur des concepts mathématiques de la théorie des ensembles. Un processus de développement en B s'effectue en plusieurs étapes pour accompagner au fur et à mesure la construction du système. Une première spécification abstraite sera ensuite raffinée progressivement jusqu'à aboutir à une spécification déterministe automatiquement traduisible en un langage de programmation. La méthode B introduit un mécanisme de vérification à travers le calcul d'obligations de preuve. A chaque étape de spécification, ces dernières garantissent la cohérence du modèle ainsi que la conformité d'une spécification par rapport à celle du niveau supérieur.

Nous allons par la suite présenter cette méthode, d'abord nous abordons le langage des substitutions généralisées ainsi que les calculs qui y sont liés, ensuite nous présentons les composants principaux de cette méthode (machines, raffinements et implémentations) ainsi

que la technique de vérification basée sur les preuves. Enfin nous offrons un aperçu rapide de l'atelier B. L'annexe 1 présente les symboles et les notations du langage B que nous avons employés dans notre outil.

3.2 Le langage de substitutions généralisées

Dans une machine B, une substitution est un moyen de représenter une transition **[Hab01]**. L'état d'une machine B est représenté par les variables qui sont typées dans l'invariant. Les substitutions font passer une machine B d'un état initial pré en un autre état post. Les substitutions en B sont utilisées comme les instructions des langages de programmation. Les substitutions peuvent être considérées comme des transformateurs de prédicats qui portent sur les variables d'état de la machine considérée. Etant donné une substitution S, la transformation d'un prédicat P en un prédicat Q s'écrit de la façon suivante : Q = [S]P qui selon E. W. Djkstra détermine pour une postcondition R et une instruction donnée S, la plus faible pré- condition P qui assure que S termine et que R est vraie après S **[BP04]**. La notation [S]P se lit " la substitution S établit le prédicat P ".

En supposant que x,y,z sont des variables, E,F,V sont des expressions, S , T, U, W sont des substitutions généralisées et I,J,P,Q, R sont des prédicats, alors les substitutions primitives sont récapitulées dans le tableau suivant :

x := E	Substitution simple
x, y :=E, F	Substitution multiple simple
skip	Substitution sans effet
P | S	Substitution préconditionnée
P ==> S	Substitution gardée
S [] T	Substitution de choix borné
@z.S	Substitution de choix non borné
S ; T	Séquencement de substitutions
W(P, S, J, V)	Substitution d'itération

Figure 3.1 : les substitutions primitives

La substitution préconditionnée permet de spécifier des contraintes sur les variables et sur les paramètres d'entrée d'une opération. Ces contraintes doivent être satisfaites avant l'exécution de l'opération pour que celle-ci produise les résultats attendus. L'opération ne peut être invoquée que si sa précondition est vérifiée.

La substitution gardée permet de spécifier des conditions et d'établir des hypothèses sous lesquelles la substitution S sera utilisée. Contrairement à la substitution préconditionnée, la garde n'a pas à être prouvée.

La substitution choix borné est une substitution non déterministe qui décrit le fait de choisir arbitrairement la substitution S ou T. Par exemple la substitution x :=1 [] x :=2 permet d'affecter à la variable x la valeur 1 ou 2.

La substitution choix non borné est aussi une substitution non déterministe, elle correspond à l'application de la substitution S pour une valeur quelconque de z. Par exemple, la substitution @y.(y ∈ E =>x :=y) permet d'affecter à la variable x n'importe quel élément de l'ensemble E.

Signalons que le séquencement n'est admis que dans les raffinements et les implémentations, et que l'itération n'est possible que dans les implémentations.

En B, des notations verbeuses sont introduites pour faciliter la lecture et introduire des abréviations. Elles sont présentées dans le tableau suivant :

Substitution	**Notation en B**
P | S	PRE P THEN S END
P ==> S	SELECT P THEN S END
S [] T	CHOICE S OR T END
@z.S	VAR z IN S END
W(P, S, J, V) S	WHILE P DO S INVARIANT J VARIANT V END BEGIN S END
P| S==>S	ASSERT P THEN S END
P==>S [] ¬P==>T	IF P THEN S ELSE T END
S[]T[]…[]U	CHOICE S OR T … OR U END
@z.(P==>S)	ANY z WHERE P THEN S END
ANY x,…,y WHERE x=E ∧…∧y=F THEN S END	LET x,…,y BE x=E∧…∧y=F IN S END
ANY x' WHERE x' ∈ E THEN x:=x' END	x :∈ E
ANY x' WHERE [x$0,x:=x,x']P THEN x:=x' END	x: P
IF P THEN x:=TRUE ELSE x:=FALSE END	x:=bool(P)
f:=f <+ {x |->E}	f(x):=E

Figure 3.2 : les notations verbeuses en B

3.2.1 Substitution multiple généralisée

La substitution multiple généralisée permet d'effectuer simultanément plusieurs substitutions. Par exemple la substitution « S || T » signifie faire simultanément S et T. Cette substitution n'est définie que si les variables modifiées par S et celles modifiées par T sont disjointes, autrement un conflit de valeur aura lieu. Signalons que cette substitution ne permet pas la réalisation des processus parallèles, elle est employée lorsque l'ordre dans lequel s'exécutent les substitutions est non défini. Cette substitution peut se ramener à une substitution généralisée primitive, par exemple, x :=E || y :=F est équivalente à x,y :=E,F

3.3 Sémantique par plus faible précondition des substitutions primitives

Pour chaque substitution, on calcule la plus faible précondition qui assure que la substitution termine et que l'état final satisfait la postcondition R. Cette idée a été introduite par E. W. Dijkstra **[Dij75]** qui détermine, pour une postcondition R et une instruction S, la plus faible précondition P qui assure que S termine et que R est vraie après S. La substitution simple [x :=E] R est le remplacement uniforme de x par E dans R. le tableau ci-dessous décrit le calcul de la plus faible précondition pour le langage des substitutions généralisées.

Cas de substitution	**Réduction**	**Condition**
[x,y :=E,F]R	[z :=F][x :=E][y :=z]R	z\E,F,R
[skip]R	R	
[P|S]R	P∧[S]R	
[P==>S] R	P=>[S]R	
[S []T]R	[S]R∧[T]R	
[@z.S]R	∀ z.[S]R	z\R
[S ;T]R	[S]([T]R)	

Figure 3.3 : le calcul de la plus faible précondition

La notation z\E,F,R signifie que z n'est pas libre dans E, F et R.

Notons que dans le cas d'une substitution préconditionnée, la plus faible précondition est une conjonction de la précondition et de la substitution de S dans R. Ceci traduit le fait que la précondition doit être toujours vérifiée pour que la substitution termine et assure la post-condition.

La substitution d'itération W(P,S,J,V) signifie « tant que le prédicat P est vrai, faire la substitution S ». L'invariant de la boucle est le prédicat J qui doit être vrai à l'entrée de la boucle et se maintenir vrai tant qu'elle s'exécute. La boucle s'arrête quand P devient faux. L'expression V est le variant de la boucle. C'est une expression positive qui décroît à chaque étape d'exécution de la boucle. La plus faible précondition qui assure que

W(P,S,J,V) termine dans un état qui satisfait la postcondition R contient toutes ces informations sur l'invariance de J et la décroissance de V **[BP04]**.

[W(P,S,J,V)]R	
⇔	
J ∧	L'invariant est une précondition
∀x .((J∧P) ⇒ [S] J) ∧	il est conservé dans la boucle.
∀x .((J ⇒ V∈ N) ∧	Le variant est un entier naturel
∀x .((J ∧P) ⇒ [n :=V][S](V<n)) ∧	qui décroît à chaque pas.
∀x .((J∧ ¬P) ⇒ R)	La sortie de boucle implique R.

Figure 3.4 : la plus faible précondition de la substitution itération

3.4 Calcul dans les substitutions généralisées

Nous abordons maintenant les concepts de terminaison des substitutions, de faisabilité et du prédicat avant après. Ces concepts sont formellement définis dans le B-Book pour toutes les substitutions généralisées.

3.4.1 Terminaison

La terminaison d'une substitution est caractérisée par le prédicat trm(S). Une substitution ne se termine pas si elle n'établit aucune post-condition. Trm(S) spécifie les conditions pour lesquelles la substitution S termine. Elle est défini comme suit :

Notation	Définition
trm(S)	[S] true ou [S](x=x)

La terminaison des substitutions primitives peut être calculée à partir de cette définition et du calcul de la plus faible précondition, pour avoir plus de détails, on peut se référer aux ouvrages de référence tels que **[Abr96]** et **[BP04]**. Nous représentons la terminaison de la substitution pré-conditionnée qui est définie de la façon suivante :

trm (P|S)	P∧trm(S)

La conjonction matérialise le fait que la précondition P doit être vérifiée pour assurer la terminaison d'une telle substitution.

3.4.2 Faisabilité

La faisabilité d'une substitution s'exprime à travers le prédicat fis(S). Une substitution est non faisable si elle établit n'importe quelle post-condition. C'est le cas d'une substitution

gardée dont la garde est toujours fausse. Le prédicat fis(S) spécifie le prédicat qui est vrai si et seulement si la substitution S est faisable.

Notation	Définition
fis(S)	¬[S]false ou ¬ [S](x ≠ x)

3.4.3 Prédicat avant-après

Le prédicat avant-après determine la relation entre les valeurs d'une variable x avant et après une substitution S. La valeur avant est noté x et la valeur après est notée x' **[BP04]**. On note $prd_x(S)$ la relation avant-après des variables x pour la substitution S. Elle est définie ainsi :

Notation	Définition
$prd_x(S)$	¬ [S](x'≠ x)

3.4.4 Relation entre les prédicats

Ces concepts ne sont pas mutuellement exclusifs, J.R. Abrial démontre dans son ouvrage **[Abr96]** les propriétés suivantes :

- La faisabilité d'une substitution est liée à l'existence d'une valeur après x' qui peut être calculée à partir de la valeur avant x, ceci s'exprime ainsi :

  $fis(S) \Leftrightarrow \exists x'.prd_x(S)$

- Toute substitution généralisée peut être mise sur une forme normalisée exprimée en fonction du prédicat avant-après et de la terminaison :

  $S = trm(S) \mid @x'.(prd_x(S) \Rightarrow x := x')$

Nous allons maintenant présenter les composants principaux d'un développement en B. Il s'agit des machines abstraites, des raffinements et des implémentations. Nous expliciterons aussi les obligations de preuve générées pour ces composants.

3.5 Machine abstraite

Une machine abstraite est la brique de base d'un modèle B. Elle possède des données (appelées variables) et des services (opérations) qui permettent de manipuler ces données encapsulées. Une machine B représente l'abstraction de l'état du système. Cet état est représenté par les variables qui sont typées par des expressions mathématiques ensemblistes dans la clause INVARIANT. Celle-ci contient aussi les propriétés, que les variables doivent vérifier, exprimées au moyen de prédicats en logique du premier ordre. L'initialisation et les opérations sont décrites à l'aide du langage des substitutions généralisées. Les substitutions font passer une machine B d'un état initial pré en un autre

état post. Elles sont utilisées comme les instructions des langages de programmation. Nous présentons par la suite l'architecture d'une machine abstraite B.

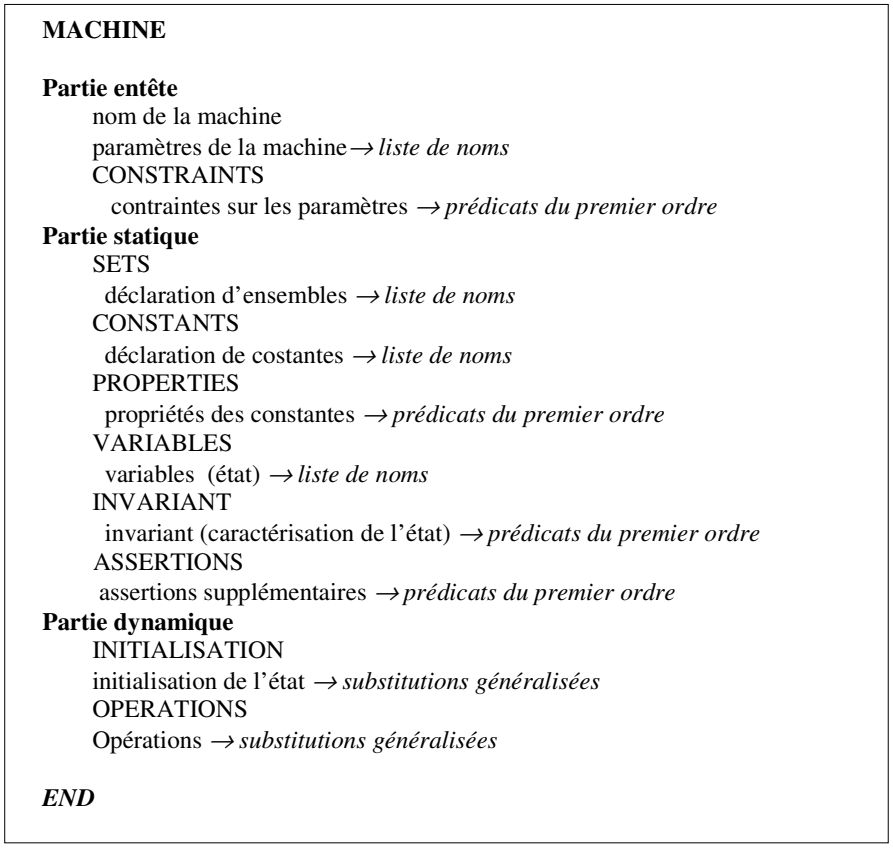

Figure 3.5 : l'architecture d'une machine abstraite en B

La partie statique de la machine correspond aux déclarations de variables, cependant nous tenons à signaler que les variables en B représentent n'importe quel objet mathématique appartenant à la théorie des ensembles que ce soit des ensembles, des relations, des fonctions ou des suites.

La partie dynamique du système est représentée à travers des opérations effectuant les descriptions d'évolutions du système modélisé. L'initialisation et le corps des opérations matérialisent ces transitions. Le principe d'encapsulation est respecté : les variables de la machine ne peuvent être modifiées que par l'intermédiaire des opérations.

Les principales clauses d'une machine B sont :

- Clause CONSTRAINTS : elle permet de typer les paramètres de la machine, et d'exprimer des propriétés, appelées contraintes, sur ces derniers.
- Clause SETS : elle permet de définir des ensembles soit de manière différée, par la simple déclaration de leurs noms via un identifiant, soit en énumérant les éléments qui les composent.
- Clause ASSERTIONS : les assertions sont des prédicats qui sont des conséquences logiques des autres axiomes et de l'invariant. Elles sont des lemmes utiles pour les preuves.

- Clause CONSTANTS : les constantes sont des données immuables, en B il est possible de déclarer des constantes abstraites qui se transforment au cours des raffinements successives en des données implémentables dans un langage de programmation.
- Clause VARIABLES : les variables dans une machine B sont similaires aux variables globales dans les langages de programmation. Une variable est par défaut abstraite mais il est possible de déclarer des variables concrètes qui seront conservées jusqu'à l'implémentation.
- Clause INVARIANT : l'invariant est un prédicat qui décrit les propriétés inchangeables que les variables doivent vérifier.
- Clause INITIALISATION : dans cette clause, une valeur initiale est affectée à chaque variable.

3.6 Les obligations de Preuve

Comme déjà énoncé, un développement en B est basé sur la formalisation des propriétés auxquelles le système doit répondre. La preuve est donc la vérification que le système construit possède les propriétés attendues. Les propriétés qui doivent être préservées sont notamment exprimées à travers des invariants. Sachant que la collecte de ces propriétés se fait le long du développement du système, la preuve accompagne donc l'élaboration du système dans ces différentes phases : spécification, raffinement et implémentation La technique de preuve se base sur la préservation de l'invariant dans la spécification et dans les niveaux qui suivent. Ceci a pour but de vérifier la cohérence interne d'un composant et la correction des raffinements par rapport aux spécifications.

En effet, l'originalité de la méthode B est de pouvoir découvrir et corriger les erreurs dès les phases amont de conception. Cette technique de correction s'effectue grâce à l'élaboration des obligations de preuves, réalisées au fur et à mesure du développement, et effectuées, non pas sur le programme lui-même puisqu'il n'existe pas encore, mais plutôt sur les différents modèles de plus en plus précis que l'on est amené à en élaborer **[Abr00]**.

Une obligation de preuve est constituée d'un but et d'un ensemble d'hypothèses. Démontrer une obligation de preuve consiste à démontrer ce but, en supposant que toutes les hypothèses sont vérifiées. L'outil commercial associé à la méthode B « atelier B » dispose d'un prouveur automatique performant qui permet de décharger automatiquement les obligations de preuve. Dans le cas où la preuve n'a pas pu être vérifié automatiquement, le prouveur interactif, de l'atelier B, permet de décharger ces obligations par l'intervention de l'utilisateur. Durant la phase de vérification des obligations de preuve, les règles de réduction ou de calcul de plus faible précondition sont utilisées.

Nous allons par la suite présenter les obligations de preuve qui sont élaborées à partir de la machine abstraite et des raffinements successifs.

La figure 3.6 présente la machine que nous allons utiliser pour la construction des obligations de preuve :

```
MACHINE
    M(X,u)
CONSTRAINTS
  C
SETS
  G
CONSTANTS
  c
PROPERTIES
  R
VARIABLES
  x
INVARIANT
  I
INITIALISATION
U
OPERATIONS
r ← nom-op (w) = PRE P THEN K END
…
END
```

Figure 3.6 : Machine abstraite

Les obligations de preuves stipulent que l'initialisation établit l'invariant et que les opérations préservent l'invariant si elles sont appelées sous leurs préconditions.

$C \wedge R \Rightarrow [U]I$	initialisation
$C \wedge R \wedge I \wedge P \Rightarrow [K]I$	opérations

Nous pouvons remarquer que dans les obligations de preuve qui se rapportent aux opérations, la précondition en constitue une hypothèse, ceci permet de garantir que la précondition devra être toujours vérifiée pour que l'opération puisse être invoquée. En effet, la vérification des préconditions à l'appel des opérations est assurée par des obligations de preuve spécifiques qui sont :

trm(U)	Pour une initialisation U
$I \wedge P \Rightarrow$ trm(K)	Pour une opération de la forme P|K et un invariant I

Ces obligations sont en général redondantes avec les obligations de preuve de préservation d'invariant et de raffinement. Elles ne sont donc pas produites, sauf si la préservation d'invariant ou la preuve de raffinement ne produisent pas d'obligation de preuve. En effet, pour toute formule R on a :

$R \Rightarrow$ true

$[S]R \Rightarrow [S]$true monotonie de la substitution

Donc dès qu'une substitution S établit une formule quelconque R, la terminaison de S est établie **[Potet]**. Nous traitons dans la section suivante les appels d'opérations et leurs vérifications en détail.

3.7 Les appels d'opération

A travers des liens spécifiques, que nous allons détailler par la suite, une machine peut appeler des services ou opérations d'une autre machine. Ceci s'effectue par l'intermédiaire d'une substitution appelée substitution appel d'opération.

Soit op une opération. Une substitution appel d'opération en B s'effectue de la façon suivante :

Paramètres_sorties ← op (paramètres_entrées)

Le terme paramètres-sorties est une liste de noms de données (d'identificateurs) désignant les paramètres effectifs de sortie de op. Leurs types doivent être identiques aux types des paramètres formels de sortie de l'opération op. Le terme paramètres_entrées appelé aussi paramètres_effectifs d'entrée est une liste d'expressions qui doivent avoir des types identiques à ceux des paramètres formels d'entrée de l'opération op.

La substitution appel d'opération permet d'appliquer la substitution d'une opération, en remplaçant les paramètres formels par des paramètres effectifs :

1. Si op est une opération sans paramètre de sortie et sans paramètre d'entrée, définie par op=S, alors la signification d'un appel de op est la substitution S.
2. Si op est une opération sans paramètre de sortie et avec des paramètres d'entrée, définie par op(X)=S, alors la signification de l'appel op(E) est la substitution : X :=E ; S.
3. Si op est une opération avec des paramètres de sortie et sans paramètres d'entrée, définie par Y←op=S, alors la signification de l'appel de R←op est la substitution : S ; R :=Y.
4. Enfin, si op est une opération avec des paramètres de sortie et des paramètres d'entrée, définie par Y←op(X)=S, la signification d'un appel de R←op(E) est : X :=E ; S ; R :=Y.

Nous allons montrer ci-dessous comment la méthode B garantit que les appels aux opérations vérifient toujours les préconditions de celles-ci.

Soit op une opération définie par : Y ← op(X)= P1|S1 et soit l'appel R←op(E). Supposons que W est un prédicat. Nous allons maintenant calculer la plus faible précondition de la substitution appel d'opération pour le prédicat W, ce calcul étant utilisé dans les vérifications des obligations de preuve. $[R \leftarrow op(E)]W \Leftrightarrow$

[X :=E ; P1|S1 ; R :=Y] W (application de la définition d'une substitution appel d'opération et remplacement de op par son corps)

⇔ [X :=E] ([P1|S1 ;R :=Y] W) (application de la réduction pour éliminer le premier séquencement)

⇔ [X :=E] ([P1|S1] ([R :=Y] W)) (application de la réduction pour éliminer le deuxième séquencement)

⇔ [X :=E] ([P1∧ ([S1] [R :=Y] W)) (application du calcul de plus faible précondition pour P1|S1)

⇔ [X :=E] P1∧ [X :=E] [S1] [R :=Y] W (distribution de la substitution sur le et logique « ∧ »)

Nous pouvons remarquer que la formule obtenue comporte bien dans la partie gauche de la conjonction, la précondition qui doit être vraie. Nous constatons donc que tout calcul de plus faible précondition se rapportant à une substitution appel d'opération vérifie la précondition de celle-ci.

Puisque la méthode B permet de garantir la vérification de la précondition lors de l'appel des opérations et la préservation de l'invariant, nous l'avons utilisée pour modéliser et vérifier les politiques de sécurité.

3.8 Raffinement

En général, la construction des systèmes complexes sûrs ne s'effectue pas d'une façon monolithique mais à travers des raffinements successifs. Les raffinements permettent de préciser petit à petit l'abstraction tirée du cahier des charges. Lors du raffinement, le non déterministe et la simultanéité tendent à disparaître. Le raffinement d'une machine s'effectue sans modification de la signature des opérations, seuls peuvent changer les variables, les ensembles et les substitutions définissant les opérations. Une substitution S est dite raffinée par une substitution T, si T peut être utilisée à la place de S sans que l'utilisateur ne s'en rende compte.

Ainsi, nous pouvons conclure que le raffinement consiste à :

- Transformer les structures de données abstraites tels que les ensembles, les relations, les fonctions et les suites en structures de données concrètes proche des langages de programmation. Il y a ainsi introduction des tableaux et des variables scalaires.
- Lever progressivement l'indéterminisme des substitutions.
- Remplacer les substitutions abstraites, les substitutions parallèles et les choix par des substitutions concrètes tels que les séquencements, les conditions et les boucles.

Nous signalons aussi que dans un raffinement, il est possible d'utiliser la substitution variable locale de la forme suivante : VAR X in S END, où X est une liste de variables deux à deux distinctes, S est une substitution. [VAR X in S END] P signifie ∀ X. [S] P, P étant un prédicat. La substitution variable locale permet d'introduire une liste non vide de variables locales accessibles en lecture et en écriture dans la substitution S.

3.9 Les obligations de preuve des raffinements

Nous allons maintenant présenter les obligations de preuve pour les raffinements. La figure 3.7 est le raffinement de la machine présentée dans la figure 3.6 :

REFINEMENT
N (X,u)
REFINES
M
SETS
H
CONSTANTS
d
PROPERTIES
O
VARIABLES
y
INVARIANT
J
INITIALISATION
V
OPERATIONS
r ← nom-op(w) = PRE Q THEN L END
…
END

Figure 3.7 : Raffinement

Les obligations de preuve pour les raffinements consistent à prouver la correction d'un raffinement vis-à-vis de ses abstractions. Il s'agit de démontrer que l'initialisation établit l'invariant sans contredire l'initialisation de l'abstraction, et que chaque raffinement d'opération préserve l'invariant sans contredire l'opération de l'abstraction. En effet, comme pour les machines abstraites, la correction de l'initialisation et des opérations des raffinements est assurée par établissement et préservation de l'invariant. Cependant, l'invariant d'un raffinement est un invariant de liaison définissant les propriétés des nouvelles variables par rapport aux variables du composant raffiné.

Pour l'initialisation, il s'agit de démontrer que l'initialisation établit l'invariant sans contredire l'initialisation de l'abstraction. Ainsi les nouvelles valeurs de l'initialisation ne doivent pas être en contradiction avec les anciennes. L'obligation de preuve est la suivante :

$$C \wedge R \wedge O \Rightarrow [V] \neg[U]\neg J$$

Pour les opérations les obligations de preuve sont les suivantes :

$C \wedge R \wedge O \wedge I \wedge J \wedge P \Rightarrow Q$	précondition
$C \wedge R \wedge O \wedge I \wedge J \wedge P \Rightarrow [L] \neg [K] \neg J$	Operation sans résultat
$C \wedge R \wedge O \wedge I \wedge J \wedge P \Rightarrow [\ [r:=r'] L\ ] \neg [K] \neg (J \wedge r=r')$	Opération avec résultat

La notation [r:=r'] L signifie le renommage de r par r' dans la substitution L.

Ceci signifie qu'une opération d'un raffinement est correcte lorsqu'elle préserve l'invariant sans contredire l'opération spécifiée et lorsque sa précondition est moins restrictive que la précondition de l'opération spécifiée (à noter la présence de la précondition de l'abstraction P dans la partie hypothèse et celle du raffinement Q dans la partie but de l'obligation de preuve). Il faut démontrer que les résultats de l'opération raffinée ne sont pas en contradiction avec celles de l'opération du niveau d'abstraction précédant.

3.10 Implémentation

L'implémentation est la dernière phase de raffinement d'une machine abstraite. Elle doit être directement traduisible en langage de programmation. Par conséquent, l'indéterminisme est interdit et toutes les déclarations de données doivent être proches de celles qu'on trouve dans les langages de programmation. Au niveau des implémentations, les seules substitutions permises sont les substitutions déterministes appelées instructions. L'équivalence avec un programme habituel est quasi immédiate. Un composant B qui est une implémentation est écrit à l'aide d'un sous ensemble du langage B appelé B0.

Dans une implémentation, les variables abstraites doivent être toutes concrétisées. Les variables concrètes, de même que les constantes concrètes doivent avoir un typage concret qui peut être :

- un entier représentable
- une valeur booléenne
- une valeur d'un ensemble différé ou énuméré
- un tableau

Les trois premiers items sont des types simples. Un tableau est l'instanciation d'une fonction totale dont le domaine est un type simple ou un produit de types simples, et dont le codomaine est un type simple. Cette fonction totale peut être en plus injective, surjective ou bijective. Pour instancier un tableau, nous pouvons employer l'opérateur maplet ($\mapsto$) entre ses termes ou l'opérateur multiplie * (produit cartésien d'ensembles). Par exemple le tableau ff est représenté de la façon suivante : ff ={ $0 \mapsto 0, 1 \mapsto 10, 2 \mapsto 21, 3 \mapsto 32$}.

Comme pour la correction du raffinement, la correction d'une implémentation n'est pas une simple vérification de cohérence interne, mais aussi une assurance que

l'implémentation respecte sa spécification. Les obligations de preuve relatives à la correction des implémentations comportent des preuves similaires à celles des raffinements, nous ne les détaillons pas dans notre mémoire.

3.11 Architecture de projets B

Le développement en B repose sur deux mécanismes essentiels, le mécanisme de raffinement matérialisé par le lien REFINES et le mécanisme de modularité matérialisé soit par le lien INCLUDES (utilisé dans une machine abstraite ou un raffinement) soit par le lien IMPORTS (utilisé dans une implémentation). L'intérêt de l'importation et de l'inclusion réside dans le fait de permettre à la machine d'appeler les services (les opérations) de la machine importée ou incluse. D'autres mécanismes existent pour effectuer des liens entre les composants (SEES, USES). Les règles qui régissent ces mécanismes se trouvent dans les manuels de référence **[Abr96]** et le manuel de référence du B.

Nous nous limitons à expliquer les clauses INCLUDES et SEES que nous avons utilisées dans notre outil.

La clause INCLUDES peut être utilisée dans une machine abstraite ou un raffinement. Elle permet d'enrichir ces derniers par des constituants (ensembles, constantes et variables) appartenant à d'autres machines ainsi que par leurs propriétés (clause PROPERTIES et INVARIANT). Le lien d'inclusion permet de découper la complexité d'un projet en construisant de manière modulaire des machines abstraites ou des raffinements.

Les opérations d'une machine incluse peuvent être utilisées dans l'initialisation et les opérations de la machine qui réalise l'inclusion. Cependant les variables de la machine incluse ne peuvent être modifiées que via les appels à ses propres opérations. Cette restriction est imposée pour assurer que l'invariant de la machine incluse est toujours préservé. En effet, puisque les obligations de preuve dans cette machine ont vérifié que ces opérations ainsi que leurs appels préservent l'invariant (paragraphe 4.7), la manipulation des variables via les services de la machine ne viole pas l'invariant. Nous signalons aussi qu'il est interdit de faire des appels simultanés d'opérations modifiant l'état (les variables) d'une machine incluse pour ne pas aboutir à un conflit de valeur.

Il est possible d'inclure plusieurs machines et plusieurs instances d'une machine, dans ce dernier cas le nom de la machine est précédé d'un renommage ou préfixe. Chaque préfixe correspond à une instance de la machine renommée.

Ajoutons qu'une instance de machine ne peut être incluse qu'une seule fois dans un projet. Cette restriction est nécessaire pour éviter les conflits d'identificateurs aboutissant à des conflits de valeur.

La clause SEES peut être utilisée dans une machine abstraite, un raffinement ou implantation.

Cette clause permet au composant de consulter les constituants (ensembles, constantes et variables) de la machine abstraite vue sans les modifier.

Ainsi, les ensembles et les constantes de la machine vue sont accessibles dans les clauses PROPERTIES, INVARIANT, ASSERTIONS et dans le corps de l'initialisation et les opérations du composant qui voit. Les variables de la machine vue sont accessibles en lecture dans le corps de l'initialisation et les opérations. Les opérations ne modifiant pas les variables (opérations de consultation) de la machine vue peuvent être utilisées dans l'initialisation et dans les opérations du composant qui effectue le lien SEES.

Signalons que si une machine est vue par un composant alors les raffinements de ce composant doivent également voir cette machine. Ceci a pour but d'assurer que le même comportement est préservé dans le composant et ses raffinements.

3.12 Atelier B

L'atelier B propose des services importants. Il offre aux utilisateurs en réseau la possibilité de synchroniser leurs travaux sur un même projet, il permet le développement modulaire d'un projet, son découpage en sous projets pour pouvoir impliquer plusieurs équipes de développeurs. Il dispose d'un analyseur syntaxique qui permet d'effectuer des vérifications syntaxiques sur les fichiers du langage B. L'atelier B permet aussi de générer automatiquement les obligations de preuve et de les démontrer d'une manière partiellement automatique. Il permet aussi de traduire les implémentations B en C ou Ada. Les principales fonctionnalités de l'atelier B sont représentées dans la figure 3.8.

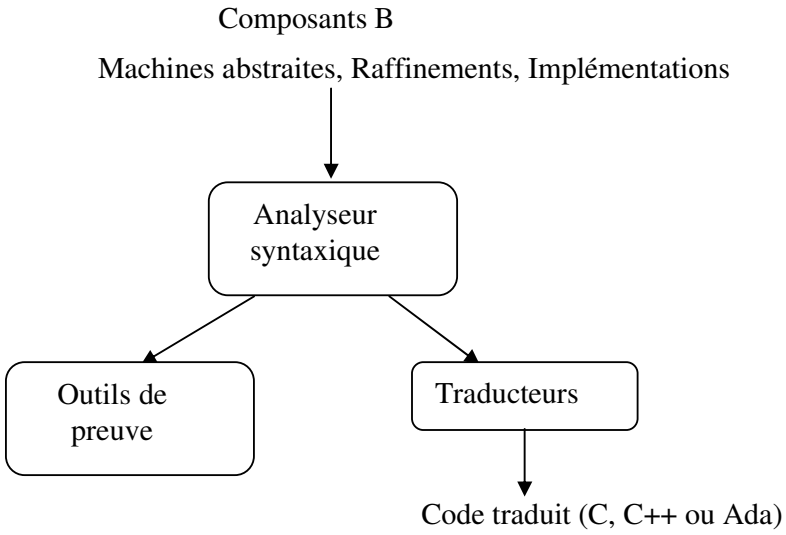

Figure 3.8 : L'atelier B **[Clea]**

Nous parlerons par la suite du prouveur de l'atelier B.

Comme déjà énoncé, la preuve sert à détecter les erreurs et à les corriger dès les phases amont de spécification. Le prouveur de l'atelier B permet d'effectuer cette tâche d'une manière partiellement automatique. Il est muni d'un prouveur interactif qui permet à l'utilisateur de piloter une activité de preuve qui n'a pas abouti automatiquement.

Pour prouver une obligation de preuve, le prouveur de l'atelier B consulte sa base de connaissance qui regroupe l'ensemble des règles mathématiques. Ces dernières sont choisies selon les mécanismes de preuve adoptés. Les forces Rapide, 0, 1, 2 et 3 regroupent les mécanismes en niveaux. Les forces plus élevées offrent plus de performance dans la preuve mais le risque que la preuve boucle s'accroît.

Une obligation de preuve possède un état : prouvé ou non prouvé. Quand l'obligation de preuve est non démontrée, on se trouve confronté à deux possibilités, soit le composant est juste mais le prouveur n'a pas trouvé l'une des démonstrations, dans ce cas il faut passer au

mode interactif pour l'aider, soit le composant est faux et l'obligation non prouvé localise l'erreur. Le cas échéant, peut être dû à un manque d'hypothèses dans la spécification.

Enfin, nous signalons que le prouveur ne peut pas affirmer qu'une obligation de preuve est clairement fausse. En fait, infirmer une obligation de preuve est un problème théoriquement beaucoup plus difficile que de la démontrer, puisqu'il s'agit de choisir des valeurs vérifiant les hypothèses mais pas les conclusions **[Hab01].**

3.13 Conclusion

B est une méthode de développement couvrant tout le spectre depuis la spécification jusqu'au code. Elle est basée sur une axiomatisation explicite de la théorie des ensembles typés. B contient un mécanisme de structuration qui est la machine abstraite, transformée au cours du développement en raffinement puis en implémentation. La méthode de développement est basée sur des théories mathématiques complètement explicitées : la théorie des substitutions généralisées, la théorie du raffinement, la théorie de la construction de logiciels en couche.

Les points clés en faveur de la méthode B sont liés au fait qu'elle permet le développement de projets dans un contexte industriel. Par exemple, la partie sécuritaire du SAET Météor représente 107000 lignes de code en langage B (1075 composants) génère 29000 obligations de preuve et 87 000 ligne de code Ada traduites. Par ailleurs les outils de preuve de l'atelier B sont performants, le pourcentage de preuves démontrées automatiquement, sans ajout de règles de l'utilisateur est supérieur à 80% pour les projets présentés **[Clea]**.

La méthode B qui au départ était une méthode de développement de logiciels prouvés sûrs est devenue au fil du temps une méthode d'analyse et de construction « système » qui peut être utilisée dès la phase de rédaction du cahier des charges d'un système, puis ensuite éventuellement dans les phases de spécification générale et de conception **[Abr00]**.

Enfin nous signalons que le langage B est bien vivant, il évolue il s'améliore. D'autres extensions y sont ajoutées mais nous ne les avons pas présentées ici. Notre but était de donner un aperçu rapide de cette méthode et de ces atouts pour notre travail. En effet, la formalisation des politiques de sécurité dans le cadre de cette méthode nous permet de vérifier la correction du modèle et de prouver des propriétés de sécurité sur tout le système de contrôle d'accès construit. Précisement, la garantie offerte par la méthode B qui se rapporte à la vérification des préconditions lors de l'appel des opérations nous a permis de garantir la sécurité des accès effectués par les sujets sur les objets. Ces accès sont réalisées par des appels à des opérations de lecture et d'écriture dont la précondition précise, selon la politique de sécurité imposée, les sujets habilités à lire ou à écrire l'objet en cours.

4 Approche et mise en œuvre

4.1 Problématique

Dans un monde où les attaques de toutes sortes se multiplient, la construction d'applications sécurisées devient une nécessité. Assurer la sécurité d'une application est d'autant plus important lorsque celle-ci est intégrée dans des systèmes critiques. Dans de tels systèmes, la moindre faille peut avoir des répercussions très graves.

En effet, une application permet d'assurer une tâche ou une fonction au sein d'une organisation. D'un point de vue technique, une application est composée des services (les opérations) qui manipulent des données (les variables). Les variables référencent souvent des ressources à protéger selon la politique de l'organisation. Il est alors nécessaire de s'assurer que les services de l'application ne puissent détourner la politique imposée. Il est aussi important de pouvoir détecter les failles de sécurité dès les phases amont de conception. Une correction tardive d'une erreur coûte infinement plus chère qu'une correction précoce.

Par ailleurs, l'utilisation des méthodes formelles, pour la spécification et la vérification des modèles de sécurité, est préconisée par des normes internationales d'évaluation, telles que les critères communs **[CC05]**, pour assurer des niveaux de certification élevés des systèmes d'information.

4.2 Approche

Pour répondre aux besoins évoqués ci-dessus, nous utilisons la méthode formelle B pour spécifier des différents politiques de sécurité proposées dans la littérature. Nous nous intéressons à la vérification de ces politiques, puisque l'objectif d'une politique de sécurité réside dans la vérification que celle-ci est bien respectée. La technique de vérification en B, basée sur les preuves, nous permet de détecter les failles de sécurité dès les phases amont de conception d'applications, c'est à dire sans exécution de celle-ci. Cette technique est appelée vérification statique. L'intérêt de cette forme de vérification est qu'elle est valable pour toute exécution et non uniquement pour certains jeux de valeurs particulières, comme c'est le cas pour le test.

Nous adoptons une démarche basée sur la technique de raffinement en B qui permet de construire successivement des applications tout en préservant au mieux les exigences de sécurité. Pour préserver les ressources critiques (les variables d'une application), tout accès direct en lecture ou écriture, effectués par les services (opérations) doit être interdit. Il faut passer par un noyau sûr qui offre ses services pour les sujets légitimes selon la politique imposée. Le raffinement de l'application principale réalise ce passage. Les obligations de preuve générées, qui se rapportent aux appels des services sécurisés, permettent de raisonner sur la sécurité de la spécification de l'application.

4.3 Objectif et concept de l'outil

Pour vérifier si une application satisfait des exigences fortes de sécurité, nous adoptons le principe de la vérification statique et des noyaux de sécurité qui interceptent tous les accès des sujets aux objets.

Nous édifions un noyau sûr offrant les services de lecture et d'écriture aux seuls sujets autorisés. Pour ce faire ce noyau, matérialisé par une machine B, a besoin de connaître les règles qui gèrent les accès des sujets aux objets. La présentation de ces règles dépend de la politique que l'organisation souhaite mettre en place (que ce soit discrétionnaire, obligatoire ou basée rôle). Nous insistons sur le fait de traiter différents types de politiques puisque c'est là où réside l'originalité de notre travail.

Ainsi, nous offrons à l'utilisateur des squelettes sous forme de machines B formalisant ces trois types de politiques, ce dernier doit les remplir de la façon indiquée (section 4.4.1). Le noyau de sécurité se base sur les règles spécifiées pour former les opérations de lecture et d'écriture pour chaque variable, les préconditions des opérations spécifiant les sujets autorisés. Pour raisonner sur la sécurité de l'application, l'outil examine la spécification de l'application, transforme toute tentative directe d'accès aux objets, en un appel aux services sûrs du noyau de sécurité. L'application principale est ainsi raffinée par une application sécurisée.

Les obligations de preuve générées par la méthode B, précisément celles qui se rapportent aux appels d'opérations, nous permettent de vérifier le respect des préconditions lors de l'appel. Dans le cas de failles de sécurité matérialisées par le non respect des préconditions, le comportement prévu de l'application ne sera pas assuré. Les obligations de preuve non prouvées signalent ces failles. La technique de vérification utilisée est une vérification statique.

Signalons que cet outil a été développé en Java. Pour analyser les machines d'entrée et construire les machines de sortie, nous avons utilisé des fonctions implémentées en Java dans la BOB (Boîte à Outils B) développée par l'équipe VASCO du LSR. La BOB offre entre autres des fonctionnalités de calcul sur les substitutions : garde, terminaison, prédicat avant après.

Pour expliquer le fonctionnement de notre outil, nous donnons son schéma avec ses éléments d'entrée et de sortie (ce sont des machines B que nous allons décrire par la suite).

4.4 Le schéma de l'outil

La figure 4.1 schématise l'outil avec ses entrées et ses sorties. APPLICATION.mch est la machine fournie par l'utilisateur. POLITIQUE.mch est la machine contenant la politique de sécurité. Cette machine varie selon le type de politiques traitées (discrétionnaire, obligatoire ou basée rôle). APPLICATION2 est la machine APPLICATION à laquelle les constantes ont été retirées et déportées dans la machine CONSTANTES. Cela permettra au composant (module) CONTROLE_ACCES d'accéder aux constantes dont il a besoin. A partir des deux machines concernant l'application et la modélisation de la politique, l'outil génère les composants suivants : CONSTANTES, APPLICATION2, CONTROLE_ACCES et APPLICATION2_REF.ref.

APPLICATION2_REF.ref est le raffinement de la machine APPLICATION2. Elle est obtenue en remplaçant tous les accès directs des sujets sur les objets par des appels des services de la machine CONTROLE_ACCES. Les liens entre les différentes machines sont explicités dans le schéma ci-après.

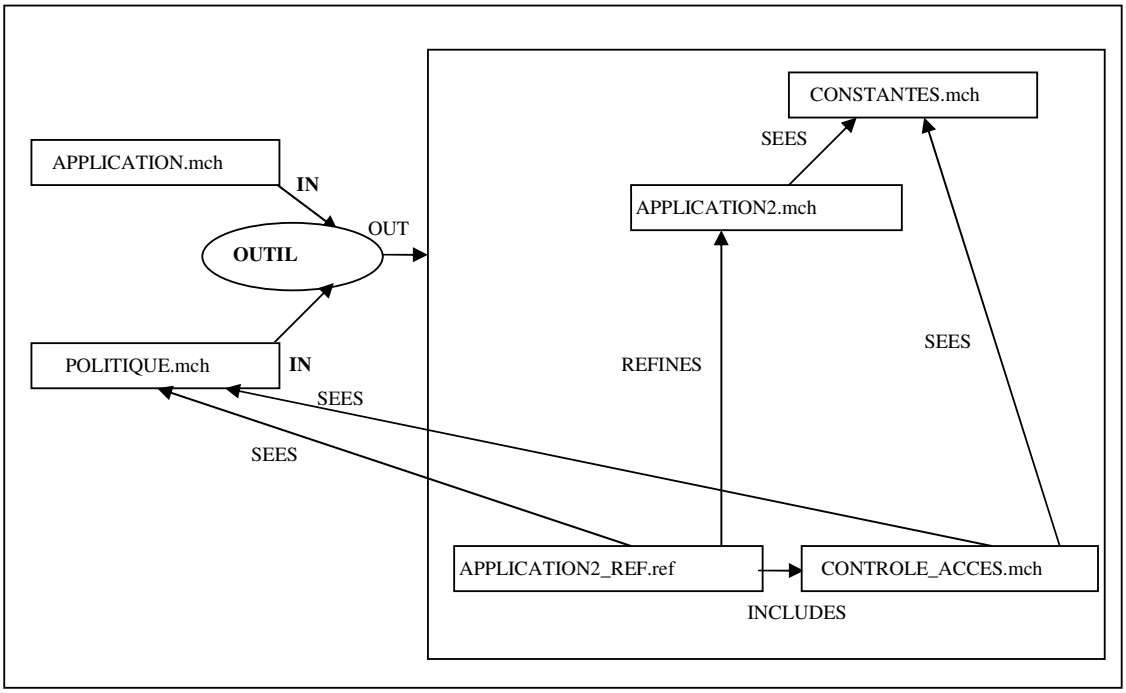

Figure 4.1 : le schéma de l'outil

Signalons que l'utilisateur de l'outil doit spécifier le chemin du répertoire où les composants doivent être générés.

Nous détaillons dans les parties qui suivent chacune des machines présentes dans le schéma et nous évoquons la manière dont l'outil va les analyser ou les produire.

Signalons que lors de la présentation de la grammaire des machines, les termes où toutes les lettres sont en majuscules sont des mots clés du langage B. Les termes en italique sont des noms réservés de l'outil.

Le terminal ident référence un identificateur, exp référence une expression. Pour avoir des détails supplémentaires sur les expressions en B, le lecteur peut se référer au manuel de référence B **[ManuB02]**.

4.4.1 La machine APPLICATION

La structure de cette machine dépend de l'application modélisée par l'utilisateur. Signalons que cette machine contient, en plus des variables à protéger, d'autres variables requises pour la spécification du cas à traiter. La sécurité de cette application dépend de la manière par laquelle les sujets agissent sur les objets.

Dans notre réalisation, nous avons traité le cas où le typage est donné par des prédicats ensemblistes. Les opérateurs de typage acceptés sont les suivants : $\in \subseteq \subset$. Le seul opérateur restant « = » sera traité dans des améliorations prochaines de l'outil. A présent, celui-ci signale par un message le non respect de cette contrainte et l'exécution est arrêtée.

Dans la suite nous désignons par VAR l'ensemble des variables de cette machine, OPE l'ensemble des opérations.

Cette machine sera ensuite reproduite par notre outil au moyen des deux machines : CONSTANTES et APPLICATION2. Dans la machine des constantes, nous regroupons toutes les constantes de l'application (Les clauses SETS, CONSTANTS et PROPERTIES). Ceci permet à toutes les autres machines du projet de les consulter (Lien SEES). La machine APPLICATION2 renferme toutes les autres clauses de la machine APPLICATION. Par conséquence, elle contient le même ensemble de variables (VAR) et d'opérations (OPE)

4.4.2 La machine POLITIQUE et la machine CONTROLE_ACCES

La machine POLITIQUE représente la spécification de la politique que l'utilisateur souhaite mettre en application. Dans le cadre de notre étude, nous nous intéressons à trois types de politiques : discrétionnaires, obligatoires et basées sur les rôles. Conformément à chaque politique, nous donnons le squelette de cette machine. Signalons que cette machine sera parcourue par l'outil pour générer la machine CONTROLE_ACCES.

Cette dernière construit les opérations de lecture et d'écriture sécurisées : ces opérations ne doivent être permises que pour les sujets habilités. Ceci peut être garanti en utilisant des opérations de lecture et d'écriture pré-conditionnées.

Elle voit les deux machines POLITIQUE et CONSTANTES. Ces opérations de lecture et d'écriture agissent sur les variables à protéger de l'application. Ces variables sont déclarées, typées et initialisées de la même manière que celle présente dans APPLICATION2. Dans notre outil, nous reproduisons exactement les mêmes clauses de déclaration de variables, d'invariant et d'initialisation présents dans APPLICATION2.

La clause OPERATIONS contient les opérations de lecture et d'écriture pour chaque variable de cette machine. Dans chaque opération de lecture et d'écriture nous construisons pour chaque variable l'ensemble contenant les sujets autorisés à exécuter de tels accès. La machine modélisant la politique est parcourue pour la récupération de ces sujets. Sachant que la structure des ces opérations dépend de la politique imposée. Nous la présentons ci-dessous pour chaque genre de politique modélisé.

4.4.2.1 Modélisation d'une politique discrétionnaire

La brique de base d'une politique discrétionnaire est la matrice d'accès représentant les droits que les sujets possèdent sur les objets (Cf. section 2.4.1). Sachant que l'effet des accès effectués par les sujets sur les variables se matérialise par la modification de la valeur de ces dernières (ou écriture) et la récupération de sa valeur (ou lecture), nous subdivisons la matrice d'accès en deux sous-matrice : lectureautorisee et ecritureautorisee.

Le schéma de cette machine se présente de la façon suivante :

MACHINE POLITIQUE
SETS
SUJET ={ Liste_ident } ; OBJET={ Liste_ident }
CONSTANTS
lectureautorisee, ecritureautorisee
PROPERTIES
lectureautorisee ∈ OBJET→ F(SUJET)
∧ ecritureautorisee ∈ OBJET→ F(SUJET)
∧ Pred_eg_Prop1
∧ Pred_eg_Prop2
END

Figure 4.2 : le schéma de la machine POLITIQUE (politique discrétionnaire)

Lectureautorisee est une fonction totale (notation →) qui associe à chaque objet l'ensemble des sujets habilités à le lire. Cet ensemble est vide lorsque l'objet ne peut pas être lu par aucun sujet de l'ensemble SUJET défini. Pred_eg_Prop1 instancie les couples de cette fonction totale.

Ecritureautorisee est aussi une fonction totale de l'ensemble OBJET vers l'ensemble des sous-ensembles finis de l'ensemble SUJET. Les valeurs de ses couples sont présentées dans Pred_eg_Prop2

Ci-dessous, nous présentons la grammaire pour expliquer les termes de cette machine :

Pred_eg_Prop1::= *lectureautorisee* = { Liste_couple }
Pred_eg_Prop2::= *ecritureautorisee* = { Liste_couple }
Liste_couple ::= Liste_couple , Couple | Couple | ε
Couple ::= (ident ↦ {Liste_ident})
Liste_ident ::= Liste_ident , ident | ident | ε

Figure 4.3 : grammaire

Détaillons les clauses de cette machine :

- La clause SETS : elle contient les ensembles énumérés des sujets qui sont les opérations et des objets les variables. Cependant pour éviter un conflit d'identificateurs avec la machine APPLICATION2, les identificateurs désignant les

objets « variables » et les sujets « opérations » dans cette machine et dans la machine APPLICATION2 doivent être différents. Dans notre outil, nous avons choisi un préfixe formé d'une lettre pour chaque variable et chaque opération c'est la lettre f.

- La clause CONSTANTS définit la liste des constantes de cette machine. Ce sont les deux sous-matrices lectureautorisee et ecritureautorisee.
- La clause PROPERTIES : dans cette clause les deux-sous matrices lectureautorisee et ecritureautorisee sont typées et instanciées. La première sous matrice (lectureautorisee) est une fonction totale qui lie chaque variable à l'ensemble des opérations qui sont autorisées à la lire. La deuxième effectue la liaison par une fonction totale entre chaque variable et l'ensemble des opérations autorisées à l'écrire.

4.4.2.2 Les services de lecture et d'écriture

D'un point de vue technique, les opérations de lecture et d'écriture générées dans la machine CONTROLE_ACCES sont les suivantes :

Pour chaque variable à protéger l'opération de lecture est la suivante :

```
val → lire_ var (su) =
      PRE
         su ∈ SUJET
         ∧
         su ∈ lectureautorisee(fvar)
      THEN
         val := var
      END
```

Figure 4.4 : service lire (politique discrétionnaire)

avec var ∈ VAR (l'ensemble des variables de la machine APPLICATION2) ∧ fvar ∈ OBJET.

Le principe est le suivant : la précondition stipule que le sujet effectuant la lecture appartient à l'ensemble des sujets. Plus précisément il doit appartenir au sous-ensemble qui, selon la politique de sécurité spécifiée, possède le droit de lecture. Cette précondition vérifiée, l'opération retourne par le paramètre val la valeur de la variable.

Le schéma de l'opération écriture est le suivant :

```
ecrire_var (su,vn) =
      PRE
         su ∈ SUJET
         ∧    vn opens exp
         ∧    su ∈ ecritureautorise(fvar)
      THEN
         var := vn
      END
```

Figure 4.5 : service ecrire (politique discrétionnaire)

avec var ∈ VAR, fvar ∈ OBJET , opens est l'opérateur ensembliste (∈, ⊂ ou ⊆) utilisé pour typer la variable dans l'invariant de la machine APPLICATION2, exp est l'expression utilisée dans le prédicat ensembliste de typage.
La précondition assure que le sujet doit appartenir à l'ensemble des sujets habilités à écrire la variable en cours. La valeur nouvelle, appelé vn, qui remplace la valeur de la variable doit vérifier l'invariant de typage de cette dernière.

4.4.2.3 Modélisation d'une politique obligatoire

Les politiques multi-niveaux sont basées sur la classification des sujets et des objets (Cf. : section 2.4.2). Il s'agit de définir un ensemble d'éléments ordonnés hiérarchiquement sur lequel une relation d'ordre partiel (>) est définie. Nous désignons cet ensemble par NIVEAU. Pour matérialiser la notion d'ordre, nous attribuons un entier à chaque élément de NIVEAU. L'affectation des entiers aux niveaux s'effectue via la relation ORDRE de la façon suivante :

$$\forall n, n'. (n, n' \in NIVEAU \times NIVEAU \wedge n > n' \Rightarrow ordre (n) > ordre (n'))$$

L'ensemble SUJET et OBJET sont décrits de la même manière que celle présentée dans la machine politique discrétionnaire.
Chaque sujet est affecté à un niveau via la fonction totale habilitation, chaque objet est affecté à un niveau via la fonction totale classification.

Nous présentons ci-dessous le squelette de cette machine :

MACHINE POLITIQUE
SETS
SUJET = {Liste_ident } ; OBJET= {Liste_ident } ; NIVEAU= {Liste_ident }
CONSTANTS
ordre, habilitation , classification
PROPERTIES
ordre ∈ NIVEAU >→ N
∧ habilitation ∈ SUJET→ NIVEAU
∧ classification ∈ OBJET→ NIVEAU
∧Pred_eg_Prop1
∧ Pred_eg_Prop2
∧ Pred_eg_Prop3
END

Figure 4.6: le schéma de la machine POLITIQUE (politique obligatoire)

Ordre est une fonction injective totale (notation $>\!\!\rightarrow$) de l'ensemble NIVEAU vers l'ensemble des entiers naturels N. Habilitation est une fonction totale de l'ensemble SUJET vers l'ensemble NIVEAU. Classification est une fonction totale de l'ensemble OBJET vers l'ensemble NIVEAU.

Ci-dessous, nous présentons la grammaire pour expliquer les termes de cette machine :

Pred_eg_Prop1::= *ordre* = { Liste_couple }

Pred_eg_Prop2::= *habilitation* = { Liste_couple }

Pred_eg_Prop3::= *classification* = { Liste_couple }

Liste_couple ::= Liste_couple , Couple | Couple | ε

Couple ::= (ident $\mapsto$ ident)

Liste_ident ::= Liste_ident , ident | ident | ε

Figure 4.7 : grammaire

4.4.2.4 Les services de lecture et d'écriture

Dans le cadre des politiques obligatoires et notamment les politiques multi-niveaux, les restrictions varient selon l'objectif visé : préserver la confidentialité ou l'intégrité. Cependant, la décision est toujours liée à la comparaison entre les niveaux des sujets et des objets.

Dans notre modélisation, ceci s'effectue grâce à la relation ordre qui attribue un entier à chaque niveau. Nous expliquons ci-dessous les contraintes qui doivent être respectées dans l'élaboration de l'ensemble des sujets autorisés pour la lecture ou l'écriture de chaque objet. Aussi, présentons-nous le schéma des opérations de lecture et d'écriture.

- Confidentialité

Le schéma de l'opération de lecture se présente ainsi :

```
val ← lire_ var (su) =
  PRE
    su ∈ SUJET
    ∧ (ordre(habilitation(su)) > (ordre(classification(fvar))

   THEN
     val := var
   END
```

Figure 4.8 : service lire (politique obligatoire confidentialité)

avec var ∈ VAR ∧ fvar ∈ OBJET.

Lors d'une opération de lecture, le niveau du sujet doit être supérieur à celui de l'objet (cf. : modèle Bell-Lapadula, section 2.4.2.2.1). Ceci se traduit formellement, dans notre modèle, par la contrainte suivante :

(ordre (habilitation(su)) > (ordre (classification(fvar)).

Pour l'opération écrire, le schéma est le suivant :

```
ecrire_var (su,vn) =
        PRE
            su : SUJET
             ∧ vn opens exp
             ∧ (ordre ( habilitation( su) ) < (ordre ( classification( fvar) )
          THEN
             var := vn
           END
```

Figure 4.9 : service ecrire (politique obligatoire confidentialité)

avec var ∈ VAR, fvar ∈ OBJET, opens est l'opérateur ensembliste (∈, ⊂ ou ⊆) utilisé pour typer la variable dans l'invariant de la machine APPLICATION2, exp est l'expression utilisée dans le prédicat ensembliste de typage.

Lors d'une opération d'écriture, le niveau du sujet doit être inférieur à celui de l'objet, ce qui peut être représenté formellement par la contrainte suivante : (ordre (habilitation(su)) < (ordre (classification(fvar)).

- Intégrité

Pour préserver l'intégrité, le dual des restrictions, concernant la confidentialité, est requis (cf. : modèle Biba, section 2.4.2.3).

```
val ← lire_ var (su) =
  PRE
    su ∈ SUJET
    ∧ (ordre(habilitation(su)) < (ordre(classification(fvar))

   THEN
     val := var
   END
```

Figure 4.10 : service lire (politique obligatoire intégrité)

avec var ∈ VAR ∧ fvar ∈ OBJET.

Pour l'opération d'écriture, sa structure est la suivante :

```
ecrire_var (su,vn) =
        PRE
          su : SUJET
           ∧ vn opens exp
           ∧ (ordre ( habilitation( su) ) > (ordre ( classification( fvar) )
          THEN
             var := vn
          END
```

Figure 4.11 : service ecrire (politique obligatoire intégrité)

Avec var ∈ VAR, fvar ∈ OBJET opens est l'opérateur ensembliste (∈, ⊂ ou ⊆) utilisé pour typer la variable dans l'invariant de la machine APPLICATION2, exp est l'expression utilisée dans le prédicat ensembliste de typage.

4.4.2.5 Modélisation d'une politique basée sur les rôles

Dans le cadre des politiques basées sur les rôles, les sujets sont affectés à des rôles et les rôles à des permissions.

Les permissions explicitent les opérations qui peuvent être exercées sur les objets (Cf. : section 2.4.3). Dans le cadre de notre étude, les opérations sont celles de lecture et d'écriture. Nous signalons qu'un sujet peut être affecté à plusieurs rôles. Cependant à un moment donné il a le droit d'exercer un seul rôle parmi ceux qu'il détient (notion du rôle actif, section 2.4.3). Cette restriction n'est pas figée, des variantes ultérieures du modèle RBAC permettent à l'utilisateur d'exécuter plusieurs rôles simultanément. Des restrictions dynamiques sont cependant imposées sur le nombre et la nature des rôles activés simultanément (cf. . : section 2.4.3.2, les contraintes dans RBAC). De tels cas seront étudiés plus tard, dans des extensions prochaines du travail où nous envisageons de traiter l'intervention des utilisateurs. Les opérations ne sont pas aptes à choisir le rôle à activer.

La hiérarchie des rôles est modélisée par une relation binaire entre les rôles. Soit r1 et r2 deux rôles, le couple (r1,r2) ∈ hierarchie signifie : r2 est le père de r1 et que r1 hérite de r2. Ceci peut être défini aussi de la manière suivante en fonction des permissions et des sujets :
∀ r1, r2 .(r1, r2 ∈ ROLE × ROLE ∧ (r1,r2) ∈ hierarchie ⇒ sujet-role^{-1} ({r2}) ⊆ sujet-role^{-1} ({r1}) ∧ role-permission({r2}) ⊆ role-permission({r1})).

La relation binaire roles-conflictuels reflète le conflit statique entre les rôles. Soit r1 et r2 deux rôles, le couple (r1,r2) ∈ roles-conflictuels signifie : r1 et r2 sont en conflit et ne peuvent pas être attribués à un même sujet. Formellement ceci se définit de la manière suivante :
∀ r1, r2 .(r1, r2 ∈ ROLE × ROLE ∧ (r1,r2) ∈ roles-conflictuels ⇒ sujet-role^{-1} {(r1)}∩ sujet-role^{-1} {(r2)}=∅) .

Les ensembles SUJET et OBJET sont les mêmes que précédemment.

MACHINE POLITIQUE

SETS

SUJET ={Liste_ident } ; OBJET={Liste_ident } ; ROLE ={Liste_ident } ;
OPERATION= {lecture, ecriture}

CONSTANTS

permission, sujet-role, role-permission, hierarchie , roles-conflictuels

PROPERTIES

permission $\in$ OPERATION $\rightarrow$ OBJET

$\wedge$ sujet-role $\in$ SUJET $\rightarrow$ F(ROLE)

$\wedge$ role-permission $\in$ ROLE $\rightarrow$ (OPERATION $\rightarrow$ OBJET)

$\wedge$ hierarchie $\in$ ROLE $\leftrightarrow$ ROLE

$\wedge$ roles-conflictuels $\in$ ROLE $\leftrightarrow$ ROLE

$\wedge$ Pred_eg_Prop1

$\wedge$ Pred_eg_Prop2

$\wedge$ Pred_eg_Prop3

$\wedge$ Pred_eg_Prop4

$\wedge$ Pred_eg_Prop5

END

Figure 4.12: le schéma de la machine POLITIQUE (politique basée sur les rôles)

Permission est une fonction totale entre l'ensemble des opérations et des objets. Soit op une opération et o un objet, (op, o) $\in$ Permission signifie : l'opération op peut être exécutée sur l'objet o.

La fonction totale sujet-role est définie entre l'ensemble des sujets et les sous ensembles finis de l'ensemble ROLE. Elle associe à un sujet l'ensemble des rôles qu'il détient.

La fonction totale role-permission matérialise les permissions accordées à chaque rôle.

Ci-après, nous présentons la grammaire pour expliquer les termes de cette machine :

Pred_eg_Prop1::= *permission* = { Liste_couple1 }

Pred_eg_Prop2::= *sujet-role* = { Liste_couple2 }

Pred_eg_Prop3::= role-permission = { Liste_couple3}

Pred_eg_Prop4::= hierarchie = { Liste_couple1 }

Pred_eg_Prop5::= roles-conflictuels = { Liste_couple1 }

Liste_couple1 ::= Liste_couple1 , Couple1 | Couple1 | ε

Couple1 ::= (ident ↦ ident)

Liste_couple2 ::= Liste_couple2 , Couple2 | Couple2 | ε

Couple2 ::= (ident ↦ { Liste_ident })

Liste_couple3 ::= Liste_couple3 , Couple3 | Couple3 | ε

Couple3 ::= (ident ↦ { Liste_couple1})

Liste_ident ::= Liste_ident , ident | ident | ε

Figure 4.13 : grammaire

4.4.2.6 Les services de lecture et d'écriture

Dans le cadre des politiques basées sur les rôles (cf. section 2.4.3), le sujet a le droit de lire l'objet si cette permission est accordée au rôle qu'il détient. Nous présentons le schéma de l'opération de lecture :

```
val ← lire_ var (su) =
   PRE
     su ∈ SUJET
     ∧ ∃ r . ( r ∈ sujet-role (su)
     ∧ ( lecture ↦ fvar) ∈ role-permission ( r)
     ∧ ( lecture ↦ fvar) ∈ permission)
   THEN
     val := var
   END
```

Figure 4.14 : service lire (politique basée sur les rôles)

avec var ∈ VAR ∧ fvar ∈ OBJET.

Le schéma de l'opération écrire est le suivant :

```
ecrire_var (su,vn) =
    PRE
      su : SUJET
      ∧ vn opens exp
      ∧ ∃ r . ( r ∈ sujet-role (su)
       ∧ ( ecriture ↦ fvar) ∈ role-permission ( r)
       ∧ ( ecriture ↦ fvar) ∈ permission)
      THEN
        var := vn
      END
```

Figure 4.15 : service ecrire (politique basée sur les rôles)

avec var ∈ VAR, fvar ∈ OBJET et var opens exp est le prédicat ensembliste de typage de la variable dans l'invariant de la machine APPLICATION2.

Ainsi, à travers la modélisation que nous avons choisi pour représenter les différentes politiques de sécurité, le noyau de sécurité des services de lecture et d'écriture a été élaboré.

4.4.3 La machine APPLICATION2_REF

Cette machine est le raffinement sécurisé de l'application dans laquelle toute tentative d'accès direct à une variable est remplacée par un appel aux services de lecture et d'écriture sécurisés. La méthode B garantit, via les obligations de preuve, que les appels à ces services de lecture et d'écriture vérifient les préconditions précisées (cf. : section 3.7 du chapitre 3).

Cette machine raffine la machine APPLICATION2 et réalise une inclusion de la machine CONTROLE_ACCES pour pouvoir utiliser les services sécurisés qui y sont définis. Comme son abstraction, cette machine voit (lien SEES) les deux machines CONSTANTES et POLITIQUE. Elle contient aussi la clause OPERATIONS. Cette dernière définit les mêmes opérations du niveau d'abstraction supérieur, cependant toute lecture et écriture d'une variable est remplacée par un appel aux services sécurisés.

En supposant que var est une variable à protéger (var ∈ VAR ∧ fvar ∈ OBJET), alors toute tentative de lecture de récupération de la valeur de la variable par un sujet et notamment par une opération op telle que op ∈ OPE ∧ fop ∈ SUJET génère le bloc suivant :

```
VAR nouvelle_var IN
nouvelle_var <-- lire_var(fop) ;
END
```

Dans le prédicat ou l'expression en cours nouvelle_var sera utilisée à la place de var.

Signalons que nous sommes en train de concevoir l'algorithme qui permet choisir le lieu convenable du mot clé END dans le cas des substitutions imbriquées.

Toute tentative d'assignation d'une valeur à une variable dans une substitution d'affectation (devient égal) génère la substitution suivante :
ecrire_var (fop, pdr) ; où pdr désigne l'expression apparaissant en partie droite de l'opérateur d'affectation (devient égal).

4.4.4 Vérification du respect de la politique imposée

Les obligations de preuve générées dans le raffinement sécurisé de l'application nous permettent de raisonner sur la sécurité de l'application. Les obligations de preuve qui se rapportant aux appels des services vérifient, si elles sont prouvées, le respect des préconditions. Un exemple d'une obligation de preuve d'appels d'opérations est donné dans la section suivante concernant l'étude de cas.

4.5 Etudes de cas

Pour illustrer le fonctionnement de notre outil, nous allons utiliser une étude de cas se rapportant au domaine bancaire. Il s'agit de la modélisation d'une carte bancaire où une politique discrétionnaire est mise en œuvre. Les machines B correspondantes se trouvent à l'annexe 2.

4.5.1 Exemples Politiques discrétionnaires

4.5.1.1 Introduction et présentation des cartes à puce

La carte bancaire que nous étudions fait partie de la famille des cartes à puce. Nous donnons d'abord un aperçu sur les différentes phases du cycle de vie d'une carte à puce **[BD00] [ADAE03]**:

- Phase amont : cette phase est caractérisée par le développement du système d'exploitation, la spécification des informations nécessaires à la pré personnalisation, la conception de la puce et son database construction.
- Phase de création : dans cette phase se déroulent la fabrication matérielle de la puce et l'inscription en mémoire ROM du programme matérialisant le système d'exploitation. Celui-ci est doté des fonctionnalités suivantes : gestion des entrées sorties, réponse au reset. L'initialisation de la carte à puce s'effectue aussi dans cette phase. Elle se caractérise par l'inscription en mémoire des données spécifiques propres à l'application dans laquelle la carte va s'insérer. Dans notre cas d'étude « carte bancaire » ce sont les codes secrets du gérant de la banque, du porteur de la carte, les nombres des essais erronés…Ces données sont protégées dans un mode de fonctionnement spécifique lecture, écriture et lecture/ écriture. Dans le cadre de notre travail, nous nous intéressons à la protection de ces données.
- Phase de circulation : cette phase est subdivisé en deux étapes : la personnalisation et l'utilisation. L'étape de la personnalisation est dédiée à l'adaptation de la carte à son porteur final, elle comporte deux aspects. L'aspect électronique où s'effectue l'écriture du nom du porteur et de son numéro de compte bancaire par exemple, ainsi que de toute information pertinente. L'aspect graphique caractérisé par l'impression graphique de tout logo, signe permettant une identification visuelle rapide de la carte.

La phase d'utilisation débute depuis l'entrée de la carte en possession du porteur, les données sont accessibles selon les règles définies dans les étapes précédentes. Lorsque la carte atteint sa date limite d'utilisation ou lorsqu'elle est brisée ou cassée, son système d'exploitation arrête de fonctionner et son cycle de vie s'achève ainsi.

Nous allons maintenant décrire le cahier de charge que nous avons modélisé. Ce cahier de charge traite les fonctionnements du système d'exploitation de la carte. Nous donnons ci-dessous la définition de quelques termes qui y sont utilisés :

- La session : Une session est synonyme de durée de connexion, elle représente le dialogue entre la carte et les divers acteurs, l'utilisateur et le gérant de la banque. Lorsqu'une carte est insérée dans un lecteur, elle se trouve alimentée d'énergie et est réinitialisée. Elle reçoit des commandes sur le port d'entrée/sortie, les exécute et envoie des réponses. Elle est caractérisée par un nombre d'opérations qui peuvent être jouées, selon le mode de fonctionnement de la carte. La session s'achève lorsque la carte est arrachée du lecteur, à la fin d'une session ou par un comportement hostile. Nous ne traitons pas ce type de comportement dans notre spécification. Toute session débute par une opération effectuant cette demande. C'est l'opération « beginSession ». Elle s'achève par l'envoi d'une demande de terminaison (endSession).
- Le mode : Durant son cycle de vie, une carte passe par différents mode de fonctionnement. la variable représentant le mode peut prendre différentes valeurs : load, invalid ou use. Chaque valeur décrit un jeu d'opérations permises. Dans le mode load, la carte est en cours de chargement et est en possession de la banque émettrice, l'inscription du code secret du gérant de la banque est donc réalisée dans ce mode de fonctionnement. Dans le mode invalid, le gérant de la banque authentifié par son code secret enregistre le code secret du porteur de la carte. Quand la carte passe dans la possession de son porteur, elle est en mode use.

Le cahier des charges est présenté à travers des variables, des opérations et des propriétés précisant les accès autorisés, nous le décrivons ci-dessous :

- Les variables :
  - status : état dans lequel s'est terminée la session précédente.
  - tryleft : un nombre entier matérialisant le nombre d'essais restants avant invalidation de la carte. La carte passe en mode invalid lorsque la variable tryleft vaut 0.
  - hpc : (Holder Pin Code), un entier représentant le code secret du porteur de la carte à puce.
  - bpc : (Bank Pin Code), un entier représentant le code secret du gérant de la banque.
  - mode : mode de fonctionnement de la carte
- Les opérations :
  - beginSession() : vérifie que la zone mémoire M ne contient que des 0 et que la session précédente s'est correctement terminée (voir variable status). Dans le cas contraire, la carte passe en mode Invalid.
  - reset() : réinitialise à 3 la valeur du compteur d'essai et fait passer la carte en mode use.

- setBPC(pin : int): enregistre la valeur du paramètre pin dans la variable bpc.
- checkPin(pin : int) : compare le paramètre pin à la valeur de la variable hpc et décrémente le compteur d'essai en cas d'erreur.
- setHPC(pin : int) : enregistre la valeur du paramètre pin dans la variable hpc. Cette fonctionnalité est réservée au gérant de la banque et ne peut être appelée qu'après vérification du code secret du gérant.
- authBank(pin : int) : compare le paramètre pin à la valeur de la variable bpc.
- endSession() : met toutes les variables de la zone M à zéro et donne une valeur à la variable status qui indique que la session s'est terminée correctement.
- crédit() : demande au terminal d'afficher à écran le solde du compte bancaire du porteur de la carte.

- Les accès autorisés aux variables sont décrits par le tableau suivant :

Accès	tryleft	hpc	bpc	status	mode
lecture	checkPin	checkPin	authBank	beginSession	∀ fonction
écriture	checkPin,reset	setHPC	setBPC	endSession	∀ fonction

Figure 4.16 : politique de sécurité

4.5.1.2 Modélisation et vérification

Nous présentons ci-dessous la modélisation de l'opération checkPin, nous mentrons comment elle sera transformée dans le raffinement sécurisé de l'application et nous montrons comment les obligations de preuve générées permettent de raisonner sur la sécurité de l'application.

```
        val <-- checkPin(pin) =
        PRE
          pin : INT
          & pin : HPC
          & mode=use

        THEN
           IF pin/=hpc & tryleft/=0
           THEN
             val:=KO_PIN_INCORRECT
             || tryleft:=tryleft-1
             || mode:=mode
           ELSE
             IF pin/=hpc & tryleft=0

             THEN

              val:=KO_PIN_INCORROCET_NOTRY
             || mode:=invalid
             || tryleft:=tryleft

           ELSE
           IF pin=hpc

            THEN

             val:=OK_PIN_CORRECT
           || mode:=mode
           || tryleft:=tryleft
          END
         END
       END

END
```

Figure 4.17 : opération checkPin dans la machine APPLICATION

Dans cette opération, il s'agit de comparer le paramètre pin au code secret du porteur de la carte. La précondition stipule que le paramètre d'entrée pin appartient à l'ensemble des entiers et à l'ensemble des codes des porteurs des cartes. C'est le typage du paramètre d'entrée. Cette opération ne pouvant être appelée que lorsque la carte est dans la phase d'utilisation par son porteur, la précondition stipule que mode est égal à use. Checkpin retourne un paramètre de sortie « val » qui prend une valeur parmi celles de l'ensemble suivant :{ KO_PIN_INCORRECT, KO_PIN_INCORROCET_NOTRY, val:=OK_PIN_CORRECT}. Le corps de l'opération effectue la tâche suivante :

Dans le cas où le pin ne correspond pas et le nombre d'essais restant est non nul, tryleft est diminué de 1 et l'opération retourne KO_PIN_INCORRECT pour déclarer la faute.

Lorsque le pin ne correspond pas et qu'il ne reste pas d'essais à effectuer, l'opération retourne la valeur KO_PIN_INCORROCET_NOTRY et la carte passe en mode invalid.

Enfin, quand le pin est correct, l'opération retourne OK_PIN_CORRECT, la valeur des variables mode et tryleft n'étant pas modifiée.

Compte tenu de son fonctionnement, cette opération a le droit de lire et écrire tryleft, de lire hpc et de modifier mode. Elle possède aussi, comme toutes les opérations, le droit de lire mode. Ces autorisations sont précisés dans la machine POLITIQUE au cœur des deux sous-matrices lecture-autorisee et ecriture-autorisee .

```
lectureautorisee={(fstatus|->{fbeginSession}), (ftryleft|->{fcheckPin}), (fhpc|->{fcheckPin}),(fbpc|->{fauthBank}), (fmode|->{fbeginSession,freset,fsetBPC,fcheckPin,fsetHPC,fauthBank,fendSession,fcredit})}
&
ecritureautorisee={(fstatus|->{fendSession}), (ftryleft|->{fcheckPin,freset}),(fhpc|->{fsetHPC}),(fbpc|->{fsetBPC}), (fmode|->{fbeginSession,freset,fsetBPC,fcheckPin,fsetHPC,fauthBank,fendSession,fcredit})}
```

Figure 4.18 : les sous-matrices lectureautorisee et ecritureautorisee de la machine POLITIQUE

Les services générés : lire-tryleft, ecrire-tryleft, lire-hpc, lire-mode, ecrire-mode dans le noyau de sécurité stipulent que les sujets le sujet, effectuant la lecture, appartient à l'ensemble des sujets habilités pour cet affaire. L'opération checkPin figure dans ces ensembles concaténée à la lettre f que nous avons choisi pour éviter tout conflit d'identificateur. Nous nous limitons ci-dessous à présenter seulement les deux opérations lire_tryleft et ecrire_tryleft.

```
val <-- lire_tryleft(su) =
    PRE
        su : SUJET &
        su : {fcheckPin}
    THEN
        val := tryleft
    END
```

Figure 4.19: le service sécurisé lire_tryleft

```
ecrire_tryleft(su,vn) =
      PRE
         su : SUJET &
         vn : TRYLEFT &
         su :{fcheckPin, freset}
      THEN
         tryleft := vn
      END
```

Figure 4.20: le service sécurisé ecrire_tryleft

Nous présentons ci-dessous le raffinement sécurisé de l'opération où tous les accès directs en lecture ou en modification sont remplacés par les appels des services sécurisées.

```
val <-- checkPin(pin) =
  BEGIN
     VAR hp,tr
     IN
     hp<--lire_hpc(fcheckPin);
     tr<--lire_tryleft(fcheckPin);
     IF pin/=hp & tr/=0
     THEN
       val:=KO_PIN_INCORRECT;
       ecrire_tryleft(fcheckPin,tr-1);
       VAR mo
       IN
       mo<--lire_mode(fcheckPin);
       ecrire_mode(fcheckPin,mo)
       END
     ELSE
       IF pin/=hp & tr=0
       THEN
        val:=KO_PIN_INCORROCET_NOTRY;
        ecrire_mode(fcheckPin,invalid);
        ecrire_tryleft(fcheckPin,tr)
       ELSE
        val:=OK_PIN_CORRECT;
        VAR mo
        IN
        mo<--lire_mode(fcheckPin);

        ecrire_mode(fcheckPin,mo);
        ecrire_tryleft(fcheckPin,tr)
       END
```

Figure 4.21 : raffinement sécurisé de l'opération checkPin

L'atelier B génère 60 obligations de preuve. Le prouveur de l'atelier B a démontré automatiquement la totalité de ces obligations dont 48 sont évidentes (obvious) et ne nécessitent aucun processus de démonstration.

Nous présentons ci-dessous une obligation de preuve qui se rapporte à l'appel d'opération suivant : ecrire_tryleft(fcheckPin,tr-1);

Dans les obligations de preuve générées, il s'agit de vérifier que les préconditions de l'opération ecrire_tryleft sont bien respectées. Les buts à démontrer sont que tr-1 appartient à TRYLEFT, que fcheckPin appartient à l'ensemble des sujets ({fbeginSession,freset,fsetBPC,fcheckPin,fsetHPC,fauthBank,fendSession,fcredit}) et que fcheckPin appartient à l'ensemble des sujets autorisés pour l'écriture ({fcheckPin, freset}).

Nous présentons ci-dessous une version simplifiée des trois obligations de preuve générées. Nous présentons seulement les hypothèses locales issues de la condition de l'instruction IF correspondante (pin ≠ hpc et tryleft ≠ 0) et les hypothèses nécessaires pour la preuve du but.

Le symbole $1 désigne la valeur courante de la variable.

- Première obligation de preuve

  Propriétés vues

  TRYLEFT= 0..3

  Hypothèses locales

  $\wedge \neg$ (pin=hpc$1) condition du IF

  $\wedge \neg$ (tryleft$1=0)

  Vérification de la précondition de l'opération appelée

  $\Rightarrow$ tryleft$1 – 1 $\in$ TRYLEFT

- Deuxième obligation de preuve

  Hypothèses locales

  $\wedge \neg$ (pin=hpc$1) condition du IF

  $\wedge \neg$ (tryleft$1=0)

  Vérification de la précondition de l'opération appelée

  $\Rightarrow$ fcheckPin $\in$ {fcheckPin, freset}

- Troisième obligation de preuve

Hypothèses locales

$\wedge \neg$ (pin=hpc$1) condition du IF

$\wedge \neg$ (tryleft$1=0)

Vérification de la précondition de l'opération appelée

$\Rightarrow$ fcheckPin $\in$

{fbeginSession,freset,fsetBPC,fcheckPin,fsetHPC,fauthBank,fendSession, fcredit}

Dans l'annexe 2, toutes les machines B sont présentes dans l'ordre suivant :

- La machine APPLICATION : Dans cette machine, le cahier des charge est modélisé.
- La machine POLITIQUE : la politique discrétionnaire est spécifiée, les deux sous-matrices sont instanciées par les sujets et les objets correspondants.
- La machine CONSTANTES regroupant les constantes de l'application, elle reformule avec la machine APPLICATION2, la machine principale de l'application.
- La machine APPLICATION2
- La machine CONTROLE_ACCES : Dans cette machine, on peut bien voir comment les opérations de lecture et d'écriture sont construites.
- Le raffinement sécurisé de l'application qui met en exergue la façon par laquelle les accès directs aux variables sont remplacés par les appels d'opérations du noyau de sécurité.

5 Conclusion et perspectives

5.1 Conclusion

Ce travail nous a permis de nous pencher sur un sujet important concernant les systèmes d'information. De nombreux incidents réels dans des domaines critiques exhibent la nécessité de construire des applications sûres. Des normes nationales préconisent l'utilisation des modèles formels de sécurité.

Nous nous sommes intéressés au contrôle d'accès. Nous avons étudié des modèles proposés dans la littérature. La méthode B a constitué le cadre formel pour la modélisation de ces politiques, ses atouts ainsi que la robustesse des outils qu'elle dispose nous a offert un appui fort dans notre travail lors de la modélisation et de la vérification.

Nous avons réalisé un outil permettant la mise en œuvre de plusieurs politiques de contrôle d'accès qui effectue la vérification statique des propriétés de sécurité par une application. Notre choix s'est orienté vers une vérification statique pour réduire les coûts à l'exécution et pour découvrir les failles de sécurité dans la phase de conception et avant l'exécution.

Ce choix est matérialisé par la construction d'un noyau de sécurité, des obligations de preuves générées par l'atelier B imposent la vérification des propriétés de sécurité modélisées. A travers un exemple appartenant au domaine bancaire, nous avons montré l'importance de notre contribution.

Un apport marquant de notre travail réside dans la construction d'un outil permettant de traiter plusieurs politiques de contrôle d'accès et de raisonner sur la vérification de celles ci dans la construction des applications. Nous envisageons d'améliorer constamment notre outil pour répondre le mieux aux exigences de sécurité.

5.2 Perspectives

Dans notre travail, nous avons développé un outil en Java pour vérifier le respect des politiques de sécurité par les applications. Nous avons utilisé des fonctions implémentées dans la boîte à outils B. Ces fonctions nous ont permis d'analyser les machines B d'entrée et leurs clauses et de construire les machines de sortie.

L'outil dans son état actuel (1226 lignes Java pour la construction du noyau de sécurité) permet la construction du noyau de sécurité pour traiter différents genres de politiques. La tâche d'élaboration du raffinement sécurisé est en cours de développement. Nous sommes en train de travailler la génération du bloc pour l'appel du service de lecture (cf. : section 4.4.3).

Nous envisageons d'améliorer les fonctionnalités de l'outil pour traiter le cas où l'invariant de l'application contient des prédicats d'égalité pour le typage des variables. Nous comptons aussi traiter les invariants qui contiennent des propriétés et des prédicats de typage.

Dans le but de renforcer la sécurité, nous envisageons de vérifier que les données entrées par l'utilisateur, dans la machine de modélisation des politiques, satisfont les contraintes imposées dans la grammaire (cf. section 4.4.2). Ceci peut s'effectuer en construisant une machine offrant les services de désignation des diverses entités du modèle. Son invariant matérialise ces contraintes. Les obligations de preuve générées par l'Atelier B et qui se

rapporte à la préservation de l'invariant à l'initialisation et dans le corps de l'opération permet d'assurer une telle vérification.

Aussi, nous pensons étendre notre travail pour prendre en considération l'intervention des utilisateurs. Nous employons le terme utilisateurs pour désigner les êtres humains se connectant au système et génèrant des composants, à leur compte, qui appellent les services (opérations) de l'application. Les utilisateurs utilisent les services de l'application pour l'intégrer dans des développements extensionnels du système en traitement. Chaque utilisateur est désigné par un identificateur unique.

Pour que ces appels s'effectuent dans un cadre sécurisé, nous envisageons de mettre en œuvre une politique de sécurité sur les utilisateurs. Le noyau de sécurité exige simultanément deux contraintes pour la modification ou la lecture d'une variables : la première stipule que l'opération ait le droit spécifié, la deuxième que l'utilisateur ait ce droit.

Signalons que les identificateurs des utilisateurs sont récupérés à leur connexion lors de l'ouverture d'une session. Une fois l'utilisateur déconnecté par fermeture de session, le premier raffinement de l'application est remplacé par un autre qui transmet un identificateur n'ayant aucun droit lors des appels de services sécurisés de lecture et d'écriture.

Cette extention permet d'élargir le périmètre de sécurité de l'application, et de prendre en considération des entités externes.

Enfin, nous mentionnons que la repoussée des recherches dans la piste de notre travail pourra faire l'objet de futurs travaux motivants, originals et nécessaires pour répondre aux besoins actuels.

Annexe 1 : Symboles en B

Comme déjà énoncé, la méthode B utilise des notations ensemblistes pour spécifier les états, les invariants, les conditions …Cependant B introduit des symboles spécifiques pour représenter ces notations. Nous présentons ci-dessous des symboles que nous avons utilisés dans notre outil :

Symbole	Signification	Syntaxe	Notation
ϕ	ensemble vide		{}
$\times$	produit cartésien	Ensemble×Ensemble	*
$\mapsto$	maplet, doublet de valeur	élément $\mapsto$ élément	|->
P	ensemble des sous-ensembles	P(Ensemble)	POW
P1	ensemble des sous-ensembles vides	P1(Ensemble)	POW1
F	ensemble des sous-ensembles finis	F(Ensemble)	FIN
F1	ensemble des sous-ensembles finis non vides	F1(Ensemble)	FIN1
..	intervalle	élément .. élément	..
$\rightarrow$	fonction totale	Ensemble $\rightarrow$ Ensemble	-->
$>\!\!\rightarrow$	fonction injective totale	Ensemble $>\!\!\rightarrow$Ensemble	>->
$+\!\!\rightarrow$	fonction partielle	Ensemble $+\!\!\rightarrow$Ensemble	+->
$\in$	appartient	élément $\in$ Ensemble	:
$\subset$	inclus	Ensemble $\subset$ Ensemble	<< :
$\subseteq$	inclus ou égal	Ensemble $\subseteq$ Ensemble	< :

Le terme Ensemble désigne une expression construisant un ensemble. Le terme élément désigne un élément d'un ensemble.

Annexe 2 : Les machines B de l'application carte bancaire

1. première machine d'entrée : la spécification du cahier de charge

```
MACHINE APPLICATION

 SETS



MODE={rien,load,invalid,use};STATUS={correct,incorrect};VAL={OK,OK_PIN_CORR
ECT,KO_MEMOIRE_NO_INITIALISE,KO_SESSION_TERMINAISON,KO_PIN_INCORRECT,KO_PIN
_INCORROCET_NOTRY,
KO_SESSION_NO_TERMINATION_MEMOIRE_NO_INITIALISE,OK_PINBANK_CORRECT,KO_PINBA
NK_INCORRECT, OK_END_SESSION,OK_SOLDE_AFFICHE}

 CONSTANTS

   HPC,BPC,TRYLEFT,binf,bsup,maxcredit,maxdebit,mindebit,mincredit

 PROPERTIES


 HPC=1000..9999
& BPC=1000..9999
& TRYLEFT=0..3
& binf : INT
& bsup : INT
& binf<bsup
& maxcredit:INT
& mincredit:INT
& maxdebit:INT
& mindebit:INT
& mindebit<maxdebit
& mincredit<maxcredit




 ABSTRACT_VARIABLES
   status,tryleft,mode,hpc,bpc,zonememoire,credit,debit

 INVARIANT
 status : STATUS & mode : MODE & tryleft :TRYLEFT  & hpc : HPC
 & bpc : BPC &  zonememoire : binf..bsup +-> NATURAL & credit:
mincredit..maxcredit
& debit : mindebit..maxdebit

 INITIALISATION

  hpc :=1000|| bpc :=1000
 ||status,tryleft,mode:=correct,3,rien
 ||zonememoire :=( binf.. bsup)*{0}
 ||credit:=maxcredit
 ||debit:=mindebit

 OPERATIONS
```

```
    val <--beginSession =
       IF
           status= correct

           & ran(zonememoire)={0}
       THEN
           ANY mm WHERE
               mm : {load,use}
           THEN
               mode:=mm
               ||val:=OK
           END

      ELSE

          IF
             status= incorrect

             & ran(zonememoire)={0}
          THEN
             mode:=invalid
             ||val:=KO_SESSION_TERMINAISON
           ELSE
               IF
                 status= incorrect

                 & ran(zonememoire)/={0}
              THEN
                 mode:=invalid
                 ||val:=KO_SESSION_NO_TERMINATION_MEMOIRE_NO_INITIALISE

              ELSE
                  IF
                   status= correct

                   & ran(zonememoire)/={0}
                  THEN
                   mode:=invalid
                   ||val:=KO_MEMOIRE_NO_INITIALISE
                  END
             END
          END
      END


  ;

reset=
       PRE
          mode=invalid
       THEN
           tryleft:=3
           ||mode := use
       END
  ;
  setBPC(pin) =
       PRE
           pin : INT & pin : BPC & mode=load
       THEN
           bpc := pin
       END
```

```
   ;
val <-- checkPin(pin) =
       PRE
           pin : INT
          & pin : HPC
           & mode=use

       THEN
           IF pin/=hpc & tryleft/=0
           THEN
             val:=KO_PIN_INCORRECT
             ||tryleft:=tryleft-1
             ||mode:=mode
           ELSE
              IF pin/=hpc & tryleft=0

              THEN

               val:=KO_PIN_INCORROCET_NOTRY
               ||mode:=invalid
               ||tryleft:=tryleft

              ELSE
                IF pin=hpc

                THEN

                 val:=OK_PIN_CORRECT
                 ||mode:=mode
                 ||tryleft:=tryleft
                END
              END
           END

        END
   ;
 setHPC(pin) =

       PRE
           pin : INT & pin : HPC & mode=invalid
       THEN
           hpc := pin
       END
   ;
 val <-- authBank(pin) =

       PRE
           pin : INT & mode=invalid
       THEN
           IF pin=bpc
           THEN
             val:=OK_PINBANK_CORRECT
           ELSE
             val:=KO_PINBANK_INCORRECT
           END
       END
   ;

val <--endSession =

       BEGIN
```

```
         status:=correct

         ||val:=OK_END_SESSION
        END
    ;
val,solde <--cerdit=
   PRE
    mode=use
   THEN
    solde:=credit-debit
    ||val:=OK_SOLDE_AFFICHE

   END

END
```

2. deuxième machine d'entrée: la politique discrétionnaire mise en oeuvre

```
MACHINE POLITIQUE

SETS


SUJET={fbeginSession,freset,fsetBPC,fcheckPin,fsetHPC,fauthBank,fendSession
,fcredit};
    OBJET={ftryleft,fhpc,fbpc,fstatus,fmode}


CONSTANTS

  lectureautorise,ecritureautorise

PROPERTIES

lectureautorise={(fstatus|->{fbeginSession}),(ftryleft|-
>{fcheckPin}),(fhpc|->{fcheckPin}),(fbpc|->{fauthBank}),(fmode|-
>{fbeginSession,freset,fsetBPC,fcheckPin,fsetHPC,fauthBank,fendSession,fcre
dit})}
&
ecritureautorise={(fstatus|->{fendSession}),(ftryleft|-
>{fcheckPin,freset}),(fhpc|->{fsetHPC}),(fbpc|->{fsetBPC}),(fmode|-
>{fbeginSession,freset,fsetBPC,fcheckPin,fsetHPC,fauthBank,fendSession,fcre
dit})}



END
```

3. première machine de sortie: les constantes

```
MACHINE CONSTANTES

 SETS



MODE={rien,load,invalid,use};STATUS={correct,incorrect};VAL={OK,OK_PIN_CORR
ECT,KO_MEMOIRE_NO_INITIALISE,KO_SESSION_TERMINAISON,KO_PIN_INCORRECT,KO_PIN
```

```
_INCORROCET_NOTRY,
KO_SESSION_NO_TERMINATION_MEMOIRE_NO_INITIALISE,OK_PINBANK_CORRECT,KO_PINBA
NK_INCORRECT, OK_END_SESSION,OK_SOLDE_AFFICHE}

 CONSTANTS

   HPC,BPC,TRYLEFT,binf,bsup,maxcredit,maxdebit,mindebit,mincredit

 PROPERTIES


 HPC=1000..9999
& BPC=1000..9999
& TRYLEFT=0..3
& binf : INT
& bsup : INT
& binf<bsup
& maxcredit:INT
& mincredit:INT
& maxdebit:INT
& mindebit:INT
& mindebit<maxdebit
& mincredit<maxcredit



END
```

4. deuxième machine de sortie : APPLICATION2

```
MACHINE APPLICATION2

 SEES CONSTANTES


 ABSTRACT_VARIABLES
   status,tryleft,mode,hpc,bpc,zonememoire,credit,debit

 INVARIANT
 status : STATUS & mode : MODE & tryleft :TRYLEFT  & hpc : HPC
 & bpc : BPC &  zonememoire : binf..bsup +> NATURAL & credit:
mincredit..maxcredit
& debit : mindebit..maxdebit

 INITIALISATION

  hpc :=1000|| bpc :=1000
 ||status,tryleft,mode:=correct,3,rien
 ||zonememoire :=( binf.. bsup) *{0}
 ||credit:=maxcredit
 ||debit:=mindebit

 OPERATIONS

     val <--beginSession =
        IF
            status= correct

            & ran(zonememoire)={0}
        THEN
            ANY mm WHERE
```

```
            mm : {load,use}
         THEN
            mode:=mm
            ||val:=OK
         END

   ELSE

       IF
          status= incorrect

          & ran(zonememoire)={0}
       THEN
          mode:=invalid
          ||val:=KO_SESSION_TERMINAISON
        ELSE
            IF
              status= incorrect

              & ran(zonememoire)/={0}
           THEN
              mode:=invalid
              ||val:=KO_SESSION_NO_TERMINATION_MEMOIRE_NO_INITIALISE

           ELSE
               IF
                status= correct

                & ran(zonememoire)/={0}
               THEN
                mode:=invalid
                ||val:=KO_MEMOIRE_NO_INITIALISE
               END
          END
       END
   END


  ;


 reset=
       PRE
          mode=invalid
       THEN
           tryleft:=3
           ||mode := use
       END
  ;
  setBPC(pin) =
       PRE
           pin : INT & pin : BPC & mode=load
       THEN
           bpc := pin
       END
  ;
val <-- checkPin(pin) =
       PRE
           pin : INT
           & pin :HPC
           & mode=use
```

```
        THEN
            IF pin/=hpc & tryleft/=0
            THEN
              val:=KO_PIN_INCORRECT
              ||tryleft:=tryleft-1
              ||mode:=mode
            ELSE
               IF pin/=hpc & tryleft=0

               THEN

                val:=KO_PIN_INCORROCET_NOTRY
                ||mode:=invalid
                ||tryleft:=tryleft

               ELSE
                 IF pin=hpc

                 THEN

                  val:=OK_PIN_CORRECT
                  ||mode:=mode
                  ||tryleft:=tryleft
                 END
               END
            END

         END
    ;
  setHPC(pin) =

        PRE
            pin : INT & pin : HPC & mode=invalid
        THEN
            hpc := pin
        END
    ;
  val <-- authBank(pin) =

        PRE
            pin : INT & mode=invalid
        THEN
            IF pin=bpc
            THEN
              val:=OK_PINBANK_CORRECT
            ELSE
              val:=KO_PINBANK_INCORRECT
            END
        END
    ;

 val <--endSession =

        BEGIN
         status:=correct
         ||val:=OK_END_SESSION
        END
    ;
val,solde <--cerdit=
   PRE
```

```
    mode=use
   THEN
    solde:=credit-debit
    ||val:=OK_SOLDE_AFFICHE

   END

END
```

5. troisième machine sortie : le noyau de sécurité

```
MACHINE CONTROLE_ACCES

SEES POLITIQUE, CONSTANTES


ABSTRACT_VARIABLES
   status,tryleft,mode,hpc,bpc,zonememoire,credit,debit

 INVARIANT
 status : STATUS & mode : MODE & tryleft :TRYLEFT  & hpc : HPC
 & bpc : BPC &  zonememoire : binf..bsup +> NATURAL & credit:
mincredit..maxcredit
& debit : mindebit..maxdebit

 INITIALISATION

  hpc :=1000|| bpc :=1000
 ||status,tryleft,mode:=correct,3,rien
 ||zonememoire :=( binf.. bsup) *{0}
 ||credit:=maxcredit
 ||debit:=mindebit

OPERATIONS

    val <-- lire_status(su) =
        PRE
            su : SUJET &

            su :{ fbeginSession}
        THEN
            val := status
        END
    ;
    ecrire_status(su,vn) =
        PRE
            su : SUJET &
            vn : STATUS &
            su :{ fendSession}
        THEN
            status := vn
        END
    ;
    val <-- lire_tryleft(su) =
        PRE
            su : SUJET &
            su : {fcheckPin}
        THEN
```

```
            val := tryleft
        END
    ;
    ecrire_tryleft(su,vn) =
        PRE
            su : SUJET &
            vn : TRYLEFT &
            su :{fcheckPin, freset}
        THEN
            tryleft := vn
        END
    ;
    val <-- lire_mode(su) =
        PRE
            su : SUJET
            & su :
{fbeginSession,freset,fsetBPC,fcheckPin,fsetHPC,fauthBank,fendSession,fcred
it}

        THEN
            val := mode
        END
    ;
    ecrire_mode(su,vn) =
        PRE
            su : SUJET &
            vn : MODE
            & su :
{fbeginSession,freset,fsetBPC,fcheckPin,fsetHPC,fauthBank,fendSession,fcred
it}

        THEN
            mode := vn
        END
    ;
    val <-- lire_hpc(su) =
        PRE
            su : SUJET &
            su :{ fcheckPin}
        THEN
            val := hpc
        END
    ;
    ecrire_hpc(su,vn) =
        PRE
            su : SUJET &
            vn : HPC &
            su :{ fsetHPC}
        THEN
            hpc := vn
        END
    ;
    val <-- lire_bpc(su) =
        PRE
            su : SUJET &
            su :{ fauthBank}
        THEN
            val := bpc
        END
    ;
    ecrire_bpc(su,vn) =
```

```
        PRE
            su : SUJET &
            vn : BPC &
            su :{ fsetBPC}
        THEN
            bpc := vn
        END
END
```

6. quatrième machine de sortie : le raffinement sécurisé de l'application

```
REFINEMENT APPLICATION2_REF
 REFINES APPLICATION2

  INCLUDES CONTROLE_ACCES
  SEES CONSTANTES,POLITIQUE



 OPERATIONS

     val <--beginSession =

     VAR st IN
     st<-- lire_status(fbeginSession);
        IF
            st= correct

            & ran(zonememoire)={0}
        THEN
            ANY mm WHERE
                mm : {load,use}
            THEN
                ecrire_mode(fbeginSession,mm);

                val:=OK
            END

       ELSE

           IF
              st= incorrect

              & ran(zonememoire)={0}
           THEN
              ecrire_mode(fbeginSession,invalid);
              val:=KO_SESSION_TERMINAISON
            ELSE
                IF
                  st= incorrect

                  & ran(zonememoire)/={0}
               THEN
                  ecrire_mode(fbeginSession,invalid);
                  val:=KO_SESSION_NO_TERMINATION_MEMOIRE_NO_INITIALISE

               ELSE
                  ecrire_mode(fbeginSession,invalid);
                  val:=KO_MEMOIRE_NO_INITIALISE
```

```
            END
          END
       END
  END

   ;

 reset=

       BEGIN
           ecrire_tryleft(freset,3);

           ecrire_mode(freset,use)

       END
   ;
   setBPC(pin) =
       BEGIN
           ecrire_bpc(fsetBPC,pin)
       END
   ;
val <-- checkPin(pin) =
       BEGIN
           VAR hp,tr
           IN
           hp<--lire_hpc(fcheckPin);
           tr<--lire_tryleft(fcheckPin);
           IF pin/=hp & tr/=0
           THEN
             val:=KO_PIN_INCORRECT;
             ecrire_tryleft(fcheckPin,tr-1);
             VAR mo
             IN
             mo<--lire_mode(fcheckPin);
             ecrire_mode(fcheckPin,mo)
             END
           ELSE
              IF pin/=hp & tr=0

              THEN

               val:=KO_PIN_INCORROCET_NOTRY;
               ecrire_mode(fcheckPin,invalid);
               ecrire_tryleft(fcheckPin,tr)


              ELSE
               val:=OK_PIN_CORRECT;
               VAR mo
               IN
               mo<--lire_mode(fcheckPin);


               ecrire_mode(fcheckPin,mo);
               ecrire_tryleft(fcheckPin,tr)
              END

             END
           END
        END
        END
```

```
    ;
  setHPC(pin) =

        BEGIN
            ecrire_hpc(fsetHPC,pin)

        END
    ;
  val <-- authBank(pin) =

        BEGIN
            VAR bp IN
            bp<--lire_bpc(fauthBank);
            IF pin=bp
            THEN
              val:=OK_PINBANK_CORRECT
            ELSE
              val:=KO_PINBANK_INCORRECT
            END
         END
        END
    ;

 val <--endSession =

        BEGIN
         ecrire_status(fendSession,correct);

         val:=OK_END_SESSION
        END


END
```

Bibliographie

[Abou03] Anas Abou El Kalam, Modèles et politiques de sécurité pour les domaines de la santé et des affaires sociales, Thèse de doctorat de l'Institut National Polytechnique de Toulouse, discipline : Informatique, soutenue le 04 décembre 2003.

[Abr96] J.-R Abrial, The B-Book - Assigning Programs to Meanings, Cambridge University Press (1996).

[Abr00] J.-R Abrial, B : 2000 et plus, École Jeunes chercheurs en programmation, March 2000.

[AD03] Anas Abou El Kalam, Yves Deswarte, Sécurité des systèmes d'information et de communication dans le domaine de la santé, Sécurité et Architecture Réseaux (SAR'03), Nancy (France), 30 juin - 4 juillet 2003.

[ADAE03] Agence pour le développement de l'administration électronique, Cartes de vie quotidienne, Journée plénière du 19 décembre 2003, panorama de la carte à puce, http://cvq07.free.fr/sources/cvq/9 .

[AEB+03] A. Abou El Kalam, R. Elbaida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, G. Trouessin, ORBAC : un modèle de contrôle d'accès basé sur les organisations, Cahiers francophones de la recherche en sécurité de l'information, Numéro II, 1er trimestre 2003, pp30-43.

[AGS83] Stanely R. Ames, Jr., Morrie Gasser: The Mitre Corporation, Roger R. Schell: DoD Computer Security Center: Security Kernel Design and Implementation: An intoduction, IEEE Computer, vol. 16, n. 7, july 1983.

[AS00] G. Ahn and R. Sandhu, Role-Based Authorization Constraints Specification, ACM Transactions on Information and System Security (TISSEC), vol. 3, n.4, novembre 2000, pp. 207-226.

[BD00] Jérôme Baumgarten, Nicolas Descamps, La carte à puce, mémoire de DESS Génie Informatique de Lille, 2000, http://phonecards.free.fr/these.htm.

[Beh00] Salimeh Behnia, test de modèles formels en B, cadre théorique et critères de couverture, thèse de doctorat de l'I.N.P.T, spécialité : Informatique et télécommunications, soutenue le 27 octobre 2000.

[Bib77] K. J. Biba, Integrity considerations for secure computer systems, Technical Report TR-3153, The Mitre Corporation, Bedford, MA, April 1997.

[BL73a] D. E. Bell and L. J. Lapadula, Secure computer systems: Mathematical foundations, Technical Report ESD-TR-73-278, vol. 1, The Mitre Corp., Bedford, MA, 1973.

[BL73b] D. E. Bell and L. J. Lapadula, Secure computer systems: A mathematical model, Technical Report ESD-TR-278, vol. 2, The Mitre Corp., Bedford, MA, 1973.

[BL74a] D. E. Bell and L. J. Lapadula, Secure computer systems: A refinement of the mathematical model, Technical Report ESD-TR-278, vol. 3, The Mitre Corp., Bedford, MA, 1974.

[BL74b] D. E. Bell and L. J. Lapadula, Secure computer systems: Mathematical foundations and model, M74-244, The Mitre Corp., Bedford, MA, 1974.

[BL75] D. E. Bell and L. J. Lapadula, Secure computer systems: Unified exposition and multics interpretation, MTR-2997, The Mitre Corp., Bedford, MA, 1975.

[Boi00] Olivier Boite, Méthode B et validation des invariants ferroviaires, Mémoire de DEA de logique et fondements de l'informatique, Université Denis Diderot (Paris 7).

[BP04] Didier Bert et Marie-Laure Potet, Spécification en B, Support de cours, Ecole des jeunes chercheurs en programmation, EJCP 2004, 24 mai – 4 juin 2004, Nantes – Le Croisic, http://www-lsr.imag.fr/users/Didier.Bert/Exams/poly-B-2004.pdf .

[CC05] Common Criteria for Information Technologie, security evaluation, Norm ISO 15408 – Version 3.0 (2005),

http://www.commoncriteriaportal.org/public/expert/index.php?menu=3

[Cup02] Frédéric Cuppens, Protection des systèmes d'informations: Menaces et solutions actuelles, BDA02, 21 Octobre 2002.

[Den76] D. E. Denning, A lattice model of secure information flow, Communications of the ACM, 19(5): 236-243, May 1976.

[Dij75] E.W. Dijkstra, Guarded commands, nondeterminacy and formal derivation of programs, In Comm. of the ACM, vol 18, p: 453-457, 1975.

[FK92] D. F. Ferraiolo and D. R. Kuhn, Role based access control, 15th National Computer Security Conference, 1992.

[FKS+01] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn and R. Chandramouli, A Proposed Standard for Role-Based Access Control, ACM Transactions on Information and System Security, vol. 4, n.3, aout 2001.

[FSG+01] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn and Ramaswamy Chandramouli, Proposed NIST Standard for Role-Based Access Control, ACM Transactions on Information and System Security, volume 4,n.3,August 2001, pages 224-274.

[Gass88] Morrie Gasser, Building a secure computer system, Van Nostrand Reinhold, New York, Library of Congress Catalog, Card number 87-27838, ISBN 0-442-23022-2, 1988.

[GD72] G. S. Graham and P. J. Denning, Protection- principles and practice, In AFIPS Press, editor, Proc. Spring Jt. Computer Conference, volume 40, pages 417-429, Montvale, N.J., 1972.

[Gir99] Pierre Girard, Which security policy for multiapplication smart cards In USENIX workshop on smartcard technology, 1999.

[GM84] J.A Goguen et J. Meseguer, Unwinding and inference control, In Proc. of the 1984 Symposium on Research in Security and Privacy, pages 75-86, 1984.

[Hab01] Henri Habrias, Spécification formelle avec B, Lavoisier 2001, 11 rue Lavoisier, 75008 Paris.

[Had 95] Stéphan Hadinger, Le contrôle d'accès au SI, projet IS@ France Télécom, www.rd.francetelecom.fr/ fr/conseil/mento20/chapitre7.pdf

[Hp]Les points essentiels de la protection de données

http://h41087.www4.hp.com/abonnement/hps/0903c.html, 1994-2003 Hewlett-Packard Company

[Hoa69] C.A.R. Hoare, An axiomatic basis for computer programming, Communications of the ACM, 12:576-583, October 1969.

[HRU76] M.H. Harrison, W. L. Ruzzo and J. D. Ullman, Protection in operating systems, Communications of the ACM,19(8): 461-471, 1976.

[ITSEC91] Information Technologie Security Evaluation Criteria, European Comunities, juin 1991.

[Jen99] Christian Damsgaard JENSEN, Un modèle de contrôle d'accès générique et sa réalisation dans la mémoire virtuelle répartie unique Arias, Thèse de doctorat de l'université Joseph Fourier, Grenoble 1, discipline : Informatique, soutenue le 29 octobre 1999.

[Lam71] Butler W. Lampson, Protection, In 5th Princeton Symposium on Information Science and Systems, pages 437-443, 1971. Reprinted in ACM Operating Systems Review 8(1): 18-24, 1974.

[Lan81] Carl E. Landwehr, Formal Models for Computer Security, ACM Computing Surveys, 13(3): 247-278, 1981.

[ManuB02] CLEARSY, Manuel de référence du langage B, version 1.8.5, 10 Janvier 2002.

[Mcl90] J. Mclean, The specification and modeling of computer security, Computer, 23(1): 9-16, January 1990.

[NO99] M. Nyanchama and S.Osborn, The graph model and conflicts of interest, ACM Trans. Inf. Syst. Sec. 2, 1.

[OSM00] S. Osborn, R. Sandhu and Q. Munawer, Configuring Role-Based Access to enforce Mandatory and Discretionary Access Control Policies, ACM Transactions on Information and System Security, vol. 3, n. 2, May 2000, pages 85-106.

[Potet] Marie-Laure Potet, Initiation rapide à la méthode B,

http://www-lsr.imag.fr/Les.Personnes/Marie-Laure.Potet .

[SAN92] R. S. Sandhu, The typed access matrix model, In Proc. of 1992 IEEE Symposium on Security and Privacy, pages 122-136, Oakland, CA, May 1992.

[SAN96] R.S. Sandhu, Role Hierarchies and Constraints for Lattice-Bases Access Controls, in 4th European Symposium on Research in Computer Security (ESORICS'96), (E. Bertino, H. Kurth, G. Martella, E. Mon, tolivo, Eds.), Rome, Italie, September 25-27, Lecture Notes in Computer Science 1146, pp. 65-79, Springer-Verlag, 1996.

[SD01] Pierangela Samarati and Sabrina de Capitani di Vimercati, Access Control: Policies, Models, and Mechanisms, FOSAD 2000, LNCS 2171, pages 137-196, 2001.

[Sec] Securing Java, http://www.securingjava.com

[SMJ01] Andreas Schaad, Jonathan Moffett et Jeremy Jacob, The Role-Based Access Control System of a European Bank: A Case Study and Discussion, SACMAT 2001, 6th ACM Symposium on Access Control Models and Technologies, Chantilly, VA, USA, ACM, 8 pages.