Méthodes formelles pour l'extraction d'attaques internes des Systèmes d'Information

Amira Radhouani

Communauté UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Amira RADHOUANI

Thèse dirigée par **Yves LEDRU**, Professeur UJF, et
codirigée par **Idani AKRAM** MCF

préparée au sein du **Laboratoire Laboratoire d'Informatique de Grenoble**
dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Méthodes formelles pour l'extraction d'attaques internes des Systèmes d'Information

Formal methods for extracting insider attacks from Information Systems

Thèse soutenue publiquement le **23 juin 2017**,
devant le jury composé de :

Monsieur JACQUES JULLIAND
PROFESSEUR, UNIVERSITE DE FRANCHE-COMTE, Président
Monsieur YAMINE AIT AMEUR
PROFESSEUR, INP TOULOUSE - ENSEEIHT, Rapporteur
Madame LEILA JEMNI BEN AYED
PROFESSEUR, UNIVERSITE DE LA MANOUBA - TUNISIE, Rapporteur
Madame AMEL MAMMAR
MAITRE DE CONFERENCES, TELECOM SUDPARIS, Examinateur
Madame AMEL BORGI
MAITRE DE CONFERENCES, UNIVERSITE DE TUNIS - EL MANAR - TUNISIE, Examinateur
Monsieur YVES LEDRU
PROFESSEUR, UNIVERSITE GRENOBLE ALPES, Co-directeur de thèse
Monsieur AKRAM IDANI
MAITRE DE CONFERENCES, GRENOBLE INP, Directeur de thèse
Madame NARJES BEN RAJEB
PROFESSEUR, UNIVERSITE DE CARTHAGE - TUNISIE, Examinateur

Résumé

La sécurité des Systèmes d'Information (SI) constitue un défi majeur car elle conditionne amplement la future exploitation d'un SI. C'est pourquoi l'étude des vulnérabilités d'un SI dès les phases conceptuelles est cruciale. Il s'agit d'étudier la validation de politiques de sécurité, souvent exprimées par des règles de contrôle d'accès, et d'effectuer des vérifications automatisées sur des modèles afin de garantir une certaine confiance dans le SI avant son opérationnalisation. Notre intérêt porte plus particulièrement sur la détection des vulnérabilités pouvant être exploitées par des utilisateurs internes afin de commettre des attaques, appelées attaques internes, en profitant de leur accès légitime au système. Pour ce faire, nous exploitons des spécifications formelles B générées, par la plateforme B4MSecure, à partir de modèles fonctionnels UML et d'une description SecureUML des règles de contrôle d'accès basées sur les rôles. Ces vulnérabilités étant dues à l'évolution dynamique de l'état fonctionnel du système, nous proposons d'étudier l'atteignabilité des états, dits indésirables, donnant lieu à des attaques potentielles, à partir d'un état normal du système. Les techniques proposées constituent une alternative aux techniques de model-checking. En effet, elles mettent en œuvre une recherche symbolique vers l'arrière fondée sur des approches complémentaires : la preuve et la résolution de contraintes. Ce processus de recherche est entièrement automatisé grâce à notre outil GenISIS qui a montré, sur la base d'études de cas disponibles dans la littérature, sa capacité à retrouver des attaques déjà publiées mais aussi des attaques nouvelles.

Mots-clefs : Attaques internes, B, preuve, résolution de contraintes, model-checking, atteignabilité, RBAC, Système d'Information (SI).

Abstract

The early detection of potential threats during the modelling phase of a Secure Information System (IS) is required because it favours the design of a robust access control policy and the prevention of malicious behaviours during the system execution. This involves studying the validation of access control rules and performing vulnerabilities automated checks before the IS operationalization. We are particularly interested in detecting vulnerabilities that can be exploited by internal trusted users to commit attacks, called insider attacks, by taking advantage of their legitimate access to the system. To do so, we use formal B specifications which are generated by the B4MSecure platform from UML functional models and a SecureUML modelling of role-based access control rules. Since these vulnerabilities are due to the dynamic evolution of the functional state, we propose to study the reachability of some undesirable states starting from a normal state of the system. The proposed techniques are an alternative to model-checking techniques. Indeed, they implement symbolic backward search algorithm based on complementary approaches : proof and constraint solving. This rich technical background allowed the development of the GenISIS tool which automates our approach and which was successfully experimented on several case studies available in the literature. These experiments showed its capability to extract already published attacks but also new attacks.

Keywords : Insider attacks, B-Method, proof, constraint solving, model-checking, reachability, RBAC, IS.

A la mémoire de mon cher oncle **Omar**: « Enfant, alors que je ne savais même pas ce que c'est qu'une thèse de doctorat, je t'ai promis de te l'offrir un jour. Mélancolique, je te dédie ce travail. Que ton âme repose en paix! »

A la prunelle de mes yeux, à la joie de ma vie, à mon petit cœur d'amour, à ma fille bien aimée **Maëlle**: "En espérant qu'un jour tu m'offrirais la tienne ;)"

Remerciements

Au terme de cette thèse, je réserve ces lignes pour exprimer ma sincère reconnaissance et ma profonde gratitude à toute personne qui, de loin ou de près, a contribué à l'accomplissement de ce travail. A toutes ces personnes, je dis un grand MERCI d'avoir fait de ce rêve une réalité...

****

MERCI *à mes directeurs de thèse...*

A M. Akram IDANI : *"Je te suis avant tout très reconnaissante de t'être donné pour m'obtenir le financement qui m'a permis de faire ce travail dans les meilleures conditions. Je te remercie d'avoir cru en moi du début jusqu'à la fin et j'espère que je ne t'ai pas déçu. Merci pour tes encouragements, ton soutien et aussi pour la pression que tu as su mettre quand il le fallait. Merci pour tes conseils, ton implication, ta rigueur, ton exigence et ton sérieux... J'ai beaucoup appris de toi tant sur le plan scientifique que sur le plan pédagogique et humain. Ce fut un grand plaisir de travailler avec toi, et j'espère un jour pouvoir être comme toi !"*

A M. Yves LEDRU : *"Ce travail n'aurait jamais vu le jour sans ta gentillesse, ta compréhension et ta flexibilité qui m'ont permis de surpasser mes contraintes personnelles. Merci pour ta disponibilité, pour l'encadrement de qualité, pour tes conseils judicieux et pour tes remarques très fines. Merci pour tes relectures innombrables des différentes versions de mon manuscrit ainsi que des différents articles. Tu m'as appris à porter attention aux plus petits détails. J'ai tiré un grand enseignement de notre collaboration et j'espère avoir été à la hauteur de tes attentes !"*

A Mme. Narjes BEN RAJEB : *"L'histoire a commencé quand j'étais ton élève ingénieure quand tu m'as transmis ta passion pour la logique et les méthodes formelles. Merci de m'avoir toujours encouragée à faire de la recherche mon métier. Merci d'être à mes côtés lors de mon obtention de tous mes diplômes universitaires (ingénieur, M2R, et maintenant doctorat). Merci d'avoir cru en moi !"*

****

MERCI *à M.Micheal LEUSCHEL qui a accepté de m'accueillir au sein de son unité de recherche à Düsseldorf pendant une semaine. Bien que ce séjour était court, la réactivité et l'implication de son équipe ainsi que sa disponibilité m'ont permis d'améliorer la performance de*

mon outil. Je le remercie profondément pour l'intérêt qu'il a porté à ce travail ainsi que pour les remarques et les pistes d'amélioration qu'il m'a proposées.

****

MERCI *à tous les membres de jury d'avoir accepté d'évaluer ce travail. Je remercie particulièrement M.Jacques JULLIAND qui m'a fait l'honneur de présider le jury. Mes vifs remerciements s'adressent également à M.Yamine AIT AMEUR et Mme.Leila JEMNI BEN AYED qui ont accepté de lire mon manuscrit et de me faire part de leurs remarques. Je remecie aussi Mme.Amel MAMMAR et Mme.Amel BORGI d'avoir accepté d'examiner ce travail.*

****

MERCI *à ma famille qui m'a épaulé et qui m'a soutenu et surtout qui m'a supporté :)*

A mon cher mari Halim : "Merci d'être là pour moi pendant mes moments de faiblesse, de déprime et aussi de bonheur. Merci de m'avoir toujours encouragé et rassuré malgré les distances qui nous ont séparé. Merci d'avoir supporté les longs trajets et sacrifié tes weekends pendant deux longues années. Sans toi je ne serai pas où j'en suis. Sache que tu es le choix le plus intelligent que j'ai fait de ma vie !"

A mes chers parents Fadhila et Azouz :" Merci de continuer à veiller sur mon bien-être malgré mon âge adulte. Merci d'avoir enduré les lourdes démarches administratives de l'école doctorale tunisienne à ma place. Merci pour vos sacrifices, vos prières et votre amour inconditionnel. J'espère avoir réussi à vous rendre fiers de moi !"

****

MERCI *infiniment !*

Table des matières

Liste des figures xiv

Liste des tableaux xv

Introduction générale 1

Motivations . . . 2
Contexte du travail . . . 3
Contributions et propositions . . . 5
Plan de la thèse . . . 6
Publications issues de ce travail de thèse . . . 8

I État de l'art 10

1 Validation formelle des modèles de contrôle d'accès basés sur les rôles 11

1.1 Fondements du modèle de contrôle d'accès basé sur les rôles . . . 12
1.1.1 Noyau du modèle RBAC . . . 13
1.1.2 RBAC hiérarchique . . . 14
1.1.3 Les contraintes de séparation de devoirs . . . 14
1.1.4 Avantages et inconvénients du modèle RBAC . . . 15
1.2 Extension du modèle RBAC . . . 15
1.3 Vulnérabilités des modèles de contrôle d'accès . . . 17
1.3.1 Classification des attaquants initiés . . . 18
1.3.2 Classification des attaques internes . . . 19
1.4 Vérification et validation du contrôle d'accès . . . 20
1.4.1 Méthodes de Vérification et Validation (V&V) formelles . . . 21
1.4.2 Panorama des approches existantes pour la validation du contrôle d'accès . . . 23
1.5 Conclusion . . . 29

2 Introduction à la méthode B 30

2.1 Fondements de la méthode B . . . 31
2.1.1 Notations de la théorie des ensembles . . . 32
2.1.2 Machine abstraite . . . 32

2.1.3 Langage des substitutions généralisées . . . 34
2.1.4 Propriétés des substitutions généralisées . . . 35
2.1.5 Obligations de preuve . . . 36
2.1.6 Exemple . . . 37
2.2 Les notions de raffinement et de modularité en B . . . 38
2.2.1 Le raffinement . . . 39
2.2.2 La modularité . . . 39
2.3 Outils de V&V . . . 39
2.3.1 AtelierB . . . 40
2.3.2 GénéSyst . . . 40
2.3.3 ProB . . . 42
2.4 Combinaison de B et de CSP . . . 43
2.4.1 Syntaxe des spécifications . . . 43
2.4.2 Exemple de contrôle d'une machine B . . . 45
2.5 Conclusion . . . 47

3 B4MSecure pour la validation conjointe en UML et B des SI sécurisés 49
3.1 Modélisation fonctionnelle . . . 51
3.1.1 Exemple illustratif : SI d'une bibliothèque . . . 51
3.1.2 Traduction des aspects structurels . . . 51
3.1.3 Génération des opérations . . . 52
3.2 Modélisation des règles de contrôle d'accès RBAC . . . 55
3.2.1 Exemple illustratif : SI d'une bibliothèque sécurisée . . . 55
3.2.2 Traduction en B des règles de sécurité . . . 58
3.3 Validation des modèles . . . 61
3.3.1 Validation du modèle fonctionnel . . . 62
3.3.2 Validation du modèle de sécurité . . . 63
3.4 Conclusion . . . 65

II Contributions de la thèse 67

4 Vérification de l'atteignabilité d'un état en B 68
4.1 Notions préliminaires . . . 69
4.2 Approche basée sur la preuve . . . 72
4.2.1 Extraction d'opérations vérifiant la propriété de succession . . . 72
4.2.2 Extraction d'un chemin symbolique d'atteignabilité . . . 74
4.2.3 Algorithme d'extraction de l'ensemble des chemins symboliques . . . 78
4.2.4 Concrétisation des séquences symboliques . . . 81
4.3 Approche basée sur la résolution de contraintes . . . 82
4.4 Étude de cas : SI d'une bibliothèque . . . 84

4.4.1 Preuve du premier chemin(Q_{symb1} / Q_1) . . . 85
4.4.2 Preuve du deuxième chemin (Q_{symb2} / Q_2) . . . 86
4.5 Comparaison avec des travaux similaires . . . 89
4.6 Bilan et discussion . . . 91

5 Extraction des scénarios d'attaque interne **93**
5.1 Motivation et notions préliminaires . . . 94
5.2 Définition formelle d'un scénario d'attaque . . . 98
5.3 Caractérisation des états indésirables . . . 100
5.4 Technique d'extraction de scénarios d'attaque . . . 102
5.4.1 Extraction de comportements malicieux . . . 103
5.4.2 Génération de l'état initial . . . 105
5.4.3 Identification des utilisateurs malicieux . . . 109
5.4.4 Automatisation de l'identification des utilisateurs malicieux . . . 111
5.4.5 Extraction automatique des attaques masquées . . . 113
5.5 Bilan et discussion . . . 115

6 Outil et expérimentations **118**
6.1 Présentation de l'outil GenISIS . . . 119
6.1.1 Architecture générale . . . 119
6.1.2 Architecture structurelle de GenISIS . . . 123
6.2 Expérimentations et discussion . . . 125
6.3 Amélioration de la performance de GenISIS . . . 127
6.3.1 Approche basée sur l'analyse syntaxique et la preuve . . . 127
6.3.2 Approche basée sur les heuristiques . . . 131
6.4 Etude de cas . . . 133
6.4.1 Modèle fonctionnel . . . 133
6.4.2 Modèle de sécurité . . . 133
6.4.3 Extraction de scénarios d'attaque . . . 136
6.5 Conclusion . . . 141

Conclusion générale **143**
Bilan de nos contributions . . . 144
Perspectives . . . 146

Bibliographie **I**

A Spécifications du SI de la bibliothèque **II**
A.1 Le modèle fonctionnel . . . II
A.2 Spécification de l'opération `addRoleSafe` . . . V

B Test et vérification des filtres de sécurité **VI**
B.1 Vérification des scénarios de cas d'utilisation normaux VI
B.2 Vérification des filtres de sécurité VIII
B.2.1 Approche par l'extraction du préambule IX
B.2.2 Approche par la génération de l'état initial XII
B.3 Bilan XIII

C Glossaire **XV**

Liste des figures

1 Rapport des attaques internes et externes [IBM 2016] 3

1.1 Le modèle RBAC [FERRAIOLO et al. 2001] 13
1.2 Le modèle RBAC avec contraintes d'autorisation 16

2.1 Structure générale d'une machine B 33
2.2 Spécification B d'une opération d'addition simple 35
2.3 Exemple d'une machine B 38
2.4 Génération des obligations de preuve de l'exemple 2.3 par AtelierB 40
2.5 STS généré par GénéSyst à partir de l'exemple 2.3 41
2.6 Architecture des composants CSP||B [SCHNEIDER et TREHARNE 2005] 43
2.7 Architecture CSP||B du contrôle de l'exemple 2.3 45
2.8 Contrôleur *Communicating Sequential Process* (CSP) de la machine `Library` . . 46
2.9 Caractérisation des ensembles de données 46
2.10 Espace d'états avec guidance par le contrôleur CSP 46
2.11 Espace d'états sans guidance par le contrôleur CSP 47

3.1 Processus de dérivation de *Unified Modeling Langage* (UML) vers B mis en oeuvre par la plate-forme B4MSecure 50
3.2 Diagramme de classe UML du SI de la bibliothèque 51
3.3 Invariant généré automatiquement 52
3.4 Spécification en B de l'opération `Member_AddLend` 54
3.5 Spécification de l'opération `Member_TakeReservedBook` 54
3.6 Méta-modèle RBAC mis en oeuvre par l'approche B4MSecure 56
3.7 Affectation des utilisateurs aux rôles 57
3.8 Contrôle d'accès aux entités du SI de la biliothèque 57
3.9 Partie statique et initialisation de la machine d'affectation des utilisateurs aux rôles 59
3.10 Opération de connection d'un utilisateur 59
3.11 Version sécurisée de l'opération `Member_AddLend` 60
3.12 Version sécurisée de l'opération `Member_New` 61
3.13 Scénario fonctionnel animé par ProB 62
3.14 Scénario animé par ProB dans le modèle de sécurité 64
3.15 Exemple d'un scénario d'attaque extrait par model-checking 65

4.1 Exemple d'un état concret du SI de la bibliothèque . . . 70
4.2 Exemple d'un état symbolique du SI de la bibliothèque . . . 70
4.3 Atteignabilité d'un état symbolique . . . 71
4.4 Chemin symbolique d'atteignabilité . . . 74
4.5 Illustration de la proposition 1 . . . 75
4.6 Illustration de la proposition 2 . . . 76
4.7 Illustration du corollaire 1 . . . 77
4.8 Illustration de la proposition 3 . . . 77
4.9 Algorithme d'extraction de l'ensemble des chemins symboliques . . . 79
4.10 Schéma illustratif du principe de partitionnement des états . . . 80
4.11 Limite de l'algorithme de recherche de séquences symboliques . . . 81
4.12 Contrôleur CSP pour guider le model-checker pour l'extraction de la séquence concrète . . . 81
4.13 Algorithme d'extraction de l'ensemble des chemins concrets . . . 83
4.14 Initialisation de la machine B . . . 84
4.15 Caractérisation des ensembles . . . 84
4.16 Système de transition construit par l'algorithme de recherche . . . 86

5.1 Etat initial de la machine fonctionnelle . . . 95
5.2 Scénario d'une attaque basique . . . 95
5.3 Scénario d'une attaque à contrainte . . . 96
5.4 Scénario d'une attaque planifiée . . . 97
5.5 Scénario d'une attaque masquée . . . 97
5.6 Opération fonctionnelle enrichie par une contrainte d'autorisation . . . 103
5.7 Définition des victimes et des attaquants . . . 104
5.8 Initialisation de la machine B . . . 104
5.9 Spécification des ensembles . . . 105
5.10 Correction de l'opération `Book_Free` . . . 110
5.11 Traduction d'une séquence fonctionnelle en CSP . . . 112
5.12 Q_{F_1} traduit en un contrôleur CSP . . . 113
5.13 Contrôleur CSP général . . . 113
5.14 Séquence fonctionnelle atteignant l'état indésirable . . . 116

6.1 Architecture générale de GenISIS . . . 120
6.2 Exemple d'un fichier CCL . . . 121
6.3 architecture structurelle détaillée de GenISIS . . . 124
6.4 Analyse de déclenchabilité de l'opération `Member_AddLend` (généré par ProB) . . . 130
6.5 Modèle fonctionnel du SI de revue des articles . . . 133
6.6 Affectations des utilisateurs aux rôles . . . 134
6.7 Modèle fonctionnel du SI de revue d'articles . . . 134
6.8 Définition des victimes et des attaquants . . . 136

6.9 Initialisation de la machine B 138
6.10 Traduction des comportements malicieux en un contrôleur CSP 139
6.11 Scénarios d'attaque avec collusion d'utilisateurs 139
6.12 Scénario d'attaque impliquant un seul utilisateur 140
6.13 Fausses attaques 141
6.14 Le processus de conception des SI sécurisés 144

B.1 Contrôleur CSP pour la vérification d'un scénario fonctionnel VII
B.2 Scénario produit par ProB à partir du modèle de sécurité VII
B.3 Structuration des tests VIII
B.4 Contrôleur CSP pour la vérification des filtres de sécurité X
B.5 Préambule de test de l'opération `Member_AddLend(Bob, bo)` X
B.6 Initialisation de la machine B X
B.7 Exemple d'un contrôleur CSP pour la vérification des filtres de sécurité XI
B.8 Scénarios de sécurité générés par ProB XII
B.9 Contrôleur CSP pour la vérification des filtres de sécurité à partir d'un état initial déclanchant l'opération à tester XII
B.10 Initialisation généré par ProB XIII

Liste des tableaux

1.1 Synthèse des approches pour la modélisation et la validation des modèles de contrôle d'accès . . . 28

2.1 Notations ensemblistes en B . . . 32
2.2 Les substitutions primitives en B . . . 34
2.3 Calcul de la plus faible précondition des substitutions primitives . . . 35
2.4 Obligations de preuve de l'initialisation et des opérations . . . 37

3.1 Relations fonctionnelles issues des attributs de classes . . . 52
3.2 Opérations de base générées par B4MSecure . . . 53

4.1 Tableau d'analogie entre la méthode B classique et Event-B . . . 73
4.2 Chemins extraits par nos algorithmes . . . 85

5.1 Des scénarios d'attaque sur le SI de la bibliothèque . . . 106
5.2 Les couples des opérations inverses générées par B4MSecure . . . 114

6.1 Tableau de synthèse des expérimentations . . . 125
6.2 Tableau de calcul des performances de GenISIS pour le traitement de l'exemple de la bibliothèque . . . 126
6.3 Tableau de calcul des performances de GenISIS pour le traitement de l'exemple du SI de revue d'articles dans une conférence . . . 126
6.4 Exemple de calcul du niveau de rapprochement de l'état initial . . . 132
6.5 Spécification des contraintes d'autorisation . . . 135

Introduction générale

« *The introduction of many minds into many fields of learning along a broad spectrum keeps alive questions about the accessibility, if not the unity, of knowledge.* »

Edward Levi

Les systèmes informatiques ne cessent de se développer et de nous envahir suivant un rythme effréné au point que nous en devenons de plus en plus dépendants. En effet, aujourd'hui, la plupart des activités humaines se font à travers des machines inter-connectées en réseau qu'il devient difficile de contrôler. Dans un tel contexte, assurer la sécurité des systèmes informatiques est devenu un enjeu majeur et un souci permanent aussi bien pour les utilisateurs que pour les entreprises. D'une part, les utilisateurs craignent la divulgation de leurs données personnelles et la violation de leur intimité, et d'autre part, les entreprises risquent des pertes financières importantes et l'atteinte à leur réputation.

Face à ces risques accrus, l'analyse des vulnérabilités d'un SI dès les phases conceptuelles est devenue cruciale et indispensable pour le développement d'un SI qui répond aux exigences de sécurité. En effet, selon la norme internationale de système de gestion de la sécurité de l'information *ISO/CEI 27001* [FERNANDEZ-TORO 2007], tout SI sécurisé doit préserver les trois propriétés fondamentales à savoir la confidentialité, l'intégrité et la disponibilité :

- **La confidentialité :** Cette propriété permet de protéger le contenu des informations à caractère secret sauvegardées dans le SI. Ainsi, pour respecter cette propriété, un SI doit se doter d'un mécanisme qui permet d'empêcher toute personne physique ou logique non autorisée à lire ces informations. Dans un monde idéal, le système doit également empêcher les ayants droits de divulguer ces informations aux individus non autorisés.
- **L'intégrité :** Afin d'éviter que les informations soient altérées, et par conséquent garantir

la validité et l'intégrité des données sauvegardées par le SI, ce dernier doit pouvoir détecter toute modification indue, accidentelle ou intentionnelle, de ces données. En effet, seuls les ayants droits doivent pouvoir accéder en écriture à ces données.

- **La disponibilité :** Le système doit être accessible et prêt à l'emploi lorsqu'un utilisateur légitime en a besoin durant les plages d'utilisation prévues.

Par conséquent, pour garantir la préservation de ces propriétés et pour éviter que le SI soit altéré, certains dispositifs de sécurité peuvent être mis en œuvre dont le contrôle d'accès. Ce dernier permet de filtrer l'accès aux ressources matérielles et logicielles du SI selon des règles préalablement définies, dans le but d'assurer que seuls les utilisateurs légitimes considérés de confiance peuvent y accéder.

Cependant, ces règles mêmes peuvent amplifier les vulnérabilités voire nuire au bon fonctionnement du SI si elles ne sont pas correctement définies. Ainsi, la V&V des règles de contrôle d'accès est indispensable pour garantir un haut niveau de confiance dans le SI. Toutefois, en l'absence des mécanismes d'automatisation, ces activités de V&V peuvent devenir fastidieuses et très difficiles à mener surtout lorsqu'il s'agit d'analyser la sécurité des SI de tailles importantes.

Pour cette raison, nous proposons, dans le cadre de ce travail de thèse, d'étudier la validation des politiques de contrôle d'accès, et de mettre en œuvre une approche automatisable permettant, d'une part, de vérifier que les règles de sécurité sont correctement définies, et d'autre part, de détecter les attaques contre le système.

Dans notre investigation nous nous intéressons à la détection d'un type d'attaque spécifique, dit attaque interne, qui est perpétré par les utilisateurs internes au SI. Ces utilisateurs sont généralement considérés comme des individus de confiance autorisées à accéder et à gérer les ressources du système. En raison de leur parfaite connaissance de l'architecture du système et des règles de sécurité appliquées, ils peuvent abuser de leurs droits pour commettre des actes illicites.

Motivations

Dans un monde où il est devenu dangereux de faire confiance, les cyberattaques internes ont connu une forte hausse lors de la dernière décennie. En effet, selon une étude faite dernièrement par IBM [IBM 2016], le taux des attaques internes est passé de 55% en 2014 à 60% en 2015 alors que le taux des attaques externes diminue constamment (Figure 1).

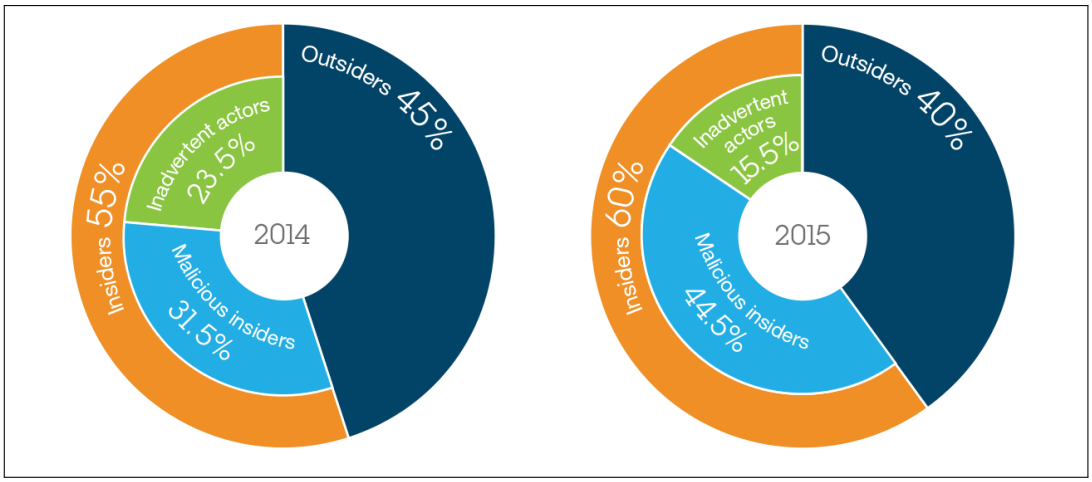


FIGURE 1 – Rapport des attaques internes et externes [IBM 2016]

Ceci est dû à l'intérêt que les organisations portent aux dispositifs de sécurité qui leur permettent de se défendre contre les intrusions externes. Ces mécanismes de haute technologie ont, certes, contribué au renforcement de la sécurité et ont aidé ces organisations à lutter contre les accès non autorisés. Cependant, elles ne sont pas suffisantes pour garantir la sécurité, et surtout pour empêcher les utilisateurs malicieux ayant déjà des accès légitimes de commettre des actes frauduleux.

Selon l'étude faite par IBM [IBM 2016] le niveau élevé de sécurisation des réseaux et des logiciels contre les pénétrations externes est la cause principale de l'amplification des attaques internes. En effet, "les criminels porteront leurs efforts sur l'utilisateur final, plutôt que sur l'exploitation de failles". C'est pour cette raison que les entreprises doivent à leur tour changer de stratégie et porter leurs efforts sur le contrôle du comportement des usagers internes.

Ces derniers, connaissant la logique de la politique de sécurité, tentent de la contourner pour s'octroyer de nouveaux privilèges leur permettant d'accéder à des ressources qui leur sont initialement interdites. Aussi, notre travail de thèse se propose-t-il de contribuer à l'analyse des comportements des utilisateurs légitimes d'un SI. Le but de notre investigation est l'extraction des scénarios d'utilisation suspects qui peuvent être exécutés par des utilisateurs malicieux en vue de débloquer certains accès interdits.

Contexte du travail

La complexité des SI sécurisés impose l'utilisation d'environnements technologiques fiables pour garantir le développement d'une application sûre qui répond à toutes les exigences et respecte toutes les contraintes de sécurité. Par conséquent, la construction de tels systèmes demande des efforts supplémentaires dès la phase de spécification pour une meilleure maîtrise des besoins afin de pouvoir anticiper et corriger les incohérences et les vulnérabilités tôt dans le cycle de vie du logiciel.

Dans les méthodes d'ingénierie des besoins, deux types d'exigences peuvent être identifiées et par la suite deux types de modèles peuvent en découler. D'une part, le **modèle fonctionnel** permet de décrire les exigences fonctionnelles, ce qui convient pour identifier les objectifs métiers d'un système et pour analyser ses besoins d'un point de vue opérationnel. D'autre part, le **modèle non-fonctionnel** qui est aussi important et qui s'inscrit dans le cadre d'une démarche qualité, permet de mettre l'accent sur les exigences non fonctionnelles notamment les aspects de performance, de qualité ou de sécurité. Dans le cas des SI sécurisés, nous nous intéressons en particulier aux exigences de sécurité et plus précisément aux règles de contrôle d'accès. C'est pourquoi nous utilisons dans la suite le terme **modèle de sécurité**.

Ce modèle sert à décrire la manière dont l'accès aux ressources est géré en suivant un mécanisme bien déterminé. En effet, plusieurs mécanismes de contrôle d'accès existent tels que le Contrôle d'Accès Discrétionnaire – ou *Discretionnary Access Control* (DAC) [LAMPSON 1974], le Contrôle d'Accès Obligatoire – ou *Mandatory Access Control* (MAC) [BELL et LAPADULA 1976, DENNING 1976], le Contrôle d'Accès Basé sur les Rôles – ou *Role Based Access Control* (RBAC), etc. Notre intérêt porte plus particulièrement sur ce dernier qui a été démontré dans [QAMAR 2011] plus adapté pour l'expression des règles de sécurité dans le cadre des SI.

Dans ce contexte, des travaux élaborés dans l'équipe VASCO [1] du Laboratoire d'Informatique de Grenoble [IDANI et al. 2010, IDANI et LEDRU 2015, LEDRU et al. 2015] proposent de mettre en œuvre, au moyen de la plate-forme B4MSecure, une modélisation conjointe en UML et B[ABRIAL 1996a] des aspects fonctionnels du SI ainsi que des règles de contrôle d'accès basé sur le rôle. D'une part, les modèles graphiques UML, fondés sur le profil SecureUML [LODDERSTEDT et al. 2002], permettent de disposer de vues compréhensibles et structurantes, et d'autre part, leurs contre-parties en méthode formelle B permettent de tirer profit des outils de V&V formelles comme des animateurs, des model-checkers, des outils de preuve ou encore des solveurs de contraintes. Ces travaux ont montré, sur la base d'études de cas concrètes, l'intérêt d'une telle approche, en comparaison avec les travaux existants, pour des objectifs de validation sur les trois plans suivants :

1. Validation du modèle fonctionnel indépendamment des règles de contrôle d'accès.
2. Validation du modèle de sécurité en faisant une abstraction sur les aspects fonctionnels.
3. Validation des liens entre le modèle fonctionnel et le modèle de sécurité.

Dans notre travail de thèse nous portons un intérêt particulier à cette dernière facette de validation. Les spécifications formelles B produites par la plate-forme B4MSecure nous serviront, donc, de points d'entrée pour notre investigation.

En effet, bien qu'elle soit très peu étudiée dans la littérature, l'analyse de l'impact du modèle

1. http://vasco.imag.fr/

fonctionnel sur le modèle de sécurité et vice versa est très importante pour la validation de la sécurité d'un SI. D'une part, cette analyse permet de vérifier que les règles de sécurité ne bloquent pas la réalisation de scénarios fonctionnels. D'autre part, il a été montré dans [LEDRU et al. 2014] que certaines règles de contrôle d'accès, dépendant des données issues du modèle fonctionnel, peuvent être contournées en faisant évoluer l'état fonctionnel du SI. Par conséquent, ces règles de sécurité peuvent être à l'origine des failles de sécurité qui sont exploitées par des utilisateurs malicieux pour commettre des attaques internes.

Contributions et propositions

L'objectif principal de ce travail de thèse est l'extraction des attaques internes des SI à partir d'une spécification formelle B. L'approche que nous proposons dans ce contexte sort du cadre spécifique du domaine de détection des attaques ou encore de la validation de la sécurité des SI et couvre le domaine plus général de la vérification de la propriété d'atteignabilité en B [ABRIAL et MUSSAT 1998]. En effet, nous contribuons sur les trois plans suivants :

1. **Vérification formelle de la propriété d'atteignabilité en B :**
   Nous avons montré que l'extraction d'attaques internes dans un SI se ramène à résoudre un problème d'atteignabilité. En effet, nous définissons un scénario d'attaque comme l'exécution d'une séquence d'opérations permettant d'atteindre un état, dit indésirable, à partir duquel certaines règles de contrôle d'accès peuvent être contournées. Ainsi, pour exhiber les scénarios d'attaque, nous analysons l'atteignabilité de ces états indésirables.

   Pour ce faire, nous proposons une approche générale qui peut être appliquée en dehors du cadre de la détection des attaques. En effet, notre approche permet également de vérifier l'évolution dynamique d'un SI et permet de démontrer qu'il est capable d'atteindre un état cible à partir d'un état initial.

   Notre approche est basée sur une recherche avec retour en arrière, qui part de l'état cible et remonte jusqu'à ce que l'état initial est rencontré. Pour la mise en œuvre de notre algorithme de recherche nous avons proposé deux techniques formelles complémentaires : l'une basée sur la preuve et l'autre basée sur la résolution de contraintes. Ces deux techniques constituent, d'une part, une alternative aux techniques de model-checking, et d'autre part, elles permettent de guider un model-checker dans son exploration pour extraire des traces d'exécution menant à un état cible.

2. **Extraction des scénarios d'attaques internes :**
   Sur ce point central à notre travail de thèse, nous contribuons au niveau des trois points suivants :

- **Classification des attaques internes :** Nous avons proposé de classer les attaques internes en quatre types différents : attaque basique, attaque à contrainte, attaque planifiée et attaque masquée. Nous avons, ainsi, proposé une caractérisation des états indésirables relatifs à chacune de ces attaques et nous avons montré comment les extraire.

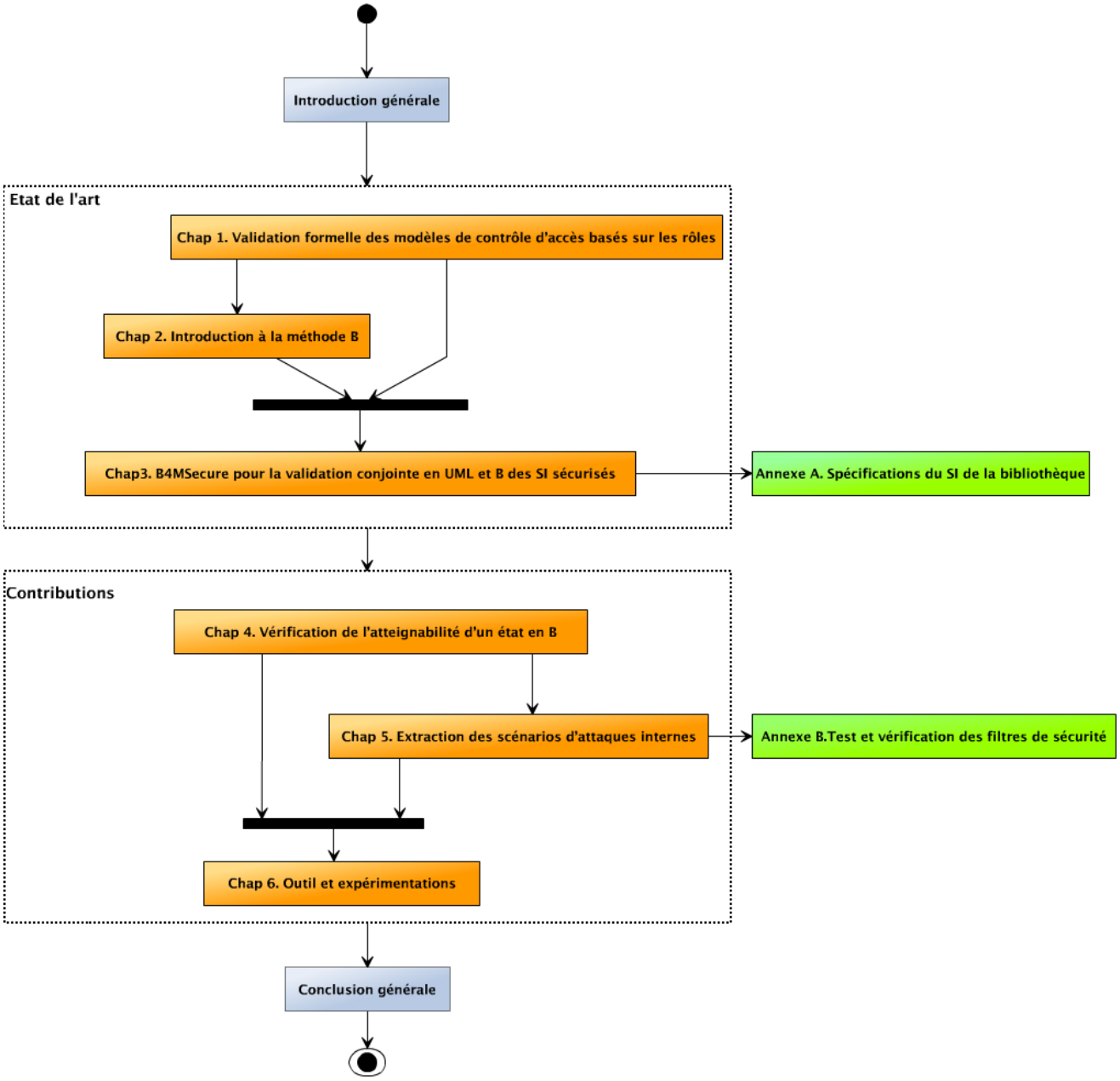
- **Synthèse des scénarios d'attaques :** Pour extraire les scénarios d'attaques à partir d'une spécification formelle B, nous avons proposé une approche en deux phases. Dans la première phase, nous identifions les états indésirables et nous étudions leur atteignabilité dans le modèle fonctionnel. Dans la deuxième phase, nous analysons la possibilité d'exécution des traces d'atteignabilité dans le modèle de sécurité en vue d'identifier les profils des utilisateurs suspects. Ainsi, nous définissons une approche pour extraire des attaques faisant intervenir un seul utilisateur ou une collusion entre plusieurs utilisateurs.

- **Outil pour l'automatisation de l'extraction de scénarios d'attaques :** Une automatisation de l'approche d'extraction de scénarios d'attaques est proposée. Cette automatisation est concrétisée à travers l'outil GenISIS (*Generator of Insider Scenarios from an Information System*).

3. **Validation de la sécurité des SI :** A un niveau exploratoire, nous montrons l'applicabilité de notre approche d'extraction de scénarios d'attaques pour la validation de la sécurité du SI. Nous proposons, ainsi, d'étendre l'approche pour la validation des scénarios de cas d'utilisation normaux. Nous proposons, également, d'utiliser nos techniques pour la génération de tests de sécurité. Ces tests permettent de vérifier que les règles de sécurité remplissent convenablement leurs missions qui consistent à empêcher les utilisateurs non habilités et à autoriser les utilisateurs légitimes à accéder aux ressources protégées.

Plan de la thèse

Ce mémoire de thèse est subdivisé en six chapitres organisés comme suit :



Dans les trois premiers chapitres nous présentons le contexte scientifique du travail ainsi que l'état de l'art :

- Le chapitre 1 intitulé "**Validation formelle des modèles de contrôle d'accès basés sur les rôles**" présente les fondements du modèle RBAC classique ainsi que son extension pour la prise en compte de certaines contraintes contextuelles. Dans ce chapitre, nous donnons également la définition des attaques internes et nous faisons une classification des types d'attaques qui nous intéressent. Finalement, nous faisons un tour d'horizon des techniques de V&V existantes pour la validation du contrôle d'accès en général et du modèle RBAC en particulier.

- Le chapitre 2 intitulé "**Introduction à la méthode B**" présente les fondements de la méthode B en mettant l'accent sur les concepts utilisés pour l'élaboration de ce travail de thèse.
- Le chapitre 3 intitulé "**B4MSecure pour la validation conjointe en UML et B des SI sécurisés**" décrit le processus de dérivation UML/B proposé par la plate-forme B4MSecure, ainsi que les activités de validation qui en découlent. Dans ce chapitre, nous mettons l'accent sur les travaux qui ont exploité cette plate-forme pour mener des activités de validation de la sécurité des SI et nous les critiquons.

Quand aux trois derniers chapitres, ils décrivent nos propositions et nos contributions :

- Le chapitre 4 intitulé "**Vérification de l'atteignabilité d'un état en B**" détaille l'approche qui permet l'extraction de traces d'exécution vérifiant l'atteignabilité d'un état. Il montre également la complémentarité des deux techniques que nous proposons pour la mise en œuvre de notre approche en recensant les points forts et les limites de chacune des deux techniques.
- Le chapitre 5 intitulé "**Extraction des scénarios d'attaques internes**" est consacré à la description de l'approche permettant d'exhiber des attaques internes. Il donne d'emblée la définition formelle d'un scénario d'attaque. En outre, il montre comment étendre notre approche de vérification de l'atteignabilité pour l'extraction des différents types d'attaques internes.
- Le dernier chapitre intitulé "**Outil et expérimentations**" décrit notre outil GenISIS. Basé sur les différentes expérimentations menées, il présente des mesures de performance de l'outil comme il propose des techniques d'amélioration. Finalement, il présente une étude de cas complète.

Publications issues de ce travail de thèse

(A) **Revues internationales**

1. Amira Radhouani, Akram Idani, Yves Ledru, Narjes Ben Rajeb, 2015, Symbolic Search of Insider Attack Scenarios from a Formal Information System Modeling. *Trans. Petri Nets and Other Models of Concurrency*, LNCS vol.10,p. 131-152 (2015).

(B) **Revues nationales**

1. Akram Idani, Yves Ledru and Amira Radhouani. Modélisation graphique et validation formelle de politiques RBAC en Systèmes d'Information. *Ingénierie des Systèmes d'Information*, vol. 19(6), p. 33-61 (2014)

(C) **Conférences et workshops internationaux avec comité de sélection**

1. Amira Radhouani, Akram Idani, Yves Ledru, Narjes Ben Rajeb : Extraction of Insider Attack Scenarios from a Formal Information System Modeling. *Proceedings of the Formal Methods for Security Workshop co-located with the PetriNets-2014 Conference* , Tunis, Tunisia, June 23rd 2014, p. 5-19.

(D) **Conférences et workshops nationaux avec comité de sélection**

1. Amira Radhouani, Akram Idani, Yves Ledru, Narjes Ben Rajeb. GenISIS : un outil de recherche d'attaques d'initié en Systèmes d'Information. *Acte de la conférence francophone AFADL 2016 (Approches Formelles pour l'Assistance au Développement de Logiciels)*.

(E) **Publication en cours de soumission dans une revue internationale**

Amira Radhouani, Akram Idani, Yves Ledru, Narjes Ben Rajeb. B Formal Reasoning about Reachability in Information Systems and Application to Threat Identification.

Première partie

État de l’art

Chapitre 1

Validation formelle des modèles de contrôle d'accès basés sur les rôles

« Security is an illusion, but it is a pleasant one. »

James Rozoff

Sommaire

1.1 Fondements du modèle de contrôle d'accès basé sur les rôles 12
1.1.1 Noyau du modèle RBAC 13
1.1.2 RBAC hiérarchique 14
1.1.3 Les contraintes de séparation de devoirs 14
1.1.4 Avantages et inconvénients du modèle RBAC 15
1.2 Extension du modèle RBAC 15
1.3 Vulnérabilités des modèles de contrôle d'accès 17
1.3.1 Classification des attaquants initiés 18
1.3.2 Classification des attaques internes 19
1.4 Vérification et validation du contrôle d'accès 20
1.4.1 Méthodes de V&V formelles 21
1.4.2 Panorama des approches existantes pour la validation du contrôle d'accès 23
1.5 Conclusion 29

Le contrôle d'accès est un mécanisme indispensable pour préserver les trois propriétés fondamentales de la sécurité informatique : la confidentialité, l'intégrité et la disponibilité. Il est exprimé par un ensemble de règles permettant de contraindre l'accès aux différentes ressources du système.

Un grand nombre de modèles de politique de contrôle d'accès ont été proposés en s'inspirant des deux modèles de base qui sont les modèles de contrôle d'accès discrétionnaires (DAC) [LAMPSON 1974] et les modèles de contrôle d'accès obligatoires (MAC) [BELL et LAPADULA

1976, DENNING 1976]. En effet, à partir de ces deux modèles, plusieurs autres modèles ont été dérivés afin de répondre à des exigences spécifiques requises pour les organisations qui les mettent en œuvre. Parmi ces modèles, nous nous intéressons aux modèles adaptés à l'expression des règles de contrôle d'accès dans les SI. Nous considérons, en particulier, le modèle RBAC qui a été un succès et a été adopté par plusieurs grandes organisations.

Dans ce chapitre, nous détaillons les fondements de ce modèle de contrôle d'accès. Nous discutons ses limites ainsi que le besoin de l'étendre pour exprimer certaines contraintes de sécurité.

Nous montrons également que parfois la définition d'une politique de contrôle d'accès n'est pas suffisante pour garantir la sécurité du SI. En effet, même si les règles de sécurité semble être bien établies, certaines failles peuvent être mises en évidence via des scénarios qui n'introduisent aucune violation de contraintes de sécurité, mais qui permettent de montrer que la politique de sécurité peut être contournée de façon malicieuse. Ces failles donnent lieu à des attaques, dites attaques internes, dont les conséquences peuvent être non négligeables. Nous présentons, ainsi, ces attaques, nous les classifions et nous caractérisons les types d'attaque qui nous intéressent.

La dangerosité de ces attaques a poussé la communauté scientifique à développer plusieurs méthodes de vérification et de validation formelles en vue d'anticiper et de corriger les failles tôt dans le cycle du vie du SI. Dans la dernière partie de ce chapitre, nous passons en revue ces méthodes, nous comparons et nous critiquons différentes approches, en particulier, pour la spécification et la validation des modèles RBAC.

1.1 Fondements du modèle de contrôle d'accès basé sur les rôles

Le modèle RBAC a été proposé par [FERRAIOLO et KUHN 1992] en s'inspirant de la structure et des fonctions des utilisateurs (*i.e leurs rôles*) au sein des organisations pour structurer les droits d'accès d'une manière assez intuitive et facile à administrer. Le modèle RBAC intercale une entité intermédiaire, qui est le rôle, entre les utilisateurs et leurs permissions. Ainsi, chaque utilisateur est affecté à un ensemble de rôles auxquels on associe les permissions. Un utilisateur se voit, donc, attribuer les droits d'accès accordés aux rôles qu'il joue.

Ce modèle, standardisé par l'*American National Standard Institute* (ANSI) [FERRAIOLO et al. 2001], a pu se développer pour répondre aux exigences accrues et pour s'adapter à la complexité croissante des systèmes. Par conséquent, plusieurs variantes du modèle RBAC ont été proposées par l'ANSI. La première variante contenait le noyau du modèle RBAC (*core RBAC*) qui inclut une collection minimale des éléments requis pour la mise en place d'un contrôle d'accès basé sur les rôles. Puis une deuxième variante est apparue nommée *RBAC hiérarchique (Hierarchical RBAC)* qui permet d'introduire la notion de hiérarchie entre les rôles. Deux autres variantes ont été introduites par la suite permettant de définir des contraintes de séparation de devoirs entre les rôles : la Séparation Statique des devoirs – ou *Static Separation of Duty* (SSD) et la Séparation Dynamique des Devoirs – ou *Dynamic Separation of Duty* (DSD) (Figure 1.1).

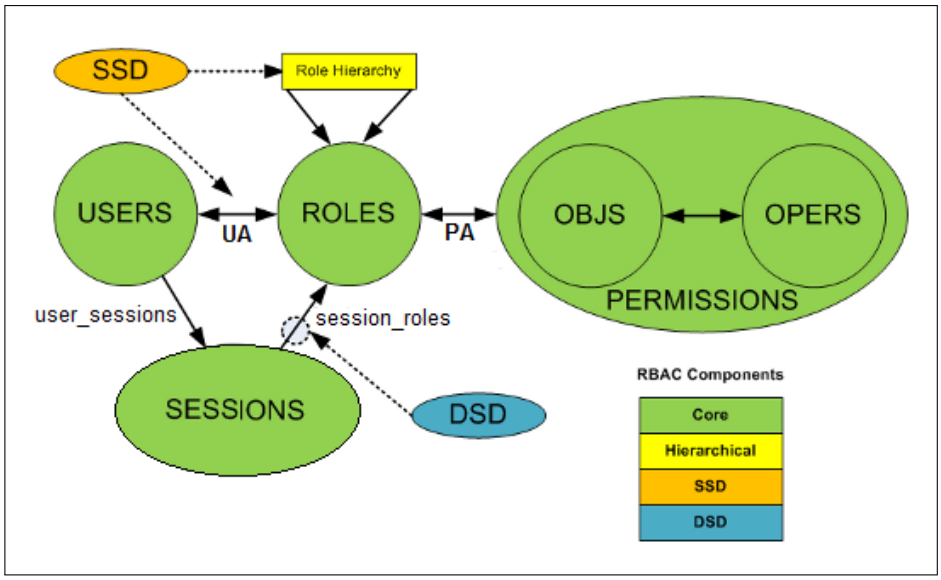


FIGURE 1.1 – Le modèle RBAC [FERRAIOLO et al. 2001]

1.1.1 Noyau du modèle RBAC

Le noyau du modèle RBAC se base sur six notions clés :

- **USERS** : Ce sont les utilisateurs autorisés à accéder au système. Ils peuvent être des personnes physiques ou des processus agissant à la place des personnes notamment des machines, des réseaux ou des agents autonomes intelligents.
- **ROLES** : Le rôle est le concept central du modèle RBAC, il correspond à une fonction ou à une responsabilité au sein de l'organisation. Chaque utilisateur du système doit être affecté à au moins un des rôles via la relation notée *UA* (*User Assignment*), telle que $UA \subseteq USERS \times ROLES$.
- **OBJECTS** (*OBJ*) : Les objets sont les ressources accessibles par les utilisateurs selon les règles de contrôle d'accès. Ils peuvent être des fichiers, des ressources matérielles, ou des entités d'une base de données.
- **OPERATIONS** (*OPS*) : Les opérations sont les actions offertes par le système qui permettent aux utilisateurs d'exécuter des fonctions sur les objets. Ces opérations peuvent être de type lecture, écriture ou exécution dans le cadre d'un système de fichiers, ou de type lecture, création, suppression et modification dans le cadre d'un Système d'Information ou d'un système de gestion de base de données.
- **PERMISSIONS** : Les permissions représentent l'ensemble des opérations autorisées pour chaque objet. Elles sont définies par la relation *PERMS* telle que $PERMS = \mathbb{P}(OBJS \times OPS)$ [1]. Les permissions sont affectées aux rôles via la relation notée *PA* (*Permission Assignment*) qui permet de spécifier les opérations autorisées pour chaque objet en fonction des rôles, telle que $PA \subseteq PERMS \times ROLES$.

1. $\mathbb{P}(E)$ dénote l'ensemble des sous ensembles de E

- **SESSIONS** : Les sessions représentent l'aspect dynamique du modèle RBAC. Elles peuvent être assimilées au mécanisme de connexion des utilisateurs au système. En effet, un utilisateur se connecte au moyen d'une session, dans laquelle il active un ensemble de rôles parmi ceux qui lui sont affectés. L'ensemble de rôles activés par un utilisateur u en même temps dans une session s est donné par la relation $session\_roles$ telle que $session\_roles \in SESSIONS \rightarrow \mathbb{P}(ROLES)$. Un même utilisateur u a aussi la possibilité d'ouvrir plusieurs sessions en même temps, d'où l'utilité de la relation $user\_sessions$ qui permet de lister les sessions en cours pour un utilisateur donné telle que $user\_sessions \in USERS \rightarrow \mathbb{P}(SESSIONS)$.

1.1.2 RBAC hiérarchique

L'idée d'introduire le concept de hiérarchie entre les rôles découle de la hiérarchie naturelle qui existe dans les organisations. D'une manière générale, un supérieur hiérarchique dans une entreprise a tous les droits de ses subordonnés, d'où la notion d'héritage de droits d'accès dans le modèle RBAC. En effet, si un rôle r_1 est hiérarchiquement supérieur à un rôle r_2 (noté $r_1 \succeq r_2$), alors r_2 hérite toutes les permissions de r_1 en plus de ses propres permissions. La notion d'héritage est exprimée par la relation RH tel que $RH \subseteq ROLES \times ROLES$.

En présence de la hiérarchie de rôles, l'ensemble des utilisateurs pouvant bénéficier des permissions associées à un rôle r est donné par la relation $authorized\_users$ telle que :

$$authorized\_users \in ROLES \rightarrow \mathbb{P}(USERS)$$
$$authorized\_users[\{r\}] = \{u \in USERS | r' \succeq r, (u, r') \in UA\}$$

Quant à l'ensemble des permissions pour un rôle r, il est donné par la relation $authorized\_permissions$ telle que :

$$authorized\_permissions \in ROLES \rightarrow \mathbb{P}(PERMS)$$
$$authorized\_permissions[\{r\}] = \{p \in PERMS | r' \succeq r, (p, r') \in PA\}$$

1.1.3 Les contraintes de séparation de devoirs

La notion de séparation de devoirs a été introduite dans le modèle RBAC pour empêcher les fraudes liées à l'abus de pouvoir des utilisateurs et pour empêcher d'instaurer des conflits d'intérêt. Ceci se traduit par le fait de restreindre les rôles d'un même utilisateur et surtout d'empêcher son affectation à des rôles conflictuels. Par exemple dans un projet logiciel, les rôles *Testeur* et *Développeur* sont conflictuels car il est souvent préconisé que le testeur d'un programme ne soit pas la personne qui l'a développé. Ces contraintes assurent qu'un utilisateur ne peut commettre des attaques majeures que s'il collabore avec d'autres utilisateurs ayant d'autres privilèges. Dans [FERRAIOLO et al. 2001], on distingue deux types de contraintes de séparation de devoirs :

- Les séparations statiques SSD : Par le biais de cette contrainte on empêche qu'un utilisateur soit affecté à un ensemble de rôles conflictuels. Elle est définie par la relation SSD telle que $SSD \subseteq (\mathbb{P}(ROLES) \times \mathbb{N})$. C'est une collection de paires (rs, n), où rs est un ensemble de rôles et n un entier naturel ≥ 2. Elle peut porter aussi bien sur la relation UA que sur la

relation *RH*. Dans le premier cas, elle garantit qu'aucun utilisateur n'est affecté à *n* rôles ou plus parmi l'ensemble *rs*. Tandis que dans le deuxième cas, elle permet d'empêcher qu'une hiérarchie de rôles amène un utilisateur à posséder les permissions d'un ensemble de rôles en conflit.

- Les séparations dynamiques DSD : Porte sur la relation *session_roles* et permet d'empêcher un utilisateur de jouer des rôles en conflit dans une même session. Cette contrainte est définie par la relation *DSD* telle que $DSD \subseteq (\mathbb{P}(ROLES) \times \mathbb{N})$. Contrairement à la SSD, la DSD autorise qu'un utilisateur soit affecté à *n* rôles conflictuels. Toutefois, il ne peut pas bénéficier des permissions de ces rôles en même temps. Il ne peut, donc, pas activer des rôles conflictuels dans une même session.

1.1.4 Avantages et inconvénients du modèle RBAC

Le modèle RBAC a connu un grand succès et a été adopté par de grandes structures telles que IBM, la NSA et de grands SI bancaires comme Desender Bank [SCHAAD et al. 2001]. Il a été également appliqué par des systèmes d'exploitation modernes tels que *Trusted Solaris*. Il a aussi été commercialisé par Microsoft sous la forme d'un logiciel nommé *Authorization Manager* qui permet d'appliquer le modèle RBAC aux systèmes d'exploitation comme *Windows 7*. De même il a connu un grand succès parmi les systèmes de gestion de base de données comme *Oracle 9* et *Sybase Adaptive Server*. Ce succès vient sans doute de l'avantage majeur que présentent ces modèles notamment la facilité d'administration et la facilité d'adaptation aux fluctuations dans les entreprises. En effet, lorsqu'un nouvel utilisateur rejoint le système, l'administrateur n'a qu'à lui affecter les rôles qui lui correspondent. De même, quand un utilisateur change de fonction ou de responsabilité l'administrateur n'a qu'à changer ses rôles.

Cependant, plusieurs critiques ont été adressées aux modèles basés sur les rôles [KALAM et al. 2003]. En effet, les règles de contrôle d'accès que peut prendre en compte un modèle RBAC sont assez restreintes et se limitent à l'expression de simples autorisations d'accès aux ressources. Il est donc impossible d'exprimer des interdictions d'accès, comme il est difficile d'exprimer des permissions spécifiques telles que les permissions qui dépendent du contexte. Par exemple, dans un SI bancaire, il ne serait pas facile d'interdire aux clients d'accéder aux comptes des autres clients ou des les autoriser à accéder uniquement à leur propre compte.

1.2 Extension du modèle RBAC

Afin de surmonter les inconvénients du modèle RBAC de base, plusieurs travaux ont suggéré des extensions au noyau de RBAC pour mieux répondre aux besoins spécifiques des entreprises. La délégation de rôle [CRAMPTON et KHAMBHAMMETTU 2008], les notions d'équipe [THOMAS 1997], d'organisation [KALAM et al. 2003], de temps [BERTINO et al. 2001], de workflow [WAINER et al. 2003] ou encore la notion de contexte [KULKARNI et TRIPATHI 2008] reflètent de telles extensions.

Étant conscients des limites du modèle RBAC de base notamment l'impossibilité d'exprimer des règles dépendant de contextes, nous nous sommes inspirée des extensions proposées et nous adoptons un modèle basé sur les rôles qui permet de prendre en compte des contraintes d'autorisation appelées aussi des contraintes contextuelles - *ou authorisation constraints* [LODDERSTEDT et al. 2002]. Il s'agit de contraintes rattachées aux permissions qui expriment un contexte d'autorisation ou une restriction d'accès. Par exemple, autoriser l'accès d'un client à son propre compte bancaire et lui interdire d'accéder aux comptes des autres clients est possible par le moyen des contraintes d'autorisation. Il est également possible d'autoriser une infirmière à consulter les dossiers médicaux des patients seulement en cas d'urgence par le biais de ces contraintes. En effet, ces contraintes sont des règles de sécurité relatives à des situations fonctionnelles qui peuvent évoluer. Par conséquent, en présence de ces contraintes, l'attribution des permissions aux utilisateurs se fait d'une manière dynamique selon l'état fonctionnel du système.

La notion de contrainte d'autorisation étant centrale dans nos travaux, nous précisons dans la définition 1 ce que nous entendons par ce terme.

Définition 1 (Contrainte d'autorisation)**.** Une contrainte d'autorisation est une règle de contrôle d'accès rattachée à une permission. Elle permet de restreindre le domaine de cette permission en imposant une condition de validité. Elle est exprimée par un prédicat logique, souvent dépendant des données fonctionnelles du système, et n'autorise l'accès à la ressource protégée par la permission que lorsque ce prédicat est satisfait.

Pour la prise en compte de telles contraintes nous proposons de modifier le schéma global du modèle RBAC en incluant la nouvelle relation *CC* (*Contextual Constraint*) telle que $CC \subseteq PERMS \times \mathbb{C}$ où $\mathbb{C}$ est un ensemble de prédicats exprimant les contraintes (Figure 1.2).

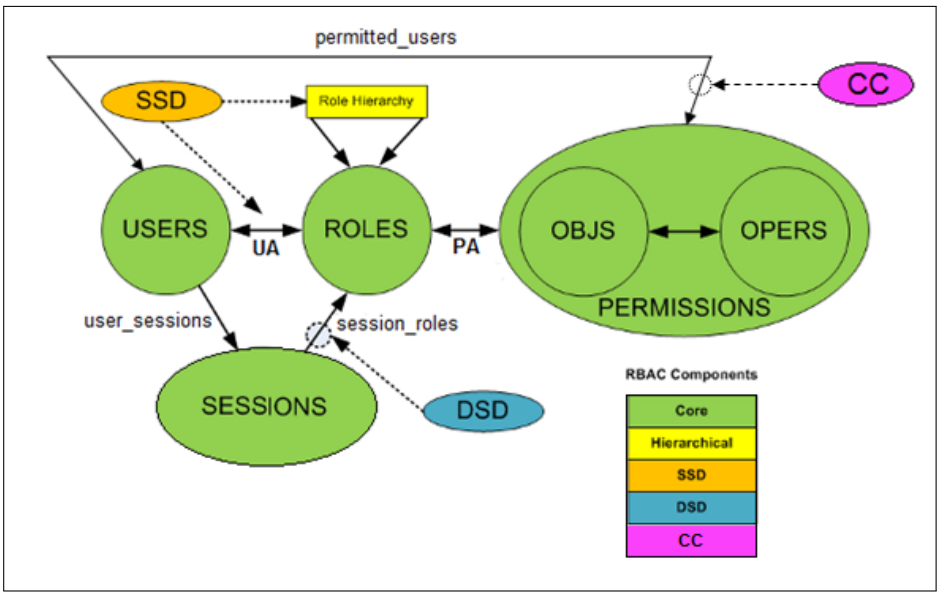


FIGURE 1.2 – Le modèle RBAC avec contraintes d'autorisation

Au modèle RBAC proposé dans [FERRAIOLO et al. 2001], nous rajoutons la relation *permitted_users*

qui permet de calculer l'ensemble des utilisateurs autorisés pour chaque permission en prenant en compte les contraintes d'autorisation en plus des rôles de chaque utilisateur.

Etant donné que l'évaluation de la contrainte d'autorisation dépend de l'état fonctionnel du système, cet ensemble est calculé dynamiquement à chaque évolution de l'état fonctionnel. Formellement, nous définissons cette relation comme suit :

$$Permitted\_users \in PERMS \rightarrow \mathbb{P}(USERS)$$

Telle que pour un état donné S, l'ensemble des utilisateurs autorisés à accéder à l'ensemble des ressources protégées par une permission p est donné par la formule suivante :

$$\begin{aligned} Permitted\_users[\{p\}] = \{u \in USERS \mid \quad & \exists r \in ROLES \cdot (\quad p \in authorized\_permissions[\{r\}] \wedge \\ & \qquad u \in authorized\_users[\{r\}]) \wedge \\ & \forall c \cdot (c \in CC[\{p\}] \Rightarrow Eval(c,S) \widehat{=} true)\} \end{aligned}$$

où $Eval(c,S)$ dénote l'évaluation de la contrainte d'autorisation c à l'état S du SI.

1.3 Vulnérabilités des modèles de contrôle d'accès

L'évolution de modèles de contrôle d'accès a certes permis de mieux répondre aux besoins de sécurité accrus des entreprises et de mieux assurer la confidentialité et l'intégrité des informations sensibles. Cependant, cela n'a pas empêché les usagers malicieux de développer eux aussi leurs moyens afin de commettre des actes illicites. Par le biais de ces actes, ils tentent de contourner les règles de contrôle d'accès dans le but d'altérer ou de détruire les données voire parfois de mettre hors d'usage le système. Ces usagers malicieux peuvent être des pirates, communément appelés *hackers*, qui cherchent à s'infiltrer dans le système soit en cassant les clés de chiffrement soit en exploitant une faille technique dans le réseau ou dans le code source de l'application. Les attaques commises par ce genre d'attaquants, appelées ***attaques externes***, constituent la première préoccupation des entreprises qui cherchent à protéger leurs systèmes. C'est ainsi qu'ils investissent beaucoup pour la mise en place de pare-feux et de mécanismes de détection d'intrusion, le cryptage des données, la redondance matérielle et logicielle, etc.

Mais, même si ces dispositifs garantissent la sécurité contre les attaques externes, ils s'avèrent insuffisants pour assurer la sécurité du système. En effet, plusieurs vulnérabilités donnant lieu à des attaques en SI sont dues à la logique même de la politique de sécurité plutôt qu'aux mécanismes d'implémentation. Les personnes malicieuses qui exploitent ces failles sont souvent des utilisateurs de confiance ayant des droits d'usage légitimes du système. Par conséquent, même si une politique de contrôle d'accès semble structurellement correcte de par l'affectation des utilisateurs aux bons rôles avec des permissions clairement identifiées, certaines attaques, dites ***attaques internes*** connues aussi sous le nom d'***attaque d'initié*** - ou ***insider attacks*** en anglais-, sont le fruit de l'exécution d'une séquence d'opérations autorisées. Ce type d'attaque est souvent négligé et beaucoup moins étudié malgré sa dangerosité et bien qu'il puisse causer des dommages considérables. Ceci est dû au fait que l'attaquant initié dispose déjà d'un accès au système et a une parfaite connaissance de la politique de contrôle d'accès.

C'est vers ce genre d'attaques que notre intérêt se tourne dans le cadre de cette thèse. Notre intérêt portera plus particulièrement sur la détection des défauts de conception des SI et des règles de contrôle d'accès qui rendent possible l'exécution de séquences d'opérations révélant des comportements malicieux et pouvant être sources d'attaques internes. Nos centres d'intérêt qui s'inscrivent dans ce contexte étant bien ciblés, il convient de définir au préalable ce que nous entendons par attaquant initié ainsi que de fixer les types d'attaques que nous cherchons à identifier.

1.3.1 Classification des attaquants initiés

Récemment, les attaques internes ont pris de l'ampleur et se sont multipliées pour devancer les attaques externes. En effet, l'étude statistique faite par [CLEARSWIFT 2013] a montré que 58% des attaques proviennent des individus internes à l'organisation, majoritairement des employés mais aussi des ex-employés agissant par vengeance ou des tierces personnes exerçant une pression ou manipulant des employés.

Selon leurs intentions, les attaquants initiés - ou *insiders* en anglais - peuvent être classés en deux grandes catégories [NURSE et al. 2014]. Nous distinguons, ainsi, les utilisateurs sans arrière-pensées des utilisateurs commettant des actes malicieux intentionnellement.

En effet, la première catégorie des utilisateurs met le système en péril en agissant soit par ignorance ou par négligence. Ce genre d'attaque, souvent lié à l'erreur humaine, constitue le principal facteur de violation de la sécurité dans les entreprises [TEAM 2013]. Oublier de fermer une session de travail avant de quitter, mordre à l'hameçon d'un site ou d'un logiciel de phishing ou divulguer naïvement des informations professionnelles sur les réseaux sociaux constituent des exemples concrets de menaces causées par des initiés accidentellement.

Quant à la deuxième catégorie des utilisateurs, elle profite de ses accès privilégiés à des fins inappropriées, que ce soit personnelles ou financières. Un des exemples récent est l'exemple du trader de la banque *Société générale Jérôme Kerviel* qui, grâce à sa très bonne connaissance des procédures de contrôle interne, parvint à masquer l'importance et le risque élevé des opérations qu'il avait faites sur le marché en introduisant dans le système informatique des opérations inverses fictives les compensant ce qui a provoqué une perte de $7.2 milliards à la banque [TIMES 2008]. La société *Fannie Mae* a aussi failli perdre toutes ses données financières, sécuritaires et stratégiques à cause de son ex-employée *Makwana* qui a été licenciée et qui a voulu se venger. Elle a donc implanté un script malveillant qu'elle a réussi à transmettre à son ancien ordinateur en vue de le propager dans tout le réseau de la société et détruire toutes sortes de données [FBI 2010].

Dans notre investigation, nous nous intéressons à cette deuxième catégorie d'utilisateurs que nous classons à son tour en deux sous-catégories. En effet, selon le nombre d'attaquants impliqués, nous distinguons les attaques réalisées par un seul initié, des attaques perpétrées grâce à une collusion entre deux ou plusieurs utilisateurs. Ces dernières sont plus dangereuses et plus facilement réalisables surtout si la collaboration implique des utilisateurs jouant différents rôles avec des privilèges différents. En effet, certains mécanismes de contrôle d'accès tels que la séparation des devoirs peuvent être mis en place pour limiter les droits d'accès individuels et par la suite minimi-

ser le risque des attaques commis par un seul utilisateur. Néanmoins, il est très difficile d'éviter les attaques collaboratives. Nous citons à titre d'exemple le cas d'un chef de projet dans une société de e-commerce. Suite à son licenciement, il s'est vengé en supprimant le projet sur lequel il a travaillé et qui a été évalué à $2.6 million. Il a réussi à accomplir sa mission avec l'aide d'un complice qui lui a donné le mot de passe du serveur stockant le projet [SILOWASH et al. 2012].

1.3.2 Classification des attaques internes

Les attaques internes ont été classées dans [BELLOVIN 2008] en trois types fondamentaux :

- **Détournement d'accès :** On dit qu'un utilisateur détourne un accès s'il utilise ses droits d'accès légitimes à des fins illicites différentes des objectifs principaux de sa fonction pour lesquels ces droits d'accès lui ont été attribués. Par exemple, un SI d'une université peut donner le droit de modification des notes des étudiants aux professeurs et ce dans le but de pouvoir corriger certaines erreurs ou intervenir dans des situations exceptionnelles. Mais un professeur malveillant peut abuser de ce droit pour fausser les notes.
- **Échec de contrôle d'accès :** Ces attaques peuvent avoir lieu suite à des problèmes techniques provoquant la désactivation ou le dysfonctionnement des mécanismes de contrôle d'accès. Certains utilisateurs au sein de l'organisation peuvent saisir l'occasion pour commettre des actes inappropriés.
- **Contournement de contrôle d'accès :** Les utilisateurs légitimes du SI ont souvent une connaissance des stratégies de sécurité mises en place par l'organisation. Ils sont donc bien placés pour contourner les droits d'accès et pour gagner de nouveaux privilèges dans le but d'accéder à des ressources qui leur sont interdites initialement. Par exemple, dans un SI hospitalier une infirmière n'a le droit d'accéder en lecture aux dossiers médicaux des patients qu'en cas d'urgence. Étant consciente de cette règle de contrôle d'accès, une infirmière malveillante peut s'arranger pour provoquer une situation d'urgence afin de s'octroyer le droit d'accès aux informations confidentielles d'un patient.

Les deux premiers types d'attaques sont difficiles à déceler et ne dépendent pas de la logique de politique de sécurité, nous nous intéressons dans le cadre de cette thèse au troisième type d'attaque qui est souvent la conséquence d'une faille dans le modèle de sécurité. Dans le modèle RBAC que nous adoptons, nous différencions quatre types de contournement de contrôle d'accès que nous nommons respectivement attaque basique, attaque à contrainte, attaque planifiée et attaque cachée.

- **Attaque basique** -ou *basic attack* en anglais- : L'attaquant cherche à contourner des règles de sécurité pour accéder à des ressources qui ne lui sont pas autorisées de par les rôles qui lui sont affectés. Par exemple, une infirmière qui accède en écriture au dossier médical d'un patient.
- **Attaque à contrainte** - ou *constrained attack* en anglais- : Le contournement du contrôle d'accès peut être favorisé par les règles de sécurité exprimées au moyen de contraintes d'autorisation telles qu'elles sont définies dans la définition 1. En effet, comme ces contraintes

dépendent de l'état fonctionnel du SI, les attaquants peuvent faire évoluer le système pour le ramener d'un état où la contrainte d'autorisation est fausse vers un état où elle devient vraie, et ce en exécutant une suite d'opérations autorisées. Par conséquent, l'attaquant peut avoir les rôles qui lui donnent la permission d'accès à la ressource cible mais la contrainte d'autorisation appliquée à cette permission l'empêche d'y accéder. L'exemple de l'infirmière qui provoque une situation d'urgence constitue un parfait un exemple d'une attaque à contrainte. Même si nous pouvons considérer que cette attaque n'est pas due à une faille dans le système qui doit être corrigée, l'identification de tels comportements malicieux est importante. En effet, si nous considérons que l'intervention et la prise en charge du patient en cas d'urgence est plus prioritaire, nous ne pouvons pas imposer plus de restrictions sur les droits d'accès d'une infirmière. Cependant, ces scénarios doivent être contrôlés et des mécanismes de traçabilité doivent être mis en œuvre pour prévenir ce genre d'attaques.

- **Attaque planifiée** - ou *planned attack* en anglais- : Certaines attaques requièrent l'accès planifié à des ressources différentes non autorisées dans un ordre bien particulier pour pouvoir accomplir l'acte malicieux. Par exemple, dans une entreprise, il n'est possible de passer d'un grade à un autre que si la fourchette salariale du grade en question est atteinte. De plus, les employés n'ont pas le droit d'accéder à ces informations ni en lecture ni en écrire. Ainsi, un employé malicieux, qui veut modifier son grade, doit planifier une attaque en prévoyant les étapes suivantes : il doit, premièrement, accéder en lecture à la fourchette salariale du grade voulu, puis modifier son salaire et finalement mettre à jour son grade.
- **Attaque masquée** - ou *hiden attack* en anglais- : Une attaque est masquée si l'attaquant parvient à faire des opérations inverses qui compensent les modifications introduites dans le système suite à son attaque. L'attaquant parvient, ainsi, à remettre le système dans son état d'origine. Par exemple, un médecin malicieux change l'affectation d'un patient pour pouvoir accéder à ses informations confidentielles qui ne sont accessibles que pour les médecins qui le suivent. Pour masquer son acte malicieux, il remet l'affectation à son état d'origine après avoir eu les informations qu'il voulait.

Pour pouvoir identifier ces attaques, une analyse approfondie du modèle de sécurité ainsi que de l'impact de l'évolution dynamique des états du SI sur les règles de contrôle d'accès est nécessaire. En effet, la dangerosité de ces attaques réside dans le fait qu'elles ne violent pas directement les contraintes de sécurité, mais qu'elles tirent profit de la possibilité d'exécution d'une séquence d'opérations autorisées pour contourner malicieusement les règles de sécurité. Ainsi, dans le cadre de la vérification et de la validation des règles de contrôle d'accès, nous nous intéressons, en particulier, à l'analyse comportementale du SI en vue d'extraire ce genre de comportements malicieux.

1.4 Vérification et validation du contrôle d'accès

La criticité du domaine de la sécurité informatique en général et du contrôle d'accès en particulier demande d'avoir recours à des techniques rigoureuses permettant la vérification de la cohérence et de la correction de la politique de sécurité. Par conséquent, avoir une base mathématique

et l'assistance d'outils pour l'automatisation de V&V sont des critères incontournables pour le choix de la méthode la plus adéquate pour la spécification des politiques de sécurité. C'est pour cette raison que les méthodes de spécification formelles s'imposent dans ce domaine particulier.

1.4.1 Méthodes de V&V formelles

Les méthodes formelles ont tenté d'apporter un cadre précis permettant de raisonner rigoureusement en s'appuyant sur un langage doté d'une syntaxe bien définie et d'une sémantique basée sur des concepts mathématiques bien établis. Ces méthodes permettent par ailleurs d'assurer un niveau élevé de précision et de cohérence et d'augmenter la confiance que l'on a en un système. Ainsi, afin de garantir une bonne qualité de développement, un éventail de langages et de techniques formels sont apparus. Certains sont spécifiques à un domaine bien particulier et d'autres sont plus généraux et s'appliquent à plusieurs domaines. Une classification de ces langages a été proposée dans [GERVAIS 2006] qui différencie les langages basés sur les états des langages basés sur les événements.

- **Langages basés sur les états :** Ces langages construisent, à l'aide de concepts mathématiques telles que la théorie des ensembles et la logique des prédicats, un modèle qui décrit les propriétés du système à développer en privilégiant les données pour une expression explicite des états statiques du système. Ils permettent également de modéliser la partie dynamique du système par l'intermédiaire des opérations ou des actions qui expriment les changements d'états et les transitions que le système peut effectuer. De nombreux langages basés sur les états existent dans la littérature tels que Z [SPIVEY 1989], VDM [JONES 1990], B [ABRIAL 1996a], TLA [LAMPORT 2002], Alloy [JACKSON 2002], etc.
- **Langages basés sur les événements :** Ces langages sont basés sur l'axiomatisation qui décrit le comportement du système. Ce comportement est souvent spécifié à l'aide d'une algèbre ou d'une séquence décrivant les relations entre les opérations ou les actions du système. Nous distinguons parmi eux les algèbres de processus comme CSP [HOARE 1978], pi-calcul [MILNER 1999], CCS [MILNER 1982] et les formalismes à base d'automate tels que les Grafcet et les réseaux de petri [DAVID et ALLA 1992].

Les méthodes formelles tirent tout leur intérêt des modèles mathématiques qui permettent de raisonner rigoureusement afin de prouver la conformité de ces modèles par rapport aux propriétés souhaitées. Ainsi, plusieurs techniques de V&V sont apparues, parmi lesquelles nous trouvons l'interprétation abstraite, le model-checking et la preuve.

- **L'interprétation abstraite :** Les techniques basées sur l'interprétation abstraite [COUSOT et COUSOT 2003] permettent de raisonner statiquement sur une approximation du système mettant l'accent sur la propriété qu'on souhaite vérifier et faisant une abstraction sur les autres problèmes. L'avantage principal de ces techniques est le fait qu'elles permettent de réduire la taille et l'espace des états quand les systèmes à vérifier sont très grands. Mais, il est difficile de les mettre en œuvre car le choix de l'abstraction qui doit rester fidèle au modèle concret n'est pas évident.

- **Le model-checking :** Cette technique [CLARKE et al. 1999] consiste à vérifier des propriétés par une énumération exhaustive des états accessibles. Il s'agit, entre autres, de vérifier que toutes les traces d'exécution d'un modèle sont incluses dans les traces autorisées par une propriété. Le cas échéant, un contre-exemple d'une séquence de transitions qui falsifie la propriété est généré. Ces techniques restent les plus efficaces et les plus sûres lorsqu'on aborde un système avec un nombre d'états fini. Cependant, on peut se heurter au problème de l'explosion combinatoire du nombre d'états lorsque le système à vérifier devient grand. Des approches de couplage de model-checking avec les techniques d'interprétation abstraite ont été proposées [CLARKE et al. 1994, MCMILLAN 1993], ce qui permet certes de réduire le problème de l'explosion combinatoire, mais malheureusement pas le problème de l'indécidabilité du choix de l'abstraction.
- **La preuve :** Les techniques basées sur la preuve [COOK 1971] sont fondées sur un ensemble d'axiomes et un ensemble de règles de raisonnement (modus ponens, récurrence, substitution, réécriture, etc.) dont l'objectif est de construire une preuve mathématique (un théorème) afin de démontrer si une propriété est vérifiée ou non. Le point fort de ces techniques est qu'elles raisonnent sur l'ensemble des traces d'exécution, de telle sorte que, si le théorème est démontré, il vaut pour la totalité du modèle. Par conséquent, la propriété est nécessairement vérifiée. Cependant, la difficulté de ces techniques réside dans la recherche d'une preuve permettant d'établir la propriété souhaitée. En outre, dans la plupart des cas, même si les théorèmes de la preuve sont spécifiés, ces derniers sont si complexes qu'il est difficile de les démontrer automatiquement. Pour cette raison, des outils de preuve semi-automatiques ou partiellement interactifs permettant à l'utilisateur d'intervenir pour guider la preuve ont été proposés.

D'autres techniques comme l'animation et le test peuvent être utilisées pour exhiber des erreurs. Mais, ces méthodes ne sont pas considérées seules comme des méthodes de V&V du moment où elles ne garantissent pas l'absence d'erreurs.

- **L'animation :** Les techniques d'animation [DUTLE et al. 2015] ou encore de simulation permettent de valider la spécification vis à vis d'un comportement attendu. Ces techniques permettent de jouer une trace d'exécution dans le but de vérifier le respect d'une propriété ou la possibilité d'exécution d'un scénario. Elles permettent, ainsi, de vérifier le respect d'une propriété sur un exemple ou l'existence d'erreurs mais pas d'en garantir l'absence. Néanmoins, elles permettent d'augmenter la confiance des utilisateurs envers la spécification en fournissant rapidement une vue de l'exécution du modèle. Elles sont souvent complétées par la preuve dans un deuxième temps pour vérifier le modèle dans sa totalité.
- **Le test :** Les tests [MYERS et SANDLER 2004] permettent aussi de détecter la présence d'erreurs à l'aide d'un jeu de tests souvent générés automatiquement à partir des spécifications. Ils ont pour but principal de montrer que le système satisfait les exigences en tentant de couvrir toutes les séquences d'utilisation spécifiées par les exigences avec des scénarios de tests positifs et négatifs. En effet, les tests positifs permettent de vérifier que le système se comporte comme prévu et donne le résultat attendu en lui fournissant des données valides. Quant

aux tests négatifs, ils permettent de vérifier que le système réagit correctement en présence des données invalides. Cependant, il est difficile de couvrir tous les scénarios d'exécution possibles. C'est pour cette raisons que le test est considéré comme une technique de vérification non exhaustive. Par ailleurs, il est toujours recommandé de combiner cette méthode avec des techniques de preuve ou de model-checking.

1.4.2 Panorama des approches existantes pour la validation du contrôle d'accès

Plusieurs travaux se sont intéressés à la validation de contrôle d'accès en général et du modèle RBAC en particulier. Selon le type de contraintes que ces travaux ont tenté de valider, nous proposons de les classer en deux catégories : validation de contraintes statiques et validation de contraintes dynamiques. En effet, les contraintes de contrôle d'accès statiques ne dépendent pas de l'état du système. Ainsi, les activités de validation de ces contraintes consistent à vérifier si, dans un contexte donné (utilisateur, rôle, etc.), l'exécution d'une action est permise ou pas . Quant aux contraintes de contrôle d'accès dynamiques, elles dépendent de l'évolution dynamique des états du système. Par conséquent, la permission de l'exécution d'une action dépend de l'évaluation de l'état courant et peut changer quand l'état change.

1.4.2.1 Travaux visant la validation statique

Certaines approches basées sur la méthode formelle Z ont été proposées pour la spécification et la validation des modèles RBAC comme celles proposées dans [ABDALLAH et KHAYAT 2006, YUAN et al. 2006]. Ces travaux proposent un modèle générique pour la représentation formelles des modèles RBAC. Toutefois, ils ne demeurent pas assez exploitables pour la validation automatique des modèles, car, malheureusement, la méthode Z manque d'outils qui supportent l'automatisation des activités de V&V.

Le langage Alloy, qui est proche du langage Z, a connu dans ce contexte plus de succès que son ancêtre grâce à son analyseur automatique *Alloy-Analyzer*. Par exemple, dans [ZAO et al. 2003], les auteurs ont proposé une technique basée sur *Alloy-Analyser* pour vérifier les caractéristiques algébriques du schéma RBAC ainsi que les incohérences entre les contraintes de sécurité. Dans le même contexte, [AHN et HU 2007, SCHAAD et MOFFETT 2002] se focalisent sur la vérification des contraintes de séparation de devoirs, et ils analysent la relation entre ces contraintes et la hiérarchie des rôles afin d'éliminer les éventuels conflits.

D'autres travaux dans le contexte de la validation des modèles RBAC ont eu recours à des méthodes semi-formelles basées sur le standard UML[OMG 2011]. Une des premières propositions est UMLsec [JÜRJENS 2002] qui a implémenté plusieurs stéréotypes dont le stéréotype "*rbac*" utilisé sur les diagrammes d'activité d'UML avec les étiquettes "*protected*", "*role*" et "*right*" dans le but de spécifier des règles RBAC. Pour les actions de V&V, UMLsec utilise un processus de simulation pour vérifier si chaque utilisateur a les permissions requises pour l'exécution des opérations sécurisées. Cependant, cette approche reste limitée pour la spécification des modèles RBAC du fait

qu'elle ne couvre pas tous les aspects tels que la hiérarchie des rôles, les contraintes de séparation de devoirs ainsi que les contraintes d'autorisation.

Une des approches concurrentes d'UMLsec est l'approche SecureUML qui est conçue spécialement pour la modélisation des politiques de sécurité à base de modèle RBAC. Elle permet une représentation structurelle des éléments liés au contrôle d'accès en étendant le diagramme de classe d'UML avec des stéréotypes permettant la spécification des différents aspects du modèle RBAC. La prise en compte de toutes les notions présentes dans les politiques RBAC ainsi que la possibilité d'exprimer les contraintes d'autorisation constituent l'avantage majeur de cette approche.

Pour la validation des modèles de sécurité exprimés en SecureUML, les auteurs dans [LODDERSTEDT et al. 2002] se servent de leur outil SecureMOVA. Ce dernier permet de créer un diagramme de classe fonctionnel puis de rattacher ses entités aux permissions. Il permet également d'exprimer les contraintes en *Object Constraint Language* (OCL). En outre, SecureMOVA offre la possibilité d'écrire des requêtes permettant de valider les règles de contrôle d'accès et ce en calculant les actions autorisées pour un rôle ou un utilisateur bien précis dans un état donné. Pour ce faire, ils utilisent la résolution de contraintes en intégrant un solveur SMT dans SecureMOVA.

Dans la même optique, les auteurs de [KUHLMANN et al. 2013] proposent une approche basée sur l'outil USE qui prend en entrée un diagramme d'objets UML et une description OCL des contraintes de sécurité et ce dans le but de vérifier automatiquement si le diagramme d'objets respecte ces contraintes. L'outil permet également de générer aléatoirement des séquences de diagrammes d'objets où les pré et les post-conditions exprimées par les contraintes OCL peuvent être vérifiées. Cette approche couvre presque toutes les notions du modèle RBAC notamment les utilisateurs, les rôles, les sessions, la hiérarchie des rôles et la séparation des devoirs. Elle permet également de prendre en compte des contraintes d'autorisation en rajoutant les attributs fonctionnels aux utilisateurs concernés. Malheureusement, cette approche engendre des duplications pour les informations déjà incluses dans le modèle fonctionnel, sans pour autant aborder la validation de l'évolution de l'état fonctionnel et son impact sur le modèle de sécurité.

La validation statique du modèle RBAC a été aussi abordée dans [FISLER et al. 2005] où les auteurs ont utilisé les arbres binaires de décision à terminaux multiples (MTBDD) pour la modélisation de la politique de sécurité. Ils ont eu recours par la suite à l'outil Margrave qu'ils ont développé pour interroger et analyser les schémas MTBDD.

Pour conclure, la validation des contraintes de contrôle d'accès statiques en considérant un état donné est, certes, indispensable et peut donner une certaine confiance dans la politique de sécurité. Cependant, la principale limitation de cette validation est qu'elle n'indique pas si l'état donné est accessible ou non. En outre, cette validation ne permet pas d'identifier les failles du modèle de sécurité qui peuvent être exploitées pour accomplir des attaques internes en faisant évoluer l'état fonctionnel du système.

Ceci dit, aucun des travaux présentés dans cette section n'a cherché à étudier l'évolution dynamique des états fonctionnels et leurs effets sur les contraintes de sécurité, bien que nous pensons que certains outils comme *Alloy-Analyser* ou SecureMova pourraient bien supporter de telles ana-

lyses.

1.4.2.2 Travaux visant la validation dynamique

La validation des contraintes dynamiques tente principalement d'identifier les stratégies suivies par les utilisateurs malveillants pour contourner certaines règles de sécurité en profitant de l'évolution dynamique des états du système. Dans ce contexte, une analyse dynamique a été proposée dans [LEDRU et al. 2011] où les auteurs proposent de modéliser une politique RBAC en utilisant le langage formel Z et de la valider en interagissant avec l'animateur *Jaza*. Cette approche étudie l'évolution des états fonctionnels et analyse l'impact de ces derniers sur le modèle de sécurité. Elle permet, ainsi, d'identifier des scénarios d'attaque malicieux qui contournent les règles de sécurité exprimées par des contraintes d'autorisation. Nous nous sommes inspirée de cette approche pour la définition des scénarios d'attaque et des comportements malicieux. Cependant l'inconvénient majeur de cette approche est le fait qu'elle soit basée sur l'animation. En effet, elle ne garantit pas l'absence d'erreur et requiert l'intervention de l'analyste pour guider l'animateur. En outre, cette approche n'a pas discuté la possibilité d'identification des stratégies faisant intervenir plusieurs utilisateurs en collusion.

[YU et al. 2009] propose une approche dans le but de détecter les violations des règles de contrôle d'accès dans le modèle RBAC. Pour ce faire, ils spécifient les contraintes d'activation de rôles, les relations hiérarchiques entre les rôles ainsi que les contraintes de séparation de devoirs en OCL. Par la suite, ils génèrent des scénarios sous la forme d'une séquence d'invocation des opérations pour les comparer avec les scénarios autorisés dans les modèles spécifiés. Leur approche peut aussi être supportée par des outils comme USE ou OCLE (*Object Constraint Language Environment*) [2] afin de vérifier des propriétés structurelles à partir des diagrammes de classes. Toutefois, cette approche est non exhaustive car il est difficile de générer et de tester toutes les séquences d'exécution du système.

Dans [MONDAL et SURAL 2008], les auteurs ont utilisé des automates temporisés pour la modélisation d'un modèle RBAC auquel ils rajoutent des contraintes temporelles qui servent à spécifier la disponibilité des rôles. Cette approche propose de construire un automate temporisé pour chacun des concepts de base à savoir les utilisateurs, les rôles et les permissions. Un automate appelé *automate contrôleur* est par la suite construit pour assurer la synchronisation entre ces différents éléments. Les auteurs ont également discuté la spécification des contraintes de nature temporelle qui sont exprimées par la logique modale CTL -*Computation Tree Logic*- [CLARKE et EMERSON 1982]. Pour la vérification de la cohérence du modèle de sécurité, les auteurs font appel au model-checking.

Les modèles RBAC avec contraintes temporelles ont été aussi adressés par [SHAFIQ et al. 2005] qui ont utilisé des réseaux de Pétri colorés pour la représentation du modèle. Une analyse d'accessibilité des états du réseau de Pétri est par la suite effectuée pour vérifier le respect des contraintes de sécurité. Cependant, aucun outil n'est utilisé pour la vérification de ces contraintes.

2. http://lci.cs.ubbcluj.ro/ocle/

Une approche similaire a été proposée dans [RAKKAY et BOUCHENEB 2009]. En effet, les auteurs proposent une description générique d'un modèle RBAC enrichi par des contraintes temporelles en utilisant les réseaux de Pétri colorés. Ils proposent également d'automatiser leur approche en tirant profit de l'outil CPN [JENSEN et al. 2007]. Cet outil offre une plate-forme complète pour la modélisation, la simulation ainsi que la génération et l'analyse de l'espace d'états des réseaux de Pétri colorés. Toutefois, aucun des travaux utilisant les automates et les réseaux de Pétri n'a abordé le problème de l'impact du modèle fonctionnel sur le modèle de sécurité.

D'autres approches ont été proposées pour la détection de scénarios malveillants en profitant de l'évolution dynamique des états du système, mais qui ne sont pas dédiées à un modèle de contrôle d'accès spécifique. Par exemple, dans [ZHANG et al. 2008], les auteurs ont proposé de modéliser les règles de contrôle d'accès dans un nouveau langage nommé RW (Read/Write). Ils ont proposé par la suite un algorithme de recherche avec retour en arrière afin d'extraire les chemins menant vers un état donné considéré comme indésirable. Cette approche permet ainsi de vérifier si un utilisateur seul ou un ensemble d'utilisateurs en coalition peut atteindre l'état spécifié et ce en combinant des opérations d'écriture et de lecture. Cependant, l'algorithme est automatisé en utilisant les techniques de pur model-checking. En outre, le langage RW est limité comparé à d'autres langages formels comme Z, Alloy et B. Par conséquent, il ne peut pas exprimer les comportements fonctionnels complexes.

Les auteurs dans [BECKER et NANZ 2010] ont aussi proposé une méthode proche de ce que nous proposons dans le présent travail. En effet, ils ont proposé une approche basée sur la preuve afin d'étudier l'atteignabilité d'un état cible satisfaisant certaines contraintes de sécurité. Ils se basent ainsi sur un système de preuve fondé sur la logique SMP (*Logic for State Modifying Policies*). Cette logique a été définie afin de raisonner sur un ensemble abstrait d'états cibles satisfaisant les contraintes de sécurité. Les auteurs expriment les propriétés d'accessibilité en termes de pré et post-conditions. Ils ont également mis en œuvre un algorithme avec retour en arrière pour l'extraction de la séquence d'opérations minimale menant à l'état cible défini. Cependant, le langage basé sur la logique SMP n'est pas adéquat pour la spécification des SI notamment pour la description des comportements fonctionnels complexes. De plus, l'abstraction faite et le raisonnement sur un modèle symbolique peut introduire une sur-approximation[ANAND et al. 2009] et produire des scénarios qui ne sont pas reproductibles dans le modèle concret.

1.4.2.3 Synthèse

Le tableau 1.1 synthétise et compare les différentes approches discutées tout au long de cette section. Nous nous basons principalement dans notre comparaison sur le langage utilisé pour la modélisation des règles de contrôle d'accès, ainsi que les outils et les techniques de V&V qui les supportent. Nous énumérons également les points forts et les points faibles de chaque approche.

En guise de conclusion, nous notons que malgré la panoplie des travaux existants pour la validation des modèles de contrôle d'accès en général et du modèle RBAC en particulier, peu sont les travaux qui ont abordé la validation dynamique dans le modèle RBAC en prenant en compte le lien entre le modèle fonctionnel et le modèle de sécurité.

L'approche proposée dans [LEDRU et al. 2011] est la seule qui a tenté d'étudier l'impact de l'évolution des états fonctionnels sur les règles de contrôle d'accès dans le modèle RBAC en analysant les règles exprimées au moyen des contraintes d'autorisation. Toutefois, le manque de support technique du langage formel Z limite cette approche et rend difficile la validation automatique des modèles de sécurité. Les autres approches basées sur des langages ou des logiques spécifiques tels que RW et SMP ou encore sur les méthodes formelles à base d'automates sont aussi limitées et ne permettent pas la prise en compte de comportements complexes des SI. En effet, nous sommes convaincue que les méthodes à base d'états qui représentent les systèmes par leurs données restent les plus adéquats pour la représentation des modèles des SI surtout si nous prenons en compte les données fonctionnelles dans la modélisation. Ceci a motivé notre choix de la méthode formelle B qui est aussi un langage basé sur les états mais dont les outils de preuve, d'animation et de model-checking sont bien développés.

Il est aussi à noter que la plupart des travaux qui ont fait la validation des contraintes dynamiques ont utilisé des techniques de model-checking dans leurs analyses ce qui pose le problème de l'explosion combinatoire du nombre d'états au moment du passage à l'échelle.

Approche	Méthode de modélisation	Outils/méthodes de validation	Avantages	Inconvénients
[LEDRU et al. 2011]	Z	Jaza/Animation	+ Etude de l'impact de l'évolution dynamique des états fonctionnels sur les règles de contrôle d'accès	- Approche interactive non automatisable
[YU et al. 2009]	UML/OCL	USE/OCLE	+ Modélisation intuitive basée sur UML	- validation non exhaustive
[MONDAL et SURAL 2008]	Automates temporisés, CTL	model-checking	+ Prise en compte des contraintes temporelles	- Langages de modélisation non adaptés au SI - Validation par pur model-checking - Pas de prise en compte du lien entre modèle fonctionnel et modèle de sécurité
[SHAFIQ et al. 2005]	Réseau de Pétri	Analyse manuelle		
[RAKKAY et BOUCHENEB 2009]	Réseau de Pétri	CPN/model-checking		
[ZHANG et al. 2008]	RW	model-checking	+ Détection des attaques avec coalition + Possibilité de détection des quatre types d'attaque internes	- Pauvreté du langage RW - Validation par pur model-checking
[BECKER et NANZ 2010]	SMP	Preuve	+ Approche basée sur la preuve	- Pauvreté du langage SMP - Possibilité d'extraction de fausses attaques

TABLEAU 1.1 – Synthèse des approches pour la modélisation et la validation des modèles de contrôle d'accès

1.5 Conclusion

Dans ce chapitre nous avons présenté les fondements du modèle RBAC et nous avons recensé les avantages d'un modèle RBAC étendu par des contraintes d'autorisation. Ces dernières, qui dépendent des états fonctionnels des SI offrent, certes, une certaine flexibilité pour l'expression de règles de contrôle d'accès de nature spécifique. Cependant, elles peuvent être à l'origine de certaines attaques internes où des utilisateurs légitimes malveillants tentent de faire évoluer les états fonctionnels du système en exécutant une séquence d'opérations autorisées en vue de s'octroyer de nouveaux droits d'accès.

Ces attaques, que nous avons classées en quatre types différents, sont souvent difficiles à identifier, et par conséquent l'utilisation de méthodes et de techniques d'analyse sophistiquées s'impose. En effet, afin d'extraire ce genre de comportements malveillants une analyse de l'impact de l'évolution dynamique des états fonctionnels du système sur les règles de contrôle d'accès est nécessaire. Toutefois, cet aspect de la validation des règles de contrôle d'accès est peu étudié dans la littérature. De plus, le peu de travaux qui ont évoqué cette problématique se basent, pour la plupart d'entre eux, sur les techniques de model-checking. C'est pour cette raison que nous proposons dans le présent travail une approche pour la validation des contraintes d'autorisation dynamiques, d'une part, et pour la réduction du problème de l'explosion combinatoire de nombre d'états, d'autre part. Nous nous basons dans notre approche sur les techniques de preuve et de résolution de contraintes, ainsi que sur la méthode formelle B connue par la maturité et la diversité des outils de V&V.

Chapitre 2

Introduction à la méthode B

« *The most painful thing about mathematics is how far away you are from being able to use it after you have learned it.* »

James Newman

Sommaire

2.1 Fondements de la méthode B 31
2.1.1 Notations de la théorie des ensembles 32
2.1.2 Machine abstraite 32
2.1.3 Langage des substitutions généralisées 34
2.1.4 Propriétés des substitutions généralisées 35
2.1.5 Obligations de preuve 36
2.1.6 Exemple 37
2.2 Les notions de raffinement et de modularité en B 38
2.2.1 Le raffinement 39
2.2.2 La modularité 39
2.3 Outils de V&V 39
2.3.1 AtelierB 40
2.3.2 GénéSyst 40
2.3.3 ProB 42
2.4 Combinaison de B et de CSP 43
2.4.1 Syntaxe des spécifications 43
2.4.2 Exemple de contrôle d'une machine B 45
2.5 Conclusion 47

La méthode B, introduite par Jean Raymond Abrial [ABRIAL 1996a] dans les années 80, est une méthode de spécification formelle orientée état. Elle est fondée sur un langage de spécification inspiré des langages Z et VDM. Elle se distingue par le fait qu'elle est une méthode de

développement complète qui couvre le cycle de vie d'un logiciel à partir de la phase d'analyse jusqu'à la phase d'implémentation. Le passage d'une phase à une autre se fait par raffinements successifs, qui partant d'une spécification abstraite aboutissent à un modèle concret traduisible en langage de programmation en passant par des descriptions de plus en plus fines du même modèle. Elle offre un processus de génération et de vérification des obligations de preuve (i.e des formules de preuve logiques) dans le but de fournir un moyen sûr et fiable pour prouver la cohérence et la correction du modèle.

Cette méthode a fait ses preuves dans de nombreux projets industriels dès son apparition. En effet, elle a été utilisée avec succès pour la première fois à la fin des années 80 par Alstom Transport dans le projet SACEM [1] pour la régulation du trafic de train. C'est ensuite vers la fin des années 90 que la méthode B a pris plus d'ampleur, notamment dans le domaine ferroviaire, avec le projet Météor [BEHM et al. 1999] et a vu naître son premier outil de preuve automatique commercialisé sous le nom d'AtelierB [2].

Actuellement, la méthode B dispose de nombreux outils pour assister les utilisateurs aussi bien dans la modélisation que dans les activités de V&V. Ce support technique a favorisé son utilisation dans différents domaines critiques tels que le diagnostic automobile [POUZANCRE 2003], les circuits électroniques [HALLERSTEDE 2003] ainsi que dans le domaine de la validation de la sécurité des cartes à puce et des règles de contrôle d'accès [DADEAU et al. 2008, LANET et REQUET 2000, REQUET 2003].

Dans ce chapitre, nous présentons les fondements de base de la méthode B en mettant l'accent sur les concepts que nous exploitons. Nous faisons également un tour d'horizon des principaux outils de V&V qui supportent l'approche et qui ont motivé le choix de la méthode B.

2.1 Fondements de la méthode B

La méthode B s'appuie sur un langage de spécification basé sur la logique du premier ordre ainsi que sur la théorie des ensembles. Une spécification abstraite B, appelée **machine abstraite**, permet de représenter les données d'un système par le biais de variables (et/ou de constantes) ayant des valeurs typées. Des propriétés invariantes sur ces données peuvent être spécifiées à l'aide de conjonctions de prédicats du premier ordre afin de préciser les contraintes que le système doit satisfaire. Une machine abstraite permet également de décrire l'évolution de l'état du système. En effet, un état, où les données sont affectées à des valeurs, peut évoluer vers un autre état grâce aux opérations. Ces dernières, définies dans un **langage de substitutions généralisées**, décrivent le changement des valeurs des données, et par conséquent la manière dont elles évoluent.

1. Système d'Aide à la Conduite, à l'Exploitation et à la Maintenance
2. http://www.atelierb.eu/

2.1.1 Notations de la théorie des ensembles

Le langage de spécification B intègre des notions issues de la théorie des ensembles afin d'exprimer et de manipuler les données. Dans le tableau 2.1, nous donnons un aperçu de certaines notations et définitions ensemblistes que nous utilisons par la suite dans nos exemples.

	Notation	**Définition**
Construction d'ensemble	$\emptyset$	Ensemble vide
	$E_1 \times E_2$	Produit cartésien des deux ensembles E_1 et E_2
	$\mathbb{P}(E)$	Ensemble des sous ensembles de E
	$\mathbb{P}_1(E)$	Ensemble des sous ensembles non vides de E
Expressions d'ensemble	$E_1 \cup E_2$	Union des deux ensembles E_1 et E_2
	$E_1 \cap E_2$	Intersection des deux ensembles E_1 et E_2
	$E_1 - E_2$	Différence d'ensembles entre E_1 et E_2
Relations et fonctions	$E_1 \leftrightarrow E_2$	Relation entre E_1 et E_2 : $\mathbb{P}(E_1 \times E_2)$
	$E_1 \nrightarrow E_2$	Fonction partielle entre E_1 et E_2
	$E_1 \rightarrow E_2$	Fonction totale entre E_1 et E_2
	$E_1 \rightarrowtail E_2$	Fonction injective partielle entre E_1 et E_2
	$E_1 \rightarrowtail E_2$	Fonction injective totale entre E_1 et E_2
Opérateurs sur les relations	$x \mapsto y$	couple (x,y) d'une relation
	$dom(r)$	Domaine de relation
	$ran(r)$	Co-domaine de relation
	$r[E]$	Image de relation
	r^{-1}	Inverse de relation
	$r \lhd E$	Soustraction sur le domaine
	$r \rhd E$	Soustraction sur le co-domaine
tels que E, E_1 et E_2 sont des ensembles, r est une relation, x et y des éléments appartenant à des ensembles		

TABLEAU 2.1 – Notations ensemblistes en B

2.1.2 Machine abstraite

Une machine abstraite, dont la structure générale est donnée par la Figure 2.1, se compose de trois parties différentes :

- **Partie entête** : Contient le nom de la machine précédé du mot clé `MACHINE`, ou du mot clé `REFINEMENT` s'il s'agit d'un raffinement d'une autre machine. On peut éventuellement rajouter des paramètres à la machine ainsi que des contraintes sur ces paramètres dans la clause `CONSTRAINTS`. Les éventuelles dépendances entre les machines abstraites sont aussi précisées dans cette partie (voir la sous-section 2.2.2).
- **Partie statique** : Contient la déclaration des ensembles abstraits ou énumérés dans la clause

SETS, ainsi que l'ensemble des variables (VARIABLES) et des constantes (CONSTANTS). Les propriétés invariantes sur les variables et les constantes, telles que les propriétés de typage, sont spécifiées respectivement dans les clauses INVARIANT et PROPERTIES. Des assertions supplémentaires peuvent être définies dans la clause ASSERTIONS sous forme d'une liste de prédicats pour exprimer des conséquences logiques des autres propriétés.

- **Partie dynamique** : Inclut une clause INITIALISATION dans laquelle les variables de la machine sont initialisées, et une autre clause OPERATIONS où les opérations offertes par le système sont spécifiées. Chacune des opérations est définie selon la forme générale suivante :

$$\texttt{Résultats} \longleftarrow \texttt{Nom\_Opération(Paramètres)} \;\widehat{=}\; \texttt{Substitution\_Généralisée}$$

Où Paramètres et Résultats sont optionnels et ils représentent respectivement la liste des entrées et des sorties de l'opération. Quant au corps de l'opération, il est défini par le biais des substitutions généralisées.

```
/* Entête de la machine */
MACHINE          M              /*Nom de la machine*/
/* Partie statique */
SETS             S,             /*Ensembles abstraits*/
                 T = {a,b,...}  /*Ensembles énumérés*/
CONSTANTS        Const          /*Les constantes*/
PROPERTIES       R              /*Spécification des constantes*/
VARIABLES        Var            /*Les variables*/
INVARIANT        Inv            /*Spécification des variables*/
/* Partie dynamique */
INITIALISATION   Init           /*L'initialisation*/
OPERATIONS       Op1;           /*Liste des opérations*/
                 Op2;
                   ...
                 Opn
END
```

FIGURE 2.1 – Structure générale d'une machine B

Remarque 2.1

Dans la suite, nous nous limitons à la forme générale donnée par la figure 2.1 et nous adoptons les notations de cette figure pour caractériser les différents composants d'une machine B. Ainsi, les ensembles abstraits seront désignés par ***S****, les ensembles énumérés par* ***T****, l'ensemble des constantes par* ***Const****, la spécification des constantes par* ***R****, les variables par* ***Var****, l'invariant par* ***Inv*** *et la substitution de l'initialisation par* ***Init****.*

2.1.3 Langage des substitutions généralisées

Le langage des substitutions généralisées est utilisé pour décrire la partie dynamique de la machine B (i.e. les opérations et l'initialisation). Il est décrit par des notations mathématiques permettant de décrire la transformation des prédicats. Il permet ainsi d'exprimer le passage d'un état *pré-opération* caractérisé par un prédicat avant l'exécution de l'opération vers un état post-opération obtenu après l'exécution de l'opération.

Cette transformation de prédicat est basée sur le calcul de la plus faible précondition, noté *Wp* - pour *Weakest Precondition*-, défini par Dijkstra [DIJKSTRA 1997] et inspiré de la logique de Hoare [HOARE 1969]. En effet, la transformation d'un prédicat P vers un prédicat Q par la substitution S, noté $Q = [S]P$, est équivalente au calcul de la plus faible pré-condition qui garantit que S se termine et que l'assertion P est vraie après la terminaison de S, on dit alors que la substitution S établit le prédicat P.

Par exemple, étant donnés :

— la substitution élémentaire de l'affectation : $(x := e)$ telleque x est une variable et e une expression qui obéit aux propriétés invariantes de x,

— un prédicat $P \widehat{=} (x > 10)$,

le calcul de la plus faible précondition, notée $[x := e](x > 10)$, aboutit au prédicat $Q \widehat{=} (e > 10)$ obtenu en remplaçant toutes les occurrences libres [3] de x par e dans P, noté $P[e/x]$.

Toute substitution généralisée en B est construite à partir des substitutions de base dites substitutions primitives que nous listons dans le tableau 2.2 en donnant aussi bien leur syntaxe mathématique que la notation verbeuse en B.

Nom de la substitution	**Syntaxe mathématique**	**Notation B**
Affectation	$x := e$	$x := e$
Affectation multiple	$x, y := e, f$	$x, y := e, f$
Sans effet	$skip$	$skip$
Séquencée	$S; T$	$S; T$
Préconditionnée	$P \mid S$	**PRE** P **THEN** S **END**
Choix borné	$S [] T$	**CHOICE** S **OR** T **END**
Gardée	$P \Longrightarrow S$	**SELECT** P **THEN** S **END**
Choix non borné	$@z.S$	**VAR** z **IN** S **END**
tels que x, y et z sont des variables, e et f des expressions, S et T des substitutions, et P un prédicat		

TABLEAU 2.2 – Les substitutions primitives en B

Ainsi, nous distinguons :

- La substitution **sans effet** : c'est une substitution muette ne produisant aucune transformation.

3. Les occurrences libres d'une variable dans un prédicat sont les occurrences de cette variable présente dans ce prédicat et qui ne sont pas sous la portée d'un quantificateur ($\forall, \exists, \{x|\ldots\}$, etc.)

- La substitution **séquencée** : spécifie deux substitutions qui seront exécutées séquentiellement dans l'ordre spécifié.
- La substitution **préconditionnée** : précise les conditions sous lesquelles une opération peut être appelée. En effet, une opération définie par une substitution préconditionnée ne peut s'exécuter que si le prédicat de la précondition P est satisfait.
- La substitution avec **choix borné** : définit deux substitutions (ou plus) qui décrivent les deux comportements possibles d'une opération sans préciser lequel sera effectivement exécuté. Un choix non déterministe sera alors fait au moment de l'exécution.
- La substitution **gardée** : définit les conditions d'activation des opérations. Ces dernières sont automatiquement déclenchées dès que le prédicat G de la garde est satisfait.
- La substitution avec **choix non borné** : définit un comportement non-déterministe dépendant du choix d'une valeur de la variable z.

Le calcul de la plus faible précondition relatif à chacune de ces substitutions est donné par le tableau 2.3.

Substitution	**Calcul de *Wp***	**Conditions**
$[x := e]R$	$R[e/x]$	
$[x,y := e,f]R$	$[z := f][x := e][y := z]R$	z non-libre dans e,f et R
$[skip]R$	R	
$[S;T]R$	$[S][T]R$	
$[P\|S]R$	$P \wedge [S]R$	
$[S[]T]R$	$[S]R \wedge [T]R$	
$[P \Longrightarrow S]R$	$P \Rightarrow [S]R$	
$[@z.S]R$	$\forall z.[S]R$	z non-libre dans R

TABLEAU 2.3 – Calcul de la plus faible précondition des substitutions primitives

2.1.4 Propriétés des substitutions généralisées

Une substitution généralisée est caractérisée par trois propriétés fondamentales qui sont la terminaison, l'état avant-après et la faisabilité. Pour illustrer ces trois propriétés sur un exemple, nous considérons l'opération simple de l'addition qui permet de rajouter la valeur de l'entier relatif passée en paramètre à la valeur de la variable d'état *Somme*. Nous donnons la spécification B de cette opération dans la figure 2.2.

$Add(valeur) \mathrel{\hat{=}}$ `PRE` $valeur \in \mathbb{Z}$ `THEN`
$Somme := Somme + valeur$
`END`

FIGURE 2.2 – Spécification B d'une opération d'addition simple

- **Prédicat de terminaison** : Noté $\mathsf{trm}(S)$ et défini par : $\mathsf{trm}(S) \Leftrightarrow [S]True$, il spécifie les conditions sous lesquelles l'exécution d'une substitution S termine. Par exemple, la terminaison de l'opération `Add` se calcule comme suit :

$$\begin{aligned}\mathsf{trm}(Add) &\Leftrightarrow [valeur \in \mathbb{Z} | Somme := Somme + valeur]True \\ &\Leftrightarrow (valeur \in \mathbb{Z}) \wedge [Somme := Somme + valeur]True \\ &\Leftrightarrow (valeur \in \mathbb{Z}) \wedge True \\ &\Leftrightarrow (valeur \in \mathbb{Z})\end{aligned}$$

  Ce qui signifie que l'opération `Add` termine si et seulement si le paramètre d'entrée de l'opération est un entier relatif.

- **Prédicat avant-après** : Noté $\mathsf{prd}_x(S)$, il permet d'établir la relation entre les variables d'état avant l'exécution de la substitution et après son exécution. Ce prédicat est défini par : $\mathsf{prd}_x(S) \Leftrightarrow \neg[S](x \neq x')$, tels que x est la valeur d'une variable d'état avant l'exécution de la substitution S, et x' est la valeur de cette variable après l'exécution. Par exemple, la valeur de la variable *Somme* de l'exemple 2.2 après l'exécution de l'opération `Add` est calculée comme suit :

$$\begin{aligned}\mathsf{prd}_{Somme}(Add) &\Leftrightarrow \neg[valeur \in \mathbb{Z} | Somme := Somme + valeur](Somme' \neq Somme) \\ &\Leftrightarrow \neg(valeur \in \mathbb{Z} \wedge Somme' \neq Somme + valeur) \\ &\Leftrightarrow valeur \notin \mathbb{Z} \vee Somme' = Somme + valeur\end{aligned}$$

- **Prédicat de faisabilité** : Noté $\mathsf{fis}(S)$ et défini par : $\mathsf{fis}(S) \Leftrightarrow \neg[S]False$, il caractérise l'espace d'état pour lequel la substitution S est faisable et permet d'aboutir à un résultat. Par exemple, la faisabilité de l'opération `Add` se calcule de la manière suivante :

$$\begin{aligned}\mathsf{fis}(Add) &\Leftrightarrow \neg[valeur \in \mathbb{Z} | Somme := Somme + valeur]False \\ &\Leftrightarrow \neg((valeur \in \mathbb{Z}) \wedge False) \\ &\Leftrightarrow True\end{aligned}$$

  On dit alors que l'opération `Add` est toujours faisable.

  La faisabilité d'une substitution peut également être exprimée en terme de prédicat avant-après. En effet, une substitution portant sur une variable x est faisable si et seulement si le calcul de la valeur finale x' est possible. Ainsi, on peut définir la faisabilité comme suit : $\mathsf{fis}(S) \Leftrightarrow \exists x' \cdot \mathsf{prd}_x(S)$

2.1.5 Obligations de preuve

Les obligations de preuve sont des formules mathématiques qui expriment les conditions de validité d'une machine B. Elles permettent d'assurer que les composants d'une machine B sont corrects et préservent les propriétés souhaitées notamment les propriétés invariantes. Il s'agit alors de générer des preuves, souvent basées sur le calcul de la plus faible précondition. Ces preuves

permettent de vérifier que l'invariant est toujours établi aussi bien par l'initialisation que par les opérations si elles sont exécutées à partir d'un état satisfaisant les propriétés de la machine notamment les contraintes C (clause `CONSTRAINTS`) ainsi que les propriétés des constantes R (clause `PROPERTIES`).

On dit que l'initialisation établit l'invariant si l'exécution de la substitution *Init* aboutit à un état qui satisfait l'invariant *Inv*. De la même manière, l'action d'une opération appelée à partir d'un état qui satisfait les propriétés de la machine ainsi que l'invariant et les préconditions nécessaires à son exécution doit aboutir à un état où l'invariant est vrai. Par conséquent, on dit que les composants de la machine B sont corrects si et seulement si les obligations de preuve présentées dans le tableau 2.4 sont établis.

Composant	Forme	Obligation de preuve
Initialisation	*Init*	$C \wedge R \Rightarrow [Init]Inv$
Opération	**PRE** P **THEN** Action **END**	$C \wedge R \wedge Inv \wedge P \Rightarrow [Action]Inv$

TABLEAU 2.4 – Obligations de preuve de l'initialisation et des opérations

2.1.6 Exemple

Nous considérons un SI d'une bibliothèque simple qui permet de gérer les prêts de livres à ses clients (figure 2.3). Cette gestion est assurée grâce aux trois opérations suivantes :

— `LendBook` : Permet à un client d'emprunter un livre s'il est disponible.

— `ReturnBook` : Permet de retourner un livre déjà emprunté.

— `GetAvailableBooks` : Permet de visualiser la liste de livres disponibles.

Nous considérons deux ensembles abstraits `Books` et `Members` qui désignent l'ensemble des livres et des clients inscrits à la bibliothèque. L'état du système est caractérisé par la variable `Lend` qui stocke les emprunts des clients. Cette variable est donc spécifiée comme étant une relation associant les clients aux livres empruntés.

```
MACHINE
   Library_Machine
SETS
   Books ;
   Members
VARIABLES
   Lend
INVARIANT
   Lend ∈ Members ↔ Books
INITIALISATION
   Lend := ∅
OPERATIONS
   LendBook(mm,bb)    ≙ PRE  mm ∈ Members ∧
                             bb ∈ Books ∧
                             bb ∉ ran(Lend)
                        THEN
                             Lend := Lend ∪ {(mm↦bb)}
                        END ;

   ReturnBook(mm,bb)  ≙ PRE  mm ∈ Members ∧
                             bb ∈ Books ∧
                             (mm ↦ bb) ∈ Lend
                        THEN
                             Lend := Lend − {(mm↦bb)}
                        END ;

   Res ← GetAvailableBooks ≙
                        BEGIN
                             Res := Books − ran(Lend)
                        END
END
```

FIGURE 2.3 – Exemple d'une machine B

2.2 Les notions de raffinement et de modularité en B

Les notions de raffinement et de modularité sont fondamentales en B et sont considérées parmi les points forts de la méthode. Toutefois, étant des notions non exploitées dans notre contribution, nous les présentons brièvement et nous invitons le lecteur intéressé à se référer au B-Book [ABRIAL 1996a].

2.2.1 Le raffinement

La notion de raffinement offre une méthodologie de développement itérative incrémentale qui permet de passer d'une spécification abstraite non déterministe à une spécification de plus en plus concrète et déterministe.

Une machine de raffinement est introduite par la clause `REFINEMENT` ou encore par la clause `IMPLEMENTATION` s'il s'agit du dernier raffinement, suivi par la clause `REFINES` pour spécifier la machine qui fait l'objet du raffinement. Elle a également la particularité de présenter la même interface et le même comportement que la machine abstraite, ainsi, cette dernière peut facilement être remplacée par la machine concrète sans que l'utilisateur ne puisse s'en rendre compte. Par conséquent, les opérations de la machine abstraite doivent être reprises intégralement en conservant la même signature mais avec des substitutions plus déterministes. Outre les opérations, les variables peuvent, elles aussi, être concernées par le raffinement. Ainsi, les variables de la machine abstraite peuvent être conservées, modifiées ou même ignorées. De nouvelles variables peuvent également être définies, dans ce cas un invariant dit de ***collage*** doit être spécifié afin de préciser la relation entre les nouvelles variables et les variables de la machine abstraite.

2.2.2 La modularité

Pour réduire la complexité du développement d'un système, la méthode B propose une décomposition modulaire qui permet de structurer la spécification en plusieurs sous-systèmes (ou composants) plus petits et de moindre complexité. Afin de faciliter leur intégration et de favoriser leur réutilisation, des mécanismes de liaison entre les différents composants sont offerts. Parmi ces mécanismes nous distinguons les mécanismes d'inclusion (clause `INCLUDES`), de consultation (clauses `USES` et `SEES`), d'importation (clause `IMPORTS`), de promotion des opérations (clause `PROMOTES`) et d'extension d'interface (clause `EXTENDS`).

2.3 Outils de V&V

La diversité des outils de support à l'automatisation des activités de V&V est l'un des atouts majeurs de la méthode B. En effet, une panoplie d'outils existe autour de cette méthode formelle, certains sont développés par des industriels tels que les outils de preuve AtelierB [ENGINEERING 2011] et B-Toolkit [ROBINSON 1997], et d'autres sont développés par des unités de recherche tels que l'animateur et le model-checker ProB [LEUSCHEL et BUTLER 2003], le parseur de spécification JBtools [VOISINET et al. 2002], le générateur d'obligations de preuve Barvey [COUCHOT et al. 2004], le générateur de systèmes de transitions étiquetées GénéSyst[D. BERT et STOULS], etc.. Nous nous intéressons en particulier aux trois outils AtelierB, ProB et GénéSyst autour desquels nous avons construit nos travaux de thèse.

2.3.1 AtelierB

L'AtelierB offre un vérificateur sémantique et syntaxique, un générateur automatique d'obligations de preuve, un outil de preuve interactif et automatique ainsi qu'un traducteur vers des langages de programmation tels que C, C++ et ADA. Il couvre la vérification de la correction de l'initialisation et des opérations par rapport à l'invariant, comme il permet de prouver la correction d'un raffinement vis à vis du modèle abstrait. Il propose de décharger automatiquement les obligations de preuve selon des niveaux de force variables (force Rapide, force 0 à force 3). Plus la force est élevée, plus le temps de preuve s'accroît. Il propose également une interaction avec les utilisateurs en cas d'échec de la preuve automatique afin d'orienter la preuve.

Par exemple, en utilisant l'AtelierB sur l'exemple de la figure 2.3, nous avons pu générer trois obligations de preuve : une relative à la clause d'initialisation et les deux autres pour vérifier la correction des deux opérations `LendBook` et `ReturnBook`. Ces obligations de preuve ont été automatiquement déchargées en utilisant le niveau de force 0 (figure 2.4).

Opération	Prouvé	Non prouvé
TOTAL	3	0
clause Initialisation	1	0
clause LendBook	1	0
clause ReturnBook	1	0

FIGURE 2.4 – Génération des obligations de preuve de l'exemple 2.3 par AtelierB

2.3.2 GénéSyst

C'est un outil développé au sein du Laboratoire d'Informatique de Grenoble (LIG) pour la génération des Système de Transitions Symbolique (STS) permettant de représenter le comportement d'une spécification écrite en méthode B événementiel. Cette dernière (noté *Event-B*)[ABRIAL 1996b] est une variante de la méthode B dédiée à la spécification des systèmes temps réels. L'une des principales différences qui la distingue de la méthode B classique est la notion d'événement qui remplace la notion d'opération. En effet, les actions du système sont représentées par des événements ayant une garde afin qu'ils soient déclenchés automatiquement une fois que la garde est satisfaite. A la différence des opérations, un événement ne contient ni paramètres ni préconditions. Ainsi, pour appliquer l'outil GénéSyst sur une spécification en B classique, une traduction de cette dernière vers Event-B est requise [BENDISPOSTO et al. 2008].

A partir d'une spécification Event-B et d'un ensemble d'états symboliques [4], l'outil GénéSyst génère un système de transitions étiquetées par les événements de la spécification. Par exemple, la figure 2.5 représente le STS obtenu en appliquant l'outil GénéSyst sur l'exemple de la figure

4. Un état symbolique est représenté par un prédicat et il est constitué de l'ensemble des états concrets du système qui satisfont ce prédicat.

2.3 (après l'avoir traduit en Event-B) afin de visualiser les transitions possibles entre les deux états $E \widehat{=} Lend = \emptyset$ et $F \widehat{=} Lend \neq \emptyset$.

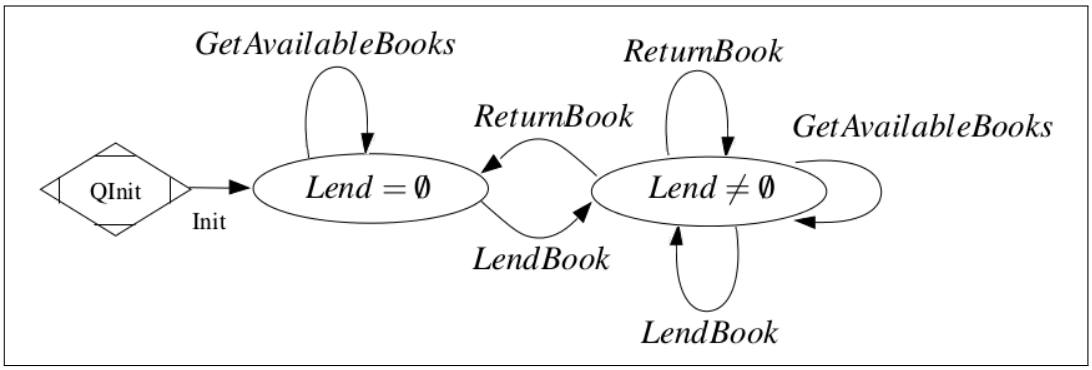


FIGURE 2.5 – STS généré par GénéSyst à partir de l'exemple 2.3

Le calcul des transitions entre deux états symboliques se fait en considérant deux sortes d'obligations de preuve. En effet, étant donnés deux états symboliques représentés respectivement par les prédicats E et F, GénéSyst génère pour tout événement ev de la machine les obligations de preuve suivantes :

(i) Les obligations de preuve de la déclenchabilité des événements :
- (A) Toujours déclenchable : $\forall x \cdot (E \Rightarrow Garde(ev))$
- (B) Non déclenchable : $\forall x \cdot (E \Rightarrow \neg Garde(ev))$
- (C) Partiellement déclenchable : $\exists x \cdot (E \wedge Garde(ev))$

(ii) Les obligations de preuve de l'atteignabilité des états par les événements :
- (D) Toujours atteignable : $\forall x \cdot (E \wedge Garde(ev) \Rightarrow [Action(ev)]F)$
- (E) Non atteignable : $\forall x \cdot (E \wedge Garde(ev) \Rightarrow [Action(ev)]\neg F)$
- (F) Partiellement atteignable : $\exists x \cdot (E \wedge Garde(ev) \Rightarrow \neg[Action(ev)]\neg F)$

Où :

— x : dénote l'ensemble des variables d'état.

— $Garde(ev)$: dénote la garde de l'événement ev.

— $Action(ev)$: dénote la substitution de l'action de l'événement ev.

— La formule (A) établit que toutes les valuations de E vérifient la garde de l'événement ev, et donc ce dernier est toujours déclenchable à partir de l'état symbolisé par E.

— La formule (B) vérifie qu'aucune des valeurs de E ne satisfait la garde de ev, et par conséquent cet événement n'est jamais déclenchable à partir de E.

— La formule (C) vérifie qu'il existe certaines valeurs de x qui satisfont la garde de ev et le prédicat E, ce qui permet de vérifier que la déclenchabilité est possible à partir d'un sous ensemble d'états de E.

— La formule (D) vérifie que la substitution de l'action de ev établit F si toutes les valuations de E vérifient la garde de ev et donc ce dernier permet toujours d'atteindre F.

— La formule (E) établit que l'action de *ev* satisfait la négation de F. Par conséquent, *ev* ne permet en aucun cas d'atteindre F.

— La formule (F) consiste à vérifier que la substitution de l'action de *ev* n'établit pas $\neg F$ [5], il est donc possible qu'elle établisse F. Ainsi, cette formule exprime l'éventualité de l'atteignabilité de F par *ev*.

Ces obligations de preuve permettent de calculer les **transitions** entre les états définis telles qu'un événement établit une transition entre E et F s'il est prouvé déclenchable à partir de E et qu'il atteint F.

2.3.3 ProB

ProB a été développé conjointement par les universités de Düsseldorf et Southampton. Il a été principalement conçu pour l'animation et le model-checking des spécifications B :

— L'animateur est muni d'une interface graphique permettant à l'utilisateur de visualiser l'historique des opérations exécutées pour atteindre un état particulier ainsi que toutes les opérations déclenchables à partir de cet état. Il a également l'avantage de pouvoir prendre en charge des spécifications de grande taille ainsi que les opérations non déterministes.

— Le model-checker couvre la vérification de violation de l'invariant, la recherche d'éventuels interblocages (*deadlocks*) ainsi que la recherche d'une séquence d'opérations qui atteint un état donné. Pour ce faire, il explore l'espace d'états en proposant différentes stratégies de recherche (en profondeur d'abord, en largeur d'abord, aléatoire, en se basant sur des heuristiques, etc.).

ProB offre également une large gamme de fonctionnalités autres que l'animation et le model-checking permettant d'optimiser les activités de V&V. Parmi ces fonctionnalités, nous nous intéressons en particulier au solveur de contraintes [Leuschel et Schneider 2014] qui permet de résoudre des problèmes de satisfaction de contraintes [Van Hentenryck 1989] exprimés en B. Le solveur de contraintes de ProB a montré son efficacité pour la résolution de certains problèmes comparé aux solveurs kodkod/SAT [Torlak et Jackson 2007] et Z3/SMT [De Moura et Bjørner 2008]. En effet, ProB peut être considéré comme un très bon solveur pour la résolution des problèmes opérant sur les entiers ainsi que pour la recherche de modèles satisfaisant des contraintes, comme il peut traiter des représentations d'ensembles abstraits. Toutefois, le solveur kodkod est plus efficace pour le traitement des ensembles énumérés et des relations complexes entre les ensembles. Quant à Z3, il est plus fort pour le traitement de certains domaines non bornés et pour la détection d'incohérence. Dans le but d'offrir un solveur de contraintes plus performant, ProB a intégré les deux solveurs kodkod et Z3 pour compléter son propre solveur. Toutes ces raisons ont motivé notre choix d'utiliser le solveur de contraintes de ProB.

ProB prend en charge également d'autres méthodes formelles telles que Z, TLA+ et CSP. Il offre aussi une technique pour guider le model-checker [Butler et Leuschel 2005] par l'algèbre

5. La notation $\neg[S]\neg R$ signifie qu'il n'est pas toujours vrai que S n'établisse pas R ; intuitivement, cela signifie qu'il existe un certain calcul de S qui établit R.

de processus CSP [HOARE 1978]. Cette technique permet de réduire le problème de l'explosion combinatoire du nombre d'états, comme elle permet la vérification de combinaison de spécifications CSP||B (*CSP parallèle B*). Cette combinaison permet de tirer profit, d'une part, de la richesse de la méthode B, et d'autre part, de l'aspect comportemental de l'algèbre de processus CSP. En effet, la méthode B permet la description des données et des propriétés d'invariance ainsi que des opérations caractérisant les transitions d'états.Quant à l'algèbre de processus CSP, elle permet d'exprimer des propriétés dynamiques telles que l'ordonnancement des opérations.

Cette complémentarité a suscité notre intérêt. Par conséquent, l'approche CSP||B a été utilisée dans le cadre de cette thèse pour spécifier et vérifier l'existence de certaines traces d'exécution révélant des comportements malicieux.

2.4 Combinaison de B et de CSP

L'approche CSP||B [SCHNEIDER et TREHARNE 2005] permet de spécifier un système conjointement par des modèles CSP et des machines B. La combinaison de ces deux méthodes formelles a pour but de contrôler l'exécution des opérations d'un système en introduisant des contraintes d'ordonnancement. Dans cette combinaison, B est utilisé pour la description des données ainsi que pour la spécification des opérations qui manipulent ces données. Quant à l'algèbre de processus CSP, elle joue le rôle d'un contrôleur d'exécution pour ces opérations.

Un composant CSP||B (figure 2.6) est un assemblage d'une ou de plusieurs machines B avec un ou plusieurs contrôleurs CSP, où chaque machine est associée à un processus CSP qui contrôle l'exécution de ses opérations. Un contrôleur peut appeler une opération de la machine qu'il contrôle en utilisant **un canal de machine**, comme il peut communiquer avec les autres contrôleurs ou avec l'environnement extérieur par le biais des **canaux de communication**.

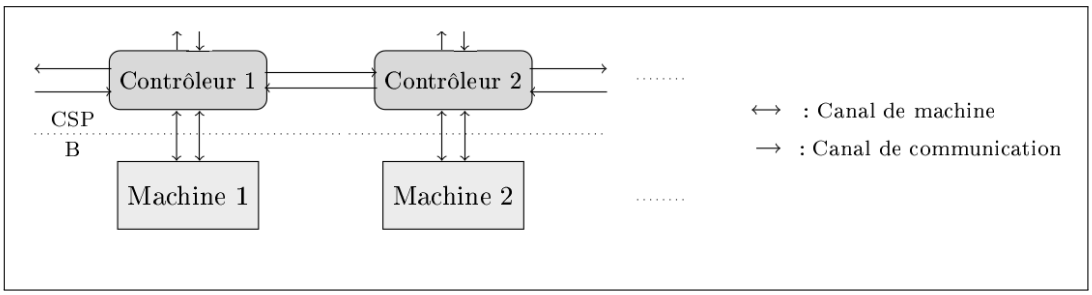


FIGURE 2.6 – Architecture des composants CSP||B [SCHNEIDER et TREHARNE 2005]

2.4.1 Syntaxe des spécifications

En CSP||B, la syntaxe des machines abstraites B reste inchangée. Ainsi, toutes les structures de la méthode B sont supportées par l'approche. Cependant, seul un sous ensemble du langage CSP est autorisé pour la description des contrôleurs dont la syntaxe est la suivante :

$$
\begin{array}{rl}
P ::= & a \rightarrow P \mid \\
& c?x\{E(x)\} \rightarrow P \mid \\
& d!v\{E(v)\} \rightarrow P \mid \\
& e?v!x\{E(x)\} \rightarrow P \mid \\
& P_1 \Box P_2 \mid P_1 \sqcap P_2 \mid \\
& \texttt{if b then } P_1 \texttt{ else } P_2 \texttt{ end} \mid \\
& S(p)
\end{array}
$$

Où :

— P est un processus défini par l'utilisateur ou un processus prédéfini. En CSP, il existe deux processus prédéfinis qui ne produisent aucun effet. Le premier, dénoté par **STOP**, conduit à l'arrêt immédiat de l'exécution en produisant un interblocage (*deadlock*). Quant à celui dénoté par **SKIP**, il conduit à la terminaison du processus avec succès.

— a est un événement. On dit qu'un processus P est préfixé par l'événement a et on note $a \rightarrow P$ s'il exécute a puis se comporte comme P. Ceci permet de définir un processus en explicitant l'événement ou la suite d'événements qu'il est censé exécuter. Il est donc possible d'enchaîner les préfixes si le processus exécute une suite d'événements. Par exemple si P exécute la suite d'événements $a_1, a_2, \ldots, a_n$ on note :

$$a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_n \rightarrow P$$

— c est un canal acceptant des entrées et d est un canal acceptant des sorties. En effet, un événement en CSP peut être représenté soit par un nom atomique simple par exemple `démarrer`, `arrêter`, `connecter`, etc. soit par un **canal** faisant passer des messages d'entrée et/ou de sortie. Un canal est exprimé sous la forme de `channel?in!out` tels que `channel` est le nom du canal (ou de l'événement), `in` est la liste des messages d'entrée et `out` est la liste des messages de sortie.

— e est un canal de contrôle (ou un canal de machine) utilisé pour la communication entre la machine B et le contrôleur CSP. L'expression $e?v!x\{E(x)\} \rightarrow P$ dénote un appel à une opération de machine B en interaction avec le contrôleur : Le processus de contrôle communique la valeur v en entrée à l'opération de la machine B et reçoit la valeur de sortie x de cette opération. $E(x)$ est un prédicat sur x tel que le processus P peut terminer si x satisfait le prédicat $E(x)$.

— $P_1 \Box P_2$ dénote un choix externe entre processus. Il permet d'exprimer un choix déterministe entre deux ou plusieurs processus. Ce choix est dit externe (ou déterministe) si le premier événement de chaque processus dépend d'un autre processus.

— $P_1 \sqcap P_2$ dénote un choix interne entre processus. Il permet d'exprimer un choix non déterministe entre deux ou plusieurs processus indépendants.

— L'expression conditionnelle **if** `b` **then** P_1 **else** P_2 **end** permet selon la valeur de b de continuer avec l'un ou l'autre des processus.

— $S(p)$ dénote l'appel récursif du processus p.

2.4.2 Exemple de contrôle d'une machine B

Nous reprenons l'exemple 2.3 de la machine B `Library_Machine` à laquelle nous associons un processus de contrôle d'exécution `Library_Process`.

L'architecture du composant CSP||B représentant les interactions entre la machine B et le contrôleur est donnée par la figure 2.7. Ces interactions sont assurées par les trois canaux de contrôle qui correspondent aux trois opérations de la machine B à savoir `LendBook`, `ReturnBook`, `GetAvailableBooks`. Outre ces trois événements, le contrôleur interagit avec l'environnement extérieur pour activer ou désactiver le processus respectivement par l'intermédiaire des événements `activate` et `deactivate`.

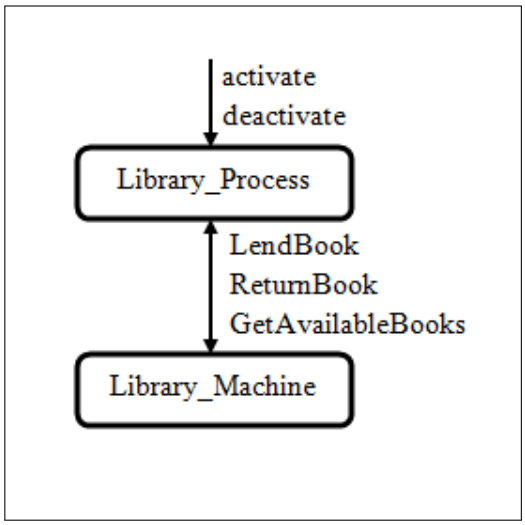


FIGURE 2.7 – Architecture CSP||B du contrôle de l'exemple 2.3

La figure 2.8 donne la spécification du processus `Library_Process`. Ce dernier, une fois activé par l'événement `activate`, se comporte comme le processus `RUN` qui spécifie la séquence valide des opérations de la bibliothèque. Il offre le choix entre trois fonctions différentes : soit lancer le processus `BORROW` qui permet d'emprunter un livre, soit lancer le processus `DISPLAY` qui permet d'afficher les livres disponibles ou encore désactiver le processus et retourner au processus principal. Le processus `BORROW` fixe l'ordre d'appel des opérations. Ainsi, un livre n'est retourné que s'il a été emprunté, en plus, par le biais des entrées des canaux nous garantissons que le livre `bb` faisant l'objet d'un emprunt ne peut être retourné que par le membre `mm` qui l'a emprunté. Ce processus impose également une contrainte supplémentaire qui consiste à ne déclencher le processus d'emprunt de nouveau que si le livre a été restitué. Quant au processus `DISPLAY`, il se limite à l'appel à l'opération `GetAvailableBooks`. Les deux processus finissent par se comporter de nouveau comme le processus `RUN`.

Le contrôle de l'ordre d'exécution des opérations d'une machine B par un processus CSP permet non seulement de décrire le comportement souhaité du système, mais également de réduire l'espace d'états en éliminant les transitions inutiles. En effet, dans le cas où nous nous intéressons à un comportement bien particulier, le contrôleur CSP permet de guider le model-checker dans son

```
Library_Process  =  activate → RUN
            RUN  =  BORROW □
                    DISPLAY □
                    deactivate → Library_Process
         BORROW  =  LendBook!mm!bb → ReturnBook!mm!bb → RUN
        DISPLAY  =  GetAvailableBooks?Res → RUN
```

FIGURE 2.8 – Contrôleur CSP de la machine `Library`

exploration pour trouver plus rapidement les chemins d'exécution qui satisfont ce comportement. Par exemple, en considérant l'ensemble des données de la figure 2.9, le contrôleur CSP donné à la figure 2.7 a permis une réduction de 50% des états explorés par le model-checker de ProB (figure 2.10) par rapport à l'espace d'états exploré par ProB sans guidance d'un contrôleur CSP (figure 2.11).

```
SETS
   Books = {b1,b2};
   Members = {m1,m2}
```

FIGURE 2.9 – Caractérisation des ensembles de données

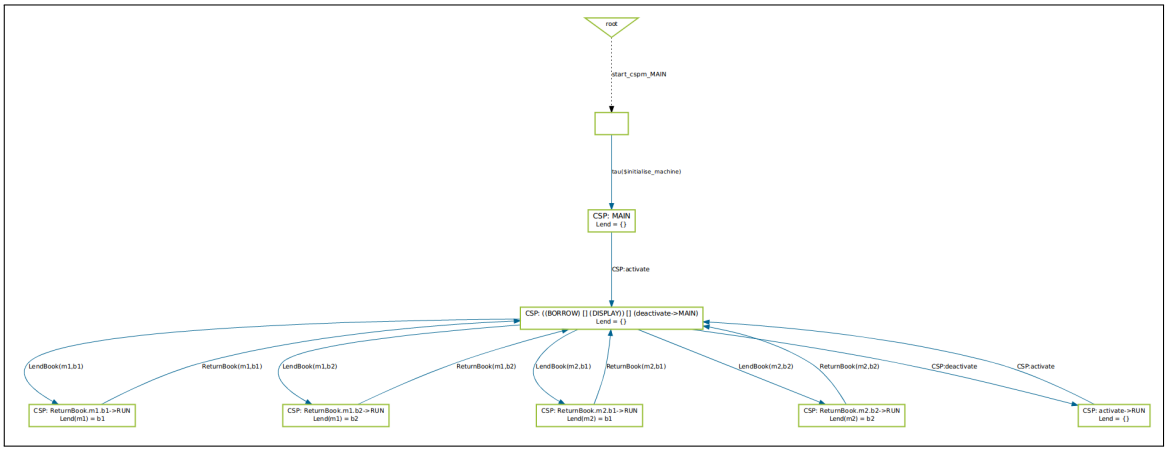


FIGURE 2.10 – Espace d'états avec guidance par le contrôleur CSP

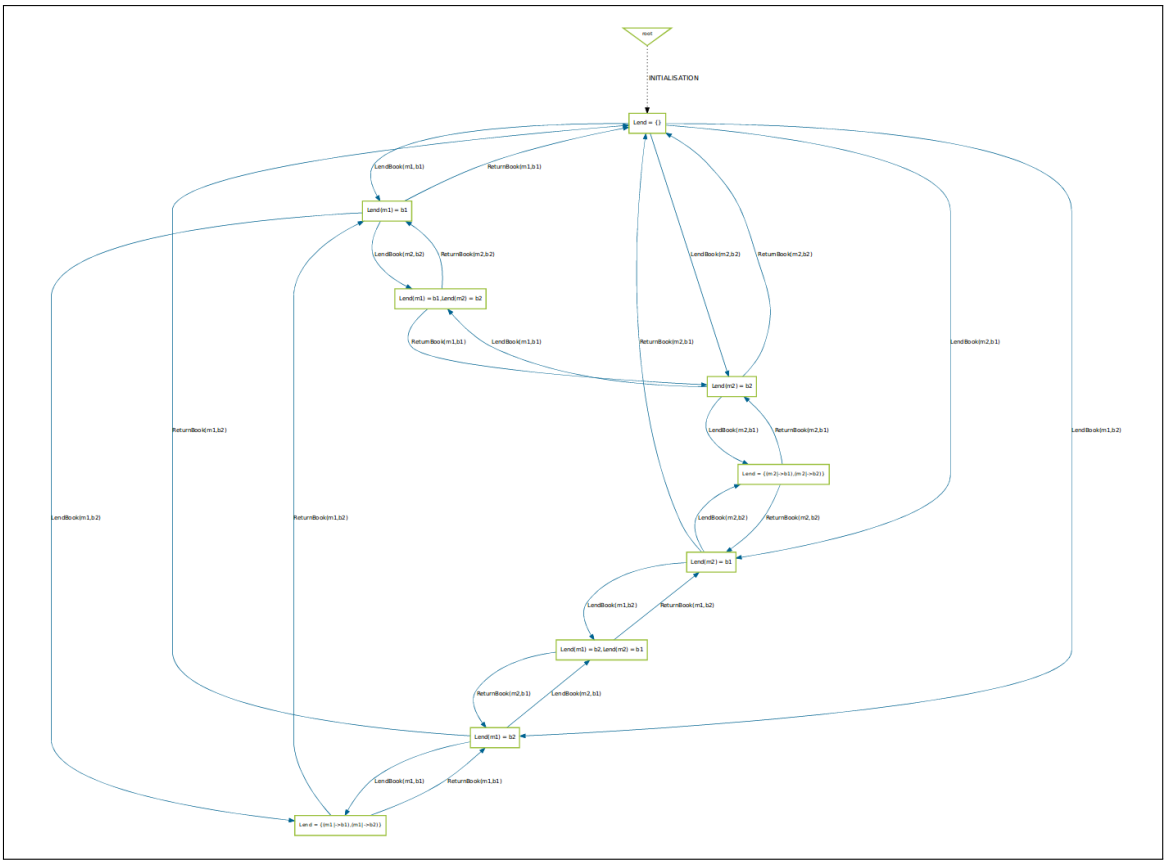


FIGURE 2.11 – Espace d'états sans guidance par le contrôleur CSP

2.5 Conclusion

La méthode B présente deux atouts majeurs assurant son succès dans le cadre de la validation des systèmes critiques. En effet, sa couverture de l'intégralité du cycle de développement d'un logiciel en plus de la disponibilité d'une large gamme d'outils pour l'automatisation de la phase de V&V fait de B une méthode complète. En outre, la richesse syntaxique du langage B offre la possibilité de modéliser des structures de données complexes pouvant être manipulées aisément en utilisant les substitutions généralisées.

Cette richesse a encouragé à combiner la méthode B avec d'autres langages formels à base d'événements tels que CSP afin de pouvoir prendre en compte l'aspect comportemental dans la modélisation. Ceci a donné naissance à l'approche CSP||B permettant de vérifier les propriétés de vivacité d'un système en plus de ses propriétés de sûreté. Dans notre investigation, nous utilisons cette approche pour guider un model-checker vers la bonne direction en spécifiant l'ordre des appels des opérations, que nous spécifions en B, ce qui évite l'exploration des transitions inutiles.

Cette exploration est faite, en particulier, dans le but de chercher des scénarios d'attaque malicieux qui tentent de contourner les règles de contrôle d'accès en faisant évoluer l'état fonctionnel du SI. Par conséquent, cette recherche requiert l'analyse de l'impact du modèle fonctionnel sur le modèle de sécurité. C'est pour cette raison que nous adoptons une approche de modélisation en B

qui permet d'expliciter le lien entre les opérations fonctionnelles et les règles de contrôle d'accès.

Chapitre 3

B4MSecure pour la validation conjointe en UML et B des SI sécurisés

« The amateur software engineer is always in search of magic, some sensational method or tool whose application promises to render software development trivial. It is the mark of the professional software engineer to know that no such panacea exist. »

Grady Booch

Sommaire

3.1 Modélisation fonctionnelle 51
3.1.1 Exemple illustratif : SI d'une bibliothèque 51
3.1.2 Traduction des aspects structurels 51
3.1.3 Génération des opérations 52
3.2 Modélisation des règles de contrôle d'accès RBAC 55
3.2.1 Exemple illustratif : SI d'une bibliothèque sécurisée 55
3.2.2 Traduction en B des règles de sécurité 58
3.3 Validation des modèles 61
3.3.1 Validation du modèle fonctionnel 62
3.3.2 Validation du modèle de sécurité 63
3.4 Conclusion 65

Dans le but de proposer un outillage dédié à la recherche de scénarios malicieux issus de règles de contrôle d'accès, nous nous orientons vers l'usage de techniques de raisonnement formel, en particulier celles autour de B. Pour ce faire, notre investigation requiert une modélisation formelle B qui soit fidèle aux exigences fonctionnelles et aux règles de contrôle d'accès et qui soit exploitable par les outils de preuve et de model-checking de B. C'est pourquoi nous tirons profit

des approches de couplage entre UML et B [IDANI 2006] qui permettent d'offrir une vue structurelle et comportementale développée selon un standard largement connu et utilisé dans le contexte de la modélisation des SI.

Nous adoptons, ainsi, une approche permettant une modélisation conjointe en UML et B aussi bien des aspects fonctionnels du SI que des règles de contrôle d'accès basées sur le modèle RBAC. Cette approche a été mise en œuvre par la plate-forme B4MSecure [1] [IDANI et LEDRU 2015] qui permet une modélisation UML et une validation en B du modèle fonctionnel et des règles de contrôle d'accès tout en gardant une séparation claire des préoccupations. En effet, le processus de dérivation de UML vers B produit deux modèles formels (figure 3.1) :

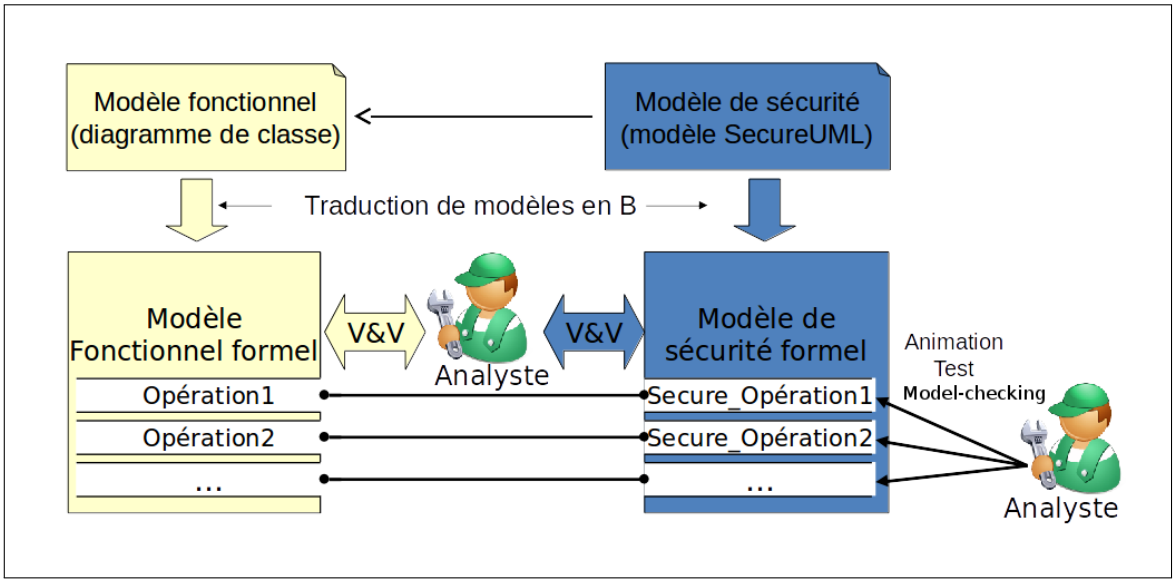


FIGURE 3.1 – Processus de dérivation de UML vers B mis en oeuvre par la plate-forme B4MSecure

— Un premier modèle B issu du modèle fonctionnel via une traduction directe du diagramme de classes UML en B. Les spécifications formelles qui en résultent peuvent être enrichies par la prise en compte de contraintes fonctionnelles et servent pour vérifier la correction du modèle fonctionnel indépendamment des règles de sécurité.

— Un deuxième modèle B qui représente la politique de sécurité, en vue de contrôler l'accès aux diverses entités fonctionnelles, issu de la traduction des diagrammes secureUML. Il peut être analysé indépendamment des aspects fonctionnels.

La plate-forme favorise également le lien entre ces deux modèles B qui est explicité par les opérations. En effet, les opérations issues du modèle fonctionnel permettent d'effectuer des opérations sans aucun contrôle d'accès, alors que celles issues du modèle de sécurité ont pour objectif de filtrer l'accès aux seules opérations autorisées pour l'utilisateur courant dans le contexte courant.

Dans ce chapitre, nous faisons un tour d'horizon de la plate-forme B4MSecure et nous montrons son intérêt pour notre travail. Dans un premier temps, nous expliquons les principes de la

1. http://b4msecure.forge.imag.fr

traduction en B du modèle fonctionnel. Dans un deuxième temps, nous abordons la modélisation du modèle RBAC et sa traduction en B. Finalement, nous décrivons les activités de validation qui peuvent être menées sur les modèles formels résultants.

3.1 Modélisation fonctionnelle

La plate-forme B4MSecure permet d'éditer des diagrammes de classe UML en vue de spécifier les exigences fonctionnelles tout en représentant les propriétés structurelles ainsi que les opérations décrivant le comportement du système. La traduction en B de ces diagrammes procède par la génération de la partie statique de la machine B à partir des attributs de classes et des associations. Quant à la partie dynamique, elle est générée à partir des opérations de base et est complétée par les opérations spécifiques définies par les utilisateurs.

3.1.1 Exemple illustratif : SI d'une bibliothèque

Le diagramme de classe UML de la figure 3.2 représente un SI d'une bibliothèque inspiré de [MAMMAR et al. 2011].

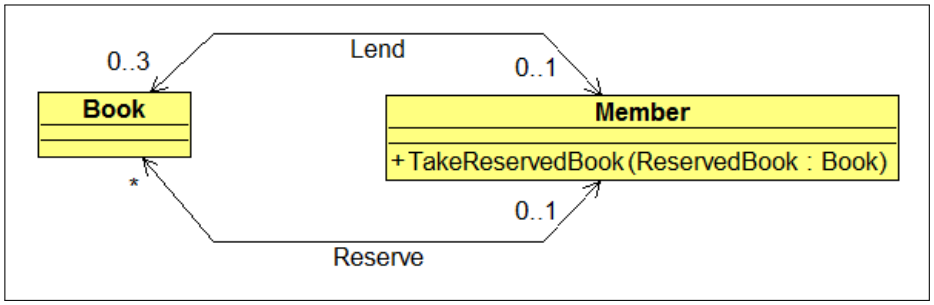


FIGURE 3.2 – Diagramme de classe UML du SI de la bibliothèque

Ce SI permet de sauvegarder l'ensemble des livres (classe `Book`) et des membres de la bibliothèque (classe `Member`).

Le système permet également de gérer l'emprunt des livres par les membres. En effet, un membre peut emprunter jusqu'à trois livres en même temps (association `Lend`). Il peut également réserver un livre qui est déjà en possession d'un autre membre (association `Reserve`) afin de pouvoir l'emprunter plus tard, une fois disponible, tout en bloquant son emprunt pour les autres membres. Ainsi, seul le membre qui l'a réservé peut l'emprunter grâce à l'opération `TakeReservedBook` de la classe `Member`.

3.1.2 Traduction des aspects structurels

La notion d'ensemble abstrait en B (*abstract set*) représente une abstraction d'un ensemble d'objets. Ainsi chaque classe `Class` du diagramme UML sera traduite en un ensemble abstrait

CLASS et donne lieu à une variable d'état Class incluse dans l'ensemble CLASS. Par exemple la classe Book donne lieu à :

(i) L'ensemble abstrait BOOK désignant l'ensemble des livres possibles.

(ii) La variable Book désignant l'ensemble des livres effectifs.

(iii) L'invariant : Book $\subseteq$ BOOK.

Quant aux attributs de classe, ils sont traduits par une relation fonctionnelle associant l'ensemble des instances effectives et le type de l'attribut. Les spécialisations de cette relation fonctionnelle dépendent de la nature de l'attribut : obligatoire ou optionnel, unique ou non. Par exemple, si nous supposons que les livres sont identifiés par un numéro unique ISBN, ce dernier donnera lieu à la variable Book_ISBN et à la propriété d'invariance qui associe l'ensemble des livres effectifs Book à l'ensemble des entiers naturels par une fonction injective totale :

$$\boxed{\texttt{Book\_ISBN} \in \texttt{Book} \rightarrowtail \textbf{NAT}}$$

Le tableau 3.1 donne les différentes traductions couvertes par l'outil :

	Optionnel	Obligatoire
Unique	$\rightarrowtail\!\!\!\!\!\!\circ$	$\rightarrowtail$
Non unique	$\rightarrow\!\!\!\!\!\!\circ$	$\rightarrow$

TABLEAU 3.1 – Relations fonctionnelles issues des attributs de classes

La traduction des associations suit le même principe que la traduction des attributs. Par exemple, l'association Reserve sera traduite en une fonction partielle associant les livres aux membres effectifs étant donné que les multiplicités qu'elle porte sont 0..* et 0..1. La figure 3.3 présente l'invariant produit automatiquement par B4MSecure à partir du modèle UML de la figure 3.2.

```
INVARIANT
          Book ⊆ BOOK ∧
          Member ⊆ MEMBER ∧
          Lend ∈ Member ↔ Book ∧
          ∀c2·c2 ∈ dom(Lend) ⇒ card(Lend[{c2}]) ≤ 3) ∧
          Reserve ∈ Book ⇸ Member
```

FIGURE 3.3 – Invariant généré automatiquement

3.1.3 Génération des opérations

L'outil B4MSecure génère automatiquement toutes les opérations de base telles que les constructeurs/destructeurs d'instances, les constructeurs/destructeurs de liens entre instances, les getters/setters

d'attributs, et les getters/setters de liens. La liste des opérations de base pour l'exemple de la bibliothèque est donnée dans le tableau 3.2.

Classe	Member	Book
Constructeur	Member_New	Book_New
Destructeur	Member_Free	Book_Free
Constructeurs de liens	Member_AddLend Member_AddReserve	Book_AddLend Book_AddReserve
Destructeurs de liens	Member_RemoveLend Member_RemoveReserve	Book_RemoveLend Book_RemoveReserve
Getters de liens	Member_GetLend Member_GetReserve	Book_GetLend Book_GetReserve
Setters de liens	Member_SetLend Member_SetReserve	Book_SetLend Book_SetReserve
Total des opérations	**20 opérations**	

TABLEAU 3.2 – Opérations de base générées par B4MSecure

Il est à noter que pour notre exemple il n'y a pas de différence entre les constructeurs de liens et les setters de liens. Mais, dans le cas général un constructeur de lien sert à créer un lien qui n'existe pas entre une instance de la classe source et une instance de la classe cible. Quant aux setters de liens, ils servent à modifier une liaison déjà existante en attribuant une autre instance de la classe cible à l'instance de la classe source. Ainsi, dans notre cas de figure nous gardons les constructeurs de liens et nous supprimons les setters de liens afin d'alléger la spécification et d'éviter les redondances.

Toutes les opérations de base générées par la plate-forme suivent la forme standard d'une substitution pré-conditionnée de la forme :

PRE Précondition **THEN** Action **END**

Elles sont également générées de telle sorte qu'elles ne violent pas les invariants de typage produits automatiquement. Ainsi, la preuve de correction du modèle fonctionnel, dans son état brut, se limite au fait que les opérations de base respectent les multiplicités sur les associations ainsi que les spécificités des attributs. Les autres contraintes spécifiques à l'application et qui sont pas explicitées dans le modèle UML doivent être complétées manuellement par l'analyste. Par exemple, l'emprunt d'un livre réservé ne doit pas être autorisé par l'application. Ainsi, l'opération `Member_AddLend`, dont le code B est donné par la figure 3.4, doit vérifier que l'instance du livre qui fait l'objet de l'emprunt n'est pas liée dans la fonction `Reserve`.

```
Member_AddLend(Instance, Lend_bookValues)=
   PRE
        /*Contraintes de typage*/
        Instance ∈ Member ∧
        Lend_bookValues ∈ Book ∧
        /*Contraintes de multiplicité*/
        card(Lend[{Instance}]) < 3 ∧
        (Instance ↦ Lend_bookValues) ∉ Lend ∧
        Lend_bookValues ∉ ran(Lend) ∧
        /*Contrainte ajoutée manuellement*/
        Lend_bookValues ∉ dom(Reserve)
   THEN
        Lend := Lend ∪ {(Instance ↦ Lend_bookValues)}
   END;
```

FIGURE 3.4 – Spécification en B de l'opération `Member_AddLend`

Outre les opérations de base, il existe les opérations spécifiques définies par les utilisateurs. Pour ce genre d'opérations, la plate-forme génère le squelette de l'opération qui sera complétée par l'analyste. Par exemple, l'opération `TakeReservedBook` de la classe `Member` doit être spécifiée de telle sorte qu'elle respecte les propriétés invariantes. Cette opération, dont la spécification est donnée par la figure 3.5, prend en paramètre un membre et un livre. Elle permet au membre d'emprunter le livre à condition qu'il l'ait réservé auparavant.

```
Member_TakeReservedBook(Instance,ReservedBook)=
       PRE
           /*Conditions imposées par l'invariant*/
           Instance ∈ Member ∧
           ReservedBook ∈ Book ∧
           ReservedBook ∉ ran(Lend) ∧
           card(Lend[Instance]) < 3 ∧
           /*Conditions relatives à l'opération*/
           (ReservedBook ↦ Instance) ∈ Reserve
       THEN
           Lend := Lend ∪{(Instance ↦ ReservedBook)} ||
           Reserve := Reserve − {(ReservedBook ↦ Instance)}
       END;
```

FIGURE 3.5 – Spécification de l'opération `Member_TakeReservedBook`

La spécification complète du modèle fonctionnel est donnée en annexe A.

3.2 Modélisation des règles de contrôle d'accès RBAC

B4MSecure met en œuvre l'approche MDS (*Model Driven Security*) en permettant l'usage du profil SecureUML pour les règles RBAC.

La Figure 3.6 représente le méta-modèle de sécurité intégré à la plate-forme B4MSecure. Dans ce méta-modèle, une permission, représentée par la méta-classe `Permission` est associée à une classe fonctionnelle et contient deux types d'actions : `MethodAction` et `EntityAction`. Une action de type `MethodAction` est rattachée à une seule opération de la classe cible d'une permission et donne l'autorisation d'invoquer cette opération. Les actions de type `EntityAction` sont plus globales, et permettent des accès en lecture, écriture, etc, aux constituants d'une classe fonctionnelle, comme ses attributs et ses associations. Quant aux utilisateurs (méta-classe `User`), ils sont affectés à des rôles (méta-classe `Role`) qui leur donnent des autorisations selon les permissions auxquelles ces rôles sont associés. Le modèle RBAC définit des hiérarchies de rôles par l'association réflexive `superRoles`. La méta-classe `Session` représente le mécanisme de connexion des utilisateurs au système. En effet, un utilisateur se connecte au moyen d'une session, dans laquelle il active un ensemble de rôles parmi ceux qui lui sont affectés.

Outre les concepts du modèle RBAC, le méta-modèle de la plate-forme comprend des extensions pour supporter les notions d'organisation et de délégation de rôles. En effet, le modeleur graphique de B4MSecure permet d'une part, de spécialiser les règles de contrôle d'accès en fonction de l'appartenance des utilisateurs à des organisations différentes, et d'autre part, il offre la possibilité de déléguer des rôles à des utilisateurs. Mais ces notions étant non exploitées dans le cadre de cette thèse, nous laissons le lecteur intéressé se référer à [IDANI et al. 2014].

3.2.1 Exemple illustratif : SI d'une bibliothèque sécurisée

Les règles de contrôle d'accès associées à notre exemple couvrent deux rôles majeurs `Librarian` et `Member` qui représentent respectivement les gestionnaires et les membres inscrits à la bibliothèque. La figure 3.7 montre un exemple d'affectation de trois utilisateurs à ces deux rôles, tels que :

— `Bob` et `John` sont des utilisateurs membres. Et,

— `Alice` est une gestionnaire de la bibliothèque.

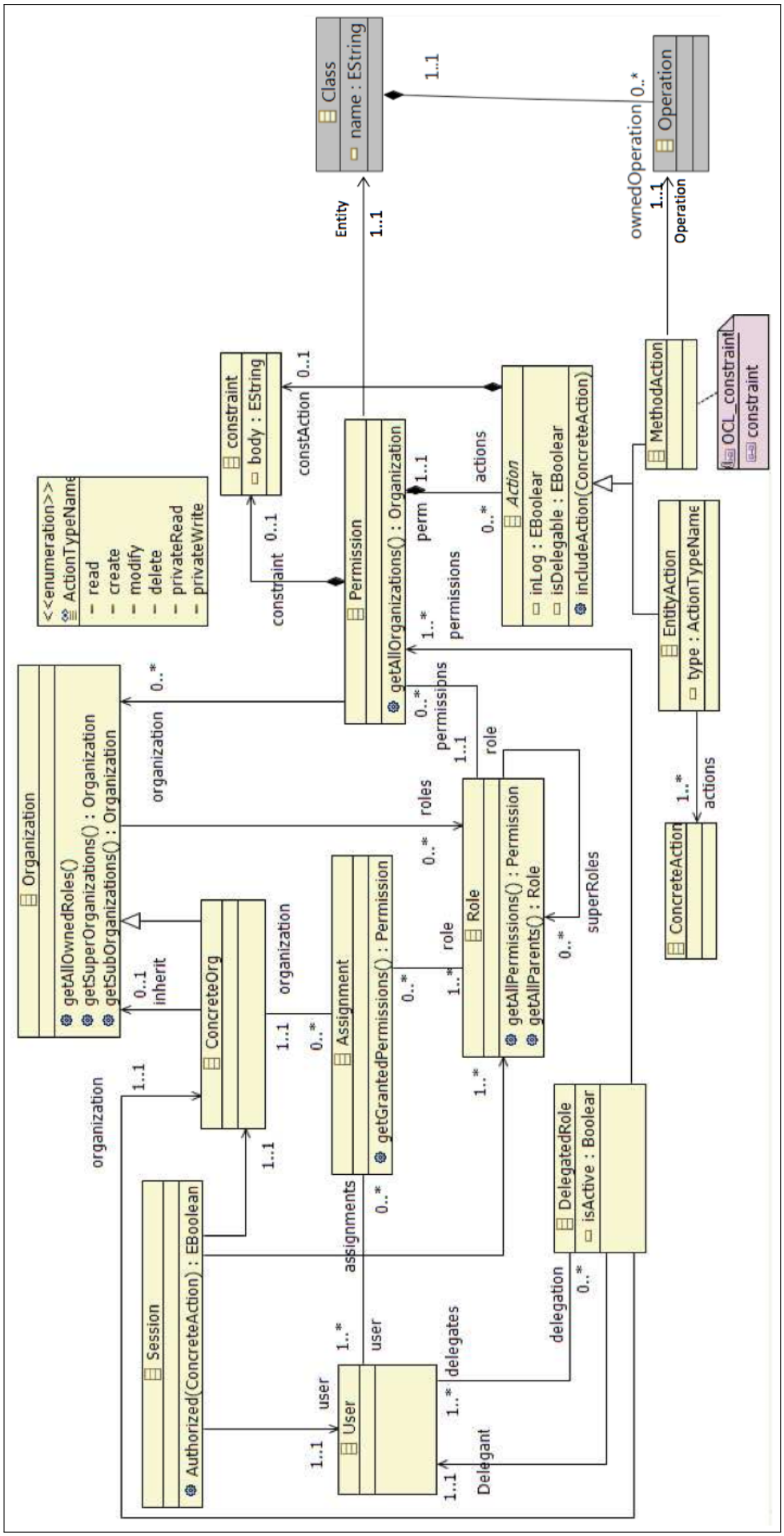


FIGURE 3.6 – Méta-modèle RBAC mis en oeuvre par l'approche B4MSecure

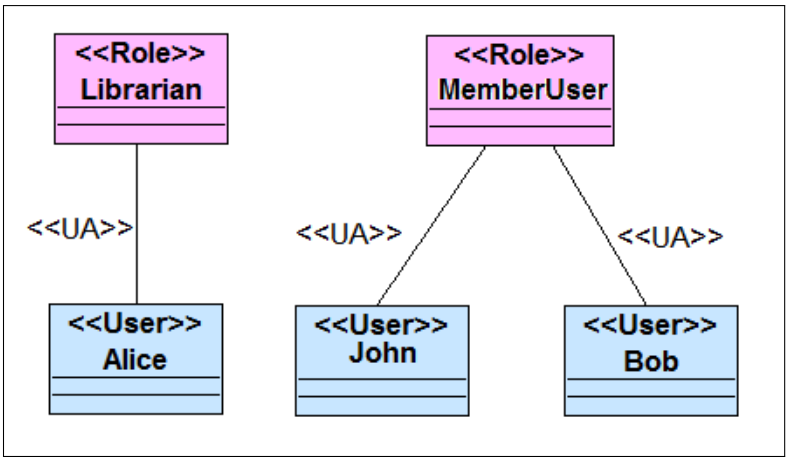


FIGURE 3.7 – Affectation des utilisateurs aux rôles

Ces rôles ont accès aux entités du modèle fonctionnel de la figure 3.2 selon les permissions qui leurs sont associées et dont le modèle SecureUML est donné par la figure 3.8.

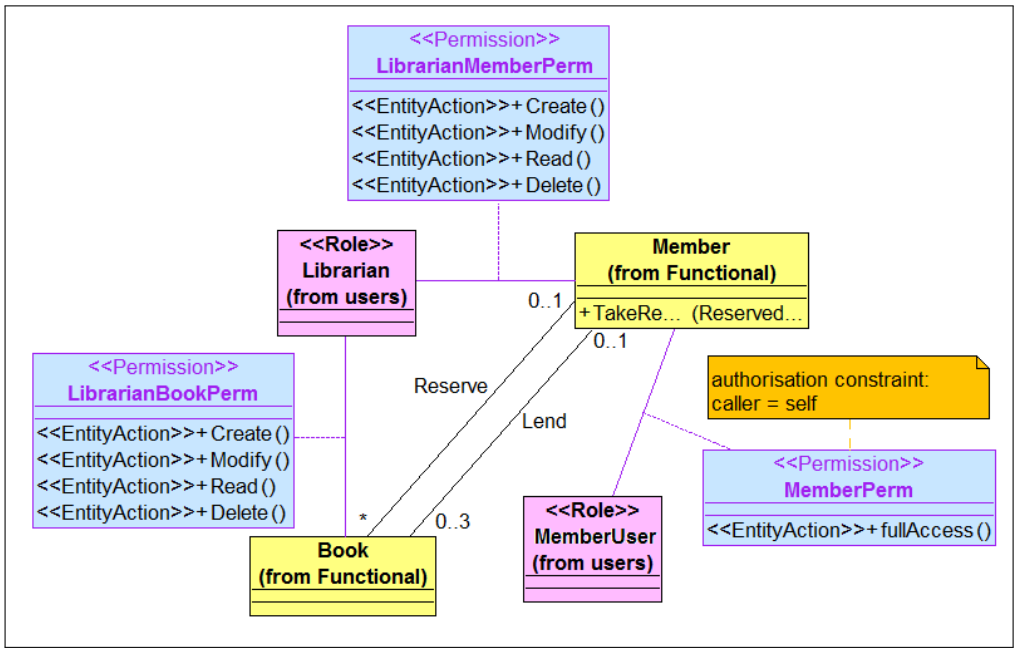


FIGURE 3.8 – Contrôle d'accès aux entités du SI de la biliothèque

Ce modèle intègre les trois permissions suivantes

- **LibrarianBookPerm** : Permet aux gestionnaires de la bibliothèque (les utilisateurs affectés au rôle `Librarian`) d'exécuter les opérations de création, de suppression, de lecture ou de modification sur les instances de la classe `Book`.
- **LibrarianMemberPerm** : Donne aux gestionnaires de la bibliothèque le droit de gérer les membres inscrits. Ainsi, ils peuvent ajouter un nouveau membre, supprimer, modifier ou encore lire les données d'un membre déjà inscrit.

- **MemberPerm** : Cette permission donne un accès total (*full access*) aux utilisateurs de rôle `MemberUser` à l'entité `Member`. Par conséquent, les utilisateurs affectés à ce rôle sont autorisés à exécuter toutes les opérations dérivées de cette classe y compris les opérations manipulant les associations tels que `Member_AddReserve`, `Member_AddLend`, etc. Cependant, une contrainte d'autorisation est associée à cette permission en vue de limiter l'accès de chaque membre uniquement à son espace personnel. Ainsi, il ne peut modifier que les informations qui le concernent et il ne peut emprunter ou réserver un livre que pour son propre compte.

3.2.2 Traduction en B des règles de sécurité

Les spécifications B issues du modèle de sécurité jouent le rôle d'un filtre qui permet de contrôler l'usage des opérations encapsulées dans la machine B fonctionnelle. Ainsi, les utilisateurs interagissent avec le modèle de sécurité qui leur donne accès uniquement aux opérations auxquelles ils ont droit dans la politique de sécurité. Par exemple, `Bob` étant un `MemberUser` il ne pourra utiliser que les opérations de la classe `Member` générées lors de la traduction du modèle fonctionnel en B. Les opérations de la classe `Book` lui sont interdites parce qu'il n'a aucune permission sur cette entité. Ainsi, à chaque opération du modèle fonctionnel, une opération sécurisée est associée dans le modèle de sécurité. L'opération sécurisée se charge de vérifier que l'utilisateur courant dispose d'une permission lui permettant d'appeler l'opération correspondante dans le modèle fonctionnel.

Pour ce faire, deux machines B issues de la traduction du modèle de sécurité sont générées. La première, nommée `UserAssignments`, contient la traduction en B de l'affectation des utilisateurs aux rôles. Quant à la deuxième, nommée `RBAC_Model`, elle inclut la première ainsi que la machine B du modèle fonctionnel et elle contient la traduction des permissions qui servira de filtre pour l'appel aux opérations du modèle fonctionnel.

3.2.2.1 Traduction de l'affectation d'utilisateurs aux rôles

La machine `UserAssignments` contient la formalisation des utilisateurs et leur affectation aux rôles. Elle définit les trois variables `roleOf`, `currentUser` et `Session` qui permettent respectivement d'associer un utilisateur à un rôle ou à un ensemble de rôles, d'identifier l'utilisateur courant du système et de caractériser une session au cours de laquelle un utilisateur se connecte au système et active un ou plusieurs rôles parmi les rôles qui lui sont attribués. D'autres variables peuvent être rajoutées pour spécifier la relation hiérarchique entre les rôles ainsi que les contraintes SSD et DSD. La figure 3.9 donne la partie statique ainsi que l'initialisation de la machine `UserAssignments` de notre exemple.

La partie dynamique de la machine contient les opérations qui permettent de gérer l'affectation des rôles aux utilisateurs ainsi que les opérations de connexion et de déconnexion des utilisateurs. Par exemple, l'opération `Connect` qui sert à connecter un utilisateur au système au travers d'une session est donnée à la figure 3.10.

```
MACHINE
        UserAssignments
SETS
        ROLES={Librarian, MemberUser};
        USERS={Alice, John, Bob, none}
VARIABLES
        roleOf, currentUser, Session
INVARIANT
        roleOf ∈ USERS → ℙ(ROLES) ∧
        currentUser ∈ USERS ∧
        Session ∈ USERS ↔ ROLES ∧
        ∀ (uu)·(uu ∈ USERS ∧ uu ∈ dom(Session) ⇒
                Session[{uu}]⊆ roleOf(uu))
INITIALISATION
        roleOf := {(Bob↦{MemberUser}),(John↦{MemberUser}),
                   (Alice↦{Librarian}),(none ↦∅)} ||
        currentUser := none ||
        Session := ∅
```

FIGURE 3.9 – Partie statique et initialisation de la machine d'affectation des utilisateurs aux rôles

```
Connect(user,roleSet)=
   PRE
       user ∈ USERS ∧ user ∉ dom(Session) ∧
       roleSet ∈ ℙ1(ROLES) ∧ roleSet ⊆ roleOf(user)
   THEN
       Session := Session ∪ ({user} × roleSet)
   END;
```

FIGURE 3.10 – Opération de connection d'un utilisateur

3.2.2.2 Traduction des permissions

La machine `RBAC_Model` permet de redéfinir les opérations fonctionnelles en y introduisant un filtre de sécurité. Ainsi, pour chaque opération fonctionnelle `Operation` une opération `secure_Operation` est générée. Par exemple, la version sécurisée de l'opération `Member_AddLend`, nommée `secure_Member_AddLend` et dont le code B est donné par la figure 3.11, filtre l'accès à l'opération fonctionnelle en ajoutant une clause `SELECT` qui correspond à une garde conditionnant le déclenchement de l'opération fonctionnelle. Elle vérifie, d'une part, que l'utilisateur courant dispose d'un rôle ayant un droit d'exécution de l'opération, et d'autre part, que la contrainte d'autorisation, indiquant que seul le membre en question peut réaliser cette opération, est respec-

tée.

```
secure_Member_AddLend(Instance, Lend_bookValues)=
   PRE
        /*Précondition de l'opération fonctionnelle*/
        Instance ∈ Member ∧
        Lend_bookValues ∈ Book ∧
        card(Lend[{Instance}]) < 3 ∧
        (Instance ↦ Lend_bookValues) ∉ Lend ∧
        Lend_bookValues ∉ ran(Lend) ∧
        Lend_bookValues ∉ dom(Reserve)
   THEN
        /*filtre de sécurité*/
        SELECT
           /*filtre correspondant au rôle*/
           Member_AddLend ∈ isPermitted[currentRole] ∧
           /*filtre de la contrainte d'autorisation*/
           Instance = currentUser
        THEN
           /*Appel à l'opération de la machine fonctionnel*/
           Member_AddLend(Instance,Lend_bookValues)}
   END;
```

FIGURE 3.11 – Version sécurisée de l'opération `Member_AddLend`

Contrairement au filtre correspondant à la contrainte d'autorisation qui doit être intégré manuellement dans la spécification, le filtre correspondant au rôle est généré automatiquement sous la forme : `Operation ∈ isPermitted[currentRole]`. La relation `isPermitted` contient tous les couples (rôles, opérations) permis par la politique de sécurité. Ce filtre vérifie, donc, que l'opération concernée est autorisée à un des rôles activés par l'utilisateur dans une session au moyen de l'opération `Connect`. L'ensemble de ces rôles est sauvegardé dans la variable `currentRole`.

3.2.2.3 Liens entre modèle fonctionnel et modèle de sécurité

Outre la redéfinition des opérations fonctionnelles par introduction des filtres de sécurité, il est possible que certaines de ces opérations fonctionnelles affectent des utilisateurs du SI à des rôles. En effet, dans le cas où des instances d'une classe fonctionnelle représentent des utilisateurs qui sont liés à un rôle particulier, le constructeur de cette classe permet d'associer les utilisateurs liés à ces instances à ce rôle. Par exemple, les utilisateurs du SI de la bibliothèque qui jouent le rôle `MemberUser` sont, en fait, sauvegardés par l'application dans la classe `Member`. Par conséquent, l'opération `secure_Member_New` appelle l'opération fonctionnelle `Member_New` pour créer

une nouvelle instance de la classe `Member`, et en même temps, elle associe ce nouveau membre au rôle `MemberUser` en appelant l'opération du modèle de sécurité `addRoleSafe` (figure 3.12). Cette dernière vérifie que l'utilisateur n'est pas déjà affecté à ce rôle comme elle vérifie qu'il n'y a pas une restriction telle qu'une contrainte SSD pour l'attribution du nouveau rôle à l'utilisateur en question (nous donnons la spécification B de cette opération en annexe A).

```
secure_Member_New(Instance)=
   PRE
        /*Précondition de l'opération fonctionnelle*/
        Instance ∈ MEMBER ∧ Instance ∉ Member
        /*Précondition de l'opération addRoleSafe*/
        ...
   THEN
        /*filtre de sécurité*/
        SELECT
           /*filtre correspondant au rôle*/
           Member_New ∈ isPermitted[currentRole-{MemberUser}]∨
           /*filtre de la contrainte d'autorisation*/
           (Instance = currentUser ∧ MemberUser ∈ currentRole)
        THEN
           /*Appel à l'opération de la machine fonctionnel*/
           Member_New(Instance)
           /*Affectation d'un utilisateur à un rôle*/
           addRoleSafe(Instance,MemberUser)
   END;
```

FIGURE 3.12 – Version sécurisée de l'opération `Member_New`

Dans ce cas l'ensemble abstrait `MEMBER` du modèle fonctionnel prend ses valeurs à partir de l'ensemble `USERS`. Pour expliciter ce lien, nous ajoutons la définition suivante au modèle fonctionnel :

```
DEFINITIONS
   MEMBER==USERS - {none}
```

3.3 Validation des modèles

L'intérêt des spécifications générées par la plate-forme B4MSecure est de pouvoir les valider au moyen d'outils dédiés à la méthode B tels que les outils de preuve, les animateurs, les model-checkers ou encore les solveurs de contraintes.

L'un des avantages majeurs de cette plate-forme est qu'elle produit des spécifications indépendantes qui permettent de mener les activités de validation sur chacun des deux modèles fonctionnel

et de sécurité pris isolément, ainsi que de valider les liens entre les deux modèles et d'analyser l'impact des règles de sécurité sur les opérations fonctionnelles.

Dans ce contexte, des activités de validation ont été suggérées dans [LEDRU et al. 2015] ce qui montrent l'intérêt de la plate-forme pour certains types de validation. Dans cette section, nous en donnons un aperçu en mettant l'accent sur les activités de validation qui nous intéressent, comme nous discutons les limites de ce qui a été proposé.

3.3.1 Validation du modèle fonctionnel

Étant donné que le modèle fonctionnel B produit automatiquement a vocation à être complété manuellement par la prise en compte de nouvelles opérations d'une part et des contraintes d'intégrité spécifiques à l'application d'autre part, il est nécessaire de conduire la preuve de correction du modèle pour disposer d'un modèle cohérent. Un outil de preuve automatique tel que AtelierB peut être utilisé dans ce cas pour identifier les opérations violant l'invariant et les corriger.

Partant d'un modèle fonctionnel structurellement correct où les propriétés invariantes sont respectées par toutes les opérations, nous nous intéressons à l'analyse comportementale du SI. En effet, les spécifications B produites par B4MSecure à partir du diagramme de classes peuvent être animées au moyen d'un outil d'animation ou explorées par un model-checker. Cela permet de voir l'évolution de l'état du système et d'observer l'effet qu'un scénario d'exécution pourrait avoir. Il permet également d'analyser les scénarios de cas d'utilisation normaux. Par exemple, la séquence d'opérations donnée par la figure 3.13 obtenue par animation permet de valider le scénario fonctionnel qui autorise l'emprunt d'un livre (e.g. `bo`) à un membre (e.g. `Bob`). Dans ce scénario, nous commençons par créer le livre `bo` ainsi que le membre `Bob` qui sont des pré-requis pour pouvoir exécuter l'opération `Member_AddLend` permettant au membre en question d'emprunter le livre.

```
Book_New(bo) ;
Member_New(Bob) ;
Member_AddLend(Bob,bo) ;
```

FIGURE 3.13 – Scénario fonctionnel animé par ProB

Cette validation par l'animation a été étudiée dans [LEDRU et al. 2015] où les auteurs ont proposé d'utiliser l'animateur ProB afin de montrer que les scénarios de cas d'utilisation normaux sont réalisables ou d'identifier les opérations manquantes dans un scénario.

Cependant, la validation par l'animation nécessite une connaissance de la séquence d'opérations composant un scénario de cas d'utilisation. Or, dans la plupart des cas les spécifications décrivant les exigences fonctionnelles précisent les objectifs que le SI doit atteindre sans pour autant décrire la façon qui lui permet de le faire. Par conséquent, la validation du comportement fonctionnel peut se ramener à un problème d'atteignabilité en vue de vérifier que le système est capable d'atteindre les objectifs qui lui sont spécifiés et d'évoluer comme prévu.

Par exemple, pour vérifier que le SI de la bibliothèque donne le moyen à un membre d'emprunter un livre déjà emprunté par un autre membre, nous pouvons utiliser un model-checker comme ProB afin d'étudier l'atteignabilité de cet objectif. Nous considérons, dans ce cas, un état initial où un membre `John` a emprunté le livre `bo` et nous cherchons par model-checker le scénario fonctionnel qui permet d'atteindre un état où le membre `Bob` devient l'emprunteur du livre.

Plusieurs scénarios fonctionnels ont été produits par ProB. Parmi ces scénarios nous nous contentons de présenter les scénarios les plus pertinents suivants :

```
Member_RemoveLend(John,bo);
Member_AddLend(Bob,bo);
```

```
Member_AddReserve(Bob,bo);
Member_RemoveLend(John,bo);
Member_TakeReservedBook(Bob,bo);
```

Le premier scénario fonctionnel autorise `Bob` à emprunter le livre après qu'il soit ramené par `John`. Dans le deuxième scénario, le livre `bo` a été réservé à `Bob` ce qui a permis d'activer l'opération `Member_TakeReservedBook` une fois le livre restitué.

Dans notre investigation, cette étude préliminaire est nécessaire pour analyser le comportement fonctionnel du SI dans son état brut avant la considération des règles de contrôle d'accès et leurs impacts sur ces comportements fonctionnels. Nous nous intéressons en particulier à l'étude de l'atteignabilité de certains objectifs dans le modèle fonctionnel dans un premier temps. Ensuite, dans un second temps, nous analysons ces comportements dans le modèle de sécurité afin de vérifier si les filtres de sécurité autorisent de tels comportements qu'ils soient normaux ou malicieux.

3.3.2 Validation du modèle de sécurité

Tout comme le modèle fonctionnel, le modèle de sécurité peut lui aussi faire l'objet d'une validation indépendante. Ainsi, nous pouvons vérifier que les propriétés invariantes sont préservées par les opérations du modèle de sécurité (telles que `connect`, `addSafeRole`, etc.) et que les règles de sécurité sont respectées notamment les règles de hiérarchie entre les rôles, les propriétés de séparation de devoirs, etc.

Ces activités de validation du modèle de sécurité doivent être complétées par l'analyse des liens entre le modèle fonctionnel et le modèle de sécurité pour pouvoir vérifier que les règles de contrôle d'accès remplissent bien leurs objectifs. En effet, certaines propriétés de sécurité doivent être vérifiées à un instant donné, pour un état donné et souvent ne portent pas sur des propriétés invariantes mais plutôt sur les propriétés de l'état fonctionnel en cours. Ces propriétés de sécurité, exprimées souvent au moyen de contraintes d'autorisation, permettent d'autoriser l'usage de certaines actions en fonction d'informations issues du modèle fonctionnel.

Dans ce contexte, les auteurs de [LEDRU et al. 2015] ont proposé d'utiliser les techniques de tests afin de vérifier que les filtres de sécurité se comportent comme prévu. Ils vérifient, ainsi, qu'ils ne bloquent pas l'accès aux opérations autorisées, et qu'ils n'autorisent pas l'accès aux opérations protégées aux utilisateurs dont le rôle ne le permet pas.

Afin de vérifier que les filtres de sécurité n'empêchent pas l'exécution des scénarios de cas d'utilisation normaux, ils ont eu recours aux techniques d'animation. En effet, il suffit d'invoquer la version sécurisée des opérations fonctionnelles après avoir connecté les bons utilisateurs et activé pour chaque utilisateur les bons rôles. Par exemple, le scénario fonctionnel de la figure 3.13 peut être rejoué dans le modèle de sécurité comme indiqué dans la figure 3.14. Pour ceci, il a fallu connecter un utilisateur avec le rôle `Librarian` comme `Alice` pour exécuter les opérations de création d'un livre et d'un membre. Ensuite, pour exécuter l'opération d'emprunt pour `Bob`, il était nécessaire de connecter l'utilisateur `Bob` dans le rôle `MemberUser`.

```
Connect(Alice, {Librarian});
secure_Book_New(bo);
secure_Member_New(Bob);
Connect(Bob, {MemberUser});
secure_Member_AddLend(Bob,bo);
```

FIGURE 3.14 – Scénario animé par ProB dans le modèle de sécurité

Ils définissent également pour chaque règle de sécurité, des tests positifs et des tests négatifs. Les tests positifs exercent le comportement normal de l'application. Ils sont destinés à se terminer par un succès. Quant aux tests négatifs, ils testent la réaction du système face à des comportements interdits et ils doivent se terminer par un échec.

Le but des tests positifs est de vérifier qu'un utilisateur peut utiliser les opérations concernées par une permission s'il dispose d'un rôle autorisé et qu'il satisfait la précondition et la contrainte d'autorisation. Tandis que les tests négatifs vérifient que les filtres de sécurité empêchent l'exécution des opérations sécurisées par les utilisateurs qui n'ont pas les droits requis.

La technique de test définie dans [LEDRU et al. 2015] a cherché à couvrir toutes les permissions et elle a permis de trouver un nombre significatif de défauts dans un SI pré-hospitalier mettant en œuvre des scénarios de prise en charge de patients. Cependant, les auteurs ne disposent pas d'une démarche pour automatiser la génération de cas de test. En effet, 144 tests ont été écrits manuellement pour mener les activités de validation sur le SI étudié. Par conséquent, ces tests sont, d'une part, fastidieux à mener, et d'autre part, ils ne garantissent pas la sécurité du modèle car il est impossible de couvrir tous les rôles et toutes les opérations dans divers états du modèle fonctionnel.

De plus, cette technique ne permet pas de vérifier que la politique de sécurité puisse résister à des scénarios d'attaque interne plus complexes comme celles qui exploitent les failles favorisées par l'évolution dynamique des états fonctionnelles. C'est pour cette raison qu'ils proposent d'utiliser les techniques d'animation. Pour ce faire, les auteurs partent d'un ensemble de scénarios d'attaque préalablement définis qu'ils tentent d'animer dans le modèle de sécurité pour vérifier si la politique de sécurité empêche la réalisation de ces comportements malicieux. Toutefois, cela ne permet pas de déceler de nouveaux scénarios d'attaque non connu à l'avance.

Dans l'approche que nous proposons, la recherche de scénarios d'attaque est vue comme un problème d'atteignabilité tel qu'un scénario d'attaque est une séquence d'opérations tentant d'at-

teindre un état cible critique appelé **état indésirable**. Par exemple, pour vérifier s'il y a une séquence d'opérations qui permet de contourner la contrainte d'autorisation associée à la permission `MemberPerm`, nous supposons qu'un utilisateur `Bob` a emprunté un livre `bo`. Nous vérifions, ainsi, s'il y a une possibilité qu'un autre utilisateur (e.g. `John`) puisse annuler sa réservation et emprunter le livre à sa place. Pour ce faire, il est possible d'utiliser le model-checker ProB pour la recherche d'une trace d'exécution permettant d'atteindre un état satisfaisant la propriété `Lend(John) = bo` que nous caractérisons comme étant un état indésirable. Le model-checker nous a permis d'identifier la trace d'exécution de la figure 5.3. Ce scénario met en évidence une collusion entre `Alice` qui utilise son rôle de bibliothécaire pour supprimer le membre `Bob`. Cette suppression permet d'effacer l'instance du membre `Bob` ainsi que tous les liens en relation avec cette instance y compris la réservation qu'il y avait pour le livre `bo`. Toutes les contraintes empêchant l'emprunt du livre `bo` étant supprimées, `John` peut alors se connecter en tant que `MemberUser` et exercer son droit d'emprunt.

```
Connect(Alice, {Librarian});
secure_Member_Free(Bob);
Connect(John, {MemberUser});
secure_Member_AddLend(John,bo);
```

FIGURE 3.15 – Exemple d'un scénario d'attaque extrait par model-checking

Ceci étant, l'utilisation de ProB purement en mode model-checking n'est certainement pas la solution idéale pour extraire les scénarios d'attaque en particulier, et pour résoudre des problèmes d'atteignabilité en général. En effet, le scénario de la figure 5.3 a été exhibé après avoir exploré plus que 500 états et essayé plus que 1000 transitions. Ce nombre important d'états explorés par ProB sur un exemple simple peut devenir problématique pour des spécifications de plus grande taille. C'est pour cette raison que nous proposons dans cette thèse une nouvelle approche pour l'extraction des scénarios d'attaque basée sur la preuve et la résolution de contraintes. Cette approche, qui sera discutée dans le chapitre suivant, permet d'une part de réduire considérablement l'espace de recherche et d'autre part de guider le model-checker vers la bonne direction.

3.4 Conclusion

Nous avons montré dans ce chapitre l'intérêt d'utiliser la plate-forme B4MSecure qui permet à la fois la modélisation graphique et la validation formelle des règles de sécurité d'un SI en tirant profit des outils disponibles autour de la méthode B.

Nous avons également présenté la structure des spécifications B générées par l'outil, comme nous avons discuté les limites des activités de validation de contrôle d'accès faites sur ce type de spécifications.

Ces dernières serviront de point d'entrée pour les activités de validation que nous menons aussi bien sur le modèle fonctionnel que sur le modèle de sécurité. Par le moyen de ces activités de validation nous cherchons, d'une part, à vérifier que le SI se comporte comme prévu, et d'autre part, à identifier des scénarios d'attaque malicieux visant à compromettre l'un des objectifs de sécurité. Nous avons montré que toutes ces activités de validation font apparaître un problème d'atteignabilité.

Dans ce contexte, un model-checker comme ProB peut être utilisé pour extraire les comportements du SI à partir des spécifications de petite taille. Cependant, il peut échouer à décider quant à l'atteignabilité d'un état dans un système de taille plus importante. Ainsi, nous proposons une approche, que nous décrivons dans le chapitre suivant, permettant de franchir un pas vers une extraction automatisée des séquences d'opérations en vue d'atteindre un état cible et permettant de guider un model-checker vers la bonne direction. Nous proposons également une extension de cette approche pour couvrir l'extraction des scénarios malicieux suivant les quatre types d'attaque interne.

Deuxième partie

Contributions de la thèse

Chapitre 4

Vérification de l'atteignabilité d'un état en B

« *Whenever a theory appears to you as the only possible one, take this as a sign that you have neither understood the theory nor the problem which it was intended to solve.* »

Karl Popper

Sommaire

4.1 Notions préliminaires 69
4.2 Approche basée sur la preuve 72
4.2.1 Extraction d'opérations vérifiant la propriété de succession 72
4.2.2 Extraction d'un chemin symbolique d'atteignabilité 74
4.2.3 Algorithme d'extraction de l'ensemble des chemins symboliques 78
4.2.4 Concrétisation des séquences symboliques 81
4.3 Approche basée sur la résolution de contraintes 82
4.4 Étude de cas : SI d'une bibliothèque 84
4.4.1 Preuve du premier chemin(Q_{symb1} / Q_1) 85
4.4.2 Preuve du deuxième chemin (Q_{symb2} / Q_2) 86
4.5 Comparaison avec des travaux similaires 89
4.6 Bilan et discussion 91

Dans le cadre de notre vérification de la sécurité d'un SI, nous cherchons à identifier des traces d'exécution révélant des comportements malicieux des utilisateurs tentant de contourner les règles de contrôle d'accès. Ainsi, nous analysons l'**atteignabilité** d'un l'état cible, dit indésirable, à partir duquel une attaque peut être perpétrée.

En effet, l'analyse de la propriété d'atteignabilité des états d'un SI permet, certes, de vérifier que le système évolue comme prévu et est capable d'atteindre un état cible à partir d'un état

initial, mais également, que le système est sécurisé en montrant que les états indésirables sont non atteignables.

D'une manière générale, la vérification de la propriété d'atteignabilité consiste à trouver une ou plusieurs traces d'exécution (ou chemins) menant à un état cible. Intuitivement, une exploration exhaustive de l'espace d'états permet d'extraire ces chemins. C'est pour cette raison que le model-checking constitue l'une des techniques les plus utilisées dans ce contexte [CHOUALI et al. 2005, JULLIAND et al. 1999, PLAGGE et LEUSCHEL 2010] en vue d'automatiser le processus d'extraction. Toutefois, ces techniques souffrent du problème de l'explosion combinatoire qui rend difficile la découverte de ces traces d'exécution. Afin de réduire l'espace de recherche, nous proposons deux approches complémentaires, l'une basée sur la preuve et l'autre basée sur la résolution de contraintes. Ces deux approches permettent également de guider le model-checker vers la bonne direction en vue d'extraire des traces d'exécution permettant d'atteindre un état cible.

Les approches que nous proposons peuvent être appliquées en dehors du contexte de la sécurité en vue d'étudier l'atteignabilité d'un état quelconque dans le SI. Ainsi, pour faciliter la compréhension de ces approches, nous faisons dans ce chapitre une abstraction sur les aspects sécuritaires et nous nous concentrons sur la présentation des fondements généraux des approches. Quant à l'applicabilité de ces approches dans le domaine des SI sécurisés, elle sera discutée dans le chapitre suivant.

4.1 Notions préliminaires

Avant de présenter les fondements théoriques de nos approches, nous commençons tout d'abord par définir les notions préliminaires nécessaires à la compréhension des approches que nous proposons. Nous fixons également les notations utilisées et nous donnons les définitions formelles des différentes notions clés que nous adoptons.

Définition 2 (Etat concret).
Un état concret (noté S_{val}) donne une valuation[a] particulière val $(val = (val_1, \ldots, val_n))$ à l'ensemble des variables d'état var $(var = (var_1, \ldots, var_n))$.

Formellement, il peut être exprimé par un prédicat $P(S_{val})$ tel que :

$$P(S_{val}) \hat{=} \bigwedge_{i=1}^{n} (var_i = val_i)$$

Où val_i est une valeur possible de var_i autorisée par l'invariant.

a. Valuer une donnée consiste à lui attribuer une valeur

Définition 3 (Etat symbolique).
Un état symbolique (noté S^E_{P}) est un sous-ensemble E de l'espace d'états caractérisé par un prédicat noté P_E.

Définition 4 (Conformité état concret - état symbolique).
Un état concret S^E_{val} satisfait un état symbolique S^E_{P}, noté $S^E_{val} \vdash S^E_{\mathsf{P}}$, si et seulement si $P(S^E_{val}) \Rightarrow \mathsf{P_E}$.

Exemple. Nous considérons l'état concret S^E_{val} de la figure 4.1 qui représente un état du SI de la bibliothèque, présenté à la section 3.1.1.

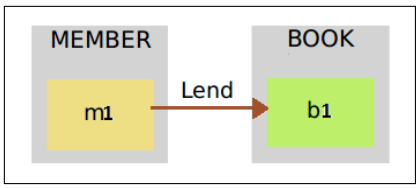


FIGURE 4.1 – Exemple d'un état concret du SI de la bibliothèque

Cet état concret peut être décrit par le prédicat suivant :

$$P(S^E_{val}) \hat{=} Member = \{m_1\} \wedge Book = \{b_1\} \wedge Lend = \{(m_1 \mapsto b_1)\} \wedge Reserve = \emptyset$$

Nous considérons également l'état symbolique S^E_{P} de la figure 4.2 qui caractérise tous les états où le membre $\mathtt{m}_1$ a emprunté le livre $\mathtt{b}_1$. Cet état peut alors être décrit par le prédicat suivant : $\mathsf{P_E} \hat{=} (m_1 \mapsto b_1) \in Lend$.

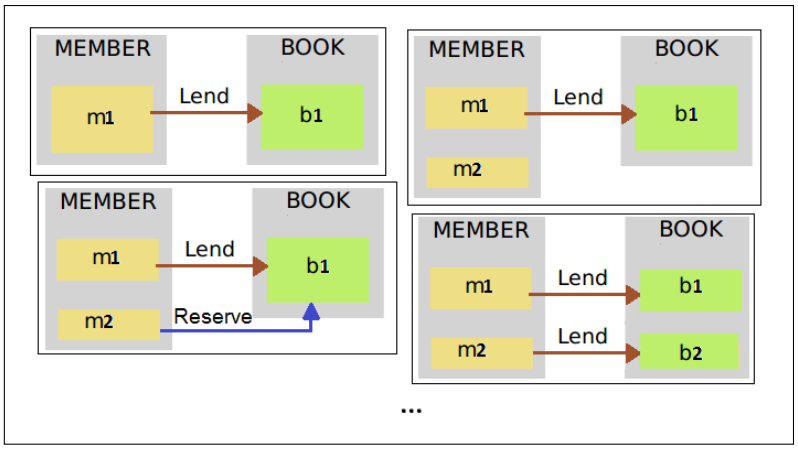


FIGURE 4.2 – Exemple d'un état symbolique du SI de la bibliothèque

L'état concret S^E_{val} étant inclus dans l'état symbolique S^E_{P}, nous pouvons dire que $S^E_{val} \vdash S^E_{\mathsf{P}}$.

Définition 5 (Une exécution).
Etant donnée une opération op_i ayant un nombre n_i de paramètres, $op_i(x_1,\ldots,x_{n_i})$ est appelée une exécution de op_i telle que $x_1,\ldots,x_{n_i}$ est une valuation possible de ses paramètres.
La liste des paramètres $x_1,\ldots,x_{n_i}$ de l'opération op_i sera notée $\mathscr{P}_i$ dans la suite.

Exemple. L'opération `Member_AddLend(m1, b1)` est une exécution de l'opération `Member_AddLend`, telle que $\mathtt{m}_1$ et $\mathtt{b}_1$ sont des valuations possibles pour les deux paramètres de l'opération.

Définition 6 (**Atteignabilité**).
Etant donnés deux prédicats $\mathsf{P_I}$ et $\mathsf{P_F}$ caractérisant respectivement l'état initial et un état cible, un SI peut alors évoluer de l'état initial concret S^I_{val} qui satisfait le prédicat $\mathsf{P_I}$ ($S^I_{val} \vdash S^I_{\mathsf{P}}$) mais ne satisfait pas le prédicat $\mathsf{P_F}$ ($S^I_{val} \nvdash S^F_{\mathsf{P}}$), vers l'état concret S^F_{val} où $\mathsf{P_F}$ devient vrai ($S^F_{val} \vdash S^F_{\mathsf{P}}$).

Nous disons que l'état symbolique S^F_{P} est atteignable à partir de l'état symbolique S^I_{P} si et seulement si :

a- S^I_{P} et S^F_{P} sont disjoints.

b- Il existe au moins une séquence d'opérations Q qui part d'un état concret S^I_{val}, telle que $S^I_{val} \vdash S^I_{\mathsf{P}}$, et mène vers un état concret S^F_{val}, telle que $S^F_{val} \vdash S^F_{\mathsf{P}}$.

La séquence Q sera désignée par :

$$\begin{aligned} Q \quad =< \quad & op_1(\mathscr{P}_1); \\ & op_2(\mathscr{P}_2); \\ & \ldots \\ & op_n(\mathscr{P}_n) > \end{aligned}$$

Nous appelons cette séquence une **exécution** ou un **chemin** sachant que $\mathsf{P_F}$ devient vrai uniquement à l'état cible S^F_{P}.

Nous notons cette atteignabilité par $S^I_{\mathsf{P}} \overset{Q}{\rightsquigarrow} S^F_{\mathsf{P}}$.

Définition 7 (**Succession**).
Si un chemin Q est composé d'une seule opération $op(\mathscr{P})$, alors nous disons que l'état S^F_{P} est un **successeur** de l'état S^I_{P} par l'exécution $op(\mathscr{P})$ et nous notons $S^I_{\mathsf{P}} \overset{<op>}{\rightsquigarrow} S^F_{\mathsf{P}}$.

Exemple. Comme le montre la figure 4.3, la séquence d'opérations Q_1 suivante :

$$Q_1 =< Member\_New(m_1); Book\_New(b_1); Member\_AddLend(m_1, b_1) >$$

permet d'atteindre l'état symbolique S^D_{P} où $P_D \hat{=} (m_1 \mapsto b_1) \in Lend$ à partir de l'état symbolique S^A_{P} représenté par le prédicat $P_A \hat{=} m_1 \notin Member \wedge b_1 \notin Book \wedge (m_1 \mapsto b_1) \notin Lend$.

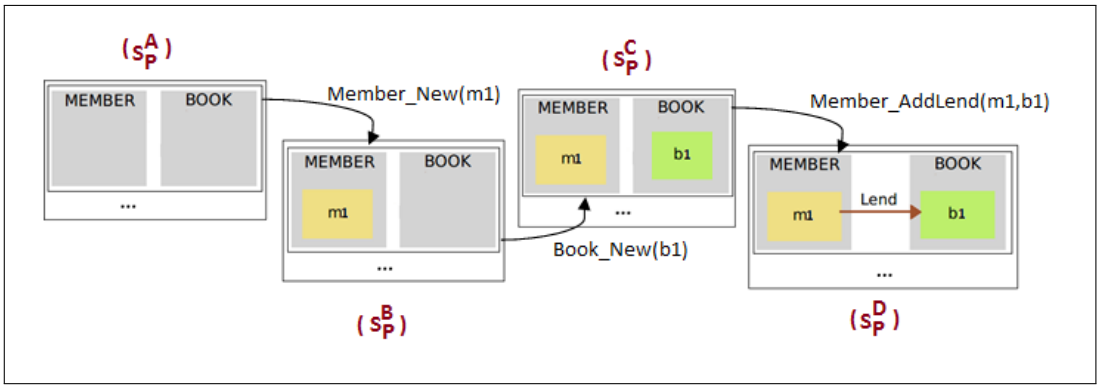


FIGURE 4.3 – Atteignabilité d'un état symbolique

Dans le cas où nous partons de l'état symbolique S_{P}^{C} tel que $P_C \hat{=} m_1 \in Member \wedge b_1 \in Book \wedge (m_1 \mapsto b_1) \notin Lend$, nous pouvons atteindre l'état cible S_{P}^{D} après l'exécution de la séquence composée d'une seule opération $Q_2 =< Member\_AddLend(m_1, b_1) >$. Nous disons, ainsi, que l'état S_{P}^{D} est le successeur de l'état S_{P}^{C}.

Définition 8 (Chemin symbolique).
Un chemin symbolique Q_{symb} est décrit par la séquence des opérations op_i qui le compose sans donner de valuation pour ces opérations :

$$Q_{symb} = <op_1; op_2; \ldots; op_n >$$

Exemple. La séquence $Q_{symb_1} =< Member\_New; Book\_New; Member\_AddLend >$ est dite une séquence d'opérations symbolique telle que Q_1 est une concrétisation possible de cette séquence.

4.2 Approche basée sur la preuve

L'idée principale de cette approche est d'extraire un chemin symbolique Q_{symb}, dans un premier temps, en utilisant un système de preuve inspiré de celui utilisé par l'outil GénéSyst, puis d'utiliser un model-checker comme ProB, dans un deuxième temps, pour identifier les chemins concrets qui en découlent.

Pour conduire la preuve permettant d'extraire Q_{symb}, nous considérons l'extraction d'un chemin composé de plus d'une opération vérifiant la propriété d'atteignabilité comme une généralisation de l'extraction d'un chemin composé d'une seule opération satisfaisant la propriété de succession. Ainsi, nous commençons par expliquer comment extraire une opération op (sans la valuation de ses paramètres) établissant la propriété $S_{\mathsf{P}}^{I} \overset{<op>}{\rightsquigarrow} S_{\mathsf{P}}^{F}$, puis nous montrons comment étendre la technique d'extraction pour exhiber un chemin contenant plusieurs opérations.

4.2.1 Extraction d'opérations vérifiant la propriété de succession

Pour vérifier la propriété de succession nous nous sommes inspirés des formules (C) et (F) proposées dans l'approche GénéSyst (que nous avons présenté dans la section 2.3.2). Ces formules expriment respectivement la possibilité de déclenchement d'une opération à partir d'un état et la possibilité d'atteignabilité d'un autre état. Elles permettent de vérifier, entre autre, si un état du système peut être le successeur d'un autre état. Par conséquent, nous nous servons de ces obligations de preuve après les avoir adaptées à la méthode B classique, en nous basant sur l'analogie donnée dans le tableau 4.1.

	Event-B	**B classique**
Type d'action	Evénement : ev	Opération : $op(x_1,\ldots,x_n)$
Forme générale des actions	Gardée : **SELECT** G **THEN** Action **END**	Préconditionnée : **PRE** P **THEN** Action **END**
Condition de déclenchement des actions	$Garde(ev)$ $\hat{=}$ $\mathsf{fis}(ev)$	$Pre(op)$ $\hat{=}$ $\mathsf{trm}(op)$

TABLEAU 4.1 – Tableau d'analogie entre la méthode B classique et Event-B

En effet, les actions en B événementiel sont exprimées par des événements qui n'ont ni paramètres ni valeur de retour et qui sont décrits par des substitutions gardées. Quant aux actions en B classique, elles sont exprimées par des opérations qui peuvent avoir des paramètres et/ou des valeurs de retours. Elles suivent, souvent, la forme standard des substitutions préconditionnées.

La condition de déclenchement des actions en Event-B se ramène à la condition de faisabilité de l'événement telle qu'elle est établie dans [D. BERT et STOULS]. En effet, selon [ABRIAL et MUSSAT 1998], toute substitution généralisée d'un événement $ev \hat{=} S$ peut se mettre sous une forme normalisée $\mathscr{F}(ev)$ telle que :

$$\mathscr{F}(ev) \hat{=} Garde(ev) \Longrightarrow Action(ev)$$

Où $Garde(ev) = fis(S)$ et $Action(ev) = S$.

Quant à la condition de déclenchement des opérations en B classique, elle se ramène à la terminaison de l'opération. En effet, selon ABRIAL [1996a], toute substitution généralisée d'une opération $op \hat{=} S$ peut se mettre sous la forme normalisée suivante :

$$\mathscr{F}(op) \hat{=} Pre(op) | Action(op)$$

Où $Pre(op) = trm(S)$ et $Action(op) = S$.

Finalement, une opération op établit la propriété de succession $S^I_{\mathsf{P}} \overset{<op>}{\rightsquigarrow} S^F_{\mathsf{P}}$ si elle permet de réaliser **une transition** entre l'état symbolique S^I_{P} et l'état symbolique S^F_{P}. Autrement dit, l'opération op doit être déclenchable à partir d'un certain état concret S^I_{val} tel que $S^I_{val} \vdash S^I_{\mathsf{P}}$ et doit atteindre un autre état concret S^F_{val} tel que $S^F_{val} \vdash S^F_{\mathsf{P}}$.

Ainsi, en nous inspirant des formules (C) et (F) nous proposons les deux lemmes suivants :

Lemme 1: Propriété de déclenchabilité

Une opération op est déclenchable à partir de S^I_{P} s'il existe une valuation des variables d'état var qui satisfait le prédicat $\mathsf{P_I}$, et il existe au moins une valuation de ses paramètres $\mathscr{P}$ qui garantit la terminaison de l'opération (i.e. sa précondition). Par conséquent, la preuve de déclenchabilité d'une opération op établit le lemme suivant :

$$\exists \mathscr{P}, var \cdot (\mathsf{P_I} \wedge Pre(op)) \qquad (1)$$

Lemme 2: Propriété d'atteignabilité

Tant qu'une opération op est déclenchable à partir de S^I_{P}, elle atteint S^F_{P} si et seulement si il existe une exécution de son action $Action(op)$ qui établit $\mathsf{P_F}$. Ainsi, la preuve d'atteignabilité d'une opération op établit le lemme suivant :

$$\exists \mathscr{P}, var \cdot (\mathsf{P_I} \wedge Pre(op) \Rightarrow \neg[Action(op)]\neg \mathsf{P_F}) \qquad (2)$$

La satisfaction des deux propriétés (1) et (2) implique la satisfaction de l'obligation de preuve la plus générale suivante :

Obligation de Preuve 1: Preuve de la propriété de succession

$$S^I_{\mathsf{P}} \overset{<op>}{\rightsquigarrow} S^F_{\mathsf{P}} \Leftrightarrow \exists \mathscr{P}, var \cdot (\mathsf{P_I} \wedge Pre(op) \wedge \neg[Action(op)]\neg \mathsf{P_F}) \qquad (3)$$

Afin d'identifier toutes les opérations op satisfaisant la propriété de succession $S^I_{\mathsf{P}} \overset{<op>}{\rightsquigarrow} S^F_{\mathsf{P}}$, nous conduisons l'obligation de preuve de la formule (3) pour toutes les opérations de la spécification et nous retenons celles pour lesquelles la preuve réussit. L'utilisation d'un outil de preuve automatique comme AtelierB peut aider à décharger ces obligations de preuve.

4.2.2 Extraction d'un chemin symbolique d'atteignabilité

En considérant la propriété de succession comme un cas particulier de la propriété d'atteignabilité, nous appliquons le même principe décrit ci-dessus pour extraire un chemin composé de plusieurs opérations $Q_{symb} = <op_1; op_2; \ldots; op_n>$. Pour ce faire, nous construisons étape par étape les successeurs intermédiaires $S^{E_i}_{\mathsf{P}}$ entre S^I_{P} et S^F_{P} tels que (Figure 4.4) :

$$\begin{array}{c} S^I_{\mathsf{P}} \overset{<op_1>}{\rightsquigarrow} S^{E_1}_{\mathsf{P}}; \\ S^{E_1}_{\mathsf{P}} \overset{<op_2>}{\rightsquigarrow} S^{E_2}_{\mathsf{P}}; \\ \vdots \\ S^{E_{i-1}}_{\mathsf{P}} \overset{<op_i>}{\rightsquigarrow} S^{E_i}_{\mathsf{P}}; \\ \vdots \\ S^{E_{n-1}}_{\mathsf{P}} \overset{<op_n>}{\rightsquigarrow} S^F_{\mathsf{P}} \end{array}$$

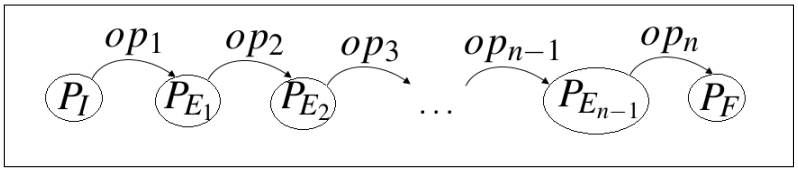


FIGURE 4.4 – Chemin symbolique d'atteignabilité

D'une manière générale, les prédicats P_{E_i} caractérisant les états intermédiaires sont construits à partir des préconditions des opérations op_{i+1} qui sont déclenchées à partir des états $S_{\mathsf{P}}^{E_i}$. Ainsi, nous extrayons les opérations symboliques op_i construisant Q_{symb} en utilisant une technique de recherche avec retour en arrière qui commence par l'extraction de l'opération op_n puis op_{n-1} et ainsi de suite jusqu'à arriver à l'opération op_1. Cette dernière doit être déclenchable à partir de l'état initial.

Nous donnons dans ce qui suit les étapes à suivre pour l'extraction de Q_{symb}. Nous détaillons également les obligations de preuve qui doivent être satisfaites par chacune des opérations dans la séquence.

Première itération : extraction de op_n

Comme nous utilisons une technique de recherche par retour en arrière, nous commençons par la recherche de la dernière opération op_n dans la séquence Q_{symb}. Pour extraire cette opération, nous nous basons sur la proposition suivante :

Proposition 1

S'il existe une opération op telle que $S_{\mathsf{P}}^{E} \overset{<op>}{\rightsquigarrow} S_{\mathsf{P}}^{F}$, alors $S_{\neg\mathsf{P}}^{F} \overset{<op>}{\rightsquigarrow} S_{\mathsf{P}}^{F}$

Démonstration. Selon la définition 6 de l'atteignabilité, si un état S_{P}^{F} est atteignable à partir d'un état S_{P}^{E}, alors les deux états S_{P}^{E} et S_{P}^{F} sont disjoints. Par conséquent, nous avons :

$$\forall var \cdot (\mathsf{P_E} \wedge \mathsf{P_F} \Rightarrow false), \text{ alors :}$$
$$\forall var \cdot (\mathsf{P_E} \Rightarrow \neg \mathsf{P_F}).$$

Ceci implique que l'état S_{P}^{E} est inclus dans l'état qui satisfait $\neg\mathsf{P_F}$ (noté $S_{\neg\mathsf{P}}^{F}$) : $S_{\mathsf{P}}^{E} \subseteq S_{\neg\mathsf{P}}^{F}$. Ainsi, s'il existe une opération op qui atteint S_{P}^{F} à partir de S_{P}^{E}, alors elle permet d'atteindre cet état à partir d'un état qui satisfait $\neg\mathsf{P_F}$ (Figure 4.5). □

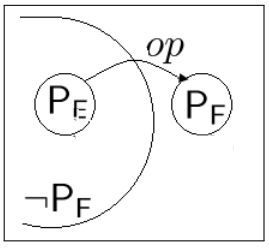


FIGURE 4.5 – Illustration de la proposition 1

Finalement, selon la proposition 1, l'opération op_n doit satisfaire la propriété suivante :

$$S_{\neg\mathsf{P}}^{F} \overset{<op_n>}{\rightsquigarrow} S_{\mathsf{P}}^{F}$$

Ainsi, par application de la formule (3), l'opération op_n doit établir l'obligation de preuve suivante :

$$\exists \mathscr{P}_n, var \cdot (Pre(op_n) \wedge \neg\mathsf{P_F} \wedge \neg[Action(op_n)]\neg\mathsf{P_F}) \qquad (4)$$

Itérations suivantes : extraction de op_i $(i \in \{1 \ldots (n-1)\})$

Maintenant que nous avons $\mathsf{P_F}$ et op_n, nous pouvons déduire le prédicat $\mathsf{P_{E_{n-1}}}$. Pour ce faire, nous nous basons sur les deux propositions 2 et 3 données ci-dessous.

Ayant le prédicat $\mathsf{P_{E_{n-1}}}$ caractérisant l'état $S_{\mathsf{P}}^{E_{n-1}}$ nous pouvons extraire l'opération op_{n-1} en appliquant la même procédure utilisée à l'itération précédente. Par conséquent, nous déduisons le prédicat $\mathsf{P_{E_{n-2}}}$ qui nous permet d'extraire op_{n-2} et ainsi de suite.

Proposition 2

$\forall var \cdot (\mathsf{P_{E_i}} \Rightarrow \neg \mathsf{P_F})$ pour tout $i \in \{1..(n-1)\}$.

Démonstration. Selon la définition 6 de l'atteignabilité, le prédicat $\mathsf{P_F}$ est initialement faux et ne devient vrai que lorsqu'on atteint l'état cible S_{P}^{F}. Ainsi, $\mathsf{P_F}$ est faux dans tous les états $S_{\mathsf{P}}^{E_i}$ et par conséquent tous les états $S_{\mathsf{P}}^{E_i}$ tels que $i \in \{1..(n-1)\}$ satisfont $\neg \mathsf{P_F}$ (Figure 4.6). Par conséquent, nous avons :

$$S_{\mathsf{P}}^{E_i} \subseteq S_{\neg \mathsf{P}}^{F}, \text{ ce qui implique :}$$
$$\forall var \cdot (\mathsf{P_{E_i}} \Rightarrow \neg \mathsf{P_F}).$$

□

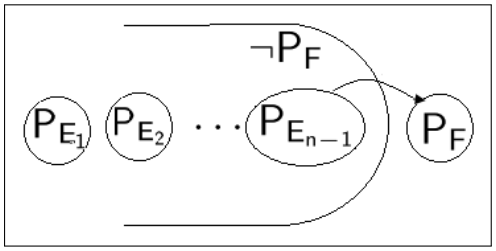


FIGURE 4.6 – Illustration de la proposition 2

D'une manière plus générale, chaque prédicat $\mathsf{P_{E_j}}$ est non satisfait dans tous les états $S_{\mathsf{P}}^{E_i}$ tels que $i < j$ (Figure 4.7), d'où le corollaire suivant :

Corollaire 1

$\forall var \cdot (\mathsf{P_{E_m}} \Rightarrow \bigwedge_{k=m+1}^{n-1} \neg \mathsf{P_{E_k}})$ pour tout $m \in \{1..(n-2)\}$.

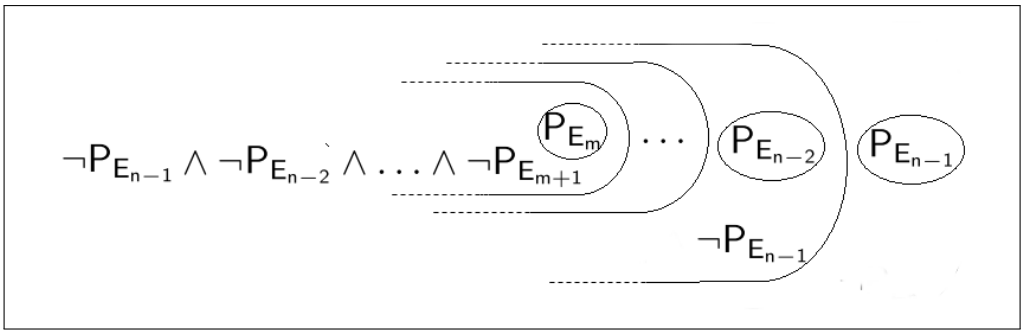


FIGURE 4.7 – Illustration du corollaire 1

Proposition 3

$P_{E_i} \widehat{=} Pre(op_{i+1})$ for each $i \in \{1..(n-1)\}$.

Démonstration. Chaque opération op_{i+1} est déclenchable à partir de l'état qui satisfait $Pre(op_{i+1})$. Par conséquent, étant donné que l'état $S_P^{E_i}$ permet de déclencher op_{i+1}, alors il doit être inclus dans l'état satisfaisant $Pre(op_{i+1})$: $P_{E_i} \Rightarrow Pre(op_{i+1})$ (Figure 4.8).

Nous avons aussi $S_P^{E_i} \overset{<op_{i+1}>}{\rightsquigarrow} S_P^{E_{i+1}}$. Il existe donc une opération op_{i+1} qui permet d'atteindre $S_P^{E_{i+1}}$ à partir de l'état satisfaisant $Pre(op_{i+1})$: $S_{Pre(op_{i+1})}^{E_i} \overset{<op_{i+1}>}{\rightsquigarrow} S_P^{E_{i+1}}$. Ainsi, nous pouvons choisir le prédicat P_{E_i} tel que $P_{E_i} \widehat{=} Pre(op_{i+1})$.

□

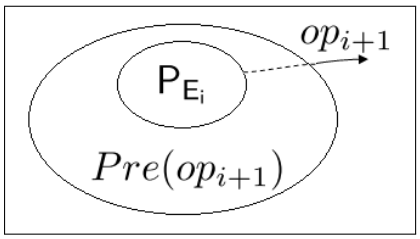


FIGURE 4.8 – Illustration de la proposition 3

A partir de la proposition 3 et du corollaire 1, nous déduisons le corollaire suivant :

Corollaire 2

$\forall var \cdot (P_{E_m} \Rightarrow \bigwedge_{k=m+1}^{n-1} \neg Pre(op_{k+1}) \wedge Pre(op_{m+1}))$ pour tout $m \in \{1..(n-2)\}$.

Plus précisément, nous cherchons, parmi les opérations de la spécification, l'opération op_i qui vérifie : $S_P^{E_{i-1}} \overset{<op_i>}{\rightsquigarrow} S_P^{E_i}$, tels que $S_P^{E_i}$ satisfait la conjonction $\bigwedge_{k=i+1}^{n-1} \neg Pre(op_{k+1}) \wedge Pre(op_{i+1})$ comme il a été démontré par le corollaire 2.

Nous rappelons que toutes les opérations op_{k+1} et op_{i+1} ont été déjà extraites dans les itérations précédentes.

En suivant le même corollaire, nous déduisons que $S_{\mathsf{P}}^{E_{i-1}}$ satisfait $\bigwedge_{k=i}^{n-1} \neg Pre(op_{k+1}) \wedge Pre(op_i)$. De plus, selon la proposition 2, les deux états $S_{\mathsf{P}}^{E_{i-1}}$ et $S_{\mathsf{P}}^{E_i}$ doivent satisfaire $\neg \mathsf{P_F}$. Finalement, l'opération op_i doit vérifier la propriété suivante :

$$S_P^{E_{i-1}} \cap \bigcap_{k=i}^{n-1} S_{\neg P}^{E_k} \cap S_{\neg P}^{F} \overset{<op_i>}{\rightsquigarrow} S_P^{E_i} \cap \bigcap_{k=i+1}^{n-1} S_{\neg P}^{E_k} \cap S_{\neg P}^{F}$$

Par conséquent, dans chaque itération i, nous cherchons à extraire l'opération op_i qui satisfait l'obligation de preuve suivante :

$$\exists \mathscr{P}_i, var \cdot \Big(\begin{array}{l} Pre(op_i) \wedge \bigwedge_{k=i}^{n-1} \neg Pre(op_{k+1}) \wedge \neg \mathsf{P_F} \wedge \\ \neg [Action(op_i)] \neg (Pre(op_{i+1}) \wedge \bigwedge_{k=i+1}^{n-1} \neg Pre(op_{k+1}) \wedge \neg \mathsf{P_F}) \end{array} \Big) \qquad (5)$$

4.2.3 Algorithme d'extraction de l'ensemble des chemins symboliques

En résumé, notre approche consiste à procéder récursivement et à extraire à chaque récursion l'ensemble des opérations symboliques $Symbolic_{op}$ qui établissent la propriété de succession entre un état cible S_{P}^{F} et un état source S_{P}^{E} à partir de l'ensemble des opérations de la spécification $Opset$. Lors de la première récursion, l'état cible correspond à la cible principale que nous cherchons à atteindre et l'état source est caractérisé par la négation de l'état cible ($\mathsf{P_E} \hat{=} \neg \mathsf{P_F}$). Ensuite, la recherche s'effectue à l'intérieur de l'état source S_{P}^{E} en le partitionnant récursivement jusqu'à ce que l'état initial soit atteint ($\mathsf{P_I} \Rightarrow \mathsf{P_{F'}}$). Nous calculons à chaque étape de la récursion, les nouveaux états $S_{\mathsf{P}}^{E'}$ et $S_{\mathsf{P}}^{F'}$ pour chaque opération o_i dans l'ensemble des opérations $Symbolic_{op}$ tels que $S_{\mathsf{P}}^{E} \overset{<o_i>}{\rightsquigarrow} S_{\mathsf{P}}^{F}$ lors de l'itération précédente comme suit :

$$\begin{array}{ll} \mathsf{P_{F'}} \hat{=} & \mathsf{P_E} \wedge Pre(o_i) \\ \mathsf{P_{E'}} \hat{=} & \mathsf{P_E} \wedge \neg Pre(o_i) \end{array}$$

L'algorithme décrivant notre approche est donné dans la figure 4.9. Le partitionnement qui en découle est illustré par la figure 4.10.

1. $Extract\_Symbolic\_Paths(Set_{Q_{symb}}, Opset, \mathsf{P_I}, \mathsf{P_F}, \mathsf{P_E}, Q_{symb})\{$
2. /* Initialisation de $Set_{Q_{symb}}$ à la première récursion */
3. if $Set_{Q_{symb}} == null$ then
4. $Set_{Q_{symb}} = \emptyset$;
5. endif

6. /* L'état initial n'est toujours pas rencontré */
7. if $(\mathsf{P_I} \not\Rightarrow \mathsf{P_F} \wedge \mathsf{P_F} \not\models false \wedge Opset \neq \emptyset)$ then
8. $Symbolic_{op} = \emptyset$;
9. /* Extraction des opération satisfaisant la propriété de succession */
10. for each $o_i \in Opset$ do
11. if $\exists \mathscr{P}_i, var \cdot ((\mathsf{P_E} \wedge Pre(o_i) \wedge \neg[Action(o_i)]\neg\mathsf{P_F})$ then
12. $Symbolic_{op} = Symbolic_{op} \cup \{o_i\}$;
13. endif
14. enddo
15. /* Partitionnement des états et appel récursif */
16. if $Symbolic_{op} \neq \emptyset$ then
17. for each $o_i \in Symbolic_{op}$ do
18. $Q_1 = o_i$; Q_{symb} ;
19. $\mathsf{P_{E_i}} = \mathsf{P_E} \wedge Pre(o_i)$;
20. $\mathsf{P_{E_{i-1}}} = \mathsf{P_E} \wedge \neg Pre(o_i)$;
21. $Opset_1 = Opset - \{o_i\}$;
22. $Extract\_Symbolic\_Paths(Set_{Q_{symb}}, Opset_1, \mathsf{P_I}, \mathsf{P_{E_i}}, \mathsf{P_{E_{i-1}}}, Q_1)$;
23. enddo
24. endif

25. /* L'état initial est atteint */
26. elseif $(\mathsf{P_I} \Rightarrow \mathsf{P_F})$ then
27. $Set_{Q_{symb}} = Set_{Q_{symb}} \cup \{Q_{symb}\}$;
28. endif
29. $\}$

FIGURE 4.9 – Algorithme d'extraction de l'ensemble des chemins symboliques

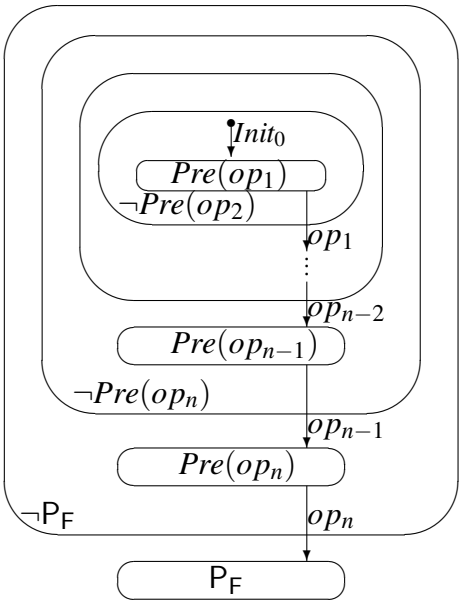


FIGURE 4.10 – Schéma illustratif du principe de partitionnement des états

- **Terminaison de l'algorithme :** La terminaison est assurée par la condition d'arrêt des appels récursifs :

$$P_I \not\Rightarrow P_F \wedge P_F \not\models false \wedge Opset \neq \emptyset$$

  — $P_I \not\Rightarrow P_F$: Cette condition vérifie à chaque récursion si l'état initial est inclus dans l'état cible. Dans ce cas, nous déduisons que l'état initial est atteint et qu'une séquence vérifiant la propriété d'atteignabilité et pouvant être déclenchée à partir de l'état initial est trouvée.

  — $P_F \not\models false$: Par cette condition, nous vérifions que l'état cible est un état valide, sinon l'algorithme s'arrête.

  — $Opset \neq \emptyset$: Étant donné que les états sources sont inférés à partir de la négation de la précondition de l'opération invoquée à l'étape précédente, chaque opération ne peut apparaître dans un chemin plus d'une fois ce qui justifie la soustraction de l'opération extraite de l'ensemble des opérations ($Opset_1 = Opset - \{o_i\}$). Ainsi, la longueur maximale d'un chemin ne peut pas excéder le nombre des opérations de la machine B. Par conséquent, l'algorithme s'arrête systématiquement quand il aura tenté toutes les opérations sans atteindre l'état initial.

- **Complétude de l'algorithme :** Bien que notre algorithme est capable d'extraire des chemins vérifiant l'atteignabilité d'une propriété donnée (voir section 4.4) et que son efficacité a été démontrée sur plusieurs études de cas, il échoue dans l'extraction de chemins invoquant la même opération plus d'une fois à cause de la négation de la précondition, comme nous l'avons déjà expliqué. De plus, pour la même raison les chemins extraits ne peuvent pas contenir deux opérations dont la précondition de l'une est incluse dans

la précondition de l'autre (i.e $Pre(op_i) \Rightarrow Pre(op_j)$ tel que $i < j$). Dans ce cas particulier, notre algorithme n'extrait qu'une partie d'un chemin. Par exemple, si nous avons un chemin $Q_{symb} =< op_1; op_2; op_3 >$ qui mène de l'état initial vers un état cible S^F_{P} et que $Pre(op_2) \Rightarrow Pre(op_3)$, alors l'algorithme ne pourra extraire que la séquence $< op_1; op_3 >$ comme l'illustre la figure 4.11.

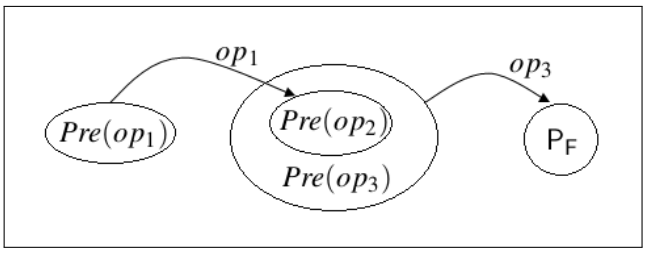


FIGURE 4.11 – Limite de l'algorithme de recherche de séquences symboliques

4.2.4 Concrétisation des séquences symboliques

Afin d'extraire des séquences concrètes, nous proposons de compléter la recherche symbolique avec les techniques de model-checking. En effet, un model-checker peut être utilisé pour trouver les valuations des paramètres des opérations symboliques, comme il peut aider à combler l'une des restrictions de la recherche symbolique notamment l'identification de séquences contenant des répétitions d'opérations.

Ainsi, ayant une séquence symbolique $Q_{symb} =< op_1; op_2; \ldots; op_n >$, nous utilisons le model-checker ProB pour extraire la séquence concrète $Q =< op_1(\mathscr{P}_1); op_2(\mathscr{P}_2); \ldots; op_n(\mathscr{P}_n) >$. Le model-checker peut être utilisé de deux manières différentes :

(i) En le guidant par imposition de l'ordre des appels des opérations. Nous utilisons donc la guidance du model-checker par l'algèbre de processus CSP. Dans le contrôleur CSP nous précisons les noms des opérations sans donner de valuations comme illustré dans la figure 4.12. Le model-checker suit alors cette trace jusqu'à ce qu'il atteigne le but nommé **goal**.

$$\texttt{MAIN} = op_1 \rightarrow op_2 \rightarrow \ldots \rightarrow op_n \rightarrow \textbf{goal} \rightarrow \texttt{STOP}$$

FIGURE 4.12 – Contrôleur CSP pour guider le model-checker pour l'extraction de la séquence concrète

Cette technique permet d'extraire des chemins concrets menant à l'état cible en offrant une guidance efficace au model-checker. Cependant, elle ne permet pas l'extraction de séquences contenant des répétitions pour une même opération.

(ii) En réduisant l'espace d'états par élimination des opérations sans intérêt pour l'atteignabilité de la propriété définie. En effet, la recherche symbolique effectuée permet de donner une idée des opérations pouvant être impliquées pour atteindre la cible. Seules ces opérations seront fournies au model-checker, sans préciser leur séquencement, en vue d'éviter l'explora-

tion de transitions inutiles. Ainsi, en explorant uniquement ces transitions, le model-checker donnera non seulement la valuation des paramètres mais également produira, s'il y a lieu, les chemins invoquant la même opération plus d'une fois.

4.3 Approche basée sur la résolution de contraintes

Nous proposons dans cette section une approche qui se base sur le même principe que l'approche précédente, notamment la preuve de succession et la technique de recherche par retour en arrière. Mais contrairement à l'approche précédente, cette approche permet l'extraction d'une séquence concrète Q menant à une cible donnée sans avoir recours à un model-checker pour identifier les valuations des paramètres des opérations. En effet, nous proposons d'utiliser un solveur de contraintes tel que ProB Calculator [1] pour résoudre la formule de succession (3). L'avantage d'un solveur de contraintes par rapport à un outil de preuve comme AtelierB, est qu'il permette de donner les valuations des paramètres pour lesquelles la propriété est vraie. Ainsi, en vue de vérifier si une opération op établit la propriété de succession $S^I_{\mathsf{P}} \overset{<op>}{\rightsquigarrow} S^F_{\mathsf{P}}$ et en même temps déterminer les exécutions $op(\mathscr{P})$ satisfaisant cette propriété, nous proposons de résoudre le problème à contraintes suivant :

$$\{\mathscr{P} | S \wedge R \wedge \exists var \cdot (Inv \wedge \mathsf{P_I} \wedge Pre(op) \wedge \neg[Action(op)]\neg \mathsf{P_F})\} \qquad \underline{(3)'}$$

Tels que (voir la remarque 2.1) :

— S : la conjonction des prédicats de valuations des ensembles énumérés et des ensembles abstraits de la machine B.

— R : les propriétés sur les constantes issues de la clause PROPERTIES de la machine B.

— Inv : l'invariant de la machine B.

Pour extraire un chemin concret composé de plusieurs opérations menant à un état cible, nous suivons un algorithme similaire à l'algorithme donné par la figure 4.9. Nous procédons donc récursivement et nous partitionnons à chaque itération l'état source en nous basant sur la précondition de l'**exécution** extraite lors de l'itération précédente. Mais, contrairement à l'algorithme précédent, cet algorithme autorise l'invocation de la même opération plus d'une fois avec des valuations différentes pour ses paramètres. En effet, comme nous exhibons des exécutions où les paramètres sont évalués, nous remplaçons les variables correspondant à ces derniers par leurs valeurs dans la précondition. Ainsi, l'état exprimé par la négation de cette précondition est moins abstrait. Il peut donc autoriser la déclenchabilité de la même opération avec des valuations différentes de ses paramètres. Ceci étant, pour garantir la terminaison de l'algorithme nous imposons une longueur maximale aux chemins extraits.

Nous donnons l'algorithme de cette approche dans la figure 4.13 en soulignant les différences avec l'algorithme précédent.

1. https://www3.hhu.de/stups/prob/index.php/ProB_Logic_Calculator

Remarque 4.1 [Complétude de l'approche basée sur la résolution de contraintes]

*L'algorithme 4.13 échoue à extraire des chemins invoquant la même **exécution** plus d'une fois à cause de la négation de la précondition de cette exécution. Mais, nous supposons que dans un SI où nous cherchons à étudier l'évolution des états fonctionnels, il n'est pas possible de répéter la même action plusieurs fois. Par exemple, une entité d'une classe n'est créée qu'une seule fois. Elle peut être modifiée plusieurs fois, mais sans qu'elle soit remise à la même valeur. Cependant, nous pouvons imaginer un scénario où nous avons besoin de créer une entité d'une classe, de la supprimer, puis de la recréer de nouveau... Pour extraire ce genre de scénarios, qui sont souvent utilisés pour réaliser des attaques masquées ou des attaques planifiées, il est nécessaire de définir plusieurs étapes d'extraction de chemin (voir chapitre 5).*

```
1. Extract_Concret_Paths(Set_Q, Opset, P_I, P_F, P_E, Q){
2.    /* Initialisation de Set_Q à la première récursion */
3.    if Set_Q == null then
4.       Set_Q = ∅ ;
5.    endif
6.    /* L'état initial n'est toujours pas rencontré */
7.    if (P_I ⇏ P_F ∧ P_F ⊭ false∧ length(Q) < max_length ) then
8.       Concret_op = ∅ ;
9.       for each o_i ∈ Opset do
10.      Param = {P | S ∧ R ∧ ∃var · (Inv ∧ P_E ∧ Pre(o_i) ∧ ¬[Action(o_i)]¬P_F)} ;
11.         if   Param ≠ ∅ then
12.            Concret_op = Concret_op ∪ o_i[Param] ;
13.         endif
14.      enddo
15.      if Concret_op ≠ ∅ then
16.         for each o_i ∈ Concret_op do
17.            Q_1 = o_i ; Q ;
18.            P_E_i = P_E ∧ Pre(o_i) ;
19.            P_E_i-1 = P_E ∧ ¬Pre(o_i) ;
-              ~~Opset_1 = Opset − {o_i} ;~~
20.            Extract_Concret_Paths(Set_Q, Opset, P_I, P_E_i, P_E_i-1, Q_1) ;
21.         enddo
22.      endif
23.   /* L'état initial est atteint */
24.   elseif (P_I ⇒ P_F) then
25.      Set_Q = Set_Q ∪ {Q} ;
26.   endif
27.}
```

FIGURE 4.13 – Algorithme d'extraction de l'ensemble des chemins concrets

4.4 Étude de cas : SI d'une bibliothèque

Afin d'illustrer nos deux approches nous considérons le modèle fonctionnel de l'exemple du SI d'une bibliothèque décrit dans la section 3.1.1.

Nous utilisons nos approches pour vérifier qu'il existe une séquence d'exécutions dans le système qui permet au membre `me` d'emprunter le livre `bo`. Nous cherchons donc des chemins qui partent d'un état initial concret S^I_{val} satisfaisant l'état symbolique S^I_{P} tel que :

$$\mathsf{P_I} \hat{=} \texttt{me} \in \texttt{Member} \land \texttt{bo} \in \texttt{Book} \land \texttt{(me} \mapsto \texttt{bo)} \notin \texttt{Lend}$$

Pour cet exemple, nous choisissons l'état initial concret S^I_{val} qui satisfait $\mathsf{P_I}$. Nous spécifions donc la clause `INITIALISATION` de la machine B comme indiqué dans la figure 4.14 ce qui permet de créer un état initial S^I_{val} tel que :

$$P(S^I_{val}) \hat{=} \texttt{Book=\{bo\}} \land \texttt{Member=\{me\}} \land \texttt{Lend} = \emptyset \land \texttt{Reserve} = \emptyset$$

```
INITIALISATION
    Book := {bo}
  || Member := {me}
  || Lend := {}
  || Reserve := {}
```

FIGURE 4.14 – Initialisation de la machine B

Dans le but d'illustrer l'approche basée sur la résolution de contraintes, nous rajoutons les valeurs `bo` et `me` aux valeurs possibles pour les instances appartenant aux ensembles respectifs `BOOK` et `MEMBER` (figure 4.15).

```
SETS
  BOOK = {bo};
  MEMBER = {me}
```

FIGURE 4.15 – Caractérisation des ensembles

Les chemins que nous cherchons doivent atteindre l'état symbolique S^F_{P} tel que :

$$\mathsf{P_F} \hat{=} \texttt{(me} \mapsto \texttt{bo)} \in \texttt{Lend}$$

En appliquant nos algorithmes de recherche, nous sommes capable d'extraire les chemins donnés dans le tableau 4.2.

Approche par la preuve	Approche par la résolution de contraintes
$Q_{symb1} =$ <Member_AddLend>	$Q_1 =$ <Member_AddLend(me,bo)>
$Q_{symb2} =$ <Member_AddReserve; Member_TakeReservedBook>	$Q_2 =$ <Member_AddReserve(me,bo); Member_TakeReservedBook(me,bo)>

TABLEAU 4.2 – Chemins extraits par nos algorithmes

4.4.1 Preuve du premier chemin(Q_{symb1} / Q_1)

Le premier chemin est composé d'une seule opération, ce qui fait de l'état S^F_{val} un successeur de l'état S^I_{val} par application de l'opération Member_AddLend dont la précondition et l'action satisfont l'obligation de preuve suivante :

$$
\begin{array}{ll}
\exists Instance, Lend\_bookValues, Member, Book, Lend, Reserve \cdot (& /*\exists \mathscr{P}_{i,var}*/ \\
\quad me \in Member \wedge bo \in Book \wedge (me \mapsto bo) \notin Lend \wedge & /*\mathsf{P_I}*/ \\
\quad Instance \in Member \wedge & /*Pre(Member\_AddLend)*/ \\
\quad card(Lend[\{Instance\}]) < 3 \wedge & \\
\quad Lend\_bookValues \in Book \wedge & \\
\quad (Instance \mapsto Lend\_bookValues) \notin Lend \wedge & \\
\quad Lend\_bookValues \notin ran(Lend) \wedge & \\
\quad Lend\_bookValues \notin dom(Reserve) \wedge & \\
\quad \neg((me \mapsto bo) \notin Lend \cup \{(Instance \mapsto Lend\_bookValues)\})) & /*\neg[Action(Member\_AddLend)]\neg \mathsf{P_F}*/
\end{array}
$$

L'utilisation de l'outil de preuve automatique AtelierB nous a aidé à décharger automatiquement cette obligation de preuve, et ce en la déclarant comme une assertion de la spécification B du système.

Ensuite, nous utilisons le model-checker ProB pour trouver les paramètres de l'opération permettant de réaliser cette atteignabilité. Ainsi, nous ne gardons que l'opération Member_AddLend et nous désactivons toutes les autres opérations. Le model-checker a été capable d'extraire l'exécution Member_AddLend(me,bo) après avoir essayé une seule transition alors qu'il a consommé 16 transitions pour arriver au même résultat sans notre guidance.

Nous avons aussi réussi à extraire cette exécution en appliquant notre deuxième approche basée sur la résolution de contraintes. En effet, le solveur de contraintes de ProB a pu résoudre automatiquement ce problème à contraintes en attestant que le couple de paramètres (me, bo) est l'unique solution :

$\{Instance, Lend\_bookValues|$ /*$\mathscr{P}_i$*/
$MEMBER = \{me\} \wedge BOOK = \{bo\} \wedge$ /* S */
$\exists Member, Book, Lend, Reserve \cdot ($ /* $\exists$var. */
$Member \subseteq MEMBER \wedge Book \subseteq BOOK \wedge$ /* inv */
$Lend \in Member \leftrightarrow Book \wedge$
$\forall c2 \cdot (c2 \in dom(Lend) \Rightarrow card(Lend[\{c2\}]) \leq 3) \wedge$
$Reserve \in Book \nrightarrow Member \wedge$
$me \in Member \wedge bo \in Book \wedge (me \mapsto bo) \notin Lend \wedge$ /*P_I*/
$Instance \in Member \wedge$ /*Pre(Member_AddLend)*/
$card(Lend[\{Instance\}]) < 3 \wedge$
$Lend\_bookValues \in Book \wedge$
$(Instance \mapsto Lend\_bookValues) \notin Lend \wedge$
$Lend\_bookValues \notin ran(Lend) \wedge$
$Lend\_bookValues \notin dom(Reserve) \wedge$
$\neg((me \mapsto bo) \notin Lend \cup \{(Instance \mapsto Lend\_bookValues)\}))\}$ /*$\neg$[Action(Member_AddLend)]$\neg\mathsf{P}_\mathsf{F}$*/

4.4.2 Preuve du deuxième chemin (Q_{symb2} / Q_2)

Ce chemin composé de deux opérations a été obtenu par application de notre algorithme avec retour en arrière qui a permis la construction du système de transition de la figure 4.16.

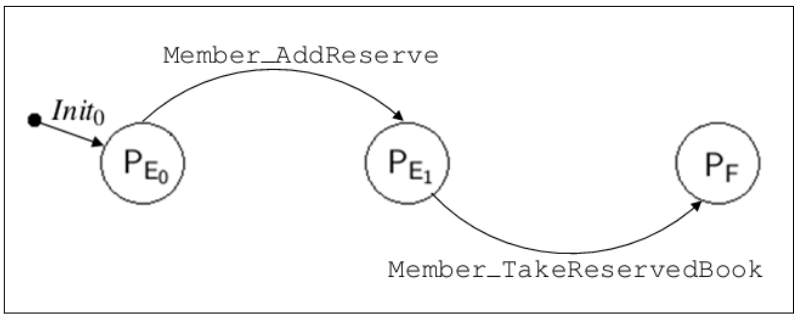


FIGURE 4.16 – Système de transition construit par l'algorithme de recherche

— *Première itération :*

Lors de la première itération, l'algorithme a extrait la dernière opération dans la séquence `Member_TakeReservedBook` dont la précondition et l'action satisfont l'obligation de preuve suivante obtenue par application de la formule (4) :

$$
\begin{array}{ll}
\exists Instance, ReservedBook, Member, Book, Lend, Reserve \cdot (& /*\exists \mathscr{P}_i, var*/ \\
Instance \in Member \wedge ReservedBook \in Book \wedge & /*Pre(Member\_TakeReservedBook)*/ \\
ReservedBook \notin ran(Lend) \wedge & \\
Reserve(ReservedBook) = Instance \wedge & \\
card(Lend[\{Instance\}]) < 3 \wedge & \\
(me \mapsto bo) \notin Lend \wedge & /*\neg \mathsf{P_F}*/ \\
\neg((me \mapsto bo) \notin Lend \cup \{(Instance \mapsto ReservedBook)\}))\} & /*\neg[Action(Member\_TakeReservedBook)]\neg \mathsf{P_F}*/
\end{array}
$$

Cette obligation de preuve a été démontrée automatiquement en utilisant l'AtelierB.

L'utilisation du solveur de contraintes ProB a également permis d'extraire l'exécution : `Member_TakeReservedBook(me,bo)` après avoir attesté que `(me,bo)` est une solution pour le problème à contraintes suivant :

$$
\begin{array}{ll}
\{Instance, ReservedBook | & /*\mathscr{P}_i*/ \\
MEMBER = \{me\} \wedge BOOK = \{bo\} \wedge & /*\ S\ */ \\
\exists Member, Book, Lend, Reserve.(& /*\ \exists var.\ */ \\
Member \subseteq MEMBER \wedge Book \subseteq BOOK \wedge & /*\ inv\ */ \\
Lend \in Member \leftrightarrow Book \wedge & \\
\forall c2 \cdot (c2 \in dom(Lend) \Rightarrow card(Lend[\{c2\}]) \leq 3) \wedge & \\
Reserve \in Book \nrightarrow Member \wedge & \\
Instance \in Member \wedge ReservedBook \in Book \wedge & /*Pre(Member\_TakeReservedBook)*/ \\
ReservedBook \notin ran(Lend) \wedge & \\
Reserve(ReservedBook) = Instance \wedge & \\
card(Lend[\{Instance\}]) < 3 \wedge & \\
(me \mapsto bo) \notin Lend \wedge & /*\neg \mathsf{P_F}*/ \\
\neg((me \mapsto bo) \notin Lend \cup \{(Instance \mapsto ReservedBook)\})\} & /*\neg[Action(Member\_TakeReservedBook)]\neg \mathsf{P_F}*/
\end{array}
$$

— *Deuxième itération :*

Lors de la deuxième itération, nous nous basons sur la précondition de l'opération : `Member_TakeReservedBook` extraite à la première itération pour calculer les nouveaux états symboliques $S_{\mathsf{P}}^{E_1}$ et $S_{\mathsf{P}}^{E_0}$. Nous commençons par vérifier si l'état initial est inclus dans la nouvelle cible $S_{\mathsf{P}}^{E_1}$ avant de chercher les opérations qui satisfont l'atteignabilité de $S_{\mathsf{P}}^{E_1}$ à partir de $S_{\mathsf{P}}^{E_0}$:

$$
\begin{array}{rcl}
\mathsf{P_{E_1}} & \hat{=} & Pre(Member\_TakeReservedBook) \wedge \neg \mathsf{P_F} \\
& \hat{=} & \exists Instance, ReservedBook \cdot (\\
& & Instance \in Member \wedge ReservedBook \in Book \wedge \\
& & ReservedBook \notin ran(Lend) \wedge \\
& & Reserve(ReservedBook) = Instance \wedge \\
& & card(Lend[\{Instance\}]) < 3 \wedge \\
& & (Instance \mapsto ReservedBook) \notin Lend)
\end{array}
$$

Remarque 4.2 [États moins abstraits dans la deuxième approche]

La deuxième approche basée sur la résolution de contraintes a l'avantage par rapport à l'approche basée sur la preuve d'avoir des états symboliques moins abstraits du à la connaissance des paramètres des opérations extraites, ce qui permet d'enlever l'opérateur existentiel et de remplacer les paramètres par leurs valeurs dans les préconditions. Ceci permet de réduire considérablement la complexité de la preuve d'atteignabilité, surtout dans les itérations bien avancées lorsque les prédicats exprimant les états symboliques deviennent de plus en plus complexes. Par exemple, pour le calcul du prédicat $\mathsf{P}_{\mathsf{E}_1}$ *dans la deuxième approche, nous remplaçons* `∃ Instance, ReservedBook` *par les paramètres* `me` *et* `bo`*, ce qui donnera le prédicat moins abstrait suivant :*

$$\begin{array}{rcl} \mathsf{P}_{\mathsf{E}_1} & \widehat{=} & Pre(Member\_TakeReservedBook(me,bo)) \wedge \neg \mathsf{P}_{\mathsf{F}} \\ & \widehat{=} & me \in Member \wedge bo \in Book \wedge \\ & & bo \notin ran(Lend) \wedge \\ & & Reserve(bo) = me \wedge \\ & & card(Lend[\{me\}]) < 3 \wedge \\ & & (me \mapsto bo) \notin Lend) \end{array}$$

Nous pouvons remarquer que $\mathsf{P}_{\mathsf{I}} \not\Rightarrow \mathsf{P}_{\mathsf{E}_1}$, car, en effet, l'état initial satisfait la contrainte `Reserve = ∅`, tandis que l'état symbolique $S_{\mathsf{P}}^{E_1}$ doit satisfaire la contrainte suivante : `Reserve(ReservedBook) = Instance`.

Comme l'état initial n'est toujours pas atteint, nous procédons à la deuxième itération dans laquelle nous exhibons les opérations vérifiant la propriété de succession entre $S_{\mathsf{P}}^{E_0}$ et $S_{\mathsf{P}}^{E_1}$. Ainsi, nous cherchons parmi toutes les opérations de la spécification les opérations qui satisfont l'obligation de preuve donnée par la formule (5). D'où, l'extraction de l'opération `Member_AddReserve` qui satisfait l'obligation de preuve suivante :

$$\begin{array}{l} \exists Instance, Reserve\_bookValues, Member, Book, Lend, Reserve \cdot \big(\text{/*}\exists \mathscr{D}_i, var\text{*/} \\ Pre(Member\_AddReserve) \wedge \\ \neg Pre(Member\_TakeReservedBook) \wedge \neg \mathsf{P}_{\mathsf{F}} \wedge \\ \neg [Action(Member\_AddReserve)] \neg (Pre(Member\_TakeReservedBook) \wedge \neg \mathsf{P}_{\mathsf{F}}) \big) \end{array}$$

L'exécution `Member_TakeReservedBook(me,bo)` est également exhibée par application de l'approche basée sur la résolution de contraintes, telle que le couple `(me,bo)` satisfait les contraintes suivantes :

$\{Instance, Reserve\_bookValues|$
$MEMBER = \{me\} \wedge BOOK = \{bo\} \wedge$
$\exists Member, Book, Lend, reserve \cdot (Inv \wedge$
$Pre(Member\_AddReserve) \wedge \neg Pre(Member\_TakeReservedBook(me, bo)) \wedge \neg \mathsf{P_F} \wedge$
$\neg[Action(Member\_AddReserve)]\ \neg(Pre(Member\_TakeReservedBook(me, bo)) \wedge \neg \mathsf{P_F}))\}$

Nous vérifions ensuite que $\mathsf{P_I} \Rightarrow \mathsf{P_{E_0}}$ sachant que :

$$\begin{aligned} \mathsf{P_{E_0}} \mathrel{\hat{=}} \quad & Pre(Member\_AddReserve) \wedge \\ & \neg Pre(Member\_TakeReservedBook) \wedge \\ & \neg \mathsf{P_F} \end{aligned}$$

Par conséquent, nous déduisons que l'état initial est atteint et qu'un chemin composé de deux opérations établit la propriété d'atteignabilité.

Ainsi, l'approche basée sur la résolution de contraintes nous a permis d'extraire une séquence concrète Q_2. Quant à l'approche basée sur la preuve, elle nous a permis l'extraction de la séquence symbolique Q_{symb2} qui peut être concrétisée en ayant recours à un model-checker comme ProB.

En utilisant, la technique de guidance du model-checker par un contrôleur CSP, comme nous l'avons expliqué dans la section 4.2.4, ProB a pu extraire la séquence Q_2 après avoir essayé uniquement 2 transitions. Il a essayé 3 transitions lorsque nous avons utilisé la technique de guidance par réduction de l'espace d'états en se limitant aux opérations qui apparaissent dans la séquence symbolique. Mais, il avait besoin de 37 transitions pour extraire la même séquence en utilisant purement le model-checker sans aucune guidance.

4.5 Comparaison avec des travaux similaires

Dans le souci de trouver des alternatives aux techniques de model-checking classiques et de limiter le problème de l'explosion combinatoire, certains travaux basés sur la preuve ont été proposés pour la vérification de la propriété d'atteignabilité en B. La plupart d'entre eux ont discuté le problème dans des systèmes fondés sur l'utilisation de la méthode B événementiel, et très peu ont étudié le problème en méthode B classique.

La preuve d'atteignabilité en Event-B a été initiée par Abrial et Mussat qui ont proposé d'étendre le langage de spécification avec la modalité *leadsto* [ABRIAL et MUSSAT 1998]. Cette modalité est notée $E \rightsquigarrow F$, elle exprime l'atteignabilité d'un état s' représenté par le prédicat F à partir d'un état s satisfaisant le prédicat E. Les obligations de preuve de la propriété d'atteignabilité sont inspirées de l'obligation de preuve de la boucle. En effet, ils expriment cette modalité sous forme de terminaison d'une boucle dont le corps est composé d'un ensemble d'événements et dont la garde est exprimée par la négation de F. Abrial et Mussat spécifient également une expression entière décroissante strictement positive appelée variant caractérisant le plus long chemin entre s et s' et ils vérifient que chaque transition décroît ce variant en préservant E ou en établissant F. Cependant, il est difficile d'identifier le variant ce qui rend difficile la mise en œuvre de cette approche d'autant plus qu'elle n'est pas outillée.

Toutefois, ces travaux ont inspiré Groslambert qui a proposé au sein de son outil JAG tool une approche pour la vérification de la propriété d'atteignabilité ainsi que toutes les propriétés de la Logique Temporelle Linéaire (LTL) [GROSLAMBERT 2007]. Son approche est basée sur les automates de Büchi qui sont par la suite traduits en des modèles Event-B à partir desquels les obligations de preuve sont générées. L'approche ainsi que l'outil sont fort utiles pour la vérification de l'ensemble des modalités LTL, mais la propriété d'atteignabilité définie dans ce cadre est différente de la propriété qui nous intéresse. En effet, nous cherchons à vérifier s'il existe un chemin menant à un état cible contrairement à l'atteignabilité en LTL qui consiste à prouver que tous les chemins atteignent la cible. Il est à noter que la propriété d'atteignabilité la plus adéquate à notre cas de figure est celle définie par la modalité CTL comme établi dans [FRAPPIER et al. 2011, MAMMAR et al. 2011].

Une des approches les plus proches de ce que nous proposons pour la vérification de la propriété d'atteignabilité en B est celle proposée dans [MAMMAR et al. 2011]. Cette approche est basée sur la génération automatique des obligations de preuve à partir d'un ensemble de chemins préalablement défini. Les chemins considérés sont exprimés par une séquence d'exécution d'opérations sous la forme $(cond \rightsquigarrow (act_1; act_2; \ldots; act_n))$ telles que les act_i désignent les actions (ou les opérations) du chemin et $cond$ spécifie la condition (ou la garde) qui doit être satisfaite pour l'exécution du chemin. Pour vérifier que le chemin atteint un état s' qui satisfait F à partir d'un état s satisfaisant E, les auteurs génèrent une obligation de preuve sous la forme générale suivante :

$$E \wedge cond \Rightarrow [(act_1; \ldots; act_n)]F$$

Toutefois, les obligations de preuve sont très nombreuses et difficiles à décharger automatiquement. Pour alléger la preuve les auteurs proposent de procéder par étapes en vérifiant dans un premier temps que l'action act_1 est exécutable [2] dans l'état s où E et $cond$ sont vrais. Ils vérifient, ensuite, à tout point du chemin i après l'exécution de chaque action act_i que les valeurs des variables vérifient la précondition de l'action suivante act_{i+1}. Enfin, ils vérifient que les valeurs des variables obtenues après l'exécution de la dernière action act_n satisfont F.

Cette technique a permis certes de réduire la preuve mais elle reste relativement compliquée à décharger automatiquement avec des outils de preuve comme AtelierB.

Dans [FRAPPIER et al. 2011], les auteurs proposent une autre approche basée sur le raffinement des substitutions. Ils partent d'une séquence d'exécution similaire à celle présentée dans la première approche afin de spécifier le chemin d'atteignabilité. Ce chemin est vu comme étant un programme qui raffine ou qui réalise la propriété d'atteignabilité. Ils utilisent les règles de raffinement définies par Morgan [MORGAN 1990] pour générer un programme P relatif au chemin défini puis ils établissent l'obligation de preuve suivante :

$$E \Rightarrow [P]F$$

2. une action ou une opération est exécutable si sa précondition est satisfaite

Le processus de génération du programme *P* étant un peu compliqué et nécessite des prérequis pour les règles de raffinement de Morgan, nous n'abordons pas ce détail et nous laissons le lecteur intéressé se référer à [FRAPPIER et al. 2011].

Cette approche a l'avantage de produire moins d'obligations de preuve comparée à l'approche précédente. Néanmoins, la génération du programme de raffinement n'est pas triviale.

De plus, les deux approches supposent donné un ensemble de chemins d'exécution afin de vérifier qu'au moins l'un de ces chemins satisfait la propriété d'atteignabilité. Cependant, contrairement aux approches que nous proposons, ces approches ne cherchent pas à extraire et à identifier les séquences qui établissent l'atteignabilité. En effet, nous avons montré à travers l'étude de cas du SI de la bibliothèque, qui a été aussi traitée par les auteurs dans [FRAPPIER et al. 2011, MAMMAR et al. 2011], que notre approche est capable de prouver l'atteignabilité sur les chemins qu'ils ont défini et qu'elle réussit à extraire.

4.6 Bilan et discussion

Nous avons proposé dans ce chapitre une technique de recherche appliquant un algorithme par retour en arrière pour la vérification de la propriété d'atteignabilité. Cette technique peut être appliquée aussi bien dans le cadre de la vérification de la sécurité des SI afin d'extraire des séquences d'opérations révélant des comportements malicieux, que dans le cadre plus général de la vérification de l'atteignabilité d'un état quelconque.

Deux approches différentes ont été proposées dans ce chapitre permettant la mise en oeuvre de cet algorithme. La première est basée sur la preuve et elle cherche à extraire des séquences d'opérations symboliques. Quant à la deuxième, elle est basée sur la résolution de contraintes et elle permet d'extraire des séquences d'exécutions concrètes.

Les deux approches présentent chacune des avantages et des limitations. En effet, bien que l'approche basée sur la preuve permette d'avoir une meilleure garantie quant à l'atteignabilité d'un état par une opération donnée sans restrictions sur l'espace d'état, elle souffre des limitations classiques de la complexité de la preuve. Les expérimentations menées dans ce cadre ont montré que la plupart des preuves ne peuvent être déchargées automatiquement, et nécessitent l'interaction de l'analyste. Ces preuves deviennent encore plus compliquées quand nous avançons dans les itérations de l'algorithme, et ce à cause de l'opérateur existentiel appliqué aux paramètres des opérations. L'application de cette approche sur plusieurs cas d'étude, que nous présentons dans le chapitre 6, a montré qu'à partir de la troisième itération aucune preuve n'est déchargée automatiquement. Ceci reste fastidieux à mener dans le cas d'un nombre d'opérations accru ou de spécifications de grande taille. En outre, cela ne correspond pas à notre objectif de disposer d'une approche automatisée d'extraction de scénarios.

Ceci étant, nous suggérons de compléter cette approche par les techniques de model-checking

après avoir réduit l'espace de recherche en éliminant les opérations prouvées comme non pertinentes pour l'atteignabilité de la propriété en question. Dans le cas où les obligations de preuves issues de l'algorithme ont été menées avec succès, le model-checker permettra juste de produire les scénarios concrets à partir des scénarios symboliques. Mais, dans le cas où des obligations de preuve de certaines opérations n'ont pas été prouvées automatiquement, nous proposons d'opter pour la preuve de non atteignabilité (formules (B) et (D) de l'approche GénéSyst présenté à la section 2.3.2) qui est plus simple à décharger et qui pourra nous servir à écarter les opérations sans intérêt. Le model-checker est par la suite utilisé pour extraire les scénarios satisfaisant la propriété d'atteignabilité en explorant uniquement les transitions non éliminées de la spécification. Cependant, le problème de la complexité de la preuve dans les itérations avancées de l'algorithme persiste et parfois nous n'arrivons pas à identifier un grand nombre d'opérations non pertinentes sans avoir recours à la preuve manuelle ou interactive.

La résolution de contraintes est une alternative intéressante car d'une part, cette technique est moins coûteuse, et d'autre part, quand l'espace d'états est borné, elle permet une prise de décision rapide. Cependant, cette technique nécessite de contraindre les domaines des variables d'état.

Compte tenu des avantages et des inconvénients de chacune de ces deux approches, il serait intéressant de les combiner pour pallier les limitations de l'une par les avantages de l'autre. Nous pouvons, par exemple avoir recours à la preuve uniquement lorsque le solveur de contraintes échoue dans la résolution ce qui permet de confirmer ou d'infirmer l'atteignabilité de l'état par l'opération en question. Ceci permet également de simplifier la preuve en ayant des valuations pour les variables relatives aux paramètres des opérations extraites lors des itérations précédentes par résolution de contraintes. En cas d'une confirmation d'atteignabilité par la preuve, nous pouvons procéder à la restriction du domaine des variables d'états jusqu'à ce qu'une solution soit trouvée par le solveur de contraintes. En terme de preuve, il est également possible de se baser sur la preuve de non atteignabilité pour écarter les opérations sans intérêt et par la suite réduire les appels au solveur de contraintes.

Toutefois, malgré ces limitations nos approches sont jusqu'à présent les seules qui n'utilisent pas les techniques de model-checking pures et qui permettent à la fois la vérification de la propriété d'atteignabilité et l'extraction de séquences satisfaisant cette propriété à partir d'une spécification B classique. En plus, l'application de nos approches sur différentes études de cas dans le contexte de la validation de la sécurité a donné lieu à des résultats très encourageants et a montré leur efficacité pour l'extraction de séquences atteignant des états indésirables et révélant des attaques sur le système. L'applicabilité de nos approches dans ce contexte sera détaillée dans le chapitre suivant en expliquant comment caractériser les états indésirables et en précisant les types d'attaque que nous sommes capable d'extraire.

Chapitre 5

Extraction des scénarios d'attaque interne

« My message for companies that think they haven't been attacked is : "You're not looking hard enough". »

James Snook

Sommaire

5.1 Motivation et notions préliminaires 94
5.2 Définition formelle d'un scénario d'attaque 98
5.3 Caractérisation des états indésirables 100
5.4 Technique d'extraction de scénarios d'attaque 102
5.4.1 Extraction de comportements malicieux 103
5.4.2 Génération de l'état initial 105
5.4.3 Identification des utilisateurs malicieux 109
5.4.4 Automatisation de l'identification des utilisateurs malicieux 111
5.4.5 Extraction automatique des attaques masquées 113
5.5 Bilan et discussion 115

Les règles de contrôle d'accès exprimées en fonction de données issues du modèle fonctionnel peuvent être à l'origine de failles de sécurité qui sont souvent exploitées par les utilisateurs du système pour commettre des attaques internes. Ces attaques ont pour objectif de faire évoluer l'état fonctionnel du système pour qu'un utilisateur malveillant initialement non autorisé soit finalement autorisé à effectuer une action qui compromet l'un des objectifs de sécurité. Ainsi, il est important d'analyser minutieusement l'impact de l'évolution de l'état fonctionnel sur les règles de contrôle d'accès pour vérifier que le système n'autorise pas ce genre de scénarios d'attaque. Une telle analyse est favorisée par la plate-forme B4MSecure étant donné que les liens entre le modèle fonctionnel et le modèle de sécurité sont très bien explicités dans les spécifications formelles qu'elle produit. C'est pour cette raison que nous nous basons sur ces spécifications pour la détection de ces attaques.

Dans notre investigation, la détection de ces attaques est assimilée à l'extraction de séquences d'opérations fonctionnelles permettant d'atteindre un état, dit indésirable, où l'accès à certaines ressources critiques du système soit débloqué. Par conséquent, l'extraction de scénarios d'attaque revient à exhiber les séquences d'opérations qui vérifient la propriété d'atteignabilité de l'état indésirable. Nous proposons, ainsi, d'appliquer nos approches pour la vérification de la propriété d'atteignabilité en B en vue de détecter les scénarios d'attaques auxquelles un SI est exposé.

Dans ce chapitre, nous montrons comment caractériser les états indésirables, comme nous expliquons comment mettre en pratique nos approches pour l'extraction de chemins vérifiant l'atteignabilité pour identifier des scénarios d'attaque interne.

5.1 Motivation et notions préliminaires

Nous nous intéressons à l'extraction de scénarios d'attaque traduisant les quatre types d'attaque interne que nous avons définis dans le chapitre 1.

Nous considérons, ainsi, comme attaque interne toute action ou séquence d'actions qui permet d'atteindre un état indésirable S_{P}^{F} en contournant l'une des règles de contrôle d'accès du SI. Suivant le type de ces règles de sécurité nous identifions le type de l'attaque commise. En effet, **une règle de sécurité**, que nous notons $\mathfrak{R}_s$, est un filtre de sécurité appliqué à une opération fonctionnelle. Dans le cadre du modèle RBAC, cette règle peut être lié au rôle, dans ce cas il s'agit d'une attaque basique. Elle peut également être une contrainte d'autorisation, dans ce cas nous avons affaire à une attaque interne à contrainte. Quant aux deux autres types d'attaque interne (i.e. l'attaque planifiée et l'attaque masquée), ils peuvent être considérés comme des combinaisons d'attaques basiques ou à contrainte.

Pour chaque attaque, nous définissons un ensemble d'**attaquants potentiels**, que nous notons **A**. Il s'agit de l'ensemble des utilisateurs qui ne remplissent pas les conditions de la règle de sécurité ciblée $\mathfrak{R}_s$. Par exemple :

— Dans le cas d'une attaque basique, l'ensemble des attaquants est l'ensemble des utilisateurs qui n'ont pas le rôle requis pour accéder à la ressource objet de l'attaque.

— Dans le cas d'une attaque à contrainte, l'ensemble des attaquants est tout utilisateur qui ne satisfait pas les conditions de la contrainte d'autorisation.

Outre la notion d'attaquant, nous définissons la notion de **victime**. En effet, certaines attaques visent à compromettre l'un des intérêts d'un utilisateur dans le système. Dans ce cas de figure, ce dernier est appelé victime de l'attaque.

Pour mieux illustrer les quatre types d'attaque interne ainsi que ces notions, nous considérons le SI de la bibliothèque présenté au chapitre 3. Nous considérons, ainsi, le modèle fonctionnel donné à la figure 3.2, le modèle de sécurité de la figure 3.8 et les utilisateurs de la figure 3.7. Nous prenons l'exemple de l'état fonctionnel schématisé par le diagramme d'objets de la figure 5.1, où `bo1` et `bo2` sont des livres de la bibliothèque, `John` et `Bob` sont des membres et `Bob` a réservé le livre `bo1`.

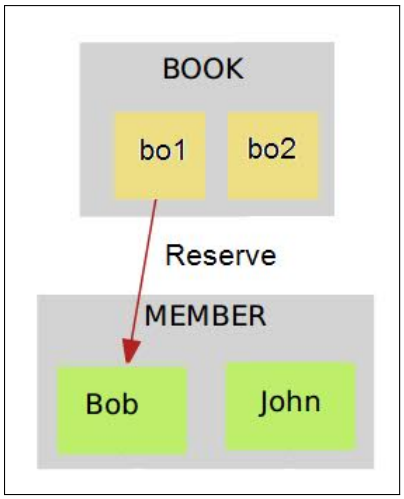


FIGURE 5.1 – Etat initial de la machine fonctionnelle

Les scénarios d'attaque suivants extraits à partir du modèle de sécurité constituent des exemples pour les quatre types d'attaque :

- **Attaque basique :** Selon les règles de contrôle d'accès, seuls les utilisateurs ayant le rôle `MemberUser` peuvent emprunter des livres. Mais, pour contourner cette règle, un utilisateur du rôle `Librarian`, comme `Alice`, peut exécuter la séquence d'opérations suivante lui permettant d'emprunter le livre `bo2` :

```
Connect(Alice, {Librarian});
secure_Member_New(Alice);
Connect(Alice, {MemberUser});
secure_Member_AddLend(Alice,bo2);
```

FIGURE 5.2 – Scénario d'une attaque basique

En effet, en tant que bibliothécaire, `Alice` a le droit d'inscrire de nouveaux membres. Elle profite, donc, de ce droit pour créer un nouveau membre pour son propre compte. Ceci lui octroie le rôle `MemberUser` en plus du rôle `Librarian` qu'elle possède. Par conséquent, elle se connecte en activant ce nouveau rôle qui lui donne le droit d'emprunter des livres.

Ce scénario d'attaque a permis d'atteindre l'état indésirable $S_{\mathsf{P}}^{F_{basique}}$ où un utilisateur considéré comme un **attaquant** ne possédant pas le rôle `MemberUser` exécute l'opération `secure_Member_AddLend`. Ainsi, le prédicat $\mathsf{P}_{F_{basique}}$ peut être défini comme suit :

$$\mathsf{P}_{F_{basique}} \hat{=} attaquant \in dom(Lend) \text{ où } attaquant \in \mathbf{A}$$

tel que l'ensemble des attaquants $\mathbf{A}$ peut être défini comme suit :

$$\mathbf{A} = \{a | a \in USERS \wedge MemberUser \notin roleOf(a)\}$$

La règle de sécurité $\mathfrak{R}_s$ visée par cette attaque est une règle de rôle. Elle peut être exprimée par le prédicat suivant :

$$\mathfrak{R}_s \hat{=} MemberUser \in roleOf(attaquant)$$

Il est à noter que, comme cette attaque ne vise pas à compromettre l'un des intérêts d'un autre utilisateur, nous pouvons considérer que l'ensemble des victimes $\mathbf{V} = \emptyset$. Il est également possible de considérer comme victime l'ensemble des utilisateurs ayant le rôle `MemberUser`. En effet, cette attaque peut priver ces utilisateurs de l'emprunt du livre `bo2`.

- **Attaque à contrainte :** En tant que `MemberUser`, `John` a le droit d'accéder à toutes les opérations de la classe `Member`. Toutefois, la contrainte d'autorisation ne lui permet d'exécuter ces opérations que pour les instances de son utilisateur. Ainsi, il ne peut ni annuler la réservation de `Bob` ni emprunter le livre `bo1` à sa place. Mais, en collaborant avec `Alice` il peut contourner la contrainte d'autorisation et emprunter le livre en mettant en œuvre le scénario suivant :

```
Connect(Alice, {Librarian});
secure_Member_Free(Bob);
Connect(John, {MemberUser});
secure_Member_AddLend(John,bo1);
```

FIGURE 5.3 – Scénario d'une attaque à contrainte

En effet, comme `Alice` a le rôle `Librarian` elle peut supprimer l'inscription de `Bob`. Cette suppression permet d'effacer l'instance du membre `Bob` ainsi que tous les liens en relation avec cette instance y compris la réservation qu'il y avait pour le livre `bo1`. Toutes les contraintes empêchant l'emprunt du livre `bo1` étant supprimées, `John` peut alors se connecter en tant que `MemberUser` et exercer son droit d'emprunt.

L'état indésirable $S_{\mathsf{P}}^{F_{contrainte}}$ peut être caractérisé par le fait qu'un autre utilisateur différent de `Bob`, considéré comme la **victime**, puisse emprunter le livre `bo1`. Ainsi nous pouvons l'exprimer par le prédicat suivant :

$$\mathsf{P}_{F_{contrainte}} \hat{=} (attaquant \mapsto bo1) \in Lend \text{ où } attaquant \in \mathbf{A}$$

Tel que $\mathbf{A} = \{a | a \in USERS \wedge a \notin \mathbf{V}\}$, où $\mathbf{V}$ dénote l'ensemble des victimes. Dans notre cas : $\mathbf{V} = \{v | v \in Member \wedge (bo1 \mapsto v) \in Reserve\}$.
La règle de sécurité $\mathfrak{R}_s$, dans ce cas, est la contrainte d'autorisation qui exige que l'emprunt soit fait par le même utilisateur qui a fait la réservation.

- **Attaque planifiée :** Pour emprunter le livre `bo1`, un utilisateur ne possédant pas le rôle `MemberUser`, comme `Alice`, doit planifier une attaque en deux étapes. Dans la première étape elle doit annuler la réservation de `Bob` et dans la deuxième étape elle emprunte le livre. Par conséquent, la combinaison des deux attaques précédentes peut constituer une attaque planifiée :

```
Connect(Alice, {Librarian});
secure_Member_Free(Bob);
secure_Member_New(Alice);
Connect(Alice, {MemberUser});
secure_Member_AddLend(Alice,bo1);
```

FIGURE 5.4 – Scénario d'une attaque planifiée

Cette attaque planifiée cible donc un premier état indésirable $S_{\mathsf{P}}^{F_1}$ qui peut être caractérisé par le prédicat suivant : $\mathsf{P}_{F_1} \hat{=} (victime \mapsto bo1) \notin Reserve$.

Une fois cet objectif atteint, l'attaquant cherche à atteindre un deuxième état indésirable $S_{\mathsf{P}}^{F_2}$ en partant de l'état déjà atteint. Ce deuxième état peut être caractérisé par le prédicat suivant : $\mathsf{P}_{F_2} \hat{=} (attaquant \mapsto bo1) \in Lend$.

L'ensemble des attaquants est alors l'ensemble des utilisateurs ne possédant pas le rôle `MemberUser`, et la victime est l'utilisateur qui a réservé le livre objet de l'attaque.

- **Attaque masquée :** Nous considérons le scénario de l'attaque basique décrit par la figure 5.2. Pour camoufler ses actions malveillantes, `Alice` peut réaliser les opérations inverses pour remettre le système à son état d'origine. Ainsi, moyennant son nouveau rôle de `MemberUser`, elle restitue le livre. Puis, elle supprime le nouvel utilisateur qui a été créé soit en utilisant le rôle `Librarian` ou le rôle `MemberUser` :

```
Connect(Alice, {Librarian});
secure_Member_New(Alice);
Connect(Alice, {MemberUser});
secure_Member_AddLend(Alice,bo2);
secure_Member_RemoveLend(Alice,bo2);
Connect(Alice, {Librarian});
secure_Member_Free(Alice);
```

FIGURE 5.5 – Scénario d'une attaque masquée

Cette attaque peut être extraite en deux parties. En effet, il est possible de considérer dans un premier temps l'atteignabilité de l'état indésirable $S_{\mathsf{P}}^{F_{basique}}$ à partir de l'état initial. Une

fois cette atteignabilité est vérifiée, nous cherchons, dans un second temps, une séquence d'opérations qui permet d'atteindre l'état initial S_{P}^{I} à partir de cet état.

Les ensembles des attaquants et des victimes sont les mêmes que ceux qui sont définis pour l'attaque basique.

Cette analyse préliminaire montre que bien que les règles de contrôle d'accès d'un SI semblent être bien définies, elles peuvent être facilement détournées en profitant de l'évolution dynamique de l'état fonctionnel. Ainsi, nous proposons une approche pour extraire de telles attaques qu'elles soient perpétrées par un seul attaquant interne ou par un ensemble d'utilisateurs en collusion.

Nous notons que, dans notre étude, nous ne considérons pas les attaques qui utilisent des opérations d'administration des règles de contrôle d'accès telles que la création d'un nouvel utilisateur, l'attribution de nouveaux rôles aux utilisateurs, etc. La raison pour laquelle nous ne considérons pas ces opérations est que sinon, l'attaquant devient très puissant, car il peut créer un utilisateur avec tous les rôles. Cependant, ces opérations peuvent être prises en compte seulement si elles sont incluses dans les opérations fonctionnelles, tel est le cas de l'opération `Member_New` qui est une opération fonctionnelle permettant de créer de nouveaux utilisateurs avec le rôle `MemberUser`.

5.2 Définition formelle d'un scénario d'attaque

Toute séquence d'opérations **fonctionnelles** qui permet d'atteindre un état indésirable est considérée comme un **comportement malicieux** qui pourrait donner lieu à un **scénario d'attaque**. En effet, ce comportement malicieux (ou suspect) ne peut être considéré comme une vraie attaque que s'il est autorisé par les règles de contrôle d'accès. Autrement dit, il existe un ensemble d'utilisateurs, parmi les attaquants potentiels, pouvant réaliser cette séquence d'opérations fonctionnelles.

Exemple. Soit la séquence d'opérations fonctionnelles suivante :

```
Member_RemoveReserve(Bob, bo1);
Member_AddLend(John,bo1);
```

Cette séquence permet d'annuler la réservation de `Bob` au livre `bo1`, puis de l'emprunter pour `John`. Elle peut, donc, être considérée comme un comportement malicieux car elle permet d'atteindre l'état $S_{\mathsf{P}}^{F_{contrainte}}$ à partir de l'état initial (donné par la figure 5.1). Cependant, elle ne donne pas lieu à un scénario d'attaque, car il n'y a aucun ensemble d'utilisateurs inclus dans l'ensemble des attaquants qui peut réaliser ce comportement malicieux. En effet, seule la victime `Bob` a le droit d'exécuter la première opération dans le scénario. Par conséquent, ce comportement malicieux ne peut être réalisé dans le modèle de sécurité qu'à travers le scénario suivant qui est plutôt un scénario d'utilisation normal :

```
Connect(Bob, {MemberUser});
secure_Member_RemoveReserve(Bob, bo1);
Connect(John, {MemberUser});
secure_Member_AddLend(John,bo1);
```

Formellement, nous définissons ces notions comme suit :

Définition 9 (**Comportement malicieux simple**).
Un comportement malicieux (ou suspect) est une séquence d'opérations fonctionnelles Q_F qui vérifie l'atteignabilité $S_{\mathsf{P}}^I \overset{Q_F}{\rightsquigarrow} S_{\mathsf{P}}^F$ telle que :

1) S_{P}^I est un état initial où une règle de sécurité $\mathfrak{R}_s$ est fausse : $\mathsf{P}_I \hat{=} \neg\mathfrak{R}_s$
2) S_{P}^F est un état indésirable où $\mathfrak{R}_s$ est vraie : $\mathsf{P}_F \hat{=} \mathfrak{R}_s$

Définition 10 (**Comportement malicieux composé**).
Un comportement malicieux composé est le séquencement d'un ensemble de comportements malicieux simples :

$$Q_F = Q_{F_1}; Q_{F_2}; \ldots; Q_{F_n}$$

Son objectif est d'atteindre l'ensemble des états indésirables $S_{\mathsf{P}}^{F_1}$, $S_{\mathsf{P}}^{F_2}$, …, $S_{\mathsf{P}}^{F_n}$ en procédant par étapes à partir de l'état initial S_{P}^I tel que :

1) $S_{\mathsf{P}}^I \overset{Q_{F_1}}{\rightsquigarrow} S_{\mathsf{P}}^{F_1}$,
2)$\forall i \in \{2..n\}$, Q_{F_i} vérifie $S_{\mathsf{P}}^{F_{i-1}} \overset{Q_{F_i}}{\rightsquigarrow} S_{\mathsf{P}}^{F_i}$

Définition 11 (**Prémisses de sécurité**).
Une politique de contrôle d'accès à base de modèle RBAC filtre l'accès aux opérations fonctionnelles selon le triplet noté (u, R, c_{auth}), appelé prémisses de sécurité, où u est un utilisateur, R un ensemble de rôles assigné à u et c_{auth} est une contrainte d'autorisation appliquée à l'opération en question.

Définition 12 (**Attaquant - Victime**).
Un attaquant a est tout utilisateur du système qui n'a accès à aucune opération op soumise à la règle de sécurité $\mathfrak{R}_s$ à l'état initial S_{P}^I tel que $\forall R, c_{auth} \cdot ((a, R, c_{auth}) \models false)$. L'ensemble des attaquants est noté $\mathbf{A}$.

Une victime v est un utilisateur qui peut être la cible de l'attaque. L'ensemble des victimes est noté $\mathbf{V}$, tel que $\mathbf{V} \cap \mathbf{A} = \emptyset$.

Définition 13 (**Scénario d'attaque**).
Un scénario d'attaque est la réalisation d'un comportement malicieux Q_F par un utilisateur $u \in \mathbf{A}$ ou par une collusion d'un ensemble d'utilisateurs $\mathscr{U} \subseteq \mathbf{A}$.

Soit $Q_F = < op_1((\mathscr{P}_1)), op_2((\mathscr{P}_2)), \ldots, op_n((\mathscr{P}_n)) >$, alors :

$$\forall op_i \cdot (op_i \in Q_F \Rightarrow \exists a, R, c_{auth} \cdot ((a, R, c_{auth}) \models true))$$

Cela signifie que les rôles R, activés par l'attaquant a tel que $a = u$ ou $a \in \mathscr{U}$, autorisent l'exécution de op_i, et s'il existe une contrainte d'autorisation c_{auth} associée à l'opération op_i, alors elle doit être satisfaite.

5.3 Caractérisation des états indésirables

Un état indésirable est un état qui débloque l'accès à une opération fonctionnelle critique $op_{critique}$ soumise à des règles de sécurité $\mathfrak{R}_s$. Dans notre investigation, nous nous intéressons à deux types d'opérations critiques :

(i) Les opérations identifiées par les exigences de sécurité. Par exemple, dans le SI de la bibliothèque, l'intégrité de l'emprunt et de la réservation d'un livre peuvent être identifiées comme étant critiques. Par conséquent, les opérations qui modifient les liens `Lend` et `Reserve` peuvent être considérées comme critiques.

Ainsi, étant donnée une opération critique $op_{critique}$, la règle de sécurité $\mathfrak{R}_s$ qu'un attaquant cherche à compromettre est toute contrainte appliquée à cette opération : $\mathfrak{R}_s \hat{=} Pre(op_{critique})$.

Cette spécification de l'état indésirable nous permet, donc, d'identifier des **attaques basiques**.

(ii) Les opérations auxquelles des contraintes d'autorisation sont associées. En effet, comme un scénario d'attaque cherche à accéder à des opérations en faisant évoluer l'état fonctionnel du système, et comme les contraintes d'autorisation expriment des règles de contrôle d'accès qui dépendent de l'état fonctionnel, nous supposons que ces opérations sont les plus sensibles à ce genre d'attaque.

Dans ce cas, l'état indésirable S^F_{P} est l'état où la contrainte d'autorisation exprimée par le prédicat c_{auth} est satisfaite, ainsi : $\mathsf{P}_F \hat{=} \mathfrak{R}_s$ tel que $\mathfrak{R}_s \hat{=} c_{auth}$.

Ce type d'état indésirable, nous permet d'extraire **les attaques à contrainte**.

Nous notons que l'atteignabilité d'un état où la contrainte d'autorisation c_{auth} est satisfaite peut rendre accessibles toutes les opérations concernées par cette contrainte. Par exemple, à partir de l'état qui satisfait la contrainte d'autorisation associée à la permission `MemberPerm` dans l'exemple du SI de la bibliothèque (figure 3.8), nous pouvons exécuter toutes les opérations de la classe `Member`.

Dans le cas où nous cherchons à extraire les scénarios d'attaque ciblant une opération critique spécifique $op_{critique}$ à laquelle une contrainte d'autorisation c_{auth} est associée, l'état indésirable est alors exprimé comme suit : $\mathsf{P}_F \hat{=} c_{auth} \wedge Pre(op_{critique})$. Par conséquent, la séquence d'opérations qui satisfait cette atteignabilité exprime un comportement malicieux qui révèle une attaque de type basique et à contrainte en même temps.

Remarque 5.1 [Caractérisation d'un état indésirable ciblant l'exécution d'une opération critique]

Par définition, le comportement malicieux est la séquence d'opérations Q_F, qui mène à l'état satisfaisant la règle $\Re_s$ où l'opération $op_{critique}$ devient déclenchable. Étant donnée que l'attaque consiste à exécuter cette opération, nous pouvons considérer que le comportement malicieux est toute la séquence :

$$< Q_F, op_{critique} >$$

Cette séquence permet d'atteindre l'état après l'exécution de l'opération cible.

Dans ce cas, le prédicat caractérisant l'état indésirable ne sera pas extrait automatiquement, mais doit être défini manuellement par l'analyste en se référant aux propriétés qu'il souhaite atteindre en enclenchant l'opération critique.

Les états indésirables d'un comportement malicieux composé peuvent à leur tour être identifiés de deux manières différentes :

(i) En nous référant aux exigences de sécurité qui peuvent identifier un ensemble d'opérations fonctionnelles critiques $op_1, op_2, \ldots, op_n$ qui ne doivent pas être exécutées ensemble. Ceci nous permet d'extraire des comportements malicieux révélant des **attaques planifiées** qui peuvent être assimilées à la composition de n attaques basiques. En effet, la séquence d'opérations Q_F exprimant l'attaque planifiée peut être exprimée par le séquencement de n séquences d'opérations :

$$Q_F = Q_{F_1}; op_1; Q_{F_2}; op_2; \ldots; op_{n-1}; Q_{F_n}; op_n$$

Tel que chaque séquence Q_{F_i} représente une attaque basique qui cherche à atteindre l'opération op_i à partir de l'état atteint par l'opération op_{i-1}. Ainsi :

$$\begin{gathered} S^I_{\mathsf{P}} \overset{Q_{F_1}}{\rightsquigarrow} S^{F_1}_{\mathsf{P}}, \\ S^{F_1}_{\mathsf{P}'} \overset{Q_{F_2}}{\rightsquigarrow} S^{F_2}_{\mathsf{P}}, \\ \ldots, \\ S^{F_{n-1}}_{\mathsf{P}'} \overset{Q_{F_n}}{\rightsquigarrow} S^{F_n}_{\mathsf{P}} \end{gathered}$$

Tout en sachant que : $\mathsf{P}_{F_i} \hat{=} Pre(op_i)$ et P'_{F_i} est le prédicat obtenu après l'exécution de l'opération op_i.

(ii) Systématiquement après l'extraction de tout comportement malicieux Q_{F_1} nous pouvons vérifier si ce comportement malicieux peut être camouflé en remettant le système dans son état d'origine, et ce dans le but d'identifier des **attaques masquées**. En effet, le comportement malicieux Q_F révélant une attaque masquée peut être composé comme suit : $Q_F = Q_{F_1}; Q_{F_2}$, tel que :

$$S_{\mathsf{P}}^{I} \overset{Q_{F_1}}{\rightsquigarrow} S_{\mathsf{P}}^{F_1}$$
$$S_{\mathsf{P}}^{F_1} \overset{Q_{F_2}}{\rightsquigarrow} S_{\mathsf{P}}^{I}$$

5.4 Technique d'extraction de scénarios d'attaque

Selon les définitions 10 et 13, l'extraction de scénarios d'attaque consiste à chercher des séquences d'exécution permettant d'atteindre les états indésirables. Ainsi nous montrons dans cette section comment étendre la technique de recherche par retour en arrière que nous avons proposée dans le chapitre précédent pour l'identification de séquences d'opérations révélant des comportements malicieux.

En nous basant sur les modèles formels générés par la plate-forme B4MSecure, nous pouvons mener notre recherche de deux manières différentes :

(i) A partir du modèle formel de sécurité, ou

(ii) En deux temps, en exhibant tout d'abord les comportements malicieux à partir du modèle fonctionnel, puis en vérifiant la possibilité d'exécution de ces comportements dans le modèle de sécurité.

Toutefois, nous favorisons la deuxième alternative aussi bien pour des raisons conceptuelles que pour des raisons techniques :

— D'un point de vue conceptuel, commencer par extraire les scénarios potentiellement dangereux à partir du modèle fonctionnel permet d'orienter la conception du modèle de sécurité afin d'empêcher l'exécution de ces scénarios. De plus, quand le modèle de sécurité évolue, il suffit de rejouer ces scénarios dans ce dernier pour vérifier que ces comportements malicieux restent non autorisés.

— D'un point de vue technique, les contraintes de sécurité dans le modèle de sécurité sont souvent compliquées, ainsi les obligations de preuve de la propriété d'atteignabilité résultant du modèle fonctionnel sont plus faciles à traiter et à décharger. Le modèle de sécurité inclut également beaucoup de variables d'état nécessaires pour le calcul des permissions ce qui rend l'espace d'état beaucoup plus grand comparé à celui du modèle fonctionnel.

  En outre, le modèle de sécurité comprend l'opération `Connect`. Cette dernière permet de connecter un utilisateur en activant un sous ensemble de ses rôles pour la réalisation des opérations fonctionnelles. Toutefois, cette opération ne manipule pas des données fonctionnelles et ne fait pas évoluer l'état fonctionnel du système. Ainsi, il n'est pas possible d'extraire les opérations de connexion en utilisant l'algorithme de retour en arrière que nous avons proposé dans le chapitre précédent. Par conséquent, pour extraire un scénario d'attaque à partir du modèle de sécurité il faut fixer un utilisateur malicieux ainsi que ses rôles et effectuer la recherche pour exhiber les scénarios qui peuvent être réalisés par cet utilisateur. Il est donc impossible d'extraire des scénarios faisant intervenir plusieurs utilisateurs en collusion en optant pour la recherche de scénarios à partir du modèle de sécurité.

Dans la suite, nous montrons comment il est possible d'extraire des scénarios impliquant plus d'un utilisateur en optant pour l'approche en deux phases : une phase de recherche de comportement malicieux dans le modèle fonctionnel et une phase de synthèse de scénarios d'attaque à partir du modèle de sécurité en nous basant sur ces comportements malicieux.

5.4.1 Extraction de comportements malicieux

Lors de cette première phase, nous étudions l'atteignabilité d'un état fonctionnel indésirable en appliquant l'approche décrite dans le chapitre précédent pour l'extraction de traces d'exécution satisfaisant une propriété d'atteignabilité à partir d'une spécification B. Lors de cette étape, nous nous intéressons à l'identification des opérations fonctionnelles révélant des comportements malicieux sans pour autant se préoccuper des utilisateurs ainsi que des permissions liées aux rôles. Toutefois, il est important de prendre en compte les contraintes d'autorisation, car, d'une part, ces dernières dépendent de l'état fonctionnel, et d'autre part, les états indésirables peuvent être exprimés en fonction de ces contraintes.

Ainsi, nous appliquons nos algorithmes de recherche sur le modèle fonctionnel enrichi par les contraintes d'autorisation. Par exemple, nous donnons dans la figure 5.6 la spécification de l'opération `Member_AddLend` après l'avoir enrichie par la contrainte d'autorisation qui lui est appliquée.

```
Member_AddLend(Instance, Lend_bookValues)=
   PRE  /*Contraintes fonctionnelles*/
        Instance ∈ Member ∧
        Lend_bookValues ∈ Book ∧
        card(Lend[{Instance}]) < 3 ∧
        (Instance ↦ Lend_bookValues) ∉ Lend ∧
        Lend_bookValues ∉ ran(Lend) ∧
        Lend_bookValues ∉ dom(Reserve)
   THEN
        SELECT  /*Contraintes d'autorisation*/
                currentUser = Instance
        THEN  /*Action de l'opération*/
                Lend := Lend ∪ {(Instance ↦ Lend_bookValues)}
        END
   END;
```

FIGURE 5.6 – Opération fonctionnelle enrichie par une contrainte d'autorisation

Ceci requiert l'addition de la nouvelle variable `currentUser` dans le modèle fonctionnel ainsi que de l'invariant de typage suivant relatif à cette variable :

```
currentUser ∈ ATTAQUANTS
```

Où ATTAQUANTS ⊆ USERS désigne l'ensemble des attaquants potentiels. Cet ensemble est fixé dés le début du processus d'extraction du comportement malicieux en fonction de l'état indésirable cible.

En guise d'exemple, nous appliquons notre algorithme de recherche par retour en arrière pour extraire des comportements malicieux attaquant l'intégrité de la réservation d'un livre. Nous supposons, ainsi, qu'un membre **victime** a réservé un livre bo quand ce dernier était emprunté par un autre utilisateur. Nous cherchons les comportements malicieux qui permettent à un utilisateur considéré comme un **attaquant** de l'emprunter à la place de la victime une fois restitué.

Comme l'attaque consiste à emprunter le livre bo par l'attaquant, alors si nous souhaitons considérer que l'état indésirable est l'état où l'attaque a réellement eu lieu, nous pouvons définir le prédicat suivant :

$$P_F \widehat{=} (attaquant \mapsto bo) \in Lend$$

Dans ce cas, l'opération fonctionnelle Member_AddLend(attaquant, bo) peut être considérée comme l'opération critique cible. En effet, elle permet d'introduire le couple (attaquant ↦ bo) dans la relation *Lend*.

Si nous supposons que nous avons une seule victime Bob, alors nous pouvons définir l'ensemble des attaquants et des victimes comme suit :

```
DEFINITIONS
  VICTIMES == {Bob};
  ATTAQUANTS == USERS - VICTIMES
```

FIGURE 5.7 – Définition des victimes et des attaquants

Dans le contexte de l'attaque que nous cherchons à extraire, toute victime v doit satisfaire les propriétés suivantes notées $R_{victime}$:

$$R_{victime} \widehat{=} v \in VICTIMES \wedge v \in Member \wedge (bo \mapsto v) \in Reserve \wedge bo \notin ran(Lend)$$

Quant à l'attaquant a, il doit satisfaire les propriétés suivantes notées $R_{attaquant}$:

$$R_{attaquant} \widehat{=} a \in ATTAQUANTS$$

Nous considérons l'état initial donné par la figure 5.8 ainsi que la spécification des ensembles BOOK et MEMBER donnée par la figure 5.9.

```
INITIALISATION
    Book := {bo}
  || Member := {Bob, John}
  || Lend := {}
  || Reserve := {(bo ↦ Bob)}
```

FIGURE 5.8 – Initialisation de la machine B

```
SETS
   BOOK = {bo}
DEFINITIONS
   MEMBER == USERS;
```

FIGURE 5.9 – Spécification des ensembles

Grâce à notre algorithme basé sur la résolution de contraintes nous avons pu extraire 14 comportements malicieux dont certains sont redondants. C'est à dire qu'ils sont composés des mêmes opérations mais qui apparaissent dans un ordre différent. Nous reportons, ainsi, les 8 comportements différents dans la deuxième colonne du tableau 5.1. Par exemple, la séquence suivante est l'une des séquences redondantes qui reproduit le même comportement de la séquence Q_{F_2} :

$$\begin{aligned} Q'_{F_2} = < \quad & Book\_Free(bo), \\ & Member\_New(Alice), \\ & Book\_New(bo), \\ & Member\_AddLend(Alice, bo) \quad > \end{aligned}$$

L'approche basée sur la preuve nous a également permis d'extraire les séquences symboliques suivantes :

$$\begin{aligned} Q_{symb_1} &= < Book\_Free,\ Book\_New,\ Member\_AddLend > \\ Q_{symb_2} &= < Member\_New, Book\_Free,\ Book\_New,\ Member\_AddLend > \\ Q_{symb_3} &= < Book\_Free,\ Book\_New,\ Member\_AddReserve, \\ &\quad Member\_TakeReservedBook > \\ Q_{symb_4} &= < Member\_New,\ Book\_Free,\ Book\_New,\ Member\_AddReserve, \\ &\quad Member\_TakeReservedBook > \\ Q_{symb_5} &= < Member\_Free,\ Member\_AddLend > \\ Q_{symb_6} &= < Member\_New,\ Member\_Free,\ Member\_AddLend > \\ Q_{symb_7} &= < Member\_Free,\ Member\_AddReserve,\ Member\_TakeReservedBook > \\ Q_{symb_8} &= < Member\_New,\ Member\_Free,\ Member\_AddReserve, \\ &\quad Member\_TakeReservedBook > \end{aligned}$$

5.4.2 Génération de l'état initial

Les séquences que nous cherchons à extraire atteignent l'état indésirable à partir d'un état initial concret. Dans l'exemple que nous avons traité dans la section précédente l'état initial (figure 5.8) a été défini manuellement. Mais, dans le cas général, l'état initial peut être généré automatiquement. En effet, il doit remplir les trois conditions suivantes :

(i) S'il existe une victime de l'attaque que nous cherchons à extraire, l'état initial doit satisfaire les propriétés de cette victime $R_{victime}$.

	Comportements malicieux	(Utilisateur, Role)
Q_{F_1}	$Book\_Free(bo)$, $Book\_New(bo)$, $Member\_AddLend(John, bo)$	(Alice,Librarian) (Alice,Librarian) (John,MemberUser)
Q_{F_2}	$Member\_New(Alice)$, $Book\_Free(bo)$, $Book\_New(bo)$, $Member\_AddLend(Alice, bo)$	(Alice,MemberUser) (Alice,Librarian) (Alice,Librarian) (Alice,MemberUser)
Q_{F_3}	$Book\_Free(bo)$, $Book\_New(bo)$, $Member\_AddReserve(John, bo)$, $Member\_TakeReservedBook(John, bo)$	(Alice,Librarian) (Alice,Librarian) (John,MemberUser) (John,MemberUser)
Q_{F_4}	$Member\_New(Alice)$, $Book\_Free(bo)$, $Book\_New(bo)$, $Member\_AddReserve(Alice, bo)$, $Member\_TakeReservedBook(Alice, bo)$	(Alice,MemberUser) (Alice,Librarian) (Alice,Librarian) (Alice,MemberUser) (Alice,MemberUser)
Q_{F_5}	$Member\_Free(Bob)$, $Member\_AddLend(John, bo)$	(Alice,Librarian) (John,MemberUser)
Q_{F_6}	$Member\_New(Alice)$, $Member\_Free(Bob)$, $Member\_AddLend(Alice, bo)$	(Alice,MemberUser) (Alice,Librarian) (Alice,MemberUser)
Q_{F_7}	$Member\_Free(Bob)$, $Member\_AddReserve(John, bo)$, $Member\_TakeReservedBook(John, bo)$	(Alice,Librarian) (John,MemberUser) (John,MemberUser)
Q_{F_8}	$Member\_New(Alice)$, $Member\_Free(Bob)$, $Member\_AddReserve(Alice, bo)$, $Member\_TakeReservedBook(Alice, bo)$	(Alice,MemberUser) (Alice,Librarian) (Alice,MemberUser) (Alice,MemberUser)

TABLEAU 5.1 – Des scénarios d'attaque sur le SI de la bibliothèque

(ii) L'état initial doit satisfaire les propriétés de l'attaquant $R_{attaquant}$. Comme nous cherchons des attaques qui peuvent être commises par une collaboration de plusieurs attaquants, nous proposons de partir d'un état initial qui satisfait cette propriété pour tout élément qui remplit les conditions de l'attaquant.

(iii) Selon la définition 10, l'état initial ne doit pas satisfaire le prédicat de l'état indésirable P_F.

Afin de générer cet état initial, nous proposons d'utiliser la technique de résolution de contraintes en vue de trouver les valuations pour les variables d'état qui satisfont ces trois conditions. Par conséquent, l'ensemble de solutions du problème à contraintes suivant constitue l'ensemble des états initiaux possibles que nous pouvons considérer pour la recherche des comportements malicieux :

$$\{var | Inv \wedge \quad \exists victime \cdot (R_{victime} \wedge \\ \forall attaquant \cdot (R_{attaquant} \Rightarrow \neg \mathsf{P}_F))\}$$

Pour notre exemple, l'état initial peut être l'une des solutions au problème suivant :

$$\begin{array}{ll} \{Book, Member, Lend, Reserve | (& /*Inv*/ \\ & Book \subseteq BOOK \wedge \\ & Member \subseteq MEMBER \wedge \\ & Lend \in Member \leftrightarrow Book \wedge \\ & Reserve \in Book \nrightarrow Member \\ & \forall c_2 \cdot (c_2 \in dom(Lend) \Rightarrow card(Lend[\{c_2\}]) \leq 3) \wedge \\ & /*\text{Il existe au moins une victime}*/ \\ & \exists victime \cdot (victime \in VICTIMES \wedge \\ & /*R_{victime}*/ \\ & victime \in Member \wedge (bo \mapsto victime) \in Reserve \wedge \\ & bo \notin ran(Lend)) \\ & /*\forall attaquant \cdot (R_{attaquant}*/ \\ & \forall attaquant \cdot (attaquant \in ATTAQUANTS \Rightarrow \\ & /*\neg \mathsf{P}_F*/ \\ & (attaquant \mapsto bo) \notin Lend))\} \end{array}$$

Nous utilisons le solveur de contraintes de ProB pour résoudre ce problème. Ainsi, nous obtenons les quatre solutions suivantes :

$$\begin{array}{ll} S_{val}^{I_1} \widehat{=} & Book = \{bo\} \wedge Lend = \{\} \wedge Reserve = \{(bo \mapsto Bob)\} \wedge Member = \{Bob\} \\ S_{val}^{I_2} \widehat{=} & Book = \{bo\} \wedge Lend = \{\} \wedge Reserve = \{(bo \mapsto Bob)\} \wedge Member = \{Bob, John\} \\ S_{val}^{I_3} \widehat{=} & Book = \{bo\} \wedge Lend = \{\} \wedge Reserve = \{(bo \mapsto Bob)\} \wedge Member = \{Bob, Alice\} \\ S_{val}^{I_4} \widehat{=} & Book = \{bo\} \wedge Lend = \{\} \wedge Reserve = \{(bo \mapsto Bob)\} \wedge Member = \{Bob, Alice, John\} \end{array}$$

Pour notre exemple, en considérant l'un ou l'autre de ces états initiaux, nous obtenons presque les mêmes séquences fonctionnelles que celles décrites dans le tableau 5.1 à quelques différences près qui consistent à :

— rajouter l'opération `Member_New(John)` aux scénarios qui impliquent le membre `John` (i.e. Q_{F_1}, Q_{F_3}, Q_{F_5}, Q_{F_7}) dans le cas où nous considérons un état initial où $John \notin Member$, et

— rajouter l'opération `Member_New(Alice)` aux scénarios qui impliquent `Alice` (i.e. Q_{F_2}, Q_{F_4}, Q_{F_6}, Q_{F_8}) dans le cas où nous considérons un état initial où $Alice \notin Member$.

Nous notons que cette technique de génération de l'état initial peut engendrer une explosion combinatoire de nombre des états initiaux générées dans le cas où le nombre de variables d'états est important et que les domaines de ces variables ne sont pas bornés. Pour palier ce problème, il est possible de définir des heuristiques de choix :

— Choisir l'état initial **minimal** tel que défini dans 14.

> **Définition 14 (Etat minimal).**
> Un état **minimal** $S_{val_{min}}$ parmi un ensemble d'états $\mathbb{S}$ est l'état qui possède le nombre le plus **petit** d'éléments dans les ensembles qui le constituent ($\sum_{v \in var} card(v)$). Au contraire, l'état **maximal** est l'état qui maximise ce nombre.

Ce choix permet de réduire le problème de l'explosion combinatoire. Mais, malheureusement, il peut alourdir la recherche car il inclut le minimum de pré-requis pour le comportement malicieux, ce qui a pour effet d'extraire de plus long chemins. Par exemple, dans notre cas d'étude, l'état initial minimal est l'état $S_{val}^{I_1}$ telle que $\sum card(v) = 3$. Le choix de cet état est non avantageux car il conduit à l'extraction de séquences plus longues que celles données dans le tableau 5.1. En effet, dans chaque comportement impliquant le membre `John`, l'opération `Member_New(John)` doit être ajoutée.

Le choix de l'état initial minimal peut également engendrer l'échec d'extraction des comportements malicieux dans le cas où certaines opérations fonctionnelles ne sont pas disponibles. Par exemple, dans le cas où le modèle fonctionnel ne fournit pas une opération pour la création d'un nouveau membre, l'état initial minimal $S_{val}^{I_1}$ ne permet pas d'extraire les comportements malicieux du tableau 5.1. Toutefois, ce problème ne se pose pas si on considère les spécifications produites par B4MSecure qui génère automatiquement toutes les opérations de base.

— Maximiser les ensembles de variables qui représentent les attaquants et minimiser les ensembles qui décrivent les propriétés des victimes. En effet, comme nous cherchons des attaques qui peuvent impliquer plusieurs utilisateurs, il peut être intéressant de choisir un état initial qui considère le maximum d'attaquants potentiels. En même temps, une seule victime dans le système peut suffire pour déceler si l'attaque sur cette victime est possible ou pas. Dans notre exemple, cela revient à maximiser l'ensemble `Member` comme l'attaquant est un utilisateur qui doit être membre de la bibliothèque, et à minimiser l'ensemble `Reserve`.

5.4.3 Identification des utilisateurs malicieux

Un comportement malicieux extrait à partir du modèle fonctionnel ne peut être considéré comme un scénario d'attaque que s'il est permis par le modèle de sécurité. Dans ce cas, un renforcement de sécurité est requis pour filtrer ces comportements malicieux.

Par conséquent, étant données les séquences d'opérations représentant les comportements malicieux extraits dans la première phase de notre approche, nous vérifions pour chaque opération op_i du comportement malicieux Q_F s'il existe des prémisses de sécurité (u_i, R_i, c_i) qui autorisent l'exécution de cette opération dans l'état atteint par l'opération op_{i-1}. Nous cherchons plus précisément dans la deuxième phase de notre approche des couples (utilisateur, rôle) qui sont autorisés à exécuter chacune des opérations étant donné que les contraintes d'autorisation ont été prises en compte lors de l'extraction des comportements malicieux.

Il est possible d'identifier les couples (utilisateur, rôle) en nous référant au modèle graphique SecureUML. Ainsi, en nous basant sur les modèles donnés aux figures 3.7 et 3.8, nous avons pu déterminer les utilisateurs malicieux susceptibles de réaliser chacune des opérations ainsi que le rôle qu'ils doivent utiliser pour y arriver. Les différents couples (utilisateur, rôle) identifiés sont synthétisés dans la troisième colonne du tableau 5.1.

Par exemple, nous savons qu'il n'y a que le bibliothécaire qui a le droit de manipuler les entités de la classe `Book`. Par conséquent, seule `Alice` peut exécuter les opérations `Book_Free` et `Book_New` en se connectant au système avec son rôle `Librarian`. Nous savons également qu'aucun membre autre que `John` ne peut exécuter l'opération `Member_AddLend(John, bo)` étant donné qu'une contrainte d'autorisation est associée à cette opération pour empêcher un membre d'emprunter un livre en faveur d'un autre membre. Le comportement malicieux Q_{F_1} révèle donc un scénario d'attaque qui peut être réalisé par l'ensemble d'utilisateurs $\mathscr{U} = \{Alice, John\}$. Il s'agit donc d'un scénario d'attaque qui met en évidence une collusion d'utilisateurs jouant différents rôles dans le SI. Tandis que, dans le scénario Q_{F_2}, `Alice` intervient seule en basculant entre les deux rôles `MemberUser` et `Librarian`. Ces deux scénarios d'attaque, tout comme les scénarios Q_{F_3} et Q_{F_4}, soulignent une erreur de spécification dans le modèle fonctionnel. En effet, le système ne devrait pas autoriser la suppression d'instances de livre liés par des relations d'emprunt ou de réservation. Nous corrigeons, donc, l'opération `Book_Free` comme indiqué dans la figure 5.10 et nous vérifions que ces comportements ne sont plus autorisés. Cette vérification peut être faite soit en appliquant notre algorithme de retour en arrière de nouveau sur la spécification corrigée soit tout simplement par animation.

```
Book_Free(Instance)=
    PRE  Instance ∈ BOOK ∧ Instance ∈ Book
         Instance ∉ ran(Lend) ∧
         Instance ∉ dom(Reserve)

    THEN  Book := Book − {Instance}
         Lend := Lend ⩤ {Instance}
         Reserve := Reserve ⩥ {Instance}
    END;
```

FIGURE 5.10 – Correction de l'opération `Book_Free`

Les quatre dernières séquences ont en commun l'opération `Member_Free(Bob)` qui peut être considérée comme la clé de ces scénarios d'attaque. En effet, cette opération, qui doit être exécutée soit par un bibliothécaire soit par le membre `Bob`, permet de supprimer l'instance du membre `Bob` ainsi que tous les liens que cette instance possède, ce qui permet de libérer le livre `bo` de sa réservation. Mais, comme `Bob` est la victime, nous supposons qu'il ne peut pas participer à un scénario où il s'attaque à lui même. Nous concluons, ainsi, que seule `Alice` peut exécuter cette opération en utilisant son rôle `Librarian`. Pour les deux scénarios Q_{F_5} et Q_{F_7} où `Alice` exécute l'opération `Member_Free(Bob)` pour autoriser `John` à emprunter le livre `bo`, nous pouvons considérer qu'il s'agit d'un comportement normal qui doit être autorisé par le système pour permettre au bibliothécaire d'intervenir et de débloquer un livre au cas où le membre qui l'a réservé ne s'est pas manifesté pendant une longue période.

Par ailleurs, les séquences extraites doivent être analysées par un expert du domaine pour décider si un renforcement de sécurité est nécessaire afin d'éviter qu'un scénario ne soit exécuté abusivement ou s'il s'agit d'un comportement normal qui doit être autorisé par le système.

Quant aux deux séquences Q_{F_6} et Q_{F_8}, elles révèlent bel et bien des scénarios d'attaque. En effet, il est clair que `Alice` abuse de son rôle de bibliothécaire pour servir ses propres intérêts. Elle commence par la suppression du membre `Bob`, puis elle s'inscrit en tant que `MemberUser` afin qu'elle puisse emprunter le livre réservé par `Bob` en utilisant le nouveau rôle qu'elle s'est octroyée. Une des corrections qui peuvent être proposées pour interdire un tel comportement est de rajouter une contrainte de séparation de devoirs statique SSD entre les deux rôles `Librarian` et `MemberUser`. Nous interdisons ainsi qu'un bibliothécaire puisse être un membre de la bibliothèque.

5.4.4 Automatisation de l'identification des utilisateurs malicieux

La recherche des couples (utilisateur, rôle) revient à trouver les paramètres des opérations de connexion qui garantissent la déclenchabilité de la version sécurisée des opérations fonctionnelles. Par conséquent, nous considérons qu'un scénario d'attaque est une séquence d'opérations fonctionnelles révélant un comportement malicieux et pouvant être exécutée dans le modèle de sécurité après avoir introduit des opérations de connexion (`Connect`) aux bons endroits.

Définition 15 (**Séquence fonctionnelle dans le modèle de sécurité**)**.**
Une séquence d'opérations Q_F extraite à partir du modèle fonctionnel peut avoir son équivalent dans le modèle de sécurité notée $secure\_Q_F$ telle que chaque opération op_i dans Q_F est remplacée par sa version sécurisée $secure\_op_i$ dans $secure\_Q_F$.

Définition 16 (**Scénario d'attaque** - complément de la définition 13)**.**
Un scénario d'attaque Q_S est la séquence $secure\_Q_F$ à laquelle certaines opérations de connexion sont ajoutées telles que Q_F représente un comportement malicieux et telles que :

$$\begin{array}{rll} Q_S = & Connect(u_1,\{r_1\}); & secure\_Q_{F_1}\,; \\ & Connect(u_2,\{r_2\}); & secure\_Q_{F_2}\,; \\ & \ldots & \ldots \\ & Connect(u_m,\{r_m\}); & secure\_Q_{F_m}\,; \end{array}$$

Où :

— $\forall i \cdot (i \in \{1..m-1\} \Rightarrow u_i \neq u_{i+1} \vee r_i \neq r_{i+1})$ et $\mathscr{U} = set(u_1,u_2,\ldots,u_m)$ [a] est l'ensemble des utilisateurs malicieux réalisant le scénario d'attaque,

— $secure\_Q_F = secure\_Q_{F_1}; secure\_Q_{F_2}; \ldots; secure\_Q_{F_m}$, et

— $(u_j, r_j, c_i) \models true$ pour toute opération $secure\_op_i$ dans $secure\_Q_{F_j}$

a. $set(l)$ dénote l'ensemble des éléments non redondants dans une liste d'éléments l

En nous basant sur la définition 16, nous donnons la séquence Q_{S_1} qui représente le scénario d'attaque correspondant au comportement malicieux Q_{F_1} :

$$\begin{array}{rll} Q_{S_1} = < & \textbf{Connect(Alice, \{Librarian\})}, \\ & secure\_Book\_Free(bo), \\ & secure\_Book\_New(bo), \\ & \textbf{Connect(John, \{MemberUser\})}, \\ & secure\_Member\_AddLend(John, bo) > \end{array}$$

Afin d'automatiser le processus d'extraction de scénarios d'attaque à partir des comportements malicieux, nous proposons d'utiliser une technique de guidance de model-checker. Notre approche

consiste à guider le model-checker ProB par une algèbre de processus CSP [HOARE 1978] dans laquelle nous spécifions l'ordre des appels des opérations fonctionnelles. Ainsi, contrôlé par le processus CSP, ProB explore la spécification B du modèle de sécurité et cherche les possibilités d'exécution de la séquence donnée en introduisant les opérations de connexions nécessaires aux bons endroits.

Par conséquent, nous traduisons chacune des séquences d'opérations fonctionnelles obtenues lors de la première phase en un processus de contrôle CSP, tel qu'une séquence d'opérations $secure\_Q_F$ de la forme :

$$\begin{aligned} secure\_Q_F = \quad & secure\_op_1(\mathscr{P}_1) \ ; \\ & secure\_op_2(\mathscr{P}_2) \ ; \\ & \ldots \ ; \\ & secure\_op_n(\mathscr{P}_n) \end{aligned}$$

sera traduite en un processus de contrôle appelé `Attack` de la forme suivante :

```
Attack     =  Connexion ; secure_op1?P1 →
              Connexion ; secure_op2?P2 →
              ...
              Connexion ; secure_opn?Pn →
              goal → STOP
Connexion  =  Connect → SKIP  □
              SKIP
```

FIGURE 5.11 – Traduction d'une séquence fonctionnelle en CSP

En suivant le processus donné par la figure 5.11, le model-checker ProB cherche toutes les traces d'exécution permettant d'atteindre la cible spécifiée par **goal**. Ainsi, il invoque les opérations $secure\_op_i(\mathscr{P}_i)$ dans l'ordre indiqué en vérifiant si les prémisses de sécurité sont satisfaites et si elles permettent la déclenchabilité de l'opération pour l'utilisateur en cours de connexion. Dans le cas où il n'est pas possible d'exécuter l'opération, le model-checker cherche à connecter un autre utilisateur ou à utiliser un autre rôle afin de satisfaire les préconditions de l'opération. C'est pour cette raison que le processus `Connexion` est optionnel avant chaque opération : il peut soit appeler une nouvelle opération `Connect` ou exécuter le processus sans effet `SKIP` pour enchaîner avec l'opération fonctionnelle suivante dans la séquence. Comme nous ne spécifions pas les valuations des paramètres de l'opération `Connect`, le model-checker parcourt toutes les valuations possibles pour extraire les utilisateurs et les rôles pertinents qui permettent la réalisation de l'opération en question.

Par exemple, le contrôleur CSP donné par la figure 5.12, issu de la traduction de la séquence Q_{F_1}, a permis l'extraction automatique du scénario d'attaque Q_{S_1}.

```
Attack1    =  Connexion ; secure_Book_Free?bo →
              Connexion ; secure_Book_New?bo →
              Connexion ; secure_Member_AddLend?John?bo →
              goal → STOP
Connexion  =  Connect → SKIP  □
              SKIP
```

FIGURE 5.12 – Q_{F_1} traduit en un contrôleur CSP

Le processus de contrôle général exprimant l'ensemble des séquences fonctionnelles exhibées peut être exprimé comme dans la figure 5.13 où chaque processus $\texttt{Attack}_i$ traduit le comportement de la séquence fonctionnelle Q_{F_i} comme indiqué dans la figure 5.11. Guidé par ce processus le model-checker produit automatiquement tous les scénarios d'attaque permettant d'atteindre l'état indésirable spécifié.

$$\texttt{Main} = \texttt{Attack}_1 \ \square\ \texttt{Attack}_2 \ \square\ \ldots\ \square\ \texttt{Attack}_n$$

FIGURE 5.13 – Contrôleur CSP général

En appliquant cette technique sur notre cas d'étude, nous avons pu extraire automatiquement tous les scénarios d'attaque donnés dans le tableau 5.1. ProB a également exhibé des scénarios où Bob participe à la réalisation des scénarios comme le scénario suivant :

$$\begin{aligned} Q'_{S_5} = < \ & \textbf{Connect(Bob, \{MemberUser\})}, \\ & secure\_Member\_Free(Bob), \\ & \textbf{Connect(John, \{MemberUser\})}, \\ & secure\_Member\_AddLend(John, bo) > \end{aligned}$$

Mais cela peut être évité en excluant de la recherche les utilisateurs qui peuvent être considérés de confiance, les victimes, ou les utilisateurs ainsi que les rôles jugés comme non pertinents.

5.4.5 Extraction automatique des attaques masquées

A partir d'une séquence d'un comportement malicieux Q_F révélant une attaque basique, à contrainte ou planifiée, il est possible d'exhiber automatiquement le comportement malicieux traduisant une attaque masquée. Cette séquence peut être extraite sans appliquer de nouveau l'algorithme d'extraction de séquences vérifiant l'atteignabilité de l'état initial à partir de l'état indésirable.

En effet, dans une attaque masquée l'attaquant cherche à camoufler ses actions en effectuant les opérations inverses. Par conséquent, en nous basant sur le standard de nommage des opérations

générées par la plate-forme B4MSecure nous pouvons identifier l'opération inverse $op_{inverse}$ à chacune des opérations op de la séquence Q_F. Dans le tableau 5.2 nous donnons les couples des opérations inverses.

Opération op	**Opération inverse** $op_{inverse}$
Class_New	Class_Free
Class_Free	Class_New
Class_AddXXX	Class_RemoveXXX
Class_SetXXX	Class_RemoveXXX
Class_RemoveXXX	Class_AddXXX
Class_GetXXX	-

TABLEAU 5.2 – Les couples des opérations inverses générées par B4MSecure

Ainsi, pour chaque séquence d'opérations composée par les opérations de base générées par B4MSecure, nous définissons la séquence d'opérations inverses correspondante :

Définition 17 (Séquence inverse).
Étant donnée une séquence d'opérations $Q_F =< op_1, op_2, \ldots, op_n >$, nous notons $Q_{F_{inverse}}$ sa séquence d'opérations inverse telle que :

$$Q_{F_{inverse}} =< op_{n_{inverse}}, \ldots, op_{2_{inverse}}, op_{1_{inverse}} >$$

En nous basant sur cette définition, nous cherchons à identifier des attaques masquées à partir des autres types d'attaques. En effet, ayant un comportement malicieux Q_F, nous générons le comportement malicieux $Q_{F_{masque}}$ tel que :

$$Q_{F_{masque}} = Q_F; Q_{F_{inverse}}$$

L'avantage majeur de cette technique est qu'elle permet de faire une première analyse rapide qui peut aider à identifier des attaques masquées. Par exemple, en considérant le comportement malicieux de l'attaque basique donné à la figure 5.2, il est possible d'extraire le comportement malicieux de l'attaque masquée de la figure 5.5 en appliquant cette approche.

En effet, en partant du comportement malicieux suivant :

$$\begin{aligned} Q_F =< \quad & Member\_New(Alice), \\ & Member\_AddLend(Alice, bo2) \quad > \end{aligned}$$

il est possible d'extraire le comportement malicieux $Q_{F_{masque}}$ de l'attaque masquée en appliquant la technique proposée :

$$\begin{aligned} Q_{F_{masque}} =< \quad & Member\_New(Alice), \\ & Member\_AddLend(Alice, bo2), \\ & Member\_RemoveLend(Alice, bo2), \\ & Member\_Free(Alice) \quad > \end{aligned}$$

Cependant, il est à noter que cette approche ne garantit pas toujours l'atteignabilité de l'état initial par la séquence $Q_{F_{inverse}}$ à partir de l'état indésirable atteint par la séquence Q_F.

Par exemple, nous considérons la séquence du comportement malicieux Q_{F_1} donnée dans le tableau 5.1. Selon la définition 17, la séquence suivante décrit un comportement malicieux d'une attaque masquée :

$$\begin{aligned} Q_{F_{masque}} =< \quad & Book\_Free(bo), \\ & Book\_New(bo), \\ & Member\_AddLend(John, bo), \\ & Member\_RemoveLend(John, bo), \\ & Book\_Free(bo), \\ & Book\_New(bo) \quad > \end{aligned}$$

Toutefois, nous pouvons remarquer que cette séquence ne permet pas d'atteindre l'état initial où le livre `bo` est réservé à la victime.

Pour cette raison, il est important de vérifier l'atteignabilité de l'état initial par les séquences générées. Cette vérification peut être faite par une simple animation.

5.5 Bilan et discussion

Nous avons proposé dans ce chapitre une approche pour l'extraction des scénarios d'attaque pouvant être commis par des utilisateurs internes au SI en tirant profit de l'évolution dynamique de l'état fonctionnel du système. Dans cette approche, nous exploitons les modèles générés par la plate-forme B4MSecure en tirant avantage de l'indépendance des modèles fonctionnel et de sécurité ainsi que des liens entre ces deux modèles. Ceci a favorisé la mise en place d'une méthode en deux phases :

— Dans la première phase nous exploitons le modèle fonctionnel afin d'extraire des séquences d'opérations permettant d'atteindre un état cible caractérisé comme étant indésirable et révélant des comportements malicieux. Pour ce faire, nous appliquons notre algorithme de recherche d'atteignabilité par retour en arrière basé sur la résolution de contraintes ou sur la preuve. L'étude de cas du SI de la bibliothèque a montré que l'approche basée sur la résolution de contraintes est moins coûteuse et plus facilement automatisable. En effet, les deux approches ont permis l'extraction des comportements malicieux reportés dans le tableau 5.1. Cependant, il est à noter que 90% des obligations de preuve générées par l'approche basée sur la preuve sont trop compliquées pour être déchargées automatiquement par l'outil de preuve AtelierB. Par conséquent, nous étions obligée de les prouver manuellement ou d'utiliser un outil de preuve interactif. Au contraire, le solveur de contraintes ProB a réussi à résoudre automatiquement toutes les contraintes relatives aux preuves dans le domaine spécifié des variables.

— Dans la deuxième phase nous utilisons une technique qui permet de guider le model-checker par l'approche CSP||B en vue de vérifier si le modèle de sécurité autorise de tels compor-

tements malicieux. Ainsi, contrôlé par un processus CSP décrivant les comportements malicieux extraits dans la première phase, le model-checker cherche à identifier l'utilisateur malicieux ou l'ensemble des utilisateurs qui peuvent former une collusion afin de réaliser les opérations des séquences fonctionnelles. Il cherche également à identifier les rôles associés aux utilisateurs permettant à ces derniers d'accomplir leurs actes malveillants.

En utilisant un model-checker classique sans guidance pour extraire les comportements malicieux du même exemple, nous avons pu extraire à partir du modèle fonctionnel la séquence donnée dans la figure 5.14 après avoir exploré plus de 200 transitions.

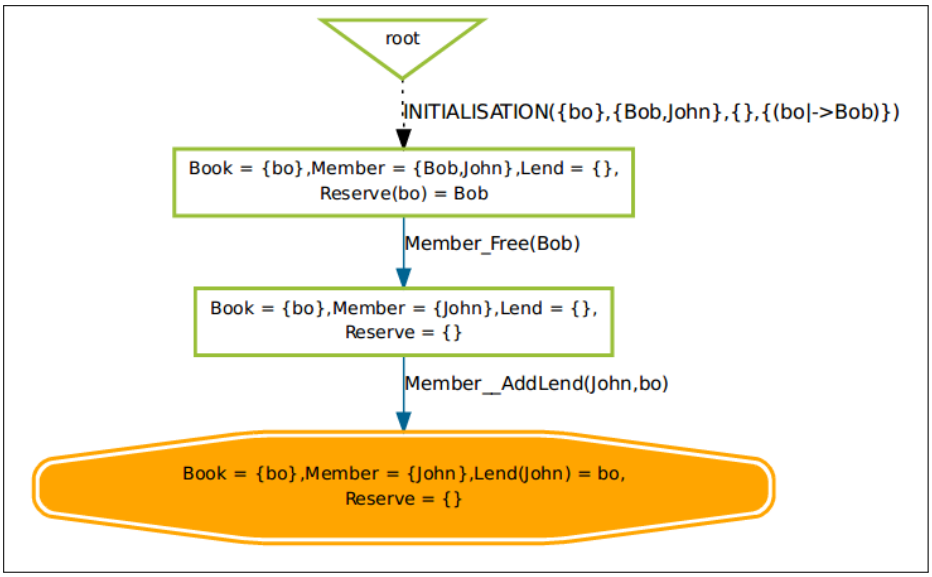


FIGURE 5.14 – Séquence fonctionnelle atteignant l'état indésirable

Ce nombre de transitions a été multiplié par trois pour extraire le scénario d'attaque incluant les opérations de connexion directement à partir du modèle de sécurité. Ceci montre deux choses essentielles :

(i) D'une part, notre approche réduit considérablement l'espace de recherche. En effet, pour trouver le même scénario d'attaque le model-checker a consommé uniquement une dizaine de transitions en le guidant par le processus CSP `Attack`$_1$ (figure 5.12) et une trentaine de transitions en fournissant le processus CSP général `Main` (figure 5.13).

(ii) D'autre part, l'idée de travailler en deux phases en commençant par la recherche des comportements malicieux à partir du modèle fonctionnel est très utile et permet de faciliter considérablement la recherche. En effet, le nombre important de variables d'état du modèle de sécurité ainsi que les gardes relatives aux filtres de sécurité dans chaque opération rend la recherche plus lourde dans le modèle de sécurité aussi bien pour un model-checker que pour un outil de preuve ou un solveur de contraintes.

Nous avons également montré dans ce chapitre, à travers l'exemple du SI de la bibliothèque, que notre approche est capable d'extraire et d'identifier des scénarios d'attaque. L'utilité de l'approche a été démontrée plus largement par des expérimentations sur différentes études de cas

plus complexes où nous avons pu extraire des scénarios d'attaques de différents types (basique, à contrainte...). Les résultats très encourageants de ces expérimentations seront discutés plus en détail dans le chapitre suivant. De plus, l'approche est complètement automatisable ce qui facilite sa mise en œuvre. Nous avons ainsi implémenté l'outil GenISIS que nous présentons dans le chapitre suivant pour l'automatisation de l'extraction des comportements malicieux. Quant à la deuxième phase, nous exploitons l'outil ProB qui offre une facilité pour la mise en œuvre d'une approche CSP||B.

Notre approche a aussi l'avantage qu'elle soit applicable dans le contexte plus général de la vérification de la sécurité du SI. En effet, il est possible d'utiliser notre technique d'extraction de scénarios d'attaque pour vérifier que les filtres de sécurité sont correctement définis. Cette vérification peut s'effectuer à deux niveaux : D'une part, nous vérifions que ces filtres n'empêchent pas la réalisation des scénarios de cas d'utilisation normaux. D'autre part, nous vérifions qu'ils autorisent l'accès aux utilisateurs ayant les droits requis, et le bloquent pour les utilisateurs qui ne remplissent pas au moins l'une des règles de sécurité. Nous expliquons plus en détail l'applicabilité de notre approche dans ce contexte dans l'annexe B.

Chapitre 6

Outil et expérimentations

« The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency. »

Bill Gates

Sommaire

6.1 Présentation de l'outil GenISIS 119
6.1.1 Architecture générale 119
6.1.2 Architecture structurelle de GenISIS 123
6.2 Expérimentations et discussion 125
6.3 Amélioration de la performance de GenISIS 127
6.3.1 Approche basée sur l'analyse syntaxique et la preuve 127
6.3.2 Approche basée sur les heuristiques 131
6.4 Etude de cas 133
6.4.1 Modèle fonctionnel 133
6.4.2 Modèle de sécurité 133
6.4.3 Extraction de scénarios d'attaque 136
6.5 Conclusion 141

Afin d'automatiser l'approche que nous avons proposée pour l'extraction de scénarios d'attaque à partir d'une modélisation B d'un SI, nous avons développé l'outil GenISIS (acronyme de Generator of Insider Scenarios from an Information System).

Cet outil principalement dédié à la vérification de la sécurité d'un SI peut aussi être utilisé dans le contexte général de la vérification de la propriété d'atteignabilité en B. En effet, nous développons dans cet outil l'approche de recherche avec retour en arrière permettant d'exhiber des

traces d'exécution menant à une cible donnée. Pour les raisons que nous avons discutées dans les chapitres précédents notamment la facilité d'automatisation et la complexité moindre des preuves, nous avons favorisé la méthode basée sur la résolution de contraintes.

Dans ce chapitre, nous présentons l'architecture de l'outil et nous discutons ses performances en nous basant sur les différentes expérimentations que nous avons menées. Nous proposons également deux techniques pour améliorer la performance de l'outil. Finalement, nous appliquons notre outil sur une étude de cas plus complexe que l'exemple du SI de la bibliothèque et nous montrons sa capacité à extraire différents types d'attaque.

6.1 Présentation de l'outil GenISIS

l'outil GenISIS est un programme java open source [1] permettant d'extraire des séquences d'opérations fonctionnelles Q_F vérifiant la propriété d'atteignabilité $S^I_{\mathsf{P}} \overset{Q_F}{\rightsquigarrow} S^F_{\mathsf{P}}$ à partir d'une spécification B d'un système. Pour ce faire, il met en œuvre l'algorithme de retour en arrière basé sur la résolution de contraintes que nous avons présenté dans le chapitre 4. Il permet également d'analyser les contraintes d'autorisation que nous décrivons dans un langage dédié en vue d'extraire des séquences d'opérations révélant des comportements malicieux.

6.1.1 Architecture générale

La figure 6.1 présente l'architecture générale de GenISIS qui prend en entrée les spécifications décrivant le SI ainsi que l'état cible que nous cherchons à atteindre afin de produire un ensemble de traces d'exécution menant à l'état cible.

6.1.1.1 Les spécifications des propriétés initiales et finales

GenISIS prend obligatoirement en entrée le prédicat $\mathsf{P_F}$ caractérisant l'état cible tel qu'il est exprimé en B ainsi que la machine B du système à analyser. Quant à l'état initial, il est extrait directement à partir de l'initialisation (clause `INITIALISATION`) de la machine B. L'utilisateur peut aussi fournir en entrée un fichier optionnel appelé *fichier CCL (Contextual Constraint Language)* qui décrit les contraintes d'autorisation dans le cas où l'outil est utilisé pour l'extraction de comportements malicieux.

Ainsi, nous avons choisi de séparer la description des contraintes de sécurité des spécifications fonctionnelles et de ne pas utiliser des spécifications fonctionnelles enrichies par des contraintes d'autorisation comme nous l'avons expliqué dans le chapitre précédent. Ce choix est surtout motivé par des raisons organisationnelles et des raisons d'évolutivité de l'outil. En effet, il est important d'un point de vue organisationnel de garder une séparation claire entre les préoccupations fonctionnelles et non fonctionnelles. En outre, cette séparation permet d'envisager plus tard un interfaçage entre GenISIS et B4MSecure afin d'avoir un outil plus complet qui permet de spécifier graphiquement et de valider la sécurité d'un SI en analysant les modèles générés. Il pourrait

1. Le code source et le programme sont fournis à l'adresse : `http://genisis.forge.imag.fr/`

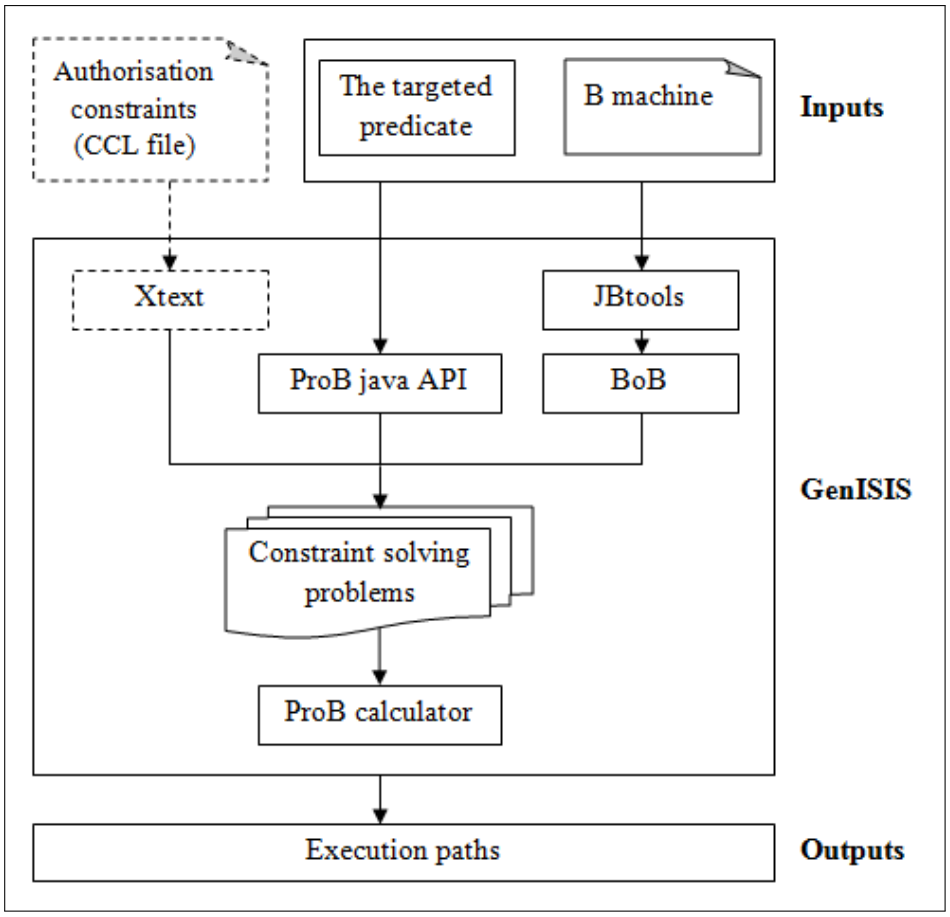


FIGURE 6.1 – Architecture générale de GenISIS

donc être utile d'analyser les spécifications fonctionnelles telles qu'elles sont générées par la plateforme B4MSecure et de les compléter en générant un autre fichier comportant la description des contraintes d'autorisation.

Le fichier CCL permet, non seulement de décrire les contraintes d'autorisation, mais aussi d'optimiser la recherche en permettant d'exclure certaines opérations jugées comme non pertinentes pour un scénario d'attaque, ou en mettant l'accent sur un ensemble d'opérations spécifique. Ceci permet d'accélérer le processus de recherche en réduisant le nombre de sollicitations du solveur de contraintes. En effet, les preuves d'atteignabilité pour les opérations exclues ne seront ni générées ni résolues. Pour décrire ce fichier CCL, nous avons défini un nouveau langage dédié inspiré du langage B et dont les composants sont introduits par les clauses suivantes :

— `OPERATIONS` : Dans cette clause nous déclarons les opérations concernées par les contraintes d'autorisation spécifiées dans le fichier. Nous énumérons ainsi les noms des opérations en les séparant par des espaces.

— `VARIABLES` : Les variables additionnelles nécessaires pour l'expression des contraintes d'autorisation et qui ne sont pas déclarées dans la machine B du modèle fonctionnel doivent être déclarées dans cette clause.

— `PROPERTIES` : Similairement à la méthode B, dans cette clause nous définissons les propriétés de typage ou d'invariance relatives aux variables déclarées dans la clause `VARIABLES`

— `Constraint` : Chaque contrainte d'autorisation doit être définie dans une instance de cette clause où nous spécifions le nom de la contrainte, le prédicat caractérisant la contrainte tel qu'il est exprimé en B, ainsi que l'ensemble des opérations qui sont concernées par la contrainte.

— `Non-critical` : Elle permet d'exclure un ensemble d'opérations du processus de recherche. Ces opérations sont considérées comme non critiques et ne peuvent pas faire partie d'un scénario d'attaque. Par exemple, nous pouvons considérer que les opérations de lecture sont non critiques car elles ne font pas évoluer l'état fonctionnel du système.

En guise d'exemple, nous donnons dans la figure 6.2 un fichier CCL que nous pouvons considérer pour l'extraction de comportements malicieux du SI de la bibliothèque.

```
  OPERATIONS   Member_TakeReservedBook
               Member_AddLend
               Member_RemoveLend
               Member_AddReserve
               Member_RemoveReserve

VARIABLES  currentUser

PROPERTIES  'currentUser ∈ Member'

Constraint  C1 {
    'Instance = currentUser'
    Operations  :  Member_TakeReservedBook
                   Member_AddLend
                   Member_RemoveLend
                   Member_AddReserve
                   Member_RemoveReserve
}
 Non-critical  :  Member_GetLend
                  Member_GetReserve
                  Book_GetLend
                  Book_GetReserve
```

FIGURE 6.2 – Exemple d'un fichier CCL

6.1.1.2 Mode opératoire

Pour extraire les chemins menant à la cible donnée à partir des spécifications reçues en entrée, GenISIS suit les étapes suivantes :

1. Analyse syntaxique des spécifications : GenISIS utilise l'analyseur syntaxique JBtools [VOISINET et al. 2002] pour charger les composants B à partir de la machine représentant la spécification fonctionnelle du SI. Une fois les composants chargés, nous nous servons de la Boite à Outils B (BoB) [2] pour manipuler ces composants. En particulier, nous exploitons ses fonctionnalités de calcul de la plus faible pré-condition d'une substitution et de calcul des prédicats de terminaison et de faisabilité des opérations ainsi que le prédicat avant-après. Nous tirons également profit des fonctionnalités offertes par l'API java de l'outil ProB [3] principalement pour l'extraction de l'état initial à partir de la machine B et pour le calcul du prédicat représentant un état atteint après l'exécution d'une suite d'opérations. Quant à la spécification CCL, nous avons développé un analyseur syntaxique au moyen de l'outil Xtext [4].

2. Génération des obligations de preuve et des problèmes à contraintes : Après avoir extrait toutes les données nécessaires, GenISIS parcourt toutes les opérations de la machine B, à l'exception des opérations exclues au moyen de la spécification CCL, et génère les obligations de preuve et les problèmes à contraintes nécessaires pour vérifier la précédence entre deux états à chaque récursion de l'algorithme. Les formules générées seront répertoriées dans des fichiers séparés pour pouvoir les examiner et les exploiter plus tard.

3. Résolution des problèmes à contraintes : Comme avancé précédemment, nous nous servons du solveur de contraintes de ProB (ProB Calculator) fourni dans l'API java de ProB pour résoudre automatiquement les problèmes à contraintes générés à l'étape précédente. Toutefois, il est possible d'envisager une extension de l'outil pour mettre en œuvre l'approche basée sur la preuve en complément de l'approche basée sur la résolution de contraintes. Nous envisageons, donc, de faire appel à l'outil de preuve AtelierB pour décharger les obligations de preuve.

2. La BoB est un outil développé au sein de l'équipe VASCO du laboratoire LIG.
3. http ://www3.hhu.de/stups/prob/index.php/ProB_Java_API
4. http://www.eclipse.org/Xtext/

6.1.2 Architecture structurelle de GenISIS

Nous donnons l'architecture structurelle détaillée de GenISIS dans la figure 6.3 qui comporte les composants suivants :

— La classe `Bspec` charge les composants de la machine B d'entrée de notre outil. Elle permet de consulter les éléments de cette machine. Elle permet également de les manipuler à l'aide de l'outil BoB après leur chargement au moyen de la classe `BspecLoader`. Cette dernière fait appel à l'outil JBtools pour effectuer l'analyse syntaxique de la machine B. Nous avons ainsi séparé les spécifications B et les calculs que nous pouvons effectuer sur ces spécifications (`Bspec`) de la manière dont nous les récupérons (`BspecLoader`). Ceci rend notre outil plus facile à maintenir, et à faire évoluer. La classe `Bspec` a aussi recours à l'outil ProB pour effectuer l'animation de scénarios des séquences d'opérations obtenues ainsi que pour l'extraction de l'état initial.

— La classe `CCLspec` stocke l'ensemble des éléments de la spécification CCL. Elle permet également d'analyser ces éléments après les avoir chargés par le biais de la classe `CCLspecLoader`.

— La classe `ConcreteOp` représente une exécution (une opération de la machine B en donnant une valuation à ses paramètres). Cette classe utilise l'outil BoB en héritant de sa classe `TOperation` et en ajoutant l'attribut `values` dans lequel nous stockons les valeurs des paramètres de l'opération.

— La classe `FunctionalAttackScenario` représente une séquence d'opérations fonctionnelles permettant d'atteindre l'état cible. Elle est composée d'au moins une opération concrète (classe `ConcreteOp`).

— La classe `InputSpec` est la classe qui implémente notre algorithme de recherche.

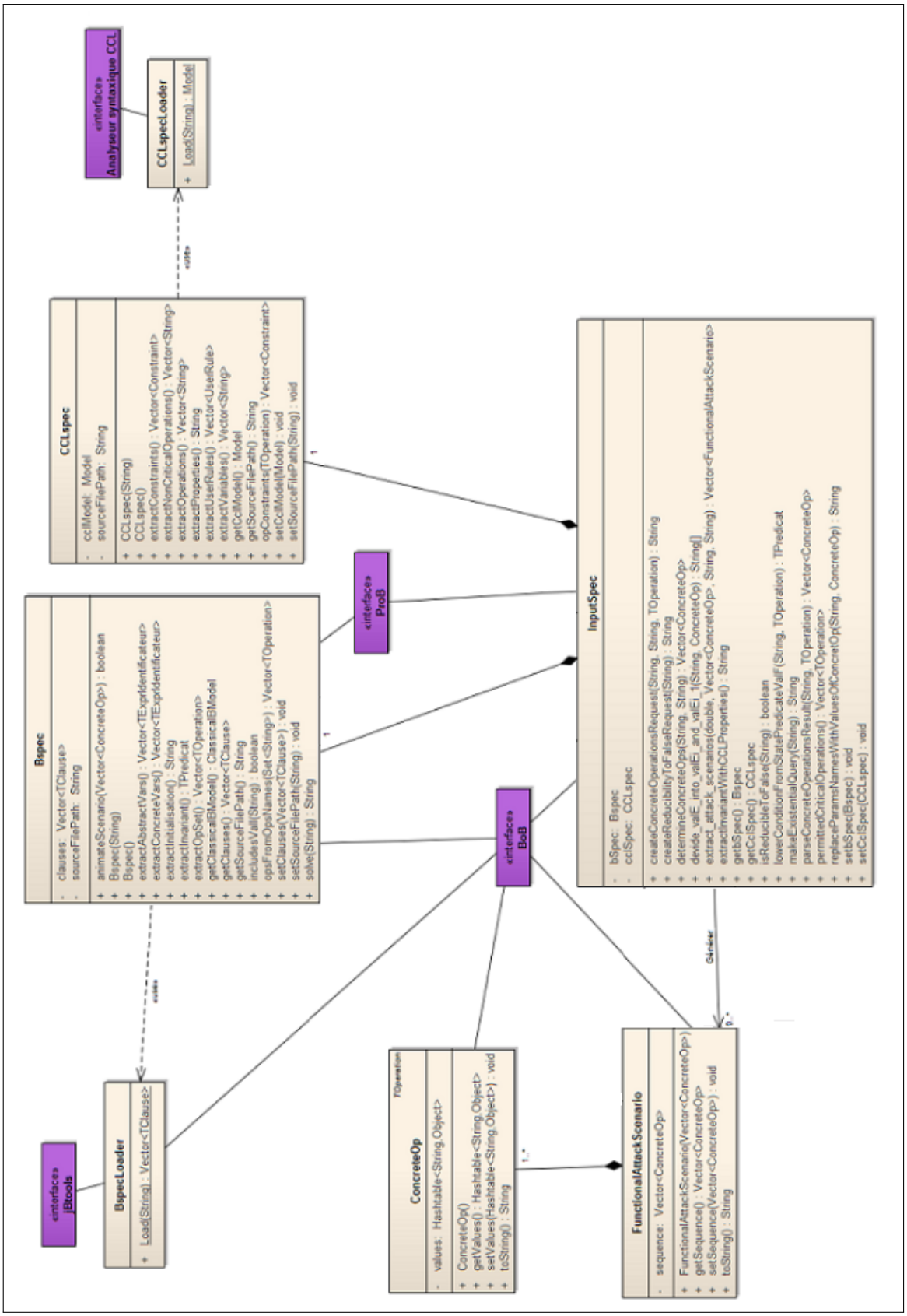


FIGURE 6.3 – architecture structurelle détaillée de GenISIS

6.2 Expérimentations et discussion

Afin d'étudier la capacité de notre outil à extraire des scénarios d'attaque, nous l'avons testé sur plusieurs cas d'étude que nous résumons dans le tableau 6.1. Les exemples traités étaient tous inspirés d'autres travaux similaires qui ont cherché eux aussi à identifier des attaques internes favorisées par l'évolution dynamique de l'état fonctionnel du système. Par conséquent, nous avons modélisé ces exemples en utilisant la plate-forme B4MSecure qui nous a permis de générer les spécifications B sur lesquelles nous avons appliqué notre outil GenISIS. Ces expérimentations nous ont permis de comparer notre approche aux autres approches. En effet, pour chaque étude de cas, nous avons ciblé l'état indésirable traité par l'article qui portait sur le même exemple, et nous avons été capable d'extraire toutes les attaques rapportées voire pour certains cas d'identifier de nouvelles attaques qui n'ont pas été discutées. Par exemple, pour le cas du SI du planificateur de réunions qui a été étudié dans [QAMAR 2011] en vue d'extraire des scénarios d'attaque moyennant une approche basée sur la méthode Z, nous avons montré dans [RADHOUANI et al. 2015] que notre approche basée sur la preuve est capable d'extraire le même scénario d'attaque que celui discuté dans [QAMAR 2011]. Nous avons également identifié un autre scénario qui n'a jamais été rapporté. Ces scénarios ont été extraits automatiquement par l'outil GenISIS.

Etude de cas	**O**	**V**	**P**	**R**	**U**	**C**	**S**
SI d'une bibliothèque [FRAPPIER et al. 2011, MAMMAR et al. 2011]	13	4	3	2	3	1	8
SI médical [LEDRU et al. 2014]	15	9	3	4	3	1	10
SI planificateur de réunions [QAMAR 2011]	23	7	5	3	3	1	8
SI bancaire [BANDARA et al. 2010]	31	11	4	2	3	1	9
SI de revue d'articles dans une conférence [ZHANG et al. 2008]	48	24	8	3	4	5	14

O : Nombre d'opérations, **V** : Nombre de variables d'états, **P** : Nombre de permissions, **R** : Nombre de rôles, **U** : Nombre d'utilisateurs, **C** : Nombre de contraintes d'autorisation, **S** : Nombre de scénarios extraits

TABLEAU 6.1 – Tableau de synthèse des expérimentations

Ces expérimentations menées sur différentes études de cas de tailles variées ont montré que la taille des spécifications fonctionnelles en terme de nombre de variables et de nombre d'opérations influe sur la performance et le temps de réponse de l'outil. En effet, la phase la plus complexe de calcul des séquences est la phase de résolution de contraintes. Ainsi, la performance de notre

outil dépend principalement de la performance du solveur de contraintes de ProB et de la taille du problème. Par conséquent, le nombre de variables ainsi que la taille des ensembles manipulés impactent considérablement aussi bien l'espace de recherche que le temps d'exécution.

Par exemple, lors de l'application de notre outil sur le SI de la bibliothèque, nous étions capable d'extraire les scénarios d'attaque discutés dans le chapitre 5 en 2746 ms en considérant un seul élément dans l'ensemble `BOOK` et un seul élément dans l'ensemble `MEMBER`. Ce temps d'exécution a été presque triplé (9943 ms) quand nous avons refait le test avec des ensembles contenant deux éléments (voir tableau 6.2).

Problème traité	**Nombre de livres**	**Nombre de membres**	**Temps d'exécution (ms)**	**Occupation mémoire (MB)**
Problème d'atteignabilité (chapitre 4)	1	1	2 746	46
	2	2	9 943	98
	3	3	12 841	106
	5	5	817 477	152
	3	10	1 854 224	146
Extraction de scénarios d'attaque (chapitre 5)	1	3	16 586	61
	3	3	19 433	94
	3	10	2 331 531	146

TABLEAU 6.2 – Tableau de calcul des performances de GenISIS pour le traitement de l'exemple de la bibliothèque

Nous donnons également les mesures de performance relatives à l'exemple du SI de revue d'articles dans une conférence dans le tableau 6.3. Cet exemple, qui sera détaillé dans la section 6.4, requiert le plus de temps de traitement vu qu'il intègre un grand nombre d'opérations ainsi qu'un nombre important de variables d'états.

Nombre d'auteurs	**Nombre d'articles**	**Nombre de reviews**	**Nombre de rapporteurs**	**Temps d'exécution (ms)**	**Occupation mémoire (MB)**
3	1	2	3	9 982	108
1	2	2	5	45 257	112
3	3	3	5	570 466	112
3	5	5	8	12 334 453	132

TABLEAU 6.3 – Tableau de calcul des performances de GenISIS pour le traitement de l'exemple du SI de revue d'articles dans une conférence

Cependant, l'avantage de l'approche en deux phases que nous utilisons est que le nombre de permissions et de rôles n'a aucune influence sur la performance de GenISIS du moment où la phase de recherche de comportement malicieux opère sur le modèle fonctionnel et fait une abstraction sur les données du modèle de sécurité. Certes, le nombre d'utilisateurs et le nombre de rôles impactent

la deuxième phase de l'approche qui exploite le model-checker ProB guidé par un contrôleur CSP, mais cet impact reste relativement faible.

Le nombre d'opérations, quant à lui, affecte considérablement le temps d'exécution et l'espace mémoire utilisé par l'outil, car, plus on a d'opérations, plus on a besoin d'appels au solveur de contraintes et plus on a besoin d'appels récursifs dans notre algorithme de recherche. Ainsi, pour améliorer la performance de notre outil nous proposons d'explorer deux pistes que nous présentons dans la section suivante : l'une basée sur l'analyse syntaxique et la preuve, et l'autre basée sur les heuristiques.

6.3 Amélioration de la performance de GenISIS

6.3.1 Approche basée sur l'analyse syntaxique et la preuve

Une analyse syntaxique des variables manipulées par les préconditions et les actions des opérations permet d'identifier les relations de dépendance entre les opérations. Ces dépendances peuvent être explorées dans le but d'améliorer la performance de GenISIS et d'éviter qu'il fasse des appels inutiles au solveur de contraintes. En effet, dans chaque récursion de notre algorithme (figure 4.13), nous essayons toutes les opérations du SI étudié en vue d'extraire celles qui satisfont la propriété de succession entre les deux états définis. Cependant, il existe des opérations qui sont **syntaxiquement indépendantes** (voir la définition 18), comme il existe des opérations qui ne peuvent pas apparaître avant certaines opérations dans une séquence. Autrement dit, il y a des opérations qui ne peuvent pas se succéder, car l'exécution de l'une rend impossible le déclenchement de l'autre. Par conséquent, ces opérations peuvent être identifiées et éliminées dès le début du calcul. L'analyse que nous faisons dans ce contexte suit les étapes suivantes :

1. **Élimination des opérations ne faisant pas évoluer l'état fonctionnel** :
   Il est évident que ces opérations ne peuvent pas être impliquées dans une séquence menant d'un état à un autre, il serait donc approprié de les exclure du processus de recherche. Il s'agit alors d'éliminer les opérations dont les actions ne modifient aucune variable d'état, telles que les opérations ayant une action sans effet (`SKIP`) ou les opérations de lecture comme tous les getters dans les spécifications que nous traitons. Ces opérations peuvent être supprimées de la spécification B ou déclarées dans la section `Non-Critical` du fichier CCL. Dans notre exemple du SI de la bibliothèque, cette première analyse, qui pourrait être automatisée, nous permet d'éliminer quatre opérations et donc d'économiser quatre appels au solveur de contraintes à chaque récursion de l'algorithme.

2. **Analyse syntaxique des dépendances entre les opérations** :
   Nous disons qu'une opération op_1 est indépendante d'une autre opération op_2 si l'exécution

de op_1 n'influence pas la déclenchabilité de op_2. Formellement, nous définissons cette indépendance syntaxique comme suit :

Définition 18 (Indépendance entre deux opérations).
op_1 est indépendante de op_2 ssi :

$$\mathbf{V}_{act}(op_1) \cap \mathbf{V}_{pre}(op_2) = \emptyset$$

tel que $\mathbf{V}_{act}(op)$ dénote les variables manipulées par l'action de l'opération op, et $\mathbf{V}_{pre}(op)$ dénote les variables impliquées dans la précondition de op.
Dans le cas où :

$$\mathbf{V}_{act}(op_1) \cap \mathbf{V}_{pre}(op_2) = \emptyset \wedge \mathbf{V}_{act}(op_2) \cap \mathbf{V}_{pre}(op_1) = \emptyset$$

Nous disons que op_1 et op_2 sont toutes les deux indépendantes l'une de l'autre et nous notons $op_1 \not\perp op_2$

Proposition 4

Si $op_1 \not\perp op_2$ alors : la séquence d'opération <op_1, op_2> est équivalente à la séquence <op_2, op_1>

Démonstration. Si la séquence d'opération <op_1, op_2> est réalisable et que op_1 est indépendante de op_2, alors op_2 est aussi déclenchable à partir de l'état qui déclenche op_1.
L'opération op_2 est aussi indépendante de op_1, alors l'exécution de op_2 à partir d'un état où op_1 est déclenchable ne change rien à la possibilité de déclenchement de op_1.
Par conséquent, la séquence <op_2, op_1> est aussi réalisable et mène au même état atteint par la séquence <op_1, op_2>. □

En exploitant la proposition 4 nous pouvons réduire le nombre de récursions de l'algorithme et en même temps éviter l'extraction de séquences équivalentes. En effet, si nous avons deux opérations op_1 et op_2 tel que $op_1 \not\perp op_2$, alors nous pouvons imposer un ordre d'apparition entre ces deux opérations. Par exemple, si nous imposons que op_1 apparaît toujours avant op_2, alors dans le cas où op_1 apparaît dans une récursion donnée de l'algorithme, nous éliminons op_2 de l'ensemble des opérations à explorer dans la récursion suivante.

Pour mieux illustrer cette notion nous prenons l'exemple des deux opérations du SI de la bibliothèque `Member_New` et `Book_New`. En effet, nous avons :
`Book_New` $\not\perp$ `Member_New` car :

$$\big(\mathbf{V}_{act}(\texttt{Book\_New}) = \{\texttt{Book}\}\big) \cap \big(\mathbf{V}_{pre}(\texttt{Member\_New}) = \{\texttt{Member}\}\ \big) = \emptyset$$
$$\wedge$$
$$\big(\mathbf{V}_{pre}(\texttt{Book\_New}) = \{\texttt{Book}\}\big) \cap \big(\mathbf{V}_{act}(\texttt{Member\_New}) = \{\texttt{Member}\}\ \big) = \emptyset$$

Ainsi, lors du déroulement de notre algorithme par retour en arrière (figure 4.13), nous imposons l'ordre `Book_New`, `Member_New`. Ainsi, dans la récursion qui suit l'extraction de l'opération `Book_New` nous excluons l'opération `Member_New` de l'ensemble des opérations *Opset*. Cette nouvelle règle a permis de gagner 26 appels au solveur de contraintes et elle nous a aussi évité l'extraction du scénario Q_{F_2} qui est équivalent au scénario Q_{F_1} :

Scénario Q_{F_1}	Scénario Q_{F_2}
`Book_Free(bo),`	`Book_Free(bo),`
`Book_New(bo),`	`Member_New(Alice),`
`Member_New(Alice),`	`Book_New(bo),`
`Member_AddLend(Alice, bo)`	`Member_AddLend(Alice, bo)`

3. **Analyse de la relation de succession entre les opérations** :
   Il existe des opérations qui ne peuvent jamais se succéder car l'exécution de l'une rend impossible la déclenchabilité de l'autre. Ainsi, si l'action d'une opération op_1 invalide l'une des contraintes de la précondition d'une opération op_2, alors l'opération op_1 peut être éliminée lors de la recherche de l'opération qui précède op_2 dans une séquence d'opérations. Formellement, nous définissons deux opérations qui ne se succèdent pas moyennant la preuve de non atteignabilité que nous avons expliquée au chapitre 4 :

Définition 19 (Non succession d'opération).
Une opération op_1 ne précède jamais une opération op_2 dans une séquence et nous notons $op_1 \not\triangleright op_2$ ssi : $\forall var \cdot Pre(op_1) \Rightarrow [Action(op_1)]\neg Pre(op_2)$

Ces trois analyses peuvent nettement améliorer la performance de GenISIS. De plus, elles ne sont pas très coûteuses du moment où elles sont effectuées une seule fois au début de l'analyse. Elles peuvent également servir pour toutes les activités de vérification d'atteignabilité ou d'extraction de scénarios d'attaque à partir du même modèle B.

Ainsi, pour chaque opération op nous calculons les deux ensembles suivants :

— $\not\perp_{op}$: l'ensemble des opérations syntaxiquement indépendantes pour lesquelles nous avons attribué un ordre d'apparition supérieur à op.

— $\not\triangleright_{op}$: l'ensemble des opérations qui ne peuvent pas précéder l'opération op,

et nous modifions la ligne 9 de l'algorithme 4.13 comme suit :

```
9. for each  o_i ∈ Opset − {⊥̸_{o_i}, ▷̸_{o_i}} do
```

Une analyse similaire appelée *Enabling Analysis* [5] est proposée dans l'outil ProB. Cette analyse permet de calculer la dépendance syntaxique entre les opérations prises deux à deux, ainsi que

5. https://www3.hhu.de/stups/prob/index.php/Tutorial_Enabling_Analysis

l'effet de l'action d'une opération sur la déclenchabilité d'une autre et ce en utilisant la résolution de contraintes. ProB permet par la suite de générer une matrice de dépendance que nous pouvons exploiter afin de déduire les deux ensembles $\not\perp_{op}$ et $\not\triangleright_{op}$. Par exemple, l'ensemble des opérations qui peuvent être exécutées avant l'opération `Member_AddLend` généré par ProB est donné par la figure 6.4.

Event	Enable
Member__TakeReservedBook	false
Book_NEW	ok
Member_NEW	ok
Book_Free	false
Member_Free	ok
Book__GetLend	false
Member__GetLend	false
Member__GetReserve	false
Book__GetReserve	false
Member__AddLend	false
Member__AddReserve	false
Member__RemoveLend	ok
Member__RemoveReserve	ok

FIGURE 6.4 – Analyse de déclenchabilité de l'opération `Member_AddLend` (généré par ProB)

Par conséquent, dans la récursion qui suit l'extraction de l'opération `Member_AddLend` nous essayons seulement 5 opérations parmi 13 pour exhiber l'opération précédente dans la séquence recherchée ce qui peut réduire considérablement le temps d'exécution.

Toutefois, étant basée sur la résolution de contraintes, l'approche proposée dans ProB dépend des domaines des variables d'états. Par conséquent, le calcul de dépendance entre les opérations doit être actualisé à chaque modification dans les domaines des variables (par exemple : ajout ou retrait d'un élément dans les ensembles énumérés). Par exemple, le résultat donné dans la figure 6.4 a été calculé en considérant le domaine des variables suivant :

```
SETS
  BOOK = {bo};
  MEMBER = {Alice, Bob, John}
```

Au contraire, l'approche basée sur la preuve que nous proposons est plus générale et permet de déduire des conclusions qui sont valables pour toutes les expérimentations menées sur le même cas d'étude.

6.3.2 Approche basée sur les heuristiques

Les expérimentations que nous avons menées sur différentes études de cas ont montré que plusieurs scénarios d'attaque menant au même état indésirable ont en commun une même opération qui peut être considérée comme la clé de l'attaque. Par exemple, nous pouvons remarquer que les deux opérations `Book_Free` et `Member_Free` sont les opérations clés qui ont ouvert l'accès aux autres opérations et qui ont rendu possible les scénarios d'attaque présentés dans le tableau 5.1. Par conséquent, la modification de ces deux opérations est suffisante pour corriger le modèle et neutraliser tous les scénarios d'attaque extraits. Ainsi, l'extraction d'un seul scénario peut parfois suffire pour identifier toutes les failles dans le modèle. C'est pour cette raison que nous proposons de faire une recherche en profondeur d'abord en commençant par extraire les chemins d'exécution les plus courts. Ceci permet, d'une part, de produire des résultats assez rapidement, et d'autre part, de donner une idée des opérations critiques qui peuvent être à l'origine de plusieurs autres attaques. Une fois ces opérations corrigées nous pouvons lancer une nouvelle recherche et ainsi de suite jusqu'à ce que l'outil ne produise plus de scénarios.

Pour extraire les chemins les plus courts d'abord, nous proposons d'examiner les préconditions des opérations extraites à la même récursion et de calculer le niveau de rapprochement de ces opérations par rapport à l'état initial. Nous procédons ainsi dans la récursion suivante avec l'opération qui nous rapproche le plus de l'état initial.

Définition 20 (Calcul du rapprochement de l'initialisation).
Nous considérons une précondition d'une opération op comme la conjonction d'un ensemble de conditions c_i : $Pre(op) = \bigwedge_i c_i$, et nous désignons par $NB_{\neg c_i}(op)$ le nombre de conditions c_i qui se sont pas satisfaites par l'état initial :

$$NB_{\neg c_i}(op) = card(\{i | i \in \{1..n\} \wedge [Init]\neg c_i\})$$

Ainsi, nous considérons que l'opération qui nous rapproche le plus de l'état initial est l'opération qui a le nombre $NB_{\neg c_i}(op)$ le plus petit.

Par exemple, lors de la première récursion de notre algorithme pour l'extraction de scénarios d'attaque du SI de la bibliothèque, nous avons exhibé quatre opérations pour lesquelles nous calculons le niveau de rapprochement de l'état initial (figure 5.8) comme indiqué dans le tableau 6.4.

op	$\bigwedge_i c_i = Pre(op)$	$[Init]\neg c_i$	$NB_{\neg c_i}$
Member_AddLend (John, bo)	John ∈ Member card(Lend[{John}])<3 bo ∈ Book (John ↦ bo) ∉ Lend bo ∉ ran(Lend) bo ∉ dom(Reserve)	✗ ✗ ✗ ✗ ✗ ✓	1
Member_AddLend (Alice, bo)	Alice ∈ Member card(Lend[{Alice}])<3 bo ∈ Book (Alice ↦ bo) ∉ Lend bo ∉ ran(Lend) bo ∉ dom(Reserve)	✓ ✗ ✗ ✗ ✗ ✓	2
Member_TakeReservedBook (John, bo)	John ∈ Member bo ∈ Book bo ∉ ran(Lend) (bo ↦ John)∈Reserve card(Lend[{John}])<3	✗ ✗ ✗ ✓ ✗	1
Member_TakeReservedBook (Alice, bo)	Alice ∈ Member bo ∈ Book bo ∉ ran(Lend) (bo ↦ Alice)∈Reserve card(Lend[{Alice}])<3	✓ ✗ ✗ ✓ ✗	2

TABLEAU 6.4 – Exemple de calcul du niveau de rapprochement de l'état initial

Nous enchaînons ainsi la récursion suivante soit avec l'opération Member_AddLend(John, bo) ou avec l'opération Member_TakeReservedBook(John, bo). Grâce à cette technique, nous avons eu besoin d'extraire seulement 3 scénarios parmi les scénarios les plus courts pour corriger et sécuriser notre modèle. Par conséquent, la sollicitation du solveur de contraintes a été réduite à plus que 80%. En effet, en appliquant notre algorithme tel qu'il est présenté dans le chapitre 4, nous avons besoin de plus que 300 appels au solveur de contraintes pour extraire les comportements malicieux donnés dans le tableau 5.1 contre 52 appels seulement en utilisant une recherche en profondeur d'abord favorisant les opérations qui nous rapprochent de l'état initial.

En combinant cette technique avec la technique basée sur l'analyse syntaxique et la preuve présentée à la section 6.3.1, 20 appels au solveur de contraintes ont suffi pour l'extraction des scénarios nécessaires à la correction des failles de sécurité.

6.4 Etude de cas

Dans cette section, nous détaillons notre approche et nous appliquons notre outil sur l'exemple du SI de revues d'articles traité dans ZHANG et al. [2008], et nous montrons que notre approche est capable d'extraire les mêmes scénarios que ceux rapportés par les auteurs et de détecter différents types d'attaque faisant intervenir divers utilisateurs.

6.4.1 Modèle fonctionnel

Le diagramme de classes de la figure 6.5 représente le modèle fonctionnel du système. Il inclut les quatre entités suivantes : `Author`, `Reviewer`, `Paper` et `Review` qui représentent respectivement les auteurs des articles, la comité de programme de la conférence, les articles soumis ainsi que les évaluations des articles.

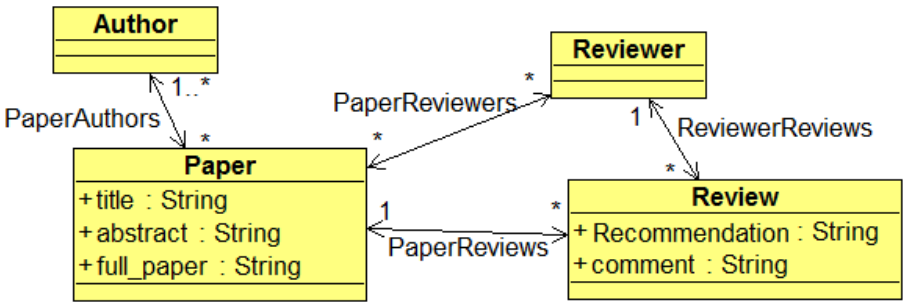


FIGURE 6.5 – Modèle fonctionnel du SI de revue des articles

Le modèle fonctionnel satisfait donc les contraintes suivantes :

— Un article doit être écrit par au moins un auteur et il peut être affecté à plusieurs rapporteurs du comité de programme pour être évalué.

— Chaque évaluation (*review*) est soumise par un seul rapporteur et ne concerne qu'un seul article.

— Un auteur a la possibilité de soumettre plusieurs articles,

— Pour chaque article, il ne peut y avoir qu'un seule évaluation (*review*) par rapporteur assigné à l'article.

Nous notons que les trois premières contraintes sont couvertes par les multiplicités spécifiées dans le diagramme de classes et par la traduction de ces dernières en B par la plate-forme B4MSecure, contrairement à la dernière contrainte qui requiert l'ajout manuel d'un invariant dans la machine B générée.

6.4.2 Modèle de sécurité

Le modèle de sécurité inclut les trois rôles suivants :

— `Author` : désignant les auteurs

— `Member` : désignant les membres du comité de programme

— `Chair` : désignant les présidents du comité de programme, sachant qu'un président du comité de programme peut être considéré comme un membre du comité, il possède alors tous les droits d'accès de ce dernier.

Nous donnons dans la figure 6.6 un exemple d'affectation des utilisateurs aux différents rôles. Nous considérons ainsi `John` et `Bob` comme des auteurs. `Bob` et `Paul` sont des membres du comité de programme et `Alice` est la présidente du comité de programme.

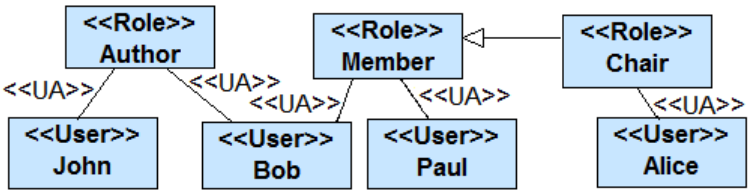


FIGURE 6.6 – Affectations des utilisateurs aux rôles

Nous modélisons dans la figure 6.7 les permissions assignées à chacun des rôles en utilisant la syntaxe secureUML.

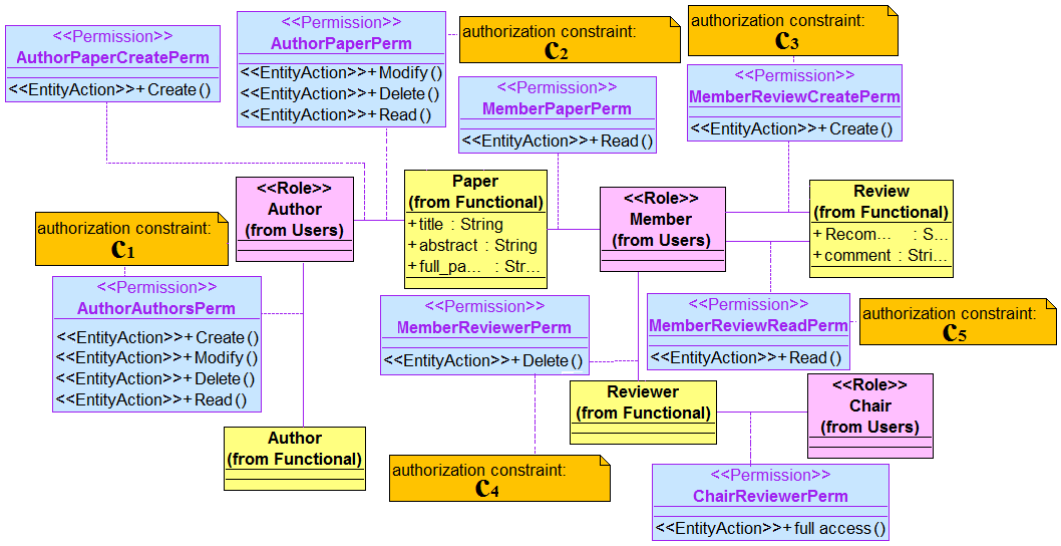


FIGURE 6.7 – Modèle fonctionnel du SI de revue d'articles

Nous donnons les définitions des contraintes d'autorisation dans le tableau 6.5.

c1	user = self
c2	user $\in$ PaperAuthors(self)
c3	user $\in$ PaperReviewers(PaperReviews(self))
c4	user = self and ReviewerReviews^{-1}(self) = $\emptyset$
c5	user $\notin$ PaperReviewers(PaperReviews(self)) or (ReviewerReviews^{-1}(user) $\cap$ PaperReviews^{-1}(PaperReviews(self)) $\neq \emptyset$)

tels que :

PaperAuthors	$\in$	Paper $\longleftrightarrow$ Author
PaperReviewers	$\in$	Paper $\longleftrightarrow$ Reviewer
PaperReviews	$\in$	Review $\longrightarrow$ Paper
ReviewerReviews	$\in$	Review $\longrightarrow$ Reviewer

TABLEAU 6.5 – Spécification des contraintes d'autorisation

Le modèle donné dans la figure 6.7 spécifie les règles de contrôle d'accès suivantes :

— **AuthorPaperCreatePerm** : Les auteurs peuvent ajouter de nouveaux articles dans le système.

— **AuthorPaperPerm** : Un auteur peut lire, modifier ou supprimer des articles si et seulement s'il satisfait la contrainte d'autorisation c_2. Ainsi, il ne peut accéder qu'à ses propres articles.

— **AuthorAuthorsPerm** : Un auteur peut créer, modifier, lire ou supprimer des instances de la classe `Author`. Mais comme la contrainte d'autorisation c_1 est attachée à cette permission, il ne peut accéder qu'à l'instance de son propre utilisateur.

— **MemberPaperPerm** : Un membre du comité de programme a le droit de lire tous les articles même ceux qui ne lui sont pas assignés en tant que rapporteur.

— **MemberReviewCreatePerm** : Un membre du comité de programme peut soumettre une note (*review*) à un article. Mais, étant donnée que cette permission est contrainte par c_3, il ne peut soumettre une note que pour les articles qui lui sont assignés à travers l'association `PaperReviewers`.

— **MemberReviewReadPerm** : Un membre du comité de programme peut lire les notes (*reviews*) des autres membres seulement s'il satisfait les deux conditions suivantes spécifiées par la contrainte d'autorisation c_5 :

  — il n'est pas assigné comme un rapporteur pour l'article en question, ou

  — il a déjà soumis sa note.

— **MemberReviewerPerm** : Un membre du comité de programme peut refuser son affectation en tant que rapporteur pour un article tant qu'il n'a pas encore soumis une note tel qu'il est spécifié par la contrainte d'autorisation c_4. Il peut, donc, supprimer l'association d'un article à son utilisateur à travers la relation `PaperReviewers`.

— **ChairReviewerPerm** : Un président du comité de programme a un access inconditionnel (*full access*) à toutes les entités de la classe `Reviewer`. Il peut donc ajouter, modifier, supprimer ou lire les données concernant les membres du comité de programme, comme il peut exécuter les opérations issues des associations `PaperReviewers` et `ReviewerReviews`. Ainsi, il peut affecter ou supprimer l'affectation d'un rapporteur à un article.

6.4.3 Extraction de scénarios d'attaque

Nous avons appliqué notre outil sur cet example [6] afin d'extraire des scénarios d'attaque visant à compromettre la confidentialité des notes attribuées par les rapporteurs aux articles.

Notre objectif est de vérifier s'il existe des scénarios d'attaque qui permettent à des rapporteurs malicieux de lire la note r_1 attribué par un collègue à un article p_1 avant de soumettre leur note.

Nous identifions ainsi deux types de victimes :

— Une première victime v_1 qui doit être un rapporteur ayant soumis la note r_1 pour l'article p_1. Les propriétés $R_{victime1}$ de cette victime peuvent alors être exprimées comme suit :

$$\begin{aligned} R_{victime1} \widehat{=} \quad & v_1 \in VICTIMES \wedge v_1 \in Reviewer \wedge r_1 \in Review \wedge p_1 \in Paper \wedge \\ & (r_1 \mapsto v_1) \in ReviewerReviews \wedge (r_1 \mapsto p_1) \in PaperReviews \wedge \\ & (p_1 \mapsto v_1) \in PaperReviewers \end{aligned}$$

— Une deuxième victime v_2 qui doit être un auteur ayant soumis l'article p_1. Nous exprimons les propriétés $R_{victime2}$ de cette victime comme suit :

$$R_{victime2} \widehat{=} v_2 \in VICTIMES \wedge v_2 \in Author \wedge p_1 \in Paper \wedge (p_1 \mapsto v_2) \in PaperAuthors$$

Quant aux attaquants, ils appartiennent à l'ensemble des rapporteurs assignés à l'article p_1 sachant qu'ils n'ont pas encore soumis de notes pour cet article. Les propriétés des attaquants $R_{attaquant}$ peuvent alors être exprimées comme suit :

$$\begin{aligned} R_{attaquant} \widehat{=} \quad & a \in ATTAQUANTS \wedge a \in Reviewer \wedge (p_1 \mapsto a) \in PaperReviewers \wedge \\ & (PaperReviews^{-1}[\{p_1\}] \cap ReviewerReviews^{-1}[\{a\}] = \emptyset) \end{aligned}$$

Si nous supposons que `Paul` est la première victime, et `John` est la deuxième victime, nous pouvons alors définir les ensembles des victimes et des attaquants comme suit :

```
DEFINITIONS
  VICTIMES == {Paul, John};
  ATTAQUANTS == USERS - VICTIMES
```

FIGURE 6.8 – Définition des victimes et des attaquants

Notre objectif est de vérifier s'il existe des scénarios d'attaque qui permettent à un utilisateur parmi l'ensemble des attaquants de lire r_1 avant de soumettre sa note.

6. Les spécifications B de l'exemple sont fournies à l'adresse http://genisis.forge.imag.fr/

Nous cherchons alors une attaque planifiée qui vise à exécuter l'opération de lecture du *review* r_1 : `Review_GetRecommendation(`r_1`)` puis l'opération de soumission d'un nouveau *review* : `Review_New(`r_2, p_1`)`.

Nous rappelons que :

— `Review_GetRecommendation` est une opération de lecture de la classe `Review` dont l'accès est conditionné par la contrainte d'autorisation c_5 afin d'interdire l'accès aux membres du comité de programme qui sont désignés comme des rapporteurs pour l'article et qui n'ont pas encore soumis leur *review*. Par conséquent, le premier état cible que nous cherchons à atteindre doit établir le prédicat $\mathsf{P}_{\mathsf{E}_1}$ qui satisfait à la fois *Pre*(`Review_GetRecommendation`) comme dans une attaque basique et la contrainte d'autorisation c_5 comme dans une attaque à contrainte. Ainsi :

$$
\begin{array}{rcl}
\mathsf{P}_{\mathsf{E}_1} & = & \mathit{Pre}(\texttt{Review\_GetRecommendation(r}_1\texttt{))} \wedge c_5 \\
& = & \texttt{r}_1 \in \texttt{Review} \wedge \\
& & \texttt{((currentUser} \notin \texttt{PaperReviewers[\{p}_1\texttt{\}])} \vee \\
& & \texttt{(ReviewerReviews}^{-1}\texttt{[\{currentUser\}]} \cap \\
& & \texttt{PaperReviews}^{-1}\texttt{[\{r}_1\texttt{\}]} \neq \emptyset\texttt{))}
\end{array}
$$

— `Review_New` est un constructeur de la classe `Review` dont l'accès est conditionné par la contrainte d'autorisation c_3. Ainsi, l'accès à cette opération est uniquement autorisé aux membres du comité de programme qui sont affectés à l'article comme étant des rapporteurs. Par conséquent, le deuxième état cible doit satisfaire $\mathsf{P}_{\mathsf{E}_2}$ tel que :

$$
\begin{array}{rcl}
\mathsf{P}_{\mathsf{E}_2} & = & \mathit{Pre}(\texttt{Review\_New(r}_2\texttt{,p}_1\texttt{))} \wedge c_3 \\
& = & \texttt{r}_2 \in \texttt{REVIEW} \wedge \texttt{r}_2 \notin \texttt{Review} \wedge \texttt{p}_1 \in \texttt{Paper} \wedge \\
& & \texttt{currentUser} \in \texttt{PaperReviewers[\{p}_1\texttt{\}]} \wedge \\
& & \texttt{(PaperReviews[\{p}_1\texttt{\}]} \cap \\
& & \texttt{ReviewerReviews}^{-1}\texttt{[\{currentUser\}]} = \emptyset
\end{array}
$$

L'attaque que nous cherchons à extraire couvre trois types d'attaques internes. En effet, elle met en œuvre une attaque planifiée qui implique à la fois une attaque à contrainte et une attaque basique.

Pour entamer notre recherche nous considérons l'état initial de la figure 6.9 qui est généré en résolvant le problème à contraintes suivant :

$$
\begin{array}{ll}
\{\mathit{Author}, \mathit{Reviewer}, \mathit{Paper}, \mathit{Review}, \mathit{PaperAuthors}, & \\
\mathit{PaperReviewers}, \mathit{PaperReviews}, \mathit{ReviewerReviews}| & \mathit{Inv} \wedge \\
& \exists \mathit{victime1} \cdot (R_{\mathit{victime1}}) \wedge \\
& \exists \mathit{victime2} \cdot (R_{\mathit{victime2}}) \wedge \\
& \forall \mathit{attaquant} \cdot (R_{\mathit{attaquant}} \Rightarrow \neg \mathsf{P}_{\mathsf{E}_1})\}
\end{array}
$$

```
INITIALISATION
    Author := {John}
  ‖ Reviewer := {Bob, Paul, Alice}
  ‖ Paper := {p1}
  ‖ Review := {r1}
  ‖ PaperAuthors := {(p1 ↦ John)}
  ‖ PaperReviewers := {(p1 ↦ Bob),(p1 ↦ Alice),(p1 ↦ Paul)}
  ‖ PaperReviews := {(r1 ↦ p1)}
  ‖ ReviewerReviews := {(r1 ↦ Paul)}
```

FIGURE 6.9 – Initialisation de la machine B

Dans cet état initial nous avons :

— un auteur `John` (une première victime) a soumis l'article p_1,

— trois rapporteurs sont désignés pour évaluer l'article : `Alice`, `Bob` et `Paul`,

— le rapporteur `Paul` (une deuxième victime) a soumis sa note (*review*) r_1 pour l'article p_1.

6.4.3.1 Extraction de comportements malicieux

Afin d'extraire les comportements malicieux menant aux états indésirables que nous avons définis, nous utilisons l'outil GenISIS après lui avoir fourni la spécification B du modèle fonctionnel ainsi que le fichier CCL décrivant les contraintes d'autorisation [7]. Ainsi, nous avons exhibé les deux séquences suivantes :

$$
\begin{aligned}
Q_{F_1} = < \; & \texttt{Reviewer\_RemovePaperReviewers(Alice, } p_1\texttt{)}, \\
& \texttt{Review\_GetRecommendation(}r_1\texttt{)}, \\
& \texttt{Reviewer\_AddPaperReviewer(Alice, } p_1\texttt{)}, \\
& \texttt{Review\_New(}r_2, p_1\texttt{)} >
\end{aligned}
$$

$$
\begin{aligned}
Q_{F_2} = < \; & \texttt{Reviewer\_RemovePaperReviewers(Bob, } p_1\texttt{)}, \\
& \texttt{Review\_GetRecommendation(}r_1\texttt{)}, \\
& \texttt{Reviewer\_AddPaperReviewer(Bob, } p_1\texttt{)}, \\
& \texttt{Review\_New(}r_2, p_1\texttt{)} >
\end{aligned}
$$

6.4.3.2 Identification de scénarios d'attaque

Les séquences fonctionnelles Q_{F_1} et Q_{F_2} révélant des comportements malicieux sont traduites au contrôleur CSP de la figure 6.10. Ce dernier sert à guider le model-cheker ProB pour l'extraction des scénarios d'attaque relatifs à ces comportements malicieux à partir de la spécification B du modèle de sécurité. Ainsi, nous avons exhibé six scénarios d'attaque dont deux sont des attaques

7. Les spécifications sont fournies à l'adresse http://genisis.forge.imag.fr/

avec collusion entre `Alice` et `Bob` (figure 6.11), une attaque faisant intervenir un seul utilisateur `Alice` (figure 6.12) et trois fausses attaques (figure 6.13)

```
Main =
   Connexion;secure_Reviewer_RemovePaperReviewers?Alice?p1 →
   Connexion;secure_Review_GetRecommendation?r1 →
   Connexion;secure_Reviewer_AddPaperReviewer?Alice?p1 →
   Connexion;secure_Review_New?r2?p1 →
   goal → STOP
   □
   Connexion;secure_Reviewer_RemovePaperReviewers?Bob?p1 →
   Connexion;secure_Review_GetRecommendation?r1 →
   Connexion;secure_Reviewer_AddPaperReviewer?Bob?p1 →
   Connexion;secure_Review_New?r2?p1 →
   goal → STOP
Connexion = Connect → SKIP  □ SKIP
```

FIGURE 6.10 – Traduction des comportements malicieux en un contrôleur CSP

Selon les deux premiers scénarios donnés dans la figure 6.11, `Alice` et `Bob` s'entraident pour l'accomplissement de l'attaque qui donne à `Bob` la possibilité de lire le *review* de `Paul` avant de soumettre le sien. En effet, `Bob` commence par renoncer à son affectation en tant que rapporteur de p_1 en effectuant l'opération `secure_Reviewer_RemovePaperReviewers` qui est autorisée à son rôle de membre du comité de programme. Cette même opération peut aussi être exécutée par `Alice` en tirant profit de son rôle de présidente du comité de programme. Ensuite, comme `Bob` n'est plus un rapporteur de l'article p_1 il peut lire les *review* de cet article en effectuant l'opération `secure_Review_GetRecommendation`. Finalement, `Alice` affecte `Bob` de nouveau comme un rapporteur pour l'article, ce qui lui permet de soumettre le review r_2 moyennant l'opération `secure_Review_New`.

Connect$(Bob, Member)$, *secure_Reviewer_RemovePaperReviewers*(Bob, p_1), *secure_Review_GetRecommendation*(r_1), **Connect**$(Alice, Chair)$, *secure_Reviewer_AddPaperReviewer*(Bob, p_1), **Connect**$(Bob, Member)$, *secure_Review_ReviewNew*(r_2, p_1)	**Connect**$(Alice, Chair)$, *secure_Reviewer_RemovePaperReviewers*(Bob, p_1), **Connect**$(Bob, Member)$, *secure_Review_GetRecommendation*(r_1), **Connect**$(Alice, Chair)$, *secure_Reviewer_AddPaperReviewer*(Bob, p_1), **Connect**$(Bob, Member)$, *secure_Review_ReviewNew*(r_2, p_1)

FIGURE 6.11 – Scénarios d'attaque avec collusion d'utilisateurs

Quant au scénario de la figure 6.12, il est exécuté seulement par `Alice` en abusant de son rôle de présidente du comité de programme. En effet, comme c'est elle qui désigne et retire les rapporteurs, elle peut se retirer de l'ensemble des rapporteurs de l'article p_1 pour avoir la permission d'accès en lecture au *review* r_1. Ensuite, elle peut s'affecter de nouveau en tant que rapporteur de l'article p_1 pour enfin soumettre son *review* r_2.

Connect$(Alice, Chair)$,
$secure\_Reviewer\_RemovePaperReviewers(Alice, p_1)$,
$secure\_Review\_GetRecommendation(r_1)$,
$secure\_Reviewer\_AddPaperReviewer(Alice, p_1)$,
$secure\_Review\_ReviewNew(r_2, p_1)$

FIGURE 6.12 – Scénario d'attaque impliquant un seul utilisateur

Une des solutions qui peut être proposée pour éviter de telles attaques est l'identification de différentes phases dans le processus de soumission et d'évaluation des articles de la conférence : une phase de soumission d'articles, une phase d'affectation des rapporteurs et une phase d'évaluation. Par conséquent, après la fin de la phase d'affectation des rapporteurs, il ne devrait pas être autorisé de supprimer ou d'affecter des rapporteurs. En outre, les rapporteurs ne peuvent pas soumettre leurs notes et leurs commentaires avant le début de la phase d'évaluation. Ou bien, nous pouvons interdire l'accès en lecture aux `reviews` par les membres du comité de programme tant que la phase d'évaluation n'est pas encore terminée. Cependant, étant donné que toutes les attaques signalées nécessitent la collusion du président du comité de programme, nous pouvons considérer que le système est assez sécurisé car le président est l'utilisateur le plus digne de confiance.

Outre ces trois attaques, le model-checker a exhibé d'autres traces d'exécution conformes aux comportements décrits par le contrôleur CSP (figure 6.13). Toutefois, ces séquences ne peuvent pas être considérées comme des scénarios d'attaque car l'utilisateur qui exécute l'opération de lecture `secure_Review_GetRecommendation` est différent de l'utilisateur qui soumet le *review* r_2. Pour cette raison, il est important que chaque scénario extrait soit examiné par l'analyste pour décider de sa pertinence. Nous pouvons également rajouter une contrainte sur les utilisateurs connectés pour éviter l'extraction de ces scénarios.

Connect$(Alice, Chair)$, $secure\_Reviewer\_RemovePaperReviewers(Bob, p_1)$, **Connect**(Bob, Member), $secure\_Review\_GetRecommendation(r_1)$, **Connect**$(Alice, Chair)$, $secure\_Reviewer\_AddPaperReviewer(Bob, p_1)$, $secure\_Review\_ReviewNew(r_2, p_1)$	**Connect**$(Alice, Chair)$, $secure\_Reviewer\_RemovePaperReviewers(Alice, p_1)$, $secure\_Review\_GetRecommendation(r_1)$, **Connect**$(Alice, Chair)$, $secure\_Reviewer\_AddPaperReviewer(Alice, p_1)$, **Connect**$(Bob, Member)$, $secure\_Review\_ReviewNew(r_2, p_1)$

Connect$(Bob, Chair)$, $secure\_Reviewer\_RemovePaperReviewers(Bob, p_1)$, $secure\_Review\_GetRecommendation(r_1)$, **Connect**$(Alice, Chair)$, $secure\_Reviewer\_AddPaperReviewer(Bob, p_1)$, $secure\_Review\_ReviewNew(r_2, p_1)$

FIGURE 6.13 – Fausses attaques

Les attaques sur ce modèle ont été déjà discutées dans ZHANG et al. [2008] où les auteurs ont utilisé une technique basée sur le model-checking pur pour les extraire. Il était également possible d'exhiber ces scénarios en utilisant le model-checker ProB sans guidance. Toutefois, ProB a essayé 633 transitions rien que pour atteindre l'état qui déclenche la première opération cible `Review_GetRecommendation(`r_1`)` exécutée par `Alice` et 969 transitions pour atteindre cette même opération en connectant l'utilisateur `Bob`. En contre partie, en le guidant par notre approche, ProB avait juste besoin de 47 transitions pour extraire le scénario d'attaque complet mettant en oeuvre la collusion entre `Alice` et `Bob`.

6.5 Conclusion

Dans ce chapitre, nous avons présenté l'outil GenISIS qui permet d'automatiser notre approche pour la détection de scénarios d'attaque ainsi que pour l'extraction de traces d'exécution vérifiant la propriété d'atteignabilité dans un SI. Nous avons aussi montré, à travers les différentes expérimentations, l'efficacité de l'outil comparé à des techniques similaires opérant dans le cadre de détection des attaques internes qui profitent de l'évolution dynamique de l'état fonctionnel du système.

Nous avons ensuite décrit les optimisations qui seront intégrées dans l'outil en vue de réduire les communications avec le solveur de contraintes. En effet, nous avons constaté que la sollicitation du solveur de contraintes est la phase la plus coûteuse qui a un impact considérable aussi bien sur le temps d'exécution que sur l'espace mémoire. Ainsi, nous avons proposé deux techniques l'une basée sur l'analyse syntaxique et la preuve et l'autre basée sur les heuristiques. Nos différentes études de cas ont montré que l'optimisation qui combine les deux techniques est très prometteuse et permet un gain qui peut aller jusqu'à la réduction de 80% d'appels au solveur de contraintes au total.

Finalement, nous avons présenté une étude de cas complète sur laquelle nous avons montré, une fois de plus, la capacité de notre outil pour l'extraction automatique de scénarios d'attaque et l'utilité de la guidance du model-checker que nous proposons dans notre approche.

Conclusion générale

« *Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.* »

Winston Churchill

L'omniprésence des systèmes informatiques et la maîtrise accrue des technologies par la plupart des utilisateurs incitent certains d'entre eux à vouloir profiter du système et à le contourner dès que l'occasion se présente pour servir leurs intérêts. De ce fait, la responsabilité des concepteurs des SI est devenue très lourde. En effet, il n'est plus question de concevoir un logiciel qui fonctionne correctement, mais, un logiciel incontournable par ses utilisateurs légitimes. Il ne s'agit donc pas de verrouiller davantage l'accès aux opérations non autorisées, mais plutôt de vérifier que le séquencement des opérations autorisées n'ouvre pas de brèches dans le système.

Notre contribution dans ce contexte vient compléter l'approche B4MSecure pour offrir une solution complète d'aide à la conception et à la validation des SI sécurisés (figure 6.14). Nous avons proposé, à ce titre, un mécanisme d'extraction de scénarios d'attaques internes à partir de spécifications formelles B qui sont à leur tour générées par la plate-forme B4MSecure à partir des modèles UML. Il s'agit, plus précisément, d'exhiber les séquences d'opérations pouvant être exécutées légitimement par des utilisateurs internes dans le but de faire évoluer l'état fonctionnel du système de façon malicieuse leur permettant de s'octroyer de nouveaux privilèges. Certaines de ces séquences peuvent être dues à une erreur de conception et serviront d'aide à la correction des modèles conceptuels, et d'autres peuvent être des scénarios d'exécution normaux qui méritent d'être contrôlés pour éviter qu'ils soient exécutés abusivement.

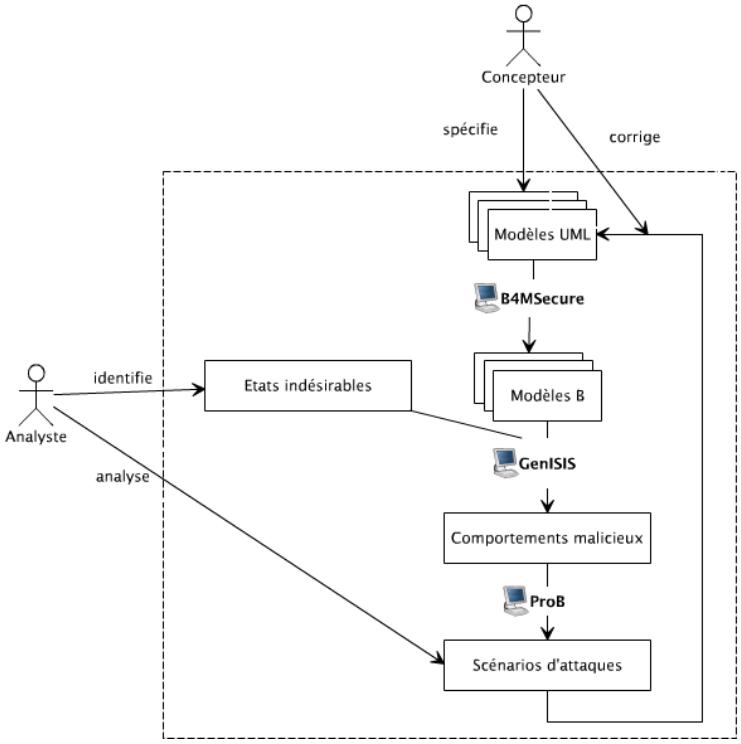


FIGURE 6.14 – Le processus de conception des SI sécurisés

Cette démarche outillée constitue une avancée dans le contexte de la conception et de la validation des SI et des modèles RBAC. En effet, en comparaison avec les approches existantes, notre approche est la première qui soit automatisable et qui étudie la validation des liens entre le modèle fonctionnel et le modèle de contrôle d'accès. Le problème des attaques internes est, aussi, un sujet d'actualité pour lequel très peu de travaux ont porté un intérêt particulier dès la phase de conception.

Bilan détaillé de nos contributions

Le problème de la détection des attaques internes visant à contourner les règles de contrôle d'accès est l'objectif principal qui a motivé notre investigation. Nous nous sommes intéressée, plus particulièrement, à l'identification de ces attaques dans le modèle RBAC étendu avec des règles spécifiques dépendant du contexte fonctionnel appelées **contraintes d'autorisation**. Nous avons proposé une formalisation de ces contraintes et nous avons montré à travers différents exemples qu'elles sont souvent à l'origine de failles de sécurité pouvant être exploitées par des utilisateurs internes malicieux.

Mais contrairement à l'approche proposée dans [QAMAR 2011], nous ne nous sommes pas limitée aux attaques ciblant ce type spécifique de règles de contrôle d'accès. En effet, notre approche couvre une plus large gamme de types d'attaques qui dépendent, d'une part, de la nature des règles de contrôle d'accès que l'attaquant cherche à contourner, et d'autre part, de la stratégie de ce der-

nier. Ainsi, selon le type de règle de contrôle d'accès, nous reconnaissons deux types d'attaques internes : *(i)* **les attaques basiques** qui sont perpétrées par des utilisateurs ne possédant pas les rôles requis pour accéder à la ressource cible, et *(ii)* **les attaques à contrainte** qui sont réalisées par des utilisateurs ne remplissant pas les conditions de la contrainte d'autorisation appliquée à la ressource cible. La stratégie suivie par l'attaquant, nous a également permis d'identifier deux autres types d'attaques : *(iii)* **les attaques planifiées** qui visent plusieurs cibles et qui tentent de réaliser un certain nombre d'opérations non autorisées dans un ordre bien déterminé, ainsi que *(iv)* **les attaques masquées** qui tentent de dissimuler les actes malicieux en remettant le système à son état d'origine après avoir réussi à accéder à des ressources non autorisées.

Nous identifions également des attaques plus complexes en combinant deux ou plusieurs types de ces attaques. En outre, nous avons l'avantage de pouvoir identifier des attaques accomplies par un seul utilisateur ou par une collusion de plusieurs utilisateurs.

Pour ce faire, nous avons proposé une approche en deux phases en tirant profit de la séparation des préoccupations fonctionnelles et de sécurité :

— Dans la première phase, nous exploitons les spécifications fonctionnelles ainsi qu'une description des contraintes d'autorisation et nous faisons abstraction des utilisateurs et de leurs affectations aux rôles. En effet, le but de cette phase consiste à identifier les comportements fonctionnels suspects qui, à partir d'un état normal, permettent d'atteindre un état indésirable.

— Dans la deuxième phase, nous étudions la faisabilité des comportements suspects dans le modèle de sécurité. Nous avons, ainsi, proposé d'utiliser une technique de guidance de model-checker par un contrôleur CSP en vue d'identifier les utilisateurs ainsi que les rôles pouvant réaliser de tels comportements.

Outre la guidance du model-checker dans son exploration par une approche CSP||B, cette approche par étapes a deux atouts majeurs : *(i)* D'une part, elle permet d'orienter la conception du modèle de sécurité en se basant sur les comportements malicieux (suspects) extraits à partir du modèle fonctionnel, et *(ii)* d'autre part, elle permet d'alléger l'analyse formelle de la propriété d'atteignabilité des états indésirables en faisant une abstraction sur les règles de sécurité lors de la première phase.

Par ailleurs, l'analyse formelle de l'atteignabilité des états indésirables que nous avons proposée nous a permis non seulement de disposer d'un mécanisme d'extraction de scénarios d'attaques, mais également d'avoir une approche plus générale pour la vérification de la propriété d'atteignabilité en B. En effet, les techniques proposées mettent en œuvre une recherche symbolique vers l'arrière fondée sur deux approches complémentaires : la preuve et la résolution de contraintes. Elles constituent une alternative intéressante aux techniques de model-checking pour une extraction automatisée des séquences d'opérations menant à une cible donnée.

Notre approche est également généralisable dans le contexte de la validation de la sécurité des

SI sur les deux plans suivants :

— Bien que notre travail a pour objectif la validation des modèles RBAC, la définition des comportements malicieux et des scénarios d'attaque que nous avons proposée rend notre approche applicable sur tout type de modèle de sécurité.

— Nous avons aussi montré, l'applicabilité de notre approche, pour la validation de scénarios de cas d'utilisation normaux et pour la génération de tests de sécurité.

Finalement, notre apport s'est concrétisé grâce à l'usage du model-checker ProB et à sa connexion avec notre outil GenISIS que nous avons expérimenté sur différents cas d'étude de tailles variées. Ces expérimentations, qu'elles soient dans le cadre de l'extraction de scénarios d'attaques ou dans le cadre plus général de la vérification de l'atteignabilité, ont donné des résultats très encourageants ce qui montre l'utilité et la viabilité de notre approche. Nous avons également proposé deux techniques pour l'amélioration de la performance de notre outil qui seront prochainement implantées dans GenISIS. Nous envisageons également de mener des expérimentations sur des études de cas concrètes lors de nos futurs travaux.

Perspectives

Les travaux présentés dans ce manuscrit de thèse, bien qu'ils donnent lieu à des contributions originales et qu'ils débouchent sur des résultats importants, présentent certaines limitations et ouvrent des axes de recherche complémentaires intéressants.

Outre les pistes d'amélioration sur le plan technique que nous avons déjà présentées dans le chapitre 6, plusieurs perspectives qui s'inscrivent dans le prolongement de nos travaux peuvent être envisagées. Nous proposons, principalement, des perspectives sur les quatre plans suivants :*(i)* Types d'attaques internes, *(ii)* Choix de l'état initial, *(iii)* Choix des ensembles, *(iv)* Génération automatique de tests.

Types d'attaques internes

En nous basant sur les types de règles de sécurité qu'un attaquant cherche à contourner ainsi que sur nos expérimentations, nous avons proposé une classification des attaques internes en quatre types différents. Cependant, il peut y avoir d'autres types d'attaques aussi intéressants et aussi dangereux qui peuvent être décelés à partir de nos spécifications. Parmi ces attaques nous évoquons ici les deux types d'attaque suivants :

- **Attaques par déni de service :** Les attaquants peuvent commettre des actes illicites, non seulement pour servir un intérêt personnel, mais également pour desservir l'intérêt de quelqu'un d'autre. Ainsi, nous pouvons imaginer des scénarios d'attaque dont l'objectif est de rendre inaccessible certaines opérations pour certains utilisateurs légitimes. Il s'agit dans ce cas de chercher les séquences d'opérations, qui à partir d'un état initial où l'opération cible op est accessible pour la victime, mène vers un état indésirable où cette opération devient

inaccessible (i.e $\mathsf{P_F} \hat{=} \neg Pre(op)$). Par exemple, dans le SI de la bibliothèque, afin d'empêcher son collègue `John` d'emprunter le livre `bo`, sur lequel portera un examen, l'utilisateur malicieux `Bob` peut le réserver infiniment. L'extraction de ce genre de comportement malicieux est important car il permet d'orienter la conception du SI. Par exemple, dans ce cas de figure nous pouvons imposer un nombre de jour limité pour une réservation.

- **Attaques accidentelles :** Dans notre investigation, nous ne nous sommes intéressée qu'aux attaques intentionnelles. Toutefois, il est possible qu'un utilisateur fasse une mauvaise manipulation par laquelle il dessert ses propres intérêts. Il s'agit, dans ce cas, de trouver des scénarios d'attaque tels que l'auteur de l'attaque (l'attaquant) est lui même la victime. Par exemple, dans le SI du réseau social Facebook [8], un utilisateur peut ouvrir l'accès à ses informations confidentielles accidentellement aux amis de l'un de ses amis en écrivant un commentaire sur le profil de ce dernier. Il serait, donc, utile d'extraire ce genre de scénario, soit pour mieux spécifier la sécurité du SI soit pour alerter les utilisateurs et pour tirer leur attention sur ce genre de manipulations.

Choix de l'état initial

Nous avons proposé dans le chapitre 5 une approche basée sur la résolution de contraintes pour la génération automatique de l'état initial à partir duquel nous démarrons notre recherche des comportements malicieux. Cette approche pose le problème de l'explosion combinatoire de nombre d'états qui peuvent être générés et qui satisfont les propriétés des attaquants et des victimes, surtout si les domaines des variables d'état sont non bornés. Nous avons également montré que le choix de l'état initial est important et peut impacter la recherche des scénarios d'attaque. A un niveau exploratoire, nous avons proposé des heuristiques de choix liées à la maximisation ou à la minimisation de nombre des éléments dans les variables ensemblistes. Cependant, nous pensons que d'autres heuristiques peuvent être définies pour améliorer l'approche proposée en prenant en compte tous les paramètres suivants : type d'attaque recherchée, les propriétés des attaquants et des victimes, l'état indésirable et les variables d'états.

Choix des ensembles

Pour l'extraction de traces d'exécution menant à une cible donnée, nous avons proposé deux approches complémentaires : la première est basée sur la preuve et la deuxième est basée sur la résolution de contraintes. Nous avons montré que cette dernière permet de palier aux problèmes de complexité posés par la première. Cependant, elle pose le problème de restriction des domaines de variables. Dans nos expérimentations, nous avons été amenée à faire des choix pour le nombre d'éléments constituant nos ensembles. Les tests de performance que nous avons effectués ont montré que ce nombre est très important et impacte considérablement la performance de notre outil aussi bien en terme de temps d'exécution qu'en terme d'occupation d'espace mémoire.

8. https://www.facebook.com

Outre ces problèmes de performance, le choix des ensembles peut parfois avoir des impacts sur les scénarios d'attaque trouvés. Par exemple, nous considérons le problème d'atteignabilité, illustré dans le chapitre 5, qui cherche à extraire les comportements malicieux permettant aux attaquants d'emprunter le livre `bo` à la place de la victime `Bob`. Les séquences d'opérations données dans le tableau 5.1 ont été extraites en considérant la définition suivante des ensembles :

```
SETS
  BOOK = {bo};
  MEMBER = {Alice, Bob, John}
```

Toutefois, il est possible d'extraire d'autres traces d'exécution en prenant plus que trois éléments dans l'ensemble `BOOK`. Par exemple, lorsqu'on a défini l'ensemble `BOOK` comme suit : `BOOK = {bo, bo1, bo2, bo3}`, nous avons extrait la séquence suivante :

```
Book_Free(bo);
Book_New(bo);
Member_AddLend(John, bo1);
Member_AddLend(John, bo2);
Member_AddLend(John, bo3);
Member_RemoveLend(John, bo1);
Member_AddLend(John, bo);
```

En effet, comme un membre n'a pas le droit d'emprunter plus que trois livres à la fois, il doit restituer l'un des 3 livres empruntés pour pouvoir prendre le livre `bo`.

Actuellement, pour avoir un maximum de confiance dans la sécurité du modèle étudié, nous faisons plusieurs expérimentations en augmentant à chaque fois le nombre des éléments dans les ensembles. Nous supposons qu'il n'y a plus de nouveaux comportements malicieux soit lorsque nous n'exhibons plus de nouvelles traces d'une expérimentation à une autre soit lorsque le temps d'exécution atteint la limite autorisée. Cependant, cette démarche est très fastidieuse si nous devons la mener pour chaque état indésirable. C'est pourquoi, nous pensons qu'une stratégie de choix des ensembles basée sur des hypothèses comme celle proposée dans [ANDONI et al. 2002] pourrait améliorer notre approche.

Génération automatique de tests

Nous avons évoqué dans l'annexe B la possibilité d'extension de l'approche que nous avons proposée pour l'extraction de scénarios d'attaque afin de valider, d'une part, les scénarios de cas d'utilisation normaux, et d'autre part, la correction des filtres de sécurité. Nous avons, ainsi, proposé une approche pour la génération de tests qui offre une couverture au niveau des opérations et au niveau des rôles. Cependant, ces tests ne couvrent ni les filtres de sécurité relatifs aux contraintes

d'autorisation, ni les règles de séparation de devoirs. Par conséquent, nous envisageons l'amélioration de cette approche pour prendre en compte tous les aspects de sécurité dans le modèle RBAC, comme nous prévoyons d'implanter ce mécanisme de génération automatique de tests dans l'outil GenISIS.

Bibliographie

ABDALLAH, A. E. et E. J. KHAYAT. 2006, «Formal Z Specifications of Several Flat Role-Based Access Control Models», dans 2006 30th Annual IEEE/NASA Software Engineering Workshop, ISSN 1550-6215, p. 282–292. 23

ABRIAL, J.-R. 1996a, The B-book : assigning programs to meanings, Cambridge University Press, ISBN 0-521-49619-5. 4, 21, 30, 38, 73

ABRIAL, J.-R. 1996b, «Extending B without Changing it (for Developing Distributed Systems)», dans First B conference, édité par H. Habrias, Putting into Practice Methods and Tools for Information System Design, IRIN. 40

ABRIAL, J.-R. et L. MUSSAT. 1998, «Introducing dynamic constraints in B», dans B'98 : Recent Advances in the Development and Use of the B Method, Second Int. B Conf., Montpellier, France, April 22-24, 1998, Proceedings, LNCS. 5, 73, 89

AHN, G.-J. et H. HU. 2007, «Towards realizing a formal rbac model in real systems», dans Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, SACMAT '07, ACM, New York, NY, USA, ISBN 978-1-59593-745-2, p. 215–224. 23

ANAND, S., C. S. PASAREANU et W. VISSER. 2009, «Symbolic execution with abstraction», Int. J. Softw. Tools Technol. Transf., vol. 11, nº 1, p. 53–67, ISSN 1433-2779. :2009

ANDONI, A., D. DANILIUC, S. KHURSHID et D. MARINOV. 2002, «Evaluating the "small scope hypothesis"», cahier de recherche, IN POPL '02 : PROCEEDINGS OF THE 29TH ACM SYMPOSIUM ON THE PRINCIPLES OF PROGRAMMING LANGUAGES. 148

BANDARA, A., H. SHINPEI, J. JURJENS, H. KAIYA, A. KUBO, R. LANEY, H. MOURATIDIS, A. NHLABATSI, B. NUSEIBEH, Y. TAHARA, T. TUN, H. WASHIZAKI, N. YOSHIOKA et Y. YU. 2010, «Security patterns : comparing modeling approaches», dans Software Engineering for Secure Systems : Industrial and Research Perspectives, édité par H. Mouratidis, IGI Global, Hershey, PA, p. 75–111. 125

BECKER, M. Y. et S. NANZ. 2010, «A logic for state-modifying authorization policies», ACM Trans. Inf. Syst. Secur., vol. 13, nº 3, ISSN 1094-9224. 26, 28

BEHM, P., P. BENOIT, A. FAIVRE et J. MARC MEYNADIER. 1999, «Meteor : A successful application of b in a large project», dans In [Wing et al, p. 369–387. 31

BELL, D. E. et L. J. LAPADULA. 1976, «Secure Computer System : Unified Exposition and MULTICS Interpretation», cahier de recherche ESD-TR-75-306, The MITRE Corporation. 4, 11

BELLOVIN, S. M. 2008, «The insider attack problem nature and scope», dans Insider Attack and Cyber Security - Beyond the Hacker, Advances in Information Security, vol. 39, édité par S. J. Stolfo, S. M. Bellovin, A. D. Keromytis, S. Hershkop, S. W. Smith et S. Sinclair, Springer, p. 1–4. 19

BENDISPOSTO, J., M. LEUSCHEL, O. LIGOT et M. SAMIA. 2008, «La validation de modèles event-b avec le plug-in prob pour rodin.», Technique et Science Informatiques, vol. 27, nº 8, p. 1065–1084. 40

BERTINO, E., P. A. BONATTI et E. FERRARI. 2001, «TRBAC : A Temporal Role-based Access Control Model», ACM Trans. Inf. Syst. Secur., vol. 4, nº 3, p. 191–233, ISSN 1094-9224. 15

BUTLER, M. J. et M. LEUSCHEL. 2005, «Combining CSP and B for specification and property verification», dans FM 2005 : Formal Methods, Int. Symposium of Formal Methods Europe, Newcastle, UK, July 18-22, 2005, Proceedings. 42

CHOUALI, S., J. JULLIAND, P. MASSON et F. BELLEGARDE. 2005, «Pltl-partitioned model checking for reactive systems under fairness assumptions», ACM Trans. Embedded Comput. Syst., vol. 4, nº 2, p. 267–301. 69

CLARKE, E. M. et E. A. EMERSON. 1982, Design and synthesis of synchronization skeletons using branching time temporal logic, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-39047-3, p. 52–71. 25

CLARKE, E. M., O. GRUMBERG et D. E. LONG. 1994, «Model checking and abstraction», ACM Trans. Program. Lang. Syst., vol. 16, nº 5, p. 1512–1542, ISSN 0164-0925. :1994

CLARKE, E. M., JR., O. GRUMBERG et D. A. PELED. 1999, Model Checking, MIT Press, Cambridge, MA, USA, ISBN 0-262-03270-8. 22

CLEARSWIFT. 2013, «The enemy within», Int. J. Inf. Sec. URL http://www.clearswift.com/sites/default/files/images/blog/enemy-within.pdf. 18

COOK, S. A. 1971, «The complexity of theorem-proving procedures», dans Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71, ACM, New York, NY, USA, p. 151–158, doi :10.1145/800157.805047. :1971

COUCHOT, J.-F., D. DÉHARBE, A. GIORGETTI et S. RANISE. 2004, «Barvey : Vérification automatique de consistance de machines abstraites B», dans AFADL'2004 - Session Outils, édité par J. Julliand. 39

COUSOT, P. et R. COUSOT. 2003, «Parsing as abstract interpretation of grammar semantics», Theor. Comput. Sci., vol. 290, nº 1, p. 531–544, ISSN 0304-3975. 21

CRAMPTON, J. et H. KHAMBHAMMETTU. 2008, «Delegation in role-based access control», Int. J. Inf. Sec., vol. 7, nº 2, p. 123–136. 15

D. BERT, M.-L. P. et N. STOULS. «GeneSyst : a tool to reason about behavioral aspects of B Event specifications. application to security properties», dans ZB 2005 : Formal Specification and Development in Z and B, 4th Int. Conf. of B and Z Users. 39, 73

DADEAU, F., J. LAMBOLEY, T. MOUTET et M.-L. POTET. 2008, «A verifiable conformance relationship between smart card applets and B security models», dans First International Conference on ASM, B and Z - ABZ'08, vol. 5238, édité par J. P. B. Egon Börger, Michael Butler et P. Boca, Springer Berlin / Heidelberg, London, United Kingdom, p. 237–250. 31

DAVID, R. et H. ALLA. 1992, Petri Nets and Grafcet : Tools for Modelling Discrete Event Systems, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, ISBN 013327537X. 21

DE MOURA, L. et N. BJØRNER. 2008, «Z3 : An efficient smt solver», dans Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg, ISBN 3-540-78799-2, 978-3-540-78799-0, p. 337–340. 42

DENNING, D. E. 1976, «A lattice model of secure information flow», Commun. ACM, vol. 19, nº 5, p. 236–243, ISSN 0001-0782. 4, 12

DIJKSTRA, E. W. 1997, A Discipline of Programming, 1re éd., Prentice Hall PTR, Upper Saddle River, NJ, USA, ISBN 013215871X. 34

DUTLE, A., C. A. MUÑOZ, A. NARKAWICZ et R. W. BUTLER. 2015, «Software validation via model animation», dans Tests and Proofs - 9th International Conference, TAP 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 22-24, 2015. Proceedings, Lecture Notes in Computer Science, vol. 9154, édité par J. C. Blanchette et N. Kosmatov, Springer, p. 92–108. 22

ENGINEERING, C. S. 2011, «Atelierb prouveur interactif - manuel de référence», cahier de recherche Version 4.0, France, FR. 39

FBI. 2010, «Fannie mae corporate intruder sentenced to over three years in prison for attempting to wipe out fannie mae financial data», . 18

FERNANDEZ-TORO, A. 2007, Management de la sécurité de l'information : implémentation ISO 27001 : mise en place d'un SMSI et audit de certification, Architecte logiciel, Eyrolles, ISBN 9782212122183. 1

FERRAIOLO, D. et R. KUHN. 1992, «Role-based access control», dans In 15th NIST-NCSC National Computer Security Conference, p. 554–563. 12

FERRAIOLO, D. F., R. SANDHU, S. GAVRILA, D. R. KUHN et R. CHANDRAMOULI. 2001, «Proposed nist standard for role-based access control», ACM Trans. Inf. Syst. Secur., vol. 4, nº 3, ISSN 1094-9224. xii, 12, 13, 14, 16

FISLER, K., S. KRISHNAMURTHI, L. A. MEYEROVICH et M. C. TSCHANTZ. 2005, «Verification and change-impact analysis of access-control policies», dans Proceedings of the 27th Int. Conf. on Software Engineering, ICSE '05, ACM, New York, NY, USA, ISBN 1-58113-963-2. 24

FRAPPIER, M., F. DIAGNE et A. MAMMAR. 2011, «Proving reachability in B using substitution refinement», Electr. Notes Theor. Comput. Sci., vol. 280, p. 47–56. 90, 91, 125

GERVAIS, F. 2006, Combinaison de spécifications formelles pour la modélisation des systèmes d'information, thèse de doctorat. :2006 :phdThesis

GROSLAMBERT, J. 2007, «Verification of LTL on B event systems», dans B 2007 : Formal Specification and Development in B, 7th International Conference of B Users, Besançon, France, January 17-19, 2007, Proceedings, p. 109–124. 90

HALLERSTEDE, S. 2003, Parallel Hardware Design in B, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-44880-8, p. 101–102. 31

HOARE, C. A. R. 1969, «An axiomatic basis for computer programming», Commun. ACM, vol. 12, nº 10, p. 576–580, ISSN 0001-0782. 34

HOARE, C. A. R. 1978, «Communicating sequential processes», Commun. ACM, vol. 21, nº 8, ISSN 0001-0782. 21, 43, 112

IBM. 2016, «A survey of the cyber security landscape», IBM X-Force 2016 Cyber Security Intelligence Index. URL http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=SEJ03320USEN. xii, 2, 3

IDANI, A. 2006, B/UML : Bridging the gap between B specifications and UML graphical descriptions to ease external validation of formal B developments, Theses, Université Joseph-Fourier - Grenoble I. URL https://tel.archives-ouvertes.fr/tel-00118718. 50

IDANI, A., M.-A. LABIADH et Y. LEDRU. 2010, «Infrastructure dirigée par les modèles pour une intégration adaptable et évolutive de UML et B», Ingénierie des Systèmes d'Information, vol. 15, nº 3, p. 87–112. 4

IDANI, A. et Y. LEDRU. 2015, «B for modeling secure information systems - the B4MSecure platform», dans Formal Methods and Software Engineering - 17th International Conference on Formal Engineering Methods, ICFEM 2015, Paris, France, November 3-5, 2015, Proceedings, p. 312–318. 4, 50

IDANI, A., Y. LEDRU et A. RADHOUANI. 2014, «Modélisation graphique et validation formelle de politiques RBAC en systèmes d'information. plateforme b4msecure», Ingénierie des Systèmes d'Information, vol. 19, nº 6, p. 33–61. 55

JACKSON, D. 2002, «Alloy : A lightweight object modelling notation», ACM Trans. Softw. Eng. Methodol., vol. 11, nº 2, p. 256–290, ISSN 1049-331X. 21

JENSEN, K., L. M. KRISTENSEN et L. WELLS. 2007, «Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems», Int. J. Softw. Tools Technol. Transf., vol. 9, nº 3, p. 213–254, ISSN 1433-2779. 26

JONES, C. B. 1990, Systematic Software Development Using VDM (2Nd Ed.), Prentice-Hall, Inc., Upper Saddle River, NJ, USA, ISBN 0-13-880733-7. 21

JULLIAND, J., P. MASSON et H. MOUNTASSIR. 1999, «Modular verification of dynamic properties for reactive systems», dans Integrated Formal Methods, Proceedings of the 1st International Conference on Integrated Formal Methods, IFM 99, York, UK, 28-29 June 1999, p. 89–108. 69

JÜRJENS, J. 2002, «UMLsec : Extending UML for Secure Systems Development», dans Proceedings of the 5th International Conference on The Unified Modeling Language, UML '02, Springer-Verlag, London, UK, UK, ISBN 3-540-44254-5, p. 412–425. 23

KALAM, A. A. E., S. BENFERHAT, A. MIÈGE, R. E. BAIDA, F. CUPPENS, C. SAUREL, P. BALBIANI, Y. DESWARTE et G. TROUESSIN. 2003, «Organization based access control», dans 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), 4-6 June 2003, Lake Como, Italy, p. 120. 15

KUHLMANN, M., K. SOHR et M. GOGOLLA. 2013, «Employing UML and OCL for designing and analysing role-based access control», Mathematical Structures in Computer Science, vol. 23, nº 4. 24

KULKARNI, D. et A. TRIPATHI. 2008, «Context-aware role-based access control in pervasive computing systems», dans Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, SACMAT '08, ACM, New York, NY, USA, ISBN 978-1-60558-129-3, p. 113–122. 15

LAMPORT, L. 2002, Specifying Systems : The TLA Language and Tools for Hardware and Software Engineers, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, ISBN 032114306X. 21

LAMPSON, B. W. 1974, «Protection», SIGOPS Oper. Syst. Rev., vol. 8, nº 1, p. 18–24, ISSN 0163-5980. 4, 11

LANET, J. L. et A. REQUET. 2000, «Formal proof of smart card applets correctness», dans Proceedings of the The International Conference on Smart Card Research and Applications, CARDIS '98, Springer-Verlag, London, UK, UK, ISBN 3-540-67923-5, p. 85–97. 31

LEDRU, Y., A. IDANI, J. MILHAU, N. QAMAR, R. L., J.-L. RICHIER et M.-A. LABIADH. 2014, «Validation of IS security policies featuring authorisation constraints», Int. Journal of Information System Modeling and Design (IJISMD). 5, 125

LEDRU, Y., A. IDANI et J. RICHIER. 2015, «Validation of a security policy by the test of its formal B specification - A case study», dans 3rd IEEE/ACM FME Workshop on Formal Methods in Software Engineering,FormaliSE 2015, Florence, Italy, May 18, 2015, p. 6–12. 4, 62, 63, 64, VIII, XIII

LEDRU, Y., N. QAMAR, A. IDANI, J.-L. RICHIER et M.-A. LABIADH. 2011, «Validation of security policies by the animation of z specifications», dans Proceedings of the 16th ACM Symposium on Access Control Models and Technologies, SACMAT '11, ACM, New York, NY, USA, ISBN 978-1-4503-0688-1. 25, 27, 28

LEUSCHEL, M. et M. BUTLER. 2003, «ProB : A Model Checker for B», dans FME 2003 : Formal Methods Europe, Lecture Notes in Computer Science, vol. 2805, Springer-Verlag. 39

LEUSCHEL, M. et D. SCHNEIDER. 2014, «Towards B as a high-level constraint modelling language - solving the jobs puzzle challenge», dans Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th Int. Conf., ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings. 42

LODDERSTEDT, T., D. A. BASIN et J. DOSER. 2002, «SecureUML : A UML-based modeling language for model-driven security», dans Proceedings of the 5th Int. Conf. on The Unified Modeling Language, UML '02, Springer-Verlag, London, UK, UK, ISBN 3-540-44254-5. 4, 16, 24

MAMMAR, A., M. FRAPPIER et F. DIAGNE. 2011, «A proof-based approach to verifying reachability properties», dans Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011, p. 1651–1657. 51, 90, 91, 125

MCMILLAN, K. L. 1993, Symbolic Model Checking, Kluwer Academic Publishers, Norwell, MA, USA, ISBN 0792393805. :1993

MILNER, R. 1982, A Calculus of Communicating Systems, Springer-Verlag New York, Inc., Secaucus, NJ, USA, ISBN 0387102353. 21

MILNER, R. 1999, Communicating and Mobile Systems : The Pi Calculus, Cambridge University Press, ISBN 9780521658690. 21

MONDAL, S. et S. SURAL. 2008, A Verification Framework for Temporal RBAC with Role Hierarchy (Short Paper), Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-89862-7, p. 140–147. 25, 28

MORGAN, C. 1990, Programming from Specifications, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, ISBN 0-13-726225-6. 90

MYERS, G. J. et C. SANDLER. 2004, The Art of Software Testing, John Wiley & Sons, ISBN 0471469122. :2004

NURSE, J. R. C., O. BUCKLEY, P. A. LEGG, M. GOLDSMITH, S. CREESE, G. R. T. WRIGHT et M. WHITTY. 2014, «Understanding insider threat : A framework for characterising attacks», dans Security and Privacy Workshops (SPW), 2014 IEEE, p. 214–228. 18

OMG. 2011, «OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1», cahier de recherche, Object Management Group. URL http://www.omg.org/spec/UML/2.4.1. 23

PLAGGE, D. et M. LEUSCHEL. 2010, «Seven at one stroke : LTL model checking for high-level specifications in B, Z, CSP, and more», STTT, vol. 12, nº 1, p. 9–21. 69

POUZANCRE, G. 2003, How to Diagnose a Modern Car with a Formal B Model ?, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-44880-8, p. 98–100. 31

QAMAR, M. N. 2011, Spécification et animation de modèles de conception de la sécurité avec Z, thèse de doctorat, Université de Grenoble. 4, 125, 144

RADHOUANI, A., A. IDANI, Y. LEDRU et N. B. RAJEB. 2015, «Symbolic search of insider attack scenarios from a formal information system modeling», T. Petri Nets and Other Models of Concurrency, vol. 10, p. 131–152. 125

RAKKAY, H. et H. BOUCHENEB. 2009, Security Analysis of Role Based Access Control Models Using Colored Petri Nets and CPNtools, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-642-01004-0, p. 149–176. 26, 28

REQUET, A. 2003, «A b model for ensuring soundness of a large subset of the java card virtual machine», Sci. Comput. Program., vol. 46, nº 3, p. 283–306, ISSN 0167-6423. 31

ROBINSON, K. 1997, The B method and the B toolkit, Springer Berlin Heidelberg, Berlin, Heidelberg, p. 576–580. 39

SCHAAD, A., J. MOFFETT et J. JACOB. 2001, «The role-based access control system of a european bank : A case study and discussion», dans Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies, SACMAT '01, ACM, New York, NY, USA, ISBN 1-58113-350-2, p. 3–9. 15

SCHAAD, A. et J. D. MOFFETT. 2002, «A lightweight approach to specification and analysis of role-based access control extensions», dans Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies, SACMAT '02, ACM, New York, NY, USA, ISBN 1-58113-496-7, p. 13–22. 23

SCHNEIDER, S. et H. TREHARNE. 2005, «CSP theorems for communicating B machines», Formal Asp. Comput., vol. 17, nº 4, p. 390–422. xii, 43

SHAFIQ, B., A. MASOOD, J. JOSHI et A. GHAFOOR. 2005, «A role-based access control policy verification framework for real-time systems», dans 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, ISSN 1530-1443, p. 13–20. 25, 28

SILOWASH, G., D. CAPPELLI, A. MOORE, R. TRZECIAK, T. SHIMEALL et L. FLYNN. 2012, «Common sense guide to mitigating insider threats», cahier de recherche CMU/SEI-2012-TR-012, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 19

SPIVEY, J. M. 1989, The Z Notation : A Reference Manual, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, ISBN 0-13-983768-X. 21

TEAM, C. I. T. 2013, «Unintentional insider threats : A foundational study», cahier de recherche CMU/SEI-2013-TN-022, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 18

THOMAS, R. K. 1997, «Team-based access control (tmac) : A primitive for applying role-based access controls in collaborative environments», dans Proceedings of the Second ACM Workshop on Role-based Access Control, RBAC '97, ACM, New York, NY, USA, ISBN 0-89791-985-8, p. 13–19. 15

TIMES, T. N. Y. 2008, «French Bank Says Rogue Trader Lost $7 Billion», . 18

TORLAK, E. et D. JACKSON. 2007, «Kodkod : A relational model finder», dans Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings, p. 632–647. 42

VAN HENTENRYCK, P. 1989, Constraint Satisfaction in Logic Programming, MIT Press, Cambridge, MA, USA, ISBN 0-262-08181-4. 42

VOISINET, J.-C., B. TATIBOUËT et A. HAMMAD. 2002, «jBTools : An experimental platform for the formal B method», dans Principles and Practice of Programming in Java (PPPJ'02), Trinity College, Dublin, Ireland, p. 137–140. 39, 122

WAINER, J., P. BARTHELMESS et A. KUMAR. 2003, «W-RBAC - A Workflow Security Model Incorporating Controlled Overriding of Constraints», International Journal of Cooperative Information Systems, vol. 12, p. 2003. 15

YU, L., R. FRANCE, I. RAY et S. GHOSH. 2009, «A rigorous approach to uncovering security policy violations in uml designs», dans Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS '09, IEEE Computer Society, Washington, DC, USA, ISBN 978-0-7695-3702-3, p. 126–135. 25, 28

YUAN, C., Y. HE, J. HE et Z. ZHOU. 2006, A Verifiable Formal Specification for RBAC Model with Constraints of Separation of Duty, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-49610-6, p. 196–210. 23

ZAO, J., H. WEE, J. CHU et D. JACKSON. 2003, «RBAC schema verification using lightweight formal model and constraint analysis», dans Proceedings of the 8th ACM Symposium on Access Control Models and Technologies, SACMAT '03, ACM. :rbacschema

ZHANG, N., M. RYAN et D. P. GUELEV. 2008, «Synthesising verified access control systems through model checking», Journal of Computer Security, vol. 16, n° 1. :journals/jcs/ZhangRG08

Annexe A

Spécifications du SI de la bibliothèque

A.1 Le modèle fonctionnel

M ACHINE *Functional*
SETS
$BOOK; MEMBER$

VARIABLES
$Book, Member, Lend, Reserve,$

INVARIANT
$Book \subseteq BOOK \wedge$
$Member \subseteq Memeber \wedge$
$Lend \in Member \longleftrightarrow Book \wedge \forall c_2 \cdot (c_2 \in dom(Lend) \Rightarrow card(Lend[\{c_2\}]) \leq 3) \wedge$
$Reserve \in Book \nrightarrow Member$

INITIALISATION
$Book := \emptyset \quad \| \; Member := \emptyset \quad \| \; Lend := \emptyset \quad \| \; Reserve := \emptyset$

OPERATIONS
$Member\_TakeReservedBook(Instance, ReservedBook) =$
PRE
$Instance \in Member \wedge ReservedBook \in Book \wedge$
$ReservedBook \notin ran(Lend) \wedge$
$(ReservedBook \mapsto Instance) \in Reserve \wedge$
$card(Lend[Instance]) < 3$
THEN
$Lend := Lend \cup \{(Instance \mapsto ReservedBook)\} \; \|$

$Reserve := Reserve - \{(ReservedBook \mapsto Instance)\}$

END;

$Member\_AddLend(Instance, Lend\_bookValues) =$

PRE

$Instance \in Member \wedge card(Lend[\{Instance\}]) < 3 \wedge$
$Lend\_bookValues \in Book \wedge$
$(Instance \mapsto Lend\_bookValues) \notin Lend \wedge$
$Lend\_bookValues \notin ran(Lend) \wedge$
$Lend\_bookValues \notin dom(Reserve)$

THEN

$Reserve := Reserve \cup \{(Instance \mapsto Lend\_bookValues)\}$

END;

$Member\_AddReserve(Instance, Reserve\_bookValues) =$

PRE

$Instance \in Member \wedge Reserve\_bookValues \in Book \wedge$
$(Reserve\_bookValues \mapsto Instance) \notin Reserve \wedge$
$Reserve\_bookValues \notin dom(Reserve) \wedge$
$(Instance \mapsto Reserve\_bookValues) \notin Lend$

THEN

$Reserve := Reserve \cup \{(Reserve\_bookValues \mapsto Instance)\}$

END;

$Member\_RemoveLend(Instance, Lend\_bookValues) =$

PRE

$Instance \in Member \wedge Lend\_bookValues \in Book \wedge$
$(Instance \mapsto Lend\_bookValues) \in Lend$

THEN

$Lend := Lend - \{(Instance \mapsto Lend\_bookValues)\}$

END;

$Member\_RemoveReserve(Instance, Reserve\_bookValues) =$

PRE

$Instance \in Member \wedge Reserve\_bookValues \in Book \wedge$
$(Reserve\_bookValues \mapsto Instance) \in Reserve$

THEN

$Reserve := Reserve - \{(Reserve\_bookValues \mapsto Instance)\}$

END;

$Member\_New(Instance) =$

PRE

$Instance \in MEMBER \wedge Instance \notin Member$

THEN

$Member := Member \cup \{Instance\}$

END;

$Member\_Free(Instance) =$

PRE

$Instance \in MEMBER \wedge Instance \in Member$

THEN

$Member := Member - \{Instance\}$

$Lend := Lend \lhd \{Instance\}$

$Reserve := Reserve \rhd \{Instance\}$

END;

$Book\_New(Instance) =$

PRE

$Instance \in BOOK \wedge Instance \notin Book$

THEN

$Book := Book \cup \{Instance\}$

END;

$Book\_Free(Instance) =$

PRE

$Instance \in BOOK \wedge Instance \in Book$

THEN

$Book := Book - \{Instance\}$

$Lend := Lend \rhd \{Instance\}$

$Reserve := Reserve \lhd \{Instance\}$

END;

$result \leftarrow Book\_GetLend(Instance) =$

PRE

$Instance \in Book \wedge Instance \in ran(Lend)$

THEN

$result := Lend^{-1}(Instance)$

END;

$result \leftarrow Member\_GetLend(Instance) =$

PRE
$Instance \in Member \wedge Instance \in dom(Lend)$
THEN
$result := Lend(Instance)$
END;

$result \leftarrow Member\_GetReserve(Instance) =$
PRE
$Instance \in Member \wedge Instance : ran(Reserve)$
THEN
$result := Reserve^{-1}(Instance)$
END;

$result \leftarrow Book\_GetReserve(Instance) =$
PRE
$Instance \in Book \wedge Instance \in dom(Reserve)$
THEN
$result := Reserve(Instance)$
END
END

A.2 Spécification de l'opération `addRoleSafe`

```
addRoleSafe(user,role) =
   PRE
       user ∈ USERS ∧ role ∈ ROLES ∧
       role ∉ (roleOf(user) ∪ getSuperRoles(roleOf(user))) ∧
       ∀ rs·(rs ∈ ℙ1(ROLES) ∧ rs ∈ dom(SSD_mutex) ⇒
            card(((closure1(Roles_Hierarchy)[roleOf(user)∪{role}]
                 ∪ roleOf(user) ∪ {role}) ∩ rs)) < SSD_mutex(rs))
   THEN
       roleOf := ({user} ⩤ roleOf) ∪
                 {(user ↦ ({role} ∪ roleOf(user)))}
   END;
```

Annexe B

Test et vérification des filtres de sécurité

« *To tell somebody that he is wrong is called criticism. To do so officially is called testing.* »

Gaurav Khurana

Sommaire

B.1 Vérification des scénarios de cas d'utilisation normaux VI

B.2 Vérification des filtres de sécurité VIII

B.2.1 Approche par l'extraction du préambule IX

B.2.2 Approche par la génération de l'état initial XII

B.3 Bilan XIII

L'EXTRACTION de scénarios d'attaque constitue une facette importante pour la validation de la sécurité d'un SI. Cependant, le test du modèle de sécurité reste important pour vérifier la conformité du modèle par rapport aux exigences. En effet, les tests permettent de vérifier si les règles de contrôle d'accès remplissent convenablement leurs objectifs qui consistent, d'une part, à interdire l'accès aux utilisateurs dont le rôle ne le permet pas, et d'autre part, à préserver ce droit pour les personnes légitimes.

Dans ce contexte, nous proposons, dans ce chapitre, d'appliquer l'approche que nous avons utilisée pour l'extraction de scénarios d'attaque afin d'automatiser le processus de vérification et ce sur les deux plans suivants :

— Vérification des scénarios de cas d'utilisation normaux

— Vérification des filtres de sécurité appliqués aux opérations fonctionnelles

B.1 Vérification des scénarios de cas d'utilisation normaux

Afin de s'assurer que les règles de contrôle d'accès n'empêchent pas la réalisation des scénarios fonctionnels spécifiés par les exigences fonctionnelles, nous cherchons s'il y a moyen de

les exécuter dans le modèle de sécurité. Par conséquent, nous vérifions s'il existe un utilisateur ou un ensemble d'utilisateurs possédant les rôles qui leur permettent d'effectuer les opérations du scénario fonctionnel.

Pour ce faire, nous utilisons la technique de guidance du model-checker par un contrôleur CSP que nous avons proposée pour l'identification des utilisateurs capables de réaliser un comportement malicieux. Ainsi, pour chaque scénario de cas d'utilisation fonctionnel $Q_F = < op_1, op_2, \ldots, op_n >$, nous cherchons s'il existe une séquence Q_S équivalente qui pourrait être exécutée dans le modèle de sécurité. Nous définissons donc un contrôleur CSP similaire à celui décrit dans la figure 5.11 avec lequel nous guidons le model-checker pour vérifier l'atteignabilité de la cible **goal**.

A ce niveau, nous ne nous soucions pas de la légitimité des utilisateurs qui sont capables de réaliser la séquence d'opérations fonctionnelle. Ainsi, nous pouvons conclure que la vérification s'est terminée avec succès du moment où **goal** est atteignable.

A titre d'exemple, nous prenons le scénario fonctionnel suivant qui permet l'emprunt d'un livre `bo` à un membre `Bob`

```
Book_New(bo) ;
Member_New(Bob) ;
Member_AddLend(Bob,bo) ;
```

Nous le traduisons en un contrôleur CSP (figure B.1) et nous vérifions la possibilité de sa réalisation dans le modèle de sécurité.

```
Test1      =  Connexion ; secure_Book_New?bo →
              Connexion ; secure_Member_New?Bob →
              Connexion ; secure_Member_AddLend?Bob?bo →
              goal → STOP
Connexion  =  Connect → SKIP  □
              SKIP
```

FIGURE B.1 – Contrôleur CSP pour la vérification d'un scénario fonctionnel

Ce calcul se termine avec succès. En effet, ProB confirme l'existence d'au moins une séquence d'opérations dans le modèle de sécurité (figure B.2) qui est conforme au comportement donné par le contrôleur CSP.

```
Connect(Alice, {Librarian}) ;
secure_Book_New(bo) ;
Connect(Bob, {MemberUser}) ;
secure_Member_New(Bob) ;
secure_Member_AddLend(Bob,bo) ;
```

FIGURE B.2 – Scénario produit par ProB à partir du modèle de sécurité

Cette séquence montre que les règles de contrôle d'accès appliquées au modèle fonctionnel n'empêchent pas la réalisation de ce cas d'utilisation normal. Elle montre également qu'il existe un moyen de le réaliser dans le modèle de sécurité.

B.2 Vérification des filtres de sécurité

Afin de tester si les filtres de sécurité agissent comme prévu ou pas, les auteurs de [LEDRU et al. 2015] définissent des tests positifs et des tests négatifs qui permettent de vérifier respectivement que les filtres de sécurité autorisent l'accès aux utilisateurs légitimes et l'interdisent pour les utilisateurs dont le rôle ne le permet pas. Dans le but de réaliser ces différents tests, ils structurent les appels d'opérations sécurisées selon les permissions qu'ils souhaitent tester en trois parties : *preamble*, *nominal/robustness* et *call*. La séquence "preamble" correspond à la séquence qui mène à un état permettant de tester les opérations concernées par la permission. La partie "nominal/robustness" permet de varier les utilisateurs ayant les rôles concernés par la permission. La séquence "nominal" correspond à un test positif, et la séquence "robustness" présente une légère variation par rapport au cas nominal qui doit mener à un échec (test négatif). Finalement, la partie "call" permet d'appeler effectivement l'opération à tester en vue de voir qu'elle peut effectivement être appelée si elle est autorisée.

Par exemple, si nous souhaitons tester la permission `MemberPerm` (figure 3.8) qui autorise l'exécution des opérations de la classe `Member` comme l'opération `Member_AddLend` par le rôle `MemberUser`, nous considérons les séquences d'opérations de la figure B.3.

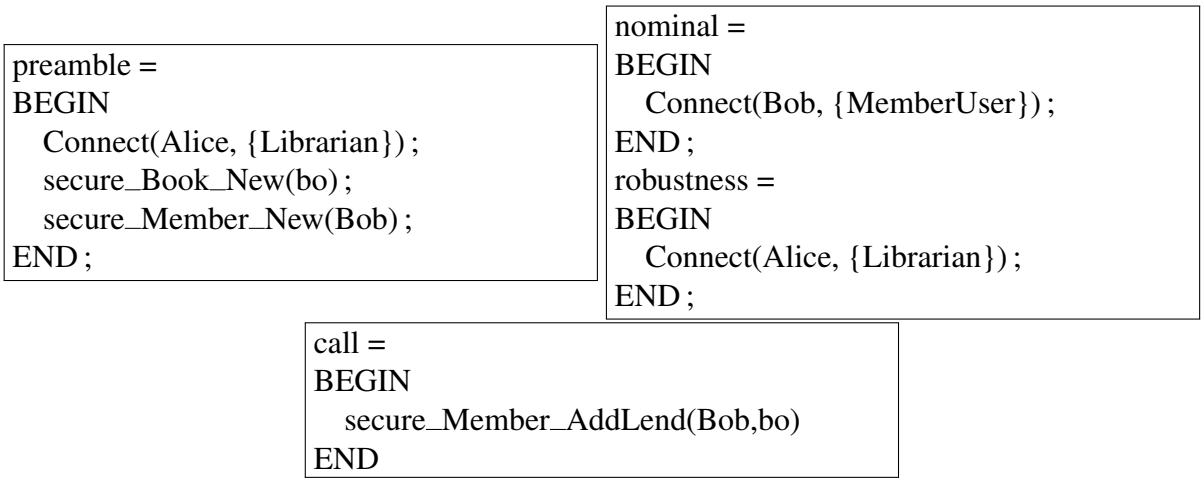


FIGURE B.3 – Structuration des tests

Le but des tests positifs est de vérifier qu'un utilisateur peut utiliser les opérations concernées par une permission s'il dispose d'un rôle autorisé et qu'il satisfait la précondition et la contrainte d'autorisation. Par exemple, la séquence « preamble ; nominal ; call » correspond à un test positif. Dans cette séquence, `Bob` emprunte le livre `bo`.

Les tests négatifs sont très proches des tests positifs, mais ils invalident l'un des éléments de la permission (utilisateur, rôle ou contrainte d'autorisation). Ceci permet de vérifier que chacune

de ces interdictions fonctionne. Contrairement au test positif, le test négatif ne peut pas s'exécuter entièrement. Par exemple, la séquence « preamble ; robustness ; call » correspond à un test négatif qui invalide le filtre du rôle. Dans cette séquence, `Alice` essaye d'emprunter le livre moyennant le rôle `Librarian`.

Afin de vérifier que les filtres de sécurité remplissent correctement leurs objectifs, ces tests devraient couvrir toutes les opérations sécurisées, tous les rôles possibles ainsi que tous les utilisateurs. Cependant, ceci peut engendrer une explosion combinatoire du nombre de tests à réaliser et peut prendre beaucoup de temps pour tout exécuter sur le modèle B, surtout en l'absence d'une approche d'automatisation des tests. Toutefois, en appliquant l'approche que nous avons proposée dans le chapitre précédent pour l'extraction de scénarios d'attaque, nous pouvons avoir une approche automatisable pour la vérification des filtres de contrôle d'accès. En effet, nous proposons deux techniques pour l'automatisation de cette activité de vérification :

— La première en utilisant notre algorithme de retour en arrière pour l'extraction du préambule du test.

— La deuxième en ayant recours à la génération de l'état initial.

B.2.1 Approche par l'extraction du préambule

Nous utilisons notre algorithme de retour en arrière dans le but d'extraire pour chaque opération op_{test} à tester la séquence d'opérations qui mène à un état S_{P}^{F} où op_{test} devient déclenchable (i.e. $\mathsf{P}_F \hat{=} Pre(op_{test})$) Cette séquence constitue le "preamble" du cas de test.

Ensuite, nous ajoutons l'opération op_{test} au préambule obtenu et nous utilisons notre méthode de guidance du model-checker pour identifier les couples (utilisateur, rôle) pouvant réaliser ces séquences d'opérations. Nous nous intéressons particulièrement aux utilisateurs pouvant accéder à l'opération op_{test}. Ainsi, afin de distinguer les utilisateurs légitimes des utilisateurs non légitimes nous définissons les deux variantes suivantes de l'opération de connexion :

— L'opération `Connect_Attacker` qui connecte un utilisateur malicieux ne possédant pas un rôle lui permettant d'accéder à l'opération op_{test}.

  Cette nouvelle variante de l'opération de connexion cherche donc à connecter un utilisateur malicieux appartenant à l'ensemble des attaquants **A** défini comme suit :

$$\mathbf{A} = \{u | u \in USERS \wedge ((\forall r \cdot (r \in roleOf(u) \Rightarrow op_{test} \notin isPermitted[r])\}$$

— L'opération `Connect_LegitimateUser` qui permet de connecter un utilisateur légitime appartenant à l'ensemble **V**. Comme l'ensemble **A** contient tous les utilisateurs non légitimes, nous pouvons exprimer l'ensemble **V** comme suit :

$$\mathbf{V} = USERS - \mathbf{A}.$$

Au final, nous avons le modèle CSP donné dans la figure B.4, tel que la séquence $< op_1, op_2, \ldots, op_n >$ constitue le préambule du test extrait par l'algorithme de retour en arrière.

```
Preamble      =  Connexion ; op1 →
                 Connexion ; op2 →
                 ... →
                 Connexion ; opn → SKIP

Connexion     =  Connect → SKIP  □
                 SKIP

Test_positif  =  Preamble ; Connect_LegitimateUser → op_test
                 →goal→STOP

Test_negatif  =  Preamble ; Connect_Attacker → op_test
                 →goal→STOP
```

FIGURE B.4 – Contrôleur CSP pour la vérification des filtres de sécurité

Par exemple, pour tester l'opération `Member_AddLend(Bob, bo)` nous avons pu extraire en appliquant notre algorithme de retour en arrière la séquence fonctionnelle représentant le préambule du test que nous donnons à la figure B.5.

```
Book_New(bo) ;
Member_New(Bob) ;
```

FIGURE B.5 – Préambule de test de l'opération `Member_AddLend(Bob, bo)`

Cette séquence permet d'atteindre l'état fonctionnel S_{P}^{F} à partir de l'état initial présenté à la figure B.6 tel que :

$$\begin{aligned}\mathsf{P}_F \hat{=}\ & Pre(Member\_AddLend(Bob, bo)) \wedge \\ & Bob \in Member \wedge \\ & card(Lend[Bob]) < 3 \wedge \\ & bo \in Book \wedge \\ & (Bob \mapsto bo) \notin Lend \wedge \\ & bo \notin ran(Lend) \wedge \\ & bo \notin dom(Reserve)\end{aligned}$$

```
INITIALISATION
    Book := {}
  || Member := {}
  || Lend := {}
  || Reserve := {}
```

FIGURE B.6 – Initialisation de la machine B

Nous définissons l'ensemble des utilisateurs non légitimes **A** comme suit :

$$\begin{aligned}\mathbf{A} = \{u \mid\ & u \in USERS \wedge \\ & \text{/* Non respect du rôle */} \\ & ((\forall r \cdot (r \in roleOf(u) \Rightarrow Member\_AddLend \notin isPermitted[r])\}\end{aligned}$$

Cet ensemble, à partir duquel l'opération `Connect_Attacker` prend les valeurs des utilisateurs, est alors évalué comme suit :

$$\mathbf{A} = \{Alice, John\}$$

Ce qui nous permet de déduire l'ensemble des utilisateurs légitimes à partir duquel l'opération `Connect_LegitimateUser` prend les valeurs des utilisateurs :

$$\mathbf{V} = USERS - \mathbf{A} = \{Bob\}\ .$$

Nous utilisons ensuite le contrôleur CSP de la figure B.7 pour conduire les tests positifs et négatifs :

```
Preamble      =  Connexion ; Book_New?bo →
                 Connexion ; Member_New?Bob → SKIP

Connexion     =  Connect → SKIP  □
                 SKIP

Test_positif  =  Preamble ; Connect_LegitimateUser →
                 Member_AddLend?Bob?bo →goal→STOP

Test_negatif  =  Preamble ; Connect_Attacker →
                 Member_AddLend?Bob?bo →goal→STOP
```

FIGURE B.7 – Exemple d'un contrôleur CSP pour la vérification des filtres de sécurité

Ce contrôleur CSP guidera le model-checker afin d'extraire les scénarios conformes à ces traces d'exécution à partir du modèle de sécurité. Pour ce cas de figure, ProB échoue dans l'extraction de séquences conformes au test négatif. Toutefois, il réussit à identifier des séquences vérifiant les traces du test positif. En effet, ces traces d'exécution ne peuvent être réalisées dans le modèle de sécurité qu'à travers les deux séquences données à la figure B.8. Ainsi, nous pouvons conclure que les filtres de sécurité appliqués à cette opération sont correctement spécifiés : ils permettent des exécutions du test positif et bloquent les exécutions du test négatif.

Connect(Alice, {Librarian}), *secure_Book_New*(*bo*), **Connect(Bob, {MemberUser})**, *secure_Member_New*(*Bob*), **Connect_LegitimateUser(Bob, {MemberUser})**, *secure_Member_AddLend*(*Bob*, *bo*)
Connect(Alice, {Librarian}), *secure_Book_New*(*bo*), *secure_Member_New*(*Bob*), **Connect_LegitimateUser(Bob, {MemberUser})**, *secure_Member_AddLend*(*Bob*, *bo*)

FIGURE B.8 – Scénarios de sécurité générés par ProB

B.2.2 Approche par la génération de l'état initial

Étant donné que le but des tests consiste à identifier les utilisateurs et les rôles pouvant accéder à chacune des opérations testées, il n'est pas nécessaire de déterminer les opérations de connexion permettant l'exécution des opérations constituant le préambule. Par conséquent, à partir d'un état initial où l'opération op_{test} est déclenchable dans le modèle fonctionnel, nous cherchons les opérations de connexion qui autorisent l'exécution de l'opération dans le modèle de sécurité.

Ainsi, nous proposons de générer un état initial qui satisfait la précondition de l'opération op_{test}. A partir de cet état initial, nous cherchons à extraire les couples (utilisateur, rôle) des opérations `Connect_LegitimateUser` et `Connect_Attacker` en utilisant un model-checker guidé par le contrôleur CSP suivant :

```
Test_positif  =  Connect_LegitimateUser → op_test
                 →goal→STOP

Test_negatif  =  Connect_Attacker → op_test
                 →goal→STOP
```

FIGURE B.9 – Contrôleur CSP pour la vérification des filtres de sécurité à partir d'un état initial déclanchant l'opération à tester

L'état initial est généré en utilisant un solveur de contraintes pour trouver une valuation aux variables d'état satisfaisant le problème suivant :

$$\{var | Inv \wedge Pre(op_{test})\}$$

Pour notre exemple, l'état initial doit satisfaire les contraintes suivantes :

$$
\begin{aligned}
\{Book, Member, Lend, Reserve \mid (\quad & /*Inv*/ \\
& Book \subseteq BOOK \wedge \\
& Member \subseteq MEMBER \wedge \\
& Lend \in Member \leftrightarrow Book \wedge \\
& Reserve \in Book \nrightarrow Member \wedge \\
& \forall c_2 \cdot (c_2 \in dom(Lend) \Rightarrow card(Lend[\{c_2\}]) \leq 3) \wedge \\
& /*Pre(Member\_AddLend(Bob,bo)*/ \\
& Bob \in Member \wedge card(Lend[\{Bob\}]) < 3 \wedge \\
& bo \in Book \wedge (Bob \mapsto bo) \notin Lend \wedge \\
& bo \notin ran(Lend) \wedge bo \notin dom(Reserve) \quad)
\end{aligned}
$$

Notre objectif étant d'avoir un état initial qui garantit la déclenchabilité de l'opération à tester, nous prenons l'état initial minimal qui satisfait ces contraintes. Par conséquent, nous considérons l'état initial de la figure B.10. A partir de cet état initial, nous utilisons le model-checker guidé par le contrôleur CSP de la figure B.9 pour identifier les opérations de connexion donnant accès à cette opération. Pour ce cas de figure, ProB réussie à extraire une seule opération de connexion légitime : `Connect_LegitimateUser(Bob, {MemberUser})` permettant d'accéder à l'opération `Member_AddLend(Bob,bo)`.

```
INITIALISATION
    Book := { bo }
  || Member := { Bob }
  || Lend := {}
  || Reserve := {}
```

FIGURE B.10 – Initialisation généré par ProB

B.3 Bilan

En comparaison par rapport aux techniques de tests proposées dans [LEDRU et al. 2015], notre approche permet de franchir un pas vers une génération automatique des scénarios de tests. Elle offre également une bonne couverture au niveau des permissions et des rôles. En effet, nous proposons de générer pour chaque opération fonctionnelle un contrôleur CSP décrivant la trace d'exécution d'un test positif et la trace d'exécution d'un test négatif. Ce contrôleur CSP guide un model-checker afin de chercher s'il y a des utilisateurs non légitimes pouvant réaliser le test négatif. Le model-checker cherche également la possibilité d'exécution du test positif en connectant un utilisateur parmi l'ensemble des utilisateurs légitimes.

Cependant, pour éviter l'explosion combinatoire du nombre de tests générés nous nous contentons de couvrir l'ensemble des opérations en considérant pour chaque opération un nombre de valuations de paramètres limité. Nous pensons que l'échantillonnage des paramètres des opéra-

tions est suffisant pour offrir une garantie satisfaisante. En effet, notre objectif est avant tout le test des filtres de sécurité et non pas le test des opérations fonctionnelles.

Annexe C

Glossaire

ANSI *American National Standard Institute*. 12

CSP *Communicating Sequential Process*. 21, 43, 44, 46–48, 113–115, 117, 118, 147, VII, X–XIII

DAC Contrôle d'Accès Discrétionnaire – ou *Discretionnary Access Control*. 4

DSD Séparation Dynamique des Devoirs – ou *Dynamic Separation of Duty*. 12, 15, 59

MAC Contrôle d'Accès Obligatoire – ou *Mandatory Access Control*. 4

OCL *Object Constraint Language*. 24, 25

RBAC Contrôle d'Accès Basé sur les Rôles – ou *Role Based Access Control*. 4, 7, 11, 12, 14–16, 19, 23–27, 29, 50, 51, 55, 95, 101, 146, 148, 151

SI Système d'Information. i, 1–6, 8, 11, 12, 15, 17–21, 27–29, 37, 48, 50, 51, 57, 61–63, 65, 66, 68–71, 83, 84, 91, 95, 99, 101, 102, 107, 111, 117–124, 127–129, 131, 134–136, 143, 145, 146, 148, 149, VI

SSD Séparation Statique des devoirs – ou *Static Separation of Duty*. 12, 14, 15, 59, 61, 112

STS Système de Transitions Symbolique. 40, 41

UML *Unified Modeling Langage*. 4, 24, 28, 50–52, 54, 145

V&V Vérification et Validation. 2, 4, 7, 21–24, 27, 29–31, 39, 42, 48