La composition des protocoles de sécurité avec la méthode B événementielle

Nazim Benaissa

Nancy-Université
Université Henri Poincaré

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

➢ Contact SCD Nancy 1 : theses.sciences@scd.uhp-nancy.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

Nancy-Université
Université Henri Poincaré

Université Henri Poincaré, Nancy 1

La composition des protocoles de sécurité avec la méthode B événementielle

Thèse

présentée le 28 mai 2010 en vue d'obtenir le grade de
Docteur, spécialité « informatique »

par

Nazim Benaissa

Composition du jury

Rapporteurs :	
Jean-Paul BODEVEIX	Professeur de l'IRIT Toulouse
Jean Goubault-Larrecq	Professeur de l'ENS Cachan
Examinateurs :	
Jean-Raymond Abrial	Professeur de l'ETH Zürich
Egon Börger	Professeur de l'Université de Pise
Dominique Cansell	Maître de conférences HDR de l'Université Paul Verlaine de Metz
Dominique Méry	Professeur de l'Université Henri Poincaré de Nancy - Directeur de thèse
Guy Perrier	Professeur de l'Université de Nancy 2

École doctorale IAEM DFD informatique UFR STMIA
Laboratoire Lorrain de Recherche en Informatique et ses Applications

Remerciements

Je tiens tout d'abord à remercier M. Jean Goubault-Larrecq ainsi que M. Jean-Paul BODEVEIX d'avoir bien voulu accepter d'être les rapporteurs de ma thèse. Je remercie également l'ensemble des membres de mon jury pour m'avoir fait l'honneur d'assister à ma soutenance.

Je remercie tout particulièrement mon directeur de thèse, M. Dominique Méry, pour m'avoir aidé, orienté et conseillé tout au long de mon Master et, par la suite, durant mes années de thèses. Je tiens également à remercier M. Dominique Cansell pour les nombreux conseils qu'il m'a prodigué durant mon apprentissage de la méthode B.

Je remercie également l'ensemble des membres de l'équipe MOSEL sans qui cette thèse ne serait pas ce qu'elle est.

Enfin, mes pensées les plus affectueuses vont vers mes parents, à qui je dois tout, ainsi qu'à mes frères sans oublier ma très chère sœur.

Table des matières

Chapitre 1

Introduction

De nos jours, la présence des réseaux à grande échelle dans notre société bouleverse nos habitudes avec l'apparition de nouveaux services distants. Ceci engendre une large interaction entre différentes parties connectées à un réseau et qui ne savent pas grand chose les unes des autres. L'utilisation de ces réseaux implique un transfert d'informations entre les différentes parties distantes, un transfert qui rend ces informations visibles à toute personne connectée à ce réseau, ceci peut poser problème dans le cas où l'on cherche à satisfaire des exigences de sécurité relatives aux informations transférées telles que la confidentialité. Des protocoles de sécurité ont été proposés pour tenter de répondre à ces exigences. Un protocole de sécurité est un échange de messages entre plusieurs entités à travers un réseau selon une convention donnée. Chaque protocole est censé garantir un ensemble de propriétés particulières, mais la question cruciale est d'être sûr que ces protocoles garantissent réellement les propriétés désirées. Afin de répondre à cette question, diverses méthodes ont été proposées, ce sujet a constitué et constitue toujours un domaine de recherche qui fait l'objet de beaucoup d'intérêt de la part des équipes de chercheurs.

Les méthodes formelles sont un moyen qui a été très largement utilisé pour montrer qu'un protocole satisfait à des exigences particulières. Les méthodes formelles sont des techniques permettant de démontrer qu'un système donné satisfait à des exigences particulières contenues dans une spécification, elles reposent sur un raisonnement rigoureux fondé sur la logique mathématique. Ces spécifications sont un cahier des charges décrivant ce que l'on attend de notre système, elles sont fondées sur un langage formel. Le challenge consiste, par la suite, à vérifier que l'implémentation du système satisfait aux exigences contenues dans la spécification, ce processus est connu sous le nom de «vérification». Les méthodes formelles utilisent différentes techniques de vérification parmi lesquelles : le model-checking et la démonstration de théorème.

Une façon de montrer que l'implémentation d'un système satisfait aux exigences de la spécification consiste à démontrer que le modèle de l'implémentation *raffine* le modèle de la spécification. Le raffinement signifie que les comportements concrets modélisés dans l'implémentation du système simulent les comportements abstraits contenus dans la spécification. Cette façon de faire permet d'obtenir des programmes *corrects par construction*. Cette notion de raffinement sera au centre de nos travaux dans cette thèse dans laquelle nous utilisons une méthode formelle nommée B événementielle qui permet d'utiliser plusieurs raf-

finements successifs pour vérifier qu'un système satisfait aux exigences d'une spécification donnée.

La méthode B événementielle utilise la logique du premier ordre munie de la théorie des ensembles. Dans un modèle B événementiel, l'état du système est modélisé par un ensemble de variables d'états dont la valeur évolue avec le déclenchement des événements du système dont les *action* modifient les valeurs des variables si une condition particulière appelée *garde* est satisfaite. Dans le modèle le plus abstrait, les propriétés désirées du système sont exprimées dans un prédicat appelé *invariant*, il faut prouver la consistance de cet invariant par rapport aux événements du système par une preuve. Un invariant dit de *collage* est ensuite défini pour chaque étape de raffinement pour lequel il faut également prouver la consistance. La méthode B événementielle peut être comparée aux *actions systems* de Back [14], à la méthode TLA^+ [58], aux programmes UNITY [35] ou encore aux ASM [27].

Nous avons utilisé la méthode B événementielle pour la vérification des protocoles de sécurité. La nature des services distants proposés de nos jours est très diversifiée, il en est de même pour les exigences de sécurité, en conséquence, les protocoles de sécurité sont de plus en plus complexes et plusieurs protocoles de structures très différentes peuvent cohabiter dans un même environnement et peuvent interagir entre eux avec de fâcheuses conséquences sur les propriétés de sûreté qu'ils sont censés préserver, d'où la nécessité de trouver des méthodes capables de traiter ce genre de situations. La méthode B événementielle a beaucoup mûri depuis son introduction et le raffinement est utilisé de différentes manières notamment en ce qui concerne la décomposition et la composition des modèles. Nous avons utilisé des techniques de décomposition et de composition de modèles B événementiels pour traiter le problème de la composition des protocoles de sécurité.

1.1 La composition des systèmes

Devant la complexité croissante des systèmes de taille réelle, il est indispensable de fournir des outils capables de nous aider à faciliter la spécification et la vérification de ce genre de système. Le raffinement est une première réponse à la problématique de la complexité des systèmes dans la mesure où il nous permet d'aborder le problème étape par étape sur plusieurs niveaux d'abstraction. Une deuxième manière de répondre aux problèmes de la complexité est de raisonner séparément sur de plus petites parties du système avant de les composer pour obtenir le système complet. Ces deux approches peuvent être utilisées simultanément pour obtenir de puissantes techniques de composition de systèmes.

Parmi les premiers travaux menés dans le cadre de la composition dans les méthodes formelles on peut citer les travaux de Abadi et Lamport [6] et de Jones [56]. Abadi et Lamport définissent deux approches possibles de composition, la première est l'approche «*top-down*», appelée également *décomposition*. Dans l'approche *top-down*, le système est décomposé en plusieurs composants. Le contexte (ou environnement) de chacun des composants est connu, il s'agit des autres composants du système, on doit donc définir pour chaque composant les propriétés nécessaires de son contexte et montrer ensuite que ces propriétés sont satisfaites par les autres composants.

La deuxième approche est l'approche dite «*bottom-up*», elle consiste à définir

des composants *réutilisables* qui ne sont pas propres à un système particulier, dans ce cas, on doit définir explicitement les propriétés que doit satisfaire l'environnement. Cette approche correspond à la propriété *assumption/guarantee* de Jones [56]. Dans cette approche, un composant satisfait une *guarantee* si son environnement satisfait l'*assumption*.

Différents mécanismes de composition de modèles, ont été proposées pour la méthode B classique par Abrial dans [10] et plus tard par Potet dans [69]. Plus tard, des travaux ont été effectués pour proposer des mécanismes de composition pour la méthode B événementielle, ils seront abordés en détails dans le chapitre 3.

1.2 Motivations

Il existe des travaux traitant de la composition de protocoles de sécurité que nous détaillerons plus loin dans le chapitre 5. En étudiant ces techniques, nous avons constaté que celles-ci fixaient de nombreuses contraintes pour la façon avec laquelle les protocoles étaient composés. Nous pensons qu'il est possible de proposer des améliorations concernant cet aspect. Un autre constat effectué est que ces travaux répondaient à la question de savoir si deux protocoles étaient composables en définissant des conditions nécessaires à la composition, par contre, elles ne disent rien si ces conditions nécessaires ne sont pas satisfaites. Nous apportons également des améliorations sur cet aspect des choses.

Dans la première partie de notre travail, nous nous sommes intéressés aux protocoles cryptographiques dans le modèle formel et aux propriétés d'*authentification* et d'établissement de clés. Nous avons proposé une technique basée sur la composition de protocoles simples, afin d'obtenir des protocoles plus complexes. Nous nous sommes ensuite intéressés à la composition des protocoles cryptographiques avec d'autres types de protocoles, nous avons mené dans ce cadre une étude de cas sur un protocole de porte-monnaie électronique nommé Mondex que nous avons composé avec un protocole d'authentification.

Les protocoles cryptographiques permettent, entre autres, de fournir la propriété d'authentification, cette propriété va de pair avec la propriété d'*autorisation*. En effet, la propriété d'autorisation permet de déterminer si un utilisateur A a le droit ou non d'accéder à une ressource, elle ne garantit cependant pas que la requête en question vient effectivement de l'utilisateur A, ceci est garanti par la propriété d'authentification. Longtemps ces problématiques d'autorisation et d'authentification ont été considérées séparément mais aujourd'hui le développement des services distants entre des entités qui ne savent pas grand chose les unes des autres nous pousse à traiter ensemble ces problématiques. Comme premier pas dans cette démarche, nous avons entrepris de modéliser différentes politiques de contrôle d'accès avec des modèles B événementiels et de vérifier que leur implémentation est correcte. Il sera par la suite possible d'envisager de combiner ces modèles dont la correction de l'implémentation est vérifiée avec des modèles de protocoles garantissant la propriété d'authentification.

1.3 Objectifs et apports de la thèse

1.3.1 Objectifs de la thèse

L'objectif de cette thèse est de contribuer au développement d'un cadre de modélisation, d'analyse et de vérification pour les protocoles de sécurité basé sur une approche modulaire. L'approche modulaire est au centre de nos travaux, elle consiste à composer des protocoles développés séparément. Cette composition peut être utilisée pour atteindre différents objectifs :

- Garantir que l'exécution dans un même environnement de plusieurs protocoles de sécurité prouvés corrects séparément n'ait pas de conséquences négatives sur les propriétés de sûreté prouvées sur chacun des protocoles.
- Prouver la correction de protocoles complexes de façon modulaire en composant des protocoles plus simples.

Ce cadre de vérification des protocoles de sécurité doit répondre à un certain nombre d'exigences en matière de modélisation, il faut veiller entre autres à :

- L'automatisation : le processus de modélisation des protocoles ainsi que celui des preuves doit être aussi automatique que possible.
- La réutilisation des preuves : lors de la composition de deux protocoles, les preuves de chacun des protocoles ne doivent pas être remises en cause, les seules supplémentaires doivent être celles relatives à la composition.
- Pouvoir composer des protocoles hétérogènes : il est indispensable de pouvoir composer des protocoles de natures différentes mais qui partagent des données communes.

1.3.2 Apports de la thèse

Le premier apport de la thèse concerne la proposition d'une méthode pour la composition des protocoles de sécurité avec la méthode B événementielle (chapitre 6), notre méthode basée sur les concepts déjà existants de décomposition des modèles et de fusion des événements. Notre méthode de composition des protocoles est également basée sur le calcul des invariants en utilisant le concept de plus faible pré-condition. Une comparaison avec les travaux existant dans le domaine est effectuée plus loin dans ce manuscrit.

Avant de composer les protocoles, il faut d'abord les modéliser séparément, un deuxième apport de cette thèse concerne une proposition d'une modélisation des protocoles cryptographiques dans le modèle formel (chapitre 6). Les principales propriétés de sûreté sont modélisées à l'aide de spécifications abstraites qui peuvent être réutilisées dans différents développements. Une modélisation en B événementiel du modèle de l'attaquant de Dolev-Yao a également été proposée et réutilisée dans plusieurs études de cas. Nous proposons également une méthode qui permet de reconstruire d'éventuelles attaques de l'attaquant.

Ces techniques de modélisation et de composition de protocoles cryptographiques sont illustrées dans le chapitre 6 sur des exemples de protocoles qui sont corrects et sur des exemples de protocoles qui sont sujets à des attaques. Les modèles complets d'un des exemples peuvent être trouvés dans l'annexe A de cette thèse.

Le troisième apport de cette thèse concerne la problématique du contrôle d'accès où une modélisation B événementielle est proposée pour les principaux modèles de politiques de contrôle d'accès avec leurs modèles d'administration

(chapiter 8). Ces modèles sont ensuite utilisés pour faire la preuve de la correction de leur implémentation (chapitre 9). Nous avons également proposé une méthode pour la modélisation et l'implémentation des politiques de contrôle d'accès composées.

Le dernier apport de cette thèse concerne certains aspects de la modélisation en B événementielle que nous avons utilisés et qui sont susceptibles d'être réutilisés dans d'autres développements utilisant le B événementiel. Ces aspects sont résumés dans le chapitre 4 de cette thèse.

1.4 Les publications

Les publications faites dans le cadre de cette thèse sont comme suit :

1. Conférences internationales avec comité de lecture et actes :
   - Nazim Benaissa, Dominique Cansell, and Dominique Méry. Integration of security policy into system modeling. In Jacques Julliand and Olga Kouchnarenko, editors, *B*, volume 4355 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2007.
   - Nazim Benaissa. Modelling attacker's knowledge for cascade cryptographic protocols. In *ABZ '08 : Proceedings of the 1st international conference on Abstract State Machines, B and Z*, pages 251–264, Berlin, Heidelberg, 2008. Springer-Verlag.
   - Nazim Benaissa and Dominique Mery. Cryptographic protocols analysis in event B. In *7th International Andrei Ershov Memorial Conference, PSI 2009*, Lecture Notes in Computer Science, pages 282–293, Novosibirsk, Russia, June 15-19 2009. Springer.
   - Nazim Benaissa and Dominique Mery. Proof-based design of security protocols. In *International Computer Science Symposium in Russia, CSR 2010*, Lecture Notes in Computer Science. Springer, 2010.
2. Atelier nationaux avec comité de lecture :
   - Nazim Benaissa and Dominique Mery. Développement combiné et prouvé de systèmes transactionnels cryptologiques. In *Atelier Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL 2009)*, pages 121–136, Toulouse, France, 26-28 January 2009. Actes AFADL.

Première partie

Le raffinement et la méthode B événementielle

Chapitre 2

La méthode B événementielle : État de l'art

2.1 Introduction

La méthode B événementielle [10] est une méthode formelle introduite par Jean Raymond Abrial. Elle est une évolution de la méthode B [1] qu'Abrial avait lui même proposée auparavant dans son livre «The B Book». La méthode B est une méthode formelle conçue pour développer des logiciels sûrs. Abrial la définit de la manière suivante [10] :«*B est une méthode pour spécifier, développer et coder les logiciels.*». On retrouve beaucoup des fondements de la méthode B dans le langage Z qu'Abrial avait introduit auparavant. Z est un langage permettant de spécifier un système informatique en utilisant la logique du premier ordre et la théorie des ensembles. Ces fondements mathématiques sont la base de la méthode B qui permet selon son auteur [10] de «*programmer en retournant aux mathématiques*». Ce *retour aux mathématiques* permet d'associer à chaque étape de la construction d'un programme une *preuve* que cette construction est correcte.

La méthode B accompagne le concepteur d'un programme à partir d'une spécification mathématique abstraite jusqu'au code informatique correspondant. Ce passage de la spécification abstraite au code concret se fait grâce à une ou plusieurs étapes de raffinements successifs. Chaque étape de raffinement est validée par des mécanismes de preuves garantissant leur correction. Chacun de ces modèles B obtenu par les raffinements successifs est appelé *machine*. Un composant B est ainsi constitué d'une machine abstraite contenant une spécification donnée et d'un ensemble de raffinements. Le but de chaque raffinement est dans le cas de la méthode B de réduire l'indéterminisme de la machine qu'elle raffine pour pouvoir arriver à l'issue de la chaine de raffinement au code informatique.

Chaque machine possède un état défini par un ensemble de variables et contient un ensemble d'*opérations*. Une opération définie un changement d'état possible de la machine. Ces changements d'états se font grâce à des *substitutions généralisées*. Une opération possède également des paramètres d'entrée et de ré-

[1] Le nom B fait référence à *Nicolas Bourbaki* un mathématicien imaginaire dont le nom a été utilisé par un groupe de mathématiciens francophones à partir des années 30 pour publier leurs œuvres.

sultat. Les opérations en B peuvent être vues comme des fonctions d'un langage de programmation qui peuvent être appelées par un opérateur extérieur. Des propriétés d'invariance peuvent être définies sur les machines B, ces propriétés doivent rester valides après l'appel des différentes opérations de la machine.

La méthode B événementielle est une évolution de la méthode B apparue en 1996 [11, 34] dont l'objectif est de modéliser toutes sortes de systèmes fermés pas forcément informatiques. Un système fermé est un système modélisé avec l'ensemble de toutes les interactions avec son environnement et il n'y a donc pas besoin de modéliser des entrées ou sorties pour communiquer avec l'environnement. La méthode B événementielle reste basée sur des raffinements successifs de machines. Les machines B événementielles sont définies par un état et des propriétés d'invariance comme dans la méthode B, cependant les opérations sont remplacés par des événements dans les machines B événementielles. Contrairement aux opérations qui sont appelées par un opérateur extérieur, les événements se déclenchent par eux-même si une condition, appelée garde de l'événement, est vérifiée. Une garde est associée à chaque événement. Plusieurs gardes d'événements peuvent être vraies au même temps, cependant, un seul événement peut se déclencher. Le choix de l'événement qui se déclenche est non déterministe.

2.2 Structure d'une machine B événementielle

Une machine B événementielle est constituée d'un ensemble de variables définissant l'état du système modélisé ainsi que d'un ensemble d'événements faisant évoluer ces variables. Ces variables doivent satisfaire les propriétés d'invariance du système. Une machine B événementielle est organisée en clauses : les clauses VARIABLES et INVARIANT contiennent respectivement la liste des variables ainsi que les propriétés d'invariance. La clause INVARIANT doit permettre au moins de typer des variables déclarées dans la clause VARIABLES. Les types sont ensembles porteurs qui sont définis dans un contexte *vu* par la machine (les contextes sont présentés dans le paragraphe suivant). La clause THEOREM contient des propriétés qui peuvent être déduites des propriétés d'invariance mais qui ne constituent pas forcément des invariants inductifs, cette clause peut également contenir des propriétés que l'on souhaite prouver afin des les utiliser dans la preuve des invariants du modèle. La clause EVENTS contient la liste des événements dont l'événement INITIALISATION qui donne une valeur initiale aux variables.

```
MACHINE
/* Nom de la machine */
SEES
/* Liste des contextes «vus» par le modèle */
VARIABLES
/* Liste des variables d'état du modèle */
INVARIANT
/* Propriétés d'invariance du système */
THEOREMS
/* Liste des théorèmes de la machine */
EVENTS
/* Les événements de la machine */
```

Les clauses que l'on vient de décrire définissent la partie dynamique du modèle, il peut cependant s'avérer nécessaire de définir une partie statique contenant les constantes du modèles ainsi qu'un ensemble de propriétés sur ces constantes. Cette partie statique constitue un *contexte*. Un contexte est constitué de plusieurs clauses. La clause SETS contient les ensembles porteurs du modèle, ces ensembles sont non vides et permettent de typer le reste des entités du modèle. Il est possible de déclarer les ensembles de la clause SETS en indiquant la liste des éléments les constituant, dans ce cas, les éléments listés sont distincts les uns des autres. La clause CONSTANTS contient la liste des constantes, et la clause AXIOMS contient l'ensemble des propriétés des constantes et notamment leurs types. La clause THEOREMS contient les propriétés déduites à partir des propriétés présentent dans la clause AXIOMS.

```
CONTEXT
/* Nom du contexte */
EXTENDS
/* Liste des contextes «vus» par le contexte */
SETS
/* Liste des ensembles porteurs */
CONSTANTS
/* Liste des constantes du contexte */
AXIOMS
/* Propriétés du contexte */
THEOREMS
/* Liste des théorèmes du contexte */
```

Un contexte peut étendre un autre contexte en introduisant de nouvelles constantes et de nouvelles propriétés, on rajoute dans ce cas le nom du contexte étendu dans la clause EXTENDS du nouveau contexte. Les machines et les contextes sont définies séparément, afin d'indiquer à une machine donnée les contextes qu'elle «*voit*», le nom du contexte est rajouté dans la clause SEES de la machine. Quand une machine voit un contexte, elle peut en utiliser les constantes ainsi que les propriétés de celle-ci figurant dans la clause AXIOMS du contextes. Les propriétés contenues dans la clause THEOREMS de la machine peuvent être déduites des propriétés présentes dans les clause AXIOMS et THEOREMS du contexte. Une machine peut voir un ou plusieurs contextes.

Nous présentons un petit exemple pour illustrer nos propos. Il s'agit d'une machine B événementielle modélisant le comportement d'une horloge. Le premier modèle abstrait ne contient que les heures. Le modèle contient une variable `heure` contient l'heure courante, cette variable est un entier naturel compris entre 0 et 23. Un événement `TICK` fait évoluer l'heure courante et un autre événement `RAZ` remet l'heure à zéro l'heure quand celle-ci dépasse 23. L'initialisation met la valeur de l'horloge à 0. La propriété d'invariance indique que l'heure doit être comprise entre 0 et 23.

```
MACHINE
      HORLOGE
VARIABLES
      heure
INVARIANT
      heure ∈ 0..23
INITIALISATION
      heure := 0
EVENTS
EVENT TICK
      WHEN
       heure < 23
      THEN
       heure := heure + 1
      END

EVENT RAZ
      WHEN
       heure = 23
      THEN
       heure := 0
      END
```

2.3 Les événements

Un événement est composé de deux parties : une garde qui définit la condition selon laquelle l'événement peut ou non se déclencher et une action qui définit l'évolution des variables d'état. Cette notion d'événement est similaire aux actions de Back [14] ou des commandes gardées de Dijkstra. La partie action des événements est constituée d'une substitution qui fait évoluer la valeur des variables. Les substitutions ont trois formes possibles :

- La substitution généralisée $x :| P(x, x')$: c'est la forme la plus générale des substitutions, elle exprime la relation entre la valeur d'une variable x avant et après la substitution. Ici, x' représente la valeur de la variable x après la substitution et P est un prédicat, ainsi la variable x a une nouvelle valeur x' qui est telle que le prédicat $P(x, x')$ est vrai.
- La substitution ensembliste $x :\in E(x)$: cette forme de substitution indique qu'une variable x est modifiée de façon à ce qu'elle devienne un

Nom	Syntaxe	Définition
Relation binaire	$s \leftrightarrow t$	$\mathcal{P}(s \times t)$
Composition	$r_1; r_2$	$\{x, y \mid x \in a \ \wedge y \in b \wedge$ $\exists z.(z \in c \wedge x, z \in r_1 \wedge z, y \in r_2)\}$
Domaine	$\mathsf{dom}(r)$	$\{a \mid a \in s \wedge \exists b.(b \in t \wedge a \mapsto b \in r)\}$
Codomaine	$ran(r)$	$\mathsf{dom}(r^{-1})$
Identity	$\mathrm{id}(s)$	$\{x \mapsto x \mid x \in s\}$
Restriction	$s \lhd r$	$\mathrm{id}(s); r$
Co-restriction	$r \rhd t$	$r; \mathrm{id}(t)$
Anti-restriction	$s \lhd\!\!\!- r$	$(\mathsf{dom}(r) - s) \lhd r$
Anti-co-restriction	$r \rhd\!\!\!- t$	$r \rhd (ran(r) - t)$
Image	$r[w]$	$ran(w \lhd r)$
Fonction partielle	$s \nrightarrow t$	$\{r \mid r \in s \leftrightarrow t \ \wedge \ (r^{-1}; r) \subseteq \mathrm{id}(t)\}$
Fonction totale	$s \rightarrow t$	$\{f \mid f \in s \nrightarrow t \ \wedge \ \mathsf{dom}(f) = s\}$
Injection totale	$s \rightarrowtail t$	$\{f \mid f \in s \rightarrow t \ \wedge \ f^{-1} \in t \nrightarrow s\}$

TAB. 2.1 – Les notations ensemblistes de la méthode B

élément d'un ensemble $E(x)$. Il faut noter que toutes les substitutions peuvent s'exprimer sous la forme généralisée, dans le cas ensembliste, on peut l'exprimer sous la forme $x :| (x' \in E(x))$.

- La substitution simple $x := Exp(x)$: cette forme de substitution ressemble à une affectation où la variable x prend la valeur de l'expression $Exp(x)$. La substitution $heure := heure + 1$ de l'événement TICK de l'exemple précédent est une substitution simple. Cette forme de subsitution peut également être exprimée sous forme normale avec $x :| (x' = Exp(x))$
- Les substitutions parallèles $x :| P(x, x') \parallel y :| P(y, y')$: Il est possible d'avoir plusieurs substitutions en parallèle, dans ce cas les substitutions sont faites au même temps mais ne peuvent, cependant, pas concerner les mêmes variables. La forme normale de ce type de substitution est $x, y :| (P(x, x') \wedge Q(y, y'))$.

La méthode B événementielle est basée sur la logique du premier ordre et la théorie des ensembles, le tableau 2.1 résume les notations ensemblistes les plus utilisées dans la méthode B. Dans ce tableau, la relation r est une relation binaire, s et t sont deux ensembles.

2.3.1 Les différentes formes d'événements

On trouve dans la méthode B événementielle trois formes d'événements, la première est la forme dite *indéterministe* où x est une variable d'état du système et t une variable locale. L'événement ne se déclenche que s'il existe une valeur de la variable t qui satisfait le prédicat $G(x, t)$. $x :| P(x_0, x, t)$ est l'action de l'événement. La présence de la variable locale t fait que cet événement est non-déterministe.

```
EVENT nom
    ANY
     t
    WHERE
     G(x, t)
    THEN
     x :| P(x, x′, t)
    END
```

La seconde forme est celle dite *gardée* dans laquelle il n'y a pas de variable locale et où la garde et l'action ne dépendent que des variables d'états du modèle :

```
EVENT nom
    WHEN
     G(x)
    THEN
     x :| P(x, x′)
    END
```

La dernière forme est celle dite simple où la garde est toujours vraie :

```
EVENT nom
    BEGIN
     x :| P(x, x′)
    END
```

Avant de passer aux obligations de preuve d'un modèle B, il faut présenter les prédicats *avant-après* des événements traditionnellement notés BA (pour before-after en anglais). Ces prédicats définissent la relation entre la valeur des variables d'état avant et après le déclenchement des événements. La valeur d'une variable x après le déclenchement d'un événement est notée x'. Par exemple le prédicat avant-après de l'événement TICK de la machine `HORLOGE` est $heure' = heure+1$. Le tableau 2.2 résume les prédicats avant-après selon la forme des événements.

Le tableau 2.3 résume la garde selon les différents type d'événements. La garde d'un événement E est notée $grd(E)$. La garde des événements simples est toujours à vrai. Il peut arriver que la garde d'un événement soit toujours fausse, dans ce cas-là l'événement ne se déclenchera jamais.

2.3.2 Les obligations de preuve

Un des aspects les plus importants de la méthode B événementielle est la preuve de la correction des modèles. Afin de garantir cette correction, une série d'obligations de preuves doivent être faites. Ces obligations de preuves concernent différents aspects du modèle, certaines servent à prouver les propriétés

L'événement : E	Le prédicat avant-après
BEGIN $x :\mid P(x, x')$ END	$P(x, x')$
WHEN $G(x)$ THEN $x :\mid P(x, x')$ END	$G(x) \quad \wedge \quad P(x, x')$
ANY t WHERE $G(t, x)$ THEN $x :\mid P(x, x', t)$ END	$\exists t \cdot (G(t, x) \quad \wedge \quad P(x, x', t))$

TAB. 2.2 – Les prédicats «avant-après»

L'événement : E	La garde : grd(E)
BEGIN $x :\mid P(x, x')$ END	$TRUE$
WHEN $G(x)$ THEN $x :\mid P(x, x')$ END	$G(x)$
ANY t WHERE $G(t, x)$ THEN $x :\mid P(x, x')$ END	$\exists t \cdot G(t, x)$

TAB. 2.3 – Les gardes des événements.

d'invariance d'une machine, à prouver la correction d'un raffinement où encore pour prouver que des propriétés présentes dans la clause THEOREMS sont correctes. Nous détaillons dans cette section les obligations de preuve associées à une machine abstraite d'un modèle B événementielle.

2.3.2.1 Faisabilité d'un événement

L'ensemble des événements d'une machine B événementielles doivent être toujours faisables. Ceci veut dire que pour une substitution d'un événement sous forme normale $x :| P(x, x')$, il doit exister une nouvelle valeur x' de la variable x qui satisfait le prédicat $P(x, x')$ où x est une valeur de la variable avant le déclenchement de l'événement que peut atteindre le système (qui vérifie l'invariant) et qui satisfait la garde de l'événement. I étant l'invariant du système, alors l'obligation de preuve qui correspondant à un événement E est comme suit :

$$I(x) \ \wedge \ \mathsf{grd}\,(E) \ \ \Rightarrow \ \ \exists x' \cdot P(x, x')$$

2.3.2.2 L'invariant d'un modèle

L'invariant du modèle est une propriété que le système doit préserver quelle que soit son évolution, ceci implique que chaque déclenchement de tout événement du système doit préserver cette propriété. L'événement INITIALISATION présent dans toutes les machines B événementielles doit également satisfaire à cette exigence. Si $I(x)$ est l'invariant du système, alors pour tout événement E du système qui a comme prédicat avant-après BA l'obligation de preuve est comme suit :

$$I(x) \ \wedge \ BA(x, x') \ \Rightarrow \ I(x')$$

Si la garde d'un événement est fausse alors le prédicat avant-après de l'événement est toujours faux et l'obligation de preuve précédente est forcément vraie. Pour l'événement INITIALISATION, il faut prouver que la substitution $x :| Init(x')$ associée à celui-ci implique la correction de l'invariant. L'obligation de preuve correspondante est comme suit :

$$Init(x') \ \Rightarrow \ I(x')$$

L'obligation de preuve associée à l'événement TICK de l'exemple précédent est comme suit :

$$heure \in 0..23 \ \wedge \ heure < 23 \ \wedge \ heure' = heure + 1 \ \Rightarrow \ heure' \in 0..23$$

Il faut également prouver les théorèmes de la clause THEOREMS, on peut utiliser pour cela l'invariant de la machine ainsi que l'ensemble des prédicats dans les clauses AXIOMES et THEOREMS du contexte.

2.4 Raffinement d'une machine B événementielle

Le raffinement est l'une des notions les plus importantes dans la méthode B événementielle, elle permet de développer le système de manière incrémentale en partant du modèle abstrait qui constitue une spécification du système. Les détails du système sont rajoutés de façon graduelle dans chaque raffinement. Le raffinement permet de conserver les propriétés déjà prouvées dans les modèles plus abstraits. Lors d'un raffinement, de nouvelles variables peuvent être ajoutées et d'autres variables peuvent disparaître. Les événements peuvent également être raffinés en renforçant leur garde et en raffinant leur actions. De nouveaux événements peuvent également apparaître, ceux-ci raffinent un événement particulier de l'abstraction appelé «*skip*», l'événement *skip* ne change pas l'état des variables (ici abstraites). Les événements ajoutés ne modifient donc pas les variables abstraites. Nous désignons dans la suite de cette section l'ensemble des variables abstraite par x et les nouvelles variables introduites lors du raffinement par y. L'invariant abstrait est désigné par $I(x)$ et le concret par $J(x, y)$.

La structure d'un raffinement est similaire à celle d'une machine abstraite avec l'ajout d'une clause REFINES qui indique le nom de la machine raffinée. La clause qui contient le nom de la machine est nommée REFINEMENT au lieu de MACHINE. La clause VARIABLES contient les variables qui étaient dans la machine raffinée et que l'on peut conserver dans le raffinement et contient également les variables introduites dans le nouveau raffinement. La clause INVARIANT contient l'invariant $J(x, y)$ que l'on appelle l'*invariant de collage* car il permet de faire le lien entre les variables abstraites et les variables introduites dans le raffinement. $J(x, y)$ permet également de typer les variables nouvellement introduites.

Il peut arriver que les événements introduits dans un raffinement se déclenchent indéfiniment et *prennent la main* sans la rendre aux événements abstraits. Il est possible de définir un *variant* dans la clause du même nom, il s'agira ensuite de prouver que ce variant évoluera de façon à permettre le déclenchement des événements abstraits. Les autres clauses sont similaires à celles déjà vues dans le modèle abstrait.

REFINEMENT
/* Nom de la machine */
REFINES
/* Nom de la machine */
SEES
/* Nom de la machine raffinée */
VARIABLES
/* Liste des variables d'état du modèle */
INVARIANT
/* Propriétés d'invariance du système */
VARIANT
/* Le variant du système */
THEOREMS
/* Liste des théorèmes de la machine */
EVENTS
/* Les événements de la machine */

2.4.1 Les obligations de preuves du raffinement

Une série d'obligations de preuve doivent être déchargées pour garantir la correction du raffinement. La première obligation de preuve concerne l'initialisation concrète. La substitution de l'initialisation, notée $y : INIT(y')$ attribue une valeur initiale aux nouvelles variables introduites ainsi qu'à celles du modèle abstrait que l'on a conservées. Il est alors nécessaire de prouver que cette initialisation n'est pas en contradiction avec l'initialisation abstraite et qu'elle vérifie le nouvel invariant concret :

$$INIT(y') \Rightarrow \exists x'.(init(x') \wedge J(x', y'))$$

La deuxième obligation de preuve concerne la preuve du raffinement des événements. Un événement raffine un événement abstrait en renforçant sa garde. L'action de l'événement raffiné peut modifier les nouvelles variables ainsi que les variables héritées du modèle abstrait. Cependant, la modification des variables abstraites doit se faire dans le cadre des modifications faites dans l'événement abstrait. Le prédicat avant-après de l'événement abstrait est noté $BAA(x, x')$ tandis que celui de l'événement concret est noté $BAC(y, y')$. L'obligation de preuve qui garantit que l'événement abstrait est correctement raffiné par l'événement concret est comme suit :

$$I(x) \wedge J(x, y) \wedge BAC(y, y') \Rightarrow \exists\, x' \cdot (\, BAA(x, x') \wedge J(x', y')\,)$$

Si l'événement concret est un nouvel événement (qui raffine *skip*), alors dans ce cas le prédicat avant-après de l'événement abstrait est tel que $BAA(x, x') = (x = x')$). Ainsi, l'obligation de preuve précédente devient :

$$I(x) \wedge J(x, y) \wedge BAC(y, y') \Rightarrow J(x, y')$$

Les deux obligations de preuves suivantes concernent le variant. Celui-ci sert à empêcher les nouveaux événements de s'exécuter indéfiniment au détriment des événements abstraits. Le variant est une expression arithmétique qui doit décroître à chaque déclenchement d'un nouvel événement. Dans la première obligation, de preuve on prouve que le variant est un entier positif :

$$I(x) \wedge J(x, y) \Rightarrow V(y) \in \mathbb{N}$$

Par la suite, il faut prouver que le variant décroît à chaque exécution d'un nouvel événement. Comme le variant est un entier positif, on est sûr que les nouveaux événements finiront par redonner la main aux événements abstraits :

$$I(x) \wedge J(x, y) \wedge BAC(y, y') \Rightarrow V(y') < V(y)$$

La dernière obligation de preuve d'un raffinement consiste à prouver qu'il n'y a pas plus de blocage dans la machine concrète que dans la machine qu'elle raffine. Cette obligation de preuve est importante dans la mesure où les raffinements se font généralement en renforçant la garde des événements, ce qui peut induire la présence de plus de blocage dans le système. Dans l'obligation de preuve suivante, les gardes des événements concrets sont notées $G_1..G_n$ et les gardes des événements abstraits sont notées $H_1..H_m$:

$$I(x) \ \wedge \ J(x,y) \ \wedge \ (G_1(x) \vee \cdots \vee G_n(x)) \quad \Rightarrow \quad (H_1(y) \vee \cdots \vee H_m(y))$$

2.4.2 Un exemple de raffinement

Nous présentons en ce qui suit un exemple simple de raffinement. Il s'agit d'un raffinement possible de la machine `HORLOGE` présentée précédemment. Nous introduisons dans ce raffinement une nouvelle variable `minute` qui complète l'heure courante contenue dans la variable `heure`. Dans ce raffinement, la variable abstraite `heure` est donc conservée.

```
VARIABLES
      heure
      minute
INVARIANT
      minute ∈ 0..59
```

Un nouvel événement est introduit pour faire évoluer les minutes, il s'agit de l'événement `TICK_MIN` :

```
EVENT TICK_MIN
      WHEN
       minute < 59
      THEN
       minute := minute + 1
      END
```

Les événements `TICK` et `RAZ` sont raffinés pour modifier à la fois les heures et les minutes :

```
EVENT TICK
REFINES TICK
    WHEN
     minute = 59
     heure < 23
    THEN
     heure := heure + 1
     minute := 0
    END

EVENT RAZ
REFINES RAZ
    WHEN
     minute = 59
     heure = 23
    THEN
     heure := 0
     minute := 0
    END
```

Le variant qui a comme valeur $59 - \texttt{minute}$ est décrémenté à chaque exécution de l'événement `TICK_MIN` jusqu'à arriver à 0 et permettre ainsi aux événements `TICK` ou `RAZ` de s'exécuter.

```
VARIANT
    59 − minute
```

2.5 Les outils de la méthode B

Plusieurs outils ont accompagné le développement de la méthode B depuis sa création jusqu'à aujourd'hui. Historiquement, le premier outil proposé s'appelle AtelierB [37], cet outil a été développé pour un projet de train automatique appelé Météor. Les prouveurs de l'atelierB ont été ensuite mis gratuitement à la disposition des chercheurs sous l'appellation de B4Free [38]. Par la suite, des interfaces ont été développées pour la suite de prouveurs B4Free telles que l'interface Click&Prove que nous avons utilisée au début de la thèse.

Récemment, un atelier pour le B événementiel nommé RODIN [33] a été développé dans le cadre d'un projet européen. Celui-ci a été développé sous la forme de plugins Eclipse. L'atelier RODIN dispose de son propre prouveur appelé NewPP mais permet d'utiliser la suite de prouveurs B4Free. Tous les modèles présentés dans ce manuscrit ont été développés avec l'outil RODIN.

2.6 Conclusion

Nous avons introduit dans ce chapitre les concepts de base de la méthode B événementielle. Nous avons présenté le raffinement dans cette méthode, nous

avons également exposé les différentes obligations de preuve relatives au modèle abstrait et aux raffinements. La notion de raffinement constitue le cœur de la méthode B événementielle, elle peut être appliquée de différentes manières, nous présentons dans le chapitre suivant les différentes techniques de raffinements ainsi que les techniques de composition et de décomposition de modèles B événementiels qui ont été utilisées dans le cadre de cette thèse.

Chapitre 3

La composition et la décomposition des modèles dans la méthode B événementielle

3.1 Introduction

Le raffinement peut être utilisé de diverses manières selon l'étude de cas traitée. Nous détaillons dans cette section les techniques de raffinement avancées de la méthode B événementielle en insistant sur les techniques utilisées dans le cadre de cette thèse. Nous avions vu précédemment que raffiner un modèle, revient à raffiner, d'une part, l'état du système (caractérisé par les variables d'état), et d'autres part, à raffiner les événements du modèle.

Raffiner l'état du système revient à ajouter de nouvelles variables indépendantes des variables abstraites et à introduire de nouvelles variables pour remplacer les variables abstraites. Pour le raffinement des événements, on peut raffiner des événements existants en renforçant leurs gardes et en raffinant leurs actions ou introduire de nouveaux événements raffinant *skip* afin d'observer des comportements concrets qui n'apparaissaient pas dans l'abstraction. La définition d'un invariant de collage est aussi importante car elle permet de faire les preuves de la correction du raffinement.

Tous ces choix possibles font qu'il y a une multitude de façons de raffiner un modèle. Le concepteur du système doit faire les bons choix notamment pour avoir de meilleurs résultats en terme de preuve automatique. Ces différentes façons de faire correspondent à des techniques de raffinements particulières. On distingue deux grandes familles de techniques de raffinements : le raffinement horizontal et le raffinement vertical.

- Le raffinement horizontal vise à introduire de nouvelles propriétés dans le système qui n'ont pas été introduites dans l'abstraction. Ceci peut se faire en introduisant de nouvelles variables avec les événements qui vont avec et qui les font évoluer. Ainsi le système modélisé est complété au fur et à mesure que l'on avance dans le processus de raffinement.

- Le raffinement vertical vise, quant à lui, à remplacer les structures abstraites par des détails plus concrets à chaque étape de raffinement. Il y a plusieurs façons de faire du raffinement vertical :
  - Le raffinement de données : dans lequel les variables abstraites sont remplacées par des variables concrètes pour se rapprocher de l'implémentation.
  - La décomposition des modèles : le développement de grands modèles peut s'avérer difficile et des techniques ont été proposées pour décomposer les modèles en plusieurs parties et les développer séparément. Deux manières de décomposer les modèles ont été proposées, l'une basée sur les variables d'état proposée par Abrial [12, 33], l'autre basée sur la décomposition d'événements proposée par Butler [41].

Il faut noter que la méthode B classique contenait déjà des mécanismes compositionnel de modèles [10]. Parmi ces mécanismes, on peut citer l'utilisation des clauses INCLUDES, EXTENDS et USES qui sont des mécanismes syntaxiques qui permettait par exemple à une machine de *voir* ce qu'il y avait dans une autre machine. Des travaux sur la compositions des modèles dans la méthode B classique ont également été faits par Potet [69]. Nous nous focalisons dans ce chapitre sur la composition et la décomposition des modèles dans la méthode B événementielle.

Nous détaillons dans la section suivante les deux techniques de décompositions de modèles dans la méthode B événementielle basées respectivement sur les variables d'états du système et sur les événements.

3.2 La décomposition par variables d'états

L'introduction de nouvelles variables et de nouveaux événements est la base de la technique de raffinement, ceci est bénéfique d'un point de vue de la facilité des preuves et de leurs automatisations. En même temps, au fur et à mesure que le nombre de variables et d'événements devient grand et le processus de raffinement peut devenir très rapidement difficile à gérer surtout pour de grands projets.

La décomposition de machines B événementielles consiste à diviser le modèle en plusieurs parties et en faire le développement séparément. La figure 3.1 prise de [12] montre la démarche de décomposition d'un modèle $\mathtt{M_n}$ ayant pour contexte $\mathtt{C_n}$ résultant d'un processus de raffinement d'un modèle abstrait $\mathtt{M_1}$ ayant pour contexte $\mathtt{C_1}$. $\mathtt{M_n}$ est décomposé en deux modèles $\mathtt{N_1}$ ayant comme contexte $\mathtt{D_1}$ et un modèle $\mathtt{P_1}$ ayant comme context $\mathtt{E_1}$. Les modèles $\mathtt{N_1}$ et $\mathtt{P_1}$ peuvent ensuite être raffinés à leurs tour. Les modèles obtenus à l'issue du raffinement peuvent ensuite être recomposés pour obtenir le système que l'on aurait obtenu en raffinant $\mathtt{M_n}$ sans le décomposer. Cette étape de recomposition n'est pas indispensable.

La principale difficulté de cette démarche est l'éventuelle présence de variables partagées entre les modèles $\mathtt{N_1}$ et $\mathtt{P_1}$. Supposons que le modèle $\mathtt{M_n}$ contient quatre événements $\mathtt{e1}, \mathtt{e2}, \mathtt{e3}, \mathtt{e4}$ avec :

- les événements $\mathtt{e1}, \mathtt{e2}$ partagent une variable $\mathtt{v1}$
- les événements $\mathtt{e2}, \mathtt{e3}$ partagent une variable $\mathtt{v2}$

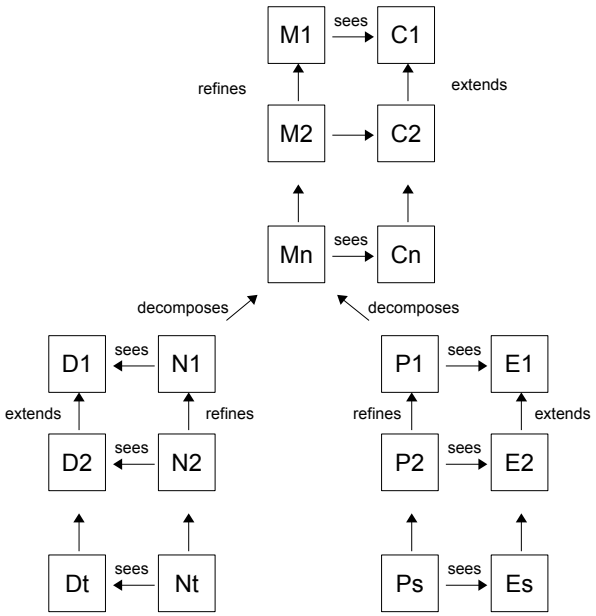


FIG. 3.1 – La décomposition de modèles.

– les événements e3, e4 partagent une variable v3

On décompose M_n de la façon suivante : N_1 contient les événements e1 et e2 et le modèle P1 contient les événements e3 et e4. La variable v2 est donc partagée entre les deux modèles résultant de la décomposition. Cette variable pose problème dans la mesure où il n'est pas évident que les propriétés prouvées sur cette variable séparément dans les raffinements $N_1, .., N_t$ d'un coté et $P_1, .., P_s$. Cette décomposition est illustrée dans la figure 3.2.

Une solution est proposée par Abrial dans [12, 33], elle consiste à définir les notions de variables externes et d'événements externes. Les variables externes sont les variables partagées entre les deux parties du systèmes. Les événements externes sont des événements rajoutés dans chacune des parties du système pour simuler les actions de l'autre partie sur les variables partagées. Dans notre exemple, il faut introduire dans N_1 un événement e3ext simulant les actions de l'événement e3 sur la variable partagée v2. Il en sera de même pour P_1 et l'événement e2. Un événement externe est une abstraction de l'événement auquel il correspond qui ne modifie que les variables partagées.

Supposons que la garde de l'événement e2 est le prédicat $G_2(v1, v2)$ et l'action de cet événement est le prédicat avant-après lui correspondant est $E_2(v1, v2, v1', v2')$. On suppose également que la garde de l'événement e2ext qui simule e2 est le prédicat $G_{2ext}(v2)$ et son prédicat avant-après $E_{2ext}(v2, v2')$. Dans ce cas, e2ext est un événement externe de e2 si on peut prouver que :

$$G_2(v1, v2) \ \wedge \ E_2(v1, v2, v1', v2') \ \Rightarrow \ G_{2ext}(v2) \ \wedge \ E_{2ext}(v2, v2')$$

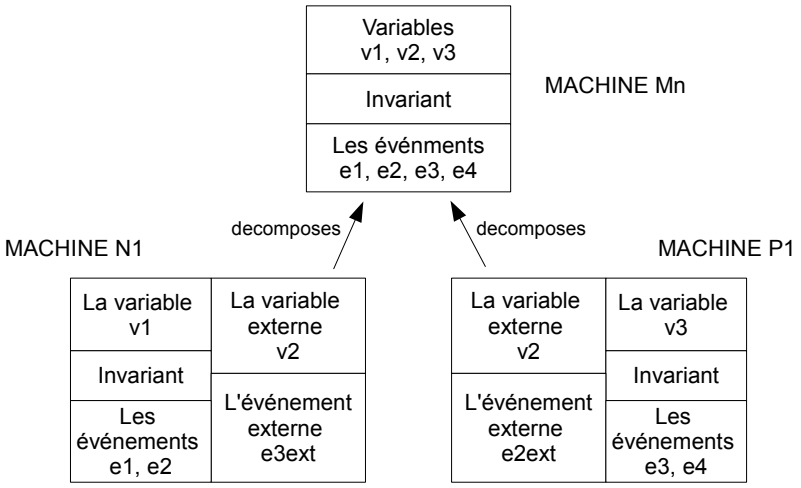


FIG. 3.2 – Les variables et événements externes.

Cette obligation de preuve signifie que l'événement externe est une abstraction de l'événement qu'il simule. Abrial a prouvé dans [12, 33] qu'à la fin du processus de raffinement, il est possible de recomposer le système pour obtenir un modèle final qui raffine le modèle initial sans obligations de preuve supplémentaires à condition de ne pas raffiner les variables partagées.

L'obligation de ne pas raffiner les variables externes peut s'avérer contraignante, elle a comme inconvénient d'obliger le concepteur du système à introduire des types concrets très tôt dans le cycle de raffinement avec des conséquences négatives sur la difficulté des preuves. Pour remédier à ce problème Abrial, Métayer et Voisin [33] proposent une solution qui permet de raffiner les variables externes et au même temps permettant de recomposer le système en étant sûr que le système obtenu après la recomposition raffine le modèle abstrait d'avant décomposition comme le résume la figure 3.3.

On suppose que le modèle `N1` est raffiné en un modèle `NR` avec comme variables `w1` et `w2`, dans ce modèle, la variable `w2` est un raffinement de la variable externe `v2`. Ce modèle contient les événements `e1_r`, `e2_r` et `e3ext_r` qui raffinent respectivement `e1`, `e2` et `e3ext`. Il en est de même avec le modèle `P` qui est raffiné en un modèle `PR` avec comme variables `w2` et `w3`, dans ce modèle, la variable `w2` est un raffinement de la variable externe `v2`. Ce modèle contient les événements `e3_r`, `e4_r` et `e2ext_r` qui raffinent respectivement `e3`, `e4` et `e2ext`. Abrial *et al* [33] prouve que le modèle `MR` obtenu en recomposant les modèles `NR` et `PR` et en enlevant les événements externes `e2ext_r` et `e3ext_r` raffine le modèle `M` mais à condition que les variables externes soient raffinées de la même manière. Si l'invariant de collage entre les modèle `N1` et `NR` est $J_N(v1, w1) \wedge w2 = h(v2)$ alors l'invariant de collage entre `N1` et `NR` doit être de la forme $J_P(v3, w3) \wedge w2 = h(v2)$. La variable partagée `v2` est donc raffiné de la même manière avec l'invariant de collage $w2 = h(v2)$ dans les deux modèles.

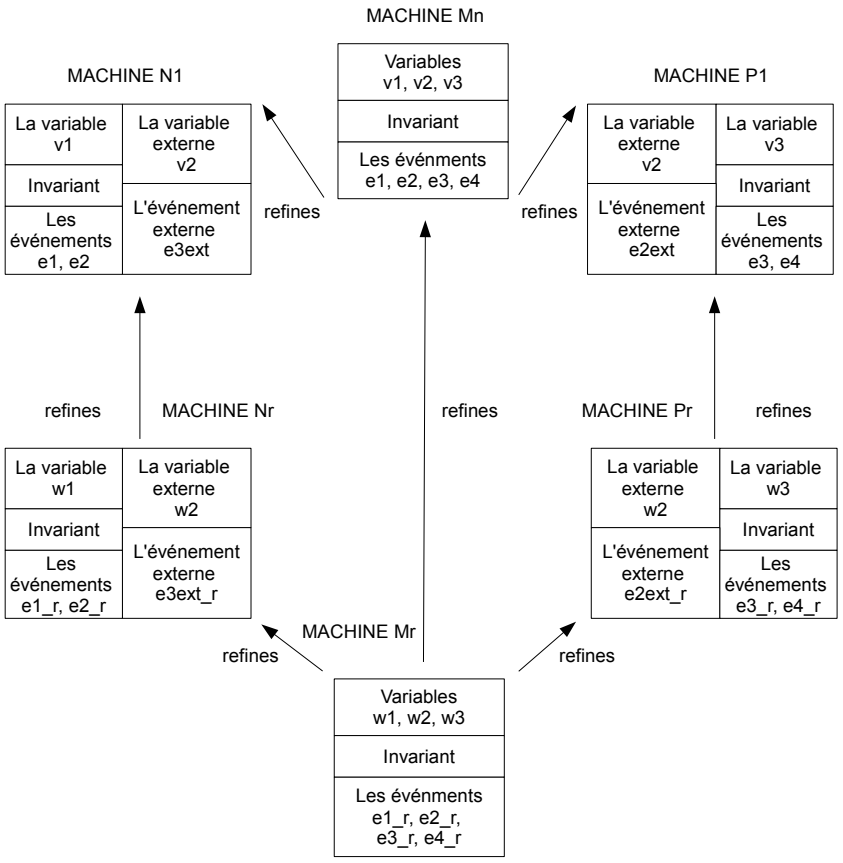


FIG. 3.3 – La décomposition et le raffinement.

3.3 La décomposition par événements

Butler [41] propose une décomposition basée sur les événements, dans cette technique, un événement atomique abstrait peut être décomposé en plusieurs sous-événements, dont au moins un raffine l'événement abstrait. La figure 3.4 montre un exemple d'une décomposition d'événement où un événement `evt` est décomposé en trois sous-événements : `start`, `step` et `end`. La ligne en continue montre que l'événement `end` raffine `evt`. Les événements `start` et `step` raffinent quand à eux l'événement *skip*. L'événement `step` est noté `step(i)` dans la figure pour dire qu'il est possible d'avoir plusieurs événements entre l'exécution de `start` et `end` et dont l'ordre d'exécution n'est pas important. Des variables sont rajoutées pour contrôler l'ordre d'exécution des événements.

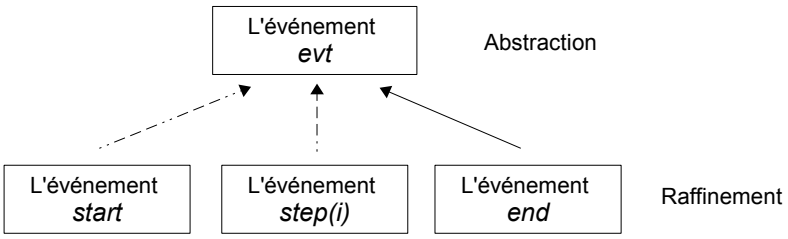


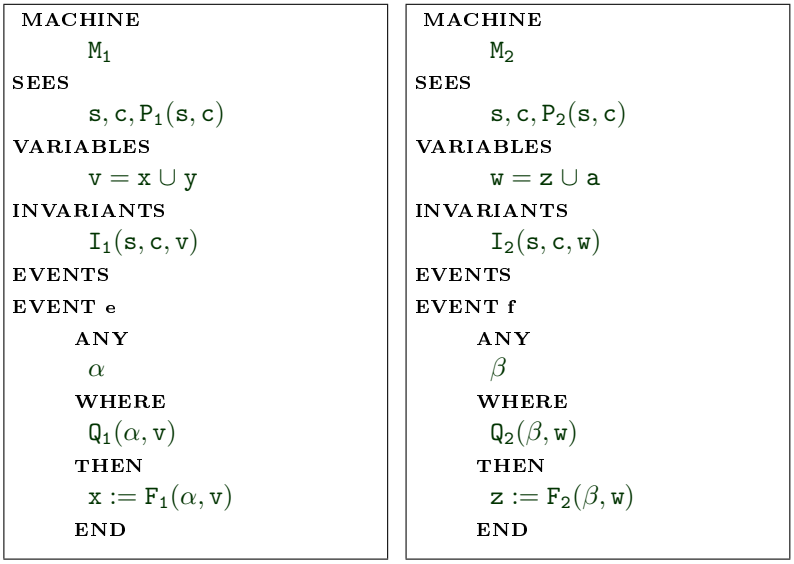
FIG. 3.4 – La décomposition par événements

La décomposition correspond à l'approche *top-down* déjà abordée dans l'introduction de cette thèse. Poppleton [68] propose une approche de composition de modèles (*bottom-up*), cette technique vise à réutiliser des composants déjà prouvés en les composant de différentes manières. Poppleton [68] propose une composition basée sur la fusion des événements qui reprend en partie les travaux de Back et Butler [15] sur la fusion et l'exécution parallèle.

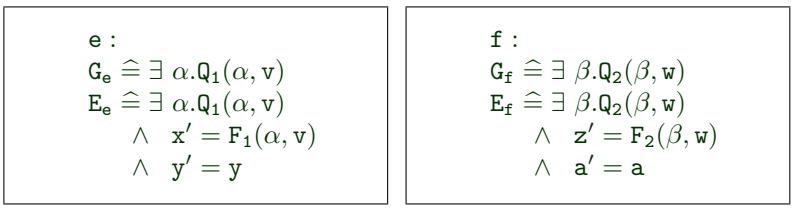
3.4 La composition des modèles par fusion d'événements

Nous présentons dans cette section une technique de composition de modèles basée sur la fusion d'événements proposée par Poppleton [68]. Cette méthode est inspirée de la méthode d'Abrial déjà présentée dans ce chapitre mais aussi des travaux de Back et Butler sur la fusion et l'exécution simultanée des événements [15]. Le but de la composition de modèles est la réutilisation de développement faits avec la méthode B événementielle prouvés précédemment. La composition de modèle est censée donner de nouveaux modèles vérifiant les propriétés de tous les modèles composés. Ceci se fait au prix de limitations dans les possibilités de combinaison ou encore au prix d'obligation de preuves supplémentaires.

La fusion d'événements consiste à fusionner deux événements issues de deux modèles différents, dans l'exemple qui va suivre chacun des deux modèles M_1 et M_2 composés ne contient qu'un seul événement (les événements `e` et `f` respectivement) mais la même technique peut être étendue dans le cas où les modèles contiennent plus d'événements.



De façon générale nous voulons fusionner deux modèles M_1 et M_2 qui ont comme invariants I_1 et I_2 respectivement. Les deux modèles contiennent respectivement les événements e et f. v est l'ensemble des variables du modèle M_1 dont un sous ensemble x qui contiennent les variables modifiées par l'événement e et un sous ensemble y contenant le reste des variables. w est l'ensemble des variables du modèle M_2 dont un sous ensemble z qui contiennent les variables modifiées par l'événement f et un sous ensemble a contenant le reste des variables du modèle M_2. s et c sont respectivement les ensembles et les contextes des deux modèles, P_1 et P_2 sont les axiomes des modèles M_1 et M_2 respectivement. Les gardes et les prédicats avant-après des deux événements sont comme suit :



La fusion des deux modèles correspond au modèle M présenté en ce qui suit :

MACHINE
M
SEES
$\mathtt{s}, \mathtt{c}, \mathtt{P}_1(\mathtt{s}, \mathtt{c}) \ \wedge \ \mathtt{P}_2(\mathtt{s}, \mathtt{c})$
VARIABLES
$\mathtt{v} = \mathtt{x} \cup \mathtt{y}$

$\mathtt{w} = \mathtt{z} \cup \mathtt{a}$
INVARIANTS
$\mathtt{I}_1(\mathtt{s}, \mathtt{c}, \mathtt{v})$

$\mathtt{I}_2(\mathtt{s}, \mathtt{c}, \mathtt{w})$
EVENTS
EVENT e $\odot$ f
ANY
α, β
WHERE
$\mathtt{Q}_1(\alpha, \mathtt{v}) \ \wedge \ \mathtt{Q}_2(\beta, \mathtt{w})$
THEN
$\mathtt{x} := \mathtt{F}_1(\alpha, \mathtt{v}) \parallel \mathtt{z} := \mathtt{F}_2(\beta, \mathtt{w})$
END

Cette fusion pose problème dans le cas de la présence de variables partagées entre les ensembles `x` et `z` dans la mesure où la définition de la substitution parallèle en B événementielle impose l'absence de variables partagées dans les deux substitution qui sont en parallèle. Ce qui n'est pas forcément le cas des deux substitutions $x := F_1(\alpha, v) \parallel z := F_2(\alpha, w)$. Poppleton [68] propose sa propre définition de la substitution parallèle qui se reflète dans la garde et le prédicat avant-après de l'événement $e \odot f$[1], nous avons gardé dans cette définition les notations du papier original dans lequel $< xz >$ est l'ensemble des variables communes entre les ensembles x et z et $< x - z >$ est l'ensemble des variables qui sont dans x et pas dans z :

$$G_{e\odot f} \mathrel{\widehat{=}} \exists\, \alpha, \beta.(Q_1(\alpha, v) \ \wedge \ Q_2(\beta, w) \ \wedge \ F_1(\alpha, v) = F_2(\beta, w))$$
$$\begin{aligned} E_{e\odot f} \mathrel{\widehat{=}} \exists\, \alpha, \beta.(&Q_1(\alpha, v) \ \wedge \ Q_2(\beta, w) \ \wedge \\ &< x - xz >' = F_1(\alpha, v) \ \wedge < z - xz >' = F_2(\beta, w) \ \wedge \\ &xz' = F_1(\alpha, v) \ \wedge \ xz' = F_2(\beta, w) \\ &y' = y \ \wedge \ a' = a \end{aligned}$$

Cette notation est quelque peu surprenante dans la mesure où F_1 et F_2 sont respectivement définis pour les ensembles de variables x et y. En effet, quand l'auteur écrit $F_1(\alpha, v) = F_2(\beta, w)$ dans la garde de $e \odot f$, il faut considérer uniquement les parties de F_1 et F_2 qui portent sur les variables communes.

Il est évident qu'il faut avoir assez de non-déterminisme dans les gardes des deux événements pour que l'événement résultant de la fusion ne soit pas toujours bloqué.

Le nouveau modèle ainsi défini vérifie certaines propriétés intéressantes prouvées dans [68]. La première propriété est le théorème suivant :

[1] Il faut noter que l'opérateur $\odot$ existait déjà dans les travaux de Back et de Butler [15].

Théorème 1 *L'événement du modèle* M *résultant de la fusion des événements* e *et* f *raffine chacun de ses événements dans leurs modèles respectifs.*

La deuxième propriété est la plus intéressante, elle concerne la préservation du raffinement par la fusion des événements. Cette propriété garantit que la fusion des événements concrets raffine la fusion des événements abstrait. Le modèle N a comme variables v_1 et v_2 et un événement $e_N(v_1, v_2)$ avec comme garde G_N et comme prédicat avant-après E_N. Le modèle P a comme variables v_2 et v_3 et un événement $e_P(v_2, v_3)$ avec comme garde G_P et comme prédicat avant-après E_P. v_2 est donc une variable partagée entre les deux modèles. Le modèle M résultant de la fusion de N et P a comme variables v_1, v_2 et v_3 et un événement $e_M(v_1, v_2, v_3) = e_N \odot e_P$ avec comme garde G_M et comme prédicat avant-après E_M. Les modèles N_R et P_R raffinent respectivement N et P. NR a comme variable w_1 et w_2 et un événement $e_{NR}(w_1, w_2)$. e_{NR} raffine e_N avec l'invariant de collage suivant :

$$J(v_1, w_1, w_2) \ \wedge \ v_2 = h(w2)$$

Il en est de même pour le modèle PR a comme variable w_2 et w_3 et un événement $e_{PR}(w_2, w_3)$. e_{PR} raffine e_P avec l'invariant de collage suivant :

$$K(v_3, w_3, w_2) \ \wedge \ v_2 = h(w2)$$

Il faut noter la limitation qui consiste à raffiner les variables partagées de la même manière dans les deux modèles à fusionner. Ceci est illustré dans les deux invariants de collage avec $v_2 = h(w2)$. [68] a prouvé que le modèle MR qui a comme variables w_1, w_2, w_3 et comme événement $e_{MR}(w_1, w_2, w_3) = e_{NR} \odot e_{PR}$ résultant de la fusion des modèles concrets NR et PR raffine le modèle M avec comme invariant de collage :

$$J(v_1, w_1, w_2) \ \wedge \ v_2 = h(w2) \ \wedge \ K(v_3, w_3, w_2)$$

Cette technique qui consiste à raffiner les variables partagées de la même manière est inspirée de la méthode de décomposition de modèles d'Abrial.

3.5 Conclusion

Nous avons exposé dans ce chapitre les principales techniques de raffinements utilisées dans la méthode B événementielle. Nous nous sommes focalisé sur les techniques basées sur la décomposition et la composition des modèles. Nous utiliserons ces techniques pour proposer une technique de composition de protocoles de sécurité, la technique de décomposition de modèle d'Abrial sera utilisée telle quelle par contre la technique basée sur la fusion des événements ne correspondait pas tout à fait à nos besoins, nous l'avons donc adaptée à notre problématique. Cette adaptation de la technique de fusion d'événements sera discutée dans le prochain chapitre qui contient également quelques aspects méthodologiques de relatifs à nos modèles B événementiels.

Chapitre 4

La méthode B événementielle : Propositions

4.1 Introduction

Nous utilisons dans le cadre de cette thèse la méthode B événementielle comme un outil de modélisation et de vérification. Tout au long du travail effectué, certains problèmes rencontrés lors du processus de modélisation et de preuve nous ont poussé à proposer de nouveaux outils pour cibler des problèmes particuliers. Nous croyons que certains problèmes rencontrés en traitant des problèmes de sécurité avec la méthode B événementielle, ne sont pas propres à ce sujet précis mais peuvent être rencontrés dans le cadre d'autres études de cas utilisant la méthode B événementielle. Nous faisons dans ce chapitre des propositions concernant la génération d'invariants et sur la composition de modèles avec fusion des événements.

Ainsi, un des problème récurrents des utilisateurs de la méthode B événementielle est la génération d'invariants, notamment les invariants de collage. Au cours de notre travail réalisé sur les protocoles cryptographiques et sur le contrôle d'accès, nous avions comme souci permanent l'automatisation de la modélisation ainsi que de la preuve de ces modèles ce qui implique l'obtention des invariants de collage de manière automatique. La génération d'invariants est une thématique traitée abondamment dans la littérature mais très peu de travail a été réalisé dans ce domaine dans le cadre de la méthode B événementielle.

4.2 La génération d'invariants

Il existe dans la littérature plusieurs méthodes de génération d'invariants pour les systèmes de transitions. Nous avons utilisé dans cette thèse la méthode dite de renforcement de l'invariant. Une des méthodes utilisées pour la génération d'invariants pour les système de transitions consiste à partir d'une propriété initiale censée être vérifiée sur tout le système et de la renforcer petit à petit pour arriver à une propriété qui soit inductive. Cependant le choix de la façon avec laquelle l'invariant est renforcé n'est pas évident, s'il est faiblement renforcé le processus de recherche d'invariant peut s'éterniser voir même être infini, si au

contraire l'invariant est trop renforcé, on peut arriver à une propriété qui n'est plus vérifiée dans tous les états du système.

Manna et Pnueli [62, 25] ont proposé deux méthodes de renforcement d'invariants, la première est basée sur le calcul de la *plus faible pré-condition* et la deuxième sur le calcul de la *plus forte post-condition*. Ils ont présenté leurs méthodes de façon générale sur des systèmes de transitions $(\mathcal{V}, \Theta, \mathcal{T})$, où :

- $\mathcal{V}$ est l'ensemble des variables.
- Θ est une condition initiale.
- $\mathcal{T}$ est l'ensemble des transitions possibles.

L'ensemble des états (interprétation de $\mathcal{V}$) est noté Σ, un langage de prédicats du premier ordre $\mathcal{A}$ est défini sur l'ensemble des variables $\mathcal{V}$. La condition initiale Θ est un prédicat de ce langage. Une transition t lie chaque état s (avec $s \in \Sigma$) à un ensemble de successeurs $t(s)$ avec $t(s) \subseteq \Sigma$. À chaque transition t est associée une assertion $\rho_t(x, x')$ appelée *relation de transition* qui lie les valeurs des variables x dans un état s à leurs valeurs x' dans un état s' successeur de s avec la transition t ($s' \in t(s)$).

Manna et Pnueli utilisent pour le calcul des invariants les concepts de *plus faible pré-condition* et de *plus forte post-condition* introduit par Dijkstra [43], ce sont des transformateurs de prédicats qui calculent, pour une propriété $P(x)$ telle que $P(x) \in \mathcal{A}$, et une transition t données, deux nouveaux prédicats $WP(t, P)(x)$ et $SP(t, P)(x)$. Intuitivement, la signification de la plus faible pré-condition $WP(t, P)(x)$ est qu'à partir d'un état satisfaisant cette propriété, on ne peut aller avec les transition contenues dans t que vers des états vérifiant la propriété $P(x)$. La définition du transformateur de prédicat $WP(t, P)$ est souvent donnée dans la littérature en utilisant un autre transformateur de prédicat $pred(t, P)(x)$ qui calcule un prédicat qui est satisfait par l'ensemble des états du système à partir desquels, avec les transition t on obtient au moins un état satisfaisant $P(x)$ (les prédécesseurs de $P(x)$ avec les transitions t) :

Définition 1 $pred(t, P)(x) \mathrel{\widehat{=}} \exists\ x'.P(x') \wedge \rho_t(x, x') \in t$

La définition de la plus faible pré-condition est comme suit :

Définition 2 $WP(t, P)(x) = \neg pred(t, \neg P)(x)$

On peut également la définir directement :

Définition 3 $WP(t, P)(x) \mathrel{\widehat{=}} \forall\ x'.\rho_t(x, x') \Rightarrow P(x')$

$SP(t, P)$ contient l'ensemble des états successeurs de $P(x)$ avec les transitions t :

Définition 4 $SP(t, P)(x) \mathrel{\widehat{=}} \exists\ x_0.\rho_t(x_0, x) \wedge P(x_0)$

Ces opérateurs présentent certaines propriétés intéressantes, pour une transition t et deux propriétés P_1 et P_2 données :

$$WP(t, P_1 \wedge P_2)(x) \Leftrightarrow WP(t, P_1)(x) \wedge WP(t, P_2)(x)$$

$$SP(t, P_1 \vee P_2)(x) \Leftrightarrow SP(t, P_1)(x) \vee SP(t, P_2)(x)$$

Les deux opérateurs sont monotones : pour P_1, P_2 et une transition t telles que $P_1(x) \Rightarrow P_2(x)$ alors $WP(t, P_1)(x) \Rightarrow WP(t, P_2)(x)$ et $SP(t, P_1)(x) \Rightarrow SP(t, P_2)(x)$.

Nous utiliserons plus loin la notation suivante :

$$WP(\mathcal{T}, P)(x) \Leftrightarrow \bigwedge_{t \in \mathcal{T}} WP(t, P)(x)$$

De façon générale, le problème de recherche d'invariants se pose de la façon suivante : nous voulons montrer que notre système satisfait une propriété P donnée, pour celà il nous faut trouver un invariant inductif I satisfaisant les propriétés suivantes :

1. I implique P.
2. I est vérifié par tous les états initiaux de S $(\Theta \Rightarrow I)$.
3. I est maintenu par toutes les transitions $\mathcal{T}$, cette condition peut être exprimée en utilisant les notions de plus faible pré-condition ou de plus forte post-condition :
   - $I \Rightarrow WP(\mathcal{T}, I)$
   - $SP(\mathcal{T}, I) \Rightarrow I$

Pour trouver l'invariant inductif dans un système de transition $(\mathcal{V}, \Theta, \mathcal{T})$, [62] propose deux techniques, la *propagation en avant* (*Forward propagation* ou encore *bottom up appraoch*) et la *propagation en arrière* (*Backward propagation* ou encore *top down approach*).

La propagation en avant Cette technique est basée sur l'opérateur SP, elle permet d'obtenir l'invariant inductif le plus fort du système de transition. Un opérateur $\mathcal{F}$ est introduit, il est définit comme suit, pour une propriété X on a :

$$\mathcal{F}(X) = \Theta \vee SP(\mathcal{T}, X)$$

[62] montre que F est monotone, i.e., pour deux propriétés X_1, X_2 telles que $X_1 \Rightarrow X_2$ alors $\mathcal{F}(X_1) \Rightarrow \mathcal{F}(X_2)$ et que le plus fort invariant inductif du système I_F *est le plus petit point fixe de l'opérateur* $\mathcal{F}$.

$\mathcal{F}$ étant monotone, si la suite suivante :

$$FALSE \rightarrow \langle S_O = \Theta \rangle \rightarrow \langle S_1 = \mathcal{F}(S_0) \rangle \rightarrow \langle S_2 = \mathcal{F}(S_1) \rangle \rightarrow \ldots$$

converge dans un nombre fini d'étapes, i.e., $S_n = \mathcal{F}(S_n)$ alors le plus fort invariant inductif du système I_F est égal à S_n.

La propagation en arrière Cette technique est basée sur l'opérateur WP, elle permet d'obtenir l'invariant inductif le plus faible du système de transition en partant d'une propriété P donnée que l'on voudrait prouver sur l'ensemble des états du système. Un opérateur $\mathcal{B}$ est introduit, il est définit comme suit, pour une propriété X on a :

$$\mathcal{B}(X) = P \wedge WP(\mathcal{T}, X)$$

[62] montre que $\mathcal{B}$ est monotone, et que le plus faible invariant inductif du système impliquant la propriété P noté I_B *est le plus grand point fixe de l'opérateur* $\mathcal{B}$.

$\mathcal{B}$ étant monotone, si la suite suivante :

$$TRUE \leftarrow \langle P \rangle \leftarrow \langle P_1 = \mathcal{B}(P) \rangle \leftarrow \langle P_2 = \mathcal{B}(P_1) \rangle \leftarrow \ldots$$

converge dans un nombre fini d'étapes, i.e., $P_n = \mathcal{B}(P_n)$ alors le plus faible invariant inductif du système I_B est égal à P_n.

Les invariants inductifs I_F et I_B pour un système de transition et une propriété P donnés vérifient les propriétés suivantes :

- $\Theta \Rightarrow I_F$
- $I_F \Rightarrow I_B$
- $I_B \Rightarrow P$
- P est une propriété de sûreté.

Ces deux techniques reposent sur la convergences des suites permettant d'obtenir les invariants inductifs, dans la pratique, cette convergence n'est pas toujours facile à obtenir notamment dans le cas de la présence de boucles. Des heuristiques ont été proposées pour obtenir des méthodes de renforcement d'invariant efficace pour des systèmes particuliers.

Souvent, lorsque les utilisateurs de la méthode B événementielle échouent à faire une preuve d'invariance, ils rajoutent dans l'invariant une partie ou la totalité de l'obligation de preuve qu'ils n'arrivent pas à décharger. Ceci n'est autre que la technique de propagation en arrière exposée plus haut. Une machine B événementielle peut être interprétée comme un système de transition $(\mathcal{V}, \Theta, \mathcal{T})$, $\mathcal{V}$ étant les variables du modèle, Θ l'initialisation et les transitions dans $\mathcal{T}$ correspondent aux événements dont les prédicats avant-après $BA(x, x')$ sont les relations de transitions $\rho_t(x, x')$. Le langage de prédicats $\mathcal{A}$ définit sur $\mathcal{V}$ est dans ce cas la logique du premier ordre avec la théorie des ensembles. On peut donc adapter les techniques de recherche d'invariant vues plus haut pour la méthode B événementielle.

La démarche est intéressante dans la mesure où l'outil RODIN utilise le calcul des plus faibles pré-conditions pour la génération des obligations de preuves. En effet, les différentes obligations de preuves d'un modèle B vues dans le chapitre 2 sont exprimées à l'aide des prédicats avant-après mais il est également possible de le faire à l'aide de la plus faible pré-condition. Dans la méthode B événementielle, la plus faible pré-conditions pour un événement e et une propriété P est notée $[e](P)$, les définition des opérateurs *pred*, *WP* sont dans le cas de la méthode B événementielle :

Définition 5 $pred[e](P)(x) \mathrel{\widehat{=}} \exists\ x'.\ P(x') \wedge BA_e(x, x') \in t$

La définition de la plus faible pré-condition est comme suit :

Définition 6 $[e](P) = \neg pred[e](\neg P)$

On peut également la définir directement :

Définition 7 $[e](P)(x) \mathrel{\widehat{=}} \forall\ x'.\ BA_e(x, x') \Rightarrow P(x')$

Par exemple, pour prouver que $I(x)$ est un invariant d'une machine abstraite, il faut prouver pour tous les événements e du modèle que :

$$I(x) \Rightarrow [e]I(x)$$

Ceci est équivalent à :

$$I(x)\ \wedge\ BA_e(x, x')\ \Rightarrow\ I(x')$$

L'outil RODIN génère les différentes obligations de preuves en utilisant le calcul de la plus faible pré-condition, nous pouvons donc utiliser les obligations générées pour la recherche d'invariant. Pour un modèle contenant les événements e_1, e_2, ..., e_n, pour une propriété de sûreté P donnée, l'opérateur de prédicats $\mathcal{B}$ est dans le cas de la méthode B défini comme suit :

$$\mathcal{B}(X) = P \wedge \bigwedge_{i=1}^{n} [e_i](X)$$

On peut donc essayer d'obtenir l'invariant inductif en calculant la suite :

$$TRUE \leftarrow \langle P \rangle \leftarrow \langle P_1 = P \rangle \leftarrow \langle P_2 = \mathcal{B}(P_1) \rangle \leftarrow \ldots$$

Si à une étape k donnée, nous obtenons une assertion intermédiaire P_k et que celle-ci s'avère non inductive, alors il existe un ensemble d'événements $fail_k$ dont la preuve de préservation d'invariance a échoué pour la propriété P_k. Pour le reste des événements e_i avec $e_i \notin fail_k$ on a $P_k \Rightarrow [e_i]P_k$. L'opérateur $\mathcal{B}$ est égal à :

$$\mathcal{B}(P_k) = P \wedge \bigwedge_{e \in fail_k} [e](P_k) \wedge \bigwedge_{e \notin fail_k} [e](P_k)$$

Puisque $\mathcal{B}(P_k) \Rightarrow P_k$ et que pour $e_i \notin fail_k$ on a $P_k \Rightarrow [e_i]P_k$ il n'est pas difficile de prouver que :

$$\boxed{\mathcal{B}(P_k) = P \wedge \textstyle\bigwedge_{e \in fail_k} [e](P_k)}$$

Il suffit donc de renforcer l'invariant avec uniquement les plus faibles pré-conditions des événements dont la preuve a échoué. Il est cependant possible d'optimiser la démarche de la façon suivante, si à une étape k et pour un événement e_j l'obligation de preuve, $P_k \Rightarrow [e_j]P_k$ a échoué et que celle-ci est de la forme :

$$P_k \Rightarrow (oblig_{j1} \wedge oblig_{j2})$$

Avec $[e_j]P_k = (oblig_{j1} \wedge oblig_{j2})$ et qu'on arrive à prouver que $P_k \Rightarrow oblig_{j1}$ mais pas $P_k \Rightarrow oblig_{j2}$ alors il suffit de renforcer l'invariant avec $oblig_{j2}$. Il est en effet facile de prouver que

$$\mathcal{B}(P_k) = P \wedge \bigwedge_{e \in fail_k \wedge e \neq e_j} [e](P_k) \wedge oblig_{j2}$$

En d'autres termes, on ne renforce l'invariant que par des parties d'obligations de preuves non déchargées. Cette optimisation permet d'éviter de réinjecter des fragments d'invariants inutilement et donc d'éviter des obligations de preuves supplémentaires. La forme des événements et notamment l'utilisation des surcharges de fonctions fait que l'on a très souvent des obligation de preuves de la forme $P_k \Rightarrow (oblig_{j1} \wedge oblig_{j2})$.

Souvent les utilisateurs de la méthode B événementielle et de l'outil RODIN utilisent un model-checker pour tester un invariant qu'ils viennent d'ajouter avant d'entamer les preuves d'invariance. Dans le cas où ils découvrent une trace violant l'invariant en question ils en concluent que l'invariant est faux mais ils

ne peuvent rien dire de la propriété initiale P qu'ils essaient de prouver. L'utilisation de la plus faible pré-condition pour la génération d'invariant présente un avantage qui est que si l'on trouve une trace violant un des invariants qui a été ajouté avec le calcul de la plus faible pré-condition alors il est possible de reconstituer une trace violant la propriété P. Nous discutons de cet aspect dans la prochaine sous-section.

4.2.1 La reconstitution des traces violant une propriété

La technique utilisée pour la génération automatique d'invariant basée sur la plus faible pré condition permet de reconstituer d'éventuelles traces violant la propriété désirée P, cela nous a été utile dans le cas des protocoles cryptographiques où nous avons non seulement montré qu'une propriété était fausse mais nous avons également reconstitué les attaques possibles. Par définition, une plus faible pré-condition d'une propriété P et un événement e contient tous les états à partir desquels, après déclenchement de l'événement e, on se retrouve dans un état satisfaisant la propriété P. Ainsi, la négation de la plus faible pré condition qui est égale d'après la définition 6 à :

$$\neg[e]P = pred[e](\neg P)$$

contient les états qui, après déclenchement de e, mènent vers au moins un état ne satisfaisant pas la propriété P. Ceci implique que si on arrive à obtenir une trace d'événements e_1, e_2, ..., e_m partant de l'initialisation du système et menant vers un état vérifiant $\neg[e]P$ alors il suffit de prolonger cette trace par l'événement e pour obtenir une trace e_1, e_2, ..., e_m, e démarrant de l'initialisation et finissant dans au moins un état qui ne satisfait pas la propriété P.

De façon générale, pour un ensemble d'événements e_1, e_2, ..., e_m et une propriété P donnée :

Théorème 2 *S'il existe un état du système x_0 tel que :*

$$Init(x_0) \Rightarrow \neg([e_1]([e_2](\ldots[e_{m-1}]([e_m](P)))))(x_0)$$

alors il existe un ensemble d'états x1, x2, ..., x_m tels que :
- $BA_{e1}(x_0, x_1) \wedge BA_{e2}(x_1, x_2) \wedge \ldots \wedge BA_{em}(x_{m-1}, x_m) \wedge$
- $\neg P(x_m)$

La preuve de ce théorème est obtenue par récurrence avec les définitions 6 et 5. Dans la pratique, ce théorème implique que pendant le processus de génération d'invariant, si un fragment d'invariant qui vient d'être ajouté par le calcul des plus faibles pré-conditions n'est pas maintenu par l'initialisation du système, il est possible de reconstituer une attaque. Il faut cependant annoter chaque fragment d'invariant ajouté par l'événement ainsi que la partie de l'invariant dont il est la plus faible pré-condition pour pouvoir ensuite remonter jusqu'à la propriété P qui a servi à générer l'invariant.

Optimisation par le Model-Checking Les concepteurs de modèles avec la méthode B événementielle choisissent comme invariant une propriété désirée et la renforcent ensuite pour obtenir un invariant qui soit inductif. Ils utilisent souvent le modèle checking à chaque fois qu'il rajoutent un invariant dans leur

système pour le vérifier sur un ensemble limité de valeurs avant d'essayer de faire les obligations de preuve correspondantes. En cas d'échec (quand le Model-Checker trouve une trace ne vérifiant pas l'invariant rajouté), le concepteur en conclut que l'invariant qui vient d'être ajouté n'est pas vérifiable sur l'ensemble des états atteignables du système mais ne nous dit rien sur la propriété de départ, en effet il se peut que l'invariant rajouté soit trop fort pour être inductif mais que la propriété de départ soit néanmoins correcte.

En renforçant l'invariant avec la technique basée sur la plus faible pré-condition, ce problème ne se pose pas. En effet, on peut compléter les traces trouvées par le Model-Checker qui ne vérifient pas un invariant donné pour trouver une trace ne vérifiant pas la propriété P.

Théorème 3 *Si pour un ensemble d'événements e_1, e_2, ..., e_m et une propriété P, il existe un ensemble d'événements Init, e'_1, e'_2, ..., e'_n et un ensemble d'états x'_0, x'_1, x'_2, ..., x'_n tels que :*

- $Init(x'_0)$
- $BA_{e'1}(x'_0, x'_1) \wedge BA_{e'2}(x'_1, x'_2) \wedge \ldots \wedge BA_{e'n}(x'_{n-1}, x'_n) \wedge$
- $\neg([e_1]([e_2](\ldots[e_{m-1}]([e_m](P)))))(x'_n)$

alors il existe un ensemble d'états x1, x2, ..., x_m tels que :

- $BA_{e1}(x'_n, x_1) \wedge BA_{e2}(x_1, x_2) \wedge \ldots \wedge BA_{em}(x_{m-1}, x_m) \wedge$
- $\neg P(x_m)$

Alors la trace *Init*, e'_1, e'_2, ..., e'_n, e_1, e_2, ..., e_m mène vers au moins un état ne satisfaisant pas la propriété P. En d'autre termes, en trouvant un contre-exemple à un invariant donné, on obtient un contre-exemple pour la propriété P. La preuve de ce théorème est également obtenue par induction avec les définitions 6 et 5. En plus de la reconstitution de traces violant les propriétés de sûretés, la génération d'invariants par la plus faible pré-condition est également intéressante dans le cadre de la composition des modèles, cet aspect sera discuté dans la section 4.3.1.

4.3 La composition des modèles

La composition ou (décomposition) des modèles B événementielle est une des notions clés de notre travail. Que ce soit pour la cryptographie ou le contrôle d'accès, nous avons tenu a structurer nos modèles en mécanismes capables d'être composés pour arriver à des systèmes complexes. L'intérêt de la démarche de composition n'est plus à démontrer dans le domaine de l'ingénierie des systèmes. Des travaux convergeant dans ce sens ont été entrepris dans le cadre de la méthode B événementielle, nous avons présenté dans le chapitre 3 la méthode de décomposition de modèle proposée par Abrial. Les travaux d'Abrial ont inspiré d'autres travaux comme la méthode de composition de modèles basée sur la fusion d'événements [68] que nous avons également présentée dans le même chapitre. La notion de fusion d'événements nous a particulièrement intéressé dans la mesure ou nous manipulons des modèles B événementielle de systèmes dans lesquels se déroulent des transactions sur plusieurs étapes. La figure 4.1 montre un exemple typique du cas d'une composition de deux systèmes : le premier garantit une propriété P_1 à l'issue de la transaction tandis que le deuxième garantit une propriété P_2. Le système résultant de la composition est censé

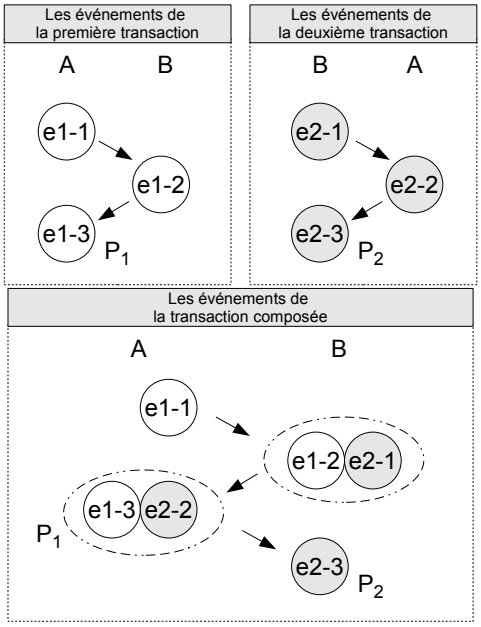


FIG. 4.1 – Un exemple de composition de transactions

garantir les deux propriétés à des instants particuliers, la fusion des événements permet ainsi de synchroniser l'exécution des deux transactions pour obtenir une transaction plus complexe.

La technique de fusion de modèles telle que proposée par Poppleton [68] est donc intéressante pour notre travail mais nous pose cependant quelques problèmes et ne correspond pas parfaitement à nos besoins en matière de composition. La raison essentielle est l'absence de la notion d'événements externes comme dans la technique d'Abrial. En effet, Poppleton dit dans son papier présentant la technique de fusion [68] en la comparant à la technique d'Abrial que «*Dans notre schéma, la fusion d'événements les privent de leur indépendance de comportement, il n'y a donc plus besoin d'événement externes*». En d'autres termes, la nature même de l'opérateur $\odot$, garantit que les deux événements fusionnés se comportent de la même manière sur les variables partagées chacun de son coté et il n'y a donc pas de problème au moment de la fusion. L'auteur reconnait lui même que cette façon de faire nécessite assez de non-déterminisme dans les actions des événements dans [68] : «*La fusion de deux modèles requiert suffisamment de non-déterminisme dans les actions des événements fusionnés sur les variables partagées pour que l'événement fusionné soit faisable. Une façon naturelle de réaliser celà est comme suit, l'événement* $e(v_1, v_2, v_3)$ *affecte à* v_1 *la valeur* $F_1(\alpha, v_1, v_2, v_3)$ *pour une certaine valeur de* α*, affecte à* v_2 *une valeur quelconque de type* V_2 *et ne change pas la valeur de* v_3*. L'événement* $f(v_1, v_2, v_3)$*, quand à lui, affecte à* v_1 *une valeur quelconque de type* V_1*, affecte à* v_2 *la valeur* $F_2(\beta, v_1, v_2, v_3)$ *pour une certaine valeur de* β*, et ne change pas la valeur de* v_3». Cette façon de faire à pour conséquence que la fusion est soit

possible si elle satisfait les conditions de l'opérateur $\odot$ soit impossible et dans ce cas l'événement résultant de la fusion n'est pas faisable.

Dans notre cas, nous fusionnons deux modèles de protocoles de sécurité pour obtenir d'autres protocoles d'une forme particulière désirée. Le protocole que nous voulons obtenir nous guide pour déterminer quels événements du premier modèle sont fusionnés avec quels événements du second modèle. Dans la figure 4.1, pour obtenir le protocole composé avec cette forme-là, il faut fusionner les événements *e1-2* avec *e2-1*, *e1-3* avec *e2-2*, *e1-1* avec *skip* et *e2-3* avec *skip*. Le problème est que ces événements ne satisfont pas toujours les conditions de la fusion des événements relatives à la présence de variables partagées. Nous avons donc entrepris une démarche qui vise à rendre deux événements *fusionables* par l'opérateur $\odot$, cette démarche est inspirée des événements externes utilisés par Abrial pour la décomposition des modèles. Rendre deux événements *fusionables* signifie obtenir par la fusion un événement qui soit faisable.

Supposons que l'on ait deux événements e et f que l'on désire fusionner appartenant respectivement à deux modèles M_1 et M_2. Le modèle M_1 a comme variables v_1, v_2 et v_3 tandis que le modèle M_2 a comme variables v_2, v_3 et v_4. Les variables v_2 et v_3 sont donc partagées entre les deux modèles et sont susceptibles de poser problème au moment de la fusion.

Considérons un premier cas de figure, l'événement e modifie les variables v_1, v_2 et v_3 et l'événement f modifie les variables v_3 et v_4.

EVENT e
WHEN
$\mathtt{G_e}(\mathtt{v_1}, \mathtt{v_2}, \mathtt{v_3})$
THEN
$\mathtt{v_1}, \mathtt{v_2}, \mathtt{v_3} :| \mathtt{F_e}(\mathtt{v_1}, \mathtt{v_2}, \mathtt{v_3}, \mathtt{v'_1}, \mathtt{v'_2}, \mathtt{v'_3})$
END

EVENT f
WHEN
$\mathtt{G_f}(\mathtt{v_3}, \mathtt{v_4})$
THEN
$\mathtt{v_3}, \mathtt{v_4} :| \mathtt{E_f}(\mathtt{v_3}, \mathtt{v_4}, \mathtt{v'_3}, \mathtt{v'_4})$
END

On suppose que la variable partagée v_3 ne pose pas de problème pour la fusion, il reste le problème de la variable v_2 qui est partagée des deux modèles, modifiée par e mais pas par f. La fusion par l'opérateur $\odot$ n'est pas possible dans ce cas. Pour rendre l'événement f *fusionable* avec e nous simulons les actions de l'événement e sur la variable v_2 dans l'événement f. Nous utilisons pour cela les prédicat ext_G_e et ext_E_e dont la définition est similaire à ceux définis par Abrial dans le cas des événements externes.

EVENT e
WHEN
$\mathtt{G_e}(\mathtt{v_1}, \mathtt{v_2}, \mathtt{v_3})$
THEN
$\mathtt{v_1}, \mathtt{v_2}, \mathtt{v_3} :| \mathtt{E_e}(\mathtt{v_1}, \mathtt{v_2}, \mathtt{v_3}, \mathtt{v'_1}, \mathtt{v'_2}, \mathtt{v'_3})$
END

EVENT f
WHEN
$\mathtt{G_f}(\mathtt{v_3}, \mathtt{v_4})$
$\mathtt{ext\_G_e}(\mathtt{v_2})$
THEN
$\mathtt{v_3}, \mathtt{v_4} :| \mathtt{E_f}(\mathtt{v_3}, \mathtt{v_4}, \mathtt{v'_3}, \mathtt{v'_4})$
$\mathtt{v_2} :| \mathtt{ext\_E_e}(\mathtt{v_2}, \mathtt{v'_2})$
END

Si l'événement f est l'événement *skip* (si l'on souhaite fusionner e avec *skip*) alors on rajoute dans le modèle M_2 un nouvel événement externe ext_e qui

a comme garde ext_G_e et comme prédicat avant-après ext_E_e simulant les actions de e sur les variables partagées v_2 et v_3. La fusion de e avec ext_e ne pose pas de problème.

Le deuxième cas de figure que nous avons utilisé lors de nos travaux sur les protocoles de sécurité est comme suit. On suppose que la variable partagée v_3 ne pose pas de problème pour la fusion. La variable partagée v_2 est un ensemble auquel l'événement e ajoute un élément α et l'événement f un élément β.

EVENT e
ANY
α
WHERE
$\mathtt{G_e}(\alpha, \mathtt{v_1}, \mathtt{v_2}, \mathtt{v_3})$
THEN
$\mathtt{v_1}, \mathtt{v_3} :| \mathtt{E_e}(\mathtt{v_1}, \mathtt{v_3}, \mathtt{v'_1}, \mathtt{v'_3})$
$\mathtt{v'_2} := \mathtt{v_2} \cup \{\alpha\}$
END

EVENT f
ANY
β
WHERE
$\mathtt{G_f}(\beta, \mathtt{v_2}, \mathtt{v_3}, \mathtt{v_4})$
THEN
$\mathtt{v_3}, \mathtt{v_4} :| \mathtt{E_f}(\mathtt{v_3}, \mathtt{v_4}, \mathtt{v'_3}, \mathtt{v'_4})$
$\mathtt{v'_2} := \mathtt{v_2} \cup \{\beta\}$
END

Dans ce cas la fusion n'est possible que si les valeurs de α et β sont les mêmes ce qui restreint considérablement le nouvel événement obtenu par la fusion. Pour remédier au problème nous modifions les événements e et f de façon à simuler les actions effectuées sur chacun d'entre eux sur la variable v_2 dans l'autre événement comme suit :

EVENT e
ANY
α
β
WHERE
$\mathtt{G_e}(\alpha, \mathtt{v_1}, \mathtt{v_2}, \mathtt{v_3})$
$\mathtt{ext\_G_f}(\beta, \mathtt{v_2}, \mathtt{v_3})$
THEN
$\mathtt{v_1}, \mathtt{v_3} :| \mathtt{E_e}(\mathtt{v_1}, \mathtt{v_3}, \mathtt{v'_1}, \mathtt{v'_3})$
$\mathtt{v'_2} := \mathtt{v_2} \cup \{\alpha, \beta\}$
END

EVENT f
ANY
α
β
WHERE
$\mathtt{G_f}(\beta, \mathtt{v_2}, \mathtt{v_3}, \mathtt{v_4})$
$\mathtt{ext\_G_e}(\alpha, \mathtt{v_2}, \mathtt{v_3})$
THEN
$\mathtt{v_3}, \mathtt{v_4} :| \mathtt{E_f}(\mathtt{v_3}, \mathtt{v_4}, \mathtt{v'_3}, \mathtt{v'_4})$
$\mathtt{v'_2} := \mathtt{v_2} \cup \{\alpha, \beta\}$
END

Le nouvel événement f devient *fusionable* avec l'événement e mais ceci n'est pas sans conséquences sur les preuves des deux modèles. En effet, nous modifions les événements e et f en ajoutant des comportements nouveaux dans chacun de ces événements afin de simuler les comportements de l'autre événement. Ces comportements nouveaux ne maintiennent pas forcément les invariants des modèles auxquels appartiennent les événements modifiés d'où l'apparition de nouvelles obligations de preuves à décharger. Ces obligations de preuves garantissent que la fusion des deux modèles se déroule correctement. La sous-section suivante traite le problème de ces nouvelles obligations de preuves.

4.3.1 Obligations de preuves et composition

Un aspect important du processus de composition des modèles est le fait de pouvoir réutiliser les preuves déjà faites dans chacun des modèles et de ne pas avoir à les refaire au moment de la composition où l'on doit juste se contenter de faire les preuves de la composition. Les modifications faites sur les modèles pour les rendre composables ne doivent donc pas avoir de conséquence sur les obligations de preuves déjà déchargées dans ces modèles développés séparément, ceci est garanti grâce à l'utilisation de la plus faible pré-condition pour le calcul des invariants de chacun des modèles composés.

En effet, si on a obtenu comme invariant inductif pour le premier modèle un prédicat I_1 qui n'est pas l'invariant inductif le plus faible alors si les nouveaux comportements introduits sur ce modèle ne maintiennent pas cet invariant on ne peut rien conclure car il est possible qu'il existe un invariant I'_1 plus faible qui soit inductif pour le nouveau modèle avec les nouveaux comportements mais dans ce cas il faut refaire toutes les preuves pour ce nouvel invariant I'_1 pour l'ensemble des événements du système. Nous montrons en ce qui suit que l'utilisation du plus faible invariant inductif nous évite ce problème.

De façon générale, soit I le plus faible invariant inductif impliquant une propriété P dans un système de transition $(\mathcal{V}, \Theta, \mathcal{T})$. Nous avons vu précédemment que I est la limite (si elle existe) de la suite :

$$TRUE \leftarrow \langle P \rangle \leftarrow \langle P_1 = \mathcal{B}(P) \rangle \leftarrow \langle P_2 = \mathcal{B}(P_1) \rangle \leftarrow \dots$$

Supposons que l'on étende le système de transition $(\mathcal{V}, \Theta, \mathcal{T})$ avec un nouvel ensemble de transitions $\mathcal{T}'$ alors le nouvel plus faible invariant inductif est obtenu en calculant le plus grand point fixe de l'opérateur $\mathcal{B}'$ avec :

$$\mathcal{B}'(X) = P \wedge WP(\mathcal{T}' \cup \mathcal{T}, X)$$

Par les propriétés de la plus faible pré-condition, on a :

$$\mathcal{B}'(X) = P \wedge WP(\mathcal{T}, X) \wedge WP(\mathcal{T}', X)$$

Et donc

$$\mathcal{B}'(X) = \mathcal{B}(X) \wedge WP(\mathcal{T}', X)$$

Et le nouvel plus faible invariant inductif I' est calculé (s'il existe) avec la suite :

$$TRUE \leftarrow \langle P \rangle \leftarrow \langle P_1 = \mathcal{B}'(P) \rangle \leftarrow \langle P_2 = \mathcal{B}'(P_1) \rangle \leftarrow \dots$$

Qui en remplaçant $\mathcal{B}'$ par sa valeur donne :

$$TRUE \leftarrow \langle P \rangle \leftarrow \langle P_1 = \mathcal{B}(P) \wedge WP(\mathcal{T}', P) \rangle \leftarrow \langle P_2 = \mathcal{B}(P_1) \wedge WP(\mathcal{T}', P_1) \rangle \leftarrow \dots$$

Il n'est pas difficile de voir que l'invariant du modèle initial I est une partie de la conjonction définissant le nouvel invariant inductif I', en d'autres termes il existe un prédicat C tel que :

$$I' = I \wedge C$$

Cette propriété nous garantit de conserver les preuves d'invariance déjà faite sur I, on doit quand même décharger le reste des obligations de preuves qui sont relatives à la faisabilité de la composition. Nous pensons que cet aspect est particulièrement important dans le cadre d'une approche de décomposition *bottom-up* où les composants sont définis indépendamment de leurs environnement. Dans ce cas, les propriétés que l'environnement doit satisfaire en cas de composition doivent être aussi faibles que possible.

4.3.2 Fusion et raffinement

Poppleton [68] prouve que le raffinement est maintenu par fusion des événements par l'opérateur $\odot$. Nous utiliserons cette propriété lors de la composition des protocoles de sécurité. Les propriétés de sûreté sont spécifiées dans des modèles abstrait avec l'utilisation de transactions abstraites. Ce modèle abstrait contient les étapes d'une transaction modélisée de façon très abstraite (sans messages) ainsi que la propriété désirée. Le protocole est ensuite modélisé à l'aide d'un (ou plusieurs) modèles concrets contenant les détails de ce protocole et notamment les événements d'envoi de message comme le montre la figure 4.2. L'invariant de collage est obtenu avec la technique de la plus faible pré-condition.

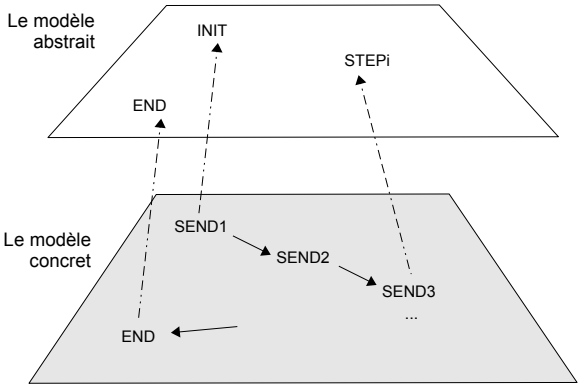


FIG. 4.2 – Les modèles abstrait et concret d'un protocole.

Les protocoles modélisés séparément sont ensuite composés à tous les niveaux d'abstraction, d'abord les spécifications sont composées pour obtenir une spécification de l'ensemble des propriétés garanties par les deux protocoles, ensuite les modèles concrets des protocoles sont composés pour obtenir le modèle du nouveau protocole en utilisant la technique exposée dans la section précédente consistant à rendre les modèles composables et à décharger les obligations de preuve de composition.

4.4 Conclusion

Nous avons présenté dans ce chapitre quelques propositions pour la méthode B événementielle concernant la génération d'invariant et la composition des modèles par fusion des événements. Les principales recommandations sont la possibilité de profiter des obligations de preuve générées par l'outil RODIN pour la génération d'invariant par la plus faible pré-condition, utiliser l'invariant ainsi obtenu pour reconstruire des traces d'événements violant une propriété de sûreté, rendre deux modèles *fusionables* en simulant dans chacun des modèles

les comportements de l'autre modèle sur les variables partagées. Nous avons également vu l'intérêt d'utiliser le plus faible invariant inductif pour la réutilisation des preuves lors de la composition des modèles. Ces techniques seront utilisées dans la suite de cette thèse

Deuxième partie

La composition des protocoles cryptographiques

Chapitre 5

Cryptographie : État de l'art

5.1 Introduction

Il est important de contrôler l'accès des données et aux ressources d'un système. Ce contrôle se fait notamment à l'aide de politiques de contrôle d'accès qui permettent de déterminer, si un utilisateur A a le droit ou non d'accéder à une ressource. Ces politiques ne garantissent cependant pas que la requête en question provienne effectivement de l'utilisateur A. En réalité, les politiques de contrôle d'accès permettent de déterminer si un utilisateur *qui prétend être* A a le droit ou non d'accéder à une ressource donnée. Déterminer si la requête vient effectivement de l'utilisateur A et non pas de quelqu'un qui veut se faire passer pour A est un problème d'*authentification*. Pour garantir la propriété d'authentification dans le système, on a recours à la cryptographie. Ceci est un exemple parmi d'autres où les protocoles cryptographiques sont utilisés. Un protocole cryptographique est une suite de messages échangés à travers des canaux de communication entre deux entités ou plus dont le but est d'assurer la communication tout en garantissant des propriétés de sûreté dans un système donné alors que les canaux de communication utilisés ne sont pas sûrs.

5.1.1 Un exemple simple de protocole

Nous présentons comme introduction un protocole cryptographique très simple. Les protocoles cryptographiques s'exécutent entre plusieurs entités, nous utiliserons dans la suite de cette thèse les lettres A (pour *Alice*), B (pour *Bob*) et C (pour *Charlie*) pour désigner les différents participants dans un protocole. Nous utiliserons la lettre I pour désigner l'*intrus* dans le système qui tentera de faire des actions malveillantes visant à compromettre la sécurité du système.

La figure 5.1 montre un protocole cryptographique simple. Le participant B envoie un *nonce* frais au participant A. Un nonce (de l'anglais *no more than once*) est un nombre aléatoire généré durant chaque nouvelle session d'un protocole cryptographique, le but étant d'éviter qu'un attaquant ne réutilise d'ancien messages pour leurrer les participants d'un protocole. Le participant A répond en renvoyant le nonce reçu avec l'identité de B le tout chiffré avec une clé K_{AB}. La clé K_{AB} n'est connue que par les agents A et B.

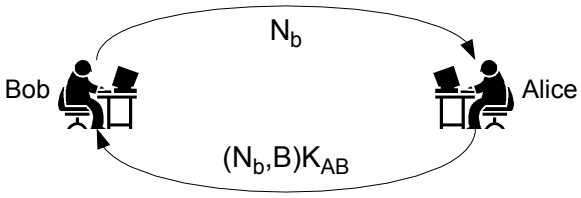


FIG. 5.1 – Un protocole cryptographique simple.

En envoyant son message, le participant A veut certifier à B qu'il est bien l'auteur du message en question. La forme du message est censée garantir son *authenticité*. En effet, le fait que le message soit chiffré par la clé K_{AB} connue uniquement par A et B est censé garantir que le message vient de A. Il y a d'autres propriétés autres que l'authenticité que peut garantir un protocole ; dans notre cas la présence du nonce N_b est censée garantir que ce message est une réponse au message envoyé par B et non pas un vieux message réutilisé par l'attaquant pour tromper B.

Nous utiliserons la notation classique suivante pour décrire les protocoles cryptographique dans cette thèse :

1. $B \rightarrow A : N_b$
2. $A \rightarrow B : \{N_b, B\}_{K_{AB}}$

Protocole 1: *Un protocole cryptographique simple*

Les protocoles utilisent des primitives cryptographiques, nous avons utilisé dans le cas de notre protocole simple un chiffrement à clé partagée, nous présentons dans la section suivante les principales primitives cryptographiques.

5.2 Les primitives cryptographiques

Les protocoles cryptographiques reposent sur un ensemble de techniques cryptographiques de base appelées aussi *primitives cryptographiques*. Les primitives les plus connues sont celles utilisées dans des standards comme *gnupg* [2] ou *openssl* [3]. Nous présentons ici les primitives les plus utilisées, notamment celles qui nous seront utiles dans la suite de cette thèse :

- Le chiffrement : Il y a deux façons de chiffrer des données : le chiffrement à clé secrète et le chiffrement à clé publique :
  1. Le chiffrement à clé secrète : cette méthode permet de crypter un texte M à l'aide d'une clé k pour obtenir un texte chiffré $\{M\}_k$. Certains modèles de preuves de protocoles cryptographiques considèrent une *cryptographie parfaite*, cela signifie que le texte M ne peut être obtenu à partir du texte chiffré sans posséder la clé k. Un exemple de chiffrement à clé secrète est le chiffrement «DES» (Data Encryption

Standard) [24]. Le chiffrement à clé secrète est aussi appelé chiffrement symétrique car le message est chiffré et décrypté par la même clé, il est aussi désigné par chiffrement à clé privée.

2. Le chiffrement à clé publique : contrairement au chiffrement à clé secrète, la clé utilisée pour chiffrer un message dans le chiffrement à clé publique est différente de celle utilisée pour le déchiffrer. Chaque participant A possède deux clés : une publique et une privée. La clé de chiffrement est publique, elle est connue par tout le monde qui peut ainsi chiffrer des messages pour un participant donné qui sera le seul à pouvoir les déchiffrer grâce à l'autre clé qui est privée. La primitive de chiffrement à clé publique la plus connue est RSA[1] [70]. On note $E_A(M)$ un message M chiffré avec la clé publique de A. Le chiffrement à clé publique a des avantages et des inconvénients par rapport au chiffrement à clé privée : d'un côté il est plus pratique car on manipule moins de clés que le chiffrement à clé privée où chaque participant partage une clé avec un autre participant. Par contre il est plus coûteux en temps de calcul. Ces avantages et inconvénients de chaque méthode par rapport à l'autre font que ces deux primitives sont utilisées de manières différentes afin de profiter de leurs avantages et sans subir leurs inconvénients : en général on utilise le chiffrement à clé publique pour établir une clé secrète utilisée temporairement entre deux participants. Le chiffrement à clé publique est aussi appelée chiffrement asymétrique.

- Signature : cette primitive repose sur le même principe que le chiffrement asymétrique dans la mesure où chaque participant dispose d'une clé privée ainsi que d'une clé publique. Un participant peut signer un message M avec sa clé privée, les autres participants peuvent vérifier si le message est vraiment signé par ce participant en y appliquant sa clé publique. Nous utilisons la notation $Sig_A(M)$ pour désigner la signature d'un message M par un participant A. La méthode RSA est également utilisée pour la signature des messages.
- L'échange de clé : ce type de primitive vise à arriver à partager une clé entre plusieurs participants, celle-ci est ensuite utilisée pour les communications suivantes. Le chiffrement à clé publique peut être utilisé comme moyen d'échange de clés, on retrouve néanmoins certaines primitives spécifiques à l'échange de clés dans beaucoup de protocoles. La primitive d'échange de clés la plus connue est celle de Diffie-Hellman. Dans le groupe $\mathbb{Z}_p^*$ où p est un grand nombre premier et a est un élément générateur de $\mathbb{Z}_p^*$ la primitive de Diffie-Hellman entre deux participants A et B se déroule comme suit : A choisit un nombre x et envoie à B a^x, B fait de même et choisit aléatoirement un nombre y et envoie à A a^y. Chacun des participants peut ensuite calculer de son côté a^{xy} grâce à la propriété de l'exponentiation modulaire $a^{xy} = (a^x)^y$ et $a^{xy} = (a^y)^x$. La valeur a^{xy} est un secret partagé entre A et B qu'ils peuvent utiliser pour établir une clé partagée. L'attaquant quant à lui possède a^x et a^y mais ne peut avoir le secret partagé.
- Les fonctions de hachage : ce type de fonction à deux caractéristiques particulières. La première est qu'elles sont résistantes à la pré-image, ce

[1] Du nom des trois personnes qui l'ont développée : Rivest, Shamir, et Adelman.

qui signifie que pour une fonction de hachage h, il est difficile de calculer un message M à partir de la valeur $h(M)$. La seconde caractéristique est qu'elles sont résistantes aux collisions, cela veut dire qu'il est difficile de trouver M_1 et M_2 tels que $h(M_1) = h(M_2)$.

5.3 La modélisation des protocoles cryptographiques

Afin de pouvoir prouver des propriétés sur les protocoles cryptographiques, il est indispensable de les modéliser. Pour modéliser les protocoles, il faut fixer certaines hypothèses de départ sur lesquelles il faudra plus tard se baser pour faire les preuves que le protocole respecte les propriétés désirées. Il existe deux modèles de protocoles cryptographiques :

- Le modèle de Dolev-Yao [44] : Dolev et Yao ont modélisé les protocoles dits en cascade dans [44]. Dans leurs modèles, ils ont supposé une cryptographie parfaite où l'on ne peut déchiffrer un texte chiffré que si on a la clé correspondante. Ce modèle est parfois appelé *modèle formel*. Généralement, la taille des messages n'est pas bornée, l'attaquant peut envoyer une infinité de messages et le nombre de sessions ou d'exécutions parallèles du protocole est illimité, ceci engendre un espace d'états à explorer infini. Pour montrer qu'un protocole respecte une propriété donnée, il faut fournir une preuve formelle que quelles que soient les actions de l'attaquant, la propriété désirée est maintenue pour le protocole.
- Le modèle calculatoire (de l'anglais *computational*) : dans ce modèle, les clés, messages en clair ou messages chiffrés, sont considérés comme des suites de bits et les primitives de chiffrements des algorithmes. L'attaquant est dans ce modèle une machine de Turing. D'après Abadi et Rogaway [7] : «Les bons protocoles sont ceux où les attaquants ne peuvent faire *quelque chose de mal* trop souvent et assez efficacement». Montrer qu'un protocole est bon dans le modèle calculatoire est étroitement lié au calcul des probabilités et à la puissance de calcul. Parmi les premiers travaux dans le modèle calculatoires on peut citer [82].

Le travail contenu dans cette thèse se situe dans le cadre du modèle formel. Dans la suite de ce chapitre nous nous focaliserons sur la présentation de ce modèle.

5.4 Les propriétés de sûreté

Les besoins en sécurité sont divers selon le type de l'application qui utilise le protocole cryptographique. Différentes propriétés de sécurité sont décrites dans la littérature, les plus importantes sont :

- La confidentialité des données : assurer que les données sensibles restent secrètes.
- L'intégrité des données : assurer que les données sensibles ne soient pas altérées.
- L'authentification de l'origine des données : cette propriété offre des garanties sur l'origine des données. Généralement il est admis que l'authentification inclut l'intégrité des données car une donnée altérée (accidentellement ou

par malveillance) n'est pas considérée comme authentique. Par contre le contraire n'est pas vrai, l'intégrité ne garantit pas l'authenticité.
- La protection du matériel cryptographique : assurer, par exemple, la confidentialité ou fraicheur des clé partagées.
- La non-répudiation : cette propriété garantit qu'un participant qui a envoyé un message ne peut nier plus tard être l'auteur de cet envoi.

La confidentialité est définie de différentes manières dans la littérature avec des définitions plus ou moins fortes. La définition classique de la confidentialité garantit qu'on ne peut déchiffrer un texte chiffré à moins d'avoir la clé. Il existe des définitions encore plus exigeantes telles que la sécurité sémantique qui exige que l'attaquant ne peut avoir aucune information à partir d'un texte chiffré que ce qu'il avait sans le texte chiffré même si l'information ne concerne pas le texte lui-même. Une autre propriété en rapport avec la confidentialité est la propriété de non-malléabilité. La non-malléabilité garantit qu'il est impossible à partir d'un message chiffré d'obtenir un autre message crypté dont le contenu décrypté est en rapport avec le premier. La propriété de non-malléabilité est plus forte que la sécurité sémantique.

5.4.1 Les propriétés d'authentification et d'établissement de clés

Nous nous sommes focalisés dans cette thèse sur les propriétés d'authentification et sur les propriétés concernant l'établissement des clés cryptographiques. Souvent une même propriété est définie de différentes manières dans la littérature, nous donnons dans cette section les définitions les plus consensuelles des propriétés de sûreté les plus importantes. Ces définitions restent cependant informelles, elles seront traduites en spécifications formelles dans le prochain chapitre de cette thèse.

[28] classe les propriétés d'authentification et d'établissement de clés en deux catégories :

- Les propriétés *orientées utilisateur* : comme son nom l'indique, cette catégorie regroupe les propriétés qui sont en rapport avec les participants du système. Les propriétés sont définies pour deux participants A et B dans un protocole. Deux propriétés importantes sont répertoriées dans cette catégorie :
  1. La propriété «*Knowledge of peer*» : A a des garanties sur l'identité de B.
  2. La propriété «*Far-end operative*» : A croit que B a *récemment* envoyé un message.

  La première propriété garantit à A que les messages qu'il reçoit dans cette session du protocole proviennent de B mais elle ne garantit pas que ces messages sont récents et qu'ils ne proviennent pas, par exemple, d'une ancienne session du protocole. La deuxième propriété garantit justement que B est opérationnel et qu'il a récemment envoyé un message. La combinaison de ces deux propriétés de base, nous donne la propriété importante appelée l'*authentification*.
- Les propriétés *orientées clé* : cette catégorie regroupe les propriétés exprimées sur les clés. Deux propriétés importantes sont classées dans cette catégorie :

1. La propriété «*Key Freshness*» : elle garantit que la clé est fraîche et qu'elle n'a pas été utilisée dans de précédentes sessions du protocole.
2. La propriété «*Key secrecy*» : elle garantit que la clé est uniquement connue de A, B et éventuellement d'un tiers de confiance.

Combiner ces deux dernières propriétés garantit que la clé est secrète et qu'elle est fraiche, dans ce cas on obtient la propriété dite «*Good key property*».

La fraîcheur des clés est généralement garantie par deux mécanismes les *estampilles* et les *nonces* :

- Les *estampilles* (*timestamps*) : ce mécanisme consiste à rajouter l'heure courante dans un message dont on veut garantir la fraîcheur, le participant qui reçoit le message décide ensuite, si celui-ci est assez *frais* ou non en comparant l'heure contenue dans le message à son horloge locale. Ce mécanisme exige une synchronisation des horloges des différents participants. Si les horloges ne sont pas bien synchronisées, il y a risque d'attaque.
- Les *nonces* : ce mécanisme fonctionne généralement de la manière suivante, un participant A génère un nouveau nonce et l'envoie à B qui renvoie un message contenant le nonce avec une information dont on veut garantir la fraîcheur, le tout crypté avec une certaine clé. Quand A reçoit ce message il est sûr que celui-ci (et donc l'information qu'il contient) a été généré après que A a généré le nonce en question. L'attaquant ne doit pas être capable de deviner la valeur des nonces utilisés.

Les propriétés de bases présentées précédemment peuvent être combinées pour exprimer des propriétés beaucoup plus complexes concernant à la fois les utilisateurs et les clés. On peut, par exemple, vouloir garantir qu'une clé est *bonne* (*good key property*) mais également être sûr qu'elle est en possession de l'autre participant du protocole et qu'il s'en sert pour la présente session. Il faut noter que la propriété key secrecy garantit que la clé n'est pas connue par des participants autres que A et B (et le tiers de confiance) mais ne garantit pas que A et B possèdent effectivement cette clé.

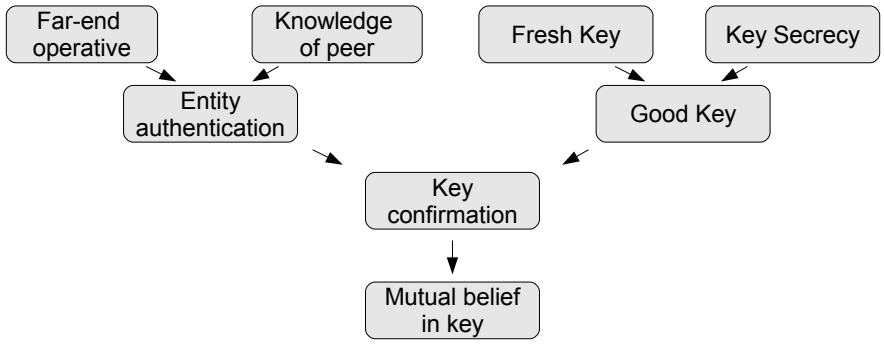


FIG. 5.2 – Une hiérarchie des propriétés de sûreté.

La figure 5.2 montre comment combiner les propriétés de base orientées utilisateur ou orientées clé pour obtenir des propriétés plus complexes dont :

- La propriété «*Key confirmation*» de A à B est établie, si B a la garantie qu'une clé K est une *bonne* clé pour communiquer avec A et que ce dernier (A) possède cette clé.
- La propriété «*Mutual belief in key*» : cette propriété est établie pour B si la clé K est *bonne* pour être utilisée avec A et que A non seulement possède cette clé (comme dans la propriété *Key confirmation*) mais considère qu'elle est bonne et souhaite l'utiliser pour communiquer avec B.

5.5 Les attaques

Il peut y avoir une infinité de façons d'attaquer un protocole, ceci est dû au fait que l'attaquant peut envoyer une infinité de messages et réutiliser les informations qu'il a glanées de différentes manières. Cependant, il y a des tactiques d'attaques connues répertoriées dans la littérature, en voici quelques-unes :

- L'écoute : c'est la façon la plus basique d'attaquer un protocole, elle consiste à écouter le canal de communication et à essayer d'obtenir des informations sensibles.
- L'altération (ou la modification) : consiste à changer un ou plusieurs champs d'un message intercepté.
- Le rejeu : cette attaque consiste à réutiliser un message (ou un fragment de message) intercepté dans une précédente exécution du protocole pour tromper les participants.
- La réflexion : cette attaque est un cas particulier du rejeu où l'attaquant retourne un message donné à celui qui l'a envoyé en espérant avoir en retour un message qu'il peut utiliser contre le participant initiateur du message original.
- Le déni de service : dans ce type d'attaque, l'attaquant empêche les participants d'un protocole d'arriver à finir l'exécution de celui-ci. Un cas typique d'une attaque par déni de service est celle où l'attaquant cherche à épuiser le nombre de connections possibles à un serveur donné afin d'empêcher d'autres utilisateurs à s'y connecter.

5.6 Les techniques de preuves

Nous présentons dans cette section les principales techniques de preuves dans le modèle formel. Il existe un grand nombre de travaux qui abordent la vérification des protocoles cryptographiques dans le modèle formel. La principale difficulté rencontrée dans le modèle formel est l'explosion de l'espace des états que peut atteindre le système à cause de la taille illimitée des messages ainsi que la non limitation du nombre de sessions du protocole et du nombre de messages envoyés par l'attaquant.

Certaines méthodes proposées imposent tout de même certaines limitations. Ceci est notamment le cas des méthodes utilisant le *model-checking* où une partie de l'espace des états est explorée en espérant trouver des attaques sur cette partie. Ces méthodes basées sur le model-checking sont essentiellement utilisées pour trouver des attaques et non pas pour prouver qu'un protocole est correct. Parmi les méthodes basées sur le model-checking, on peut citer l'approche proposée par Lowe [61] utilisant l'outil FDR pour trouver l'attaque

sur le protocole de Needham-Schroeder [66]. FDR est un model-checker pour l'algèbre des processus CSP, Lowe modélise chaque participant au protocole par un processus exécutant une étape du protocole. D'autres méthodes proposent d'utiliser d'autres model-checker comme $Mur\phi$ [65] et *Brutus* [36].

D'autres méthodes n'imposent pas de restriction sur l'espace des états à explorer, on peut citer parmi elles la méthode inductive introduite par Paulson [67]. Cette méthode se base sur le principe de la preuve par induction connu en mathématique, ces preuves sont effectuées en utilisant le l'assistant à la démonstration Isabelle. L'attaquant se base sur un ensemble contenant tous les messages ayant transité dans le système pour former de nouveaux messages. Cette méthode se limite à deux primitives cryptographiques : le hachage et le chiffrement.

Certaines méthodes proposées sont basées à la fois sur le *model-checking* et sur des preuves formelles, c'est notamment le cas de l'outil NRL Analyser [64]. Cet outil est un programme Prolog qui commence par un état qui viole une propriété de sûreté et essaye d'atteindre l'état initial pour essayer de trouver des attaques. L'outil permet également d'établir des preuves que des classes d'états ne sont pas atteignables.

Il y a également des méthodes basées sur des logiques construites spécialement pour raisonner sur les protocoles cryptographiques. La première de ces logiques est la logique BAN introduite par Burrows, Abadi et Needham [32]. Dans la logique BAN, le protocole est retranscrit sous la forme de formules de cette logique et un ensemble de règles d'inférences sont alors appliquées pour essayer de dériver des formules contenant les propriétés désirées. D'autres logiques ont été introduites par la suite telles que la logique GNY [49] et SVO [80].

5.6.1 Les techniques de composition de protocoles

La composition des protocoles a fait l'objet d'un intérêt particulier ces dernières années et un ensemble de travaux ont été mené dans ce sens. La question qui se pose est la suivante : comment garantir qu'un protocole prouvé correct par rapport à un ensemble de propriétés de sûreté conserve ces propriétés une fois exécuté dans un environnement où d'autres protocoles sont exécutés en parallèle. Cependant, la définition de la composition varie d'une étude à une autre : dans certains travaux la composition est une simple exécution parallèle des deux protocoles dans un même environnement, dans d'autres travaux la définition est plus large et les messages des protocoles composés peuvent être imbriqués. La composition des protocoles peut être interprétée de deux manières différentes :

- Soit on cherche a prouver que deux protocoles distincts puissent s'exécuter parallèlement sans risque d'interagir de façon à violer les propriétés de sûreté qu'ils garantissent. Parmi ces travaux on peut citer la méthode de Cortier *et al* [39] qui utilise une méthode d'unification des fragments de messages provenant des protocoles composés pour prouver que les deux protocoles ne produisent pas les mêmes fragments. On peut citer également les travaux de Guttman *et al* [51] qui définit une relation d'indépendance entre les protocoles, la propriété d'indépendance de deux protocoles signifie qu'ils ne manipulent pas les même fragments de messages. Ces deux approches fournissent des conditions syntaxiques suffisantes pour la composition des protocoles.

– Soit on considère les choses de façon plus large : on cherche à combiner deux protocoles de manière à obtenir un protocole plus complexe garantissant plus de propriétés de sûreté que ses deux composants, cette approche est appelée *component-based design of protocols*. Dans ce cas chaque étude définit les limitations dans la façon de combiner les messages des protocoles. Parmi ces travaux, on peut citer les études de Datta *et al* [42].

5.7 Conclusion

Nous avons présenté dans ce chapitre des concepts en rapport avec les protocoles cryptographiques et leur composition en nous focalisant sur ce qui est lié à nos travaux de thèse. Nous avons ainsi présenté diverses propriétés d'authentification et de partage de clés, les différents types d'attaques ainsi que diverses techniques de preuves présentes dans la littérature.

Chapitre 6

La modélisation et la composition des protocoles cryptographiques

6.1 Introduction

Les protocoles cryptographiques sont des systèmes complexes qui nécessitent l'utilisation d'outils et de concepts pour aider le concepteur pour arriver à les modéliser. Nous pensons donc qu'il est nécessaire de proposer des méthodes qui permettent de guider le concepteur dans le processus de développement d'un protocole cryptographique, ceci sans perdre de vue les exigences en matière de sécurité de ces protocoles. Le but est donc de proposer une méthode de développement qui permette de concevoir des protocoles *corrects par construction*. Les protocoles cryptographiques sont un sujet qui a fait l'objet de nombreux travaux où diverses méthodes formelles ont été employées. En étudiant les divers travaux réalisés dans le domaine, il apparaît que la majeure partie de ces travaux s'intéressent à la preuve des protocoles. Ces preuves sont obtenues (parfois automatiquement) sans fournir une analyse du fonctionnement du protocole en déterminant par exemple quelle partie du protocole garantit quelle propriété de sûreté désirée. Nous pensons qu'une analyse détaillée du fonctionnement des protocoles existant est le meilleur moyen d'obtenir des informations qui permettront plus tard de prouver d'autres protocoles ou d'en concevoir des nouveaux. Les techniques présentées dans ce chapitre exposent et étendent les travaux déjà publiés dans nos publications [20, 22].

La méthode proposée dans cette thèse est en même temps une méthode d'analyse, de conception et de preuve. L'idée générale est la composition de *mécanismes* cryptographiques simples garantissant un ensemble de propriétés pour obtenir des protocoles plus complexes garantissant plus de propriétés. Un protocole cryptographique est, ainsi, obtenu en combinant un ensemble de *mécanismes*. Chacun de ces mécanismes apporte au protocole une ou plusieurs propriétés de sûreté données parmi les propriétés génériques de la figure 5.2 qui sont :

– La propriété «*Knowledge of peer*».

- La propriété «*Far-end operative*».
- La propriété «*Key Freshness*».
- La propriété «*Key secrecy*».

L'idée est de combiner ces mécanismes pour obtenir des protocoles satisfaisant plus de propriétés selon la hiérarchie des propriétés présentée dans la figure 5.2. Nous nous intéressons dans cette thèse aux protocoles d'authentification et d'établissement de clés. Le mécanisme 1 est un exemple de mécanisme garantissant la propriété dite *Knowledge of peer*, celui-ci est contenu dans le standard international *ISO/IEC 9798 Part2* [5]. Ce standard regroupe des protocoles pour l'authentification avec cryptographie symétrique sans tiers de confiance.

1. $B \rightarrow A : N_b$
2. $A \rightarrow B : \{N_b, B\}_{K_{AB}}$

Le mécanisme 1

Le participant B envoie un nonce et attend la réponse de A contenant le nonce initialement envoyé par B avec l'identité de l'agent qui a initié le mécanisme (B), le tout crypté par une clé partagée entre A et B. Si le nonce N_b est frais, ce mécanisme garantit en plus la propriété *Far-end operative*. Une variante de ce mécanisme consiste à renvoyer un message contenant le nonce avec cette fois, l'identité de l'agent dont on veut attester l'identité (A), ceci nous donne le mécanisme 2.

1. $B \rightarrow A : N_b$
2. $A \rightarrow B : \{N_b, A\}_{K_{AB}}$

Le mécanisme 2

Le concepteur d'un protocole cryptographique garantissant une authentification mutuelle entre deux participants A et B peut penser à utiliser par exemple le mécanisme 2 pour authentifier B à A et le mécanisme 1 pour authentifier A à B pour obtenir le protocole 2 suivant :

1. $A \rightarrow B : N_a$
2. $B \rightarrow A : \{N_a, B\}_{K_{AB}}, N_b$
3. $A \rightarrow B : \{N_b, B\}_{K_{AB}}$

Protocole 2: *Le résultat de la composition.*

Le protocole 2 ainsi obtenu est censé garantir la propriété d'authentification pour l'agent A à l'étape 2 et pour l'agent B à l'étape 3. Les deux mécanismes 1 et 2 sont prouvés corrects dans le modèle formel, la question que l'on doit se poser est : *en est-il de même pour le protocole 2 obtenu en composant ces deux mécanismes ?* . La réponse à cette question est non car une attaque est possible sur ce protocole, l'attaque 1 où I est l'attaquant, se présente comme suit :

1. $I_A \rightarrow B : N_i$
2. $B \rightarrow I_A : \{N_i, B\}_{K_{AB}}, N_b$
3. $I_A \rightarrow B : N_b$
4. $B \rightarrow I_A : \{N_b, B\}_{K_{AB}}, N'_b$
5. $I_A \rightarrow B : \{N_b, B\}_{K_{AB}}$

L'attaque 1

L'attaquant I engage en parallèle deux sessions du protocole avec l'agent B. À l'issue de l'attaque 1, le participant B croit qu'il communique avec l'agent A alors qu'il communique en réalité avec l'attaquant I.

Un des intérêts de notre démarche est de pouvoir réutiliser les preuves de correction d'un mécanisme déjà prouvé pour montrer la correction de plusieurs protocoles. Il faut donc définir précisément quelles sont les conditions qui font qu'une composition de deux mécanismes déjà prouvés est possible ou non.

Nous avons choisi d'illustrer dans cette thèse notre méthode sur le protocole de *Yahalom*. Le choix de ce protocole, présenté pour la première fois dans le papier [32] présentant la logique BAN, n'est pas fortuit. En effet, ce protocole est souvent étudié par des chercheurs utilisant les méthodes formelles car il constitue un bon challenge pour tester leur méthode à cause de sa forme atypique par rapport aux autres protocoles présents dans la littérature. Yahalom est un protocole d'établissement de clés en 4 étapes basé sur un tiers de confiance noté S :

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : B, \{A, N_a, N_b\}_{K_{BS}}$
3. $S \rightarrow A : \{B, K_{AB}, N_a, N_b\}_{K_{AS}}, \{A, K_{AB}\}_{K_{BS}}$
4. $A \rightarrow B : \{A, K_{AB}\}_{K_{BS}}, \{N_b\}_{K_{AB}}$

Protocole 3: *Le protocole de Yahalom.*

[28] dit de ce protocole qu'il est «*fréquemment étudié par les chercheurs utilisant les méthodes formelles pour la vérification des protocoles comme Paulson [67]. La raison est qu'il a une structure inhabituelle, de plus il est sujet à des attaques subtiles, cela fait de lui un bon challenge pour tester une nouvelle technique.*»

Une autre raison qui a motivé notre choix de ce protocole pour illustrer notre approche est qu'il existe une version modifiée de celui-ci proposée par Burrows *et al* (voir le protocole 4). Contrairement au protocole de Yahalom qui a été prouvé correct, la version modifiée est sujette à des attaques, ceci constitue un exemple intéressant pour notre démarche dans la mesure où l'on montre qu'une combinaison correcte (par rapport au critère que nous introduirons plus loin dans cette thèse) de mécanismes nous permet d'obtenir un protocole correct (Yahalom) et qu'une combinaison qui ne satisfait pas aux conditions que nous introduirons plus loin dans cette thèse donne un protocole incorrect. Un autre point intéressant est que ces deux exemples constituent un cas typique de réutilisation de mécanismes car les deux protocoles ont des similitudes, les preuves de mécanisme faites une fois pourront donc être réutilisées.

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : B, N_b, \{A, N_a\}_{K_{BS}}$
3. $S \rightarrow A : N_b, \{B, K_{AB}, N_a\}_{K_{AS}}, \{A, K_{AB}, N_b\}_{K_{BS}}$
4. $A \rightarrow B : \{A, K_{AB}, N_b\}_{K_{BS}}, \{N_b\}_{K_{AB}}$

Protocole 4: *Le protocole de Yahalom modifié par Burrows et al.*

Le protocole de Yahalom a été prouvé correct par rapport à plusieurs propriétés de sûreté, les diverses études faites sur ce protocole ont démontré que les propriétés suivantes étaient vérifiées :

- À l'issue de l'étape 3, le participant A a la garantie que B est le participant avec qui il exécute la session courante du protocole (les propriétés *Knowledge of peer* et *Far-end operative*).
- À l'issue de l'étape 3, le participant A a également la certitude que la clé K_{AB} n'est connue que par son vis-à-vis B et le tiers de confiance S (la propriété *Key secrecy*).
- À l'issue de l'étape 3, le participant A a également la certitude que la clé K_{AB} est fraîche.
- À l'issue de l'étape 4, le participant B a la garantie que A est le participant avec qui il exécute la session courante du protocole (les propriétés *Knowledge of peer* et *Far-end operative*).
- À l'issue de l'étape 4, le participant B a également la certitude que la clé K_{AB} n'est connue que par son vis-à-vis A et le tiers de confiance S (la propriété *Key secrecy*).
- À l'issue de l'étape 4, le participant B a la certitude que la clé K_{AB} est fraiche mais uniquement si on suppose que l'agent A se comporte *normalement* en transmettant à B la clé qu'il vient de recevoir de S et non pas une ancienne clé.
- À l'issue de l'étape 4, le participant B a la certitude que l'agent A est en possession de la clé K_{AB} (la propriété *Key confirmation*).

La preuve de ces propriétés sur ce protocole a été faite de différentes manières dans divers travaux, mais à chaque fois ces propriétés ont été prouvées sur ce protocole en entier sans savoir quelle partie du protocole garantit quelle propriété.

Nous avons décomposé le protocole de Yahalom en plusieurs mécanismes distincts, chacun a été prouvé comme garantissant des propriétés particulières, nous présentons ici ces mécanismes de façon informelle pour expliquer notre approche de façon générale avant d'introduire la modélisation et les preuves formelles des mécanismes et de leur composition :

1. Le mécanisme 3 garantit au participant A que B est le participant avec qui il exécute la session courante du protocole (les propriétés *Knowledge of peer* et *Far-end operative*). Le symbole X remplace d'autres données présentes dans les fragments cryptés.

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : B, \{A, N_a\}_{K_{BS}}$
3. $S \rightarrow A : \{B, X, N_a\}_{K_{AS}}$

Le mécanisme 3

La présence de ces données de type X est importante car si on omet de les insérer dans ce mécanisme on obtient le mécanisme 4 qui lui ne garantit plus les propriétés désirées. Nous nous servirons justement de ce mécanisme incorrect pour montrer comment on trouve les attaques avec notre méthode.

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : B, \{A, N_a\}_{K_{BS}}$
3. $S \rightarrow A : \{B, N_a\}_{K_{AS}}$

Le mécanisme 4

2. Le mécanisme 5 est contenu dans le standard *ISO/IEC 11770-2* [4] sous le nom de *Key Establishment Mechanism 7*. Il garantit à A et à B la propriété *Key secrecy*. Ce mécanisme est fréquemment utilisé dans les protocoles de distribution de clé avec un tiers de confiance, nous l'avons réutilisé plusieurs fois pour faire la preuve du protocole Kerberos [57] et du protocole Bauer-Berson-Feiertag [16].

1. $A \rightarrow B : A$
2. $B \rightarrow S : B, A$
3. $S \rightarrow A : \{B, K_{AB}\}_{K_{AS}}, \{A, K_{AB}\}_{K_{BS}}$
4. $A \rightarrow B : \{A, K_{AB}\}_{K_{BS}}$

Le mécanisme 5

Ce mécanisme peut lui-même être décomposé en deux mécanismes, le mécanisme 6 garantissant la propriété *Key secrecy* pour A et le mécanisme 7 garantissant cette même propriété pour B.

1. $A \rightarrow B : A$
2. $B \rightarrow S : B, A$
3. $S \rightarrow A : \{B, K_{AB}\}_{K_{AS}}$

Le mécanisme 6

1. $B \rightarrow S : B, A$
2. $S \rightarrow A : \{A, K_{AB}\}_{K_{BS}}$
3. $A \rightarrow B : \{A, K_{AB}\}_{K_{BS}}$

Le mécanisme 7

3. Le mécanisme 8 vise à permettre à A d'avoir une certitude sur la fraîcheur de clé envoyée par S. Ce mécanisme est souvent combiné avec le mécanisme 5.

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : B, A, N_a$
3. $S \rightarrow A : \{K_{AB}, N_a\}_{K_{AS}}$

Le mécanisme 8

4. Le mécanisme 9 est un mécanisme atypique que nous n'avons retrouvé dans aucun autre protocole étudié, il permet à l'agent B d'avoir des certitudes sur la fraîcheur de la clé envoyée par S à condition d'avoir, au préalable, prouvé la propriété *key secrecy* sur cette clé. Ce mécanisme doit être combiné avec un autre mécanisme garantissant la propriété *key secrecy* pour obtenir en plus la propriété de fraîcheur de la clé. Contrairement aux mécanismes 3 ou 5 qui peuvent être utilisés comme des protocoles à part entière pour, respectivement, les propriétés de *entity authentication* et *key secrecy*, le mécanisme 9 doit être combiné avec un autre mécanisme garantissant la propriété *key secrecy*. Un mécanisme apporte des propriétés supplémentaires à un protocole sans forcément les garantir quand il est utilisé séparément, ceci constitue une différence fondamentale entre ce que nous désignons par mécanismes et les protocoles.

1. $B \rightarrow S : B, A, N_b$
2. $S \rightarrow A : \{B, K_{AB}, N_b\}_{K_{AS}}$
3. $A \rightarrow B : K_{AB}, \{N_b\}_{K_{AB}}$

Le mécanisme 9

5. Le dernier mécanisme (10) permet à B d'avoir des garanties sur l'identité de A (la propriété *Knowledge of peer*) et sur le fait qu'il soit actif (la propriété *Far-end operative*). Ce mécanisme est basé sur le fait que les agents A et B partagent une clé secrète K_{AB}.

1. $B \rightarrow S : B, \{A, N_b\}_{K_{BS}}$
2. $S \rightarrow A : \{B, X, N_b\}_{K_{AS}}$
3. $A \rightarrow B : \{N_b\}_{K_{AB}}$

Le mécanisme 10

En combinant l'ensemble de ces mécanismes, nous obtenons le protocole de Yahalom mais avant d'aborder la combinaison des mécanismes, il faut prouver que ces mécanismes sont eux-mêmes corrects, sinon il faut arriver à retrouver les attaques éventuelles.

Nous avons présenté dans cette section la notion de *mécanisme*, les mécanismes sont combinés pour obtenir des protocoles complexes. Contrairement aux protocoles, un mécanisme peut garantir une propriété P_1 à condition qu'une propriété P_2 soit satisfaite, ceci permet de combiner ce mécanisme avec différents mécanismes satisfaisant la propriété P_2. Nous présentons dans la section

suivante la démarche proposée en B événementielle pour modéliser les mécanismes et les propriétés de sûreté ainsi que la preuve de leur correction. Nous présentons également une méthode pour retracer d'éventuelles attaques.

6.2 La modélisation des mécanismes cryptographiques

La modélisation de chaque mécanisme cryptographique se fera en deux étapes, la première est un modèle abstrait qui spécifie la propriété que l'on veut préserver avec ce mécanisme. Ce premier modèle abstrait est ensuite raffiné avec un modèle concret qui introduit les détails du mécanisme qui permettent de préserver la propriété spécifiée dans le modèle abstrait. Le modèle concret inclut également le modèle de l'attaquant choisit pour prouver le mécanisme.

6.2.1 Le modèle abstrait

Le modèle abstrait sert de spécification à la propriété de sûreté désirée, il permet également de structurer le déroulement d'une session du mécanisme en précisant quelle propriété doit être prouvée à chaque étape du déroulement du mécanisme. Il est basé sur la notion de *transaction abstraite*. Une transaction abstraite correspond à une session du protocole pour un participant donné, un agent qui souhaite participer à une session du protocole entame une nouvelle transaction abstraite. Intuitivement, une transaction abstraite correspondra à un nonce dans le modèle concret (pour les protocoles utilisant les nonces). Les ensembles porteurs d'un modèle abstrait sont :

- `TRANSACTION` : l'ensemble des transactions.
- `AGENT` : l'ensemble des participants.
- `KEY` : l'ensemble des clés.

Ces notions d'agent et de clé sont communes à tous les protocoles d'où leur présence à ce niveau d'abstraction, il est également nécessaire de les avoir pour exprimer l'ensemble des propriétés d'authentification et d'établissement de clé.

Nous introduisons deux agents particuliers qui sont l'intrus `I` et le tiers de confiance `S` ces deux derniers sont évidemment distincts :

CONSTANT
`I`
`S`
AXIOMS
$I \in AGENT$
$S \in AGENT$
$I \neq S$

Une transaction a une source qui est l'agent qui l'a initiée (l'agent qui génère le nonce dans le modèle concret) et une destination présumée qui correspond à l'agent avec lequel l'agent qui a initié la transaction croit communiquer. La source et la destination présumée des transactions sont contenues dans deux fonctions totales `T_SRC` et `T_B_DST` (pour *transaction believed destination*).

CONSTANT
T_SRC
T_B_DST
AXIOMS
T_SRC ∈ TRANSACTION → AGENT
T_B_DST ∈ TRANSACTION → AGENT

Les variables du modèle dépendant de la propriété spécifiée, cependant il y a des variables que l'on retrouve quelle que soit la propriété exprimée, nous introduisons des variables qui permettent de structurer l'évolution d'une transaction d'une étape à une autre du mécanisme. Ces variables seront appelées `init`, `step1`, ..., `stepN`, `end` une transaction appartient à ces ensembles au fur et à mesure que son exécution avance :

VARIABLES
init
step1
step2
...
end
INVARIANT
init ∈ TRANSACTION
step1 ∈ init
...

D'autres variables sont ajoutées selon la propriété exprimée :

6.2.1.1 La propriété d'authentification

Pour exprimer les propriétés d'authentification qui sont les propriétés *Knowledge of peer* et *Far-end operative*, on utilise une variable supplémentaire qui est `T_DST`, cette variable est une fonction booléenne qui indique que la destination présumée d'une transaction est active et participe à la transaction courante.

VARIABLES
T_DST
INVARIANT
T_DST ∈ TRANSACTION → BOOL

Au début d'une transaction cette variable est initialisée à `FALSE` par l'événement `INIT`, à la fin de la transaction (événement `END`), il faut que l'agent qui a initié la transaction soit capable de dire que la destination présumée est active.

EVENT INIT
ANY
T
WHERE
grd1 : T ∈ TRANSACTION
grd2 : T ∉ init
grd3 : T ∉ end
THEN
act1 : init := init ∪ {T}
act2 : T_DST(T) := FALSE
END

EVENT END
ANY
T
WHERE
grd1 : T ∈ init
grd2 : T ∉ end
grd3 : T_DST(T) = TRUE
THEN
act1 : end := end ∪ {T}
END

De son côté la destination présumée met la variable T_DST à TRUE si elle participe à la transaction correspondante avec l'événement ACTIVE.

EVENT ACTIVE
ANY
T
B
WHERE
grd1 : T ∈ TRANSACTION
grd2 : B ∈ AGENT
grd3 : B = T_B_DST(T)
THEN
act1 : T_DST(T) := TRUE
END

L'invariant exprimant la propriété d'authentification indique que pour toutes les transactions arrivées à la fin de l'exécution du mécanisme, la destination présumée est active.

INVARIANT
∀ T·T ∈ end ⇒ T_DST(T) = TRUE

6.2.1.2 Les propriétés d'établissement de clés

Les propriétés d'établissement de clé nécessitent également l'introduction de nouvelles variables, la première est une variable nommée T_KEY. Cette variable contient la clé choisie par le participant à l'issue d'une session d'un mécanisme d'établissement de clé. Si le mécanisme se déroule en deux étapes init et end alors le type de la variable T_KEY est une fonction totale de l'ensemble des transactions finies (end) à celui des clés.

VARIABLES
T_KEY
INVARIANT
T_KEY $\in$ end $\rightarrow$ KEY

La propriété «*key secrecy*» La première propriété d'établissement de clé est la propriété de *key secrecy* dans laquelle il faut être sûr que la clé n'est connue au plus que par la source de la transaction, sa destination présumée et le tiers de confiance. Nous introduisons une nouvelle variable KEY_MEM contenant les clés connues par chaque agent, c'est donc une fonction totale de l'ensemble des agents à l'ensemble des sous-ensemble des clés.

VARIABLES
KEY_MEM
INVARIANT
KEY_MEM $\in$ AGENT $\rightarrow \mathbb{P}$(KEY)

Une transaction est initiée par sa source dans l'événement INIT, à la fin de la transaction (événement END) la source de la transaction est censée choisir une clé qui doit satisfaire la propriété de *key secrecy*.

EVENT INIT
ANY
T
WHERE
grd1 : T $\in$ TRANSACTION
grd2 : T $\notin$ init
grd3 : T $\notin$ end
THEN
act1 : init $:=$ init $\cup$ {T}
END

Dans la *garde 4* de l'événement END la source de la transaction doit choisir une clé de session qui doit être secrète. Il rajoute également la clé choisie à la l'ensemble des clés qui sont en sa possession.

EVENT END
ANY
T
K
WHERE
grd1 : $T \in init$
grd2 : $T \notin end$
grd3 : $K \in KEY$
grd4 : $\forall X.X \in AGENT \wedge$
$X \neq T\_SRC(T) \wedge X \neq T\_B\_DST(T) \wedge X \neq S$
$\Rightarrow K \notin KEY\_MEM(X)$
THEN
act1 : $end := end \cup \{T\}$
act2 : $T\_KEY(T) := K$
act3 : $KEY\_MEM(T\_SRC(T)) := KEY\_MEM(T\_SRC(T)) \cup \{K\}$
END

Au niveau abstrait, l'attaquant peut essayer de rajouter des clés à sa mémoire avec un événement ATTACK. L'invariant garantissant la propriété de *key secrecy* énonce que pour chaque clé choisie à l'issue de l'exécution du mécanisme ne doit être connue que par la source, la destination présumée et le tiers de confiance.

INVARIANT
$\forall T, X \cdot$
$T \in end \wedge$
$X \in AGENT \wedge X \neq T\_SRC(T) \wedge X \neq T\_B\_DST(T) \wedge X \neq S$
$\Rightarrow$
$T\_KEY(T) \notin KEY\_MEM(X)$

La propriété «*key freshness*» Pour la propriété de fraicheur des clé (*Key Freshness*), nous introduisons une variable KEY_FRS (pour *key freshness*). De manière générale, si une clé est destinée à une transaction particulière alors au moment où elle est générée (au niveau du tiers de confiance par exemple) la valeur de la transaction à laquelle elle est destinée est indiquée dans la variable KEY_FRS. Au moment de recevoir une clé qu'il attendait, un agent doit être capable de savoir si cette clé est destinée à la transaction courante ou non.

VARIABLES
KEY_FRS
INVARIANT
$KEY\_FRS \in KEY \nrightarrow TRANSACTION$

L'agent source initie la transaction avec l'événement INIT, à la fin de la transaction (événement END), la source de la transaction est censée choisir une clé qui doit correspondre à la transaction courante.

EVENT INIT
ANY
T
WHERE
grd1 : $T \in TRANSACTION$
grd2 : $T \notin init$
grd3 : $T \notin end$
THEN
act1 : $init := init \cup \{T\}$
END

EVENT END
ANY
T
K
WHERE
grd1 : $T \in init$
grd2 : $T \notin end$
grd3 : $K \in dom(KEY\_FRS)$
grd4 : $KEY\_FRS(KEY) = T$
THEN
act1 : $end := end \cup \{T\}$
act2 : $T\_KEY(T) := K$
END

Un événement KEY_GEN correspondant à la génération des clés est rajouté, une clé peut être générée soit par le tiers de confiance, soit par l'un ou l'autre des participants à un protocole ou encore par l'attaquant.

L'invariant de la propriété *key freshness* indique que la clé choisie pour une transaction donnée est une clé qui a été générée pour cette transaction.

INVARIANT
$\forall T \cdot T \in end \Rightarrow KEY\_FRS(T\_KEY(T)) = T$

Certains mécanismes (comme le mécanisme 9) ne garantissent la propriété de fraîcheur de la clé qu'une fois combiné avec des mécanismes garantissant la propriété *key secrecy*. Dans ce cas, l'invariant stipule que la propriété de fraîcheur n'est obtenue que si la propriété de secret est garantie :

```
INVARIANT
      ∀ T·
      T ∈ end ∧
/* Si T satisfait la propriété key secrecy... */
          (∀ X·
          X ∈ AGENT ∧ X ≠ T_SRC(T) ∧ X ≠ T_B_DST(T) ∧ X ≠ S
          ⇒
          T_KEY(T) ∉ KEY_MEM(X))
/* ... alors elle satisfait la propriété key freshness. */
      ⇒
      KEY_FRS(T_KEY(T)) = T
```

Un même mécanisme garantissant la propriété *key freshness* pourra donc être combiné avec différents mécanismes garantissant le secret de la clé. De façon plus générale, on peut combiner de cette manière différentes propriétés de sûreté afin d'obtenir des propriétés plus complexes suivant la hiérarchie des propriétés illustrée dans la figure 5.2. En combinant cet invariant avec l'invariant de la propriété *key secrecy* on obtient l'invariant de la propriété de la fraîcheur de la clé. De façon générale, si P_1 et P_2 sont deux propriétés de sûretés et que l'on prouve qu'un mécanisme M_1 garantit la propriété P_1 et qu'un mécanisme M_2 garantit la propriété $P_1 \Rightarrow P_2$ et sous réserve de faire la preuve que la composition des deux mécanismes M_1 et M_2 est possible (nous présentons les obligations de preuve relatives à la composition plus loin dans ce chapitre) alors le mécanisme résultant de la composition satisfait les propriétés P_1 et P_2.

Les modèles abstraits présentés dans cette section sont propres à des propriétés données et non à des mécanismes particuliers. Un même modèle abstrait correspondant à une propriété particulière pourra par la suite être raffiné de différentes manières chacune correspondant à un mécanisme particulier. Nous verrons aussi quand nous aborderons la composition des mécanismes que les modèles abstrait seront composés pour exprimer plusieurs propriétés de sûreté.

6.2.2 Le modèle concret

Les détails de chaque mécanisme sont introduits dans le modèle concret. Les mécanismes se distinguent les uns des autres par la structure de leurs messages, il faut cependant pouvoir trouver une démarche de modélisation qui soit assez générale pour être ré-appliquée sur les différents mécanismes sans fournir un effort de réflexion pour chaque étude de cas de la part du concepteur. Le but est d'obtenir des modèles concrets automatiquement à partir de la structure du mécanisme. L'autre exigence est que la modélisation choisie se doit d'être efficace en terme de preuve automatique.

Nous proposons d'obtenir le modèle concret d'un mécanisme en rajoutant exhaustivement les types de messages qui le composent. Chaque type de message sera modélisé par un ensemble contenant les messages transitant par ce canal et un ensemble de fonctions totales modélisant chacune un attribut du message. Le domaine de chacune de ces fonctions est l'ensemble des messages envoyés par ce canal, le codomaine est quant à lui le type de la donnée présente dans l'attribut du message.

Nous introduisons des types supplémentaires pour modéliser les différents attributs des messages, ces nouveaux types dépendent du type des protocoles modélisés. Pour les protocoles d'authentification et de transport de clé basés sur un cryptage symétrique ou asymétrique, nous avons besoin des types suivants : MSG pour les messages, NONCE pour les nonces et TIME pour l'horodatage. TIME est une valeur de date et d'heure estampillée sur un message pour dater l'instant de son envoi. Tous les messages quelle que soit leur structure ont un attribut commun qui est MSG_SRC permettant d'indiquer l'agent qui a envoyé ce message. Cet attribut est modélisé par une constante.

SET
MSG
NONCE
TIME
CONSTANT
MSG_SRC
AXIOMS
MSG_SRC $\in$ MSG $\rightarrow$ AGENT

Nous pouvons ensuite ajouter, à partir de ces types, les variables nécessaires à la modélisation d'un message donné. Par exemple, pour un message dont les attributs sont :

$$A \rightarrow B : Att1, Att2, ..., AttN$$

où les attributs $Att1, Att2, ..., attN$ sont respectivement de type $T1, T2, ..., Tn$, les variables rajoutées sont comme suit :

VARIABLES
CHAN
CHAN_Att1
CHAN_Att2
...
CHAN_AttN
INVARIANT
CHAN $\subseteq$ MSG
CHAN_Att1 $\in$ CHAN $\rightarrow$ T1
CHAN_Att2 $\in$ CHAN $\rightarrow$ T2
...
CHAN_AttN $\in$ CHAN $\rightarrow$ Tn

6.2.2.1 Les événements

Nous nous focaliserons dans la suite de cette thèse sur les protocoles à base de nonces. Considérons le mécanisme 11 qui se veut être aussi général que possible. Ce mécanisme est constitué des messages *1* jusqu'au message F. Chacun de ces messages est constitué d'un nombre donné d'attributs. Deux messages qui se suivent peuvent être liés par des attributs communs d'où la condition $Att1i = Att2j \ \wedge Att1k = Att2l \ \wedge ...$ entre les messages *1* et *2*.

1. $A \rightarrow B : Att11, Att12, ..., Att1N$
2. $B \rightarrow X : Att21, Att22, ..., Att2M$
$Avec\ Att1i = Att2j\ \wedge Att1k = Att2l\ \wedge ...$
3. $X \rightarrow \ldots$
$\ldots$
$F.\ Y \rightarrow A : AttF1, AttF2, ..., AttFO$

Le mécanisme 11

Dans le dernier message F, l'agent A teste si ce message est conforme à une condition particulière qui dépend de chaque mécanisme, cette condition peut être liée à la valeur d'un nonce envoyé préalablement. Cette condition que nous appelons `COND_CHANF` est une fonction booléenne sur l'ensemble des messages présents dans le canal de communication `CHAN_F`.

L'agent A génère un nonce frais au début du mécanisme et l'utilise pour la session courante. Au lieu de remplacer les transactions par les nonces dans ce modèle concret, nous laisserons cette démarche pour un raffinement ultérieur tout en gardant à l'esprit que les transactions que nous manipulons dans ce modèle concret correspondent en réalité à des nonces. Nous conservons donc les variables `init` et `end` qui indiquent l'évolution d'une transaction particulière. Par contre, les variables `T_DST` et `KEY_FRS` qui permettent à un agent de savoir si les propriétés de sûreté sont conservées sont supprimées, c'est en se basant sur la structure des messages qu'un agent est censé avoir des garanties sur les propriétés de sûreté.

À partir du schéma du mécanisme 11 et de la condition `COND_CHANF`, nous pouvons générer automatiquement les événements modélisant le déroulement du mécanisme. Nous avons besoin d'un événement pour chaque type de message. Le premier événement `SEND1` choisit une transaction T qui n'a pas été utilisée et la rajoute dans l'ensemble `init`, il génère également un nouveau message et l'envoie à travers le canal `CHAN1`.

EVENT SEND1
ANY
T
Att11
Att12
...
Att1N
msg1
WHERE
grd1 : T ∈ TRANSACTION
grd2 : T ∉ init
grd3 : T ∉ end
grd4 : Att11 ∈ T11
grd5 : Att12 ∈ T12
grd6 : ...
grd5 : Att1N ∈ T1N
grd5 : msg1 ∉ CHAN1
THEN
act1 : init := init ∪ {T}
act2 : CHAN1 := CHAN1 ∪ {msg1}
act3 : CHAN1_Att11(msg1) := Att11
act4 : CHAN1_Att12(msg1) := Att12
act5 : ...
act6 : CHAN1_Att1N(msg1) := Att1N
END

Dans le deuxième événement SEND2, l'agent B reçoit un message *msg1* par le canal *CHAN1* et envoie un message *msg2* par le canal *CHAN2* en respectant une éventuelle condition de la forme $Att1i = Att2j \ \wedge Att1k = Att2l \ \wedge ...$ entre les attributs de ces deux messages.

EVENT SEND2
ANY
Att21
Att22
...
Att2M
msg2
msg1
WHERE
grd1 : $\texttt{Att21} \in \texttt{T21}$
grd2 : $\texttt{Att22} \in \texttt{T22}$
grd3 : ...
grd5 : $\texttt{Att2M} \in \texttt{T2M}$
grd6 : $\texttt{msg2} \notin \texttt{CHAN2}$
grd7 : $\texttt{msg1} \in \texttt{CHAN1}$
grd8 : $\texttt{CHAN1\_Att1i}(\texttt{msg1}) = \texttt{Att2j} \wedge \texttt{CHAN1\_Att1k}(\texttt{msg1}) = \texttt{Att2l}$
THEN
act1 : $\texttt{CHAN2} := \texttt{CHAN2} \cup \{\texttt{msg2}\}$
act2 : $\texttt{CHAN2\_Att21}(\texttt{msg2}) := \texttt{Att21}$
act3 : $\texttt{CHAN2\_Att22}(\texttt{msg2}) := \texttt{Att22}$
act4 : ...
act5 : $\texttt{CHAN1\_Att2M}(\texttt{msg2}) := \texttt{Att2M}$
END

Les événements SEND3 jusqu'à SENDF sont similaires à l'événement SEND2. Dans l'événement final END, l'agent A prend une transaction T impliquée dans une session en cours $\texttt{T} \in \texttt{init}$ et teste si le message msgF reçu est conforme à la condition COND_CHANF, si c'est le cas il termine la session avec succès et met la transaction T dans l'ensemble end.

EVENT END
ANY
T
msgF
WHERE
grd1 : $\texttt{T} \in \texttt{TRANSACTION}$
grd2 : $\texttt{T} \in \texttt{init}$
grd3 : $\texttt{msgF} \notin \texttt{CHANF}$
grd4 : $\texttt{COND\_CHANF}(\texttt{T} \mapsto \texttt{msgF}) = \texttt{TRUE}$
THEN
act1 : $\texttt{end} := \texttt{end} \cup \{\texttt{T}\}$
END

La chaîne de raffinements entre les événements La figure 6.1 résume la chaîne de raffinements entre les événements du modèle abstrait et ceux du modèle concret. De manière générale, l'événement SEND1 où la transaction est

initiée raffine l'événement INIT. L'événement END raffine quant à lui l'événement END abstrait où la transaction se termine. Pour les autres événements, la chaîne de raffinement est comme suit :

- Pour les propriétés d'authentification : L'événement abstrait ACTIVE est raffiné par l'un des événements SEND concret dans lequel la destination présumée participe à la session courante du mécanisme.
- Pour la propriété *key freshness* : L'événement KEY_GEN est raffiné par tous les événements où une clé est générée. Au niveau concret, une clé peut être générée par un attaquant, le tiers de confiance ou par un simple participant dans le système.

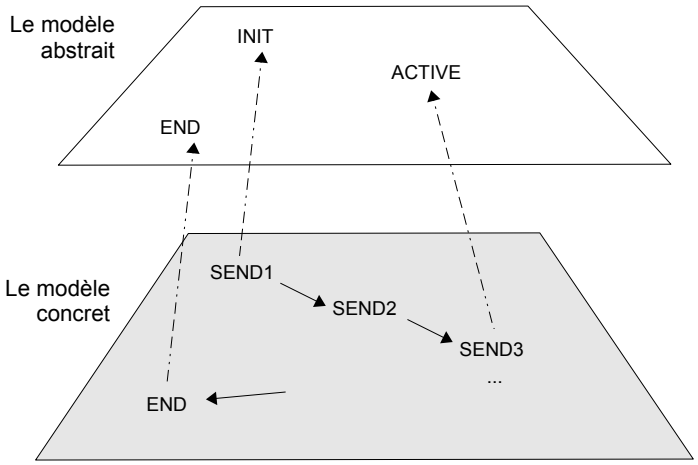


FIG. 6.1 – La chaine de raffinement entre les événements abstraits et les événements concrets.

Les variables T_DST et KEY_FRS qui permettaient à un agent de *tricher* dans l'événement END abstrait pour savoir si la propriété désirée est vérifiée ou non ont été supprimées et les gardes correspondantes dans cet événement ont été remplacées par des tests sur la structure des messages et notamment la garde 4 de l'événement END concret. Pour faire les preuves du raffinement, il faut trouver un invariant de collage entre les variables supprimées et la structure des messages permettant de faire les preuves sur les gardes des événements raffinés. Par exemple, dans le cas de la propriété d'authentification, il faut prouver comme invariant :

INVARIANT

$$\forall\, \mathtt{T}, \mathtt{msgF} \cdot$$
$$\mathtt{T} \in \mathtt{init} \land \mathtt{msgF} \in \mathtt{CHANF} \land$$
$$\mathtt{COND\_CHANF}(\mathtt{T} \mapsto \mathtt{msgF}) = \mathtt{TRUE}$$
$$\Rightarrow$$
$$\mathtt{T\_DST}(\mathtt{T}) = \mathtt{TRUE}$$

L'obtention de l'invariant de collage peut s'avérer être une tâche délicate, nous proposons une méthode qui nous permet de les générer automatiquement. Mais avant d'aborder la génération de l'invariant de collage, on se doit de compléter notre modèle concret car jusqu'à maintenant, les événements ajoutés reflètent un fonctionnement normal du mécanisme et n'incluent pas les actions de l'attaquant.

6.2.2.2 Le modèle de l'attaquant

Modéliser les connaissances de l'attaquant est indispensable pour pouvoir prouver les propriétés du système. Nous avons modélisé dans notre publication [18] l'attaquant du modèle de Dolev-Yao [44] une première fois en B événementiel et nous l'avons appliqué au cas des protocoles en cascades [44] pour retrouver les preuves déjà faites manuellement dans le papier de Dolev-Yao [44]. Le but de cette démarche était de modéliser l'attaquant de façon à pouvoir réutiliser ce modèle sur différentes études de cas. Nous présentons dans cette section notre modèle en B événementiel de l'attaquant de Dolev-Yao et l'appliquerons sur le modèle concret des mécanismes cryptographiques. Tout comme le modèle concret des mécanismes, nous insistons sur le fait que ce modèle doit pouvoir être généré automatiquement pour un mécanisme donné.

Le modèle formel suppose une cryptographie parfaite, l'attaquant ne peut donc lire que les messages non-chiffrés ou ceux chiffrés avec une clé en sa possession. On suppose également que l'attaquant a le contrôle total du canal de communication :

- Il peut intercepter et effacer les messages du canal de communication.
- Il peut générer un nombre infini de messages.
- Il peut décrypter les parties des messages dont il possède la clé de cryptage.
- Il peut découper les messages en fragments cryptés.

Ce que nous désignons par fragments de messages sont les parties cryptées d'un messages, par exemple le message :

$$S \rightarrow A : B, \{B, K_{AB}\}_{K_{AS}}, \{A, K_{AB}\}_{K_{BS}}$$

contient les deux fragments $\{B, K_{AB}\}_{K_{AS}}$ et $\{A, K_{AB}\}_{K_{BS}}$.

Tous ces comportements sont modélisés par des événements appropriés et les connaissances que récolte l'attaquant sont modélisées par des variables. Les connaissances de l'attaquant sont de différentes natures :

- Des clés cryptographiques.
- Des nonces.
- Des fragments cryptés de messages qui même s'il ne peut pas lire ce qu'ils contiennent, il peut cependant les réutiliser pour former de nouveaux messages qui ont un fragment de la même structure.

Nous avions déjà utilisé une variable `KEY_MEM` contenant les clés connues par chaque agent pour exprimer la propriété *key secrecy* dans le modèle abstrait, nous réutilisons cette même variable pour modéliser les clés connues par l'attaquant (qui est un agent comme un autre). Si la propriété exprimée dans le modèle abstrait était autre que la propriété *key secrecy*, alors on introduit cette variable dans le modèle concret.

VARIABLES
KEY_MEM
INVARIANT
KEY_MEM $\in$ AGENT $\rightarrow \mathbb{P}$(KEY)

Une variable N_MEM contient les nonces obtenus par l'attaquant, cette variable est un ensemble de transactions car nous avons fait le choix de garder dans le modèle concret la notion de transaction avant de la raffiner plus tard vers les nonces.

VARIABLES
N_MEM
INVARIANT
N_MEM $\subseteq$ TRANSACTION

Les fragments sont, quant à eux, contenus dans plusieurs ensembles FRAG1, FRAG2, ... selon leurs types. Deux fragments de même type sont stockés dans le même ensemble même s'ils proviennent de deux types différents de messages. Un fragment qui a une structure $Att1, Att2, \ldots, AttN$ où les attributs $Att1$, $Att2$, ..., $AttN$ sont respectivement de type $T1$, $T2$, ..., Tn, est déclarée de la façon suivante :

VARIABLES
FRAG
FRAG_Att1
FRAG_Att2
...
FRAG_AttN
INVARIANT
FRAG $\subseteq$ MSG
FRAG_Att1 $\in$ CHAN $\rightarrow$ T1
FRAG_Att2 $\in$ CHAN $\rightarrow$ T2
...
FRAG_AttN $\in$ CHAN $\rightarrow$ Tn

Les connaissances de l'attaquant sont donc contenues dans les ensembles KEY_MEM(I), N_MEM ainsi que dans les ensembles de fragments FRAG1, FRAG2, ... Deux questions doivent être posées concernant ces variables :

1. Comment sont récoltées les connaissances de l'attaquant ?
2. Comment sont utilisées les connaissances de l'attaquant ?

Pour répondre à la première question, nous devons ajouter les événements permettant à l'attaquant de récolter les informations circulant sur les différents canaux de communication. La récolte des informations se fait de trois façons différentes :

1. En écoutant les différents canaux de communications : Pour ce faire, un événement LISTENi est rajouté pour chaque type de message. Dans le cas

général, illustré par le mécanisme 12, l'agent X envoie à l'agent Y un message qui a une partie non cryptée dont les attributs peuvent inclure des clés et des nonces ainsi qu'une partie cryptée avec différents types de fragments.

$\ldots$
$i.\ X \rightarrow Y : Att\_i1, Att\_i2, ..., \{Att\_il, ..., Att\_im\}_{Att\_ik}$,...
$\ldots$

Le mécanisme 12

où *Att_i1* est de type KEY, *Att_i2* est de type TRANSACTION. Dans ce cas, l'événement LISTENi est comme suit :

EVENT LISTENi
ANY
msgi
frag
WHERE
grd1 : msgi $\in$ CHANi
grd2 : frag $\in$ MSG
grd3 : frag $\notin$ FRAG
THEN
act1 : KEY_MEM(I) := KEY_MEM(I) $\cup$ {CHAN_Att_i1(msgi)}
act2 : N_MEM := N_MEM $\cup$ {CHAN_Att_i2(msgi)}
act3 : FRAG := FRAG $\cup$ {frag}
act4 : FRAG_Att_il(frag) := CHANi_Att_il $\cup$ {msgi}
act5 : ...
act6 : FRAG_Att_im(frag) := CHANi_Att_im $\cup$ {msgi}
act7 : FRAG_Att_ik(frag) := CHANi_Att_im $\cup$ {msgi}
END

2. En déchiffrant les fragments cryptés avec les clés en sa possession, l'attaquant peut utiliser les clés en sa possession pour déchiffrer les fragments récoltés en écoutant les canaux de communication. De façon générale, pour un type de fragment FRAGi dont la structure est comme suit :

$$\{Att\_i1, Att\_i2, ..., Att\_in\}_{Att\_ik}$$

où *Att_i1* est de type KEY et *Att_i2* de type TRANSACTION, on rajoute l'événement DECRYPTi permettant de décrypter les fragments de ce type et d'en extraire les informations sensibles, si la clé de chiffrement du fragment est connue de l'attaquant :

```
EVENT DECRYPTi
    ANY
     frag
    WHERE
     grd1 : frag ∈ FRAGi
     grd2 : FRAGi_Att_ik(frag) ∈ KEY_MEM(I)
    THEN
     act1 : KEY_MEM(I) := KEY_MEM(I) ∪ {FRAG_Att_i1(frag)}
     act2 : N_MEM := N_MEM ∪ {FRAG_Att_i2(frag)}
     ...
    END
```

3. Un attaquant peut également acquérir des informations sensibles par d'autres moyens comme, par exemple, des connaissances initiales. Pour modéliser ce genre de situation, il suffit de rajouter des événements qui rajoutent les valeurs souhaitées aux variables modélisant les connaissances de l'attaquant que sont KEY_MEM(I), N_MEM ainsi que dans les ensembles de fragments FRAG1, FRAG2, ...

La réponse à la seconde question consiste à définir comment l'attaquant utilise l'ensemble des informations en sa possession pour attaquer le protocole. Les attaques sont généralement commises soit en effaçant des messages présents dans un des canaux de communication, soit en générant de nouveaux messages et en les envoyant aux participants honnêtes du mécanisme. La première façon de faire des attaques est facile à modéliser, il suffit de rajouter un événement INTERCEPTi pour chaque canal de communication, cet événement efface des messages choisies aléatoirement.

```
EVENT INTERCEPTi
    ANY
     msg
    WHERE
     grd1 : msg ∈ CHANi
    THEN
     act1 : CHANi := CHANi\{msg}
     act2 : ...
    END
```

Pour la génération des messages, nous utilisons deux types événements, le premier type permet de générer pour chaque canal de communication des messages de type correspondant à ces canaux à partir des fragments que l'attaquant a en sa possession. Pour un canal CHANi dont la structure est :

...
$i.\ X \rightarrow Y : Att\_i1, Att\_i2, ..., \{Att\_il, ..., Att\_im\}_{Att\_ik}$,...
...

Le mécanisme 13

où *Att_i1* est de type KEY, *Att_i2* est de type TRANSACTION et le fragment $\{Att\_il, ..., Att\_im\}_{Att\_ik}$ est de type FRAG, un événement GENERATEi qui génère de nouveaux messages avec les fragments, les clés et les nonces qu'il possède :

EVENT GENERATEi
ANY
msg
frag
T
key
WHERE
grd1 : msg ∈ MSG
grd2 : msg ∉ CHANi
grd3 : frag ∈ FRAG
grd4 : T ∈ N_MEM
grd5 : key ∈ KEY_MEM(I)
THEN
act1 : CHANi := CHANi\{msg}
act2 : CHANi_Att_i1(msg) := key
act3 : CHANi_Att_i2(msg) := T
act4 : CHANi_Att_il(msg) := FRAG_Att_il ∪ {frag}
...
act5 : CHANi_Att_im(msg) := FRAG_Att_im ∪ {frag}
act6 : CHANi_Att_ik(msg) := FRAG_Att_ik ∪ {frag}
END

L'attaquant doit être capable de générer des messages en utilisant des fragments qu'il a lui-même formés en utilisant des données et des clés en sa possession. Un événements GEN_FRAGi est ajouté pour chaque type de fragment. De façon générale, pour un fragment de type FRAGi de la forme :

$$\{Att\_i1, Att\_i2, ..., Att\_in\}_{Att\_ik}$$

où *Att_i1* est de type KEY et *Att_i2* de type TRANSACTION et *Att_in* de type AGENT, on rajoute l'événement GEN_FRAGi qui génère de nouveaux fragments de ce type :

```
EVENT GEN_FRAGi
    ANY
     frag
     T
     key1
     key2
     A
    WHERE
     grd1 : frag ∉ FRAGi
     grd2 : T ∈ N_MEM
     grd3 : key1 ∈ KEY_MEM(I)
     grd3 : key2 ∈ KEY_MEM(I)
     grd4 : A ∈ AGENT
    THEN
     act1 : FRAGi := FRAGi ∪ {frag}
     act2 : FRAGi_Att_i1(msg) := key1
     act3 : FRAGi_Att_i2(msg) := T
     ...
     act4 : FRAGi_Att_in(msg) := A
     act5 : FRAGi_Att_ik(msg) := key2
    END
```

Dans ce modèle d'attaquant de type Dolev-Yao, nous avons été aussi généraux que possible pour pouvoir générer automatiquement ces événements et variables à partir d'un mécanisme particulier. Il faut noter qu'il est possible d'affaiblir ou de renforcer les capacités de l'attaquant pour un cas particulier donné.

La génération d'invariant Pour prouver qu'un mécanisme implémente une propriété de sûreté spécifiée dans le modèle abstrait, il faut arriver à trouver un invariant de collage entre d'un côté, les variables concrètes constituées des messages et des connaissances de l'attaquant, et de l'autre les variables abstraites utilisées pour exprimer les propriétés de sûreté. Jusqu'à présent, nous avons modélisé les mécanismes pour obtenir les modèles abstraits et concrets de façon automatique à partir de la structure du mécanisme et des propriétés désirées, nous utiliserons également une méthode automatique pour la génération de l'invariant de collage. Nous utilisons pour la génération de l'invariant la méthode dite de renforcement de l'invariant avec le calcul de la plus faible pré-condition déjà présentée dans le chapitre 4.2.

Le choix de la propriété initiale à partir de laquelle l'invariant est construit nous est imposé par la propriété de sûreté désirée et le mécanisme utilisé. Dans le cas de la propriété d'authentification, la garde *3* de l'événement `END` abstrait ($\texttt{grd3} : \texttt{T\_DST}(\texttt{T}) = \texttt{TRUE}$) est remplacée dans le modèle concret par la garde *4* de l'événement `END` qui le raffine ($\texttt{grd4} : \texttt{COND\_CHANF}(\texttt{T} \mapsto \texttt{msgF}) = \texttt{TRUE}$). Ceci implique qu'il faut prouver la propriété :

INVARIANT
$\forall\, \mathtt{T}, \mathtt{msgF} \cdot$
$\mathtt{T} \in \mathtt{init} \wedge \mathtt{msgF} \in \mathtt{CHANF} \wedge$
$\mathtt{COND\_CHANF}(\mathtt{T} \mapsto \mathtt{msgF}) = \mathtt{TRUE}$
$\Rightarrow$
$\mathtt{T\_DST}(\mathtt{T}) = \mathtt{TRUE}$

Il en est de même avec les gardes *4* de l'événement `END` abstrait des autres propriétés de sûreté. L'invariant inductif est donc construit à partir de cette propriété par la méthode du renforcement.

La reconstitution des attaques Nous avons discuté dans le chapitre 4.2 de l'intérêt de la génération de l'invariant avec le calcul de la plus faible précondition en ce qui concerne la reconstitution des traces d'événements violant une propriété de sûreté. Nous utilisons cette technique pour reconstituer les attaques. Nous montrons dans la section suivante deux exemples dans lesquelles les techniques de génération automatique d'invariant et de reconstitution d'attaques ont été appliquées.

6.2.3 Exemple de mécanismes prouvés

Nous allons illustrer notre méthode sur deux exemples, le premier est un mécanisme correcte pour lequel nous générons automatiquement un modèle B événementiel ainsi que l'invariant inductif permettant d'obtenir la propriété de sûreté désirée. Le second est un mécanisme qui est sujet à des attaques pour lequel nous avons généré automatiquement un modèle B événementiel à partir duquel nous avons automatiquement reconstitué des attaques en utilisant la technique exposée dans le section précédente.

Le premier mécanisme est un mécanisme du protocole de Yahalom, il s'agit du mécanisme 3. Comme nous l'avions expliqué, ce mécanisme garantit la propriété d'authentification pour le participant A qui, à l'issue d'une session du mécanisme, a des garanties sur l'identité de B et sur sa participation à la session courante du protocole.

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : B, \{A, N_a\}_{K_{BS}}$
3. $S \rightarrow A : \{B, X, N_a\}_{K_{AS}}$

Le mécanisme 3

Nous utilisons le modèle abstrait correspondant à la propriété d'authentification pour vérifier cette propriété sur ce mécanisme. Le modèle concret est quant à lui généré comme expliqué dans la section 6.2.2. Trois canaux de communications sont nécessaires à la modélisation de ce mécanisme ainsi que les variables correspondant à chacun des attributs des messages. L'attribut X du canal `CHAN3` est de type quelconque (que nous avons appelé `DATA`).

```
VARIABLES
/* Le premier canal de communication */
      CHAN1
      CHAN1_A
      CHAN1_Na
/* Le deuxième canal de communication */
      CHAN2
      CHAN2_B
      CHAN2_A
      CHAN2_KBS
/* Le troisième canal de communication */
      CHAN3
      CHAN3_B
      CHAN3_X
      CHAN3_Na
      CHAN3_KAS
```

Les événements du mécanisme sont obtenus comme expliqués dans la section 6.2.2. Il s'agit des événements :

- SEND1 qui correspond à l'envoi sur le canal CHAN1 et qui raffine l'événement abstrait INIT.
- SEND2-0 qui correspond à l'envoi sur le canal CHAN2 par la vraie destination de la session courante et qui raffine l'événement abstrait ACTIVE.
- SEND2-1 qui correspond à l'envoi sur le canal CHAN2 par un agent autre que la vraie destination de la session courante et qui raffine l'événement SKIP.
- SEND3 qui correspond à l'envoi sur le canal CHAN3 et qui raffine l'événement SKIP.
- END qui correspond à la réception du message final sur CHAN3 et qui raffine l'événement END.

Les modèles complets de ce mécanisme peuvent être trouvés dans l'annexe A de cette thèse.

Les connaissances de l'attaquant sont modélisées comme expliqué dans la section 6.2.2.2 avec les variables N_MEM et KEY_MEM ainsi que deux variables FRAG2 et FRAG3 contenant respectivement les fragments de type $\{A, N_a\}_{K_{BS}}$ (du canal CHAN2) et $\{B, X, N_a\}_{K_{AS}}$(du canal CHAN3). L'ensemble des événements caractérisant le comportement de l'attaquant sont rajoutés.

La propriété initiale à partir de laquelle l'invariant est généré est obtenue selon la méthode exposée dans la section 6.2.2.2. Cette propriété est dans notre cas comme suit :

```
INVARIANT  /* La propriété P */
    ∀msg3·
    msg3 ∈ CHAN3 ∧
    CHAN3_Na(msg3) ∈ init ∧
    CHAN3_B(msg3) = T_B_DST(CHAN3_Na(msg3)) ∧
    K1(CHAN3_Kas(msg3)) = T_SRC(CHAN3_Na(msg3)) ∧
    K2(CHAN3_Kas(msg3)) = S
    ⇒
    T_DST(CHAN3_Na(msg3)) = TRUE
```

Le reste des invariants est obtenu par le calcul de la plus faible pré-condition, chacun des invariants est annoté avec l'invariant et l'événement dont il est la plus faible pré-condition. L'invariant complet a été obtenu à l'issue de ce processus en rajoutant 14 invariants à la propriété initiale P, en voici les 4 premiers générés à titre d'exemple.

```
INVARIANT  /* INV1 obtenu avec [SEND3](P) */
    ∀msg2·
    msg2 ∈ CHAN2 ∧
    K1(CHAN2_Kbs(msg2)) = CHAN2_B(msg2) ∧
    K2(CHAN2_Kbs(msg2)) = S ∧
    CHAN2_Na(msg2) ∈ init ∧
    CHAN2_B(msg2) = T_B_DST(CHAN2_Na(msg2))
    ⇒
    T_DST(CHAN2_Na(msg2)) = TRUE
```

```
INVARIANT  /* INV2 obtenu avec [GENERATE3](P) */
    ∀frag·
    frag ∈ FRAG3 ∧
    FRAG3_Na(frag) ∈ init ∧
    FRAG3_B(frag) = T_B_DST(FRAG3_Na(frag)) ∧
    K1(FRAG3_Kas(frag)) = T_SRC(FRAG3_Na(frag)) ∧
    K2(FRAG3_Kas(frag)) = S
    ⇒
    T_DST(FRAG3_Na(frag)) = TRUE
```

```
INVARIANT  /* INV3 obtenu avec [GENERATE2](INV1) */
    ∀frag·
    frag ∈ FRAG2 ∧
    K1(FRAG2_Kbs(frag)) = T_B_DST(FRAG2_Na(frag)) ∧
    K2(FRAG2_Kbs(frag)) = S ∧
    FRAG2_Na(frag) ∈ init
    ⇒
    T_DST(FRAG2_Na(frag)) = TRUE
```

INVARIANT /* INV4 obtenu avec [LISTEN2](INV3) */
$\forall$msg2·
msg2 $\in$ CHAN2 $\wedge$
K1(CHAN2_Kbs(msg2)) = T_B_DST(CHAN2_Na(msg2)) $\wedge$
K2(CHAN2_Kbs(msg2)) = S $\wedge$
CHAN2_Na(msg2) $\in$ init
$\Rightarrow$
T_DST(CHAN2_Na(msg2)) = TRUE

Le plus faible invariant inductif est trouvé, il est constitué de ces 14 invariants. Il sera utilisé lors de la composition de ce mécanisme avec d'autres mécanismes. Les obligations de preuves sont quasiment toutes déchargées automatiquement sauf quand il s'agit de renforcer l'invariant, là, on prend manuellement les plus faibles pré-conditions générées par l'outil RODIN et on les utilise pour renforcer l'invariant. Il est cependant possible d'envisager d'automatiser ce processus à l'aide d'un plug-in qui sera dédié à cette tâche.

6.2.4 Exemple de la reconstitution d'attaques

Nous utilisons pour illustrer la reconstitution des attaques, une version modifiée du mécanisme 3 dans laquelle nous avons enlevé le champ X du troisième message pour obtenir le mécanisme 4. La présence de la donnée X dans le troisième message du mécanisme 3 empêche l'attaquant d'utiliser des fragments de type $\{B, X, N_a\}_{K_{AS}}$ à la place des fragments de type $\{A, N_a\}_{K_{BS}}$ pour former des messages pour le deuxième canal de communications. D'un point de vue formel, le modèle B événementiel de ce mécanisme est semblable à celui du mécanisme 3 sauf que les fragments dont dispose l'attaquant sont contenus dans un seul ensemble FRAG qui contient aussi bien les fragments $\{B, N_a\}_{K_{AS}}$ et $\{A, N_a\}_{K_{BS}}$ qui ont la même structure.

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : B, \{A, N_a\}_{K_{BS}}$
3. $S \rightarrow A : \{B, N_a\}_{K_{AS}}$

Le mécanisme 4

Ce changement implique que l'attaquant dispose d'un nouveau moyen d'attaquer le mécanisme par rapport à celui du mécanisme 3, il consiste à récupérer des fragments du deuxième message et à les injecter dans le troisième canal et vice versa.

Les invariants sont générés à partir de la même propriété P que celle du mécanisme 3, le premier invariant obtenu est également le même que l'invariant INV1 de ce même mécanisme. Les autres invariants obtenus sont comme suit :

INVARIANT /* INV2 obtenu avec [GENERATE3](P) */
$\forall$frag·
frag $\in$ FRAG $\wedge$
FRAG_N(frag) $\in$ init $\wedge$
FRAG_A(frag) = T_B_DST(FRAG_N(frag)) $\wedge$
K1(FRAG_K(frag)) = T_SRC(FRAG_N(frag)) $\wedge$
K2(FRAG_K(frag)) = S
$\Rightarrow$
T_DST(FRAG_N(frag)) = TRUE

INVARIANT /* INV3 obtenu avec [LISTEN2](INV2) */
$\forall$msg2·
msg2 $\in$ CHAN2 $\wedge$
CHAN2_Na(msg2) $\in$ init $\wedge$
CHAN2_A(msg2) = T_B_DST(CHAN2_Na(msg2)) $\wedge$
K1(CHAN2_Kbs(msg2)) = T_SRC(CHAN2_Na(msg2)) $\wedge$
K2(CHAN2_Kbs(msg2)) = S
$\Rightarrow$
T_DST(CHAN2_Na(msg2)) = TRUE

INVARIANT /* INV4 obtenu avec [SEND2-1](INV3) */
$\forall$msg1·
msg1 $\in$ CHAN1 $\wedge$
CHAN1_Na(msg1) $\in$ init $\wedge$
CHAN1_A(msg1) = T_B_DST(CHAN1_Na(msg1))
$\Rightarrow$
T_DST(CHAN1_Na(msg1)) = TRUE

INVARIANT /* INV5 obtenu avec [GENERATE1](INV4) */
$\forall$T·
T $\in$ N_MEM $\wedge$
T $\in$ init
$\Rightarrow$
T_DST(T) = TRUE

INVARIANT /* INV6 obtenu avec [LISTEN1](INV5) */
$\forall$msg1·
msg1 $\in$ CHAN1 $\wedge$
CHAN1_Na(msg1) $\in$ init
$\Rightarrow$
T_DST(CHAN1_Na(msg1)) = TRUE

INVARIANT /* INV7 obtenu avec [SEND1](INV6) */
$\forall T \cdot$
$\neg T \in \texttt{init} \wedge$
$\neg T \in \texttt{end}$
$\Rightarrow$
$\bot$

L'invariant INV7 n'est pas établi par l'initialisation qui affecte la valeur $\varnothing$ aux variables init et end. L'attaque est donc reconstituée en suivant les annotation des différents invariants de la façon suivante :

1. L'événement Initialisation avec $\texttt{init} := \varnothing$ et $\texttt{end} := \varnothing$ qui ne vérifient pas l'invariant INV7, l'invariant INV6 n'est donc plus maintenu par l'événement SEND1.
2. L'événement SEND1 où un agent A envoie un message de la forme :

   $$A \rightarrow B : A, N_a$$

   est déclenché, l'invariant INV6 n'est plus vérifié ce qui induit que l'invariant INV5 n'est plus maintenu par l'événement LISTEN1.
3. L'événement LISTEN1 où l'attaquant écoute le premier canal de communication est déclenché, l'attaquant obtient la valeur du nonce N_a. L'invariant INV5 n'est plus vérifié ce qui induit que l'invariant INV4 n'est plus maintenu par l'événement GENERATE1.
4. L'événement GENERATE1 où un l'attaquant génère des messages pour le premier canal de communication est déclenché. La structure du message envoyé est donnée par la partie gauche de l'invariant INV4 : celle-ci indique entre autres que $\texttt{CHAN1\_A(msg1)} = \texttt{T\_B\_DST(CHAN1\_Na(msg1))}$ ce qui nous donne comme message :

   $$I \rightarrow X : B, N_a$$

   L'invariant INV4 n'est plus vérifié ce qui induit que l'invariant INV3 n'est plus maintenu par l'événement $\texttt{SEND2} - 1$.
5. L'événement $\texttt{SEND2} - 1$ où un participant autre que la vraie destination de la session courante du protocole (N_a) envoie un message dans le deuxième canal est déclenché. La structure du message envoyé est donnée par la partie gauche de l'invariant INV3 : celle-ci indique entre autres que :
   – $CHAN2\_A(msg2) = T\_B\_DST(CHAN2\_Na(msg2))$.
   – $K1(CHAN2\_Kbs(msg2)) = T\_SRC(CHAN2\_Na(msg2))$.
   – $K2(CHAN2\_Kbs(msg2)) = S$.
   ce qui nous donne comme message :

   $$A \rightarrow S : A, \{B, N_a\}_{K_{AS}}$$

   L'invariant INV3 n'est plus vérifié ce qui implique que l'invariant INV2 n'est plus maintenu par l'événement LISTEN2.

6. L'événement LISTEN2 où l'attaquant écoute le deuxième canal de communication est déclenché, l'invariant INV2 n'est plus vérifié car l'attaquant intercepte le message envoyé lors de l'étape précédente et obtient le fragment $\{B, N_a\}_{K_{AS}}$ dont la caractéristique est qu'il ne respecte pas l'invariant INV2. L'invariant INV2 n'est plus vérifié ce qui induit que la propriété P n'est plus maintenu par l'événement GENERATE3.
7. L'événement GENERATE3 est déclenché, l'attaquant génère un message pour le troisième canal de communication en utilisant le fragment obtenu lors de l'étape précédente. Les caractéristiques de ce message sont données par la partie gauche de l'implication de la propriété P :

$$S \rightarrow A : \{B, N_a\}_{K_{AS}}$$

L'attaque obtenue est résumée comme suit :

$1.A \rightarrow B : A, N_a$
$2.I \rightarrow A : B, N_a$
$3.A \rightarrow S : A, \{B, N_a\}_{K_{AS}}$
$4.S \rightarrow A : \{B, N_a\}_{K_{AS}}$

L'attaque 2

Cette attaque n'est pas la seule possible sur ce mécanisme, il en existe d'autres. Certaines de ces attaques peuvent être trouvées manuellement, d'autres sont beaucoup moins facile à trouver d'où la nécessité d'utiliser une méthode de reconstitution d'attaques. Il faut noter également qu'il est possible d'optimiser la démarche à l'aide du model-checker *ProB* [60] comme présenté dans la section 4.2.1. Dans notre cas, celui-ci trouve rapidement des traces violant les invariants contenant des propriétés sur le premier canal de communication CHAN1 (comme l'invariant INV6). Ces traces sont ensuite prolongées pour reconstruire des attaques complètes. Il faut noter que nous n'avons pas réussi à trouver des traces violant des propriétés sur les autres canaux de communications CHAN2 et CHAN3 avec le model-checker d'où l'intérêt de combiner les deux méthodes de recherche d'invariant avec la plus faible pré-condition avec le model-checking.

6.3 La composition des mécanismes cryptographiques

Nous avons vu dans la section précédente comment modéliser, prouver et reconstituer les attaques pour un mécanisme cryptographique. Nous présentons dans ce chapitre la méthode proposée pour la composition des mécanismes pour obtenir des protocoles cryptographiques complexes. La technique proposée repose sur la composition des modèles B événementielle basée sur la fusion des événements présentée dans le chapitre 3. Elle repose également sur la méthode que nous avons proposée dans la section 4.3 pour rendre les événements *fusionnables* en simulant le comportement de chacun d'entre eux sur les variables partagées dans l'autre événement. La démarche globale de composition

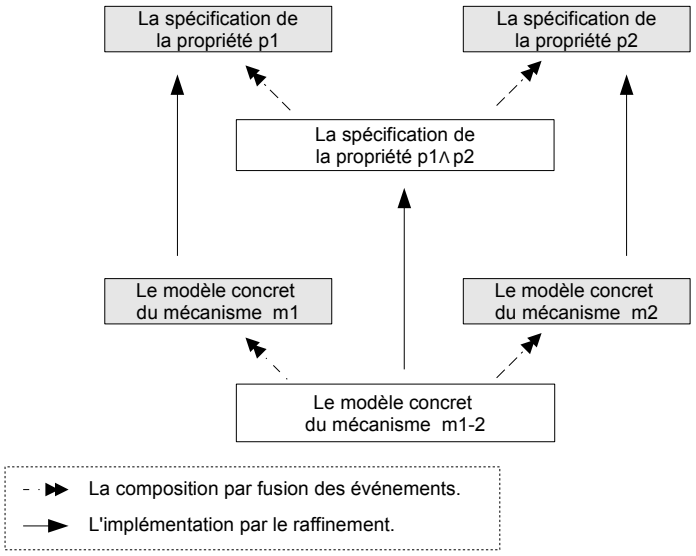


FIG. 6.2 – La composition des mécanismes.

des protocoles est résumée dans la figure 6.2 où deux mécanismes *M1* et *M2* vérifiant respectivement deux propriétés *P1* et *P2* sont composés pour obtenir un nouveau mécanisme *M1_2* vérifiant les deux propriétés *P1* et *P2*.

- Les événements des modèles abstraits spécifiant les propriétés *P1* et *P2* sont fusionnés pour obtenir une spécification des deux propriétés dans un même modèle.
- Les événements des modèles concrets des mécanismes *M1* et *M2* sont fusionnés pour obtenir un modèle d'un nouveau mécanisme *M1_2*.

6.3.1 La composition des spécifications abstraites

La composition de deux spécifications de deux propriétés *P1* et *P2* est guidé par la structure du mécanisme qu'on veut obtenir en résultat comme le montre la figure 6.3. il est possible fusionner deux spécifications de plusieurs façons différentes pour obtenir des mécanismes différents. Un événement *e1_i* de la première spécification est fusionné avec :

1. Soit un événement *e2_j* de la deuxième spécification pour synchroniser les deux mécanismes de la manière voulue.
2. Soit avec l'événement *Skip*.

Il faut noter que les deux spécifications fusionnées peuvent partager des variables utilisées pour exprimer les propriétés de sûreté qui sont `T_DST`, `KEY_FRS`, `KEY_DST` ainsi que les variables utilisées pour modéliser les étapes d'une transaction `init`, `end`, ...

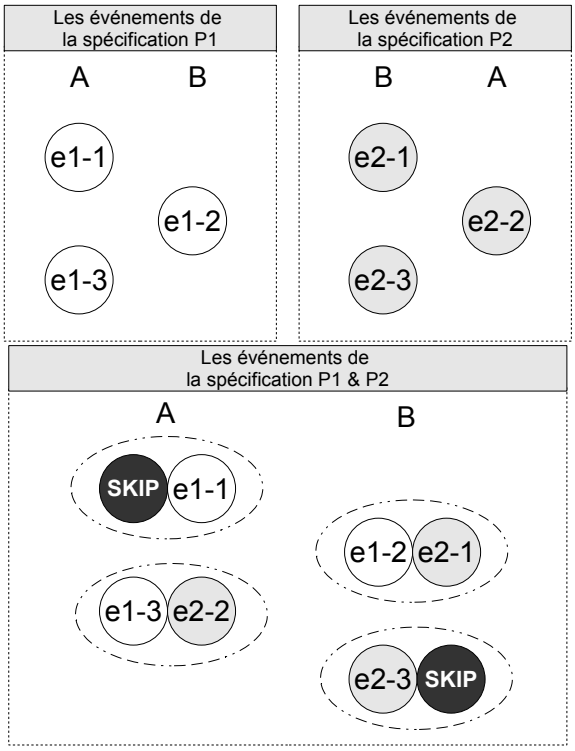


FIG. 6.3 – La composition des spécifications.

6.3.2 La composition des modèles concrets

Nous présentons dans cette sous-section la technique proposée pour la fusion des modèles concrets des deux mécanismes. La fusion est guidée par les messages envoyés par les deux mécanismes. Si le premier mécanisme 14 contient les canaux de communications `CHAN1_1`, `CHAN1_2`, ..., `CHAN1_F` :

---

(1). $A \rightarrow B : Att1\_11, Att1\_12, ..., Att1\_1n$
(2). $B \rightarrow X : Att1\_21, Att1\_22, ..., Att1\_2m$
(3). $X \rightarrow \ldots$
...
(F). $Y \rightarrow A : Att1\_F1, Att1\_F2, ..., Att1\_Fp$

Le mécanisme 14

Le second mécanisme 14 contient les canaux de communications `CHAN2_1`, `CHAN2_2`, ..., `CHAN2_G` :

---

(1). $C \rightarrow D : Att2\_11, Att2\_12, ..., Att2\_2q$
(2). $D \rightarrow \ldots$
...
(E). $W \rightarrow Z : Att1\_21, Att2\_22, ..., Att2\_2r$
(G). $Z \rightarrow C : Att2\_E1, Att2\_E2, ..., Att2\_Es$

Le mécanisme 15

Alors il y a plusieurs façons de combiner ces deux mécanismes, le choix revient au concepteur du protocole cryptographique pour choisir la manière avec laquelle il veut synchroniser les deux mécanismes 14 et 15. Il peut par exemple choisir de manière à obtenir comme résultat le mécanisme 16, dans lequel les agents A, B, X, ..., Y du mécanisme 16 correspondent respectivement aux agents Z, C, D, ..., W du mécanisme 15 et les messages *2*, *3*, ..., F du mécanisme 16 sont superposés avec respectivement les messages *1*, *2*, ..., E du mécanisme 15.

---

(1). $A\_Z \rightarrow B\_C : Att1\_11, Att1\_12, ..., Att1\_1n$
(2). $B\_C \rightarrow X\_D : Att1\_21, Att1\_22, ..., Att1\_2m,$
$Att2\_11, Att2\_12, ..., Att2\_2q$
(3). $X\_D \rightarrow \ldots$
...
(F). $Y\_W \rightarrow A\_Z : Att1\_F1, Att1\_F2, ..., Att1\_Fp,$
$Att1\_21, Att2\_22, ..., Att2\_2r$
($G+1$). $A\_Z \rightarrow B\_C : Att2\_E1, Att2\_E2, ..., Att2\_Es$

Le mécanisme 16

Du point de vue de la méthode B événementielle, les canaux de communications qui sont superposés sont renommés dans les deux modèles pour qu'ils

aient le même nom. Il faut noter que certains attributs de messages peuvent être partagés par les deux mécanismes, dans ce cas, ces attributs sont renommés de façon à avoir le même nom dans les deux modèles. Les deux modèles de mécanisme partagent, ainsi, des variables qui sont des canaux de communications ou des attributs de messages. La fusion de deux événements `SEND1_i` du premier mécanisme, et `SEND2_j` du second mécanisme qui envoient tous les deux des messages sur un canal de communications `CHAN` partagé entre les deux modèles de mécanisme ne pose pas de problème dans la mesure où l'action faite par les deux événements sur la variable partagée `CHAN` (et d'éventuels attributs communs) est la même.

6.3.2.1 La composition des connaissances de l'attaquant

La composition de deux mécanismes paraît, à première vue, bénéfique car permettant de garantir plus de propriétés de sûreté mais l'attaquant se retrouve également renforcé par la composition car il bénéficie des connaissances récoltées en attaquant les deux mécanismes en même temps comme, par exemple, dans l'attaque 1. Dans le modèle B événementiel de l'attaquant que nous avons présenté dans la section 6.2.2.2, les connaissances qu'il a récoltées sont contenues dans les variables `KEY_MEM`, `N_MEM` et les ensembles de fragments de différents types. Les actions de l'attaquant sont modélisées avec les événements : `LISTENi`, `DECRYPTi`, `INTERCEPTi`, `GENERATEi` et `GEN_FRAGi`. Intuitivement, pour prouver que la composition est possible, il faut prouver que les connaissances additionnelles acquises par l'attaquant avec un mécanisme ne lui permettent pas d'attaquer l'autre mécanisme. Du point de vue des modèles B événementiels, il faut injecter les connaissances de l'attaquant d'un mécanisme dans le modèle de l'autre mécanisme et prouver que les propriétés désirées sont toujours maintenues. C'est justement cela que nous faisons en rendant les événements des deux modèles *fusionables* en simulant le comportement des événements fusionnés sur les variables partagées. Dans ce cas les variables partagées sont les connaissances de l'attaquant et les canaux de communications partagés, ainsi, l'attaquant de chacun des modèles gagne en puissance en simulant le comportement de l'attaquant de l'autre modèle sur les variables partagées.

Les variables `KEY_MEM`, `N_MEM` ont le même nom dans les deux modèles et il n'y a donc pas besoin de les renommer, par contre, les variables contenant des fragments doivent être renommées de telle sorte que les fragments de même type dans les deux mécanismes aient le même nom dans les deux modèles.

Les événements sont traités de la façon suivante :

- Les événements `LISTEN` : Les événements `LISTEN` des deux modèles qui *écoutent* des canaux de communications partagés sont fusionnés entre eux, les autres sont fusionnés avec *Skip*.
- Les événements `DECRYPT` : Les événements `DECRYPT` des deux modèles qui *décryptent* des fragments partagés entre les deux modèles sont fusionnés entre eux, les autres sont fusionnés avec *Skip*.
- Les événements `INTERCEPT` : Les événements `INTERCEPT` des deux modèles qui interceptent des messages des canaux de communications partagés sont fusionnés entre eux, les autres sont fusionnés avec *Skip*.
- Les événements `GENERATE` : Les événements `GENERATE` des deux modèles qui génèrent de nouveaux messages pour des canaux de communications partagés sont fusionnés entre eux, les autres sont fusionnés avec *Skip*.

– Les événements `GEN_FRAG` : Les événements `GEN_GRAG` des deux modèles qui génèrent des fragments partagés des deux modèles sont fusionnés entre eux, les autres sont fusionnés avec *Skip*.

Á l'issue des modifications des deux modèles pour les rendre fusionables, de nouvelles obligations de preuves apparaissent dans chacun des deux modèles. Ces nouvelles obligations de preuves sont celles qui garantissent que la fusion est possible. Comme expliqué dans la section 4.3.1, les preuves faites séparément sur chacun des modèles restent valides car l'invariant était le plus faible des invariants inductifs et il doit être renforcé.

Nous proposons maintenant deux exemples, le premier est celui d'une composition réussie et le deuxième est celui d'une composition impossible.

Exemple d'une composition réussie Nous appliquons la technique de composition des mécanismes sur l'exemple du protocole de Yahalom. La première étape consiste à fusionner les mécanismes 6 et 7 pour obtenir le mécanisme 5 qui garantit la propriété *key secrecy* pour les deux agents A et B. Si la fusion des spécifications ne pose pas de problèmes particuliers, la fusion des deux modèles concrets pose en revanche le problème des deux fragments $\{B, K_{AB}\}_{K_{AS}}$ et $\{A, K_{AB}\}_{K_{BS}}$ provenant des deux mécanismes. En effet, ces fragments ont la même structure et ceci engendre des obligations de preuve supplémentaires pour chacun des modèles des mécanismes pour rendre la fusion possible, ces obligations de preuves sont cependant prouvées.

Le mécanisme obtenu est ensuite fusionné avec, respectivement :

– le mécanisme 8 pour obtenir la propriété de *key freshness* pour A.
– le mécanisme 9 pour obtenir la propriété de *key freshness* pour B.
– le mécanisme 3 pour obtenir la propriété d'authentification pour A.
– le mécanisme 10 pour obtenir la propriété d'authentification pour B.

Ces compositions ne posent pas de problèmes particuliers car la structure des fragments qui les composent n'est pas similaire et la composition n'engendre que des modifications mineures sur les modèles des mécanismes.

Exemple d'une composition impossible Nous essayons maintenant de combiner différents mécanismes pour obtenir le protocole de Yahalom modifié par Burrows *et al.* Nous allons non seulement réutiliser les mécanismes déjà prouvés pour le protocole de Yahalom mais également réutiliser certaines compositions déjà faite pour prouver ce protocole. Nous partons ainsi du mécanisme obtenu en composant les mécanismes 6, 7 et 8 pour le protocole de Yahalom mais cette fois, celui-ci est composé une deuxième fois avec le mécanisme 8 pour que l'agent B ait la propriété de *key freshness*. On obtient ainsi le mécanisme 17 suivant :

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : B, N_b, A, N_a$
3. $S \rightarrow A : \{B, K_{AB}, N_a\}_{K_{AS}}, \{A, K_{AB}, N_b\}_{K_{BS}}$
4. $A \rightarrow B : \{A, K_{AB}, N_b\}_{K_{BS}}$

Le mécanisme 17

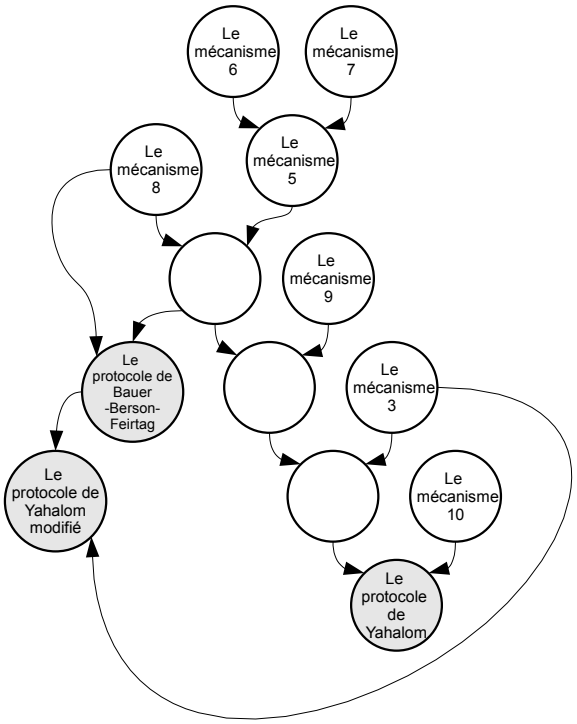


FIG. 6.4 – La composition des mécanismes.

Le mécanisme ainsi obtenu garantit les propriétés de *key secrecy* et *key freshness* pour les deux agents. Ce mécanisme correspond à un protocole qui existe dans la littérature, il a été proposé par Bauer, Berson et Feiertag [16]. En cherchant à prouver le protocole de Yahalom modifié nous avons donc réutilisé une partie des compositions effectuées pour la preuve du protocole de Yahalom et nous avons également prouvé le protocole de Bauer-Berson-Feiertag. Les compositions effectuées sont résumées par la figure 6.4.

Pour prouver le protocole de Yahalom modifié, nous composons maintenant le mécanisme 17 avec le mécanisme 3 pour satisfaire la propriété d'authentification pour A, pour obtenir le mécanisme 18 suivant :

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : B, \{A, N_a, N_b\}_{K_{BS}}$
3. $S \rightarrow A : \{B, K_{AB}, N_a\}_{K_{AS}}, \{A, K_{AB}, N_b\}_{K_{BS}}$
4. $A \rightarrow B : \{A, K_{AB}, N_b\}_{K_{BS}}$

Le mécanisme 18

Le fragment $\{B, K_{AB}, N_a\}_{K_{AS}}$ était déjà présent dans le mécanisme 3 mais pas le fragment $\{A, K_{AB}, N_b\}_{K_{BS}}$. Ces deux fragments ont la même structure, et en appliquant la méthode déjà exposée permettant de rendre la fusion des deux modèles B événementiels possible, on obtient des obligations de preuves supplémentaires pour le modèle du mécanisme 3. En appliquant la méthode de génération d'invariant vue précédemment, nous arrivons à un invariant ne satisfaisant pas l'initialisation du modèle et nous avons à reconstituer une attaque pour le mécanisme 18 (et donc une attaque pour le protocole de Yahalom modifié). Cette attaque viole la propriété d'authentification pour l'agent A. Nous présentons en ce qui suit les invariants obtenus sur le modèle du mécanisme 3 modifié pour rendre la fusion possible ainsi que la reconstitution de l'attaque. Le modèle original du mécanisme contenait 14 invariants dont les 4 premiers ont été présentés dans la section 6.2.3 (Le modèle complet peut être trouvé dans l'annexe A).

INVARIANT /* INV15 obtenu avec [LISTEN3](INV2) */

$\forall msg3 \cdot$
$msg3 \in CHAN3 \wedge$
$CHAN3\_Nb(msg3) \in init \wedge$
$CHAN3\_A(msg3) = T\_B\_DST(CHAN3\_Nb(msg3)) \wedge$
$K1(CHAN3\_Kbs(msg3)) = T\_SRC(CHAN3\_Nb(msg3)) \wedge$
$K2(CHAN3\_Kbs(msg3)) = S$
$\Rightarrow$
$T\_DST(CHAN3\_Nb(msg3)) = TRUE$

INVARIANT /* INV16 obtenu avec [SEND3](INV15) */

$\forall msg2 \cdot$
$msg2 \in CHAN2 \wedge$
$K1(CHAN2\_Kbs(msg2)) = CHAN2\_B(msg2) \wedge$
$CHAN2\_A(msg2) = T\_B\_DST(CHAN2\_Nb(msg2)) \wedge$
$CHAN2\_Nb(msg2) \in init \wedge$
$K2(CHAN2\_Kbs(msg2)) = S$
$\Rightarrow$
$T\_DST(CHAN2\_Nb(msg2)) = TRUE$

INVARIANT /* INV17 obtenu avec [GENERATE2](INV16) */

$\forall frag, T \cdot$
$T \in N\_MEM \wedge$
$frag \in FRAG2 \wedge$
$T \in init \wedge$
$FRAG2\_A(frag) = T\_B\_DST(T) \wedge$
$K2(FRAG2\_Kbs(frag)) = S$
$\Rightarrow$
$T\_DST(Tb) = TRUE$

INVARIANT /∗ INV18 obtenu avec [LISTEN2](INV17) ∗/
$\forall msg2, T \cdot$
$msg2 \in CHAN2 \wedge$
$T \in N\_MEM \wedge$
$T \in init \wedge$
$K2(CHAN2\_Kbs(msg2)) = S \wedge$
$CHAN2\_A(msg2) = T\_B\_DST(T)$
$\Rightarrow$
$T\_DST(T) = TRUE$

INVARIANT /∗ INV19 obtenu avec [SEND2-1](INV18) ∗/
$\forall msg1, T \cdot$
$msg1 \in CHAN1 \wedge$
$T \in init \wedge$
$T \in N\_MEM \wedge$
$CHAN1\_A(msg1) = T\_B\_DST(Tb)$
$\Rightarrow$
$T\_DST(Tb) = TRUE$

INVARIANT /∗ INV20 obtenu avec [GENERATE1](INV19) ∗/
$\forall T \cdot$
$T \in init \wedge$
$T \in N\_MEM$
$\Rightarrow$
$T\_DST(T) = TRUE$

INVARIANT /∗ INV21 obtenu avec [LISTEN1](INV20) ∗/
$\forall msg1 \cdot$
$msg1 \in CHAN1 \wedge$
$CHAN1\_Na(msg1) \in init$
$\Rightarrow$
$T\_DST(CHAN1\_Na(msg1)) = TRUE$

INVARIANT /∗ INV22 obtenu avec [SEND1](INV21) ∗/
$\forall T \cdot$
$\neg T \in init \wedge$
$\neg T \in end$
$\Rightarrow$
$\bot$

L'invariant INV22 contredit l'initialisation, nous pouvons donc reconstituer notre attaque de la façon suivante :

1. `INV22` = [`SEND1`](`INV21`) :

$$A \rightarrow B : A, N_a$$

2. `INV21` = [`LISTEN1`](`INV20`) : l'attaquant intercepte le message précédent sur le canal `CHAN1` et garde le nonce N_a.
3. `INV20` = [`GENERATE1`](`INV19`) :

$$I \rightarrow A : B, N_a$$

4. `INV19` = [`SEND2` $-$ `1`](`INV18`) :

$$A \rightarrow S : A, N'_a, \{B, N_a\}_{K_{AS}}$$

5. `INV18` = [`LISTEN2`](`INV17`) : l'attaquant intercepte le message précédent sur le canal `CHAN2` et garde le fragment $\{B, N_a\}_{K_{AS}}$.
6. `INV17` = [`GENERATE2`](`INV16`) :

$$I \rightarrow S : A, N_a, \{B, N_a\}_{K_{AS}}$$

7. `INV16` = [`SEND3`](`INV15`) :

$$S \rightarrow A : \{A, K_{AB}, N_a\}_{K_{BS}}, \{B, K_{AB}, N_a\}_{K_{AS}}$$

8. `INV15` = [`LISTEN3`](`INV2`) : l'attaquant intercepte le message précédent sur le canal `CHAN2` et garde le fragment $\{B, K_{AB}, N_a\}_{K_{AS}}$.
9. `INV2` = [`GENERATE3`](`P`) :

$$I \rightarrow A : \{B, K_{AB}, N_a\}_{K_{AS}}$$

L'attaque obtenue correspond à l'attaque de Syverson [79] sur le protocole de Yahalom modifié, elle est résumée comme suit dans l'attaque 3 :

---

1. $A \rightarrow B : A, N_a$
2. $I \rightarrow A : B, N_a$
3. $A \rightarrow S : A, N'_a, \{B, N_a\}_{K_{AS}}$
4. $I \rightarrow S : A, N_a, \{B, N_a\}_{K_{AS}}$
5. $S \rightarrow A : \{A, K_{AB}, N_a\}_{K_{BS}}, \{B, K_{AB}, N_a\}_{K_{AS}}$
6. $I \rightarrow A : \{B, K_{AB}, N_a\}_{K_{AS}}$

L'attaque 3

Á l'issue de cette session du protocole, A croit que B participe à la session mais, en réalité, il s'agit de l'attaquant I.

6.4 Comparaison avec les travaux existants

Notre démarche n'impose aucune restriction sur la façon de composer les protocoles contrairement aux travaux où les protocoles ne sont pas composés mais juste exécutés en parallèle. Parmi ces travaux, on peut citer la méthode de Cortier *et al* [39] qui utilise une méthode d'unification des fragments de messages provenant des protocoles composés pour prouver que les deux protocoles ne produisent pas les mêmes fragments ou encore les travaux de Guttman *et al* [51] qui définit une relation d'indépendance entre les protocoles, la propriété d'indépendance de deux protocoles signifie qu'ils ne manipulent pas les même fragments de messages. Ces deux approches fournissent des conditions syntaxiques suffisantes mais uniquement pour des protocoles s'exécutant parallèlement.

Parmi les autres approches qui traitent de la combinaison au sens large du terme on peut citer les travaux de [42] et la logique PCL, celle-ci offre plus de possibilités que les autres approches présentées plus haut concernant les différentes manières de composer les protocoles. Deux façons de composer les protocoles sont proposées : la première appelée *composition parallèle* consiste globalement à pouvoir exécuter les protocoles en même temps. La deuxième, appelée *composition séquentielle*, permet une composition séquentielle des différentes étapes des deux protocoles, celle-ci est illustrée dans le papier [42] sur un exemple de la composition d'un protocole à base de signature (protocole 5) garantissant le propriété d'authentification avec un protocole basé sur le mécanisme de Diffie-Hellman (déjà présenté dans la section 5.2).

1. $A \rightarrow B : N_a$
2. $B \rightarrow A : N_b, Sig_B(N_b, N_a, A)$
3. $A \rightarrow B : Sig_A\{N_b, N_a, B\}$

Protocole 5: *Un protocole d'authentification à base de signature.*

La composition donne le protocole *ISO-9798-3* suivant :

1. $A \rightarrow B : g^a$
2. $B \rightarrow A : g^b, Sig_B(g^b, g^a, A)$
3. $A \rightarrow B : Sig_A(g^b, g^a, B)$

Protocole 6: *Le protocole ISO-9798-3*

À partir des modèles B événementiels du mécanisme de Diffie-Hellman avec notamment l'élément générateur g et du protocole 5 il est très aisé d'obtenir une composition de ces deux protocoles avec notre technique de composition. En effet, les nonces sont modélisés avec des transactions abstraites, il suffit donc d'effectuer un simple raffinement de données en raffinant les transactions aux type correspondant à g^x et d'effectuer ensuite la composition par fusion des événements qui est dans ce cas triviale (sans obligations de preuve supplémentaire) car les deux protocoles composés ne partagent pas de fragments ayant la

même structure. La composition séquentielle correspond de manière générale à un raffinement de données suivi d'une composition par fusion des événements.

Contrairement à la logique PCL qui détermine la façon avec laquelle les compositions sont réalisées (parallèles et séquentielles) nous ne fixons aucune limite pour la façon de composer les protocoles.

Notre démarche permet de non seulement pouvoir fournir une preuve de correction des protocoles et de leurs combinaison mais elle permet également de dire qu'une combinaison est impossible et de fournir d'éventuelles attaques grâce à l'utilisation des plus faibles invariants. Ceci n'est, semble-t-il, pas le cas de la logique PCL qui permet d'affirmer qu'une composition est correcte en vérifiant que les propriétés d'invariance de chacun des protocoles sont maintenues par l'autre protocole, mais en cas d'échec de la preuve de composition rien ne peut être déduit.

Nous avons également vu qu'un mécanisme permet d'apporter des propriétés supplémentaires à un protocole si celui-ci garantit déjà certaines propriétés comme le mécanisme 9 qui apporte à un protocole la propriété de la fraîcheur de la clé si celui-ci garantit déjà la propriété *key secrecy*. Un même mécanisme peut ainsi ré-appliqué sur un plus grand nombre d'études de cas.

Nos travaux se sont orientés, pour l'instant, vers les propriétés d'authentification et d'établissement de clés, il est nécessaire de pouvoir exprimer d'autres types de propriétés, ceci constitue une perspective immédiate de notre travail.

Après les protocoles cryptographiques, nous avons envisagé d'étendre la combinaison à d'autres protocoles garantissant des propriétés de sécurité autres que les propriétés cryptographiques.

6.5 La composition avec des protocoles non cryptographiques

Nous avons modélisé dans le cadre d'une étude de cas un système de porte-monnaie électronique appelé Mondex. Mondex [78] est un système de paiement électronique qui fut introduit pour la première fois par la *National Westminster Bank* en 1990 avant d'être repris par *MasterCard*. Le système est basé sur les cartes à puce servant de porte-monnaie électroniques contenant chacun un solde d'argent. Une des spécificités du système est que les transactions sont *hors-ligne*, ceci signifie que les transactions se passent entre deux porte-monnaie sans l'intervention d'une autorité centralisée (voir figure 6.5). Au cours d'une transaction, chaque porte-monnaie doit donc être capable de garantir lui-même les mesures de sécurité inhérentes à ce genre de système, sans l'intervention d'une quelconque autorité de régulation. Le protocole Mondex garantit les propriétés suivantes :

- Pas d'argent créé dans le système.
- Pas d'argent perdu dans le système.

Mondex ne fournissait pas de garantie sur les identités des deux portes-monnaie, en d'autres termes, le système garantit qu'il n'y a pas de création ni de disparition d'argent mais ne garantit pas que l'argent aille dans le bon porte-monnaie d'où l'idée de combiner ce protocole avec un protocole d'authentification.

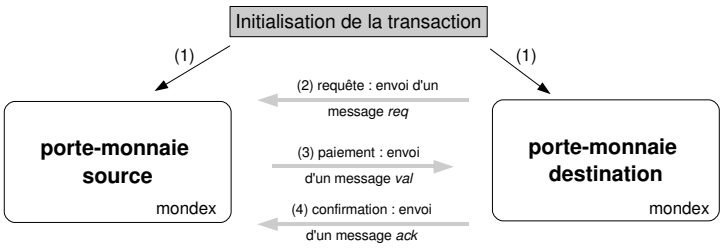


FIG. 6.5 – Le protocole Mondex.

Grandy et ses coauteurs [50] apportent un traitement de la vérification du Mondex en intégrant les éléments propres à la cryptographie ; leur approche est fondée sur une hypothèse de choix de la méthode cryptographique retenue et sur une preuve à l'aide de l'outil KIV. Cet exercice nous est apparu comme intéressant et nous avons donc estimé qu'il serait plus opportun d'utiliser une approche basée sur la composition des protocoles, surtout que notre modélisation du protocole Mondex est basée sur l'utilisation de transactions abstraites comme dans les protocoles cryptographiques. Nous avons donc fusionné la spécification du protocole Mondex avec le modèle abstrait de la propriété d'authentification pour obtenir une spécification à la fois des propriétés de préservation d'argent et d'authentification. La fusion des modèles concret a été réalisée de la même manière que celle de la section 6.3.2 où nous avons fusionné les modèles concrets des mécanismes cryptographiques. Une description détaillée de notre modèle B événementiel du protocole Mondex et de la composition avec un protocole d'authentification peut être trouvée dans notre publication [21].

6.6 Conclusion

Nous avons présenté dans ce chapitre notre méthode pour la modélisation et la composition des protocoles de sécurité, cette approche peut être utilisée pour prouver qu'un protocole complexe est correct en réutilisant les preuves de chacun de ses composants. Elle peut également être utilisée pour garantir que les exécutions de plusieurs protocoles dans un même environnement puissent se dérouler sans porter préjudice aux propriétés de sûreté de chacun de ces protocoles. Nous avons veillé à proposer une démarche qui soit aussi automatique que possible même si l'ajout des invariants issus du calcul de la plus faible pré-condition se fait, pour l'instant, manuellement mais l'ensemble de la démarche est susceptible d'être automatisée. Nous avons également fourni une méthode de reconstitution des attaques avec une démarche qui combine le calcul de la plus faible pré-condition et du model-checking. Une perspective immédiate de ce travail consiste à étendre la démarche pour combiner les protocoles cryptographiques avec des protocoles garantissant divers propriétés de sûreté et notamment les propriétés d'autorisation.

Troisième partie

L'implémentation des politiques de contrôle d'accès

Chapitre 7

Le contrôle d'accès : État de l'art

7.1 Introduction

Contrôler l'accès aux données et ressources est vital pour assurer le bon fonctionnement d'un système informatique. Par "bon fonctionnement" nous entendons que le système doit satisfaire un ensemble d'exigences exprimées sous la forme de propriétés de sécurité. Les propriétés de sécurité les plus importantes sont la *confidentialité* qui garantit qu'une donnée n'est lue ou écrite que par un utilisateur autorisé à le faire, *l'intégrité* garantit qu'aucune donnée du système ne puisse subir aucune altération. Satisfaire ces propriétés requiert un contrôle rigoureux de tous les accès aux données et ressources du système, ceci est réalisé par la mise en place de politiques de contrôle d'accès. Une politique de contrôle d'accès définit de manière exhaustive toutes les actions qu'un utilisateur est autorisé à effectuer sur les données ou ressources d'un système.

Le contrôle d'accès permet de déterminer si un utilisateur A avait le droit ou non d'accéder à une ressource. Ces politiques ne garantissent pas cependant que la requête en question vient effectivement de l'utilisateur A. En réalité les politiques de contrôle d'accès permettent de déterminer si un utilisateur *qui prétend être* A à le droit ou non d'accéder à une ressource donnée. Déterminer si la requête vient effectivement de l'utilisateur A et non pas de quelqu'un qui veut se faire passer pour A est un problème d'*authentification* que nous avons abordé dans la première partie de cette thèse. Les problème d'*autorisation* (contrôle d'accès) et d'authentification ont longtemps été abordés séparément dans la littérature mais aujourd'hui le contrôle d'accès doit souvent se faire sur des données ou des services distants entre des entités qui ne savent pas grand chose les unes des autres d'où la nécessité de voir les choses différemment et de traiter ensemble les problème d'autorisation et d'authentification. Comme un premier pas dans cette démarche, nous avons entrepris de modéliser différentes politiques de contrôle d'accès avec des modèles B événementiels et de vérifier que leur implémentation est correcte. Il sera par la suite possible d'envisager de combiner ces modèles dont la correction de l'implémentation est vérifiée avec des modèles de protocoles garantissant les propriétés d'authentification. Nous commencerons par présenter l'état de l'art dans le domaine du contrôle d'accès, nous verrons

ensuite comment les politiques de contrôle d'accès sont modélisées et comment nous vérifions leur implémentation.

Nous verrons plus loin dans ce chapitre qu'une politique de contrôle d'accès peut décrire non seulement les actions qu'un utilisateur est autorisé à effectuer mais également contenir des interdictions, des obligations voire des recommandations. Il est important de préciser qu'une politique de sécurité plus stricte n'est pas forcément meilleure qu'une politique de sécurité plus permissive. En effet autant certains systèmes ont besoin de politiques de sécurité strictes, d'autres, en revanche, requièrent plus de flexibilité pour leur bon fonctionnement. La mise en œuvre d'une politique de contrôle d'accès dans un système passe en général par trois étapes principales :

- Définir la politique de sécurité : cette étape consiste à établir de manière abstraite les règles qui régulent le contrôle d'accès aux ressources du système.
- Écrire le modèle de la politique de contrôle d'accès : cette étape consiste à décrire de manière formelle la politique de sécurité décrite dans l'étape précédente.
- La mise en œuvre de la politique de contrôle d'accès : cette étape consiste à implémenter la politique de sécurité formellement décrite dans l'étape précédente, en utilisant un ensemble de mécanismes de bas niveau. Ces mécanismes peuvent être soit logiciels soit matériels.

Il est important de préciser que les modèles de description de politiques de contrôle d'accès sont indépendants des mécanismes utilisés pour leur mise en oeuvre. Cette distinction entre le modèle formel abstrait et sa mise en œuvre concrète permet d'étudier les deux séparément : il est ainsi possible de prouver des propriétés sur un modèle de politique de contrôle d'accès ou encore de comparer deux modèles différents indépendamment de leur mise en œuvre respective. D'un autre côté, cette distinction nous permet également d'utiliser un même mécanisme pour implémenter différentes politiques de sécurité. Si un mécanisme était conçu pour mettre en œuvre une politique de sécurité particulière, tout changement dans cette dernière impliquerait inévitablement de changer l'ensemble du système. Il apparaît donc que cette séparation entre le modèle de la politique de sécurité et le mécanisme de sa mise en œuvre a des avantages indéniables. Cependant ceci a un prix, il est nécessaire de prouver que le mécanisme implémente correctement le modèle de la politique de sécurité. Avec la complexité grandissante des politiques de contrôle d'accès cette tâche devient de plus en plus délicate. Globalement, si des propriétés de sûreté sont prouvées correctes dans le modèle de la politique de sécurité et qu'en plus l'implémentation de la politique de sécurité avec un mécanisme particulier est également prouvée, il en résulte que le système obtenu satisfait ces propriétés de sûreté.

7.2 Les modèles de description de politiques de sécurité

Il y a un large éventail de modèles de description de politiques de contrôle d'accès. Avant de présenter les modèles les plus importants, il est nécessaire d'introduire quelques notions préliminaires :

7.2.1 Notions préliminaires

7.2.1.1 Utilisateurs, sujets et objets

Les sujets soumettent au système des requêtes pour accéder aux objets. Ainsi les sujets sont les entités *actives* du système tandis que les objets sont les entités *passives* du système. Il est cependant important de distinguer les utilisateurs des sujets. Les sujets agissant au nom des utilisateurs, les utilisateurs correspondent, par exemple, aux numéros utilisateurs dans un système et les sujets aux processus lancés avec ces numéros d'utilisateur. Un sujet correspond à un seul utilisateur, par contre plusieurs sujets peuvent être associés à un utilisateur. Dans certains cas, une entité peut être au même temps un sujet et un objet du système.

Pour rappel, pour un ensemble E et une relation d'ordre $\leq$, un treillis est une structure ordonnée $(E, \leq)$ toute paire d'élément $\{a, b\}$ admet une borne supérieure et une borne inférieure.

7.2.2 Les modèles de contrôle d'accès obligatoire

Dans les politiques de contrôle d'accès obligatoires (MAC : Mandatory Access Control), les règles du contrôle d'accès sont établies et imposées aux utilisateurs par une autorité centrale. Même le propriétaire d'une ressource ne peut établir lui même les conditions d'accès à cette ressource. La forme la plus utilisée pour le contrôle d'accès obligatoire (également appelé mandataire) est l'utilisation de classes de sécurité auxquelles appartiennent aussi bien les sujets que les objets du système. Une relation de dominance $\leq$ est définie sur l'ensemble des classes de sécurité (CS). L'ensemble des classes de sécurité CS muni de la relation de dominance $\leq$ est un treillis. La sémantique de la relation $\leq$ ainsi que la classification des différentes entités du système dans différents niveaux de sécurité sont établies selon que le système est conçu pour garantir la confidentialité ou l'intégrité des données. On peut donc identifier deux types de modèles de descriptions de politiques de contrôle d'accès selon que ceux-ci garantissent la confidentialité ou l'intégrité des données.

7.2.2.1 Modèles garantissant la confidentialité

Les modèles garantissant la confidentialité ont pour but d'éviter que des informations sensibles ne soient divulguées à des utilisateurs qui n'en ont pas le droit. Dans ces modèles, la sémantique de la relation de dominance $\leq$ est telle que deux classes *sc1* et *sc2* avec $sc2 \leq sc1$ alors le flot d'information est autorisé dans le sens *sc2* vers *sc1*. Ceci implique que pour les objets, elle reflète le degré d'importance de l'information contenue dans l'objet, ainsi, un objet o_1 appartenant à une classe cs_1 contient des informations plus sensibles qu'un objet o_2 appartenant à une classe cs_2 si $cs_2 \leq cs_1$. Pour les sujets, la relation $\leq$ traduit le degré de fiabilité d'un sujet dans le système : un sujet s_1 appartenant à une classe cs_1 est moins enclin à divulguer des informations sensibles qu'un sujet s_2 appartenant à une classe cs_2 si $cs_2 \leq cs_1$. Un utilisateur peut se connecter au système avec n'importe quelle classe dominée par la classe à laquelle il appartient. Une fois connecté à une classe donnée, l'utilisateur créera un sujet appartenant à cette classe avec lequel il fera des requêtes pour accéder à un objet donné. La suite donnée à cette requête dépend de deux

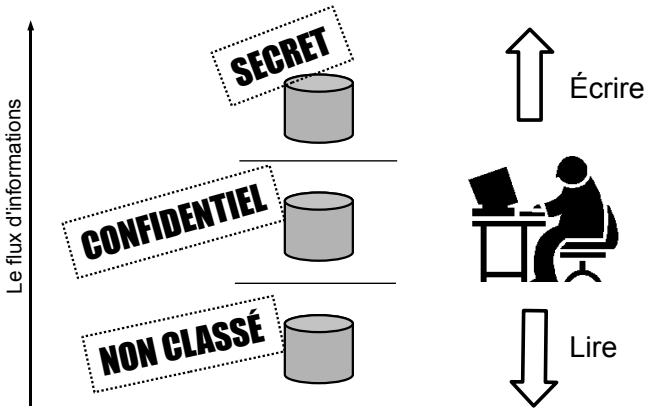


FIG. 7.1 – Les propriétés "No read up" et "No write down"

règles introduites pour la première fois par Bell et LaPadula [17] lorsqu'ils ont introduit leur modèle pour le contrôle d'accès.

7.2.2.1.1 Le modèle de Bell-LaPadula Le modèle de Bell et LaPadula contient, en plus du treillis $(CS, \leq)$, l'ensemble des sujets S, l'ensemble des objets O, l'ensemble des actions (*lire, écrire,...*) qu'un sujet peut effectuer sur un objet A, ainsi qu'une fonction $\lambda \in S \cup O \rightarrow CS$ indiquant la classe à laquelle est connecté chaque sujet ou objet du système. Le système est modélisé sous la forme de machine à états où chaque état est un triplet (b, M, λ) où :

- $b \in \mathbb{P}(S \times O \times A)$ est l'ensemble des accès courants dans le système.
- M est la matrice des permissions d'accès associant les objets aux sujets en indiquant les droits qui s'y appliquent.

Afin de garantir la confidentialité, Bell et LaPadula ont introduit deux règles régissant l'accès aux données (Voir figure 7.1) :

- La propriété simple (ou "No read up") : Un état (b, M, λ) est dit sûr si et seulement si :
  $\forall (s, o, lire) \in b.\lambda(o) \leq \lambda(s)$.
- La propriété * (ou "No write down") : Un état (b, M, λ) est dit sûr si et seulement si :
  $\forall (s, o, ecrire) \in b.\lambda(s) \leq \lambda(o)$.

Ces deux règles empêchent toute fuite d'informations d'une classe donnée vers une classe inférieure. Ceci est illustré par la figure 7.1 où, par exemple, un sujet avec un niveau de sécurité *Confidentiel* ne peut lire une information classée *Secret* et l'écrire sur un objet *non classifié*.

Le modèle de Bell et LaPadula présente néanmoins certains inconvénients, en effet, le fait d'empêcher les sujets d'écrire sur les objets de classe inférieure

les oblige à surclasser ces objets pour pouvoir les modifier. A terme, la surclassification des objets entraine une dégradation du système, une solution trouvée pour ce problème consiste à déclasser ces objets à l'aide d'un processus dit de confiance qui outrepasse les règles du modèle. Une autre faille dans le modèle de Bell et LaPadula fut présentée par McLean en 1987 [63], celui-ci montra qu'un système appelé Système Z satisfaisant les propriétés énoncées par Bell et LaPadula n'est pas sûr pour autant. L'ensemble des objets du système est déclassé par un sujet de niveau minimal qui autorise donc l'accès à ces objets à tous les sujets du système, ceci est possible si le niveau de sécurité d'un objet est contenu dans l'objet lui même et peut être modifié par un sujet de niveau minimal du système car l'écriture dans les niveaux dominant est autorisée.

7.2.2.2 Modèles garantissant l'intégrité des données

Dans les modèles garantissant la confidentialité vus dans le paragraphe précédent, un sujet avec une classification faible peut écrire dans des objets avec une classification supérieure ce qui constitue une menace pour l'intégrité des données. En effet, si par exemple un cheval de Troie est implanté dans l'ordinateur d'un utilisateur du système et qu'il crée un processus (un sujet), ce dernier peut librement écrire sur les objets du système avec une classification supérieure. Un modèle dual à celui proposé par Bell et LaPadula fut proposé par Biba en 1977 [23]. Dans le modèle de Biba, la sémantique de la relation de dominance $\leq$ est la suivante : pour les objets, elle reflète l'importance des dégâts potentiels engendré par une altération de l'objet. Ainsi, un objet o_1 appartenant à une classe cs_1 contient des informations plus importantes qu'un objet o_2 appartenant à une classe cs_2 si $cs_2 \leq cs_1$. Pour les sujets, la relation $\leq$ traduit la confiance qu'on a dans ce sujet : un sujet s_1 appartenant à une classe cs_1 est moins enclin à altérer des informations sensibles qu'un sujet s_2 appartenant à une classe cs_2 si $cs_2 \leq cs_1$. Biba introduit deux règles duales à celles de Bell et LaPadula :

- La propriété "No read down" : Un sujet n'est autorisé à lire un objet seulement si la classe de l'objet domine celle du sujet.
- La propriété "No write up" : Un sujet n'est autorisé à écrire dans un objet seulement si la classe du sujet domine celle de l'objet.

Le but de ces deux règles est d'empêcher qu'une information contenue dans un objet avec une classification faible (et donc une information peut fiable) ne soit retranscrite dans un objet avec une classification plus haute (voir figure 7.2).

Le modèle de Biba présente des inconvénients similaires à ceux du modèle de Bell et LaPadula : cette fois ce sont les sujets qui se retrouvent en bas de l'échelle de classification. Ceci est dû au fait qu'un sujet est dégradé s'il veut accéder à un objet de classe dominée par la classe de l'objet.

7.2.3 Les modèles de contrôle d'accès discrétionnaire

Le contrôle d'accès discrétionnaire donne (ou refuse) l'accès à un objet pour un sujet donné sur la base de l'identité de ce dernier. Ce type de contrôle d'accès est dit discrétionnaire car un sujet est habilité à transmettre ses privilèges à d'autres sujets en d'autres termes il est à la discrétion du propriétaire d'un

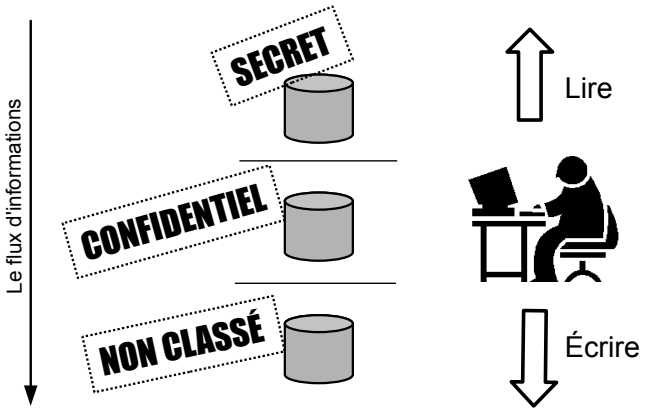


FIG. 7.2 – Les propriétés "No read down" et "No write up"

objet de définir les droits d'accès des autres sujets sur cet objet. Le modèle discrétionnaire le plus basique est celui à base de matrice d'accès.

7.2.3.1 le modèle à base de matrice d'accès

Ce modèle fût proposé pour la première fois par Lampson [59] où une matrice est utilisée pour mémoriser les privilèges d'utilisateurs pour des systèmes d'exploitation. Le concept de la matrice de privilège fut repris plus tard par Harrison, Ruzzo et Ullman qui étendirent le modèle d'origine et introduisirent un nouveau modèle connu sous le nom du modèle HRU [52]. Dans le modèle HRU, les sujets (contenus dans l'ensemble S) ont des privilèges sur l'ensemble des objets (O). Ces privilèges sont soit des actions classiques telles que la lecture, l'écriture ou l'exécution, soit des privilèges propres au modèle telles que *la possession*. L'ensemble de ces privilèges est contenu dans une matrice. La matrice des privilèges peut être modifiée de manière discrétionnaire par l'intermédiaire de commandes spéciales. Le modèle HRU contient entre autres une commande qui permet à un sujet de transmettre des privilèges qu'il a sur un objet à d'autres utilisateurs. Il y également des commandes qui permettent de créer/supprimer des sujets et des objets.

Les politiques de contrôle d'accès souffrent d'un inconvénient important qui est la vulnérabilité aux chevaux de Troie. Ceci est dû d'une part au fait qu'il n'y a pas de distinction entre utilisateur et sujet (comme dans les politiques mandataires) et d'autre part au fait qu'elles ne contiennent pas de mécanismes de contrôle du flux d'informations. Un programme cheval de Troie exécuté par un utilisateur peut à l'insu de ce dernier utiliser ses privilèges pour transférer des informations de manière malveillante.

Les modèle d'accès discrétionnaires furent enrichis avec le temps par des fonctionnalités supplémentaires telles que la notion de *groupe*, un utilisateur peut dans ce cas appartenir à un ou plusieurs groupes et ainsi hériter des priv-

ilèges associés aux groupes auxquels ils appartiennent. Des hiérarchies peuvent également être établies sur les groupes ou même sur les objets. Par exemple, les fichiers et les répertoires d'un système de fichier sont organisés de manière hiérarchique, dans ce cas un sujet a besoin du privilège "exécuter" sur le répertoire afin de pouvoir manipuler les fichiers contenus dans ce répertoire.

7.2.3.1.1 Permissions et interdictions Dans le modèle HRU, les autorisations spécifient les permissions, un sujet est autorisé à faire une action sur un objet s'il y a une permission explicite dans la matrice des privilèges. Dans d'autres modèles il est possible de spécifier des interdictions, dans ce cas, un sujet est autorisé par défaut à faire une action sur un objet sauf s'il existe une interdiction à la faire, on parle dans ce cas d'autorisations négatives. Certains modèles combinent les permissions et les interdictions, par exemple dans un modèle discrétionnaire avec des groupes d'utilisateurs, si on veut accorder un privilège à l'ensemble d'un groupe sauf un utilisateur particulier, une permission est accordée au groupe et une interdiction pour l'utilisateur en question est ajoutée. Combiner les interdictions et les permissions a cependant un prix car des problèmes d'incomplétude et d'inconsistance peuvent apparaître :

- L'incomplétude : si pour une requête d'accès particulière aucune autorisation (permission ou interdiction) n'est spécifiée.
- L'incohérence : si pour une requête donnée il y a en même temps une permission et une interdiction.

L'étude de la complétude et de la cohérence d'une politique de sécurité se situe dans la première problématique identifiée dans le début de ce chapitre, à savoir prouver des propriétés sur une politique de sécurité indépendamment de son implémentation. Diverses solutions ont été proposées pour ces problèmes : pour l'incomplétude il suffit de considérer que par défaut une permission ou une interdictions est signifiée à l'utilisateur. Pour l'inconsistance une solution consiste à introduire des règles de résolution de conflits, un exemple de règle est de décider que les interdictions prévalent sur les permissions ou encore d'associer un niveau de priorité à chaque permission/interdiction.

7.2.4 Modèle de la muraille de Chine

Les modèles de contrôle d'accès obligatoire ou discrétionnaire présentent chacun des inconvénients majeurs, le contrôle d'accès discrétionnaire ne permettant pas de contrôler les flux de données et sont vulnérables à certaines attaques de type cheval de Troie, tandis que le contrôle d'accès obligatoire est souvent jugé trop rigide. Des modèles combinant aussi bien des propriétés obligatoires et discrétionnaires sont apparus pour tenter de remédier à ces lacunes. Le modèle de la muraille de Chine est un exemple de ces modèles hybrides. Ce modèle fut introduit en 1989 par Brewer et Nash [30]. Le principe de base de ce modèle est que les privilèges sont donnés à un utilisateur dynamiquement en fonction de ses accès antérieurs, le but étant d'éviter d'éventuels conflits d'intérêts. Pour ce faire, en plus de l'ensemble des sujets S et de l'ensemble des objets O, un ensemble de compagnies C est introduit. Les objets sont organisés en trois niveaux (voir figure 7.3) :

- Les objets de base : sont l'ensemble des objets du système, chaque objet appartenant à une compagnie.

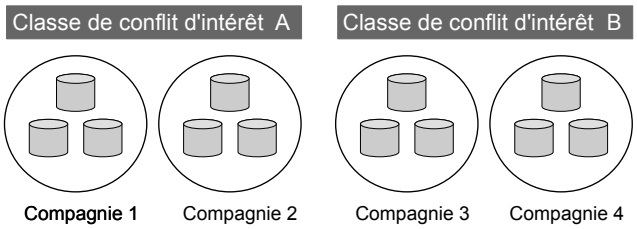


FIG. 7.3 – L'organisation des objets sous le modèle de la muraille de chine.

- Les ensembles de données de chaque compagnie (ou *company datasets*) : contient les objets appartenant à une même compagnie.
- Les classes de conflit d'intérêt : contient les ensembles de données des compagnies qui ont un conflit d'intérêt.

L'accès aux objets est régi par deux règles l'une concerne la lecture et l'autre l'écriture :

- La règle de lecture : un sujet s est autorisé à faire une écriture sur un objet o dans les deux cas suivants :
  1. Si o est dans les ensembles de données que d'autres objets déjà lus par s.
  2. Si o est dans une classe de conflit d'intérêt dans laquelle le sujet s n'a accédé aucune information.

  A chaque fois qu'un sujet accède à un objet, une *muraille de chine* s'érige pour ce sujet autour de l'ensemble de données auquel appartient l'objet. La règle de lecture à elle seule n'est pas suffisante pour empêcher certains flux d'information indirecte. Par exemple, si un sujet *s1* peut lire dans les ensembles de données de la *compagnie 1* et de la *compagnie 3* et qu'un sujet *s2* peut lire dans les ensembles de données de la *compagnie 2* et de la *compagnie 3*, dans ce cas *s2* ne devrait pas accéder aux données de la *compagnie 1* (car elle est dans la même classe de conflit d'intérêt que la *compagnie 2*) mais le sujet *s1* peut contourner cette restriction en lisant dans les données de la *compagnie 1* et en écrivant dans les données de la *compagnie 3* que *s2* pourra lire. Pour contrôler tous les flux de données, une règle d'écriture est ajoutée.
- La règle d'écriture : un sujet s est autorisé à faire une lecture sur un objet o si les deux conditions suivantes sont réunies :
  1. s est autorisé à lire o selon la règle de lecture.
  2. s ne peut lire aucun objet o' appartenant à un ensemble de données différent de celui de o.

  La deuxième condition de la règle d'écriture est rigide et des solutions ont été proposées pour l'assouplir en introduisant la notion de données *aseptisées*. Une façon d'aseptiser les données est de les débarrasser d'informations sensibles, d'autres techniques proposent de cacher l'identité des

compagnies à qui appartiennent les données. La seconde condition de la règle d'écriture devient : s ne peut lire aucun objet o' appartenant à un ensemble de données différent de celui de o et *contenant des données non aseptisées.*

7.2.5 Les modèles de contrôle d'accès basé sur les rôles

Dans la pratique, la taille des systèmes d'information devient de plus en plus grande avec un grand nombre de sujets et d'objets ce qui induit une croissance du nombre d'autorisations à gérer d'où la nécessité d'avoir des politiques de contrôle d'accès capables de gérer les grands systèmes. Certains systèmes sont même constitués de plusieurs organisations indépendantes qui interagissent, ce genre de système est difficile à gérer avec une politique mandataire où l'administration est centralisée. De plus la notion de *possession* d'un objet par un sujet sur laquelle est basée les modèles discrétionnaires ne convient pas à beaucoup de systèmes où les données appartiennent aux sociétés plutôt qu'aux employés. Une alternative consiste à accorder les privilèges sur la base de la *fonction exercée* plutôt que sur la possession, ceci est le principe du contrôle d'accès basé sur les *rôles* (RBAC). Le contrôle d'accès basé sur les rôles à comme avantage de combiner la flexibilité des autorisations explicites avec des contraintes additionnelles relatives à l'organisation. Le contrôle d'accès basé sur les rôles fut introduit pour la première fois en 1996 par Ferraiolo et Kuhn [46], dans ce modèle les privilèges sont accordés aux rôles et non pas accordés directement aux utilisateurs. Les utilisateurs héritent des privilèges accordés aux rôles qu'ils jouent dans le système. Un grand nombre de modèles basés sur les rôles furent proposés et implémentés. Un premier standard pour le modèle RBAC connu sous le nom du standard NIST fût introduit en 2000 par Sandhu *et al* [74]. Plus récemment, en 2004, un deuxième standard a été publié sous le nom de ANSI (American National Standard for Information). Les deux standards ne différent pas sur les concepts de base du modèle RBAC.

Dans le standard NIST, le modèle RBAC est constitué de trois niveaux principaux [1], les fonctionnalités de chaque niveau incluant les fonctionnalités des niveaux précédents :

- Le noyau RBAC (*Core* RBAC ou *Flat* RBAC) qui contient les fonctionnalités de base du modèle.
- Le RBAC hiérarchique (*hierarchical RBAC*) : qui défnit une hiérarchie des rôles.
- Le RBAC avec contraintes (*constrained RBAC*) : permet de spécifier des contraintes pour l'applications des privilèges.

7.2.5.1 Le noyau RBAC

Le noyau RBAC contient les aspects essentiels du modèle, le concept de base est que les utilisateurs jouent des rôles et que les rôles ont des *permissions.* Ce modèle contient trois ensembles d'entités de base : les utilisateurs U, les rôles R et les permissions P (figure 7.5) :

[1] Le standard NIST définit un quatrième niveau appelé Symmetric RBAC mais celui-ci n'est pas concerné par les travaux menés dans cette thèse.

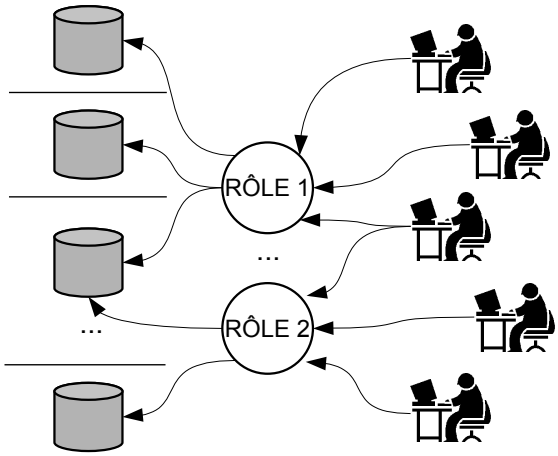


FIG. 7.4 – Le contrôle d'accès basé sur les rôles

- Un utilisateur est dans le modèle RBAC un humain, un processus ou un ordinateur.
- Un rôle est une fonction ou un titre dans une organisation à laquelle sont rattachés une autorité ou un ensemble de responsabilités.
- Une permission est une autorisation positive pour accéder à un objet, dans la communauté RBAC, les termes *autorisation*, *droit d'accès* ou encore *privilège* sont également utilisés pour désigner les permissions. La notion d'objet n'est pas incluse dans le noyau RBAC du standard NIST pour que le modèle soit le plus général possible, cependant pour un système donné les permissions peuvent être remplacées par un couple constitué d'une action et d'un objet(figure 7.6). Il faut également noter que le standard n'inclut pas d'interdictions mais les auteurs du standard précisent néanmoins que celles-ci peuvent être rajoutées au besoin.

FIG. 7.5 – Le noyau RBAC selon le standard NIST.

La comparaison entre les concepts de rôle et de groupe vus précédemment dans les modèles discrétionnaires a longtemps été un sujet de débat [74]. Le groupe

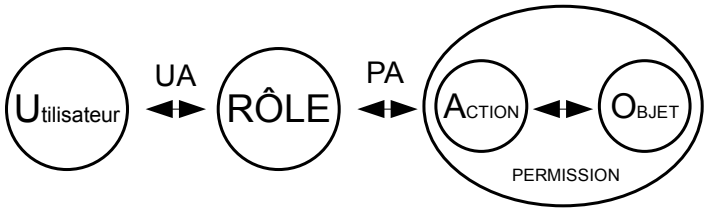


Fig. 7.6 – Les permissions remplacées par les actions et les objets.

est un ensemble d'utilisateurs tandis que les rôles sont un ensemble de privilèges mais les deux peuvent être utilisés de manière analogue. Les auteurs du standard NIST ont estimé que le modèle *Core RBAC* peut être considéré comme un modèle de contrôle d'accès basé sur les groupes mais que le modèle RBAC complet (avec tous les niveaux) a beaucoup plus de fonctionnalités et est considéré comme un modèle à part entière [74].

La figure 7.5 résume les relations entre les différentes entités du modèle RBAC. La relation entre utilisateurs et rôles (notée *UA* pour *user assignment*) est une relation *plusieurs-à-plusieurs*. Il en est de même pour la relation entre rôles et permissions (notée *PA* pour *permission assignment*).

7.2.5.2 Le RBAC hiérarchique

Le RBAC hiérarchique accorde les permissions du noyau RBAC et introduit en plus une hiérarchie des rôles. La hiérarchie des rôles est un moyen de modéliser les niveaux de responsabilité dans une organisation donnée. D'un point de vue mathématique, la hiérarchie des rôles est une relation d'ordre partiel entre les rôles notée RH. Si, par exemple, deux rôles *r1* et *r2* sont tels que $(r1, r2) \in RH$, alors une personne qui joue le rôle *r1* héritera également des privilèges (et des contraintes) du rôle *r2*. La figure 7.7 montre le modèle RBAC hiérarchique.

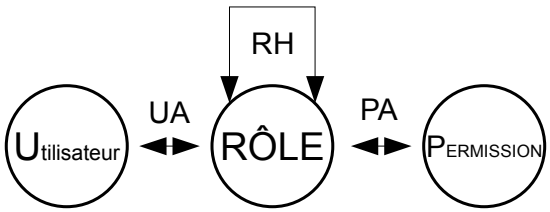


Fig. 7.7 – Le RBAC hiérarchique.

Souvent la hiérarchie des rôles est représentée dans la littérature comme une arborescence avec les rôles dominant en haut et les sous rôles en bas. Par exemple, dans la figure 7.8, le rôle *directeur* hérite des privilèges du rôle *chef de projet 1* qui lui même hérite des privilèges du rôle *employé*.

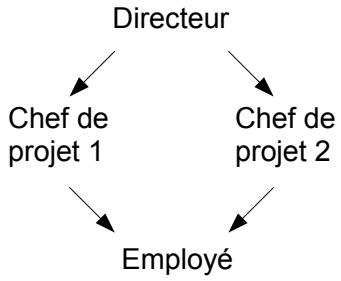


FIG. 7.8 – Un exemple d'un arborescence de rôles.

7.2.5.3 Le RBAC avec contraintes

Le RBAC avec contraintes permet d'exprimer des contraintes sur les rôles que peut jouer un sujet donné afin de mettre en place des mécanismes permettant d'éviter d'éventuels conflits d'intérêt. Le but étant d'éviter qu'un même sujet ne cumule trop d'autorité au sein d'une organisation. Cette démarche appelée "*séparation des pouvoirs*" (*SoD : separation of duty*) vise à impliquer autant d'intervenants que nécessaire dans un processus donné. Le standard NIST identifie deux types de séparation des pouvoirs :

7.2.5.3.1 La séparation statique des pouvoirs La séparation statique des pouvoirs (*SSoD : Static Separation of Duty*) vise à prévenir les conflits d'intérêt que peut engendrer l'affectation des rôles aux sujets. La séparation statique des pouvoirs empêche un sujet de jouer deux rôles conflictuels, cet ensemble de rôles conflictuels est défini au préalable dans la politique de sécurité, le standard NIST contient une relation *SSoD* qui indique les rôles en conflits. Il est important de préciser que ces contraintes sont héritées dans la hiérarchie des rôles, ceci a pour conséquence qu'à chaque fois qu'un rôle est affecté à un utilisateur, il faut vérifier que l'utilisateur ne joue aucun autre rôle en conflit avec son nouveau rôle soit directement, soit indirectement à travers la hiérarchie des rôles. Il faut également s'assurer que deux rôles liés par la hiérarchie des rôles ne soient pas en conflit l'un avec l'autre par rapport à la séparation statique des pouvoirs. Ce dernier point constitue un aspect de la cohérence des politiques RBAC.

7.2.5.3.2 La séparation dynamique des pouvoirs La séparation statique des pouvoirs peut parfois s'avérer rigide En effet, il n'est pas toujours nécessaire d'empêcher un utilisateur de jouer deux rôles en conflit, parfois il suffit de l'en empêcher de le faire au même temps, c'est la séparation dynamique des pouvoirs (*DSoD : Dynamic Separation of Duty*). Dans la séparation dynamique des pouvoirs, il y a uniquement un sous-ensemble des rôles assignés à un utilisateur qui sont *activés*. Contrairement à la séparation statique des pouvoirs, la séparation dynamique des pouvoirs peut concerner deux rôles dont l'un domine l'autre hiérarchiquement à condition bien sûr qu'il n'y ait qu'un seul d'entre eux qui soit activé. Les rôles actifs à un moment donné sont contenus dans des

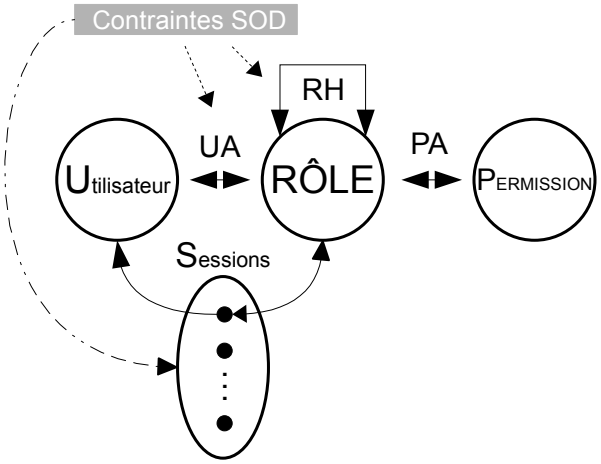


Fig. 7.9 – Le modèle RBAC avec contraintes *SoD*

sessions. Comme le montre la figure 7.9, une session indique qu'un utilisateur joue un ensemble de rôles particuliers à un moment donné.

Le modèle RBAC avec contraintes peut contenir des contraintes autres que la séparation des pouvoirs, le standard NIST ne pose pas de limites par rapport aux contraintes qui peuvent être définies. Par exemple, [76] introduit des contraintes sur le nombre de rôles que peut jouer un utilisateur au même temps, ce genre de contraintes sont connues sous le nom de *cardinality constraints*. Les mêmes auteurs ont également introduit le concept de *prerequisite roles*, selon ce concept, un utilisateur peut jouer un rôle uniquement si un autre rôle donné lui est déjà associé. De même pour les permissions, un rôle peut se voir attribué une permission donnée uniquement s'il possède déjà une autre permission définie à l'avance.

L'aspect administration n'est pas non plus abordé dans ce standard mais a été abondamment traité dans la littérature, nous reviendrons là-dessus plus loin dans le chapitre consacré à l'administration des modèles de description des politiques de contrôle d'accès. Les auteurs du standard ont délibérément laissé une grande marge de manœuvre aux utilisateurs du modèle en réduisant au minimum les concepts qui font partie du modèle afin de le rendre le plus général que possible. Cette simplicité du modèle RBAC a eu comme conséquence l'émergence d'un grand nombre de modèles basés sur RBAC avec des extensions qui permettent de modéliser des aspects spécifiques à un système donné :

- TMAC ou *Team Based Access Control* : a introduit la notion d'*équipe*.
- OrBAC ou *Organization Based Access Control* : a introduit la notion d'*organisation*.
- TRBAC ou *Temporal Role Based Access Control* : a introduit la notion de *temps*.

7.2.6 Le modèle de contrôle d'accès basé sur l'organisation

L'idée d'affecter des utilisateurs à des rôles a eu un grand succès après son apparition mais très vite des adaptations ont été proposées pour des utilisations particulières. En effet, parfois il était nécessaire qu'un utilisateur joue un rôle donné uniquement dans une partie du système, la notion d'*équipe* fut d'abord introduite par [71] dans le modèle TMAC. La notion d'équipe est transversale à l'affectation des sujets aux rôles ainsi un sujet s peut jouer un rôle *r1* uniquement dans une équipe *t1* et un autre rôle *r2* uniquement dans une autre équipe *t2*. La notion d'organisation est ensuite apparue dans le modèle OrBAC [8].

Le modèle OrBAC (Organization Based Access Control) est une extension de RBAC destinée à l'origine aux SICSS (modèles et politiques de sécurités pour les systèmes d'information et communication en santé et social), mais son utilisation englobe aujourd'hui tout domaine où une politique de sécurité doit être mise en œuvre. Comme son nom l'indique, OrBAC est un modèle basé sur la notion d'organisation, il permet ainsi de gérer simultanément plusieurs politiques de sécurité associées aux différentes organisations au sein d'un même système. OrBAC reprend la notion de rôle telle qu'elle a été définie dans RBAC, i.e. que les utilisateurs sont affectés à des rôles dont ils héritent des privilèges. Une notion de *vue* ou groupe d'objets est également introduite comme une abstraction des objets du système. La construction de ces groupes doit posséder une sémantique ; celle-ci est liée à la façon avec laquelle les différents rôles effectuent différentes actions sur ces objets. Il faut noter qu'il existe des similitudes avec la notion de vue telle qu'elle est utilisée dans les bases de données relationnelles où il s'agit de regrouper des objets qui ont des propriétés similaires. De même que pour les objets les actions sont également regroupées dans des activités, ceci implique qu'il y a dans OrBAC deux niveaux d'abstraction :

- Niveau concret : qui comporte les sujets, actions et objets recensés dans le système.
- Niveau abstrait : qui comporte les rôles, activités et vues sur lesquelles les différentes permissions et interdictions sont exprimées.

Les sujets, actions et objets sont respectivement affectés aux rôles, activités et vues. Le modèle OrBAC utilise les prédicats de la logique classique de premier ordre pour formaliser les différentes entités du système.

7.2.6.1 Les relations empower, use et consider

Les sujets sont affectés à un ou plusieurs rôles afin de bénéficier des privilèges de ceux-ci. Contrairement à RBAC, les sujet jouent des rôles dans des organisations, ce qui implique que les sujets sont affectés aux rôles avec une relation ternaire incluant l'organisation, un prédicat *empower* est utilisé comme suit : $empower(org, s, r)$ est vrai signifie que org habilite le sujet s à jouer le rôle r.

De même que pour les rôles et les sujets, les activités sont une abstraction des différentes actions autorisées dans le système. La relation liant les actions aux activités est également une relation ternaire incluant les organisations. Un prédicat *consider* est utilisé comme suit : $consider(org, a, act)$ est vrai signifie que l'action a est considérée comme une activité act dans l'organisation org.

Comme dans les bases de données relationnelles, une vue dans OrBAC correspond à un ensemble d'objets ayant une propriété commune. La relation liant les objets aux vues auxquelles ils appartiennent est aussi une relation ternaire

incluant l'organisation. Un prédicat *use* est utilisé comme suit : $use(org, o, v)$ est vrai signifie que l'organisation *org* utilise l'objet o dans la vue v.

7.2.6.2 Définition d'une politique de sécurité dans OrBAC

En plus des permissions, le modèle OrBAC permet également de définir des interdictions. Comme dans RBAC, les privilèges ne sont pas attribués directement aux sujets. Les privilèges sont définis sur les entités du niveau abstrait du modèle OrBAC (les rôles, activités et vues) :

- $permission(org, r, act, v, c)$: signifie que l'organisation *org* accorde au rôle r la permission de réaliser l'activité *act* sur la vue v dans le contexte c.
- $interdiction(org, r, act, v, c)$: signifie que l'organisation *org* interdit au rôle r la permission de réaliser l'activité *act* sur la vue v dans le contexte c.

La notion de contexte dans OrBAC permet d'exprimer des permissions (ou interdictions) contextuelles. Considérons l'exemple d'une politique de sécurité dans un environnement médical, si on veut restreindre l'accès aux fichiers malades à leurs médecins traitant, il faut rajouter la permission suivante :

$$permission(hopital, medecin, consulter, fichier\_malade, medecin\_traitant)$$

Pour pouvoir utiliser cette notion de contexte, un nouveau prédicat *define* est introduit dans le modèle OrBAC : $define(org, s, a, o, c)$ qui signifie qu'au sein de l'organisation *org*, le contexte c est vrai entre le sujet s, l'objet o et l'action a. La notion de contexte du modèle OrBAC correspond aux contraintes du modèle RBAC.

Le modèle OrBAC permet également de spécifier des obligations et des recommandations sans toutefois préciser comment celles-ci peuvent être mises en œuvre. À notre connaissance, les obligations ne sont prises en compte dans aucun autre modèle de politiques de sécurité, sauf peut-être pour être définies de manière abstraite comme des actions que doit exécuter un sujet et ce en réaction à un événement qui s'est produit dans le système.

7.2.6.3 Le formalisme de représentation d'une politique de sécurité dans OrBAC

Le langage défini par les concepteurs d'OrBAC pour représenter les différentes entités et relations vues précédemment est un sous-ensemble de la logique du premier ordre. Les symboles de constantes correspondent aux instances des entités : organisation, sujet, objet, action, rôle, vue, activité, contexte, les symboles de variables sont elles également typées de la même manière. Les symboles de constantes, et de variables constituent les termes de ce langage. Les symboles de prédicats correspondent aux relations définies précédemment dans notre langage, *empower*, *use*, *consider*, *permission*, *interdiction*, *define*. Les formules atomiques du langage sont construites à partir des termes (constantes et variables) reliés par les prédicats. Les formules du langages sont construites à partir des formules atomiques, des connecteurs booléens $\neg$, $\vee$, $\wedge$, $\Rightarrow$, et des quantificateurs $\forall$, $\exists$. La détermination de la valeur de vérité d'une formule s'effectue de manière classique en logique du premier ordre.

7.2.6.4 Hiérarchies dans OrBAC

Le modèle OrBAC permet de définir des hiérarchies de rôle (comme dans RBAC) mais aussi des hiérarchies d'organisations, les hiérarchies permettent l'héritage des privilèges (permissions ou interdictions), si par exemple *r1* est un sous-rôle de *r2*, pour une organisation o, une activité a et une vue v dans le contexte ctx alors

$$permission(o, r2, a, v, ctx) \Rightarrow permission(o, r1, a, v, ctx)$$

Comme dans les politiques discrétionnaires, la présence des autorisations négatives pose le problème de la cohérence du modèle. Ce problème est encore plus sérieux dans OrBAC à cause de la présence des hiérarchies de rôles et d'organisations.

Comme nous le précisions pour le modèle RBAC, celui-ci a été conçu aussi simple que possible pour être général. Le soin était laissé aux utilisateurs de l'adapter à leurs besoins ce que les concepteurs de OrBAC ont fait pour les systèmes d'information et communication en santé et social. Le prix de ce gain en expressivité par rapport à RBAC est une perte en généralité pour le modèle.

7.3 Les politiques d'administration

Les politiques de contrôle d'accès régissent l'accès des utilisateurs aux objets. Cependant les privilèges des utilisateurs ne sont pas figés et peuvent évoluer dans le temps. Les politiques d'administration visent à déterminer qui a le droit d'attribuer (ou de révoquer) un privilège à un utilisateur donné. Ces politiques d'administration dépendent de la nature de la politique de sécurité. Pour les politiques mandataires, l'administration est généralement centralisée : un objet reçoit au moment de sa création le niveau de sécurité du sujet qui l'a créé mais seul un administrateur pourra par la suite changer les niveaux de sécurité d'un objet ou d'un sujet.

Les politiques d'administration des modèles discrétionnaires ou des modèles basés sur les rôles sont beaucoup plus variées.

7.3.1 L'administration des politiques discrétionnaires

Les politiques d'administration pour les modèles discrétionnaires sont variées :

- Administration centralisée : Un administrateur s'occupe de gérer les privilèges de tout le système. Il est possible d'avoir un groupe d'administrateurs ayant les mêmes prérogatives pour gérer les privilèges.
- Administration hiérarchique : Un "super" administrateur peut déléguer une partie de ses prérogatives à d'autres administrateurs qui, à leur tour, peuvent gérer les privilèges des autres utilisateurs.
- Administration coopérative (collégiale) : Dans ce cas les privilèges ne peuvent être accordés (ou enlevés) aux utilisateurs par un seul administrateur mais nécessite l'accord de plusieurs administrateurs à la fois.
- Administration basée sur la possession : Dans ce schéma, chaque objet appartient à un utilisateur et les privilèges sont accordés de manière discrétionnaire par chaque utilisateur sur les objets qu'il possède. Un objet est généralement possédé par l'utilisateur qui le crée.

- Administration décentralisée : Un utilisateur qui possède un objet peut déléguer la gestion des privilèges sur cet objet à d'autres utilisateurs. Dans certains cas ceux-ci peuvent même déléguer l'administration à leur tour à d'autres utilisateurs. Ce mode d'administration à l'avantage d'être flexible mais pose un certain nombre de questions parmi lesquelles :
  - L'utilisateur qui possède un objet ne contrôle plus l'accès à celui-ci et ne sait même plus qui est autorisé à y accéder une fois qu'il a délégué son administration à d'autres utilisateurs.
  - La révocation des privilèges devient complexe.
  - Que faire des privilèges accordés par un utilisateur dont on vient de révoquer les droits d'administrations ?

7.3.2 L'administration des politiques basées sur les rôles

Même si RBAC connait un grand développement, l'administration des modèles RBAC n'est pas encore arrivée à maturité notamment en ce qui concerne les systèmes de grande taille (il n'est pas rare d'avoir des systèmes avec plus d'un millier de rôles et des dizaines de milliers d'utilisateurs). Les solutions proposées sont basées sur le principe de décentralisation de l'administration, cette solution s'est imposée d'elle-même car il est évident qu'un groupe réduit d'administrateurs ne peut administrer des grands systèmes. Les modèles les plus connus sont le modèle ARBAC97 [73] et SARBAC [40]. Le principe de base des deux approches est de définir un ensemble de rôles particuliers avec une hiérarchie chargés de l'administration du système, les rôle non administratifs sont désignés par *rôles réguliers*.

7.3.2.1 Le modèle ARBAC97

ARBAC97 est le premier modèle d'administration proposé pour RBAC, il a été proposé pour le modèle RBAC96 [76]. Ce modèle distingue les rôles administratif des autres rôles du modèle RBAC. L'administration des privilèges des rôles administratifs est du ressort du responsable en chef de la sécurité (Chief Security Officer). Le modèle RBAC est constitué de plusieurs composants, chacun de ces composants doit être administré : création et suppression des rôles, et des permissions, affectation et retrait des permissions aux rôles, création et suppression des utilisateurs, affectation et retraits des utilisateurs aux rôles, la gestion de la hiérarchie des rôles ainsi que les contraintes du modèle RBAC. Afin de faciliter l'administration de l'ensemble de ces composants, le modèle ARBAC97 contient trois composants : URA97 (User Role assignement), PRA97 (Permission Role Assignement) et le RRA97 (Role Role Assignement).

7.3.2.1.1 Le modèle URA97 Le modèle URA97 permet à un ensemble d'utilisateurs affectés à un ensemble de rôles d'administrateurs (ensemble AR) de contrôler l'affectation des sujets aux rôles (l'ensemble des rôles réguliers est désigné par R). Il faut noter qu'une hiérarchie est également définie sur les rôles d'administration. Le modèle URA97 est constitué à son tour de deux parties, une partie gère l'affectation des utilisateurs aux rôles et l'autre la révocation. L'affectation des sujets aux rôles se fait à l'aide d'une relation appelée "*can_ assign*" qui détermine quel rôle administratif peut affecter un utilisateur à un rôle par-

ticulier. Avant de donner la définition formelle de *can_assign*, il faut d'abord introduire la notion de *condition pré-requise* [73] :

Définition 8 *Une condition pré-requise est une expression booléenne utilisant les opérateurs logiques* $\wedge$ *et* $\vee$ *sur des termes* x *et* $\overline{x}$*,* x *étant un rôle régulier* ($x \in R$)*. Une condition pré-requise est évaluée pour un utilisateur* u *en interprétant* x *à vrai si* $\exists x'.(x,x') \in RH \wedge (u,x) \in UA$ *et* $\overline{x}$ *est vrai si* $\forall x'.(x,x') \notin RH \vee (u,x) \in UA$*. Pour l'ensemble des rôles réguliers* R*, l'ensemble* CR *contient l'ensemble de toutes les possibles conditions pré-requises qui peuvent être formée avec les rôles dans* R*.*

La relation *can_assign* est définie comme suit :

$$can\_assign \subseteq AR \times CR \times 2^{\mathrm{R}}$$

$can\_assign(x, y, \{a, b, c\})$ signifie qu'un membre du rôle administratif x (ou d'un rôle supérieur hiérarchiquement) peut affecter un utilisateur satisfaisant la condition y à l'un des rôles a, b ou c.

De même que pour les affectations, une relation *can_revoke* est introduite pour l'administration de la révocation des utilisateur des rôles auxquels ils sont affectés, la relation *can_revoke* est définie comme suit :

$$can\_revoke \subseteq AR \times 2^{\mathrm{R}}$$

$can\_revoke(x, \{a, b, c\})$ signifie qu'un membre du rôle administratif x (ou d'un rôle supérieur hiérarchiquement) peut révoquer un utilisateur affecté à l'un des rôles a, b ou c.

7.3.2.1.2 Le modèle PRA97 Le modèle PRA97 est similaire au modèle URA97 avec deux relations : *can_assignp* et *can_revokep* qui déterminent comment est administrée l'affectation (la révocation) des permissions aux rôles.

7.3.2.1.3 Le modèle RRA97 L'administration de la hiérarchie des rôles consiste en l'ajout, la suppression ou la modification de la hiérarchie des rôles. Une relation *can_modify* est utilisée pour administrer toute modification sur la hiérarchie des rôles, elle est définie comme suit :

$$can\_modify \subseteq AR \times 2^{\mathrm{R}}$$

$can\_revoke(x, \{a, b, c\})$ signifie qu'un membre du rôle administratif x (ou d'un rôle supérieur hiérarchiquement) peut modifier la hiérarchie entre les rôles a, b ou c. L'administration de la hiérarchie des rôles peut, cependant, poser problème. Considérons l'exemple de la figure 7.11 : supposons qu'un rôle administrateur AdR a le droit de changer la hiérarchies des rôles $R3, R5, R6, R8$ avec la relation $can\_modify(AdR, \{R3, R5, R6, R8\})$. Si le rôle AdR décide d'introduire un lien $(R5, R9) \in RH$. Cette modification de la hiérarchie pose problème dans le sens où le rôle AdR introduit une hiérarchie entre des rôles qui ne font pas partie de l'ensemble $\{R3, R5, R6, R8\}$ tels que $R2$ et $R9$.

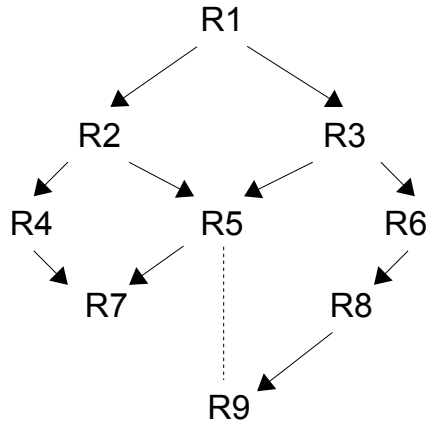


FIG. 7.10 – La modification de la hiérarchie des rôles.

Pour remédier à ce problème, le modèle RRA97 introduit les concepts d'*intervalle de rôles*, d'*intervalle d'autorité* et d'*intervalle encapsulé*. L'intervalle de rôle définit des intervalles basés sur la hiérarchie des rôles RH. pour définir la notion d'intervalle, nous avons besoin de la relation d'ordre RH au sens stricte sans la réflexivité, nous l'appellerons RHS.

Définition 9 *Soient x, y deux rôles tels que $(y, x) \in RH$, un intervalle de rôle (x, y) est définie par : $(x, y) = \{z \in R | (y, z) \in RHS \wedge (z, x) \in RHS\}$. x et y sont les extrémités de l'intervalle et n'appartiennent pas à celui-ci.*

Dans la figure 7.11, $(R2, R7) = \{R4, R5\}$.

Un intervalle encapsulé est défini comme suit :

Définition 10 *Un intervalle (x, y) est encapsulé si pour tout $r1 \in (x, y)$, $r2 \notin (x, y)$ on a :*

- $(r2, r1) \in RHS \Leftrightarrow (r2, y) \in RHS$
- $(r1, r2) \in RHS \Leftrightarrow (x, r2) \in RHS$

Intuitivement, un intervalle de rôle est encapsulé si tous les rôles qu'il contient ont une relation hiérarchique identique avec les rôles qui n'appartient pas à l'intervalle. Le but est que chaque administrateur ne puisse modifier que des hiérarchies dans des intervalles encapsulés.

L'intervalle d'autorité est défini comme suit :

Définition 11 *Tout intervalle appartenant au co-domaine de la relation can_modify est un intervalle d'autorité.*

Définition 12 *Deux intervalles X et Y se chevauchent partiellement si :*
$X \cap Y \neq \varnothing \wedge X \nsubseteq Y \wedge Y \nsubseteq X$

Le modèle RRA97 impose au moment de la définition de la relation *can_modify* que les intervalles d'autorité ne se chevauchent pas partiellement et qu'ils soient

encapsulés pour éviter qu'une modification de la hiérarchie dans un intervalle donné n'ait des effets de bords non désirés sur le reste de la hiérarchie des rôles.

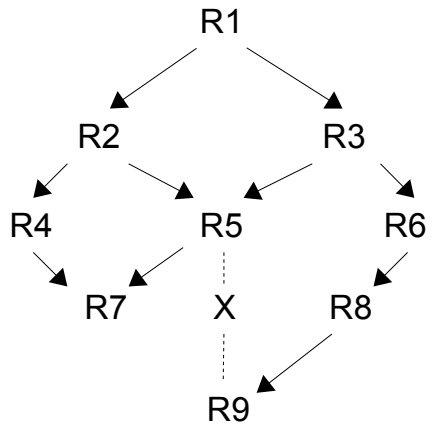


FIG. 7.11 – Création d'un rôle.

Le problème illustré dans la figure 7.10 peut également se présenter si le rôle *AdR* décide de créer un nouveau rôle X avec $(R5, X) \in RH$ et $(X, R9) \in RH$ (voir la figure 7.10). Cette modification de la hiérarchie pose problème dans le sens où le rôle *AdR* introduit une hiérarchie entre des rôles qui ne font pas partie de l'ensemble $\{R3, R5, R6, R8\}$ tels que *R2* et *R9*. Lorsqu'un rôle est créé dans le modèle RRA, l'administrateur insère le rôle créé dans la hiérarchie des rôles et doit lui désigner un rôle père et un rôle fils (sauf pour l'administrateur en chef du système). Pour éviter d'éventuels effets de bords lors de la création d'un rôle, les notions d'*intervalle d'autorité immédiate* et d'*intervalle de création de rôle* est introduite.

Définition 13 *L'intervalle d'autorité immédiate pour un rôle r notée* $IA_{immdiate}(r)$ *est l'intervalle d'autorité* (x, y) *tel que*
- $r \in (x, y)$
- $\forall (x', y').(x', y') \subset (x, y) \Rightarrow r \notin (x', y')$

Définition 14 *Un intervalle* (x, y) *est un intervalle de création si :*
- $IA_{immdiate}(x) = IA_{immdiate}(y)$ *ou*
- x *est une extrémité de* $IA_{immdiate}(y)$ *ou*
- y *est une extrémité de* $IA_{immdiate}(x)$

La figure 7.12 montre un exemple de hiérarchie de rôle. Si les intervalles d'autorité sont $(R1, R10)$ et $(R2, R8)$ alors quelques intervalles de création possibles sont en pointillés. $((R6, R1), (R8, R10), (R10, R2))$. Par contre, $(R4, R1)$ n'est pas un intervalle de création. Dans le modèle RRA97, les rôles père et fils d'un nouveau rôle créé doivent constituer une intervalle de création.

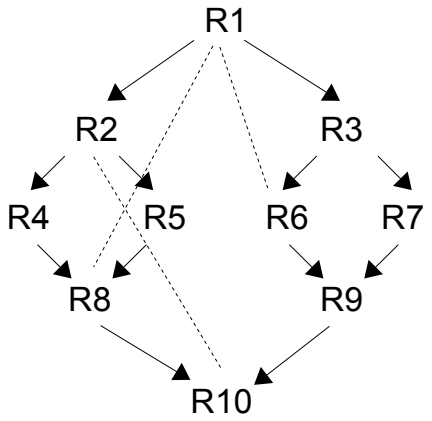


FIG. 7.12 – Les intervalles de création de rôles.

7.4 La formalisation des politiques de sécurité

Il existe un très grand nombre de formalisations de modèles de contrôle d'accès, nous présentons dans cette section certaines de ces formalisations. Les méthodes formelles utilisées sont :

- La logique : comme la logique de premier ordre, des systèmes formels tels que le λ-calcul et des langages de spécifications tel que Z [77].
- Les algèbres de processus.
- Les réseaux de Petri.

Le modèle OrBAC déjà présenté dans la section 7.2.6 utilise une formalisation basée sur la logique de première ordre. [55] utilise le λ-calcul typé pour formaliser RBAC, il a ainsi définit le modèle λ-RBAC. Dans [53], les auteurs utilisent ALLOY [54], un langage déclaratif de spécification pour spécifier le modèle RBAC. ALLOY est basé sur la logique du premier ordre et est très influencé par le langage Z [77]. [83] décrit les modèles RBAC en utilisant une modélisation avec le langage Z. [29] introduit une extension du π-calcul pour formaliser les modèles RBAC. Nous présenterons dans le chapitre 8 notre propre modélisation des modèles de contrôle d'accès.

Ces formalisations des politiques de contrôle d'accès sont utilisées pour la vérification de ces modèles. La vérification concerne deux aspects :

- Vérifier que la politique de sécurité est cohérente et complète, la cohérence garantit qu'il n'y a pas de conflits dans la politique telle qu'elle a été formalisée tandis que la complétude garantit qu'il y a dans la formalisation de la politique de sécurité assez d'éléments permettant de garantir le bon fonctionnement du système.
- Vérifier que l'implantation de la politique de sécurité est conforme à la politique telle qu'elle a été formalisée au préalable.

La vérification du premier aspect concernant la cohérence et la complétude d'une politique de sécurité est traitée en abondance dans la littérature. Deux méthodes sont utilisées : le *model-checking* et le *theorem proving*. À titre d'exemple, [45] utilise le théorème prouveur Isabelle pour prouver des propriétés sur

les modèles RBAC. Quand au méthodes utilisant le model-checking, elle sont très nombreuses, on peut citer entre autre les travaux de [13].

Les politiques de contrôle d'accès sont soit implémentée dans des applications dédiées à cette tâche soit elle sont implémentées dans des systèmes où une autre politique de contrôle d'accès existe déjà. C'est ce second aspect de l'implémentation qui nous intéresse, nous en discutons dans la section suivante.

7.5 L'implémentation des politiques de contrôle d'accès

La comparaison des modèles de contrôles d'accès a été largement abordée dans la littérature. Beaucoup de travaux ont été faits pour comparer l'expressivité des politiques de sécurité. La question posée ici est : est il possible d'implémenter une politique de contrôle d'accès A dans un système où une autre politique de contrôle d'accès est déjà implémentée B. Plus un modèle de contrôle d'accès est expressif, plus il peut implémenter de modèle de contrôles d'accès. Certains travaux utilisent le terme simulation pour parler d'implémentation mais il n'existe pas de consensus sur la définition exacte du concept de simulation (ou implémentation). La définition sur laquelle nous nous sommes basée est similaire à celle utilisée dans [75] :

Définition 15 *Un politique A est implémentée par une politique B si chaque accès autorisé dans la politique B est autorisé dans la politique A.*

Sandhu [75] a montré comment configurer le modèle RBAC pour implémenter des politiques de contrôle d'accès discrétionnaires. Thomas [81] a proposé une implémentation des politiques TMAC sur un système du type Security-Enhanced Linux. Glenn [48] a proposé des solutions pour implémenter RBAC sous un système Unix où les rôles sont implémentés sous la forme de comptes utilisateurs Unix. Mais la preuve de la correction de ces implémentations n'est pas abordée dans les études de cas que nous avons trouvées dans la littérature, hormis l'étude faite par Brucker et Wolf [31] qui proposent une preuve de correction de leur implémentation. Nous proposons dans le chapitre 9 une méthode pour la vérification de l'implémentation d'une politique de sécurité dans une autre politique de sécurité.

7.6 La composition des politiques de sécurité

Il peut s'avérer nécessaire de combiner plusieurs politiques de sécurité différentes afin de modéliser certains cas de figure qui se présentent dans la réalité où la politique de sécurité est une combinaison hétérogène de plusieurs politiques. C'est le cas, par exemple, d'une organisation formée de plusieurs départements qui ont des politiques de sécurité indépendantes qui sont modélisées dans des modèles de contrôle d'accès différents. Chaque politique doit être prise en considération tout en restant gérée de manière autonome. Un autre exemple est celui d'une politique de sécurité complexe qui, même si elle est gérée par un seul administrateur, peut être formulée de manière incrémentale en assemblant plusieurs modules développés séparément. Pour satisfaire ces besoins, Bonatti *et*

al [26] ont proposé un cadre pour la combinaison de politiques de contrôle d'accès sous forme d'une algèbre. Les politiques composées sont formulées à l'aide d'expressions algébriques construites avec des opérateurs auxquels est associée une sémantique.

Nous présentons dans cette section les opérateurs de cette algèbre en nous focalisant sur les opérateurs qui nous seront utiles dans la suite de cette thèse. Avant de détailler ces opérateurs, nous présentons des définitions où l'ensemble des sujets est noté S, celui des objets O et les actions A :

Définition 16 *Un terme d'autorisation est un triplet de la forme (s, o, a) où s est une constante ou une variable dans S, o est une constante ou une variable dans O et a est une constante ou une variable dans A. Un terme est dit de base s'il est constitué d'un triplet de constantes.*

La deuxième définition est celle d'une politique de sécurité :

Définition 17 *Une politique de sécurité est un ensemble de termes de base.*

La troisième définition est celle des *environnements* :

Définition 18 *Un environnement e est une fonction partielle qui associe à un identificateur d'une politique de sécurité un ensemble de termes de base.*

La sémantique d'un identificateur X sous un environnement e est noté $[\![X]\!]_e$ avec :

$$[\![X]\!]_e = e(X)$$

- L'addition : cet opérateur compose deux politiques en retournant leur union. Ceci a pour conséquence qu'un sujet donné peut obtenir une autorisation particulière s'il est autorisé à l'avoir dans l'une ou l'autre des politiques de sécurités composées. Cet opérateur est noté «+» et est défini formellement comme suit :

  $$[\![P_1 + P_2]\!]_e = [\![P_1]\!]_e \cup [\![P_2]\!]_e$$

  Un exemple où l'application de cet opérateur peut s'avérer utile est le cas de deux organisations qui partagent le même bâtiment alors l'entrée par la porte principale doit être autorisée aux personne qui ont cette permission dans l'une où l'autre des politiques de contrôle d'accès des deux organisations.
- La conjonction : Cet opérateur compose deux politiques en retournant leur intersection. Dans la politique composée, un sujet bénéficie d'une autorisation s'il en bénéficie déjà dans les deux politiques composées. Cet opérateur est noté «&» et est défini formellement comme suit :

  $$[\![P_1 \& P_2]\!]_e = [\![P_1]\!]_e \cap [\![P_2]\!]_e$$

  Un exemple d'application de cet opérateur est le cas de deux organisations partageant des documents confidentiels. L'accès à ces documents n'est autorisé a un sujet que si toutes les organisations qui ont un droit de regard sur le document en question l'y autorisent.

- La soustraction : cet opérateur compose deux politiques de sécurité en gardant les autorisations présentes dans la première politiques et pas dans la deuxième. Cet opérateur est noté «−», sa sémantique est comme suit :

$$[\![P_1 - P_2]\!]_e = [\![P_1]\!]_e \backslash [\![P_2]\!]_e$$

  Cet opérateur est introduit dans l'algèbre pour inclure les autorisations négatives qui sont souvent contenues dans les modèles de description de contrôle d'accès.
- La fermeture : cet opérateur permet de fermer la politique en utilisant des règles d'inférence préalablement établies.
- Délimitation de portée : cet opérateur permet de limiter l'application d'une politique de sécurité sur un ensemble limité de sujets, d'actions ou d'objets.
- Surcharge : cet opérateur remplace une partie d'une politique de sécurité par une autre.
- L'opérateur *Template* : permet de spécifier partiellement des politiques de sécurité à l'aide de paramètres, ces paramètres pourront être introduits plus tard pour compléter la politique de sécurité.

L'aspect modulaire de cette algèbre rend pratique la manipulation des différentes politiques de sécurité à différents niveau d'abstraction. Cet aspect nous sera d'une grande utilité dans la suite de notre thèse.

7.7 Conclusion

Les travaux de notre thèse se situent principalement dans le deuxième axe de vérification qui concerne la vérification de la correction de l'implémentation. Nous présentons dans le prochain chapitre une méthode basée sur le raffinement pour vérifier la correction de implémentation d'une politique de contrôle d'accès dans un système où une autre politique est déjà implémentée. Plusieurs études proposent des solutions possibles pour l'implémentation de politiques avec d'autres politiques de contrôles d'accès mais nous n'avons pas connaissance de méthode prouvant formellement la correction de cette implémentation.

Chapitre 8

La formalisation des politiques de sécurité avec le B événementiel

8.1 Introduction

Nous aborderons dans ce chapitre la formalisation des politiques de contrôle d'accès avec la méthode B événementielle. La formalisation est une étape préalable à la vérification. Nous avons présenté dans le chapitre 7.4 les différents formalismes qui ont déjà été utilisés pour formaliser les politiques de sécurité ; nous avons vu que le choix du formalisme dépend du type de propriétés de sécurité à vérifier. Dans notre cas, nous souhaitons prouver qu'une implémentation d'une politique de sécurité modélisée avec un modèle de contrôle d'accès particulier dans un système qui repose sur un autre modèle de contrôle d'accès est correcte en utilisant le raffinement. Avant de détailler la technique utilisée pour vérifier que l'implémentation est correcte, nous présentons d'abord dans ce chapitre comment sont formalisées les politiques de contrôle d'accès. Les politiques de contrôle d'accès formalisées en B événementiel seront utilisés dans le chapitre 9 pour vérifier la correction de l'implémentation.

Nous avons présenté dans le chapitre 7 différents modèles de description de politiques de sécurité ainsi que les méthodes utilisées pour l'administration des modèles. Nous présentons au début de ce chapitre comment sont formalisées en B événementiel les politiques de sécurité indépendamment du modèle avec lequel elles sont décrites avant de donner, comme exemple, comment sont formalisées les politiques modélisées avec le modèle RBAC.

8.2 Le modèles B événementiel général pour les politiques de contrôle d'accès

Le premier modèle en B événementiel présenté dans ce chapitre se veut général et indépendant du modèle de contrôle d'accès dans lequel est décrite la politique de sécurité. Ce modèle B événementiel, composé du contexte `CONTEXT_ACP`

et de la machine MACHINE_ACP sera par la suite adapté par un procédé de raffinement pour chaque modèle de contrôle d'accès.

8.2.1 Le contexte du modèle B événementiel : CONTEXT_ACP

La première étape de la formalisation des politiques de sécurité en B événementiel est le choix des types de base du modèle. Le système a une politique de sécurité à un instant donné qui est susceptible d'être modifiée par un administrateur. Tout accès dans le système doit être conforme à la politique de sécurité courante. Le premier type de base introduit dans notre modèle est le ACP (*Access control policy*), ce type englobe toutes les politiques de sécurité qui peuvent être formalisées dans le système. Nous introduisons également les types USER et PERMISSION. Le type USER correspond aux entités actives du système sur lesquelles sont définies les autorisations de la politique de sécurité. Le type PERMISSION contient les autorisations positives dont peut bénéficier un utilisateur du système. La notion de permission est laissée délibérément abstraite car la nature des permissions change d'un système à un autre, le type sera affiné selon la nature du système.

Une politique de sécurité est caractérisée par les privilèges accordés aux utilisateurs du système. Nous introduisons un paramètre ACP_PERMISSION qui contient l'ensemble des privilèges associés à chaque utilisateur. Ce paramètre est donc une fonction totale qui associe à chaque politique de sécurité, un ensemble de couples utilisateur/permission. D'un point de vue B événementiel, ce paramètre est une constante déclarée dans la clause CONSTANT du modèle B, ceci implique que les privilèges associés à une politique de sécurité sont constants. Ce choix peut paraître surprenant car une politique de sécurité n'est pas figée, mais au lieu de changer les attributs d'une politique de sécurité, c'est la politique de sécurité elle-même qui sera modifiée. Ce choix est motivé par notre volonté de simplifier les modèles B événementiels : il est en effet plus simple de changer la valeur de la variable qui contient la politique de sécurité courante que de changer l'ensemble des attributs de la politique de sécurité.

$$\mathtt{ACP\_PERMISSION} \in \mathtt{ACP} \rightarrow (\mathtt{USER} \leftrightarrow \mathtt{PERMISSION})$$

Si par exemple, pour un utilisateur *user*, une permission *perm* et une politique de sécurité *acp* on a :

$$(\mathtt{user} \mapsto \mathtt{perm}) \in \mathtt{ACP\_PERMISSION}(\mathtt{acp})$$

alors l'utilisateur *user* a la permission *perm*. L'ensemble de ces types et paramètres sont contenus dans le contexte CONTEXT_ACP.

8.2.2 La machine MACHINE_ACP

Nous présentons dans cette section le contenu de la machine MACHINE_ACP qui formalise le comportement du système. Cette machine B événementielle contient deux variables d'état : C_ACP (*Current Access Control Policy*) qui contient la politique de sécurité courante du système. La deuxième variable LOG enregistre tous les privilèges utilisés par les utilisateurs du système, cette variable nous permet d'exprimer des propriétés de sûreté du système. Le type des deux variables est comme suit :

$$\texttt{C\_ACP} \in \texttt{ACP}$$
$$\texttt{LOG} \in (\texttt{USER} \times \texttt{PERMISSION}) \leftrightarrow \texttt{ACP}$$

Les variables d'état du système évoluent quand les événements se déclenchent. L'utilisation d'une permission par un utilisateur est modélisé par un événement ACCESS :

EVENT ACCESS
ANY
user, perm
WHERE
grd1 : user ∈ USER
grd2 : perm ∈ PERMISSION
grd3 : perm ∈ ACP_PERMISSION(C_ACP)
THEN
act1 : LOG := LOG ∪ {(user ↦ perm) ↦ C_ACP}
END

Le changement de la politique de sécurité courante par une action administrative correspond à un changement de la variable C_ACP par un événement ADMIN qui sera présenté dans la sous-section suivante.

Lorsqu'un utilisateur user veut utiliser une permission perm, il faut vérifier que la politique de sécurité l'y autorise avant de rajouter au journal LOG le couple $(user \mapsto perm)$. Il faut noter que la politique de sécurité en vigueur au moment du déclenchement de l'événement est également rajoutée dans la variable LOG avec le couple $(user \mapsto perm)$. Ceci nous permet d'exprimer des propriétés de sûreté telle que :

$$\forall\, \texttt{user}, \texttt{perm}, \texttt{acp} \cdot (\texttt{user} \mapsto \texttt{perm}) \mapsto \texttt{acp} \in \texttt{LOG} \Rightarrow (\texttt{user} \mapsto \texttt{perm}) \in \texttt{ACP\_PERMISSION}(\texttt{acp})$$

Cet invariant exprime le fait qu'un utilisateur qui a bénéficié d'un privilège avait l'autorisation correspondante dans la politique de sécurité qui était en vigueur au moment où il a utilisé ce privilège. Abrial [9] a modélisé en B événementiel une politique de contrôle d'accès de personnes (P) dans un ensemble de bâtiments (BAT), une constante $\texttt{AUTH} \in \texttt{P} \leftrightarrow \texttt{BAT}$ contient les droits d'accès des personnes aux bâtiments et une variable de type fonction $\texttt{SIT} \in \texttt{P} \rightarrow \texttt{BAT}$ indique la position de chaque personne à un instant donné. Un invariant $\texttt{SIT} \subseteq \texttt{AUTH}$ garantissait que les accès aux bâtiments se font en accord avec la politique de sécurité. Mais cette façon de modéliser pose problème dans le cas où la politique de sécurité contenue dans AUTH change, en effet une personne peut se retrouver dans un bâtiment alors qu'elle n'est plus censée y être et la propriété $\texttt{SIT} \subseteq \texttt{AUTH}$ n'est plus vérifiée. Pour éviter ce problème nous avons utilisé un invariant plus faible en ajoutant la politique de sécurité courante dans la variable LOG mais un invariant qui reste également vérifié quand la politique de sécurité est susceptible de changer.

8.2.2.1 La formalisation de l'administration

Nous présentons dans cette section la formalisation de l'administration de manière générale avant de l'adapter plus tard aux différents modèles d'administration. Pour formaliser l'administration, nous rajoutons dans le modèle B événementiel de la politique de sécurité de la section précédente des constantes, des variables et des événements additionnels. Nous avons présenté dans le chapitre un résumé des différentes façons possibles pour administrer et gérer les privilèges des utilisateurs d'un système. Pour trouver la manière la plus adéquate de formaliser l'administration de la politique de sécurité nous avons étudié les différents modèles d'administration de politique de sécurité. Considérons par exemple cas du modèle RRA97 présenté dans la chapitre 7.3.2.1.3 : pour savoir si un utilisateur *user* qui est affecté à un rôle administratif *AdR* a le droit de mettre le rôle *r1* comme sous-rôle de *r2* il faut vérifier qu'il existe un intervalle d'autorité *IA* avec :

1. $\{r1, r2\} \subseteq IA$
2. La relation *can_modify* permet le changement : $can\_modify(AdR, IA)$.
3. L'intervalle d'autorité *IA* est encapsulé.

Il apparaît que la décision de permettre ou pas la modification de la politique de sécurité par l'utilisateur *user* dépend de la politique d'administration (ici la relation *can_modify*) de la politique de sécurité courante (ici la hiérarchie des rôle pour déterminer si l'intervalle d'autorité *IA* est encapsulé). Pour formaliser en B événementiel les politiques d'administration, nous avons introduit un type `AACP` qui contient les politiques d'administration. De manière générale, pour formaliser le fait qu'un utilisateur a le droit de changer la politique de sécurité d'une manière donnée, nous introduisons dans le contexte du modèle B événementiel une constante :

$$\texttt{ACP\_ADMIN} \in (\texttt{AACP} \times \texttt{ACP} \times \texttt{USER}) \rightarrow \mathbb{P}(\texttt{ACP})$$

Cette constante donne pour chaque triplet formé par la politique d'administration, la politique de sécurité courante et un utilisateur toutes les politiques de sécurité par lesquelles cet utilisateur peut remplacer la politique courante. Par exemple, un utilisateur `user`, une politique d'administration `aacp` peut remplacer la politique courante qui a la valeur `acp1` par une politique `acp2` si on a $\texttt{acp2} \in \texttt{ACP\_ADMIN}(\texttt{aacp} \mapsto \texttt{acp1} \mapsto \texttt{user})$.

De même que pour la variable `C_ACP` qui contient la politique de sécurité courante, nous introduisons une variable `C_AACP` qui contient la politique d'administration en cours. Il faut noter que de manière générale, la politique d'administration ne change que rarement dans la vie d'un système de contrôle d'accès, de rares changements sont effectués par un administrateur en chef.

$$\texttt{C\_AACP} \in \texttt{AACP}$$

Un événement `admin` permettant de modifier la politique de sécurité est introduit, lorsqu'un utilisateur `user` veut remplacer la politique de sécurité courante par une nouvelle politique `new_ACP` il faut vérifier que la politique d'administration le permet :

```
EVENT ADMIN
    ANY
     user, new_ACP
    WHERE
     grd1 : user ∈ USER
     grd2 : new_ACP ∈ ACP
     grd3 : new_ACP ∈ ACP_ADMIN(C_AACP ↦ C_ACP ↦ user)
    THEN
     act1 : C_ACP := new_ACP
    END
```

Certains systèmes ne permettent pas de changer de politique d'administration tandis que d'autres le permettent mais uniquement par un super-utilisateur qu'on appelle aussi parfois l'officier de sécurité en chef. Dans ce cas précis, il faut d'abord introduire la notion de super-utilisateur pour chaque type de politique d'administration (celui-ci s'appellera root pour les systèmes LINUX) et rajouter un événement qui lui permet de changer de politique d'administration :

```
CONSTANT
      SUPER_USER ∈ USER
```

Avec l'événement :

```
EVENT SUPER_ADMIN
    ANY
     user, new_AACP
    WHERE
     grd1 : user = SUPER_USER
     grd2 : new_AACP ∈ AACP
    THEN
     act1 : C_AACP := new_AACP
    END
```

Un système de contrôle d'accès formalisé en B événementiel est donc régi par deux paramètres ACP_PERMISSION et ACP_ADMIN. Nous avons voulu dans ce premier modèle B avoir une formalisation qui soit aussi générale que possible. Nous raffinerons ce modèle B événementiel pour l'adapter à chaque modèle de contrôle d'accès. Nous présentons dans la suite de cette section comme exemple une formalisation en B événementiel du modèle RBAC ainsi que son modèle d'administration ARBAC97. Nous présentons également comme exemple d'un modèle de contrôle d'accès discrétionnaire la formalisation du contrôle d'accès aux fichiers sous LINUX POSIX. Nous verrons ensuite à la fin de ce chapitre comment traiter certains problème de cohérence des paramètres ACP_PERMISSION et ACP_ADMIN.

8.3 RBAC avec le B événementiel

La machine MACHINE_ACP peut être raffinée de différentes manières pour formaliser les différents modèles de contrôle d'accès, nous présentons dans cette section la formalisation du modèle RBAC avec ses différents niveaux. Nous introduisons de nouveaux types pour les politiques de sécurité de types RBAC : Core RBAC, RBAC hiérarchique et le modèle RBAC avec contraintes.

Les types sont habituellement déclarés en B événementiel dans la clause SET comme des *ensembles porteurs*, mais une limitation syntaxique nous empêche de déclarer qu'un ensemble porteur est contenu dans un autre ensemble porteur. Pour contourner cette limitation, nous avons déclaré le type de base ACP comme un ensemble porteur et les autres types de politiques de contrôle d'accès comme des constantes dans la clause CONSTANTS. Nous avons défini un contexte B événementiel CONTEXT_RBAC qui étend le contexte CONTEXT_ACP en rajoutant trois types : CoreRBAC, HierRBAC et ConstRBAC pour les modèles Core RBAC, RBAC hiérarchique et le modèle RBAC avec contraintes respectivement.

SET
ACP
CONSTANT
CoreRBAC
HierRBAC
ConstRBAC
AXIOMS
$\texttt{CoreRBAC} \subseteq \texttt{ACP}$
$\texttt{HierRBAC} \subseteq \texttt{ACP}$
$\texttt{ConstRBAC} \subseteq \texttt{ACP}$

Pour définir le paramètre ACP_PERMISSION pour chaque type de politique de sécurité, nous avons besoin d'introduire les notions propres à RBAC telles que la notion de rôles ainsi que les différentes relations liant les entités du modèle RBAC : UA, PA, RH. Il faut noter que les relations UA et PA sont définies pour tous les types de modèles RBAC tandis que la relation RH n'est pas définie pour les modèles de type CoreRBAC.

SET
ROLE
CONSTANT
UA
PA
RH
AXIOMS
$\texttt{UA} \in (\texttt{CoreRBAC} \cup \texttt{HierRBAC} \cup \texttt{ConstRBAC}) \rightarrow (\texttt{USER} \leftrightarrow \texttt{ROLE})$
$\texttt{PA} \in (\texttt{CoreRBAC} \cup \texttt{HierRBAC} \cup \texttt{ConstRBAC}) \rightarrow (\texttt{ROLE} \leftrightarrow \texttt{PERMISSION})$
$\texttt{RH} \in (\texttt{HierRBAC} \cup \texttt{ConstRBAC}) \rightarrow (\texttt{ROLE} \leftrightarrow \texttt{ROLE})$

Nous pouvons maintenant définir le paramètre ACP_PERMISSION. Pour les politiques de type CoreRBAC, une autorisation est délivrée à l'utilisateur si ce dernier joue un rôle qui lui permet d'avoir la permission qu'il demande :

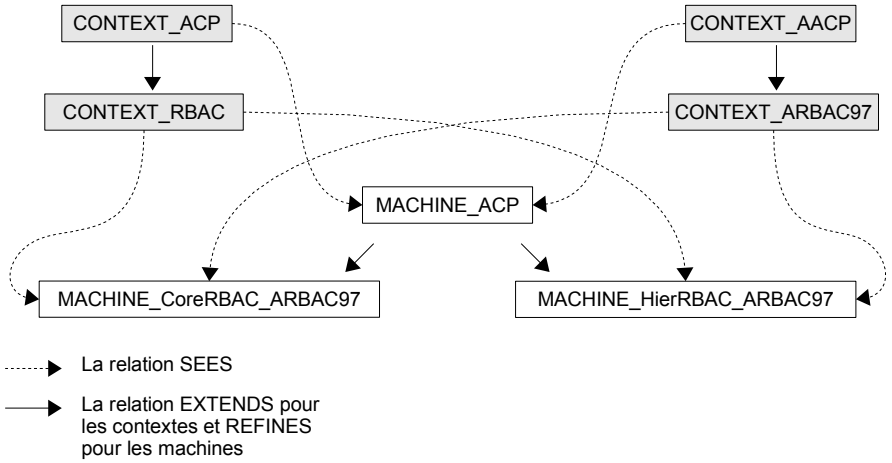


FIG. 8.1 – Un exemple de machines et les contextes pour les modèles RBAC et ARBAC97.

AXIOMS

$$\forall acp \cdot acp \in \texttt{CoreRBAC} \Rightarrow \texttt{ACP\_PERMISSION}(acp) = \texttt{UA}(acp); \texttt{PA}(acp)$$

Pour les politiques de type `HierRBAC`, une autorisation est délivrée à l'utilisateur si ce dernier joue un rôle qui domine hiérarchiquement un rôle qui lui permet d'avoir la permission qu'il demande :

AXIOMS

$$\forall acp \cdot acp \in \texttt{HierRBAC}$$
$$\Rightarrow$$
$$\texttt{ACP\_PERMISSION}(acp) = \texttt{UA}(acp); \texttt{RH}(acp); \texttt{PA}(acp)$$

La machine `MACHINE_ACP` est raffinée en une machine `MACHINE_CoreRBAC`, `MACHINE_HierRBAC` ou `MACHINE_ConstRBAC` selon le type de la politique de sécurité voulue. On garde dans ce raffinement les événements `ACCESS` et `ADMIN` ainsi que les variables `C_ACP` et `LOG`. Les changements principaux consistent à restreindre le type de la variable `C_ACP` qui était $\texttt{C\_ACP} \in \texttt{ACP}$ au type du modèle qu'on veut formaliser, par exemple, $\texttt{C\_ACP} \in \texttt{CoreRBAC}$. Il faut également charger le contexte `CONTEXT_RBAC` dans la clause `SEES` de la machine. Nous présentons dans la table 8.3, comme exemple, la machine `MACHINE_CoreRBAC`.

La figure 8.1 montre un exemple des relations entre les contextes et les machines pour arriver à une formalisation d'un système de contrôle d'accès régi par une politique CoreRBAC avec une politique d'administration ARBAC97 ainsi qu'un système de contrôle d'accès régi par une politique HierRBAC avec une politique d'administration ARBAC97. Les contextes `CONTEXT_RBAC`

MACHINE
Machine_CoreRBAC
REFINES
Machine_ACP
SEES
Context_RBAC
VARIABLES
C_ACP
C_LOG
C_AACP
INVARIANTS
inv1 : C_ACP ∈ CoreRBAC
inv2 : C_LOG ⊆ USER × PERMISSION × CoreRBAC
inv3 : ∀ user, perm, acp · (user ↦ perm ↦ acp) ∈ C_LOG
⇒
(user ↦ perm) ∈ ACP_PERMISSION(acp)
inv4 : C_AACP ∈ AACP
EVENTS
EVENT INITIALISATION
BEGIN
act1 : C_LOG := ∅
act2 : C_ACP :∈ CoreRBAC
act3 : C_AACP :∈ AACP
END

EVENT ACCESS
REFINES ACCESS
ANY
user, perm
WHERE
grd1 : user ∈ USER
grd2 : perm ∈ PERMISSION
grd3 : perm ∈ ACP_PERMISSION(C_ACP)
THEN
act1 : C_LOG := C_LOG ∪ {(user ↦ perm) ↦ C_ACP}
END

EVENT ADMIN
REFINES ADMIN
ANY
user, new_ACP
WHERE
grd1 : user ∈ USER
grd2 : new_ACP ∈ ACP
grd3 : new_ACP ∈ ACP_ADMIN(C_AACP ↦ C_ACP ↦ user)
THEN
act1 : C_ACP := new_ACP
END

TAB. 8.1 – La machine MACHINE_CoreRBAC

et CONTEXT_ARBAC97 étendent respectivement les contextes CONTEXT_ACP et CONTEXT_AACP pour définir les paramètres ACP_PERMISSION et ACP_ADMIN.

Les machines MACHINE_CoreRBAC_ARBAC97 et MACHINE_HierRBAC_ARBAC97 raffinent la machine MACHINE_RBAC et "voient" les contextes CONTEXT_RBAC et CONTEXT_ARBAC97.

8.3.1 La séparation des pouvoirs

Nous avons vu dans la section précédente comment est défini le paramètre ACP_PERMISSION pour les politiques de type CoreRBAC et HierRBAC. Le paramètre ACP_PERMISSION pour les politiques de type ConstRBAC nécessite l'introduction de quelques notions supplémentaires propres à la séparation des pouvoirs notamment la notion de SESSION. Ces notions sont introduites au niveau du contexte des politiques RBAC contenu dans CONTEXT_RBAC. Deux constantes SSoD et DSoD contenant les rôles en conflit pour la séparation statique et dynamique respectivement :

$$\begin{array}{l} \mathtt{SSoD} \in \mathtt{ConstRBAC} \rightarrow \mathbb{P}(\mathbb{P}(\mathtt{ROLE})) \\ \mathtt{DSoD} \in \mathtt{ConstRBAC} \rightarrow \mathbb{P}(\mathbb{P}(\mathtt{ROLE})) \end{array}$$

SSoD et DSoD sont définies à l'aide de l'ensemble des parties de l'ensemble des parties de l'ensemble de rôles. Les ensembles de rôles incompatibles sont habituellement formalisés à l'aide d'une relation entre rôles (ce qui donnerait dans notre cas $\mathtt{SSoD} \in \mathtt{Const\_RBAC} \rightarrow (\mathtt{ROLE} \leftrightarrow \mathtt{ROLE})$. L'inconvénient de cette formalisation est qu'elle ne permet pas de formaliser le cas où le nombre de rôles en conflit entre eux est supérieur à deux. Pa exemple il serait impossible d'exprimer le fait qu'un sujet puisse être affecté aux rôles r1 et r2 au même temps mais pas aux rôles r1, r2 et r3 au même temps.

Vérifier qu'une politique de sécurité donnée acp satisfait la propriété de séparation des pouvoirs statiques revient à prouver la correction de la formule :

$$\forall \mathtt{user} \cdot \mathtt{user} \in \mathtt{USER} \Rightarrow \mathtt{RH}(\mathtt{acp})[\mathtt{UA}(\mathtt{acp})[\{\mathtt{user}\}]] \notin \mathtt{SSoD}(\mathtt{acp})$$

Le modèle RBAC avec contraintes utilise la notion de *session* pour mettre en œuvre la séparation dynamique des pouvoirs. Nous introduisons donc un nouvel ensemble porteur SESSION dans la clause SET de notre contexte CONTEXT_RBAC. Comme discuté dans le chapitre 7, une session est caractérisée par un ensemble de rôles (voir figure 7.9). Chaque utilisateur est, à un instant donné, dans une seule session. Nous introduisons une relation SR pour formaliser la relation entre sessions et rôles, et une fonction US indiquant dans quelle session se trouve chaque utilisateur :

SET
SESSION
CONSTANT
SR
US
AXIOMS
$\mathtt{SR} \in (\mathtt{ConstRBAC}) \rightarrow (\mathtt{SESSION} \rightarrow \mathbb{P}(\mathtt{ROLE}))$
$\mathtt{US} \in (\mathtt{ConstRBAC}) \rightarrow (\mathtt{USER} \rightarrow \mathtt{SESSION})$

Nous pouvons à présent définir le paramètre ACP_PERMISSION pour les politiques de type ConstRBAC. Un utilisateur obtient une permission uniquement si la session dans laquelle il se trouve le lui permet :

AXIOMS

$\forall acp \cdot acp \in ConstRBAC$

$\Rightarrow$

$ACP\_PERMISSION(acp) = US(acp); SR(acp); RH(acp); PA(acp)$

La machine MACHINE_ConstRBAC raffine la machine MACHINE_ACP en gardant les variables C_ACP et C_LOG et en renforçant le type de la variable C_ACP qui était $C\_ACP \in ACP$ au type ConstRBAC. Les événements ACCESS et ADMIN qui ne changent pas puisqu'ils utilisent le paramètre ACP_PERMISSION qu'on vient de définir dans le contexte CONTEXT_RBAC pour les politiques de type ConstRBAC. Un événement Change_SESSION propre aux politiques de type RBAC avec contraintes est ajouté, celui-ci permet à un utilisateur de changer de session.

Un changement de session pour un utilisateur revient à changer la politique de sécurité courante avec une autre politique qui a exactement les même attributs sauf pour la valeur de la fonction US qui prend en compte le changement de session voulu. Nous introduisons une fonction ACP_SESSION qui donne pour une politique de sécurité donnée la nouvelle politique obtenue en changeant uniquement la session d'un utilisateur donné :

CONSTANT

ACP_SESSION

AXIOMS

$ACP\_SESSION \in (ConstRBAC \times USER \times SESSION) \rightarrow ConstRBAC \wedge$

$\forall\ acp1, acp2, user, session.$

$acp2 = ACP\_SESSION(acp1 \mapsto user \mapsto session)$

$\Rightarrow$

$(RH(acp1) = RH(acp2) \wedge$

$UA(acp1) = UA(acp2) \wedge$

$PA(acp1) = PA(acp2) \wedge$

$SR(acp1) = SR(acp2) \wedge$

$SSoD(acp1) = SSoD(acp2) \wedge$

$DSoD(acp1) = DSoD(acp2) \wedge$

$US(acp2) = US(acp2) \Leftarrow \{user \mapsto session\}$

Nous pouvons maintenant présenter l'événement change_SESSION, lorsqu'un utilisateur veut changer de session, il faut s'assurer que les rôles qu'ils jouent dans cette session sont parmi les rôles que la relation UA lui autorise de jouer. Avant d'autoriser le changement de session il faut également s'assurer que ce changement de session est conforme à la séparation des pouvoirs dynamique, et donc que les rôles auxquels est affecté l'utilisateur par le biais de sa nouvelle session ne font pas partie de l'ensemble des ensembles de rôles incompatibles DSoD.

EVENT change_SESSION
ANY
user, session, new_ACP
WHERE
grd1 : user ∈ USER
grd2 : session ∈ SESSION
grd3 : new_ACP ∈ ACP
grd4 : new_ACP = ACP_SESSION(C_ACP ↦ user ↦ session)
grd5 : SR(new_ACP)(session) ⊆ UA(new_ACP)[{user}]
grd6 : RH(new_ACP)[SR(new_ACP)(session)] ∉ DSoD(new_ACP)
THEN
act1 : C_ACP := new_ACP
END

Il est ainsi possible de prouver comme propriété d'invariance, le fait que chaque utilisateur est dans une session qui respecte la séparation dynamique des pouvoirs :

INVARIANT
∀user.user ∈ USER
⇒
RH(C_ACP)[SR(C_ACP)((US(C_ACP))(user))] ∉ DSoD(C_ACP)

8.4 ARBAC97 avec le B événementiel

Nous présentons dans cette section la formalisation en B événementiel du modèle ARBAC97. Ce modèle d'administration pour le modèle RBAC a été déjà présenté dans le chapitre 7.3.2.1. Il se compose de trois modèles URA97, PRA97 et le modèle RRA97 qui administre les relations UA, PA et RH respectivement. Nous étendrons le contexte CONTEXT_AACP avec des notions supplémentaires pour formaliser ARBAC97 avec le B événementiel. Nous appellerons le contexte ainsi obtenu CONTEXT_ARBAC97. La première notions introduite est un nouveau sous-type pour les politiques administratives de type ARBAC97 :

CONSTANT
ARBAC97
AXIOMS
ARBAC97 ⊆ AACP

8.4.1 Les modèles URA97, PRA97 et RRA97

Les rôles administratifs du modèle URA97 étant distincts des rôles *normaux* du modèle RBAC, il nous faut introduire un nouvel ensemble porteur A_ROLE contenant les rôles administratifs ainsi qu'une relation A_UA liant les utilisateurs aux rôles administratifs qu'ils jouent. La relation can_modify du modèle URA97 est quand à elle formalisée comme suit :

SET
  A_ROLE
CONSTANT
  A_UA
  can_assign
AXIOMS
  A_UA $\in$ ARBAC97 $\rightarrow$ (USER $\leftrightarrow$ A_ROLE)
  can_assign $\in$ ARBAC97 $\rightarrow$ (A_ROLE $\times$ $\mathbb{P}$(ROLE) $\times$ $\mathbb{P}$(ROLE) $\times$ $\mathbb{P}$(ROLE))

Si, par exemple, on a pour un rôle administratif ar1, un ensemble de rôles r1, r2, r3, r4, r5 et une politique d'administration arbac97 :

$$(\texttt{ar1}, \{\texttt{r1}, \texttt{r2}\}, \{\texttt{r3}\}, \{\texttt{r4}, \texttt{r5}\}) \in \texttt{can\_assign}(\texttt{arbac97})$$

Alors un administrateur affecté au rôle administratif ar1 peut affecter un utilisateur qui joue l'un des rôles r1 ou r2 et qui ne joue pas le rôle r3 à l'un des rôles r4 ou r5.

Nous introduisons de manière semblable le reste des relations propres aux modèles URA97, PRA97 et RRA97 déjà introduites dans le chapitre 7.3.2.1 :

CONSTANT
  can_revoke
  can_assignP
  can_revokeP
  can_modify
AXIOMS
  can_revoke $\in$ ARBAC97 $\rightarrow$ (A_ROLE $\times$ $\mathbb{P}$(ROLE))
  can_assignP $\in$ ARBAC97 $\rightarrow$ (A_ROLE $\times$ $\mathbb{P}$(ROLE) $\times$ $\mathbb{P}$(ROLE) $\times$ $\mathbb{P}$(ROLE))
  can_revokeP $\in$ ARBAC97 $\rightarrow$ (A_ROLE $\times$ $\mathbb{P}$(ROLE))
  can_modify $\in$ ARBAC97 $\rightarrow$ (A_ROLE $\times$ $\mathbb{P}$(ROLE))

Nous pouvons à présent définir le paramètre ACP_ADMIN pour les politiques administratives de type ARBAC97. L'axiome suivant montre comment la relation UA d'une politique rbac de type HierRBAC peut être administrée avec une politique arbac97 donnée de type ARBAC97. La politique rbac est remplacée par une politique new_RBAC dans laquelle l'utilisateur user joue le rôle role si l'administrateur a_user a le droit de la faire selon la politique d'administration arbac97 et la relation can_assign qui lui est associée.

AXIOMS

$$
\begin{array}{l}
\forall\, \texttt{arbac97}, \texttt{rbac}, \texttt{new\_rbac}, \texttt{a\_user}, \texttt{user}, \texttt{role}. \\
\texttt{arbac97} \in \texttt{ARBAC97} \land \texttt{rbac} \in \texttt{HierRBAC} \land \texttt{new\_rbac} \in \texttt{HierRBAC} \land \\
\texttt{a\_user} \in \texttt{USER} \land \texttt{user} \in \texttt{USER} \land \texttt{role} \in \texttt{ROLE} \land \\
\quad (\exists\, \texttt{a\_role}, \texttt{R}, \texttt{R1}, \texttt{R2}. \\
\quad \texttt{a\_role} \in \texttt{A\_ROLE} \land (\texttt{a\_user} \mapsto \texttt{a\_role}) \in \texttt{A\_UA} \land \\
\quad \texttt{R} \subseteq \texttt{ROLE} \land \texttt{role} \in \texttt{R} \land \\
\quad \texttt{R1} \subseteq \texttt{ROLE} \land \texttt{R2} \subseteq \texttt{ROLE} \land \\
\quad \texttt{R1} \subseteq \texttt{UA}(\texttt{rbac})[\{\texttt{user}\}] \land \texttt{R2} \cap \texttt{UA}(\texttt{rbac})[\{\texttt{user}\}] = \varnothing \land \\
\quad (\texttt{a\_role} \mapsto \texttt{R1}, \texttt{R2}, \texttt{R}) \in \texttt{can\_assign}(\texttt{arbac97})) \land \\
\texttt{UA}(\texttt{new\_rbac}) = \texttt{UA}(\texttt{rbac}) \lhd\!\!+ \{\texttt{user} \mapsto \texttt{role}\} \land \\
\texttt{RH}(\texttt{new\_rbac}) = \texttt{RH}(\texttt{rbac}) \land \\
\texttt{PA}(\texttt{new\_rbac}) = \texttt{PA}(\texttt{rbac}) \\
\Rightarrow \\
\texttt{new\_rbac} \in \texttt{ACP\_ADMIN}(\texttt{arbac97} \mapsto \texttt{rbac} \mapsto \texttt{user})
\end{array}
$$

Des axiomes similaires sont rajoutés pour montrer comment sont administrés les changements sur les relations PA en utilisant les relations can_assignP et can_revokeP ainsi que la relation RH administrée à l'aide de la relation can_modify. Nous donnons ici l'axiome correspondant aux changements sur la relation RH car nous en aurons besoin plus tard dans cette thèse.

AXIOMS

$$
\begin{array}{l}
\forall\, \texttt{arbac97}, \texttt{rbac}, \texttt{new\_rbac}, \texttt{a\_user}, \texttt{role1}, \texttt{role2}. \\
\texttt{arbac97} \in \texttt{ARBAC97} \land \texttt{rbac} \in \texttt{HierRBAC} \land \texttt{new\_rbac} \in \texttt{HierRBAC} \land \\
\texttt{a\_user} \in \texttt{USER} \land \texttt{role1} \in \texttt{ROLE} \land \texttt{role2} \in \texttt{ROLE} \land \\
\quad (\exists\, \texttt{a\_role}, \texttt{R}. \\
\quad \texttt{a\_role} \in \texttt{A\_ROLE} \land (\texttt{a\_user} \mapsto \texttt{a\_role}) \in \texttt{A\_UA} \land \\
\quad \texttt{R} \subseteq \texttt{ROLE} \land \texttt{role1} \in \texttt{R} \land \texttt{role2} \in \texttt{R} \land \\
\quad (\texttt{a\_role} \mapsto \texttt{R}) \in \texttt{can\_modify}(\texttt{arbac97})) \land \\
\texttt{RH}(\texttt{new\_rbac}) = \texttt{RH}(\texttt{rbac}) \lhd\!\!+ \{\texttt{role1} \mapsto \texttt{role2}\} \land \\
\texttt{UA}(\texttt{new\_rbac}) = \texttt{UA}(\texttt{rbac}) \land \\
\texttt{PA}(\texttt{new\_rbac}) = \texttt{PA}(\texttt{rbac}) \\
\Rightarrow \\
\texttt{new\_rbac} \in \texttt{ACP\_ADMIN}(\texttt{arbac97} \mapsto \texttt{rbac} \mapsto \texttt{user})
\end{array}
$$

Après avoir introduit l'ensemble des axiomes permettant de définir le paramètre ACP_ADMIN, il faut préciser que cette constante ainsi définie est la plus petite possible, afin de limiter les modifications de politiques de sécurité à celles prévues dans ces axiomes.

8.5 Linux POSIX avec le B événementiel

Nous étendrons dans cette section les contextes CONTEXT_ACP et CONTEXT_AACP pour définir les paramètres ACP_PERMISSION et ACP_ADMIN comme cela a été fait pour RBAC et ARBAC97, nous appelons ce nouveau contexte CONTEXT_LINUX. Linux POSIX est basé sur un système de contrôle d'accès discrétionnaire, nous

introduisons un nouveau type de politique de contrôle d'accès LINUX. Dans le cas de Linux POSIX, les permissions sont des accès aux fichiers, les permission que nous avons définies de manière abstraite dans CONTEXT_ACP seront raffinées en un couple constitué d'une opération et d'un fichier. Une opération peut être une écriture, une lecture ou une exécution

SET
FILE
OPERATION $= \{\texttt{read}, \texttt{write}, \texttt{execute}\}$
CONSTANT
LINUX
root
AXIOMS
LINUX $\subseteq$ ACP
root $\in$ USER

Nous utilisons une fonction bijective totale pour lier les permissions abstraites définies dans le contexte CONTEXT_ACP aux permissions concrètes de Linux constituées d'une opération et d'un fichier. Nous appelons cette fonction OF2P :

CONSTANT
OF2P
AXIOMS
OF2P $\in$ (OPERATION $\times$ FILE) $\rightarrowtail\!\!\!\!\!\rightarrow$ PERMISSION

La notions de groupe qu'on retrouve souvent dans les modèles discrétionnaires est présente dans Linux. Les utilisateurs peuvent appartenir, dans Linux, à un ou plusieurs groupes. Nous rajoutons un type GROUP ainsi qu'une fonction U_GROUP pour l'affectation des sujets aux groupes.

SET
GROUP
CONSTANT
U_GROUP
AXIOMS
U_GROUP $\in$ LINUX $\rightarrow$ (USER $\rightarrow \mathbb{P}$(GROUP))

Les fichiers appartiennent à des utilisateurs et à des groupes d'utilisateurs. La fonction File_U indique l'utilisateur à qui appartient le fichier, une fonction File_G indique, quand à elle, le groupe auquel est affecté le fichier.

CONSTANT
File_U
File_G
AXIOMS
File_U $\in$ LINUX $\rightarrow$ (FILE $\nrightarrow$ USER)
File_G $\in$ LINUX $\rightarrow$ (FILE $\nrightarrow$ GROUP)

Les privilèges sont accordés dans Linux selon que l'utilisateur soit le propriétaire du fichier ou s'il fait partie du groupe auquel appartient le fichier. Trois fonctions définissent, pour chaque fichier, les privilèges associés respectivement au propriétaire du fichier (`File_U_P`), aux membres du groupe auquel est associé le fichier (`File_G_P`) ainsi qu'aux reste des utilisateurs du système (`File_P_P`).

$$\begin{array}{l}\texttt{File\_P\_P} \in \texttt{LINUX} \rightarrow (\texttt{FILE} \nrightarrow \mathbb{P}(\texttt{OPERATION})) \\ \texttt{File\_U\_P} \in \texttt{LINUX} \rightarrow (\texttt{FILE} \nrightarrow \mathbb{P}(\texttt{OPERATION})) \\ \texttt{File\_G\_P} \in \texttt{LINUX} \rightarrow (\texttt{FILE} \nrightarrow \mathbb{P}(\texttt{OPERATION}))\end{array}$$

Il faut noter que pour une politique de sécurité Linux donnée `linux`, les domaines des fonctions `File_U(linux)`, `File_G(linux)`, `File_P_P(linux)`, `File_U_P(linux)`, et `File_P_P(linux)` doivent être égaux. Le contenu de ces domaines correspond au système de fichiers présent dans le système.

Nous pouvons à présent définir le paramètre `ACP_PERMISSION` pour les politiques LINUX, un utilisateur `user` peut effectuer une opération sur un fichier si :

- Il est le *root*.
- L'opération peut être effectuée par n'importe quel utilisateur du système.
- L'opération peut être effectuée par le propriétaire du fichier et `user` est le propriétaire du fichier.
- L'opération peut être effectuée par un membre du groupe auquel appartient le fichier et `user` est membre de ce groupe.

Le paramètre `ACP_PERMISSION` est, ainsi, défini comme suit :

$$\begin{array}{l}\forall\, \texttt{linux}, \texttt{user}, \texttt{operation}, \texttt{file} \cdot \\ \texttt{user} \mapsto \texttt{OF2P}(\texttt{operation} \mapsto \texttt{file}) \in \texttt{ACP\_PERMISSION}(\texttt{linux}) \\ \Leftrightarrow \\ \texttt{user} \in \texttt{USER} \wedge \texttt{linux} \in \texttt{LINUX} \wedge \\ \texttt{operation} \in \texttt{OPERATION} \wedge \texttt{file} \in \texttt{FILE} \wedge \\ (\\ \quad \texttt{user} = \texttt{root} \vee \\ \quad \texttt{operation} \in \texttt{File\_P\_P}(\texttt{linux})(\texttt{file}) \vee \\ \quad (\texttt{action} \in \texttt{File\_U\_P}(\texttt{linux})(\texttt{file}) \wedge \texttt{File\_U}(\texttt{linux})(\texttt{file}) = \texttt{user}) \vee \\ \quad (\\ \quad\quad \exists \texttt{grp} \cdot \texttt{grp} \in \texttt{U\_GROUP}(\texttt{linux})(\texttt{user}) \wedge \\ \quad\quad (\texttt{operation} \in \texttt{File\_G\_P}(\texttt{linux})(\texttt{file}) \wedge \\ \quad\quad \texttt{File\_G}(\texttt{linux})(\texttt{file}) = \texttt{grp}))) \\ \quad) \\ )\end{array}$$

L'administration de la politique de sécurité sous Linux se fait, à l'image des modèles discrétionnaires, principalement par les propriétaires des fichiers, ainsi que par le super utilisateur *root*. La politique d'administration que nous allons formaliser avec le paramètre `ACP_ADMIN` contient les règles d'administration que l'on retrouve dans Linux POSIX et dans une majeure partie des systèmes Linux, cependant, ce paramètre peut être étendu ou modifié pour correspondre

à un système Linux donné. L'administration sous Linux concerne la modification des différentes entités qui formalisent l'accès aux fichiers File_U, File_G, File_P_P, File_U_P, et File_P_P mais également l'affectation des utilisateurs aux groupes U_GROUP. Nous présentons ici les fragments de la politique d'administration sous Linux POSIX qui seront nécessaires à la présentation de nos études de cas dans la suite de ce document. Nous étendons le contexte CONTEXT_AACP pour obtenir CONTEXT_AdLINUX contenant la politique d'administration sous Linux POSIX.

Nous introduisons d'abord un nouveau type pour la politique d'administration sous Linux POSIX :

SET
AdLINUX
CONSTANT
AdLINUX ∈ AACP

De façon générale, le super utilisateur peut tout modifier dans le système, en utilisant notamment les commandes :

- *chown* : qui permet de changer le propriétaire d'un fichier et donc l'attribut File_U.
- *chmod* : qui permet de changer les privilèges sur un fichier et donc les attributs File_P_P, File_U_P et File_G_P.

Ceci se traduit dans la paramètre ACP_ADMIN avec l'axiome suivant où l'utilisateur root remplace la politique de sécurité linux courante par une autre politique new_linux sans contraintes :

AXIOMS
∀ adlinux, linux, new_linux
adlinux ∈ AdLINUX ∧ linux ∈ LINUX ∧ new_linux ∈ LINUX
⇒
new_linux ∈ ACP_ADMIN(adlinux ↦ linux ↦ root)

Les attributs File_U, File_G sont établis au moment de la création du fichier et ne peuvent être modifiés (sauf par le super utilisateur) :

- L'attribut File_U prend comme valeur le créateur du fichier.
- L'attribut File_G prend comme valeur un des groupes auxquels appartient l'utilisateur qui créa le fichier.
- Les attributs File_P_P, File_U_P et File_G_P reçoivent des valeurs initiales.

Ceci est reflété dans le paramètre ACP_ADMIN : si un utilisateur user veut rajouter un fichier file avec comme groupe propriétaire grp et comme privilèges les trois ensembles de privilèges p1, p2, p3 dans les attributs File_P_P, File_U_P et File_G_P.

AXIOMS

$\forall$ adlinux, linux, new_linux, file, user, grp, p1, p2, p3
adlinux $\in$ AdLINUX $\wedge$ linux $\in$ LINUX $\wedge$ new_linux $\in$ LINUX $\wedge$
file $\notin$ dom(FILE_U) $\wedge$ user $\in$ USER $\wedge$
grp $\notin$ GROUP $\wedge$ grp $\in$ U_GROUP(linux)(user) $\wedge$
p1 $\subseteq$ OPERATION $\wedge$ p2 $\subseteq$ OPERATION $\wedge$ p3 $\subseteq$ OPERATION $\wedge$
FILE_U(new_linux) = FILE_U(linux) $\lhd\!\!+$ {file $\mapsto$ user}
FILE_G(new_linux) = FILE_G(linux) $\lhd\!\!+$ {file $\mapsto$ grp}
FILE_P_P(new_linux) = FILE_P_P(linux) $\lhd\!\!+$ {file $\mapsto$ p1}
FILE_U_P(new_linux) = FILE_U_P(linux) $\lhd\!\!+$ {file $\mapsto$ p2}
FILE_G_P(new_linux) = FILE_G_P(linux) $\lhd\!\!+$ {file $\mapsto$ p3}
U_GROUP(new_linux) = U_GROUP(linux)
$\Rightarrow$
new_linux $\in$ ACP_ADMIN(adlinux $\mapsto$ linux $\mapsto$ user)

Les privilèges d'un fichier existant peuvent être modifiés par le propriétaire du fichier, ceci revient à rajouter l'axiome suivant où un utilisateur `user` propriétaire d'un fichier `file` lui attribut les les trois ensemble de privilèges `p1`, `p2`, `p3` dans les attributs `File_P_P`, `File_U_P` et `File_G_P`.

AXIOMS

$\forall$ adlinux, linux, new_linux, file, user, p1, p2, p3
adlinux $\in$ AdLINUX $\wedge$ linux $\in$ LINUX $\wedge$ new_linux $\in$ LINUX $\wedge$
file $\in$ dom(FILE_U) $\wedge$ user $\in$ USER $\wedge$ user = FILE_U(file) $\wedge$
p1 $\subseteq$ OPERATION $\wedge$ p2 $\subseteq$ OPERATION $\wedge$ p3 $\subseteq$ OPERATION $\wedge$
FILE_U(new_linux) = FILE_U(linux)
FILE_G(new_linux) = FILE_G(linux)
FILE_P_P(new_linux) = FILE_P_P(linux) $\lhd\!\!+$ {file $\mapsto$ p1}
FILE_U_P(new_linux) = FILE_U_P(linux) $\lhd\!\!+$ {file $\mapsto$ p2}
FILE_G_P(new_linux) = FILE_G_P(linux) $\lhd\!\!+$ {file $\mapsto$ p3}
U_GROUP(new_linux) = U_GROUP(linux)
$\Rightarrow$
new_linux $\in$ ACP_ADMIN(adlinux $\mapsto$ linux $\mapsto$ user)

L'appartenance des utilisateurs Linux aux groupes est contrôlée par le super utilisateur *root* mais Linux offre la possibilité de définir des administrateurs pour les groupes. L'utilisateur *root* peut déléguer l'affectation des utilisateurs à un groupe particulier à d'autres utilisateurs désignés comme administrateurs de ce groupe. Un administrateur peut par la suite affecter un utilisateur à un groupe à l'aide de la commande Linux *gpasswd*. Nous introduisons une constante `GROUP_AD` qui contient les administrateurs de chaque groupe :

GROUP_AD $\in$ AdLINUX $\rightarrow$ (GROUP $\nrightarrow$ $\mathbb{P}$(USER))

Dans le paramètre `ACP_ADMIN`, un administrateur `a_user` d'un groupe `grp` veut rajouter un utilisateur `user` à ce groupe, se traduit par l'axiome suivant :

AXIOMS

$\forall$ adlinux, linux, new_linux, a_user, user, grp .
adlinux $\in$ AdLINUX $\wedge$ linux $\in$ LINUX $\wedge$ new_linux $\in$ LINUX $\wedge$
a_user $\in$ USER $\wedge$ grp $\in$ GROUP $\wedge$
a_user $\in$ GROUP_AD(AdLinux)(grp) $\wedge$
user $\in$ dom(U_GROUP(linux)) $\wedge$
U_GROUP(new_linux) = U_GROUP(linux) $\lhd$ {user $\mapsto$ (U_GROUP(linux) $\cup$ {grp})}
FILE_U(new_linux) = FILE_U(linux) $\wedge$
FILE_G(new_linux) = FILE_G(linux) $\wedge$
FILE_P_P(new_linux) = FILE_P_P(linux) $\wedge$
FILE_U_P(new_linux) = FILE_U_P(linux) $\wedge$
FILE_G_P(new_linux) = FILE_G_P(linux)
$\Rightarrow$
new_linux $\in$ ACP_ADMIN(adlinux $\mapsto$ linux $\mapsto$ user)

Comme pour le paramètre ACP_ADMIN définit pour les politiques ARBAC97, après avoir introduit l'ensemble des axiomes permettant de définir le paramètre, il faut préciser que cette constante ainsi définie est la plus petite possible. Ceci afin de limiter les modifications de politiques de sécurité à celles prévues dans ces axiomes.

Il faut noter qu'il est possible d'utiliser les modèles B événementiel des politiques de contrôle d'accès pour vérifier certaines propriétés de cohérence sur ces modèles. Il est par exemple possible d'utiliser le model-checker *ProB* pour vérifier la propriété de séparation statique des pouvoirs dans le modèle RBAC ou encore vérifier que les intervalles d'autorités contenus dans la relation can_modify du modèle ARBAC97 soient encapsulés. Le but est de vérifier qu'une politique de sécurité est cohérente avant de l'implémenter.

8.6 Conclusion

Nous avons présenté dans ce chapitre notre modélisation des politiques de contrôle d'accès, l'aspect le plus important de notre modélisation est son aspect paramétrique : les politiques de contrôles d'accès sont définies à l'aide de deux paramètres ACP_PERMISSION et ACP_ADMIN qu'il est possible d'instancier pour un modèle de politique de sécurité donné. Nous utiliserons dans le chapitre suivant ces paramètres pour définir une méthode générale de vérification de l'implémentation d'une politique de sécurité dans un système utilisant un autre modèle de politique de sécurité. Nous avons vu comment nous avons modélisé le modèle RBAC, son modèle d'administration ARBAC97 et le modèle de contrôle d'accès de Linux POSIX. Il faut noter qu'une autre étude de cas concernant la modélisation du modèle OrBAC avec le B événementiel peut être trouvée dans notre publication [19].

Chapitre 9

L'implémentation par raffinement

9.1 Introduction

Après avoir présenté dans les chapitres précédents les différents modèles de contrôle d'accès ainsi que leur formalisation en B événementiel, nous abordons dans ce chapitre le problème de la vérification de l'implémentation d'un modèle de contrôle d'accès dans un autre système basé sur un autre modèle de contrôle d'accès. Notre travail est basé sur l'application de différentes méthodes de raffinements pour satisfaire à un certain nombre d'exigences que nous nous sommes fixés. D'abord notre méthode de vérification se doit d'être la plus générale possible en d'autre termes qu'elle soit facilement ré-applicable sur plusieurs études de cas. Ainsi nous donnons au début de ce chapitre la méthode générale et nous l'appliquerons ensuite sur une étude de cas qui est l'implémentation du modèle RBAC sur un système Linux. L'autre exigence que nous nous sommes imposés concerne la taille des modèles et la difficulté des preuves et leur automatisation ainsi que la réutilisation de preuves déjà faites pour un cas particulier dans d'autres études de cas. Nous utiliserons pour atteindre ces objectifs les différentes stratégies de raffinements et de composition des modèles déjà présentées dans le chapitre 3.

9.2 L'implémentation par raffinement de données

Le raffinement de données est un raffinement structurel permettant de changer des structures abstraites par des structures plus concrètes tout en conservant des propriétés de sûreté définies dans le modèle abstrait. Cette particularité du raffinement de données correspond au problème de l'implémentation où le modèle de contrôle d'accès que l'on veut implémenter constitue le modèle abstrait et le modèle dans lequel on veut l'implémenter sera le modèle concret.

Nous désignerons dans la suite de ce chapitre le type de la politique de contrôle d'accès que nous voulons implémenter par `T_A_ACP` (*pour abstract access control policy*) et le type de sa politique d'administration par `T_A_AACP`. La politique sur laquelle nous réalisons l'implémentation est quant à elle de

type `T_C_ACP` (*pour concrete access control policy*) et le type de sa politique d'administration par `T_C_AACP`.

Nous commençons par poser le problème de l'implémentation : un utilisateur `a_user` veut accéder à une ressource donnée, cet accès doit se faire conformément à la politique de type `T_A_ACP` mais en utilisant uniquement les mécanismes de la politique de sécurité de type `T_C_ACP`. Pour ce faire, il peut être nécessaire d'avoir une interface entre le nom d'utilisateur `a_user` et un nom d'utilisateur `c_user` correspondant dans la politique de type `T_C_ACP`. Par exemple, si l'on veut implémenter un serveur d'une application où le contrôle d'accès se fait selon le modèle RBAC sous un système Linux, le concepteur peut vouloir mettre en place des noms d'utilisateur propres à son serveur comme le montre la figure 9.1.

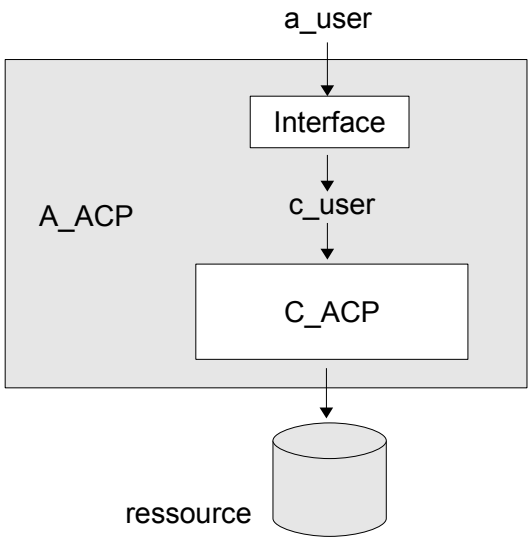


FIG. 9.1 – L'implémentation de `T_A_ACP` par `T_C_ACP`

Il peut s'avérer nécessaire de configurer les attributs de la politiques de sécurité `T_C_ACP` pour satisfaire à l'implémentation. Le concepteur peut avoir une idée plus ou moins précise sur la façon de configurer la politique `T_C_ACP` pour satisfaire les exigences de l'implémentation. Il peut y avoir plusieurs façons de configurer `T_C_ACP` pour implémenter `A_ACP`. Par exemple, comme nous le verrons plus loin en détail dans ce chapitre, il y a plusieurs options possibles pour implémenter le modèle RBAC dans Linux. On peut soit implémenter les rôles RBAC en comptes utilisateurs Linux soit les implémenter en groupes de Linux. Une configuration peut initialement être incomplète et complétée au fur et à mesure que les preuves interactives sont déchargées dans l'outil RODIN. Ce processus de preuves interactives est important dans la mesure où il permet de compléter une configuration initiale qui peut être incomplète.

Afin de vérifier l'implémentation, il faut au préalable formaliser les deux politiques de contrôle d'accès avec leur politique d'administration respective en

suivant la méthodologie exposée dans le chapitre 8. À l'issue de cette formalisation nous obtenons :

- `CONTEXT_A_ACP` : le contexte de la politique `T_A_ACP`.
- `CONTEXT_A_AACP` : le contexte de la politique d'administration de type `T_A_AACP`.
- `MACHINE_A_ACP` : le machine de la politique `T_A_ACP` avec la politique d'administration `T_A_AACP` :. Cette machine contient les variables similaires aux machines `MACHINE_ACP` du chapitre 8 que nous renommons comme suit :
  1. `A_ACP` de type `T_A_ACP` qui contient la politique de sécurité courante.
  2. `A_LOG` avec $\texttt{A\_LOG} \in (\texttt{USER} \times \texttt{PERMISSION}) \leftrightarrow \texttt{T\_A\_ACP}$ contient l'ensemble des accès déjà effectués dans le système.
- `CONTEXT_C_ACP` : le contexte de la politique de type `T_C_ACP`.
- `CONTEXT_C_AACP` : le contexte de la politique d'administration de type `T_C_AACP`.
- `MACHINE_C_ACP` : le machine de la politique de type `T_C_ACP` avec la politique d'administration de type `T_C_AACP`. Cette machine contient des variables similaires aux machines `MACHINE_ACP` du chapitre 8 que nous renommons comme suit :
  1. `C_ACP` de type `T_C_ACP` qui contient la politique de sécurité courante.
  2. `C_LOG` avec $\texttt{C\_LOG} \in (\texttt{USER} \times \texttt{PERMISSION}) \leftrightarrow \texttt{T\_C\_ACP}$ contient l'ensemble des accès déjà effectués dans le système.

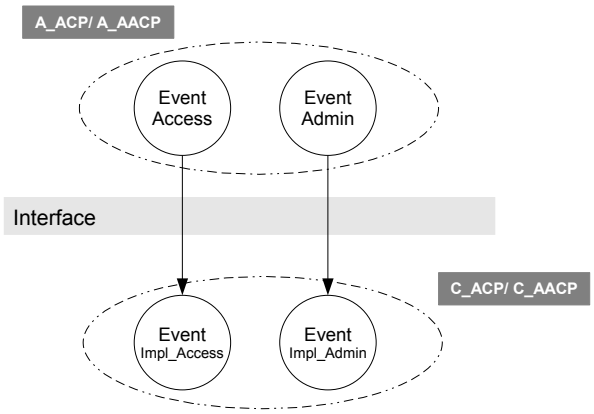


FIG. 9.2 – Le raffinement pour l'implémentation.

Nous renommons les événements `ACCESS` et `ADMIN` du modèle concret `MACHINE_C_ACP` en `Impl_ACCESS` et `Impl_ADMIN` pour éviter de les confondre avec ceux du modèle abstrait. La figure 9.2 montre la chaine de raffinement entre les événements `ACCESS` et `ADMIN` des deux modèles `MACHINE_A_ACP` et les événements

`Impl_ACCESS` et `Impl_ADMIN` du modèle `MACHINE_C_ACP`. L'outil RODIN permet de désigner les événements qui se raffinent et engendrent automatiquement les obligations de preuve qui en découlent. Le raffinement de l'événement `ACCESS` du modèle abstrait par l'événement `Impl_ACCESS` du modèle concret vise à garantir que chaque accès que fait un utilisateur qui se fait au niveau concret est également autorisé dans le niveau abstrait.

Si nécessaire une interface entre utilisateurs abstraits et concrets est rajoutée dans le modèle concret :

VARIABLES
INTERFACE
INVARIANTS
INTERFACE $\in$ USER $\leftrightarrow$ USER

L'événement `Impl_ACCESS` concret devient ainsi :

EVENT Impl_ACCESS
ANY
a_user, c_user, perm
WHERE
grd1 : a_user $\in$ USER
grd2 : c_user $\in$ USER
grd3 : c_user $\in$ INTERFACE[{a_user}]
grd4 : perm $\in$ PERMISSION
grd5 : perm $\in$ ACP_PERMISSION(C_ACP)
THEN
act1 : C_LOG := C_LOG $\cup$ {(user $\mapsto$ perm) $\mapsto$ C_ACP}
END

L'outil RODIN engendre les obligations de preuves correspondantes, la plus importante de ces obligations de preuves consiste à prouver le **Théorème *access***, celui-ci indique que chaque accès dans la politique de sécurité concrète est autorisé par la politique de sécurité abstraite :

Théorème *access* :
$\forall$ a_user, c_user, perm·
a_user $\in$ USER $\wedge$ c_user $\in$ USER $\wedge$
perm $\in$ PERMISSION $\wedge$
c_user $\in$ INTERFACE[{a_user}]] $\wedge$
c_user $\mapsto$ perm $\in$ ACP_PERMISSION(C_ACP)
$\Rightarrow$
a_user $\mapsto$ perm $\in$ ACP_PERMISSION(A_ACP)

Prouver ce théorème nécessite de d'avoir les contextes des politiques de sécurité impliqués dans l'implémentation notamment la définition du paramètre `ACP_PERMISSION` pour les politiques `A_ACP` et `C_ACP`. Il est également indispensable de formaliser la configuration de la politique de sécurité `C_ACP`

et d'utiliser les propriétés de cette configuration particulière pour prouver ce théorème.

La configuration d'une implémentation d'une politique de sécurité est un ensemble de propriétés sur les attributs des politiques source et cible. La configuration correspond à l'invariant de collage entre les modèles abstraits et concrets correspondant aux deux politiques de sécurité. Nous désignerons dans la suite de cette thèse l'invariant de collage entre les deux variables C_ACP et A_ACP par CONFIG :

$$\texttt{CONFIG}(\texttt{C\_ACP}, \texttt{A\_ACP})$$

La configuration choisie doit permettre de prouver le **Théorème *access***. Le concepteur peut partir d'une configuration initiale qui modélise la façon dont la politique de sécurité cible est configurée, par exemple, on peut implémenter les rôles RBAC avec des comptes Unix ou avec des groupes Unix. Cette configuration initiale devra ensuite être complétée pour pouvoir prouver le le **Théorème *access***. Trouver cette configuration est un problème de recherche d'invariant, car il s'agit de trouver l'invariant de collage entre les modèles abstraits et concrets correspondant aux deux politiques de sécurité. Pour trouver cette invariant nous utilisons la technique basée sur la plus faible pré-condition déjà exposée dans la section 4.2. Cette technique a permis d'obtenir des configurations complètes pour les études de cas que nous avons menées. Nous verrons un exemple de configuration dans la section 9.2.3 où un système Unix est configuré pour implémenter une politique de type RBAC.

9.2.1 L'implémentation de l'administration

Le problème de l'implémentation de l'administration peut être posé de la manière suivante : un utilisateur a_user veut remplacer la politique abstraite courante A_ACP par une autre politique new_A_ACP alors au niveau concret la politique courante concrète C_ACP est remplacée par une politique new_C_ACP. Il faut vérifier que si le changement est autorisé au niveau concret alors il est également autorisé au niveau abstrait. L'événement Impl_ADMIN de l'implémentation est comme suit :

EVENT Impl_ADMIN
ANY
a_user, c_user, new_A_ACP, new_C_ACP
WHERE
grd1 : a_user ∈ USER
grd2 : c_user ∈ USER
grd3 : c_user ∈ INTERFACE[{a_user}]
grd4 : new_A_ACP ∈ T_A_ACP
grd5 : new_C_ACP ∈ T_C_ACP
grd6 : CONFIG(new_C_ACP, new_A_ACP)
grd7 : new_C_ACP ∈ ACP_ADMIN(C_AACP ↦ C_ACP ↦ c_user)
THEN
act1 : C_ACP := new_C_ACP
END

L'événement Impl_ADMIN du modèle doit raffiner l'événement ADMIN abstrait, l'outil RODIN nous engendre l'obligation de preuve suivante que nous devons prouver et que nous appellerons **Théorème *admin 1***. Ce théorème consiste à démontrer que chaque action d'un administrateur au niveau concret conforme à la politique C_AACP correspond à une action au niveau abstrait conforme à la politique A_AACP.

Théorème *admin 1* :

$$\begin{array}{l}\forall\, a\_user, c\_user, new\_A\_ACP, new\_C\_ACP \cdot \\ a\_user \in USER \land c\_user \in USER \land \\ new\_A\_ACP \in T\_A\_ACP \land new\_C\_ACP \in T\_C\_ACP \land \\ new\_C\_ACP \in CONFIG[\{new\_A\_ACP\}] \land \\ new\_C\_ACP \in ACP\_ADMIN(C\_AACP \mapsto C\_ACP \mapsto c\_user) \\ \Rightarrow \\ new\_a\_ACP \in ACP\_ADMIN(a\_AACP \mapsto a\_ACP \mapsto a\_user)\end{array}$$

Pour faire la preuve de ce théorème, il faut d'abord configurer la politique d'administration C_AACP pour qu'elle implémente la politique A_AACP. De même que pour les variables A_ACP et C_ACP, il faut générer un invariant de collage entre les deux variables A_AACP et C_AACP, on le désigne par Ad_CONFIG :

$$Ad\_CONFIG(C\_AACP, A\_AACP)$$

Les politiques d'administration ne pouvant être changées que par le super-utilisateur à travers l'événement SUPER_ADMIN, la preuve que la configuration Ad_CONFIG est maintenue par cet événement est triviale dans la mesure où l'on suppose que le super-utilisateur à un comportement qui ne nuit pas au système. Nous utilisons pour cela un événement de type *keep* qui maintient la propriété Ad_CONFIG où A_SUPER_USER est le super-utilisateur de la politique A_AACP et C_SUPER_USER est le super-utilisateur de la politique C_AACP. Cet événement raffine l'événement SUPER_ADMIN abstrait :

EVENT SUPER_ADMIN
ANY
a_user, new_A_AACP, new_C_AACP
WHERE
grd1 : a_user = A_SUPER_USER
grd2 : A_SUPER_USER ∈ INTERFACE[{a_user}]
grd3 : new_C_AACP ∈ T_C_AACP
grd4 : Ad_CONFIG(new_C_AACP, new_A_AACP)
THEN
act1 : C_AACP := new_C_AACP
END

Nous récapitulons l'ensemble de notre méthodologie dans la figure 9.3. globalement, notre démarche consiste à réutiliser les contextes déjà définis pour chacune

des politiques de sécurité avec leurs politiques d'administration correspondantes et a trouver les configurations correspondant aux invariants de collages `CONFIG` et `Ad_CONFIG` entre les politiques de sécurité et entre les politiques d'administration.

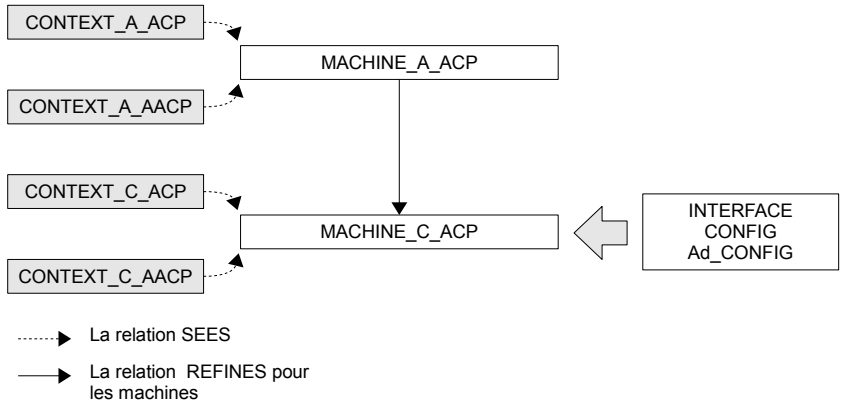


FIG. 9.3 – Le schéma général de l'implémentation.

9.2.2 L'environnement de l'implémentation

Dans notre méthodologie pour l'implémentation présentée dans cette section, nous n'avons pas pris en considération l'environnement dans lequel se fait l'implémentation. Si, par exemple, un serveur d'une application donnée avec la politique de sécurité OrBAC est implémenté sur une politique RBAC dans un système SELinux (où RBAC est disponible par défaut). Quand un utilisateur OrBAC de l'application se connecte avec un nom d'utilisateur OrBAC et celui-ci est traduit en mot de passe SELinux par une interface pour avoir accès aux objets. Les utilisateurs de SELinux qui n'ont rien à voir avec l'application OrBAC implémentée peuvent interférer avec les objets dont l'accès est censé être réglementé par OrBAC. On peut craindre qu'un administrateur de SELinux puisse modifier la politique de sécurité et donc que la configuration de l'implémentation ne soit plus correcte. Il est possible d'introduire de nouveaux événements pour prendre en considération l'environnement dans lequel est réalisée l'implémentation, ces événements agissent comme des démons et peuvent engendrer une situation où la configuration de l'implémentation ne soit plus vérifiée. Ces démons ne sont pas forcément malveillants, ils peuvent découler d'une utilisation normale du système sur lequel est réalisée l'implémentation. Il est ainsi possible d'introduire, si nécessaire, les événements `ACCESS` et `ADMIN` du modèle de la politique de sécurité sur laquelle se fait l'implémentation. Ceci aura comme conséquence d'avoir des obligations de preuves supplémentaires générées par RODIN pour s'assurer que la configuration de l'implémentation reste une propriété d'invariance. La figure 9.4 montre les événements de cette implémentation.

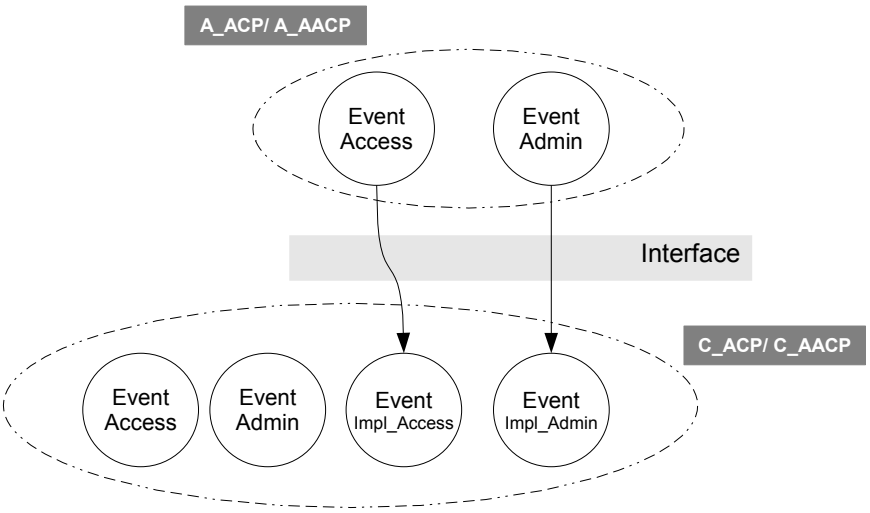


FIG. 9.4 – Le raffinement pour l'implémentation.

9.2.3 Étude de cas : Le modèle HierRBAC sous Linux

Nous présentons dans cette section une étude de cas que nous avons menée et où nous avons appliqué la méthodologie exposée dans la section précédente. Nous avons ainsi prouvé la correction de l'implémentation d'une politique de sécurité de type RBAC muni d'une politique d'administration de type ARBAC97 sur un système Linux POSIX. Plusieurs façons d'implémenter RBAC sous Linux ont été proposées dans la littérature. Glenn [48] propose une solution où les rôles de RBAC sont implémentés sous la forme de comptes utilisateurs Linux, une autre solution a été proposée par Sandhu et Ahm [72], celle-ci consiste à implémenter les rôles Linux sous forme de groupes Linux. Néanmoins, aucune de ces solutions ne proposent une preuve de leur implémentation. Brucker et Wolf [31] prouvent que leur implémentation est correcte mais ne propose pas de méthode générale pour prouver la correction d'une implémentation, de plus, parmi toutes les implémentations de politique de sécurité proposées dans la littérature (pas uniquement pour RBAC), nous n'avons pas trouvé une implémentation qui prenne en compte l'administration.

Nous prenons en considération l'environnement de l'implémentation dans notre étude de cas. Ainsi, le comportement des utilisateurs Linux qui n'ont rien à avoir avec l'implémentation mais qui risque néanmoins d'en perturber le fonctionnement. Les événements de nos modèles sont donc ceux de la figure 9.4.

Nous avons présenté dans le chapitre 8 la formalisation B événementielle des modèles RBAC, Linux ainsi que ARBAC97. Les contextes décrivant chaque politique de contrôle d'accès peuvent être chargés dans l'outil RODIN. La prochaine étape dans la concrétisation de l'implémentation est de définir l'interface ainsi que la configuration.

Il s'agit, dans notre étude de cas, d'implémenter un serveur pour une application où les droits d'accès aux fichiers suivent le modèle RBAC. Les utilisateurs se connectent à cette application avec un nom d'utilisateur propre à l'application. Les différents composants de la politique RBAC (UA, RH, PA) sont implémentés comme suit :

- PA : les rôles sont implémentés comme des comptes Linux et héritent donc des privilèges de ceux-ci notamment pour l'accès aux fichiers. Nous utilisons une relation R2L_USER qui associe les rôles aux comptes Linux. Cette relation est déclarée comme variable comme suit :

VARIABLES
R2L_USER
INVARIANTS
R2L_USER $\in$ ROLE $\leftrightarrow$ USER

- RH : nous utilisons la notion de groupe sous Linux pour implémenter la hiérarchie des rôles. Chaque rôle est associé à un groupe. Quand un rôle r1 est un sous-rôle de r2 alors le compte Linux associé à r1 (R2L_USER(r1)) est membre du groupe associé à r2. Pour associer les rôles aux groupes, nous introduisons une relation R2GROUP.

VARIABLES
R2GROUP
INVARIANTS
R2GROUP $\in$ ROLE $\leftrightarrow$ GROUP

- UA : l'association des utilisateurs RBAC aux rôles n'est quant à elle pas implémentée, elle est gérée par l'application RBAC elle même. Elle est stockée dans un fichier pour la consulter ou la modifier au besoin.

Tout cela ce traduit dans la configuration initiale, ainsi que dans l'interface. Cette dernière est définie en fonction de la politique de sécurité RBAC puisqu'elle inclut dans sa définition le paramètre UA :

VARIABLES
INTERFACE
INVARIANTS
INTERFACE $\in$ HierRBAC $\rightarrow$ (USER $\leftrightarrow$ USER)
$\forall$rbac.rbac $\in$ HierRBAC $\Rightarrow$ INTERFACE(rbac) = UA(rbac);R2L_USER

Il faut noter que le sous-ensemble de fichiers du système Linux qui sont gérés par une politique RBAC sont les fichiers qui sont la propriété d'un compte Linux dédié à un rôle RBAC par la relation R2L_USER. Dans une politique Linux donnée, ces fichiers sont donc le domaine de la relation FILE_U(linux);$(\text{R2L\_USER})^{-1}$. Comme configuration initiale nous avons une série d'invariants. Le premier indique que les droits des ces fichiers gérés par RBAC qui leur sont attribués sous Linux avec l'attribut FILE_U_P correspondent à ceux qui leur sont attribués sous RBAC avec la relation PA.

INVARIANTS
/* INV 1 */
$\forall$ rbac, linux.CONFIG(rbac, linux)
$\Rightarrow$
$\forall$file, operation, role.
file $\in$ FILE $\wedge$ operation $\in$ OPERATION $\wedge$ role $\in$ ROLE $\wedge$
file $\in$ dom(FILE_U(linux); (R2L_USER)$^{-1}$) $\wedge$
role $\in$ (FILE_U(linux); (R2L_USER)$^{-1}$)[{file}] $\wedge$
operation $\in$ FILE_U_P(linux)(file)
$\Rightarrow$
(role $\mapsto$ OF2P(operation $\mapsto$ file)) $\in$ PA(rbac)

Le deuxième invariant permet d'implémenter la hiérarchie des rôles, il indique que pour mettre un rôle r1 comme sous-rôle de r2 sous RBAC il faut que le compte LINUX associé à r2 soit membre du groupe associé au rôle r1, il pourra ainsi en hériter les privilèges.

/* INV 2 */
$\forall$ rbac, linux.CONFIG(rbac, linux)
$\Rightarrow$
$\forall$r1, r2·
r1 $\in$ dom(R2L_USER) $\wedge$ r2 $\in$ dom(R2L_USER) $\wedge$
R2GROUP(r1) $\in$ U_GROUP(linux)(R2L_USER(r2)) $\wedge$
$\Rightarrow$
r1 $\in$ (RH(rbac))[{r2}]

Quand nous avons essayé de prouver notre implémentation avec cette configuration, celle-ci s'avéra incomplète. Nous avons ensuite généré le reste de l'invariant avec la plus faible pré-condition, nous désignerons en ce qui suit les utilisateurs Linux *normaux* (des comptes utilisateurs qui ne sont pas utilisés pour implémenter les rôles) comme ceux faisant partie d'un ensemble N_USER. Il faut noter que parmi les propriétés que nous avons obtenue, certaines ne sont pas intuitives, nous nous efforcerons de leur associer une explication.

L'invariant numéro trois indique que les relations R2L_USER et R2GROUP sont des fonctions injectives. Si deux rôles sont associés au même compte Linux, ils hériteront tous les deux des privilèges de l'autre.

/* INV 3 */
R2L_USER $\in$ ROLE $\rightarrowtail$ USER $\wedge$
R2GROUP $\in$ ROLE $\rightarrowtail$ GROUP $\wedge$
dom(R2L_USER) = dom(R2GROUP)

L'invariant numéro quatre indique que les comptes Linux utilisés pour implémenter les rôles ne doivent pas être attribués au reste des utilisateurs normaux de Linux :

$$/* \text{ INV } 4 */$$
$$\texttt{N\_USER} \cap \texttt{ran}(\texttt{R2L\_USER}) = \varnothing$$

L'invariant numéro cinq indique que l'utilisateur `root` ne doit pas être utilisé pour implémenter un rôle :

$$/* \text{ INV } 5 */$$
$$\texttt{root} \notin \texttt{ran}(\texttt{R2L\_USER})$$

L'invariant numéro six indique que les groupes utilisés pour implémenter la hiérarchie des rôles ne doivent pas être confondus avec les groupes des utilisateurs ordinaires :

$$/* \text{ INV } 6 */$$
$$\forall\, \texttt{user} \cdot \texttt{user} \in \texttt{N\_USER} \Rightarrow \texttt{ran}(\texttt{R2L\_GROUP}) \cap \texttt{U\_GROUP}(\texttt{user}) = \varnothing$$

Les fichiers gérés avec la politique RBAC ne doivent pas être en accès public sous Linux, il est clair que si c'est le cas n'importe quel utilisateur Linux pourra y accéder :

$$/* \text{ INV } 7 */$$
$$\forall\, \texttt{rbac}, \texttt{linux}.\texttt{CONFIG}(\texttt{rbac}, \texttt{linux})$$
$$\Rightarrow$$
$$\forall\, \texttt{file} \cdot$$
$$\texttt{file} \in \texttt{dom}(\texttt{File\_U}(\texttt{linux})); (\texttt{R2USER})^{-1})$$
$$\Rightarrow$$
$$\texttt{FILE\_P\_P}(\texttt{linux})(\texttt{file}) = \varnothing$$

Un rôle de RBAC ne peut restreindre l'héritage de ses privilèges aux rôles qui lui sont supérieurs, ainsi, les droits d'accès aux fichiers contrôlés par RBAC sont les mêmes pour le propriétaire du fichiers et pour les membres du groupe associé au fichier :

$$/* \text{ INV } 8 */$$
$$\forall\, \texttt{rbac}, \texttt{linux}.\texttt{CONFIG}(\texttt{rbac}, \texttt{linux})$$
$$\Rightarrow$$
$$\forall\, \texttt{file} \cdot$$
$$\texttt{file} \in \texttt{dom}(\texttt{File\_U}(\texttt{linux})); (\texttt{R2USER})^{-1})$$
$$\Rightarrow$$
$$\texttt{FILE\_U\_P}(\texttt{linux})(\texttt{file}) = \texttt{FILE\_G\_P}(\texttt{linux})(\texttt{file})$$

Pour pouvoir implémenter correctement la hiérarchie des rôles, le compte Linux propriétaire d'un fichier ainsi que le groupe auquel est associé ce même fichier sont associés au même rôle avec les relation `R2L_USER` et `R2GROUP` :

$$
\begin{array}{l}
/* \text{ INV 9 } */ \\
\quad \forall\, \texttt{rbac}, \texttt{linux}.\texttt{CONFIG}(\texttt{rbac}, \texttt{linux}) \\
\quad \Rightarrow \\
\quad \texttt{FILE\_U}(\texttt{linux}); \texttt{R2L\_USER}^{-1} = \texttt{FILE\_G}(\texttt{linux}); \texttt{R2GROUP}^{-1}
\end{array}
$$

9.2.4 Étude de cas : Le modèle ARBAC97 sous Linux

Nous présentons dans cette section la vérification d'une implémentation possible du modèle RRA97 qui est le composant du modèle ARBAC97 qui permet d'administrer la hiérarchie des rôles. Ceci est une possible implémentation de l'administration pour l'application RBAC présentée dans la section précédente. Nous avions déjà présenté dans le chapitre 7.3.2.1.3 la formalisation du modèle RRA97 dans le modèle B événementiel. Pour implémenter le modèle RRA97, il faut notamment implémenter la relation `can_modify` qui associait chaque rôle administrateur à un ensemble de rôles RBAC sur lesquels il a le pouvoir de changer la hiérarchie. Cette implémentation se fera à l'aide de la commande *gpasswd* déjà définie dans la section 8.5. Cette commande permettait notamment de définir des administrateurs pour les groupes sous UNIX, ces administrateurs pouvaient entre autre ajouter des utilisateurs à un groupe dont il sont administrateurs.

Pour implémenter le modèle RRA97, il faut faire des choix sur la façon d'implémenter les rôles administratifs ainsi que les différents composants `A_UA` et `can_modify`. Ces choix sont ensuite formalisés dans le contexte dédié à l'implémentation comme nous avons fait pour l'implémentation de RBAC.

9.2.5 L'implémentation des rôles administratifs

Les utilisateurs de notre application ont un compte utilisateur propre à celle-ci qu'ils soient des administrateurs ou des utilisateurs ordinaires. Avec son compte, un utilisateur peut jouer des rôles administratifs ARBAC97 où des rôles RBAC. Nous avons fait le choix d'implémenter les rôles administratifs sous forme de comptes utilisateurs Linux (comme pour les rôles RBAC), nous utilisons pour cela une relation `AdR2L_USER` similaire à `AdR2L_USER`. Nous utilisons deux relations différentes car les types des rôles RBAC et des rôles administratifs ne sont pas les mêmes.

9.2.5.1 L'implémentation de `A_UA`

Tout comme pour le composant `UA` dans le modèle RBAC, nous inclurons le composant `A_UA` dans l'interface de notre implémentation. L'interface déjà définie pour l'implémentation de RBAC est ainsi étendue de la façon à lier un utilisateur de notre application aux comptes utilisateurs Linux pour jouer soit un rôle RBAC soit un rôle d'administrateur ARBAC97.

INVARIANTS

$$\begin{array}{l}\texttt{INTERFACE} \in (\texttt{HierRBAC} \times \texttt{ARBAC97}) \rightarrow (\texttt{USER} \leftrightarrow \texttt{USER}) \\ \forall\ \texttt{rbac}, \texttt{arbac97.rbac} \in \texttt{HierRBAC} \wedge \texttt{arbac97} \in \texttt{ARBAC97} \\ \Rightarrow \\ \texttt{INTERFACE}(\texttt{rbac} \mapsto \texttt{arbac97}) = (\texttt{UA}(\texttt{rbac}); \texttt{R2L\_USER}) \\ \qquad\qquad \cup\ (\texttt{A\_UA}(\texttt{rbac}); \texttt{AdR2L\_USER})\end{array}$$

9.2.6 L'implémentation de `can_modify`

C'est la partie la plus importante de l'implémentation du modèle RRA97. La hiérarchie des rôles dans RBAC est implémentée avec les groupes Linux. Pour rappel, un rôle `r1` est un sous rôle de `r2` si le compte Linux associé à `r1` (`R2L_USER(r1)`) est membre du groupe associé à `r2` (`R2GROUP(r2)`). Afin de contrôler la hiérarchie des rôles, il faut désigner les comptes Linux associés aux rôles administrateurs de RRA97 comme administrateurs des groupes Linux associés aux rôles. Ceci se traduit dans la configuration initiale de l'implémentation de l'administration `Ad_CONFIG` de la façon suivante :

INVARIANT

$$\begin{array}{l}\forall\ \texttt{arbac97}, \texttt{linux.AdCONFIG}(\texttt{arbac97}, \texttt{linux}) \\ \Rightarrow \\ \quad \forall\ \texttt{adrole}, \texttt{role} \\ \quad \texttt{adrole} \in \texttt{A\_ROLE} \wedge \texttt{role} \in \texttt{ROLE} \wedge \\ \quad \texttt{AdR2L\_USER}(\texttt{adrole}) \in \texttt{GROUP\_AD}(\texttt{AdLinux})(\texttt{R2GROUP}(\texttt{role})) \wedge \\ \quad \Rightarrow \\ \quad \exists \texttt{I}.\texttt{I} \in \mathbb{P}(\texttt{ROLE}) \wedge \texttt{role} \in \texttt{I} \wedge \texttt{adrole} \mapsto \texttt{I} \in \texttt{can\_modify}\end{array}$$

Les politiques administratives ne sont pas sujettes à des modifications importantes sauf par le super-utilisateur mais celui-ci n'est pas supposé avoir un comportement malveillant, de ce fait la preuve que la configuration de l'administration est une propriété d'invariance, n'est pas difficile. Par contre il faut tout de même prouver que celle-ci implémente correctement l'administration en prouvant le **Théorème *admin 1***.

9.3 L'implémentation des politiques composées

Faire la preuve de l'implémentation d'un modèle de politiques de sécurité avec un seul raffinement peut s'avérer coûteux en termes d'obligations de preuves interactives d'autant plus que les politiques de contrôle d'accès sont de plus en plus complexes. Nous proposons d'utiliser des techniques de raffinements avancées pour faciliter l'implémentation de politiques complexes. Nous avons présentés dans la section 7.6 une algèbre de composition de politiques de contrôle d'accès proposée par Bonatti *et al* [26].

L'idée générale de l'approche que nous proposons est comme suit, si une politique de contrôle d'accès de type P est obtenue en composant deux politiques de sécurité de types P_1 et P_2 alors la preuve de l'implémentation de la politique

de type P peut être obtenue en réutilisant les preuves de l'implémentation des politiques de type P_1 et $P2$ si elles sont déjà faites au préalable. Dans l'étude de cas présentée dans la section 9.2.3 où nous avons implémenté une politique de type RBAC hiérarchique, il aurait été possible de se baser sur une implémentation de politiques de type CoreRBAC et de l'étendre avec la hiérarchie des rôles sans avoir à refaire les preuves de l'implémentation de CoreRBAC.

D'un point de vue B événementiel le problème est posé de la façon suivante : un modèle de politique de contrôle d'accès T_A_ACP, est formalisé par un modèle B événementiel composé d'un contexte `CONTEXTE_A_ACP` et une machine `MACHINE_A_ACP`. Si A_ACP est implémentée par une politique de type T_C_ACP, nous appelons `CONTEXTE_C_ACP` le contexte de l'implémentation et `MACHINE_C_ACP` la machine de l'implémentation. Si cette politique de type T_A_ACP est obtenue en composant deux politiques de type $T\_A\_ACP_1$ et $T\_A\_ACP_2$, il faut alors répondre à deux questions :

- Comment formaliser en B événementiel la politique composée de type T_A_ACP à partir de la formalisation de $T\_A\_ACP_1$ et de $T\_A\_ACP_2$.
- Comment réutiliser les preuves d'implémentation de T_ACP_1 et de T_ACP_2 pour prouver la correction de l'implémentation de T_A_ACP.

9.3.1 La formalisation des politiques composées

Nous avons présenté un résumé d'une algèbre pour politiques de contrôle d'accès avec un ensemble d'opérateurs permettant de composer les politiques. Nous proposons dans cette section une formalisation des politiques obtenue en appliquant les opérateurs «+» et «&». Nous introduisons des opérateurs en B événementiel sous forme de fonctions qui prennent comme arguments deux politiques de type déterminé et renvoient la politiques obtenue par composition. La sémantique de ces opérateurs est introduite sous forme d'axiomes.

9.3.1.1 La sémantique des opérateurs «+» et «&» en B événementiel

L'opérateur «+» qui combine deux politiques en unissant les autorisations des deux politiques combinées. Quand cet opérateur est appliqué à deux politiques de sécurité `acp1` et `acp2` ceci résulte en un nouvelle politique `acp1Uacp2`. Une constante `OP_UNION` est introduite pour formaliser l'opérateur. La sémantique de l'opérateur en B événementiel est formalisée par un axiome sur la constante `OP_UNION` :

CONSTANT
OP_UNION
AXIOMS
P1_UNION_P2 ⊆ T_ACP
OP_UNION ∈ (T_ACP × T_ACP) → T_ACP
/* La sémantique de l'opérateur «+» */
∀ acp1, acp2, acp1Uacp2.
acp1 ∈ T_ACP ∧ acp2 ∈ T_ACP ∧ acp1Uacp2 ∈ T_ACP ∧
acp1Uacp2 = OP_UNION(acp1 ↦ acp2)
⇒
ACP_PERMISSION(acp1Uacp2) =
ACP_PERMISSION(acp1) ∪ ACP_PERMISSION(acp2)

L'opérateur «&» combine deux politiques en accordant les privilèges à un utilisateur uniquement si les deux politiques composées l'y autorise. Quand cet opérateur est appliqué à deux politiques de sécurité acp1 et acp2 ceci résulte en une nouvelle politique acp1Iacp2. Une constante OP_INTER est introduite pour formaliser l'opérateur. La sémantique de l'opérateur en B événementiel est formalisée par un axiome sur la constante OP_INTER :

CONSTANT
OP_INTER
AXIOMS
OP_INTER ∈ (T_ACP × T_ACP) → T_ACP
/* La sémantique de l'opérateur «&» */
∀ acp1, acp2, acp1Iacp2.
acp1 ∈ T_ACP ∧ acp2 ∈ T_ACP ∧ acp1Iacp2 ∈ T_ACP ∧
acp1Iacp2 = OP_INTER(acp1 ↦ acp2)
⇒
ACP_PERMISSION(acp1Iacp2) =
ACP_PERMISSION(acp1) ∩ ACP_PERMISSION(acp2)

9.3.2 L'implémentation des politiques composées

Étant données deux politiques de sécurités A_ACP1 et A_ACP2 déjà implémentées séparément à l'aide de deux politiques de sécurités C_ACP1 et C_ACP2 en utilisant deux configuration CONFIG1 et CONFIG2 avec :

CONFIG1(A_ACP1, C_ACP1) ∧ CONFIG2(A_ACP2, C_ACP2)

et un opérateur de composition de politique de sécurité OP alors la question à laquelle nous répondons dans cette section est : *sous quelles conditions est ce que la politique de sécurité* OP(C_ACP1 ↦ C_ACP2) *implémente elle la politique* OP(A_ACP1 ↦ A_ACP2) *?* Une question sous-jacente concerne la réutilisation des preuves déjà faites pour l'implémentation des deux politiques A_ACP1 et A_ACP2.

Le problème de l'implémentation des politiques composées n'est intéressant que si les deux politiques composées sont implémentés dans deux systèmes qui partagent une partie commune. Dans le cas contraire, où les deux politiques sont implémentées dans deux systèmes qui sont totalement disjoints alors nous verrons plus loin dans cette section que les preuves de l'implémentation de la politique résultant de la composition sont triviales. Dans le cas où les deux systèmes dans lesquels les politiques sont implémentés ont des parties communes, il peut y avoir des interférence dans le fonctionnement des deux implémentations qui peuvent avoir des effets de bords indésirables. Par exemple, si deux politiques de sécurité, une de type RBAC et l'autre de type TMAC sont implémentées dans un même système Linux en utilisant les comptes utilisateurs Linux pour implémenter les rôles de RBAC et les équipes du modèle TMAC alors l'utilisation d'un même compte utilisateur pour implémenter un rôle et une équipe à des conséquences nuisibles sur le système composé dans la mesure où un sujet jouant le rôle en question héritera des droits que confère l'appartenance à l'équipe en question.

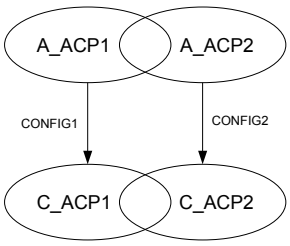


FIG. 9.5 – La composition des politiques.

La figure 9.5 résume notre problème, les deux implémentations `C_ACP1` et `C_ACP2` partagent des attributs communs. Ces attributs communs constituent des variables partagées entre les deux modèles qui peuvent poser problème au moment de leurs combinaison. Nous utiliserons les techniques de raffinements présentés dans le chapitre 3 de cette thèse pour assurer que la compositions de ces modèles est correcte. Les attributs communs constituent les variables partagées, les événements `ACCESS` des deux modèles des deux politiques de sécurité au niveau abstraits et concrets ne font que lire les politiques de sécurité et ne posent donc pas de problèmes pour la composition, par contre les événements `ADMIN` modifient les politiques de sécurités et donc les attributs communs entre elle ceci peut engendrer des interférences.

Au niveau abstrait, l'événement `ADMIN` du modèle de la politique `A_ACP1` modifie la variable `A_ACP1`, mais également la variables `A_ACP2`, en effet les interférences éventuelles au niveau concrets doivent être répercutés au niveau abstrait sinon le raffinement ne serait pas correct. Il en est de même avec la variable `ADMIN` du modèle de la politique `A_ACP2`.

9.3.2.1 L'implémentation de l'opérateur «+» :

Nous abordons dans cette section l'implémentation de l'opérateur «+», nous utilisons la technique proposée par Abrial [12, 33] et présentée dans le chapitre 3 de cette thèse basée sur les variables et les événements externes. La figure 9.6 résume la démarche que nous proposons pour obtenir la machine abstraite MACHINE_A_ACP1 + ACP2 ainsi que son implémentation MACHINE_C_ACP1 + ACP2.

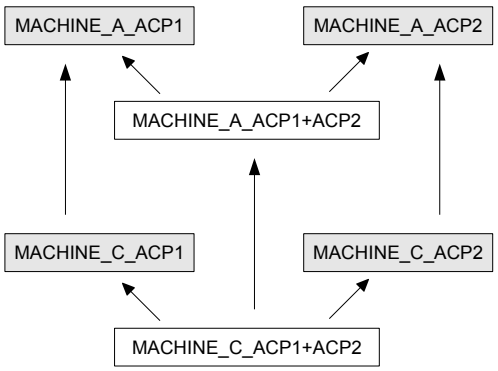


FIG. 9.6 – La composition des politiques.

Nous renommons les événements ADMIN en ADMIN1 et ADMIN2 pour les modèles MACHINE_A_ACP1 et MACHINE_A_ACP2 respectivement, il en est de même pour les événements ACCESS. Les variable A_LOG du modèle est renommée A_LOG1 et A_LOG2 respectivement dans les deux machines. Le nouveau modèle MACHINE_A_ACP1 + ACP2 construit par raffinement des deux autres contient les événements des deux modèles réunis. Á partir de la définition de l'opérateur OP_UNION, on déduit le théorème suivant :

$$
\begin{array}{l}
\forall\, \mathtt{user}, \mathtt{perm} \cdot \\
\mathtt{user} \in \mathtt{USER} \land \mathtt{perm} \in \mathtt{PERMISSION} \land \\
\mathtt{user} \mapsto \mathtt{perm} \in \mathtt{A\_LOG1} \cup \mathtt{A\_LOG2} \land \\
\Rightarrow \\
\mathtt{user} \mapsto \mathtt{perm} \in \mathtt{ACP\_PERMISSION}(\mathtt{OP\_UNION}(\mathtt{A\_ACP1} \mapsto \mathtt{A\_ACP2}))
\end{array}
$$

Ce théorème signifie que tout accès dans le nouveau système obtenu par composition est conforme à la politique obtenue en appliquant l'opérateur «+» sur les politiques A_ACP1 et A_ACP2.

Un événement externe ext_ADMIN1 est rajouté dans le modèle MACHINE_A_ACP2 qui est une abstraction de ADMIN1 mais qui ne modifie que la variable A_ACP2. De façon symétrique, un événement externe ext_ADMIN2 est rajouté dans le

modèle MACHINE_A_ACP1 qui est une abstraction de ADMIN2 mais qui ne modifie que la variable A_ACP1. Les variables et les événements externes du modèle abstrait sont résumés par la figure 9.7.

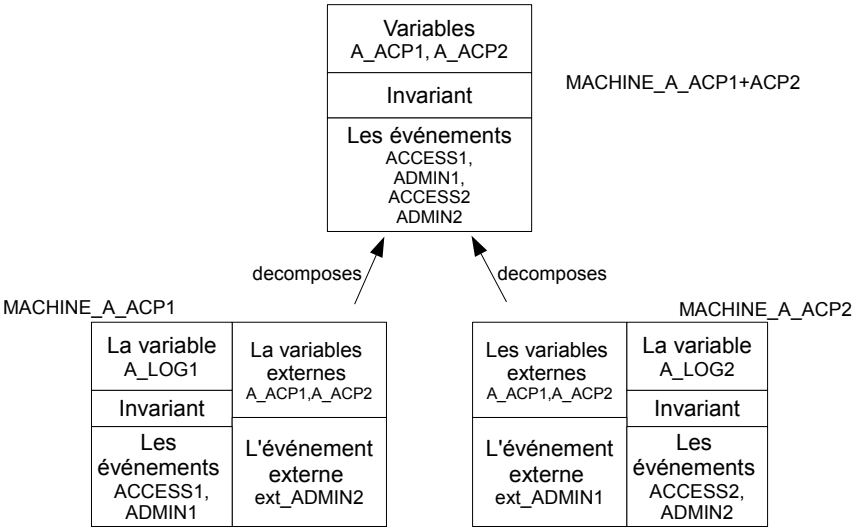


FIG. 9.7 – Les variables et événements externes.

Afin de pouvoir recombiner les deux modèles concrets MACHINE_C_ACP1 et MACHINE_C_ACP2, il est indispensable de démontrer que les variables partagées que sont A_ACP1 et A_ACP2 sont raffinées de la même manière. Ces deux variables sont raffinées chacune dans son modèle d'origine par les invariants de collage CONFIG1 et CONFIG2, il faut montrer en plus que :

- L'événement externe ext_ADMIN2 raffine la variable A_ACP1 en C_ACP1 avec l'invariant de collage CONFIG1.
- L'événement externe ext_ADMIN1 raffine la variable A_ACP2 en C_ACP2 avec l'invariant de collage CONFIG2.

En d'autre termes, il s'agit de démontrer que les actions administratives de chaque politique de sécurité au niveau concret maintient la configuration de l'autre politique de sécurité. Par contre il n'est pas nécessaire de prouver la correction du raffinement de chacune des variables A_ACP1 et A_ACP2 par les événements ADMIN1 et ADMIN2 respectivement. Une fois les preuves faites, il est possible de recombiner les deux modèles en supprimant les événements externes et en gardant les événements concrets ADMIN1, ADMIN2, ACCESS1 et ADMIN2. pour arriver au modèle final MACHINE_C_ACP1+ACP2 qui raffine MACHINE_C_ACP1+ACP2 avec les deux invariants : CONFIG1 et CONFIG2.

En utilisant le **Théorème *access*** déduit à partir de chacune des configurations CONFIG1 et CONFIG2 pour les deux politiques, il n'est pas difficile de déduire le **Théorème *access*** pour la politique composée :

Théorème *access :*

$$\forall\, a\_user, c\_user, perm \cdot$$
$$a\_user \in USER \land c\_user \in USER \land$$
$$perm \in PERMISSION \land$$
$$c\_user \in INTERFACE[\{a\_user\}]] \land$$
$$c\_user \mapsto perm \in ACP\_PERMISSION(OP\_UNION(C\_ACP1 \mapsto C\_ACP2))$$
$$\Rightarrow$$
$$a\_user \mapsto perm \in ACP\_PERMISSION(OP\_UNION(A\_ACP1 \mapsto A\_ACP2))$$

Ce théorème signifie que les accès autorisés en combinant les politiques concrètes sont également autorisés en combinant les politiques abstraites.

9.3.2.2 L'implémentation de l'opérateur «&» :

Nous utiliserons pour l'implémentation de l'opérateur «&», la technique de fusion des événements proposée par Poppelton [68] présentée dans le chapitre 3 de cette thèse. L'idée globale de la démarche est similaire à l'implémentation de l'opérateur «+» mais diffère dans la façon de combiner les modèles. Nous utiliserons dans le cas de l'opérateur «&» la fusion des événements. Notamment pour fusionner les deux événements ACCESS1 et ACCESS2. En fusionnant ces deux événements, nous obtenons un événement ACCESS1 $\odot$ ACCESS2 :

EVENT ACCESS1⊙ACCESS2
ANY
user, perm
WHERE
grd1 : user ∈ USER
grd2 : perm ∈ PERMISSION
grd3 : perm ∈ ACP_PERMISSION(A_ACP1)
grd4 : perm ∈ ACP_PERMISSION(A_ACP2)
/* Les gardes 3 et 4 sont remplacées par : */
grd34 : perm ∈ ACP_PERMISSION(OP_INTER(A_ACP1 ↦ A_ACP2))
THEN
act1 : A_LOG1 := A_LOG1 ∪ {(user ↦ perm) ↦ A_ACP1}
act2 : A_LOG2 := A_LOG2 ∪ {(user ↦ perm) ↦ A_ACP2}
END

Les événements ADMIN1 et ADMIN2 sont quand à eux fusionnés avec l'événement *skip*, on obtient ainsi la machine abstraite MACHINE_A_ACP1&ACP2. Le problème des variables partagées et de l'interférence entre les deux modèles se pose de la même manière que pour l'opérateur «+». La technique proposée par Poppleton [68] établit que la fusion des machines concrètes raffinent la fusion des machines abstraites à condition de raffiner les variables partagées de la même manière dans les deux modèles composés.

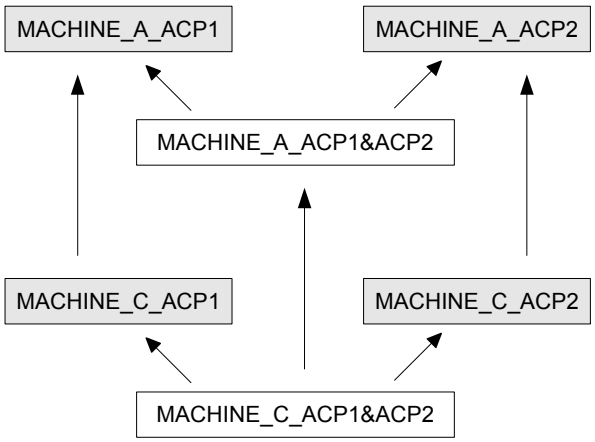


FIG. 9.8 – L'implémentation de l'opérateur «&»

Les deux événements ACCESS1 et ACCESS2 concrets sont fusionnés de la même manière que les événements abstraits pour obtenir l'événement ACCESS1⊙ACCESS2 concret :

EVENT Impl_ACCESS
ANY
a_user, c_user, perm
WHERE
grd1 : a_user ∈ USER
grd2 : c_user ∈ USER
grd3 : c_user ∈ INTERFACE[{a_user}]
grd4 : perm ∈ PERMISSION
grd5 : perm ∈ ACP_PERMISSION(C_ACP1)
grd6 : perm ∈ ACP_PERMISSION(C_ACP2)
/* Les gardes 5 et 6 sont remplacées par : */
grd56 : perm ∈ ACP_PERMISSION(OP_INTER(C_ACP1 ↦ C_ACP2))
THEN
act1 : C_LOG1 := C_LOG1 ∪ {(user ↦ perm) ↦ C_ACP1}
act2 : C_LOG2 := C_LOG2 ∪ {(user ↦ perm) ↦ C_ACP2}
END

La condition de raffiner les variables partagées de la même manière se traduit en prouvant en plus des raffinements déjà prouvés pour chaque modèle séparé que :

- L'événement ADMIN2 raffine la variable A_ACP1 en C_ACP1 avec l'invariant de collage CONFIG1.
- L'événement ADMIN1 raffine la variable A_ACP2 en C_ACP2 avec l'invariant de collage CONFIG2.

En utilisant le **Théorème *access*** déduit à partir de chacune des configurations CONFIG1 et CONFIG2 pour les deux politiques, il n'est pas difficile de déduire le **Théorème *access*** pour la politique composée :

Théorème *access* :

$$\forall\, a\_user, c\_user, perm \cdot$$
$$a\_user \in USER \land c\_user \in USER \land$$
$$perm \in PERMISSION \land$$
$$c\_user \in INTERFACE[\{a\_user\}]] \land$$
$$c\_user \mapsto perm \in ACP\_PERMISSION(OP\_INTER(C\_ACP1 \mapsto C\_ACP2))$$
$$\Rightarrow$$
$$a\_user \mapsto perm \in ACP\_PERMISSION(OP\_INTER(A\_ACP1 \mapsto A\_ACP2))$$

On obtient ainsi le modèle de l'implémentation de la politique composée avec l'opérateur «&».

9.4 Conclusion

Nous avons présenté dans ce chapitre une méthode générale pour l'implémentation des politiques de contrôle d'accès avec leurs administration. La méthode est générale dans le sens où elle est définie sur des modèles de politiques de contrôle d'accès paramétrés avec ACP_PERMISSION et ACP_ADMIN. La démarche de l'implémentation peut être optimisée en étant effectuée sur plusieurs raffinements grâce aux opérateurs «+» et «&» et aux techniques de décomposition de modèles. Nous avons appliqué cette méthode sur une étude de cas dans laquelle une politique de contrôle d'accès RBAC avec une politique d'administration de type ARBAC97 a été implémentée dans un système Linux.

Chapitre 10

Conclusion

Dans cette thèse, nous avons proposé une méthode pour la composition des protocoles de sécurité avec la méthode B événementielle basée sur la décomposition des modèles et sur la fusion des événements. Nous avons également utilisé le calcul des invariants en utilisant le concept de la plus faible pré-condition. Nous avons appliqué notre méthode sur les protocoles cryptographiques dans le modèle formel pour vérifier les propriétés d'authentification et d'établissement de clés. Les protocoles sont d'abord modélisés séparément avant d'être composé pour former des protocoles plus complexes. Les principales propriétés de sûreté sont modélisées à l'aide de spécifications abstraites qui peuvent être réutilisées dans différents développements. Nous avons également proposé une modélisation en B événementielle du modèle de l'attaquant de Dolev-Yao qui peut être réutilisée dans plusieurs études de cas. Nous avons également proposé une méthode qui permet de reconstruire d'éventuelles attaques de l'attaquant.

Nous nous sommes également intéressés à la problématique du contrôle d'accès où nous avons proposé une modélisation B événementielle pour les modèles de politiques de contrôle d'accès que nous avons appliquée sur quelques exemples (RBAC, DAC). Une modélisation des modèles d'administration a également été proposée. Nous avons ensuite utilisé ces modèles pour faire la preuve de la correction de leur implémentation dans d'autres modèles de politique de contrôle d'accès. Nous avons également proposé une méthode pour la modélisation et l'implémentation des politiques de contrôle d'accès composées.

Nous pensons que certains aspects méthodologiques relatifs à la méthode B événementielle que nous avons utilisés tels que la génération d'invariants, la reconstitution des traces d'événements violant une propriété de sûreté ou encore le fait de rendre deux modèles *fusionables* en conservant les preuves déjà faites sur ces modèles peuvent être réutilisés dans d'autres développement de modèles en B événementiel.

De nombreuses méthodes ont été proposées pour la composition et la décomposition des systèmes en général et pour la méthode B en particulier mais, paradoxalement, il y a peu d'études de cas complexes où ces techniques ont été appliquées. Nous pensons avoir apporté une contribution dans ce sens avec la composition des protocoles de sécurité.

10.1 Perspectives

Une première perspective de notre travail consiste à ré-appliquer notre méthode de composition de protocoles sur des études de cas de protocoles cryptographiques plus complexes. Il est aussi nécessaire d'élargir le spectre des propriétés de sécurité relatives aux protocoles cryptographiques, il est en effet nécessaire de pouvoir spécifier des propriétés autres que les propriétés d'authentification et d'établissement de clés. Il est aussi envisagé de mécaniser les différentes étapes de modélisation dans un outil de composition des protocoles, pour cela il faut :

- Mécaniser le processus de modélisation des protocoles cryptographiques dans la méthode B événementielle. L'idée est de se baser sur une description des protocoles de sécurité dans un langage dédié et d'obtenir mécaniquement le modèle B événementiel concret du protocole. Il existe déjà plusieurs langages de description de protocoles cryptographiques tels que HLPSL (*The High Level Protocol Specification Language*) développé dans le cadre du projet AVISPA [1].
- Mécaniser le processus de génération d'invariants, jusqu'à maintenant la génération des fragments d'invariant par le calcul de la plus faible précondition se fait en récupérant les obligations de preuves calculées par l'outil RODIN et en les réinjectant manuellement dans la clause INVARIANTS. Ceci pourrait être réalisé grâce à un plug-in qui sera rajouté à l'outil RODIN et qui pourra être utilisé dans des développements autres que celui des protocoles.

Nous avons au cours de notre thèse proposé une modélisation pour les modèles de politique de contrôle d'accès et utilisé la modélisation obtenue pour vérifier la correction des implémentation de ces politiques. Ces modèles peuvent également être utilisés pour vérifier la consistance de ces politiques. Les problèmes de consistance sont nombreux. Par exemple, dans le modèle ARBAC97, il faut vérifier les différentes propriétés des intervalles d'autorités, dans le modèle OrBAC, il faut vérifier la consistance des permissions et des interdictions définies. Pour RBAC, il faut vérifier la consistance de séparation des pouvoirs statique avec la hiérarchie des rôles. Nous pouvons, pour cela utiliser l'outil RODIN ainsi que le model-checker ProB.

Il est également envisagé d'aborder le problème des obligations et des recommandations dans les politiques de contrôle d'accès. Ces notions sont exprimées dans les politiques de contrôle d'accès mais de façon abstraites comme dans le modèle OrBAC. Nous pensons qu'il est nécessaire de proposer des mécanismes plus concrets qui permettent de satisfaire à ce genre d'exigences.

Nous devons également effectuer d'autres études de cas de vérification de l'implémentation de politiques de contrôle d'accès et notamment des politiques composées. De plus en plus de modèles de politiques de contrôle d'accès sont proposées pour des applications particulières sans qu'il n'existe une implémentation de ces politiques, ceci nous fournit des études de cas potentielles. Nous pouvons par exemple prouver l'implémentation des politiques TMAC sur des systèmes RBAC proposée par [81].

L'autre perspective de notre travail concerne la vérification de mécanismes de sécurité garantissant à la fois les propriétés d'authentification et les propriétés d'autorisation. Il existe de nombreux travaux dans ce sens utilisant, entre autres mécanismes, les certificats digitaux [47].

Annexe A

Les modèles B événementiels du mécanisme 3

CONTEXT C0
SETS
TRANSACTION
MSG
AGENT
CONSTANTS
I
S
T_SRC
T_B_DST
MSG_SRC
MSG_DST
AXIOMS
axm1 : $S \in AGENT$
axm2 : $I \in AGENT$
axm3 : $S \neq I$
axm4 : $T\_SRC \in TRANSACTION \rightarrow AGENT$
axm5 : $T\_B\_DST \in TRANSACTION \rightarrow AGENT$
axm6 : $MSG\_SRC \in MSG \rightarrow AGENT$
axm7 : $MSG\_DST \in MSG \rightarrow AGENT$
END

CONTEXT C1
EXTENDS C0
SETS
S_KEY
CONSTANTS
K1
K2

AXIOMS

axm1 : $K1 \in S\_KEY \rightarrow AGENT$

axm2 : $K2 \in S\_KEY \rightarrow AGENT$

END

MACHINE M0
SEES C0
VARIABLES

T_DST

init

end

INVARIANTS

inv1 : $T\_DST \in TRANSACTION \rightarrow BOOL$

inv2 : $init \subseteq TRANSACTION$

inv3 : $end \subseteq init$

inv5 : $\forall T \cdot T \in end \Rightarrow T\_DST(T) = TRUE$

EVENTS
Initialisation

begin

/* L'initialisation des variables avec des ensembles vides */

end

Event $INIT \mathrel{\widehat{=}}$

any

T

where

grd1 : $T \in TRANSACTION$

grd2 : $T \notin init$

grd3 : $T \notin end$

then

act1 : $init := init \cup \{T\}$

act2 : $T\_DST(T) := FALSE$

end

Event $ACTIVE \mathrel{\widehat{=}}$

any

T

B

where

grd1 : $T \in TRANSACTION$

grd2 : $B \in AGENT$

grd3 : $B = T\_B\_DST(T)$

then

act1 : $T\_DST(T) := TRUE$

end

Event *END* $\widehat{=}$

any

T

where

grd1 : $T \in TRANSACTION$

grd2 : $T \in init$

grd3 : $T\_DST(T) = TRUE$

then

act1 : $end := end \cup \{T\}$

end

END

MACHINE M1
REFINES M0
SEES C1
VARIABLES

init
end
CHAN1
CHAN2
CHAN3
CHAN1_A
CHAN1_Na
CHAN2_B
CHAN2_A
CHAN2_Na
CHAN2_Kbs
CHAN3_B
CHAN3_Na
CHAN3_Kas
CHAN3_Kab
FRAG2
FRAG3
FRAG2_A
FRAG2_Na
FRAG2_Kbs
FRAG3_B
FRAG3_Na
FRAG3_Kas
FRAG3_Kab
N_MEM
KEY_MEM

INVARIANTS

Tinv1 : $CHAN3 \subseteq MSG$
Tinv2 : $CHAN2 \subseteq MSG$
Tinv3 : $CHAN1 \subseteq MSG$
Tinv4 : $CHAN1\_A \in CHAN1 \rightarrow AGENT$
Tinv5 : $CHAN1\_Na \in CHAN1 \rightarrow TRANSACTION$
Tinv6 : $CHAN2\_B \in CHAN2 \rightarrow AGENT$
Tinv7 : $CHAN2\_A \in CHAN2 \rightarrow AGENT$
Tinv8 : $CHAN2\_Na \in CHAN2 \rightarrow TRANSACTION$
Tinv9 : $CHAN2\_Kbs \in CHAN2 \rightarrow S\_KEY$
Tinv10 : $CHAN3\_B \in CHAN3 \rightarrow AGENT$
Tinv11 : $CHAN3\_Na \in CHAN3 \rightarrow TRANSACTION$
Tinv12 : $CHAN3\_Kas \in CHAN3 \rightarrow S\_KEY$
Tinv13 : $FRAG2 \subseteq MSG$
Tinv14 : $FRAG3 \subseteq MSG$
Tinv15 : $FRAG2\_A \in FRAG2 \rightarrow AGENT$
Tinv16 : $FRAG2\_Na \in FRAG2 \rightarrow TRANSACTION$
Tinv17 : $FRAG2\_Kbs \in FRAG2 \rightarrow S\_KEY$
Tinv18 : $FRAG3\_B \in FRAG3 \rightarrow AGENT$
Tinv19 : $FRAG3\_Na \in FRAG3 \rightarrow TRANSACTION$
Tinv20 : $FRAG3\_Kas \in FRAG3 \rightarrow S\_KEY$
Tinv21 : $FRAG3\_Kab \in FRAG3 \rightarrow S\_KEY$
Tinv22 : $CHAN3\_Kab \in CHAN3 \rightarrow S\_KEY$
Tinv23 : $N\_MEM \subseteq TRANSACTION$
Tinv24 : $KEY\_MEM \in AGENT \rightarrow \mathbb{P}(S\_KEY)$
/* La propriété P */
P : $\forall msg3 \cdot$
$msg3 \in CHAN3 \wedge$
$CHAN3\_Na(msg3) \in init \wedge$
$CHAN3\_B(msg3) = T\_B\_DST(CHAN3\_Na(msg3)) \wedge$
$K1(CHAN3\_Kas(msg3)) = T\_SRC(CHAN3\_Na(msg3)) \wedge$
$K2(CHAN3\_Kas(msg3)) = S$
$\Rightarrow$
$T\_DST(CHAN3\_Na(msg3)) = TRUE$
/* INV1=[SEND3](P) */
INV1 : $\forall msg2 \cdot$
$msg2 \in CHAN2 \wedge$
$K1(CHAN2\_Kbs(msg2)) = CHAN2\_B(msg2) \wedge$
$K2(CHAN2\_Kbs(msg2)) = S \wedge$
$CHAN2\_Na(msg2) \in init \wedge$
$CHAN2\_B(msg2) = T\_B\_DST(CHAN2\_Na(msg2))$
$\Rightarrow$
$T\_DST(CHAN2\_Na(msg2)) = TRUE$
/* INV2=[GENERATE3](P) */
INV2 : $\forall frag \cdot$
$frag \in FRAG3 \wedge$

$FRAG3\_Na(frag) \in init \wedge$
$FRAG3\_B(frag) = T\_B\_DST(FRAG3\_Na(frag)) \wedge$
$K1(FRAG3\_Kas(frag)) = T\_SRC(FRAG3\_Na(frag)) \wedge$
$K2(FRAG3\_Kas(frag)) = S$
$\Rightarrow$
$T\_DST(FRAG3\_Na(frag)) = TRUE$

/* INV3=[GENERATE2](INV1) */

INV3 : $\forall frag \cdot$
$frag \in FRAG2 \wedge$
$K1(FRAG2\_Kbs(frag)) = T\_B\_DST(FRAG2\_Na(frag)) \wedge$

$K2(FRAG2\_Kbs(frag)) = S \wedge$
$FRAG2\_Na(frag) \in init$
$\Rightarrow$
$T\_DST(FRAG2\_Na(frag)) = TRUE$

/* INV4=[LISTEN2](INV3) */

INV4 : $\forall msg2 \cdot$
$msg2 \in CHAN2 \wedge$
$K1(CHAN2\_Kbs(msg2)) = T\_B\_DST(CHAN2\_Na(msg2)) \wedge$

$K2(CHAN2\_Kbs(msg2)) = S \wedge$
$CHAN2\_Na(msg2) \in init$
$\Rightarrow$
$T\_DST(CHAN2\_Na(msg2)) = TRUE$

INV5 : $\forall T, Kbs \cdot$
$T \in N\_MEM \wedge$
$Kbs \in KEY\_MEM(I) \wedge$
$K1(Kbs) = T\_B\_DST(T) \wedge$
$K2(Kbs) = S \wedge$
$T \in init$
$\Rightarrow$
$T\_DST(T) = TRUE$

INV6 : $\forall msg1, Kbs \cdot$
$msg1 \in CHAN1 \wedge$
$Kbs \in KEY\_MEM(I) \wedge$
$CHAN1\_Na(msg1) \in init \wedge$
$K1(Kbs) = T\_B\_DST(CHAN1\_Na(msg1)) \wedge$
$K2(Kbs) = S$
$\Rightarrow$
$T\_DST(CHAN1\_Na(msg1)) = TRUE$

INV7 : $\forall Kbs, T \cdot$
$Kbs \in KEY\_MEM(I) \wedge$
$K1(Kbs) = T\_B\_DST(T) \wedge$
$K2(Kbs) = S$
$\Rightarrow$
$\bot$

INV8 : $\forall T, Kas \cdot$
$T \in N\_MEM \wedge$
$Kas \in KEY\_MEM(I) \wedge$

$$
\begin{array}{l}
T \in init \land \\
K1(Kas) = T\_SRC(T) \land \\
K2(Kas) = S \\
\Rightarrow \\
T\_DST(T) = TRUE
\end{array}
$$

INV9 : $\forall msg1, Kas \cdot$

$$
\begin{array}{l}
msg1 \in CHAN1 \land \\
Kas \in KEY\_MEM(I) \land \\
K1(Kas) = T\_SRC(CHAN1\_Na(msg1)) \land \\
K2(Kas) = S \land \\
CHAN1\_Na(msg1) \in init \\
\Rightarrow \\
T\_DST(CHAN1\_Na(msg1)) = TRUE
\end{array}
$$

INV10 : $\forall Kas, T \cdot$

$$
\begin{array}{l}
Kas \in KEY\_MEM(I) \land \\
T \in N\_MEM \land \\
K1(Kas) = T\_SRC(T) \land \\
K2(Kas) = S \land \\
T \notin init \\
\Rightarrow \\
\bot
\end{array}
$$

INV11 : $\forall Kas, T \cdot$

$$
\begin{array}{l}
Kas \in KEY\_MEM(I) \land \\
K1(Kas) = T\_SRC(T) \land \\
K2(Kas) = S \land \\
T \notin init \\
\Rightarrow \\
\bot
\end{array}
$$

INV12 : $\forall Kas, T \cdot$

$$
\begin{array}{l}
Kas \in KEY\_MEM(I) \land \\
T \in N\_MEM \land \\
K1(Kas) = T\_SRC(T) \land \\
K2(Kas) = S \land \\
T \in init \\
\Rightarrow \\
\bot
\end{array}
$$

INV13 : $\forall Kas, T \cdot$

$$
\begin{array}{l}
Kas \in KEY\_MEM(I) \land \\
K1(Kas) = T\_SRC(T) \land \\
K2(Kas) = S \land \\
T \in init \\
\Rightarrow \\
\bot
\end{array}
$$

INV14 : $\forall T \cdot$

$$
\begin{array}{l}
T \in init \land \\
T \in N\_MEM \land \\
T\_B\_DST(T) = I \\
\Rightarrow \\
T\_DST(T) = TRUE
\end{array}
$$

EVENTS
Initialisation

begin

/* L'initialisation des variables avec des ensembles vides */

end

Event *SEND1* $\widehat{=}$
refines *INIT*

any

T
$msg1$

where

grd1 : $T \in TRANSACTION$
grd2 : $T \notin init$
grd3 : $T \notin step2$
grd4 : $msg1 \in MSG$
grd5 : $msg1 \notin CHAN1$
grd6 : $msg1 \notin CHAN2$
grd7 : $msg1 \notin CHAN3$
grd8 : $MSG\_SRC(msg1) = T\_SRC(T)$
grd9 : $MSG\_DST(msg1) = T\_B\_DST(T)$

then

act1 : $init := init \cup \{T\}$
act2 : $CHAN1 := CHAN1 \cup \{msg1\}$
act3 : $CHAN1\_A(msg1) := T\_SRC(T)$
act4 : $CHAN1\_Na(msg1) := T$

end

Event *SEND2-0* $\widehat{=}$
refines *SEND2*

any

T
B
$msg2$
$msg1$
KBS

where

grd1 : $T \in TRANSACTION$
grd2 : $B \in AGENT$
grd3 : $T \notin end$
grd4 : $msg2 \in MSG$
grd5 : $msg2 \notin CHAN2$
grd6 : $msg2 \notin CHAN3$
grd7 : $msg2 \notin CHAN1$

grd8 : $MSG\_SRC(msg2) = B$
grd9 : $msg1 \in CHAN1$
grd10 : $MSG\_DST(msg1) = B$
grd11 : $KBS \in S\_KEY$
grd12 : $K1(KBS) = B$
grd13 : $K2(KBS) = S$
grd14 : $B = T\_B\_DST(T)$
grd15 : $CHAN1\_Na(msg1) = T$

then

act1 : $CHAN2 := CHAN2 \cup \{msg2\}$
act2 : $CHAN2\_B(msg2) := B$
act3 : $CHAN2\_A(msg2) := CHAN1\_A(msg1)$
act4 : $CHAN2\_Na(msg2) := CHAN1\_Na(msg1)$
act5 : $CHAN2\_Kbs(msg2) := KBS$

end

Event *STEP2-1* $\widehat{=}$

any

T
B
$msg2$
$msg1$
KBS

where

grd1 : $T \in TRANSACTION$
grd2 : $B \in AGENT$
grd3 : $msg2 \in MSG$
grd4 : $msg2 \notin CHAN2$
grd5 : $msg2 \notin CHAN3$
grd6 : $msg2 \notin CHAN1$
grd7 : $MSG\_SRC(msg2) = B$
grd8 : $msg1 \in CHAN1$
grd9 : $MSG\_DST(msg1) = B$
grd10 : $KBS \in S\_KEY$
grd11 : $K1(KBS) = B$
grd12 : $K2(KBS) = S$
grd13 : $B \neq T\_B\_DST(T)$
grd14 : $CHAN1\_Na(msg1) = T$

then

act1 : $CHAN2 := CHAN2 \cup \{msg2\}$
act2 : $CHAN2\_B(msg2) := B$
act3 : $CHAN2\_A(msg2) := CHAN1\_A(msg1)$
act4 : $CHAN2\_Na(msg2) := CHAN1\_Na(msg1)$
act5 : $CHAN2\_Kbs(msg2) := KBS$

end

Event *SEND3* $\widehat{=}$

any

$msg2$
$msg3$
KAS
KBS
KAB

where

grd1 : $msg3 \in MSG$
grd2 : $msg3 \notin CHAN2$
grd3 : $msg3 \notin CHAN3$
grd4 : $msg3 \notin CHAN1$
grd5 : $MSG\_SRC(msg3) = S$
grd6 : $msg2 \in CHAN2$
grd7 : $KAS \in S\_KEY$
grd8 : $K1(KAS) = CHAN2\_A(msg2)$
grd9 : $K2(KAS) = S$
grd10 : $KBS \in S\_KEY$
grd11 : $CHAN2\_Kbs(msg2) = KBS$
grd12 : $K1(KBS) = CHAN2\_B(msg2)$
grd13 : $K2(KBS) = S$
grd14 : $KAB \in S\_KEY$

then

act1 : $CHAN3 := CHAN3 \cup \{msg3\}$
act2 : $CHAN3\_B(msg3) := CHAN2\_B(msg2)$
act3 : $CHAN3\_Na(msg3) := CHAN2\_Na(msg2)$
act4 : $CHAN3\_Kas(msg3) := KAS$
act5 : $CHAN3\_Kab(msg3) := KAB$

end

Event *SEND4* $\widehat{=}$
refines *END*

any

T
$msg3$
KAS

where

grd1 : $T \in TRANSACTION$
grd2 : $T \in init$
grd3 : $msg3 \in CHAN3$
grd4 : $CHAN3\_B(msg3) = T\_B\_DST(T)$
grd5 : $CHAN3\_Na(msg3) = T$

grd6 : $KAS \in S\_KEY$
grd7 : $K1(KAS) = T\_SRC(T)$
grd8 : $K2(KAS) = S$
grd9 : $CHAN3\_Kas(msg3) = KAS$

then

act1 : $end := end \cup \{T\}$

end

Event $GENERATE2 \widehat{=}$

any

$msg2$
B
$frag$

where

grd1 : $B \in AGENT$
grd2 : $msg2 \in MSG$
grd3 : $msg2 \notin CHAN1$
grd4 : $msg2 \notin CHAN2$
grd5 : $msg2 \notin CHAN3$
grd6 : $MSG\_SRC(msg2) = I$
grd7 : $frag \in FRAG2$

then

act1 : $CHAN2 := CHAN2 \cup \{msg2\}$
act2 : $CHAN2\_B(msg2) := B$
act3 : $CHAN2\_A(msg2) := FRAG2\_A(frag)$
act4 : $CHAN2\_Na(msg2) := FRAG2\_Na(frag)$
act5 : $CHAN2\_Kbs(msg2) := FRAG2\_Kbs(frag)$

end

Event $GENERATE1 \widehat{=}$

any

$msg1$
A
T

where

grd1 : $T \in N\_MEM$
grd2 : $A \in AGENT$
grd3 : $msg1 \in MSG$
grd4 : $msg1 \notin CHAN1$
grd5 : $msg1 \notin CHAN2$
grd6 : $msg1 \notin CHAN3$
grd7 : $MSG\_SRC(msg1) = I$

then

act1 : $CHAN1 := CHAN1 \cup \{msg1\}$

act2 : $CHAN1\_A(msg1) := A$
act3 : $CHAN1\_Na(msg1) := T$

end

Event $GENERATE3 \widehat{=}$

any

$msg3$
$frag$

where

grd1 : $msg3 \in MSG$
grd2 : $msg3 \notin CHAN1$
grd3 : $msg3 \notin CHAN2$
grd4 : $msg3 \notin CHAN3$
grd5 : $MSG\_SRC(msg3) = I$
grd6 : $frag \in FRAG3$

then

act1 : $CHAN3 := CHAN3 \cup \{msg3\}$
act2 : $CHAN3\_B(msg3) := FRAG3\_B(frag)$
act3 : $CHAN3\_Na(msg3) := FRAG3\_Na(frag)$
act4 : $CHAN3\_Kas(msg3) := FRAG3\_Kas(frag)$
act5 : $CHAN3\_Kab(msg3) := FRAG3\_Kab(frag)$

end

Event $LISTEN1 \widehat{=}$

any

$msg1$

where

grd1 : $msg1 \in CHAN1$

then

act1 : $N\_MEM := N\_MEM \cup \{CHAN1\_Na(msg1)\}$

end

Event $LISTEN2 \widehat{=}$

any

$msg2$
$frag$

where

grd1 : $msg2 \in CHAN2$
grd2 : $frag \in MSG$
grd3 : $frag \notin FRAG2$

then

act1 : $FRAG2 := FRAG2 \cup \{frag\}$
act2 : $FRAG2\_A(frag) := CHAN2\_A(msg2)$
act3 : $FRAG2\_Na(frag) := CHAN2\_Na(msg2)$

act4 : $FRAG2\_Kbs(frag) := CHAN2\_Kbs(msg2)$

end

Event $LISTEN3 \widehat{=}$

any

$msg3$

$frag$

where

grd1 : $msg3 \in CHAN3$

grd2 : $frag \in MSG$

grd3 : $frag \notin FRAG3$

then

act1 : $FRAG3 := FRAG3 \cup \{frag\}$

act2 : $FRAG3\_B(frag) := CHAN3\_B(msg3)$

act3 : $FRAG3\_Na(frag) := CHAN3\_Na(msg3)$

act4 : $FRAG3\_Kas(frag) := CHAN3\_Kas(msg3)$

act5 : $FRAG3\_Kab(frag) := CHAN3\_Kab(msg3)$

end

Event $DECRYPT2 \widehat{=}$

any

$frag$

where

grd1 : $frag \in FRAG2$

grd2 : $FRAG2\_Kbs(frag) \in KEY\_MEM(I)$

then

act1 : $N\_MEM := N\_MEM \cup \{FRAG2\_Na(frag)\}$

end

Event $DECRYPT3 \widehat{=}$

any

$frag$

where

grd1 : $frag \in FRAG3$

grd2 : $FRAG3\_Kas(frag) \in KEY\_MEM(I)$

then

act1 : $N\_MEM := N\_MEM \cup \{FRAG3\_Na(frag)\}$

end

Event $GEN\_FRAG2 \widehat{=}$

any

$frag$

T

Kbs

A

where

grd1 : $frag \in MSG$
grd2 : $frag \notin FRAG2$
grd3 : $T \in N\_MEM$
grd4 : $Kbs \in KEY\_MEM(I)$
grd5 : $A \in AGENT$

then

act1 : $FRAG2 := FRAG2 \cup \{frag\}$
act2 : $FRAG2\_A(frag) := A$
act3 : $FRAG2\_Na(frag) := T$
act4 : $FRAG2\_Kbs(frag) := Kbs$

end

Event $GEN\_FRAG3 \widehat{=}$

any

$frag$
T
Kas
B
Kab

where

grd1 : $frag \in MSG$
grd2 : $frag \notin FRAG3$
grd3 : $T \in N\_MEM$
grd4 : $Kas \in KEY\_MEM(I)$
grd5 : $B \in AGENT$
grd6 : $Kab \in S\_KEY$

then

act1 : $FRAG3 := FRAG3 \cup \{frag\}$
act2 : $FRAG3\_B(frag) := B$
act3 : $FRAG3\_Na(frag) := T$
act4 : $FRAG3\_Kas(frag) := Kas$
act5 : $FRAG3\_Kab(frag) := Kab$

end

Event $ADD\_KEY \widehat{=}$

any

K

where

grd1 : $K \in S\_KEY$
grd2 : $I = K1(K)$

then

act1 : $KEY\_MEM(I) := KEY\_MEM(I) \cup \{K\}$

end

END

Bibliographie

[1] The AVISPA project. `http://avispa-project.org/`.

[2] The GNU privacy guard. `http://www.gnupg.org/`.

[3] The OpenSSL project. `http://www.openssl.org/`.

[4] *ISO. Information Technology-Security techniques-Key Management-Part 2 : Mechanisms using Symmetric Techniques ISO/IEC 11770-2.* 1996.

[5] *ISO. Information Technology-Security techniques-Entity Authentication-Part 2 : Mechanisms using Symmetric Encypherment Algorithms ISO/IEC 9798-2.* 2nd edition, 1999.

[6] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Trans. Program. Lang. Syst.*, 17(3) :507–535, 1995.

[7] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *TCS '00 : Proceedings of the International Conference IFIP on Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics*, pages 3–22, London, UK, 2000. Springer-Verlag.

[8] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, June 2003.

[9] J.-R. Abrial. Etude système : méthode et exemple. http ://www.atelierb.societe.com/documents.html.

[10] J.-R. Abrial. *The B-book : assigning programs to meanings.* Cambridge University Press, New York, NY, USA, 1996.

[11] J. R. Abrial. *Modeling in Event-B : System and Software Engineering.* Cambridge University Press., 2009.

[12] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, decomposition, and instantiation of discrete models : Application to Event-B. *Fundam. Inf.*, 77(1-2) :1–28, 2007.

[13] Tanvir Ahmed and Anand R. Tripathi. Static verification of security requirements in role based cscw systems. In *SACMAT '03 : Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 196–203, New York, NY, USA, 2003. ACM.

[14] R. J. R. Back. On correct refinement of programs. *Journal of Computer and System Sciences*, 23(1) :49–68, 1979.

[15] Ralph-Johan Back and Michael J. Butler. Fusion and simultaneous execution in the refinement calculus. *Acta Inf.*, 35(11) :921–949, 1998.

[16] R. K. Bauer, T. A. Berson, and R. J. Feiertag. A key distribution protocol using event markers. *ACM Trans. Comput. Syst.*, 1(3) :249–255, 1983.

[17] D. E. Bell and L. J. LaPadula. Secure computer systems : Mathematical foundations. Technical Report ESD-TR-278, Bedford, MA, 1973.

[18] Nazim Benaissa. Modelling attacker's knowledge for cascade cryptographic protocols. In *ABZ '08 : Proceedings of the 1st international conference on Abstract State Machines, B and Z*, pages 251–264, Berlin, Heidelberg, 2008. Springer-Verlag.

[19] Nazim Benaissa, Dominique Cansell, and Dominique Méry. Integration of security policy into system modeling. In Jacques Julliand and Olga Kouchnarenko, editors, *B*, volume 4355 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2007.

[20] Nazim Benaissa and Dominique Mery. Cryptographic protocols analysis in Event B. In *7th International Andrei Ershov Memorial Conference, PSI 2009*, Lecture Notes in Computer Science, pages 282–293, Novosibirsk, Russia, June 15-19 2009. Springer.

[21] Nazim Benaissa and Dominique Mery. Développement combiné et prouvé de systèmes transactionnels cryptologiques. In *Atelier Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL 2009)*, pages 121–136, Toulouse, France, 26-28 January 2009. Actes AFADL.

[22] Nazim Benaissa and Dominique Mery. Proof-based design of security protocols. In *International Computer Science Symposium in Russia, CSR 2010*, Lecture Notes in Computer Science. Springer, 2010.

[23] K. J. Biba. Integrity considerations for secure computer systems. Technical Report TR-3153, Bedford, MA, 1977.

[24] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In *CRYPTO '90 : Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, pages 2–21, London, UK, 1991. Springer-Verlag.

[25] Nikolaj Bjørner, Anca Browne, and Zohar Manna. Automatic generation of invariants and intermediate assertions. *Theor. Comput. Sci.*, 173(1) :49–87, 1997.

[26] Piero Bonatti, Sabrina De Capitani di Vimercati, and Pierangela Samarati. An algebra for composing access control policies. *ACM Trans. Inf. Syst. Secur.*, 5(1) :1–35, 2002.

[27] E. Borger and Robert F. Stark. *Abstract State Machines : A Method for High-Level System Design and Analysis.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[28] Colin. Boyd and Anish Mathuria. *Protocols for authentication and key establishment.* Springer, Berlin ; London, 2003.

[29] Chiara Braghin, Daniele Gorla, and Vladimiro Sassone. A distributed calculus for rôle-based access control. In *CSFW '04 : Proceedings of the 17th IEEE workshop on Computer Security Foundations*, page 48, Washington, DC, USA, 2004. IEEE Computer Society.

[30] D. F. C. Brewer and M. J. Nash. The chinese wall security policy. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 215–228, Oakland, CA, 1989.

[31] Achim D. Brucker and Burkhart Wolff. A case study of a formalized security architecture. *Electronic Notes in Theoretical Computer Science*, 80 :24 – 40, 2003. Eighth International Workshop on Formal Methods for Industrial Critical Systems (FMICS'03).

[32] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1) :18–36, 1990.

[33] J. R. Abrial C. Metayer and L. Voisin. Rodin deliverable 3.2. Event-B language. Technical report, School of Computing Science, Newcastle University, 2005.

[34] Dominique Cansell and Dominique Méry. *The event-B Modelling Method : Concepts and Case Studies*, pages 33–140. EATCS Textbook in Computer Science. Springer, 2007.

[35] K. Mani Chandy. *Parallel program design : a foundation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[36] E. M. Clarke, S. Jha, and W. Marrero. Verifying security protocols with Brutus. *ACM Trans. Softw. Eng. Methodol.*, 9(4) :443–487, 2000.

[37] ClearSy. *Atelier B, Manuel Utilisateur*, 2001.

[38] ClearSy. *Web site B4free set of tools for development of B models*, 2004.

[39] Véronique Cortier and Stéphanie Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1) :1–36, 2009.

[40] Jason Crampton. Administrative scope and role hierarchy operations. In *SACMAT '02 : Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 145–154, New York, NY, USA, 2002. ACM.

[41] Kriangsak Damchoom and Michael Butler. Applying event and machine decomposition to a flash-based filestore in Event-B. pages 134–152, 2009.

[42] Anupam Datta, Ante Derek, John C. Mitchell, and Arnab Roy. Protocol composition logic (PCL). *Electron. Notes Theor. Comput. Sci.*, 172 :311–358, 2007.

[43] Edsger W. Dijkstra and Carel S. Scholten. *Predicate calculus and program semantics*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[44] Danny Dolev and Andrew C. Yao. On the security of public key protocols. Technical report, Stanford, CA, USA, 1981.

[45] Michael Drouineaud, Maksym Bortin, Paolo Torrini, and Karsten Sohr. A first step towards formal verification of security policy properties for RBAC. In *QSIC '04 : Proceedings of the Quality Software, Fourth International Conference*, pages 60–67, Washington, DC, USA, 2004. IEEE Computer Society.

[46] D. F. Ferraiolo and R. Kuhn. Role-based access control. In *Proceedings of the 15th NIST-NSA National Computer Security Conference*, pages 215–228, Baltimore, MD, 1992.

[47] Brian Gladman, Carl Ellison, and Nicholas Bohm. Digital signatures, certificates and electronic commerce, 1999.

[48] F. Glenn. RBAC in UNIX administration. In *RBAC '99 : Proceedings of the fourth ACM workshop on Role-based access control*, pages 95–101, New York, NY, USA, 1999. ACM.

[49] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society, 1990.

[50] Holger Grandy, Markus Bischof, Kurt Stenzel, Gerhard Schellhorn, and Wolfgang Reif. Verification of mondex electronic purses with kiv : From a security protocol to verified code. In *FM*, pages 165–180, 2008.

[51] Joshua D. Guttman and F. Javier Thayer. Protocol independence through disjoint encryption. In *CSFW '00 : Proceedings of the 13th IEEE workshop on Computer Security Foundations*, page 24, Washington, DC, USA, 2000. IEEE Computer Society.

[52] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8) :461–471, 1976.

[53] J. Chu J. Zao, H. Wee and D. Jackson. RBAC schema verification using lightweight formal model and constraint analysis. Technical monograph, MIT, Cambridge, 2002.

[54] D. Jackson. Alloy 3.0 reference manual. `http://alloy.mit.edu/reference-manual.pdf`,.

[55] Radha Jagadeesan, Alan Jeffrey, Corin Pitcher, and James Riely. Lambda-RBAC : Programming with role-based access control. *CoRR*, abs/0712.1205, 2007.

[56] Cliff B. Jones. Specification and design of (parallel) programs. In *IFIP Congress*, pages 321–332, 1983.

[57] J. Kohl and C. Neuman. *The Kerberos Network Authentication Service (V5)*. RFC Editor, United States, 1993.

[58] Leslie Lamport, John Matthews, Mark Tuttle, and Yuan Yu. Specifying and verifying systems with tla+. In *EW 10 : Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 45–48, New York, NY, USA, 2002. ACM.

[59] Butler W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8(1) :18–24, 1974.

[60] Michael Leuschel and Michael J. Butler. Prob : an automated analysis toolset for the b method. *STTT*, 10(2) :185–203, 2008.

[61] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In *TACAs '96 : Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 147–166, London, UK, 1996. Springer-Verlag.

[62] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems : safety*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[63] John McLean. The specification and modeling of computer security. *Computer*, 23(1) :9–16, 1990.

[64] Catherine Meadows. The nrl protocol analyzer : An overview. *The Journal of Logic Programming*, 26(2) :113 – 131, 1996.

[65] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murϕ. In *SP '97 : Proceedings of the 1997 IEEE Symposium on Security and Privacy*, page 141, Washington, DC, USA, 1997. IEEE Computer Society.

[66] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12) :993–999, 1978.

[67] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Comput. Secur.*, 6(1-2) :85–128, 1998.

[68] Michael Poppleton. The composition of Event-B models. In *ABZ '08 : Proceedings of the 1st international conference on Abstract State Machines, B and Z*, pages 209–222, Berlin, Heidelberg, 2008. Springer-Verlag.

[69] M-L. Potet. Spécifications et développements formels : Etude des aspects compositionnels dans la méthode b. Technical report, Institut National Polytechnique de Grenoble, 2002.

[70] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2) :120–126, 1978.

[71] Thomas K. Roshan. Team-based access control (TMAC) : a primitive for applying role-based access controls in collaborative environments. In *RBAC '97 : Proceedings of the second ACM workshop on Role-based access control*, pages 13–19, New York, NY, USA, 1997. ACM.

[72] R. Sandhu and G. Ahn. Decentralized group hierarchies in unix : An experiment and lessons learned. In *In Proceedings of 21st NIST-NCSC National Information Systems Security Conference*, pages 486–502, 1998.

[73] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1) :105–135, 1999.

[74] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The nist model for role-based access control : towards a unified standard. In *RBAC '00 : Proceedings of the fifth ACM workshop on Role-based access control*, pages 47–63, New York, NY, USA, 2000. ACM.

[75] Ravi Sandhu and Qamar Munawer. How to do discretionary access control using roles. In *RBAC '98 : Proceedings of the third ACM workshop on Role-based access control*, pages 47–54, New York, NY, USA, 1998. ACM.

[76] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2) :38–47, 1996.

[77] J. M. Spivey. *The Z notation : a reference manual.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[78] Susan Stepney, David Cooper, and Jim Woodcock. An electronic purse : Specification, refinement, and proof. Technical monograph PRG-126, Oxford University Computing Laboratory, July 2000.

[79] Paul F. Syverson. A taxonomy of replay attacks. In *CSFW*, pages 187–191, 1994.

[80] Paul F. Syverson and Paul C. Van Oorschot. On unifying some cryptographic protocol logics. In *SP '94 : Proceedings of the 1994 IEEE Symposium on Security and Privacy*, page 14, Washington, DC, USA, 1994. IEEE Computer Society.

[81] Roshan K. Thomas. TMAC : A primitive for applying RBAC in collaborative environment. In *2nd ACM, Workshop on RBAC*, pages pages 13–19, November 1997.

[82] Andrew C. Yao. Theory and application of trapdoor functions. In *SFCS '82 : Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Washington, DC, USA, 1982. IEEE Computer Society.

[83] Chunyang Yuan, Yeping He, Jianbo He, and Zhouyi Zhou. A verifiable formal specification for RBAC model with constraints of separation of duty. In Helger Lipmaa, Moti Yung, and Dongdai Lin, editors, *Inscrypt*, volume 4318 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 2006.