

Implantation d'une politique de contrôle d'accès discrétionnaire avec Focal

Andréa-Diane Kadja et Florian Brecht

8 octobre 2007

Encadrants : Lionel Habib
Mathieu Jaume
Charles Morisset

Table des matières

1	Politiques de sécurité	3
2	Contrôle d'accès	4
2.1	Généralités sur le contrôle d'accès	4
2.2	Sous Unix	4
2.3	ACL et Capabilities	5
2.3.1	Les ACL	5
2.3.2	Les Capabilities	5
3	Déroulement du projet	6
3.1	Analyse du problème	6
3.2	Objectifs	6
3.3	Méthode	6
4	Focal	6
4.1	Généralités	6
4.2	Syntaxe	7
5	Modèle général	7
6	Modèle discrétionnaire	11
6.1	Définition et implantation du modèle	11
6.2	$\mathbb{P}[\]$ -correction de l'implantation	13
6.3	$\mathbb{P}[\]$ -complétude de l'implantation	13
6.4	\mathcal{R} -correction	14
6.5	\mathcal{W} -conformité	15
7	Programme Focal	16
7.1	Structure	16
7.2	Exécution	16
7.2.1	ACL et Capabilities	16
7.2.2	ACL	18
7.2.3	Capabilities	19
7.3	Preuves Zenon	20
8	Conclusion	22
	Références	24

Résumé

L'objectif du travail présenté est d'utiliser un cadre formel permettant la spécification et la définition de politiques de sécurité, pour spécifier une politique de contrôle d'accès du type Unix dans l'environnement de développement Focal. Nous définissons dans un premier temps ce qu'est une politique de sécurité, et en quoi consiste le contrôle d'accès, puis nous introduisons les notations et concepts qui nous permettent de définir formellement une politique discrétionnaire. Dans un deuxième temps, nous présentons le développement formel implantant cette politique discrétionnaire.

1 Politiques de sécurité

La sécurité informatique est un terme général couvrant un vaste domaine du monde de l'informatique et du traitement de l'information, qui a beaucoup évolué et est devenu une préoccupation majeure au cours des dernières années. En effet, un nombre croissant de personnes utilisent leur ordinateur personnel pour accéder à des ressources sur internet et de plus en plus d'industries dépendent de systèmes informatiques et de réseaux pour effectuer des transactions commerciales. La sûreté des systèmes informatiques est un domaine complémentaire de la sécurité qui consiste d'une part à mettre en oeuvre des méthodes et des moyens pour éviter les défaillances "naturelles" résultant d'un phénomène physique ou d'une erreur de conception, et d'autre part à se protéger des défaillances résultant d'une action intentionnelle malveillante visant des systèmes d'information. On peut dire que la sûreté garantit que, sous certaines conditions données, la probabilité qu'une erreur d'exécution se produise est inférieure à un seuil donné (sûreté de fonctionnement) et que l'utilisateur ne pourra pas exécuter une action non prévue par le programme (sûreté de programmation).

Toutefois ces moyens ne sont mis en oeuvre qu'une fois qu'est déterminé ce qui doit être protégé et contre quelles attaques cette protection est nécessaire. C'est l'objet d'une politique de sécurité. La sûreté est donc un moyen de s'assurer qu'une politique de sécurité sera respectée. Même si un programme est sûr, il permettra à un utilisateur d'exploiter les failles d'une politique de sécurité mal définie.

Selon [5], définir une politique de sécurité consiste à

- Identifier les besoins en termes de sécurité, c'est à dire les risques informatiques pesant sur le système et leurs éventuelles conséquences.
- Elaborer des règles et des procédures à mettre en oeuvre pour les risques identifiés.

Une politique de sécurité est indépendante de son implantation : c'est une spécification de besoins.

2 Contrôle d'accès

Dans cette section, nous présentons ce qu'est le contrôle d'accès, et donnons des exemples.

2.1 Généralités sur le contrôle d'accès

Une politique de sécurité a pour but, entre autres, la protection des informations dans un système informatique, et est généralement définie [4] dans le but d'empêcher un utilisateur

- de détruire ou de modifier les données d'un autre utilisateur ;
- de lire ou de copier les données d'un autre utilisateur sans son autorisation.

C'est l'objet du contrôle d'accès. Le contrôle d'accès est n'importe quel mécanisme par lequel le système autorise ou interdit des accès effectués par des sujets (utilisateurs, processus, ...) sur des objets (fichiers, programmes, ...). L'une des techniques pour mettre en oeuvre une politique de contrôle d'accès consiste à définir un moniteur de référence. Un moniteur de référence est un programme chargé d'appliquer la politique de sécurité. Les trois propriétés classiques qui peuvent être garanties par une politique de contrôle d'accès sont la confidentialité, qui garantit que seules les personnes autorisées peuvent lire les données, l'intégrité, qui garantit que seules les personnes autorisées peuvent modifier les données, et la disponibilité, qui garantit que les personnes autorisées peuvent accéder aux données. Dans ce document, nous nous intéressons principalement aux politiques de confidentialité et d'intégrité. On distingue les politiques selon les mécanismes mis en oeuvre pour gérer les autorisations pour les accès. On parle de modèle discrétionnaire lorsque les décisions sont déléguées à des personnes responsables pour leurs données. En pratique, le propriétaire d'un objet dispose de tous les droits sur cet objet, et décidera qui a accès à la donnée en lecture ou en écriture.

Dans un modèle discrétionnaire, le bon fonctionnement du système repose donc sur la confiance : la décision d'accorder ou non un droit d'accès est prise par un sujet. Au contraire, dans un modèle obligatoire (ou mandataire), les sujets ne peuvent modifier les droits d'accès aux objets que sous certaines conditions qui sont définies par la politique de sécurité.

2.2 Sous Unix

La politique de sécurité du système UNIX est fondée sur le principe que chaque objet admet un identifiant, un propriétaire, un groupe et un ensemble de droits d'accès (en lecture, en écriture, en exécution) répartis en trois groupes :

- les droits du propriétaire

- les droits du groupe auquel appartient l'objet
- les droits des autres utilisateurs.

Etant donné que la gestion des droits est déléguée aux propriétaires des fichiers, il s'agit d'une politique discrétionnaire.

2.3 ACL et Capabilities

Les listes de contrôle d'accès (*Access Control List*) et les Capabilities sont deux méthodes permettant une gestion plus fine des droits d'accès aux fichiers que la méthode employée par les systèmes Unix. En effet, sous Unix, seuls les droits d'accès pour trois catégories d'utilisateurs sont définis : le propriétaire, le groupe (ensemble de sujets), et tous les autres sujets. Les ACL et Capabilities permettent l'énumération exhaustive des droits d'accès pour chaque sujet et chaque objet.

2.3.1 Les ACL

Avec les ACL, une information est rattachée à chaque objet. Il s'agit d'une liste qui spécifie pour chaque sujet les droits d'accès à cet objet. Voici un petit exemple :

Exemple On considère l'ensemble d'objets {toto, titi, tata}, l'ensemble de sujets {diane, florian}, ainsi que les droits d'accès {lecture, écriture}.

objet	liste
toto	[(diane,lecture) ; (florian,écriture) ; (florian,lecture)]
titi	[(diane,lecture) ; (florian,lecture)]
tata	[(diane,écriture)]

Ainsi, l'utilisateur diane a l'autorisation d'accéder en écriture à l'objet tata.

2.3.2 Les Capabilities

Les Capabilities fonctionnent sur le même principe que les ACL, sauf que les listes de droits d'accès sont rattachées aux sujets au lieu des objets. Les Capabilities sont l'autre face de la médaille du discrétionnaire. Chaque sujet a conscience de sa propre existence et de son interaction avec le monde.

Exemple En reprenant les mêmes sujets, objets et droits d'accès que ci-dessus, voici l'exemple repris en Capabilities :

sujet	liste
diane	[(toto,lecture) ; (titi,lecture) ; (tata,écriture)]
florian	[(toto,lecture) ; (titi,lecture) ; (toto,écriture)]

3 Déroulement du projet

Dans ce projet, nous nous intéressons plus particulièrement aux politiques de contrôle d'accès discrétionnaires.

3.1 Analyse du problème

La politique de contrôle d'accès que nous allons définir doit être non ambiguë ; un manque de précision dans la définition d'une telle politique peut induire des failles qui peuvent permettre de violer cette politique.

Nous devons prouver qu'elle garantit le respect de certaines des propriétés classiques des systèmes informatiques sous certaines conditions. L'utilisateur d'une politique de sécurité, encore plus que pour une autre application, doit avoir confiance en ce programme. Cet utilisateur doit être sûr que le programme implantant la politique de sécurité fonctionne correctement (sûreté) et garantit les propriétés de sécurité demandées.

Il serait donc inutile de définir une politique de contrôle d'accès si on ne prouve pas des propriétés portant sur cette politique.

3.2 Objectifs

Nous allons implanter en Focal une politique de contrôle d'accès discrétionnaire (i.e. à la Unix). Cette implantation doit respecter la spécification que nous définissons. Nous prouvons que l'implantation est correcte par rapport au modèle.

3.3 Méthode

Pour atteindre les objectifs, nous décrivons la politique avec une approche formelle. Les méthodes formelles sont des techniques permettant de raisonner rigoureusement, à l'aide de logique mathématique, sur des programmes informatiques, afin de démontrer leur validité par rapport à une certaine spécification (une description mathématique formelle des propriétés souhaitées). Ces techniques sont basées sur la sémantique des programmes, c'est-à-dire sur des modèles de calcul décrivant les exécutions effectives de ces programmes dans tous les environnements possibles.

4 Focal

Dans cette section, nous présentons brièvement l'environnement Focal.

4.1 Généralités

Le but du projet Focal [6] est de construire un environnement dédié au développement de programmes certifiés. Il s'agit d'un langage permettant de

spécifier, développer et vérifier une application par une approche formelle. L'une des caractéristiques importantes de Focal est que l'on peut définir des propriétés et les prouver. Focal est un langage de programmation fonctionnel avec des traits objets, comme l'héritage multiple, la liaison tardive, la paramétrisation. Les composantes sont :

- Les espèces, qui peuvent être vues comme des ensembles de méthodes. Ce sont les unités de programmation (que l'on peut comparer aux classes de Java). Dans une espèce, les méthodes peuvent être déclarées ou définies.
- Les collections sont des espèces complètement définies (toutes les méthodes doivent être définies et tous les paramètres instanciés).
- Le type de représentation est le type de données manipulées par les fonctions dans les espèces, qui doivent avoir un seul type support.
- Une interface est une liste de toutes les méthodes, définies ou déclarées, d'une espèce. Ainsi l'utilisateur final qui veut savoir quelles fonctions utiliser sans se soucier des détails de l'implémentation, se sert des interfaces.
- Les entités sont les objets manipulés (objets mathématiques, paires, ...) qui sont accessibles via des fonctions appropriées.

Programmer en Focal consiste à spécifier le problème en introduisant des propriétés et des déclarations dans un premier temps, puis à introduire pas à pas les définitions, la représentation en utilisant l'héritage, et enfin les preuves (c'est le procédé de raffinement).

4.2 Syntaxe

- Les commentaires se situent entre `(*` et `*)` .
- Les fonctions sont introduites par `sig` lorsqu'elles sont déclarées et par `let [rec]` lorsqu'elles sont définies.
- Les propriétés sont introduites par `property` lorsqu'elles sont déclarées et par `theorem` lorsqu'on donne la preuve.
- Le type support est introduit par `rep`.
- Si `species` est le nom d'espèce sur laquelle on veut appeler une fonction `fun`, l'appel de `fun` sur l'espèce `species` s'effectue de la manière suivante : `species !fun`

5 Modèle général

Nous reprenons le cadre formel introduit dans [3] dans lequel il est possible de spécifier puis implémenter une politique discrétionnaire, et vérifier que l'implantation respecte la spécification.

On introduit les notions suivantes :

- \mathcal{S} est l'ensemble des sujets (les utilisateurs),
- \mathcal{O} est l'ensemble des objets (les données),

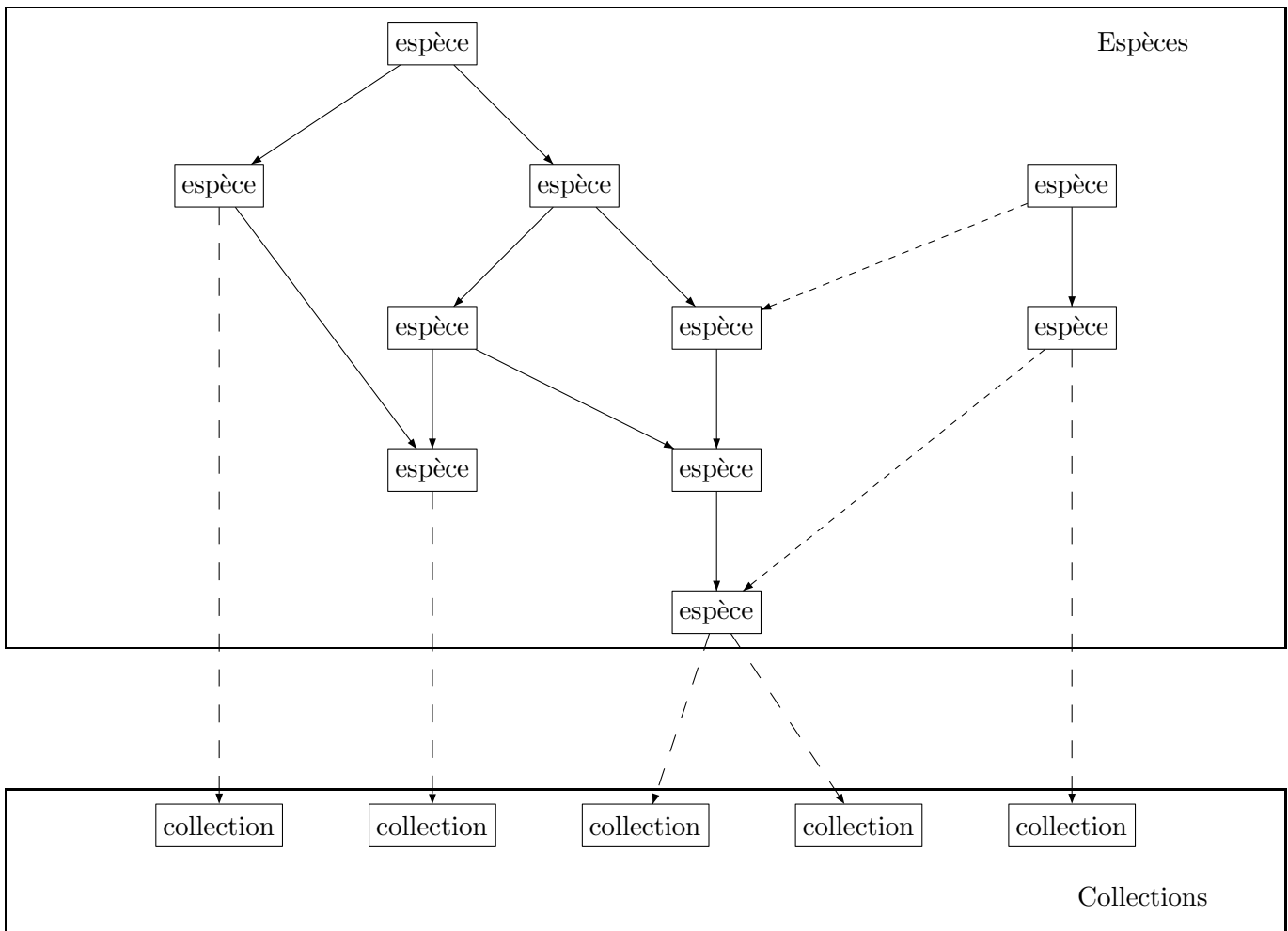


FIG. 1 – Structure générique d'un programme FOCAL

– \mathcal{A} est l'ensemble des modes d'accès aux objets (r : *read*, w : *write*).

On note \mathbb{A} l'ensemble de tous les accès : $\mathbb{A} = \mathcal{S} \times \mathcal{O} \times \mathcal{A}$.

Soient $s \in \mathcal{S}, o \in \mathcal{O}, x \in \mathcal{A}$, le triplet $a = (s, o, x) \in \mathbb{A}$ représente l'accès du sujet s à la ressource o avec le mode d'accès x .

On définit les états, qui sont des descriptions du système à un instant donné. On note Σ l'ensemble de tous les états possibles. La fonction $\Lambda : \Sigma \rightarrow \mathcal{P}(\mathbb{A})$ donne les accès courants d'un état. Si l'on note \mathbf{SF} l'ensemble des fonctions de sécurité d'un état, la fonction $\Upsilon : \Sigma \rightarrow \mathbf{SF}$ permet de les obtenir. Le prédicat $\Omega : \Sigma \rightarrow \mathbb{B}$ détermine si un état est sûr. L'ensemble des états sûrs $\{\sigma \in \Sigma \mid \Omega(\sigma)\}$ est noté $\Sigma_{|\Omega}$.

Les informations de sécurité, associées aux sujets et/ou aux objets, d'une politique de contrôle d'accès sont regroupées dans un paramètre de sécurité noté ρ .

Définition 1 (Politique de contrôle d'accès) *Une politique de contrôle d'accès est définie par l'ensemble de ses sujets \mathcal{S} , objets \mathcal{O} , modes d'accès \mathcal{A} , états Σ , ainsi que par le prédicat Ω , qui détermine si un état est sûr. Le paramètre de sécurité ρ permet de fournir des informations supplémentaires. On note $\mathbb{P}[\rho]$ la politique de contrôle d'accès. $\mathbb{P}[\rho] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma, \Omega)$*

Une politique de sécurité caractérise les états sûrs. Pour faire évoluer le système (modifier les accès courants), les utilisateurs doivent soumettre des requêtes. On définit donc \mathcal{R} , l'ensemble des requêtes, de la forme $\langle +, s, o, x \rangle$ ou $\langle -, s, o, x \rangle$, où $s \in \mathcal{S}$, $o \in \mathcal{O}$, et $x \in \mathcal{A}$. On appelle sémantique des requêtes la spécification du sens des requêtes. Nous décrivons en partie cette sémantique en caractérisant les propriétés que doit vérifier un état après application réussie d'une requête.

Définition 2 *La relation $\llbracket \mathcal{R} \rrbracket_{\Sigma}$ est définie de la façon suivante :*

$$\begin{aligned} \langle +, s, o, x \rangle, \sigma \in \llbracket \mathcal{R} \rrbracket_{\Sigma} &\Leftrightarrow (s, o, x) \in \Lambda(\sigma) \\ \langle -, s, o, x \rangle, \sigma \in \llbracket \mathcal{R} \rrbracket_{\Sigma} &\Leftrightarrow (s, o, x) \notin \Lambda(\sigma) \end{aligned}$$

La sémantique des requêtes est aussi décrite par un partitionnement de l'ensemble des requêtes. Ce partitionnement repose sur la notion d'accès potentiels associés à un état.

Définition 3 (Accès potentiels d'un état) *On définit la fonction $\mathcal{W} : \Sigma \rightarrow \wp(\wp(\mathbb{A}))$ qui permet de caractériser l'ensemble des accès qui peuvent être ajoutés de manière « sûre » dans un état sans changer les fonctions de sécurité.*

$$\mathcal{W}(\sigma) = \left\{ A \subseteq \wp(\mathbb{A}) \mid \forall \sigma' \in \Sigma \right. \\ \left. (\Upsilon(\sigma') = \Upsilon(\sigma) \wedge \Lambda(\sigma') = \Lambda(\sigma) \cup A) \Rightarrow \Omega(\sigma') \right\}$$

Définition 4 (Partitionnement des requêtes) *Le partitionnement des requêtes suivant est introduit :*

$$\mathcal{R} = \mathcal{R}^{\otimes} \cup \mathcal{R}^{\ominus} \cup \mathcal{R}^{\odot}$$

L'ensemble \mathcal{R}^{\otimes} (resp. \mathcal{R}^{\ominus}) contient les requêtes élargissant (resp. rétrécissant), au sens de l'inclusion, les accès potentiels tandis que \mathcal{R}^{\odot} contient les autres requêtes.

Définition 5 (Modèle de contrôle d'accès) *Un modèle de contrôle d'accès, noté $\mathbb{M}[\rho]$, est formé d'une politique de sécurité et d'une sémantique des requêtes afin de pouvoir modifier l'état du système. $\mathbb{M}[\rho] = (\mathbb{P}[\rho], \llbracket \mathcal{R} \rrbracket_{\Sigma})$*

Implanter un modèle de contrôle d'accès $\mathbb{M}[\rho]$ consiste à définir un ensemble Σ_I d'états initiaux et une fonction de transition τ qui permet de passer d'un état à un autre état avec une requête dans \mathcal{R} . C'est la fonction de transition, notée τ , qui gère les requêtes.

Définition 6 (Implantation) *L'implantation d'un modèle de contrôle d'accès $\mathbb{M}[\rho] = (\mathbb{P}[\rho], \llbracket \mathcal{R} \rrbracket_{\Sigma})$ basé sur une politique $\mathbb{P}[\rho] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma, \Omega)$ est une paire (τ, Σ_I) où Σ_I est un ensemble d'états initiaux et où $\tau : \mathcal{R} \times \Sigma \rightarrow \mathcal{D} \times \Sigma$ est une fonction de transition avec $\mathcal{D} = \{\text{yes}, \text{no}\}$.*

Dans la suite, on note $\Gamma_{\tau}^n(E)$ (resp. $\Gamma_{\tau}(E)$) l'ensemble des états d'un système qui sont accessibles par n applications successives (resp. par un nombre fini d'applications) de la fonction τ à partir d'un état appartenant à l'ensemble E .

On introduit à présent les principales propriétés d'une implantation.

Définition 7 ($\mathbb{P}[\rho]$ -correction) *Une implantation (τ, Σ_I) est $\mathbb{P}[\rho]$ -correcte si et seulement si tout état accessible à partir d'un état initial est sûr :*

$$\mathbb{P}[\rho] \vdash (\tau, \Sigma_I) \Leftrightarrow \Gamma_{\tau}(\Sigma_I) \subseteq \Sigma_{|\Omega}$$

Lemme 1 *L'implantation (τ, Σ_I) d'une politique $\mathbb{P}[\rho] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma, \Omega)$ est $\mathbb{P}[\rho]$ -correcte si*

$$\begin{aligned} & \Sigma_I \subseteq \Sigma_{|\Omega} \\ \wedge \quad & \forall \sigma, \sigma' \in \Sigma \quad \forall R \in \mathcal{R} \quad \forall d \in \mathcal{D} \quad (\Omega(\sigma) \wedge \tau(R, \sigma) = (d, \sigma')) \Rightarrow \Omega(\sigma') \end{aligned}$$

PREUVE.

On raisonne par induction sur le nombre n de transitions qu'on effectue.

$$\forall n \in \mathbb{N} \quad \Gamma_{\tau}^n(\Sigma_I) \subseteq \Sigma_{|\Omega}$$

1. si $n = 0$, on a que $\forall \sigma \in \Sigma_I \quad \Gamma_{\tau}^0(\sigma) = \sigma$ et donc $\sigma \in \Sigma_I \subseteq \Sigma_{|\Omega}$.

2. Supposons la propriété vraie au rang n et montrons-la au rang $n + 1$. Soit σ_0 un état initial et σ_{n+1} un état tel que $\sigma_{n+1} \in \Gamma_\tau^{n+1}(\sigma)$. Il existe un état σ_n appartenant à $\Gamma_\tau^n(\sigma_0)$ tel que pour une requête R et une décision d , $\tau(r, \sigma_n) = (d, \sigma_{n+1})$ et par hypothèse d'induction $\Omega(\sigma_n)$ est vrai. De plus, par hypothèse, pour toute requête R et toute décision d on a $(\Omega(\sigma) \wedge \tau(R, \sigma) = (d, \sigma')) \Rightarrow \Omega(\sigma')$. On en déduit que $\Omega(\sigma_{n+1})$ est vrai, ce qui nous permet de conclure. ◀

Définition 8 ($\mathbb{P}[\rho]$ -complétude) *L'implantation (τ, Σ_I) est dite $\mathbb{P}[\rho]$ -complète ssi tout état sûr est atteignable :*

$$\mathbb{P}[\rho] \models (\tau, \Sigma_I) \Leftrightarrow \Sigma_{|\Omega} \subseteq \Gamma_\tau(\Sigma_I)$$

Définition 9 (\mathcal{R} -correction) *Une fonction de transition τ est \mathcal{R} -correcte par rapport à $\|\mathcal{R}\|_\Sigma$ et on note $\|\mathcal{R}\|_\Sigma \vdash \tau$ si et seulement si :*

$$\forall \sigma_1, \sigma_2 \in \Sigma \quad \forall R \in \mathcal{R} \quad \tau(R, \sigma_1) = (\text{yes}, \sigma_2) \Rightarrow (R, \sigma_2) \in \|\mathcal{R}\|_\Sigma$$

Définition 10 (\mathcal{W} -conformité) *la fonction de transition τ est dite \mathcal{W} -conforme si et seulement si :*

$$\begin{aligned} & \forall \sigma_1, \sigma_2 \in \Sigma \quad \forall d \in \mathcal{D} \quad \forall R \in \mathcal{R} \\ & \tau(R, \sigma_1) = (d, \sigma_2) \Rightarrow \\ & \left(\begin{array}{l} d = \text{yes} \Rightarrow \left(\begin{array}{l} R \in \mathcal{R}^\otimes \Rightarrow \mathcal{W}(\sigma_1) \subseteq \mathcal{W}(\sigma_2) \\ \wedge \quad R \in \mathcal{R}^\otimes \Rightarrow \mathcal{W}(\sigma_2) \subseteq \mathcal{W}(\sigma_1) \end{array} \right) \\ \wedge \quad d = \text{no} \Rightarrow \mathcal{W}(\sigma_2) \subseteq \mathcal{W}(\sigma_1) \end{array} \right) \end{aligned}$$

Définition 11 ($\mathbb{M}[\rho]$ -correction) *Etant donné un modèle de contrôle d'accès $\mathbb{M}[\rho] = (\mathbb{P}[\rho], \|\mathcal{R}\|_\Sigma)$, l'implantation (τ, Σ_I) est dite $\mathbb{M}[\rho]$ -correcte si et seulement si elle est $\mathbb{P}[\rho]$ -correcte et que τ est \mathcal{R} -correcte et \mathcal{W} -conforme. Dans ce cas, on écrit $\mathbb{M}[\rho] \vdash (\tau, \Sigma_I)$.*

6 Modèle discrétionnaire

6.1 Définition et implantation du modèle

L'ensemble SF des fonctions de sécurité d'un état se résume à une unique fonction de sécurité, notée f_d . La fonction de sécurité $f_d : \mathbb{A} \rightarrow \mathbb{B}$ qui prend en entrée un accès, indique si cet accès est "légal". Ainsi, si $f_d((s, o, x))$ est vrai, alors le sujet s a bien l'autorisation d'accéder à l'objet o sous le mode d'accès x donné. On définit l'ensemble des états Σ_d comme l'ensemble des couples (m, f_d) où m est un ensemble d'accès courants. Dans la suite, nous définissons le prédicat de sécurité Ω_d , ainsi que la fonction de transition τ_d dont nous prouvons la correction.

Le prédicat Ω_d de notre politique de sécurité est défini de la façon suivante :

$$\Omega_d(\sigma) \Leftrightarrow (\forall a \in \mathbb{A}, a \in \Lambda(\sigma) \Rightarrow f_d(a))$$

L'ensemble des états sans accès courants $\{\sigma \in \Sigma \mid \Lambda(\sigma) = \emptyset\}$ est noté Σ_\emptyset . D'après la définition de Ω_d , $\Sigma_\emptyset \subseteq \Sigma_{|\Omega_d}$. L'implantation en Focal de Ω_d correspond à :

```
letprop omega (st in self) =
  all s1 in s, all o1 in o, all m1 in m,
  (s_a!est_element(a!create(s1,o1,m1), !lambda (st))
   -> !fd(a!create(s1,o1,m1)));
```

Le paramètre de sécurité ρ_d de la politique discrétionnaire que nous définissons étant vide, nous notons $\mathbb{P}[]$ (au lieu de $\mathbb{P}[\rho_d]$) cette politique discrétionnaire.

τ_d est défini de la façon suivante :

$$\tau_d(R, \sigma) = \left\{ \begin{array}{ll} (yes, \sigma \setminus (s, o, x)) & \text{si } R = \langle -, (s, o, x) \rangle \\ (yes, \sigma \cup \{(s, o, x)\}) & \text{si } R = \langle +, (s, o, x) \rangle \wedge f_d((s, o, x)) \\ (no, \sigma) & \text{si } R = \langle +, (s, o, x) \rangle \wedge \neg f_d((s, o, x)) \end{array} \right\}$$

L'implantation en Focal de cette fonction de transition est donnée ci-dessous.

```
let tau(r1 in r, st in self) =
  if r!is_get(r1)
  then
    (if !fd(a!create (r!get_s(r1),
                     r!get_o(r1),
                     r!get_m(r1)))
    then
      (d!yes, !add(st, a!create(r!get_s(r1),
                              r!get_o(r1),
                              r!get_m(r1))))
    else (d!no, st))
    else (* r!is_rel(r1) *)
    (d!yes, !del(st, a!create(r!get_s(r1),
                              r!get_o(r1),
                              r!get_m(r1)))));
```

On se donne l'ensemble des états initiaux $\Sigma_{I_d} = \{\sigma \in \Sigma \mid \Lambda(\sigma) = \emptyset\}$. On a $\Sigma_{I_d} = \Sigma_\emptyset$.

Le partitionnement des requêtes est le suivant :

$$\mathcal{R} = \underbrace{\{ \langle +, s, o, x \rangle \}}_{\mathcal{R}^\ominus} \cup \underbrace{\{ \langle -, s, o, x \rangle \}}_{\mathcal{R}^\ominus} \cup \underbrace{\emptyset}_{\mathcal{R}^\circ} \quad (1)$$

6.2 $\mathbb{P}[\]$ -correction de l'implantation

Nous prouvons que notre implantation (τ_d, Σ_{I_d}) , est $\mathbb{P}[\]$ -correcte, c'est-à-dire que tous les états que l'on peut atteindre avec la fonction de transition sont sûrs. Pour cela, nous commençons par montrer la proposition suivante :

Proposition 1 $\forall \sigma, \sigma' \in \Sigma \forall r \in \mathcal{R} \forall d \in \mathcal{D} \quad (\Omega_d(\sigma) \wedge \tau_d(r, \sigma) = (d, \sigma')) \Rightarrow \Omega_d(\sigma')$

PREUVE.

$\forall \sigma_1 \in \Sigma$ t.q. $\Omega_d(\sigma_1)$, $\forall r \in \mathcal{R} \forall d \in \mathcal{D}$

1. si $r = \langle +, s, o, x \rangle$
 - (a) si $\tau_d(r, \sigma_1) = (yes, \sigma_2)$ alors par définition de τ_d , $\sigma_2 = (\Lambda(\sigma_1) \cup \{(s, o, x)\}, f_d)$. Par hypothèse $\Omega_d(\sigma_1)$ est vrai, donc $\forall (s', o', x') \in \Lambda(\sigma_1)$, $f_d((s', o', x'))$ est vrai. De plus, par définition de τ_d , on a $f_d((s, o, x))$. On peut donc en conclure que $\forall (s'', o'', x'') \in \Lambda(\sigma_1) \cup \{(s, o, x)\}$, $f_d((s'', o'', x''))$ est vrai, et donc $\Omega_d(\sigma_2)$ est vrai.
 - (b) si $\tau_d(r, \sigma_1) = (no, \sigma_2)$ alors $\sigma_1 = \sigma_2$ par définition de τ_d et comme on a $\Omega_d(\sigma_1)$ par hypothèse, on a aussi $\Omega_d(\sigma_2)$.
2. si $r = \langle -, s, o, x \rangle$ alors $\tau(r, \sigma_1) = (yes, \sigma_2)$ avec $\sigma_2 = (\Lambda(\sigma_1) \setminus \{(s, o, x)\}, f_d)$. On a $\forall a \in \Lambda(\sigma_2)$, $a \in \Lambda(\sigma_1)$ avec $f_d(a)$ car $\Omega_d(\sigma_1)$ est vrai par hypothèse. Donc, par définition de Ω_d , on a aussi $\Omega_d(\sigma_2)$.

◀

En Focal, cette propriété se code de la manière suivante :

```
letprop secure(tau in r -> self -> (d * self)) =
  all st1 st2 in self, all r1 in r, all d1 in d,
  tau(r1, st1) = (d1, st2) ->
  !omega(st1) -> !omega(st2);
```

Nous avons montré que $\forall \sigma, \sigma' \in \Sigma \forall r \in \mathcal{R} \forall d \in \mathcal{D} \quad (\Omega(\sigma) \wedge \tau(r, \sigma) = (d, \sigma')) \Rightarrow \Omega(\sigma')$. De plus $\Sigma_I \subseteq \Sigma_{|\Omega}$. D'après le lemme 1, l'implantation (τ, Σ_I) est $\mathbb{P}[\]$ -correcte.

6.3 $\mathbb{P}[\]$ -complétude de l'implantation

Pour montrer que l'implantation (τ_d, Σ_{I_d}) est $\mathbb{P}[\]$ -complète, on montre qu'à partir d'un état initial, on peut accéder à n'importe quel état sûr du système. On rappelle que l'ensemble des état initiaux sont les états qui n'ont aucun accès courant. Nous allons commencer par montrer la proposition suivante :

Proposition 2 $\forall \sigma \in \Sigma_{|\Omega_d} \sigma \in \Gamma_{\tau_d}(\Sigma_\emptyset)$

PREUVE. Nous raisonnons par induction sur le nombre $n = |\Lambda(\sigma)|$ (avec $\sigma \in \Sigma_{|\Omega_d}$) d'accès courants d'un état sûr et montrons :

$$\forall n \in \mathbb{N} \forall \sigma \in \Sigma_{|\Omega_d} |\Lambda(\sigma)| = n \Rightarrow \sigma \in \Gamma_{\tau_d}(\Sigma_\emptyset)$$

1. Si $n = 0$ alors $\sigma \in \Sigma_\emptyset$ et on a $\sigma \in \Gamma_{\tau_d}(\Sigma_\emptyset)$.
2. Supposons la propriété vraie au rang n et montrons-la au rang $n + 1$.
Soit σ_2 un état sûr du système tel que $\Lambda(\sigma_2) = n + 1$. Soit $a \in \Lambda(\sigma_2)$ et $\sigma_1 = (\Lambda(\sigma_2) \setminus \{a\}, f_d)$. On a que $|\Lambda(\sigma_1)| = n$.
Montrons que $\sigma_1 \in \Sigma_{|\Omega_d}$. En effet, $\forall m \in \Lambda(\sigma_1), m \in \Lambda(\sigma_2)$. De plus, on a $f_d(m)$ car $\Omega_d(\sigma_2)$. Donc, par définition de Ω_d , on a aussi $\Omega_d(\sigma_1)$.
Par hypothèse de récurrence, $\sigma_1 \in \Gamma_{\tau_d}(\Sigma_\emptyset)$. Par définition de τ_d , $\tau_d(\langle +, a \rangle, \sigma_1) = (yes, \sigma)$ car on a $f_d(a)$. Donc $\sigma_2 \in \Gamma_{\tau_d}(\Sigma_\emptyset)$. ◀

On a $\forall \sigma \in \Sigma_{|\Omega_d}, \sigma \in \Gamma_{\tau_d}(\Sigma_\emptyset)$ d'après la proposition 2. De plus, on a $\Sigma_{I_d} = \Sigma_\emptyset$. Donc, $\forall \sigma \in \Sigma_{|\Omega_d}, \sigma \in \Gamma_{\tau_d}(\Sigma_{I_d})$. Ce qui achève la preuve de la $\mathbb{P}[\]$ -complétude.

On prouve également la $\mathbb{P}[\]$ -complétude si $\Sigma_{I_d} \neq \Sigma_\emptyset$. Pour cela, nous devons commencer par montrer la proposition suivante :

Proposition 3 $\Sigma_\emptyset \subseteq \Gamma_{\tau_d}(\Sigma_d)$

PREUVE.

Nous prouvons cette propriété par induction sur le nombre $n = |\Lambda(\sigma)|$ (avec $\sigma \in \Sigma_d$) d'accès courants d'un état et montrons :

$$\forall n \in \mathbb{N} \forall \sigma \in \Sigma_d, |\Lambda(\sigma)| = n \Rightarrow \Sigma_\emptyset \subseteq \Gamma_{\tau_d}(\sigma)$$

1. Si $n = 0$ alors $\sigma \in \Sigma_\emptyset$ et comme $\sigma \in \Gamma_{\tau_d}(\sigma)$, on a $\Sigma_\emptyset \subseteq \Gamma_{\tau_d}(\sigma)$.
2. Supposons la propriété vraie au rang n et montrons-la au rang $n + 1$.
Soit $\sigma_2 \in \Sigma_d$ tel que $|\Lambda(\sigma_2)| = n + 1$. Soit $a \in \Lambda(\sigma_2)$. On a $\tau_d(\langle -, a \rangle, \sigma_2) = (yes, \sigma_1)$ t.q. $\sigma_1 = (\Lambda(\sigma_2) \setminus \{a\}, f_d)$ par définition de τ_d . On a $|\Lambda(\sigma_1)| = n$ et par hypothèse de récurrence, $\Sigma_\emptyset \subseteq \Gamma_{\tau_d}(\sigma_1)$. Donc $\Sigma_\emptyset \subseteq \Gamma_{\tau_d}(\sigma_2)$. ◀

Comme $\Sigma_{I_d} \subseteq \Sigma_d$, on a $\Sigma_\emptyset \subseteq \Gamma_{\tau_d}(\Sigma_{I_d})$ d'après la proposition 3. D'autre part, $\forall \sigma \in \Sigma_{|\Omega_d} \sigma \in \Gamma_{\tau_d}(\Sigma_\emptyset)$ par la proposition 2. Donc, $\forall \sigma \in \Sigma_{|\Omega_d}, \sigma \in \Gamma_{\tau_d}(\Sigma_{I_d})$. Ce qui achève la preuve de la $\mathbb{P}[\]$ -complétude si $\Sigma_{I_d} \neq \Sigma_\emptyset$.

6.4 \mathcal{R} -correction

Montrons que la fonction de transition τ_d est \mathcal{R} -correcte par rapport à $\|\mathcal{R}\|_\Sigma$.

PREUVE.

Soit r une requête. Soient σ, σ' deux états tels que $\tau_d(r, \sigma) = (yes, \sigma')$.

1. si $r = \langle +, s, o, x \rangle$ alors par définition de τ_d , $\sigma' = (\Lambda(\sigma) \cup \{(s, o, x)\}, f_d)$.
Comme $(s, o, x) \in \Lambda(\sigma')$, d'après la définition de la sémantique des requêtes, on a bien $(r, \sigma') \in \llbracket \mathcal{R} \rrbracket_\Sigma$
2. si $r = \langle -, s, o, x \rangle$ alors par définition de τ_d , $\sigma' = (\Lambda(\sigma) \setminus \{(s, o, x)\}, f_d)$.
Comme $(s, o, x) \notin \Lambda(\sigma')$, d'après la définition de la sémantique des requêtes, on a bien $(r, \sigma') \in \llbracket \mathcal{R} \rrbracket_\Sigma$

◀

En Focal, la propriété de \mathcal{R} -correction se code de la manière suivante :

```
letprop r_correct(tau in r -> self -> (d * self)) =
  all st1 st2 in self, all r1 in r, all d1 in d,
  tau(r1, st1) = (d1, st2) ->
  d!equal(d1, d!yes) ->
  !sem_req(r1, st2);
```

Nous avons en outre codé en Focal une deuxième propriété liée à la \mathcal{R} -correction :

```
letprop r_correct_bis(tau in r -> self -> (d * self)) =
  all st1 st2 in self, all r1 in r, all d1 in d,
  tau(r1, st1) = (d1, st2) ->
  d!equal(d1, d!no) ->
  !equal(st1, st2);
```

6.5 \mathcal{W} -conformité

Montrons que la fonction de transition τ_d est \mathcal{W} -conforme.

PREUVE.

Soient r une requête, d une décision, $\sigma = (m, f_d)$ et $\sigma' = (m', f_d)$ deux états tels que $\tau_d(r, \sigma) = (d, \sigma')$. Si $d = no$ alors, d'après la définition de τ_d , $\sigma = \sigma'$ et donc $\mathcal{W}(\sigma) = \mathcal{W}(\sigma')$, ce qui nous permet de conclure. Si $d = yes$ alors deux cas sont possibles :

1. si $r = \langle -, s, o, x \rangle$ alors d'après la définition de τ_d , $\sigma' = (m \setminus \{(s, o, x)\}, f_d)$.
Nous devons montrer que $\mathcal{W}(\sigma) \subseteq \mathcal{W}(\sigma')$. Soit E un ensemble d'accès tel que $E \in \mathcal{W}(\sigma)$. Par définition de \mathcal{W} , $\Omega_d((m \cup E, f_d))$ est vérifié, et donc pour tout accès (s, o, x) appartenant à $m \cup E$, $f_d((s, o, x))$ est satisfait. Or $m' = m \setminus \{(s, o, x)\}$. On en déduit que $\Omega_d((m' \cup E, f_d))$ est vérifié, ce qui nous permet de conclure que $E \in \mathcal{W}(\sigma')$.
2. si $r = \langle +, s, o, x \rangle$ alors d'après la définition de τ_d , $\sigma' = (m \cup \{(s, o, x)\}, f_d)$.
Nous devons montrer que $\mathcal{W}(\sigma') \subseteq \mathcal{W}(\sigma)$. Soit E un ensemble d'accès tel que $E \in \mathcal{W}(\sigma')$. Par définition de \mathcal{W} , $\Omega_d((m' \cup E, f_d))$ est vérifié, et donc pour tout accès (s, o, x) appartenant à $m' \cup E$, $f_d((s, o, x))$ est satisfait. Or $m' = m \cup \{(s, o, x)\}$. On en déduit que $\Omega_d((m \cup E, f_d))$ est vérifié, ce qui nous permet de conclure que $E \in \mathcal{W}(\sigma)$.

◀

7 Programme Focal

Dans cette section, nous présentons le programme Focal.

7.1 Structure

L'implémentation de notre modèle discrétionnaire a été faite en plusieurs étapes, d'après la structure décrite ci-dessous :

- La base du programme est constituée du modèle général, qui contient les espèces représentant les sujets, les objets, les modes d'accès, la politique de contrôle d'accès (caractérisée par le prédicat Ω), les requêtes, ainsi que la sémantique des requêtes.
- On ajoute une couche au modèle général pour définir le modèle discrétionnaire : on ajoute les espèces `policy_dis` et `model_dis`, qui affinent les espèces `policy` et `model` définies dans le modèle général.

Dans `policy_dis`, on implante le prédicat Ω , spécifique à la politique discrétionnaire. De même, dans `model_dis`, on implante la fonction de transition τ pour la politique discrétionnaire.

- La couche suivante est la spécialisation du modèle discrétionnaire en utilisant d'une part les ACL, et d'autre part les Capabilities.

On définit ainsi l'espèce `states_acl` (resp `states_capabilities`) où on déclare une fonction `acl` (resp. `capabilities`) qui renvoie pour un objet donné (resp. sujet) la liste des accès autorisés pour chaque sujet (resp. chaque objet). On définit également l'espèce `policy_acl` (resp. `policy_capabilities`) dans laquelle on implante la fonction de sécurité f_d à l'aide de la fonction `acl` (resp. `capabilities`), et l'espèce `model_acl` (resp. `model_capabilities`).

7.2 Exécution

7.2.1 ACL et Capabilities

Trace d'exécution :

```
les sujets:
mathieu, therese, charles
les objets:
ssurf, focal_pas_a_pas, photos_de_vacances
les modes d'accès:
read, write

accès autorisés:
( mathieu - ssurf - write )
( mathieu - ssurf - read )
( therese - ssurf - write )
( therese - ssurf - read )
( charles - ssurf - read )
```

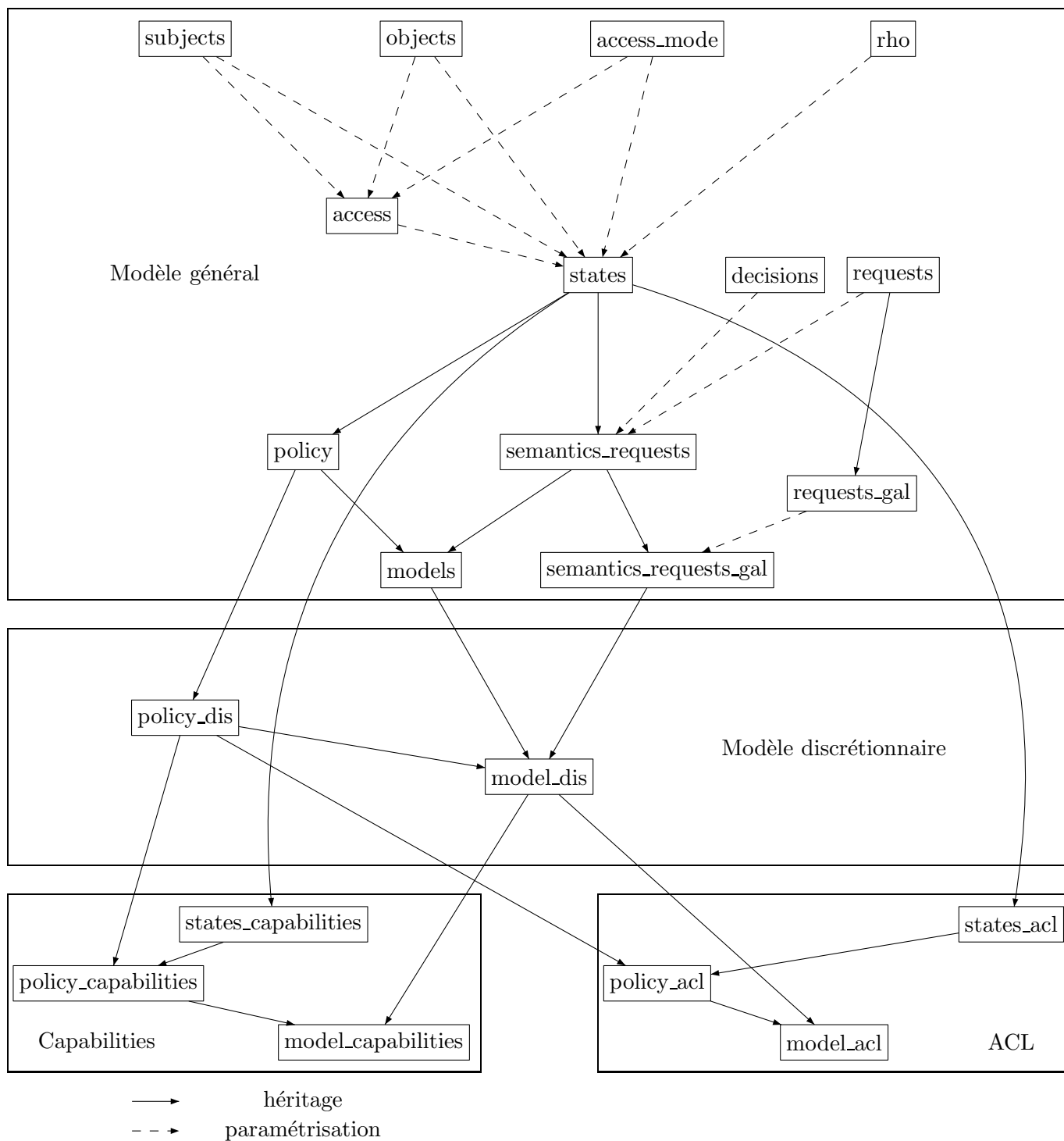


FIG. 2 – Structure du programme

```
( mathieu - photos_de_vacances - write )
```

7.2.2 ACL

Code Focal :

```
(* Creation de l'etat vide initial *)
let st1 = c_model_acl!create(c_set_of_access!vide);;
(* 1ere requete *)
let r1 = c_requests_gal!create(c_get_release!get,
                               c_subjects!mathieu,
                               c_objects!ssurf,
                               c_access_mode!write);;

let e1 = c_model_acl!tau(#r1,#st1);;
let d1 = #first(#e1);;
let st1b = #scnd(#e1);;
(* 1ere requete bis *)
let r1b = c_requests_gal!create(c_get_release!get,
                                c_subjects!charles,
                                c_objects!ssurf,
                                c_access_mode!read);;

let e1b = c_model_acl!tau(#r1b,#st1b);;
let d1b = #first(#e1b);;
let st2 = #scnd(#e1b);;
(* 2eme requete *)
let r2 = c_requests_gal!create(c_get_release!get,
                                c_subjects!mathieu,
                                c_objects!photos_de_vacances,
                                c_access_mode!read);;

let e2 = c_model_acl!tau(#r2,#st2);;
let d2 = #first(#e2);;
let st3 = #scnd(#e2);;
(* 3eme requete *)
let r3 = c_requests_gal!create(c_get_release!release,
                                c_subjects!therese,
                                c_objects!focal_pas_a_pas,
                                c_access_mode!read);;

let e3 = c_model_acl!tau(#r3,#st3);;
let d3 = #first(#e3);;
let st4 = #scnd(#e3);;
(* 4eme requete *)
let r4 = c_requests_gal!create(c_get_release!release,
                                c_subjects!mathieu,
                                c_objects!ssurf,
                                c_access_mode!write);;

let e4 = c_model_acl!tau(#r4,#st4);;
let d4 = #first(#e4);;
let st5 = #scnd(#e4);;
```

Trace d'exécution :

```

exemple implantation ACL
accès courants: []
requete: < + : mathieu : ssurf : write >
reponse: yes
accès courants: [( mathieu - ssurf - write ),]
requete: < + : charles : ssurf : read >
reponse: yes
accès courants: [( mathieu - ssurf - write ),
                  ( charles - ssurf - read ),]
requete: < + : mathieu : photos_de_vacances : read >
reponse: no
accès courants: [( mathieu - ssurf - write ),
                  ( charles - ssurf - read ),]
requete: < - : therese : focal_pas_a_pas : read >
reponse: yes
accès courants: [( mathieu - ssurf - write ),
                  ( charles - ssurf - read ),]
requete: < - : mathieu : ssurf : write >
reponse: yes
accès courants: [( charles - ssurf - read ),]

```

7.2.3 Capabilities

Code Focal :

```

(* Creation de l'etat vide initial *)
let st1c = c_model_capabilities!create(c_set_of_access!vide);;
(* 1ere requete *)
let r1c = c_requests_gal!create(c_get_release!get,
                               c_subjects!mathieu,
                               c_objects!ssurf,
                               c_access_mode!write);;

let e1c = c_model_capabilities!tau(#r1c,#st1c);;
let d1c = #first(#e1c);;
let st1bc = #scnd(#e1c);;
(* 1ere requete bis *)
let r1bc = c_requests_gal!create(c_get_release!get,
                                 c_subjects!charles,
                                 c_objects!ssurf,
                                 c_access_mode!read);;

let e1bc = c_model_capabilities!tau(#r1bc,#st1bc);;
let d1bc = #first(#e1bc);;
let st2c = #scnd(#e1bc);;
(* 2eme requete *)
let r2c = c_requests_gal!create(c_get_release!get,
                               c_subjects!mathieu,
                               c_objects!photos_de_vacances,
                               c_access_mode!read);;

let e2c = c_model_capabilities!tau(#r2c,#st2c);;
let d2c = #first(#e2c);;

```

```

let st3c = #scnd(#e2c);;
(* 3eme requete *)
let r3c = c_requests_gal!create(c_get_release!release,
                                c_subjects!therese,
                                c_objects!focal_pas_a_pas,
                                c_access_mode!read);;

let e3c = c_model_capabilities!tau(#r3c,#st3c);;
let d3c = #first(#e3c);;
let st4c = #scnd(#e3c);;
(* 4eme requete *)
let r4c = c_requests_gal!create(c_get_release!release,
                                c_subjects!mathieu,
                                c_objects!ssurf,
                                c_access_mode!write);;

let e4c = c_model_capabilities!tau(#r4c,#st4c);;
let d4c = #first(#e4c);;
let st5c = #scnd(#e4c);;

```

Trace d'exécution :

```

exemple implantation CAPABILITIES
accès courants: []
requete: < + : mathieu : ssurf : write >
reponse: yes
accès courants: [( mathieu - ssurf - write ),]
requete: < + : charles : ssurf : read >
reponse: yes
accès courants: [( mathieu - ssurf - write ),
                 ( charles - ssurf - read ),]
requete: < + : mathieu : photos_de_vacances : read >
reponse: no
accès courants: [( mathieu - ssurf - write ),
                 ( charles - ssurf - read ),]
requete: < - : therese : focal_pas_a_pas : read >
reponse: yes
accès courants: [( mathieu - ssurf - write ),
                 ( charles - ssurf - read ),]
requete: < - : mathieu : ssurf : write >
reponse: yes
accès courants: [( charles - ssurf - read ),]

```

7.3 Preuves Zenon

Zenon est un démonstrateur automatique dans Focal. Il permet de prouver des propriétés du programme. Lorsqu'une preuve est correcte, Zenon génère un script Coq qui peut servir à la certification du programme. Nous avons utilisé Zenon dans le modèle discrétionnaire afin de prouver la correction de la fonction de transition τ_d vis-à-vis du prédicat de sécurité Ω_d , ainsi que sa correction par rapport à la sémantique des requêtes (\mathcal{R} -correction).

Nous donnons ici les trois propriétés que nous avons prouvées ainsi que l'architecture de chacune des preuves.

```

theorem tau_secure:
  all st1 st2 in self, all r1 in r, all d1 in d,
    !tau(r1, st1) = (d1, st2)
    -> !omega(st1)
    -> !omega(st2)
proof:
  <1>1 assume r1 in r
        st1 st2 in self
        d1 in d
        H1: !tau(r1, st1) = (d1, st2)
        H2: !omega(st1)
        prove !omega(st2)
  <2>1 assume H3: r!is_get (r1)
        prove !omega(st2)
        .....
  <2>2 assume H4: r!is_rel (r1)
        prove !omega(st2)
        .....
  <2>f qed by r!get_or_rel, <2>1, <2>3
<1>f qed;

```

```

theorem tau_r_correct :
  all st1 st2 in self, all r1 in r, all d1 in d,
    !tau(r1, st1) = (d1, st2)
    -> d!equal(d1, d!yes)
    -> !sem_req(r1, st2)
proof:
  <1>1 assume st1 st2 in self
        r1 in r
        d1 in d
        H1: !tau(r1, st1) = (d1, st2)
        prove d!equal(d1, d!yes)
        -> !sem_req(r1, st2)
  <2>1 assume H2: r!is_get (r1)
        prove d!equal(d1, d!yes)
        -> !sem_req(r1, st2)
        .....
  <2>2 assume H3: r!is_rel (r1)
        prove d!equal(d1, d!yes)
        -> !sem_req(r1, st2)
        .....
  <2>f qed by r!get_or_rel, <2>1, <2>2
<1>f qed;

```

```

theorem tau_r_correct_bis :
  all st1 st2 in self, all r1 in r, all d1 in d,
    !tau(r1, st1) = (d1, st2)
    -> d!equal(d1, d!no)
    -> !equal(st1, st2)
proof:
  <1>1 assume st1 st2 in self
        r1 in r
        d1 in d
        H1: !tau(r1, st1) = (d1, st2)
    prove d!equal(d1, d!no)
        -> !equal(st1, st2)
  <2>2 assume H2: r!is_get (r1)
    prove d!equal(d1, d!no)
        -> !equal(st1, st2)
    .....
  <2>3 assume H3: r!is_rel (r1)
    prove d!equal(d1, d!no)
        -> !equal(st1, st2)
    .....
  <2>f qed by r!get_or_rel, <2>1, <2>2
<1>f qed;

```

8 Conclusion

Nous avons vu comment définir une politique de contrôle d'accès en se plaçant dans un cadre formel. Cette approche formelle nous a permis de démontrer certaines propriétés de cette politique. Partant de la spécification d'une politique de contrôle d'accès générale, nous avons instancié la fonction de transition τ et le prédicat de sécurité Ω pour spécifier une politique de contrôle d'accès discrétionnaire.

Le langage Focal nous a permis d'implanter cette politique en respectant la spécification. De plus, nous avons prouvé plusieurs propriétés du programme à l'aide de Zenon. Elles nous ont permis de voir que sur une preuve papier, de nombreuses propriétés qu'on utilise (t.q. la transitivité de l'égalité par exemple) sont implicites mais qu'elles ne vont pas soi.

La continuation naturelle du stage serait d'implanter les requêtes administratives. La fonction de sécurité f_d pourrait alors être modifiée. Cela poserait des problèmes concernant la correction et la complétude de la politique de sécurité.

Un prolongement possible du projet est d'enrichir le modèle en ajoutant une nouvelle fonction de sécurité. En complément de f_d qui définit tous les accès autorisés, nous pourrions ajouter une fonction g_d qui définit tous les accès obligatoires, c'est-à-dire les accès courants sans lesquels un état n'est pas sûr. Ce nouveau modèle implique que l'état vide (sans accès courants) n'est plus forcément sûr. Dans ce cas de figure, nous perdons un certain

nombre de propriétés. Nous avons commencé à implanter ce nouveau modèle.

Références

- [1] Y. Falcone. Un cadre formel pour le test de politique de sécurité. Master's thesis, Université Grenoble 1, 2006. <http://www-verimag.imag.fr/~falcone/files/dea/html/>.
- [2] M. Jaume and C. Morisset. Formalisation and implementation of access control models. In *Information Assurance and Security (IAS'05) International Conference on Information Technology, ITCC*, pages 703–708. IEEE CS Press, 2005.
- [3] M. Jaume and C. Morisset. Towards a formal specification of access control. In P. Degano, R. Kusters, L. Vigano, and S. Zdancewic, editors, *Proceedings of the Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis, FCS-ARSPA'06*, pages 213–232, 2006.
- [4] B.W. Lampson. Protection. *Operating Systems Review*, 8(1) :18–24, January 1974.
- [5] Stéphane Natkin. *Les protocoles de sécurité d'internet*. Dunod, 1999.
- [6] Focal project. *Focal, version 0.2 Tutorial and reference manual*. LIP6 – INRIA – CNAM, sept 2004. Distribution available at : <http://focal.inria.fr>.