Prise en Compte de Règles de Sécurité Temporelles dans une Spécification TEFSM d'un Système

Wissam Mallouli, Amel Mammar, Ana Cavalli

Prise en Compte de Règles de Sécurité Temporelles dans une Spécification TEFSM d'un Système[1]

Wissam Mallouli*, Amel Mammar, Ana Cavalli****

*Montimage EURL, 39 rue Bobillot,75013 Paris
wissam.mallouli@montimage.com

**TELECOM & Management SudParis, 9 Rue Charles Fourier, 91011 Évry Cedex
{amel.mammar, ana.cavalli}@it-sudparis.eu

RÉSUMÉ. *Dans cet article, nous proposons une approche formelle de modélisation conjointe des aspects fonctionnels et sécuritaires d'un système. Cette approche consiste à enrichir la spécification fonctionnelle d'un système, exprimée à l'aide d'une TEFSM (Timed Extended Finite State Machine), par des règles de sécurité temporelles modélisées en Nomad. Nomad est un langage formel bien adapté pour l'expression de propriétés de sécurité telles que les permissions, les interdictions et les obligations. Le modèle fonctionnel sécurisé global ainsi obtenu peut être utilisé à plusieurs fins tels que la génération de code, le test du système, la vérification du modèle(model checking), etc. Afin de s'assurer de la faisabilité de notre approche, celle-ci a été expérimentée à plusieurs applications distribuées.*

ABSTRACT. *In this paper, we propose a formal approach for a joint modelling of functional and security aspects of a system. This approach consists in enriching the functional specification of a system, expressed with a TEFSM (Timed Extended Finite State Machine), with timed security rules modelled in Nomad. Nomad is a formal language suitable for expressing security properties such as permissions, prohibitions and obligations. The overall obtained security model can be used for several purposes such as code generation, system testing, verification (model checking), etc. To demonstrate the feasibility of our approach, we experienced it on several distributed applications.*

MOTS-CLÉS : *Règles de sécurité temporelles, langage Nomad, modèles TEFSM.*

KEY WORDS: *Timed security rules, Nomad language, TEFSM models*

1. The research leading to these results has received funding from the European Community's Seventh Framework Program (FP 7/2007-203) under the grant agreement number 215995 (http ://www.shields-project.eu/).

1. Introduction

Cette dernière décennie a connu une prolifération importante des systèmes complexes dans divers domaines. Ces systèmes sont souvent caractérisés par des aspects fonctionnels temporisés et une demande accrue en termes de sécurité. Ceci est principalement dû au caractère distribué et de plus en plus complexe de ces systèmes. Par conséquent, les développeurs de ce type de systèmes sont non seulement confrontés aux exigences fonctionnelles mais doivent également prendre en considération les besoins en sécurité de leurs applications. Les besoins fonctionnels représentent les services que doit fournir le système à ses utilisateurs finaux, tandis que les exigences en sécurité désignent les propriétés que doit remplir le système pour garantir la qualité de service requise.

Les systèmes industriels sont souvent conçus en deux étapes. La première étape consiste à modéliser le système à développer d'un point de vue purement fonctionnel. À cette fin, plusieurs modèles formels basés sur la théorie des automates temporisés ont été proposés dans la littérature[ALU 94, PAR 97]. Une fois cette spécification fonctionnelle établie, cette dernière est complétée par l'intégration des besoins en sécurité du système à réaliser. Cependant, les ingénieurs développeurs de ce type de systèmes ne sont pas nécessairement experts dans le domaine de la sécurité, et enrichir une spécification existante par des règles (besoins) de sécurité peut s'avérer comme une tâche loin d'être facile.

Pour répondre à ce problème et assister les développeurs dans leur tâche, nous avons déjà proposé un processus formel permettant d'augmenter une spécification fonctionnelle d'un système par des règles de sécurité exprimées en langage OrBAC [Abo 03] (**Or**ganisationnal **B**ased **A**ccess **C**ontrol). Ces règles de sécurité expriment l'obligation, la permission, ou l'interdiction à un utilisateur de réaliser une action donnée sous certaines conditions appelées *contexte*. Les règles considérées par ce processus ne prennent pas en compte les aspects temporisés du système. Une approche, présentée dans [LI 07], propose de traduire des règles de sécurité (sans aspects temporisés) en observeurs communicants avec la description fonctionnelle du système spécifiée par le formalisme des EFSMs (**E**xtended **F**inite **S**tate **M**achine). Afin de pouvoir prendre en compte des règles de sécurité temporisées, nous proposons dans ce papier une approche basée sur le langage Nomad [CUP 05] (**N**on-atomic **a**ctions and **d**eadlines) qui permet de modéliser les aspects temps par la notion de *délai*. En utilisant ce langage, il est, par exemple, possible d'exprimer l'obligation pour un client d'envoyer le paiement de sa facture sous un délai d'une semaine après sa réception.

La contribution principale de ce travail porte sur la définition d'une approche formelle prouvée pour l'intégration de règles de sécurité à contextes temporisés dans une spécification TEFSM [PAR 97]. Une telle approche d'intégration consiste en l'ajout d'horloges pour représenter l'écoulement du temps, le renforcement de gardes et l'ajout d'états et de transitions pour rendre l'exécution de certaines actions possible uniquement à des instants précis déterminés par les valeurs des horloges. La spécification TEFSM ainsi obtenue est appelée *TEFSM sécurisée* car elle englobe l'ensemble des règles de sécurité. Notons que l'intégration des règles de sécurité dans la spécification fonctionnelle d'un système n'est pas une fin en soi. La TEFSM sécurisée produite peut être utilisée à plusieurs fins tels que la génération de code [PAL 02], la preuve du système [ALA 97], le model checking [CLA 04] ou la génération automatique de tests [ISO 94, CAV 93], etc.

D'un point de vue théorique, l'approche d'intégration que nous avons développée classifie les règles de sécurité en trois catégories. Les deux premières incluent les règles de base avec contextes

simples dont les actions sont atomiques ou décomposables. Un contexte simple ne comporte qu'un seul opérateur de temps sans aucun connecteur logique. La troisième catégorie est générale et traite des règles de sécurité plus élaborées à contextes complexes. Nous avons montré que ces règles peuvent être décomposées en une ou plusieurs règles de base sur lesquelles le processus d'intégration défini pour les deux premières catégories peut être appliqué [MAL 08]. Par souci de place, ce papier présente uniquement l'intégration de deux types de règles de sécurité. Une description plus complète du processus d'intégration est disponible dans [MAL 08].

La suite du papier est structurée comme suit. La section 2 présente les concepts de base pour la modélisation d'un système d'un point de vue fonctionnel et sécuritaire. Dans la section 3, nous donnons les principes généraux de notre approche d'intégration de règles de sécurité temporelles dans une TEFSM. Les algorithmes d'intégration de deux types de régles de sécurité avec actions atomiques sont décrits dans la section 4. Pour montrer la validité de notre approche, une preuve de correction de l'algorithme d'intégration d'une prohibition est donnée en section 5. Enfin, nous terminons par une conclusion et des perspectives.

2. Préliminaires

2.1. *Modélisation des systèmes communicants par des TEFSMs*

Modéliser un système a pour but de produire une spécification abstraite opérationnelle de ce dernier. Cette spécification peut porter sur les différentes facettes du système notamment les aspects fonctionnels incluant les contraintes temporisées. Une telle spécification permet une meilleure compréhension du système, et peut être également utilisée comme entrée des outils de validation, des model checkers et des générateurs de test pour la vérification de propriétés sur le système considéré.

Pour la modélisation des systèmes communicants, nos travaux reposent sur l'utilisation du modèle de TEFSM défini dans [BOZ 04] et supporté par le langage IF [BOZ 04]. Notre choix pour ce modèle est justifié comme suit. Ce modèle, assez facile à manipuler, fournit les principaux concepts utiles pour la modélisation des systèmes temporisés. De plus, un nombre conséquent d'outils de simulation et de génération de tests supportant ce modèle existent déjà et sont open source.

Définition 1 *Une TEFSM M est un 7-tuple $M = <S, s_0, I, O, \vec{x}, \vec{c}, Tr>$ avec S désignant un ensemble fini d'états, s_0 un état initial, I représentant un ensemble fini de symboles d'entrée (des messages éventuellement paramétrés), O dénotant un ensemble fini de symboles de sortie (des messages éventuellement paramétrés), $\vec{x}$ et $\vec{c}$ représentant respectivement des ensembles finis de variables et d'horloges, et Tr un ensemble fini de transitions. Une transition tr est un 4-tuple $tr =< s_i, s_f, G, Act >$ où s_i et s_f représentant respectivement l'état initial et l'état final de la transition, la garde de la transition G désigne un prédicat portant sur les variables $\vec{x}$ et les horloges $\vec{c}$, et Act représente une séquence d'actions atomiques incluant l'émission de messages d'entrée et de sortie, l'affectation des variables, l'assignation des horloges, et la création/destruction des processus.*

L'exécution de toute transition est considérée comme instantanée. Autrement dit, toutes les actions associées à une transition sont accomplies simultanément avec un temps d'exécution nul. La progression du temps s'opère donc dans certains états avant le déclenchement des transitions franchissables.

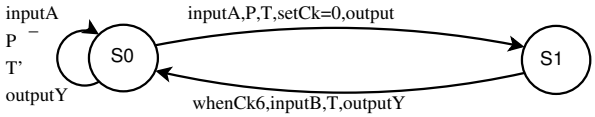


Figure 1. *Exemple d'une TEFSM simple à deux états.*

Nous illustrons la notion de TEFSM à travers l'exemple de la Figure 1. Cette TESFM est composée de deux états S_0, S_1 et trois transitions libellées par deux entrées A et B, deux sorties X et Y, une garde (ou prédicat) P sur les variables, une horloge Ck et trois tâches T, T' et T''. Le comportement de cette TEFSM est comme suit : initialement le système est dans l'état S_0, le prédicat P est évalué à l'occurrence de l'entrée A. Si P est vérifié, la tâche T est exécutée, l'horloge Ck est lancée, le message de sortie X est émis et le système arrive dans l'état destination S_1. Sinon, le système boucle sur le même état initial S_0 en émettant le signal de sortie Y avec exécution cette fois-ci de l'action T'. Une fois la TEFSM dans l'état S_1, l'état S_0 peut être atteint quand la valeur de l'horloge dépasse la valeur 6 et que le message d'entrée B est reçu. Dans ce cas, l'action T'' est réalisée et le message de sortie Y est émis.

Notations : Dans le reste du papier les notations suivantes sont utilisées. Pour chaque action a apparaissant dans la séquence d'actions Act : *avant*(a) (resp. *apres*(a)) désigne les actions de Act exécutées avant (resp. après) l'action a. *avant*(a) (resp. *apres*(a)) est vide si l'action *a* est la première (resp. dernière) action de *Act*. La séquence d'actions *Act* peut être réécrite sous la forme suivante : $(avant(a); a; apres(a))$. Par exemple, la séquence d'action de la transition t_1 de la figure 1 peut être réécrite en : $(Act = (avant(T); T; apres(T)))$ avec $(avant(T) = input\ A)$ et $(apres(T) = (set\ Ck := 0; output\ X))$.

2.2. *Spécification des règles de sécurité en Nomad*

Parmi les différents langages développés pour la description formelle de politiques de sécurité (comme SPL[RIB 01], TPL[HER 00] et PDL[LOB 99]), nous avons choisi le langage formel Nomad. En combinant les logiques déontique et temporelle, Nomad offre les concepts nécessaires pour décrire des règles de sécurité incluant des aspects temporels. Ces règles de sécurité s'expriment sous forme de permissions, obligations et prohibitions d'accomplir une action donnée si une certaine action a été (resp. sera) réalisée t unités de temps dans le passé(resp. dans le futur).

2.2.1. *Syntaxe et sémantique du langage Nomad*

Pour modéliser l'aspect fonctionnel d'un système, nous considérons qu'une action atomique correspond au concept d'actions tel que décrit dans le modèle des TEFSMs.

Définition 2 *(Action atomique) Une action atomique désigne l'une des actions suivantes : affectation d'une variable, assignation d'une horloge, un message d'entrée, un message de sortie, création/destruction d'un processus.*

Définition 3 *(Action non-atomique) Une action non-atomique est définie inductivement comme suit : Si A et B sont des actions atomiques, alors $(A; B)$ (resp. $(A||B)$), qui signifie que "A est immédia-*

tement suivie par B" (resp. A et B sont exécutées simultanément), est une action non atomique. De même, si A et B sont des actions (atomiques ou pas), alors $(A;B)$ et $(A||B)$ sont des actions non-atomiques.

Définition 4 *(Formule) Si A est une action alors $start(A)$ (débuter A) et $done(A)$ (finir A) sont des formules.*

Nous donnons ci-après quelques propriétés sur les actions et les formules :

– Si α et β sont des formules alors $\neg\alpha$, $(\alpha \wedge \beta)$ et $(\alpha \vee \beta)$ sont des formules.

– Si α est une formule alors $O^d\alpha$ (α est vrai d unités de temps dans le passé si $d \leq 0$, α sera vrai après d unités de temps si $d \geq 0$) est aussi une formule.

– Si α est une formule alors $O^{<d}\alpha$ (dans un délai de d unités de temps dans le passé, α était vrai si $d \leq 0$, α sera vrai dans un délai de d unités de temps si $d \geq 0$) est une formule.

– Si α et γ sont des formules, alors $(\alpha|\gamma)$ est une formule exprimant que α est vrai dans le contexte γ.

Dans la suite du papier, nous utilisons la notation $O^{[<]d}$ pour couvrir les deux cas O^d et $O^{<d}$. Notons également qu'avec Nomad, nous considérons que le temps est discret. Le choix de l'unité de temps peut être important et dépend du système étudié. En fonction de la précision désirée, des unités de secondes, millisecondes ou microsecondes peuvent être utilisées.

Définition 5 *(Règle de sécurité) Si α et β sont des formules, alors $\mathcal{R}$ (α | β) est une règle de sécurité avec $\mathcal{R}$ dénotant l'un des opérateurs déontiques suivants : {$\mathcal{P}$, $\mathcal{F}$, $\mathcal{O}$}. La sémantique de cette règle de sécurité est comme suit : $\mathcal{P}$(α | β) (resp. $\mathcal{F}$(α | β), $\mathcal{O}$(α | β)) : il est permis (resp. interdit, obligatoire) d'exécuter α quand le contexte β est vérifié.*

Plus de détails sur la syntaxe et la sémantique du langage Nomad peuvent être trouvés dans [CUP 05].

2.2.2. *Exemples de spécification de règles de sécurité*

Nous présentons dans cette section quelques règles de sécurité exprimées en Nomad.

Exemple 1 :

$$\mathcal{P}(start\ (input\ ReqWrite(user, file.doc))| \\ (O^{\leq -5s}\ done\ (output\ AuthOK(user)))\wedge\ (\neg\ done\ (output\ DisconnectOK(user))))$$

Cette règle exprime la permission accordée à un utilisateur pour une demande d'écriture dans le fichier 'file.doc', s'il a été authentifié depuis au moins 5 secondes, et que sa session est encore en cours.

Exemple 2 :

$$\mathcal{O}(start\ (output\ DisconnectOK(user))| \\ (O^{\leq -30min}\neg\ done\ (input\ Message(user)))\wedge\ (\neg\ done(output\ DisconnectOK(user))))$$

Cette règle oblige le système à déconnecter tout utilisateur qui resterait inactif depuis 30 minutes.

Exemple 3 :

$$\mathcal{F}(start\ ((output\ AuthOK(user)))|$$
$$O^{\leq -0.001s}\ done\ (output\ AuthOK(user))\wedge\ (\neg\ done(output\ DisconnectOK(user))))$$

Pour prévenir le déni de service, cette règle interdit au système d'accepter, dans la même milliseconde, deux connections simultanées pour le même utilisateur.

3. Méthodologie d'intégration de règles de sécurité

L'intégration de règles de sécurité dans une spécification TEFSM, représentant les aspects comportementaux d'un système, produit une TEFSM sécurisée qui respecte l'ensemble des règles intégrées. De manière générale, nous avons établi un algorithme d'intégration pour chaque type de règles de sécurité (permission, obligation, interdiction). En fonction du type de la règle, l'intégration d'une règle consiste à ajouter de nouveaux états et/ou transitions, modifier les gardes des transitions mais également à déclencher des horloges pour représenter l'écoulement du temps. Le processus d'intégration d'une politique de sécurité (ensemble de règles de sécurité) que nous avons développé procède donc comme suit. Pour chaque transition de la spécification initiale, on identifie les règles de politique à intégrer, c.à.d, les règles sur lesquelles un des algorithmes établis est applicable.

Pour appliquer le processus d'intégration que nous avons développé, nous supposons que les règles de sécurité forment un ensemble cohérent sans règles redondantes ou contradictoires. Plusieurs travaux ont été développés sur la vérification de la cohérence d'une politique de sécurité (Voir par exemple [CUP 06]). La vérification de cette cohérence n'étant pas le but de ce papier, nous considérons que la politique de sécurité à intégrer a été préalablement vérifiée. Par exemple, la politique de sécurité composée, entre autres, des deux règles suivantes $\mathcal{O}$ *(start(A)* | O^{-d} *done(B))* et $\mathcal{F}$ *(start(A)* | O^{-d} *done(B))* est incohérente car il n'est pas possible d'obliger le système à exécuter l'action A dans le contexte ($C = O^{-d}$ done(B)) si cette action est interdite dans le même contexte.

Conformément à la syntaxe du langage Nomad, une règle de sécurité peut être de formes diverses. Il serait bien évidemment fastidieux de traiter chacune de ces formes. Par conséquent, nous classifions les règles Nomad suivant les deux principales classes suivantes :

1) *Règle de sécurité de base* : cette classe inclut les règles de la forme $\mathcal{R}(start(A)|O^{[<]d}done(B))$ avec A et B désignant des actions. Une règle de cette classe ne contient qu'un seul opérateur temporel mais aucun connecteur logique. Afin de faciliter l'intégration de cette classe de règle, nous distinguons également deux sous-classes :

a) *Règles de sécurité de base avec actions atomiques* : A et B sont des actions atomiques dans ce cas.

b) *Règles de sécurité de base avec actions non-atomiques* : A ou B ou les deux sont des actions non-atomiques.

2) *Règles de sécurité générales* : cette classe englobe toutes les règles de sécurité qui n'appartiennent pas à la première classe. Une règle de cette classe peut donc contenir plusieurs opérateurs contextuels ou/et connecteurs logiques.

Cette classification étant établie, nous avons élaboré, pour chaque type de règles, un algorithme qui permet d'intégrer des règles de sécurité de la première classe. Nous avons également défini un en-

semble de règles de réécriture prouvées ayant pour but de transformer (décomposer) toute règle de sécurité de la seconde classe en règles plus simples de la première classe sur lesquelles les algorithmes définis pour cette classe peuvent être réutilisés. Ces règles de réécriture portent principalement sur les opérateurs contextuels, temporels et les connecteurs logiques.

Par souci de place, nous nous restreignons dans cet article à la présentation de deux algorithmes d'intégration de règles de base à actions atomiques. Une description complète de l'ensemble des algorithmes d'intégration ainsi que des règles de réécriture peut être trouvée dans [MAL 08].

4. Intégration de règles de sécurité avec actions atomiques

Cette section les algorithmes que nous avons développés pour l'intégration de deux types de règles de sécurité (prohibition et obligation) dans une TEFSM représentant l'aspect fonctionnel d'un système.

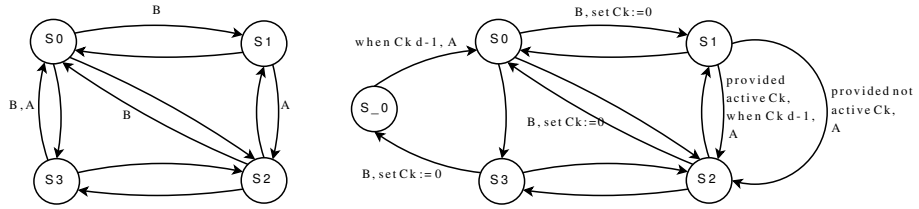
4.1. *Intégration d'une prohibition de la forme* $\mathcal{F}(start(A)|O^{<-d}done(B))$

Une règle de prohibition est relative à une action présente déjà dans la spécification TEFSM initiale du système. Une solution naïve pour intégrer une telle règle serait d'inhiber toutes les transitions (rendre faux les gardes) où l'action B (resp. l'action A) apparaît. De cette manière, aucune action B (resp action A) n'est exécutée. Par conséquent, la règle est bien respectée soit parce que le contexte n'est jamais vérifié ou que l'action prohibée n'est jamais exécutée. L'inconvénient principal de cette solution est qu'elle élimine des comportements possibles du système qui ne violeraient pas la règle considérée. En effet, cette règle de prohibition n'interdit pas l'exécution de l'action A si l'action B a été exécutée, par exemple, $(d+1)$ unités de temps dans le passé.

Algorithme 1 Intégration d'une règle de prohibition

ENTRÉES: Le modèle TEFSM $M =< S, s_0, I, O, \vec{x}, \vec{c}, Tr >$ et la règle de prohibition $\mathcal{F}$ *(start(A)* | $O^{<-d}$ *done(B))*

1: Définir une horloge globale Ck ;
2: **pour chaque** (transition tr telle que ($tr \in Tr \wedge tr =< S_i, S_j, G, Act >$)) **faire**
3: **si** ($B \in Act$) **alors**
4: /*remise à zéro de l'horloge Ck après exécution de l'action B*/
5: $tr :=< S_i, S_j, G, \{avant(B), B, set\ Ck := 0, apres(B)\} >$;
6: **si** (($A \in Act$) $\wedge$ $A \in apres(B)$) **alors**
7: /*la transition tr est de la forme $< S_i, S_j, G, \{avant(B), B, apres(B) \cap avant(A), A, apres(A)\} >$*/
8: /*créer un nouvel état S' et une nouvelle transition tr'*/
9: $tr :=< S_i, S', G, \{avant(B), B, apres(B) \cap avant(A)\} >$;
10: $tr' :=< S', S_j, \{when\ (Ck > d-1)\}, \{A, apres(A)\} >$;
11: **finsi**
12: **sinon**
13: **si** ($A \in Act$) **alors**
14: /*créer une nouvelle transition tr'*/
15: $tr :=< S_i, S_j, \{G, provided\ not\ active\ Ck\}, \{avant(A), A, apres(A)\} >$;
16: $tr' :=< S_i, S_j, \{G, provided\ active\ Ck, when\ (Ck > d-1)\}, \{avant(A), A, apres\ (A)\} >$;
17: **finsi**
18: **finsi**
19: **fin pour**



(a) Spécification initiale (b) Spécification sécurisée

Figure 2. *Intégration de la règle de prohibition : $\mathcal{F}$ (start(A) | $O^{<-d}$ done(B))*

L'idée de base pour intégrer une règle de prohibition est de vérifier son contexte avant exécution de l'action prohibée A. Si le contexte est vérifié, l'action A doit être interdite sinon, elle peut être exécutée tout en respectant la règle de prohibition. Comme nous devons traiter des contextes temporels, nous devons définir une horloge pour gérer le temps. L'intégration d'une règle de prohibition de la forme $\mathcal{F}(start(A)|\ O^{<-d}done(B))$ est réalisée comme suit (Voir Algorithme 1) :

– création d'une horloge publique globale Ck modifiable par toutes les TEFSMs,

– l'horloge Ck est remise à zéro à chaque occurrence de l'action B (*set* $Ck = 0$). Intuitivement, l'horloge Ck mesure le temps écoulé depuis la dernière exécution de l'action B. Avant la première exécution de l'action B, l'horloge Ck est simplement inactive.

– Avant exécution de l'action prohibée A, on vérifie si l'horloge Ck est active (*provided active* Ck). Si c'est le cas, on examine sa valeur (*When* $Ck > d - 1$) pour déduire si l'action A peut être exécutée ou pas. Si l'horloge n'est pas active (*provided not active* Ck), l'action A peut être réalisée car cela signifie que l'action B n'a pas été exécutée auparavant.

La Figure 2.b est le résultat de l'application de l'algorithme 1 pour l'intégration de la règle de prohibition *$\mathcal{F}$ (start(A) | $O^{<-d}$ done(B))* sur la TEFSM de la Figure 2.a. Cette règle exprime que l'action A ne doit pas être exécutée si l'action B a été réalisée depuis moins de d unité de temps.

4.2. *Intégration d'une obligation de la forme $\mathcal{O}$ (start(A) | O^{-d} done(B))*

Pour intégrer une règle d'obligation, nous utilisons un processus RHP qui assure l'exécution de l'action obligatoire si cette dernière n'a pas encore été exécutée par le système initial. Dans le cas d'une règle d'obligation de la forme *$\mathcal{O}$ (start(A) | O^{-d} done(B))*, un tel processus peut être créé n fois (par l'instruction *fork*), avec n représentant le nombre maximum d'occurrences de l'action B qui peuvent être exécutées durant d unités de temps. Ce processus met à jour une horloge et attend jusqu'à l'écoulement du temps avant d'exécuter l'action obligatoire A. L'algorithme 2 décrit formellement ces différentes étapes. Nous posons comme hypothèse que la spécification TEFSM initiale n'est pas sécurisée, c.à.d, qu'elle n'exécute pas d'action A au bout de d unités de temps depuis l'exécution de l'action B.

La Figure 3 illustre l'intégration d'une règle d'obligation de la forme *$\mathcal{O}$ (start(A) | O^{-d} done(B))* dans la spécification TEFSM de la Figure 2.a dans laquelle il existe plusieurs occurrences de l'action atomique B.

Algorithme 2 Intégration d'une règle d'obligation

ENTRÉES: Le modèle TEFSM $M =< S, s_0, I, O, \vec{x}, \vec{c}, Tr >$ et la règle de prohibition $\mathcal{O}$ *(start(A)* | O^{-d} *done(B))*

1: **pour chaque** (transition tr telle que ($tr \in Tr \wedge tr =< S_i, S_j, G, Act >$)) **faire**
2: **si** ($B \in Act$) **alors**
3: $tr :=< S_i, S_j, G, (avant(B); B; fork\ RHP\ apres(B)) >$
4: /*RHP est un nouveau processus qui traite la règle d'obligation*/
5: **finsi**
6: **fin pour**
7: **pour** RHP process **faire**
8: Définir une nouvelle horloge Ck
9: Définir un nouveau état Wait
10: Définir deux transitions tr_1 et tr_2
11: $tr_1 :=< S_0, Wait, \_, set\ Ck := 0 >$
12: $tr_2 :=< Wait, \_, when\ Ck > d - 1, (A; stop) >$
13: **fin pour**

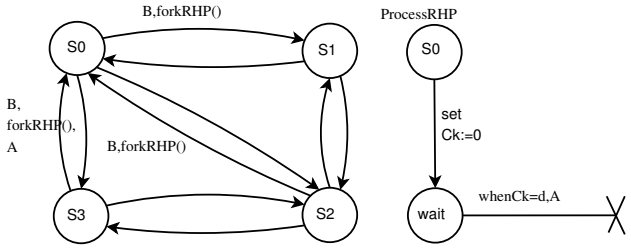


Figure 3. *Intégration de la règle d'obligation $\mathcal{O}$ (start(A) | O^{-d} done(B)).*

5. Preuve de correction de l'approche proposée

Pour s'assurer de la correction de l'approche d'intégration proposée, nous avons prouvé formellement l'ensemble des algorithmes développés et également les diverses règles de réécriture [MAL 08]. Cet article présente uniquement la preuve de correction établie pour l'algorithme 1. Pour prouver cet algorithme, on considère une horloge globale $gck(t)$ du système. En élaborant la preuve de l'algorithme 1, nous démontrons que l'intégration de la règle de prohibition $\mathcal{F}$ *(start(A)* | $O^{<-d}$ *done(B))* produit une spécification TEFSM sécurisée. Pour cela, nous devons prouver que l'action A ne peut être réalisée avant d unités de temps depuis l'exécution de l'action B. Pour établir cette preuve, nous définissons les variables suivantes :

– t_B : l'instant d'exécution de l'action B.

– k : le temps écoulé depuis l'exécution de l'action B.

– t : une variable temps. Par définition $gck(t) = t$.

– $Ck(t)$: une horloge qui est remise à zéro à chaque exécution d'une occurrence de l'action B.

Nous devons donc établir que pour chaque entier positif k, le prédicat suivant est satisfait :

$$((k < d) \wedge (gck(t) = t_B + k)) \Rightarrow \neg start(A) \qquad (1)$$

Pour prouver que l'action A ne peut pas être exécutée à l'instant $(gck(t) = t_B + k)$, il suffit de monter qu'à cet instant toutes les transitions du système sécurisé Tr dont l'action inclut A sont infranchissables, c.à.d, leurs gardes sont fausses. Dans le système sécurisé, l'algorithme 1 ajoute la

garde ($when\ Ck > d-1$) avant chaque exécution de l'action A. Nous devons donc établir que pour chaque entier positif k la formule suivante :

$$((k < d) \wedge (gck(t) = t_B + k)) \Rightarrow Ck(t_B + k) < d \qquad (2)$$

Par application de la déduction naturelle, nous devons montrer que :

$$Ck(t_B + k) \leq d \qquad (3)$$

sous l'hypothèse : $(k < d) \wedge (gck(t) = t_B + k)$. Pour prouver la formule (3), nous proposons de montrer que :

$$Ck(t_B + k) \leq k \qquad (4)$$

En effet si la formule (4) est vraie, et sachant que $(k < d)$ on peut facilement déduire le but (3). Pour montrer la formule (4), procédons par induction sur k :

1) Cas de base ($k = 0$) : $gck(t) = t_B + 0 = t_B$. On doit prouver que $(Ck(t_B) \leq 0)$. À l'instant t_B, l'action B est exécutée. D'après l'algorithme 1, l'horloge Ck est activée $(set\ Ck := 0)$ à l'issue de l'exécution de l'action B. Par conséquent $Ck(t_B) = 0$, le but est donc vrai car $(0 \leq 0)$.

2) Hypothèse d'induction ($k = n$) : supposons que le but à prouver est vrai jusqu'au rang $k = n$ où $(n < (d-1))$, c.à.d, à l'instant $(gck(t) = t_B + n)$, on a : $(ck(t_B + n) \leq n)$. Supposons également qu'à l'instant $(t_B + n)$, le système est dans l'état S.

3) Étape d'induction ($k = n + 1$) : prouvons le but (4) pour le rang (n+1) $(k = n + 1)$. Nous devons prouver quand l'horloge globale gck est incrémentée d'une unité de temps $(gck(t) = t + n + 1)$, on a $(ck(t + n + 1) \leq n + 1)$. Dans l'intervalle de temps entre $gck(t) = t_B + n$ et $gck(t) = t_B + n + 1$, le système passe de l'état S à l'état S' (égal éventuellement à S) en exécutant un ensemble de transitions T (éventuellement vide). Prouvons donc que :

$$(gck(t) = t_B + n + 1) \Rightarrow ck(t_B + n + 1) \leq n + 1 \ (5)$$

Notons que les seules transitions qui modifient l'horloge Ck sont celles labellisées par l'action B que l'on note Tr_B. Ces transitions ont été modifiées par l'algorithme 1 en ajoutant une action de remise à zéro de Ck. Par conséquent, deux cas sont à distinguer

a) $Tr_B \cap T \neq \emptyset$: au moins une action B est exécutée remettant ainsi l'horloge Ck à zéro : $Ck(t_B + n) = 0$. Comme le temps ne progresse pas durant le franchissement des transitions T, et que les horloges Ck et gck progressent simultanément, nous en déduisons que $ck(t_B + n + 1) = ck(t_B + n) + 1 = 0 + 1 = 1$. Le but (5) est donc vrai car $(1 \leq n + 1)$.

b) $Tr_B \cap T = \emptyset$: durant l'exécution des transitions T, la valeur de Ck reste inchangée car aucune transition ne la modifie. Les horloges Ck et gck progressent simultanément dans l'état S'. Donc, $Ck(t_B + n + 1) = Ck(t_B + n) + 1$. Comme $Ck(t_B + n) \leq n$ (hypothèse d'induction), on en déduit que $Ck(t_B + n + 1) \leq n + 1$.

6. Conclusion et travaux futurs

Dans ce papier, nous avons présenté une approche formelle d'intégration de règles de sécurité temporisées, spécifiées en Nomad, dans une spécification fonctionnelle d'un système décrite par une TEFSM. Cette approche consiste en un ensemble d'algorithmes prouvés permettant d'augmenter la spécification TEFSM du système par diverses règles exprimées en termes d'obligation, d'interdiction et de permissions tout en prenant en compte l'aspect temps. Considérer des règles temporisées

constitue une avancée significative par rapport à l'état de l'art puisqu'aucun des travaux existants ne s'intéresse à ce type de règles. Afin de s'assurer de la faisabilité de notre approche, celle-ci a été expérimentée à plusieurs applications distribuées.

Notons que l'approche d'intégration proposée n'induit pas des modifications importantes sur la TEFSM initiale. En effet par exemple, l'intégration des permissions/prohibitions ne modifie que les prédicats des transitions. Seules les règles d'obligation nécessitent l'ajout de nouveaux états et de transitions supplémentaires. Néanmoins même dans ce cas, la complexité de l'algorithme reste linéaire (en O(n)) puisqu'elle est proportionnelle au nombre de règles à intégrer. De plus, l'expérience a montré que le nombre de règles constituant une politique de sécurité est en général peu important. Cependant, nous pensons que notre approche peut être améliorée en réduisant le nombre d'horloges ajoutées. En effet, des règles dépendantes peuvent être intégrées simultanément en définissant q'une seule horloge. Par exemple, les règles $P(start(A)|O^{-5}done(B))$ et $P(start(A)|O^{-3}done(B))$ peuvent intégrées par la définition d'une horloge unique Ck que le système vérifie ($Ck = 3 \vee Ck = 5$) avant toute exécution de l'action A.

Nous travaillons actuellement sur la génération automatique de cas de test à partir de la spécification TEFSM sécurisée obtenue en utilisant l'outil `TestGen-IF` [CAV 08] développé au sein de notre équipe. Cet outil est basé sur l'algorithme Hit-or-Jump[CAV 99] dédié à la génération de cas de séquences de test à partir de spécification IF. Les cas de test ainsi générés peuvent être utilisés pour le test de conformité d'une implémentation déjà existante du système de gestion de missions vis-à-vis des différentes règles de sécurité intégrées.

7. Bibliographie

[Abo 03] ABOU EL KALAM A., BAIDA R. E., BALBIANI P., BENFERHAT S., CUPPENS F., DESWARTE Y., MIÈGE A., SAUREL C., TROUESSIN G., « Organization Based Access Control », *Policy'03*, June 2003.

[ALA 97] ALAGAR V. S., RAMANATHAN G., « Functional Specification and Proof of Correctness for Time Dependent Behaviour of Reactive Systems », *Formal Asp. Comput.*, vol. 3, n° 3, 1997, p. 253-283.

[ALU 94] ALUR R., DILL D. L., « A Theory of Timed Automata », *Theoretical Computer Science*, vol. 126, n° 2, 1994, p. 183–235.

[BOZ 04] BOZGA M., GRAF S., OBER I., OBER I., SIFAKIS J., « The IF Toolset », *SFM*, 2004, p. 237-267.

[CAV 93] CAVALLI A. R., FAVREAU J. P., PHALIPPOU M., « Formal Methods for Conformance Testing : Results and Perspectives », *Protocol Test Systems*, 1993, p. 3-17.

[CAV 99] CAVALLI A., LEE D., RINDERKNECHT C., ZAIDI F., « Hit-or-Jump : An Algorithm for Embedded Testing with Applications to IN Services », *Formal Methods for Protocol Engineering And Distributed Systems*, Beijing, China, october 1999, p. 41–56.

[CAV 08] CAVALLI A., OCA E. M. D., MALLOULI W., LALLALI M., « Two Complementary Tools for the Formal Testing of Distributed Systems with Time Constraints », *The 12-th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Vancouver, British Columbia, Canada, October 27-29 2008.

[CLA 04] CLARKE E. M., KROENING D., « Tutorial : Software Model Checking », *ICFEM*, 2004, p. 9-10.

[CUP 05] CUPPENS F., CUPPENS-BOULAHIA N., SANS T., « Nomad : A Security Model with Non Atomic Actions and Deadlines », *CSFW*, 2005, p. 186-196.

[CUP 06] CUPPENS F., CUPPENS-BOULAHIA N., GHORBEL M. B., « High-level conflict management strategies in advanced access control models », *Workshop on Information and Computer Security (ICS)*, Timisoara, Roumania, September 2006.

[HER 00] HERZBERG A., MASS Y., MIHAELI J., NAOR D., RAVID Y., « Access Control Meets Public Key Infrastructure, Or : Assigning Roles to Strangers », *IEEE Symposium on Security and Privacy*, 2000, p. 2-14.

[ISO 94] ISO/IEC/9646-1, « Information Technology - Open Systems Interconnection - Conformance testing methodology and framework Part 1 : General Concepts », 1994.

[LI 07] LI K., MOUNIER L., GROZ R., « Test Generation from Security Policies Specified in Or-BAC », *COMPSAC (2)*, 2007, p. 255-260.

[LOB 99] LOBO J., BHATIA R., NAQVI S. A., « A Policy Description Language », *AAAI/IAAI*, 1999, p. 291-298.

[MAL 08] MALLOULI W., MAMMAR A., CAVALLI A. R., « Integration of Timed Security Policies within a TEFSM Specification », Technical Report n° TI-PU-08-868, 2008, Telecom SudParis.

[PAL 02] PALERI V. K., « Automatic Generation of Code Optimizers from Formal Specifications », *The Compiler Design Handbook*, p. 61-97, 2002.

[PAR 97] PARK J.-C., MILLER R. E., « A Compositional Approach for Designing Multifunction Time-Dependent Protocols », *ICNP*, 1997, p. 105-112.

[RIB 01] RIBEIRO C., ZÚQUETE A., FERREIRA P., GUEDES P., « SPL : An Access Control Language for Security Policies with Complex Constraints », *In Proceedings of the Network and Distributed System Security Symposium*, 2001.