Vérification d'exclusions mutuelles par analyse statique de modèles SysML

Ludovic Apvrille et Pierre de Saqui-Sannes

ludovic.apvrille@telecom-paristech.fr

pdss@isae.fr

Résumé : Le langage SysML couplé à des outils de vérification formelle permet d'analyser un modèle de systèmes temps réel pour détecter les erreurs de conception au plus tôt dans le cycle de vie. Dans cet esprit, l'outil TTool met en œuvre trois approches de vérification formelle complémentaires : la détection de comportements non désirés et de violation temporelles, la mise en évidence de failles de sécurité et la recherche d'invariants. Cet article insiste sur la troisième approche et expose les principes d'une analyse statique de modèles SysML ciblée sur la détection d'états ou d'événements en exclusion mutuelle. A partir des diagrammes de blocs et de machines à états du modèle SysML, TTool construit un réseau de Petri et sa matrice d'incidence, point de départ de la résolution d'un système d'équations et d'identification d'invariants. L'optimisation de l'algorithme de Farkas par des heuristiques et la remontée du réseau de Petri au modèle SysML font l'objet d'une attention particulière. L'exemple d'un four à micro-ondes montre que la recherche d'invariants par ce type d'analyse statique peut amener à détecter des problèmes de conception non identifiées par les méthodes de vérification que TTool proposait jusqu'alors. Ajoutons à ceci que TTool est un outil en accès libre qui implante entièrement la recherche d'invariants et d'éléments en exclusion mutuelle sans intervention d'aucun outil externe.

Mots clés : Modélisation, SysML, logiciel en accès libre, vérification formelle, réseaux de Petri, invariants, exclusion mutuelle.

1) Introduction

La complexité des systèmes temps réel justifie le recours à des langages de modélisation et à des outils qui favorisent la détection des erreurs de conception au plus tôt dans le cycle de vie du système. Le langage AVATAR (basé sur SysML [8]) et l'outil TTool présentés notamment dans un précédent numéro de cette revue [1] répondent à cette demande. Le concepteur ou la conceptrice d'un système temps réel peut en effet exprimer les exigences utilisateurs et système, les propriétés à vérifier (unicité d'un jeton sur un réseau local, par exemple), les fonctions et services à offrir et, enfin, l'architecture du système et les comportements des entités qui la composent. L'outil TTool offre à ce concepteur ou cette conceptrice des techniques complémentaires de simulation et vérification formelle de modèles qui servent à valider une architecture et à confronter des comportements à un ensemble d'exigences et propriétés.

En termes de vérification formelle de modèles, l'outil TTool [9] utilise UPPAAL [2] pour détecter des violations temporelles et ProVerif [3] pour mettre en évidence des failles de sécurité. Dans les deux cas, la construction de l'espace d'état du système expose l'utilisateur ou l'utilisatrice de TTool aux défauts connus de cette approche. C'est pourquoi le présent article explore une autre voie de type « analyse statique de modèle ». Il s'agit en l'occurrence de transposer à un modèle SysML un type de vérification appliqué avec succès aux réseaux de Petri, à savoir la « recherche d'invariants ». Contrairement aux approches qui demandent d'exprimer des invariants et qui vérifient ensuite leur (non) satisfaction sur un modèle UML ou SysML, il s'agit bien ici de construire un modèle AVATAR et de rechercher ensuite des invariants pour prouver des propriétés dont l'exemple type est l'exclusion mutuelle d'accès à une ressource

partagée. C'est d'ailleurs sur ce thème de l'exclusion mutuelle entre états ou actions des machines à états d'un modèle AVATAR que cet article met l'accent en prenant pour exemple un système connu de tous : un four à micro-ondes.

L'article rappelle au paragraphe 2 les grandes caractéristiques du langage AVATAR et situe la contribution de cet article parmi les trois approches de vérification de modèles que l'outil TTool implante à ce jour. Le paragraphe 3 explique ensuite comment TTool applique à AVATAR la recherche d'exclusions mutuelles entre états ou action. Le paragraphe 4 traite l'exemple du four à micro-ondes. Enfin, le paragraphe 5 conclut l'article.

2) Le langage AVATAR et la vérification de modèles

Le modèle AVATAR d'un système temps réel regroupe trois familles de diagrammes aptes à couvrir les phases amont du cycle de développement de ce système :

- **Recueil des exigences** : les exigences et les propriétés sont respectivement décrites par des diagrammes d'exigences et TEPE [6], sans omettre de préciser les hypothèses sous lesquelles le modèle est valide.
- **Analyse** : un diagramme de cas d'utilisation est documenté au moyen de scénarios (diagrammes de séquences) et d'organigrammes (diagrammes d'activités).
- **Conception** : un diagramme d'instances de bloc décrit l'architecture du système ; des diagrammes machine à états précisent le comportement interne des blocs de cette architecture.

L'outil TTool propose trois approches de vérification complémentaires pour confronter aux exigences et propriétés, l'architecture et les comportements qui forment la conception. Ces trois approches ont un canevas commun (figure 1) : soit TTool traduit le modèle AVATAR dans un langage formel pour lequel existe déjà un outil de vérification formelle (cas de ProVerif et UPPAAL), soit il implante ses propres techniques de vérification formelle (la recherche d'invariants).

La figure 1 décrit dans ses parties gauche et centrale les deux approches de vérification présentées dans un précédent numéro de cette revue [1]. L'outil UPPAAL (via les automates temporisés) est utilisé pour détecter des erreurs logiques et des violations temporelles tandis que l'outil ProVerif (via le Pi-calculus) est utilisé pour identifier des failles de sécurité, en particulier dans les protocoles cryptographiques.

La partie droite de la figure 1 décrit la contribution de cet article, à savoir la transposition à AVATAR d'une vérification par analyse statique basée sur la technique des « invariants » développée pour les réseaux de Petri. En quelques mots, TTool extrait du modèle AVATAR des informations nécessaires à la construction d'un réseau de Petri pour ensuite construire la matrice d'incidence de ce réseau et résoudre le système d'équations qui en dérive. Le résultat final est une liste d'invariants qui formulent des relations entre états ou actions des machines à états contenus dans les blocs qui composent l'architecture du système. Notons dès à présent que, parmi toutes les formes d'invariants possibles, nous nous intéresserons à ceux qui expriment des exclusions mutuelles entre états ou actions de machines à états.

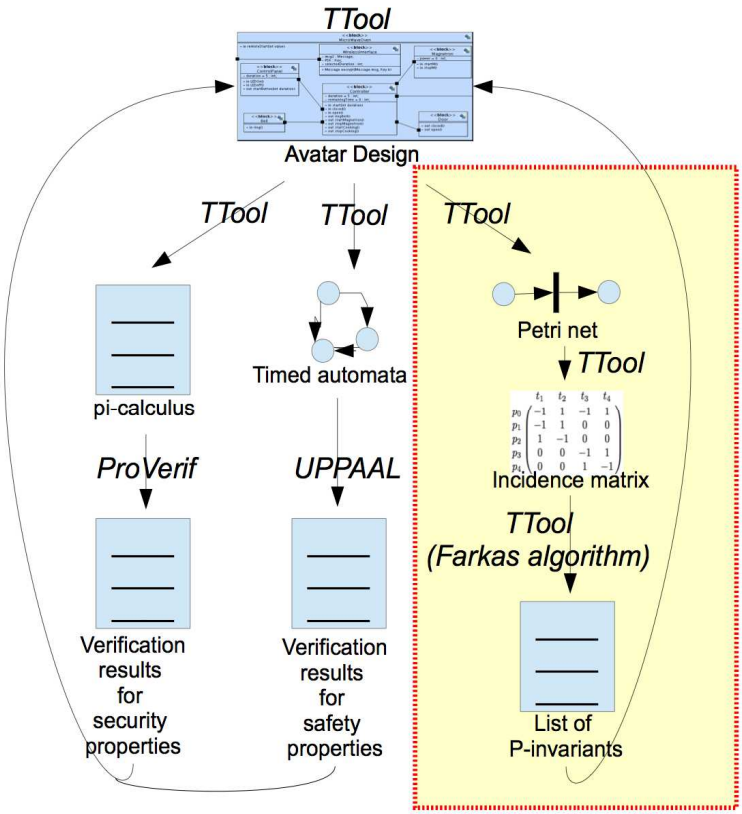

Figure 1. Les trois approches de vérification mises en œuvre par l'outil TTool

3) **Comment TTool recherche les invariants et les états ou actions en exclusion mutuelle**

3.1 Principe général

L'analyse statique d'un modèle de conception AVATAR se fait comme suit:

1. **Traduction vers un réseau de Petri**. Le modèle de conception est traduit en un réseau de Petri. Cette traduction est générique dans la mesure où elle pourrait s'appliquer à tout modèle construit à partir d'éléments communiquant de façon synchrone ou asynchrone, et dont le comportement est décrit à l'aide d'états et de primitives de communications.

2. **Construction de la matrice d'incidence**. Le réseau de Petri est traduit sous la forme d'une matrice d'incidence *A*. Celle-ci représente les évolutions possibles du nombre de jetons des places du réseau en fonction des transitions réalisées [4] [7].

3. **Calcul des invariants**. Afin de trouver les invariants, nous devons résoudre le système d'équations W.A=0 [7]. L'algorithme de Farkas [5] est alors utilisé pour résoudre ce système d'équations, par une technique de triangularisation de la matrice *A*.

4. **Filtrage des invariants**. Nous nous intéressons seulement aux invariants minimaux ; de valeur 1, ces invariants représentent un ensemble de places dont le nombre total de jetons est exactement égal à 1. Nous écartons de plus les invariants qui concernent un seul bloc AVATAR. En effet, un bloc AVATAR ne comportant pas d'activité parallèle, les places construites depuis la machine à états d'un bloc AVATAR sont nécessairement incluses dans un invariant.

5. **Remontée au modèle**. Les invariants sont dessinés directement au niveau du modèle. En particulier, lorsqu'un utilisateur ou une utilisatrice passe son pointeur de souris sur un état d'une machine à états, TTool affiche à côté de cet état la liste des états qui sont en exclusion mutuelle avec lui.

Nous détaillons à présent le processus de traduction d'un modèle AVATAR en un réseau de Petri.

3.2 Processus de traduction

Une conception AVATAR est constituée d'un ensemble de blocs qui communiquent par des signaux en transit sur des ports synchrones ou asynchrones. Le comportement de chaque bloc est décrit à l'aide d'une machine à états SysML. AVATAR support les états hiérarchiques, mais sans opérateur "H" ou "H*". De plus, un bloc AVATAR possède un flux d'exécution unique. Ainsi, les opérateurs f*ork/join* ne sont pas supportés.

A partir d'une conception AVATAR, l'outil TTool construit un réseau de Petri - sans extension temporelle ni jeton coloré - formé des éléments de base suivants :

- Un ensemble de places ;
- Un ensemble de transitions ;
- Un ensemble d'arcs« place vers transition » ou « transition vers place »;
- Une fonction associant un poids à un arc ;
- Une fonction associant un marquage initial à chaque place.

Le processus de traduction fonctionne comme suit :

- Chaque état d'une machine à états AVATAR est traduit par une place de réseau de Petri.
- Les transitions des machines à états sont traduites par des transitions du réseau de Petri sans prendre en compte les conditions et les actions de ces transitions.
- Les opérateurs d'envoi et de réception sont traduits en prenant en compte leur nature synchrone ou asynchrone. La traduction d'une communication synchrone repose sur l'identification de l'ensemble des opérateurs susceptibles d'être impliqués dans cette communication, et sur la génération d'une place par opérateur et d'une transition par couple possible d'opérateurs. Les communications asynchrones prennent en compte les files pleines, soit par écriture bloquante, soit par écrasement. Chacune des politiques a nécessité la création d'un patron sous la forme d'un réseau de Petri équivalent.
- Les intervalles temporels sur les transitions des machines à états ne sont pas pris en compte.

- Les choix sont individuellement traduits par une place, en omettant les conditions associées aux branches de ces choix.

Ainsi, notre processus de traduction ne prend pas en considération la totalité des opérateurs du langage AVATAR. De fait, le processus de traduction vers les réseaux de Petri étend le nombre de chemins possibles du modèle AVATAR initial. Il en résulte que deux états indiqués par TTool comme étant en exclusion mutuelle le sont réellement ; par contre, deux états qui ne sont pas indiqués comme étant en exclusion mutuelle le sont peut-être. Un message du type "D'autres exclusions mutuelles sont possibles sur cet état" avertit l'utilisateur ou l'utilisatrice de TTool lorsque cette situation d'incertitude se présente.

4) **Etude de cas : un four à micro-ondes**

Cette étude de cas illustre d'une part l'intérêt des invariants pour mettre en évidence des situations d'exclusion mutuelle et, d'autre part, l'intégration graphique dans TTool du résultat de la recherche de ces invariants.

4.1 Description du four à micro-ondes

Le four à micro-ondes se démarre à l'aide d'un bouton *start* ou d'une télécommande. Une contrainte forte de sûreté de fonctionnement s'applique : le magnétron doit être systématiquement éteint lorsque la porte est ouverte. Une double contrainte de sécurité s'y ajoute : (1) le four à micro-ondes ne doit pouvoir être commandé que par sa propre télécommande (propriété d'authenticité) et (2) le message de démarrage envoyé par la télécommande ne doit pas pouvoir être espionné.

Ainsi, le four à micro-ondes doit respecter à la fois des contraintes de sûreté de fonctionnement et de sécurité. Les deux aspects sont traités dans le même modèle AVATAR et les preuves formelles de associées peuvent être réalisées avec une approche de type presse-bouton depuis ce même modèle AVATAR.

4.2. Conception

Le four à micro-ondes est construit autour d'une architecture de blocs incluant des ports de communication et des messages échangés entre blocs. Cette architecture comprend un bloc principal *RemotelyControlledMicrowave* (figure 2) qui inclut tous les autres blocs du système : la télécommande *RemoteControl* et le four à micro-ondes *MicroWaveOven*, lui même constitué de plusieurs blocs : un panneau de contrôle *ControlPanel*, une sonnerie *Bell*, une interface réseau *WirelessInterface* pour recevoir les signaux de la télécommande, un magnétron *Magnetron*, une porte *Door* et un contrôleur principale *Controller*.

Le modèle déclare deux types de données : *Key* et *Message*. De plus, une note placée en haut du diagramme exprime à la fois des contraintes et des propriétés de sécurité.

En termes de communication entre blocs, les ports synchrones sont représentés par un carré noir et les ports asynchrones par un carré blanc. Les liens entre blocs permettent de relier les ports entre eux afin de composer les machines à états des différents blocs. La délégation de ports permet à un bloc b1 contenu dans un bloc b2 d'utiliser les

ports de communication de b2. Ainsi, dans notre exemple, le canal asynchrone déclaré au niveau du bloc principal peut être utilisé par tous les blocs.

Enfin, un observateur *ObserverProp1* a été ajouté au modèle de conception. Son utilisation est explicitée dans la suite de l'article.

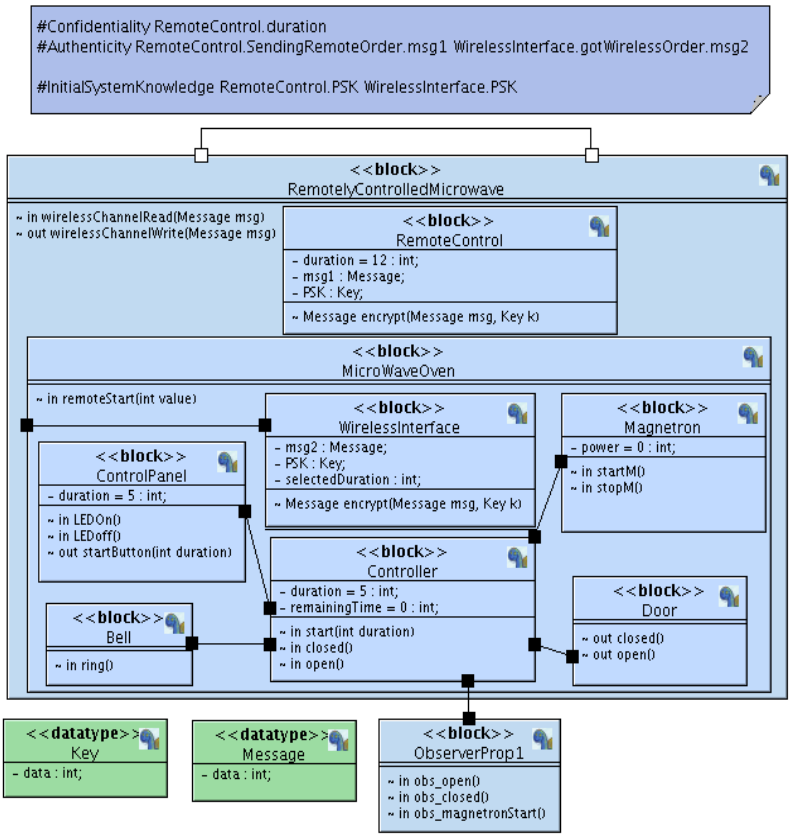

Figure 2. Diagramme d'instances de bloc du four à micro-ondes

Des machines à états dotent tous ces blocs d'un comportement. Ceci vaut autant pour les blocs du système que pour l'observateur.

4.3. Vérification formelle de propriétés de sûreté

La vérification formelle confronte le modèle de conception à des propriétés. Nous reformulons ici l'une des propriétés de sûreté énoncée au paragraphe 4.1 pour affirmer que « le magnétron doit être éteint lorsque la porte est ouverte ». Deux solutions s'offrent à nous pour vérifier cette propriété.

- La première utilise l'observateur *ObserverProp1*. Sa machine à états possède un état d'erreur *error* dont TTool étudie l'accessibilité par un appel à l'outil UPPAAL, ce qui implique une exécution du modèle à la recherche des branches passant par cet état *error*.

- La deuxième consiste à vérifier que l'état *DoorOpened* de *Door* est en exclusion mutuelle avec l'état *Running* de *Magnetron*. L'analyse est cette fois statique et évite donc de construire l'espace d'états du modèle AVATAR. TTool calcule les invariants et les affiche dans l'arbre situé à gauche de l'écran de l'outil (voir

figure 3). La sélection d'un invariant dans cet arbre provoque l'affichage d'une inscription en rouge à côté de chaque opérateur de cet invariant. Les éléments du même invariant sont tous en exclusion mutuelle.

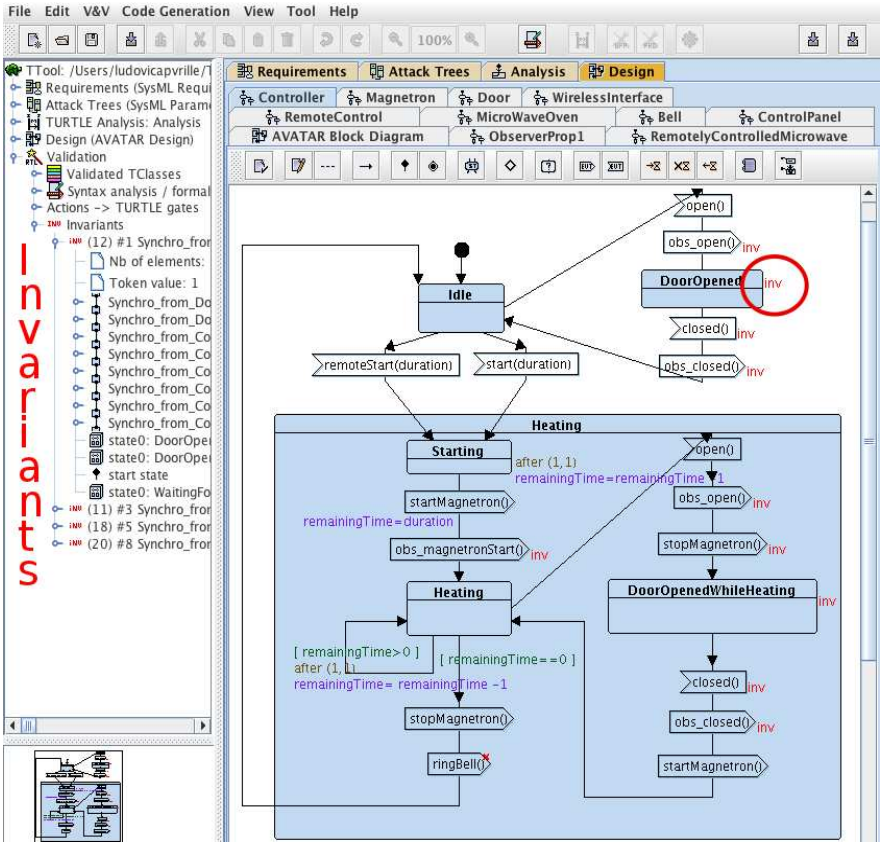

Figure 3. Affichage des invariants dans TTool

L'affichage des invariants ne dispense pas l'utilisateur ou l'utilisatrice de TTool du travail d'interprétation de ces invariants. Dans le cas où la propriété à vérifier peut se traduire par une recherche d'exclusion mutuelle entre états ou actions de machines à états, TTool offre une fonctionnalité supplémentaire : le fait d'amener le pointeur de souris sur un état d'une machine à états d'un bloc suffit pour que TTool énumère, à côté de l'état pointé, les états en exclusion mutuelle avec lui.

Très visuelle, cette fonctionnalité nous permet de comparer aisément deux versions du modèle du four à micro-ondes dont nous ne détaillerons pas ici les machines à états. A la question « Existe-t-il un état en exclusion mutuelle avec l'état *DoorIsOpened* du bloc *Door* ? », l'analyse d'un premier modèle a produit le résultat indiqué par la figure 4. En clair, la propriété de sûreté liant le fonctionnement du magnétron à la fermeture de la porte n'est pas satisfaite. Nous avons alors corrigé le modèle en ajoutant un verrou à la porte : lorsque l'utilisateur ou l'utilisatrice désire ouvrir la porte, le contrôleur désactive la magnétron avant de déverrouiller la porte. La figure 5 montre que l'état *DoorIsOpened* du contrôleur est en exclusion mutuelle avec l'état *Running* du magnétron ; ainsi, le magnétron ne peut pas être en marche lorsque la porte est ouverte. La propriété de sûreté recherchée est cette fois satisfaite.

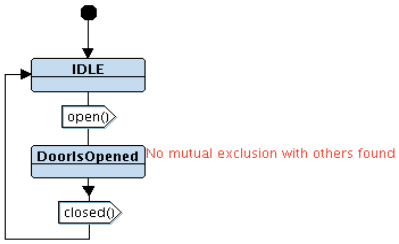

Figure 4. Exclusions mutuelles de l'état*DoorIsOpened* du bloc *Door* (premier modèle)

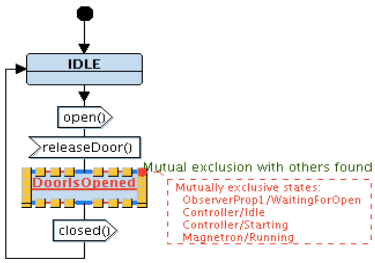

Figure 5. Exclusions mutuelles de l'état*DoorIsOpened* du bloc *Door* (deuxièmemodèle)

5) **Conclusion**

Le logiciel TTool supporte plusieurs profils UML temps réel et un dialecte temps réel de SysML dénommé « AVATAR ». Les premiers développements de TTool ont donné aux utilisateurs de ces langages des simulateurs de modèle, des modules de vérification de sûreté et de sécurité respectivement appareillés à UPPAAL et ProVerif, et des générateurs de code System C, Java et C/POSIX. TTool est un logiciel en accès libre, disponible pour les plateformes Linux, Mac OS et Windows. Il est été utilisé pour l'enseignement, les projets de recherche et les études industrielles.

L'outil TTool décrit en [1] conditionnait la vérification d'un modèle AVATAR à la possibilité de l'exécuter. Cet article prend le contre-pied de [1] en proposant une analyse statique des modèles AVATAR. L'exemple du four à micro-ondes montre d'ailleurs que les deux approches a priori « opposées » sont utilement complémentaires.

En pratique, l'outil TTool traduit un modèle AVATAR dans un réseau de Petri pour calculer ensuite la matrice d'incidence de ce réseau, résoudre le système d'équations qui en découle et énumérer les invariants. L'algorithme de Farkas et les heuristiques développés par le premier auteur de cet article, permettent de calculer rapidement ces invariants.

Automatiser ce calcul d'invariants était un premier pas. Encore fallait-il que l'interprétation des invariants ne devienne pas une tâche rédhibitoire pour la majorité des utilisateurs de langage de type UML/SysML. L'interface présenté dans cet article résulte d'une double volonté : afficher les résultats de la recherche d'invariants sur le modèle AVATAR lui-même et assister encore plus l'utilisateur ou l'utilisatrice lorsque la propriété à vérifier peut se traduire en un problème d'exclusion mutuelle entre états ou actions des machines à états du modèle AVATAR.

L'outil TTool affiche parfois des messages indiquant qu'il ne peut pas déterminer si deux états sont en exclusion mutuelle ou non. Cela tient aux heuristiques ajoutées à l'algorithme de Farkas. Au titre des limitations actuelles, ajoutons aussi que le recours à des réseaux de Petri plus riches sémantiquement (réseaux de Petri colorés, par exemple) permettrait de traduire de façon plus complète les modèles AVATAR.

6) **Bibliographie**

[1] L. Apvrille, P. de Saqui-Sannes, "AVATAR/TTool : un environnement en mode libre pour SysML temps réel", Revue Génie Logiciel, Vol. 58, pp.22-26, Sept. 2011.

[2] J. Bengtsson, W. Yi., "Timed Automata : Semantics, Algorithms and Tools". In Lecture Notes on Concurrency and Petri Nets, pp.87–124, LNCS 3098, Springer (2004).

[3] B. Blanchet, "Using Horn Clauses for Analyzing Security Protocols". In Formal Models and Techniques for Analyzing Security Protocols, Cryptology and Informa-tion Security Series Vol. 5 pp.86–111, IOS Press (2011).

[4] M. Diaz, "Les réseaux de Petri :modèlesfondamentaux, Traité IC2, SérieInformatique et systèmesd'information", Hermès-Lavoisier, mai 2001.

[5] J. Farkas, « Theorie den einfachenUngleichungen ». Journal fürdie reine undangewandteMathematik (Crelle's Journal), issue 124, pp.1{27 (1902).

[6] D. Knorreck, L. Apvrille, P. de Saqui-Sannes, "TEPE: A SysML Language for Time-Constrained Property Modeling and Formal Verification", Proceedings of the Third IEEE International Workshop UML and Formal Methods (UMLFM'2010), Shanghai, China, November, 2010.

[7] T. Murata, “Petri Nets: Properties, Analysis and Applications”. In: Proceedings of the IEEE, Vol. 77(4) pp.541-580 (1989).

[8] SysML, OMG Systems ModelingLanguages, OMG http://www.sysmlomg.org/.

[9] TTool, outil UML/SysML pour la modélisation de systèmes complexes, http://ttool.telecom-paristech.fr/.