# AVATAR/TTool
# Un environnement open-source pour SysML temps réel

L. Apvrille (1), P. de Saqui-Sannes (2)

(1) Institut Telecom, Telecom ParisTech, LTCI CNRS,
2229, route des Crêtes, B.P. 193, F-06904 Sophia-Antipolis, France
ludovic.apvrille@telecom-paristech.fr

(2) CNRS ; LAAS ; 7 avenue du Colonel Roche, F-31077 Toulouse, France
Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse France
pdss@isae.fr

**Résumé.** Cet article partage une expérience de modélisation de systèmes temps réel s'appuyant sur le langage AVATAR dérivé de SysML, l'outil *open-source* TTool et la méthode associée. AVATAR enrichit SysML par son langage TEPE d'expression de propriétés. Sa sémantique formelle par traduction vers les automates temporisés autorise les preuves de sureté et celle obtenue par traduction vers le pi-calcul rend possible les preuves de sécurité. Exécutable sur Linux, MacOS et Windows, l'outil *open-source* TTool mise quant à lui sur l'accessibilité aux non spécialistes pour offrir un éditeur de diagrammes, un simulateur de modèles, une interface aux outils de vérification formelle UPPAAL et ProVerif, et un générateur de code C Posix. L'article illustre l'approche AVATAR sur une étude de cas pédagogique et recense les projets académiques et industriels qui font appel à TTool.

**Mots-clés** : Modélisation, SysML, Simulation de modèle, Vérification de modèle, Outil open-source, Temps réel.

## 1. La méthode AVATAR supportée par l'outil TTool

Au contraire des langages UML [9] et SysML [8] normalisés par l'OMG, AVATAR est un environnement qui inclut une méthode outillée, adaptée aux systèmes temps réel et distribués et concentrée sur les phases amont du cycle de vie. Cette méthode – supportée par l'outil open-source TTool [11] - comprend les étapes suivantes :

- **recueil des exigences** à partir du cahier des charges et identification de propriétés temporelles et de sécurité ;
- **identification des hypothèses** de modélisation sous lesquelles le modèle fait des abstractions acceptables ;
- **analyse** guidée par les cas d'utilisation et documentée par des scénarios ;
- **conception** d'une architecture de blocs communicants ;
- **explicitation des propriétés** à vérifier sur cette architecture ;
- **conception détaillée et incrémentale** des comportements internes des blocs de l'architecture avec simulation de modèle et vérification formelle du point de vue « sûreté » (logique du comportement, respect des contraintes temporelles) et « sécurité » (détection des failles de sécurité) ;
- **prototypage** sur la base d'un code dérivé du modèle AVATAR (ce point n'est pas traité dans cet article).

## 2. Une étude de cas pédagogique

L'article déroule la méthode sur un exemple académique, à savoir un four à micro-ondes de type « réchauffe plat ». Ce four rudimentaire démarre à l'appui sur un bouton situé sur le four et chauffe pendant la durée associée à ce bouton. L'adjonction d'une télécommande de démarrage avec canal de communication sécurisé donne prétexte à illustrer la vérification de propriétés de sécurité. Les autres composants du four sont bien plus classiques et amènent à vérifier des propriétés de sûreté de type « Le contrôleur du four interrompt le fonctionnement du magnétron quand la porte est ouverte pendant la cuisson ».

## 3. RECUEIL DES EXIGENCES

Le langage AVATAR reprend à son compte le diagramme d'exigences de SysML [9] pour décrire une structure arborescente d'exigences exprimées en langage naturel. Ce diagramme décompose des exigences complexes en exigences élémentaires de type utilisateur ou système dont on peut dériver des exigences techniques.

La figure 1 reproduit des fragments du diagramme d'exigences développé pour le four à micro-ondes et illustre l'association de propriétés de sûreté et de sécurité à des exigences. Les propriétés qui seront modélisées formellement après la conception sont ici décrites sous la forme de *testcases* SysML stéréotypés <<Property>> : ces propriétés sont à cette étape de simples identifiants.

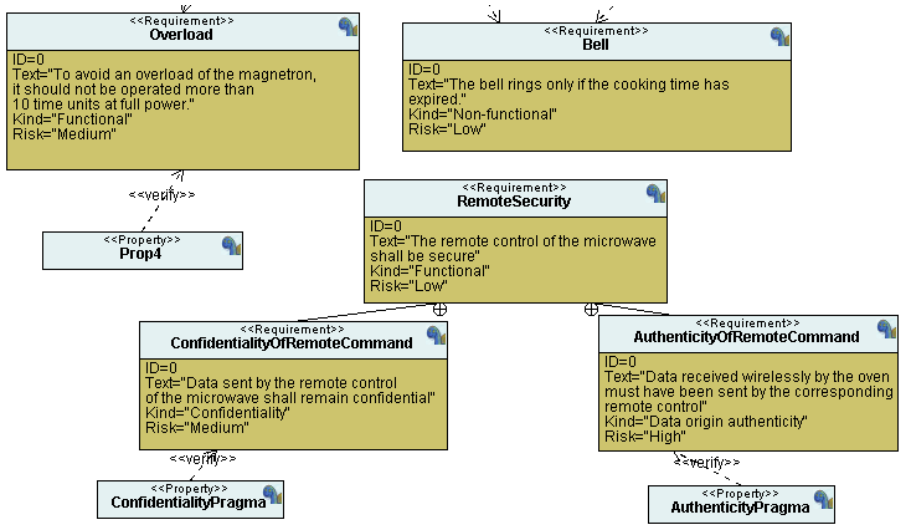


Figure 1 – Extrait du diagramme d'exigences

## 4. HYPOTHÈSES DE MODÉLISATION

Un modèle est une abstraction d'un système réel qui ignore nécessairement certains détails d'implantation voire des pans entiers des fonctionnalités que ce système doit offrir. Il faut donc exprimer clairement au travers d'une liste des hypothèses simplificatrices sous lesquelles le modèle est valide.

Les hypothèses sous-jacentes au modèle du four s'expriment au travers de notes insérées dans le diagramme d'exigences. Ces hypothèses concernent les entrées, les sorties, les conditions d'initialisation, les conditions de terminaison et enfin les fonctions du contrôleur de four à micro-ondes.

## 5. ANALYSE

Le diagramme de cas d'utilisation délimite le périmètre du système et relie les fonctions que celui-ci doit implanter à des acteurs humains et matériels externes. La documentation de cas d'utilisations par des diagrammes de séquences répond bien aux besoins de description d'un système communiquant soumis à des contraintes de temps.

Le modèle du four à micro-ondes inclut un diagramme de cas d'utilisation extrêmement simple, limité au cas d'utilisation « Heat Plate » documenté par deux diagrammes de séquences. Le premier diagramme traite le système comme une boite noire qui interagit avec ses acteurs externes. Le second éclate le premier pour faire apparaître la structure interne du système, facilitant la transition vers la définition d'une architecture par un diagramme de blocs.

## 6. CONCEPTION, CONCEPTION DÉTAILLÉE ET PROPRIÉTÉS

Un diagramme de blocs décrit une hiérarchie de blocs dotés de ports (asynchrones ou synchrones) nécessaires aux opérations de composition et renommage. Les attributs d'un bloc caractérisent son état et les méthodes participent à la définition de son comportement. Des blocs dits cryptographiques définissent en outre des fonctions cryptographiques de base : cryptage et décryptage symétrique et asymétrique, calcul de MAC, etc.

Le diagramme de la figure 2 décrit un système qui relie un four à micro-ondes à une télécommande via une communication sans fil sécurisée. Le four est décomposé pour faire apparaître un contrôleur relié à des capteurs

(boutons, interface avec la télécommande) et des actionneurs (magnétron). La liaison sans fil entre la télécommande et le four est modélisée par un canal public (tous les autres canaux sont privés, ce qui revient à dire qu'un attaquant ne peut écouter les données qui y transitent). Deux types de données sont définis (Message, Key). En complément, un pragma *#InitialCommonKnowledge* specifie les valeurs partagées par les blocs avant l'exécution du système. En l'occurrence, la télécommande et le récepteur sans-fil du four partagent une clé nommée PSK (Pre-Shared Key).

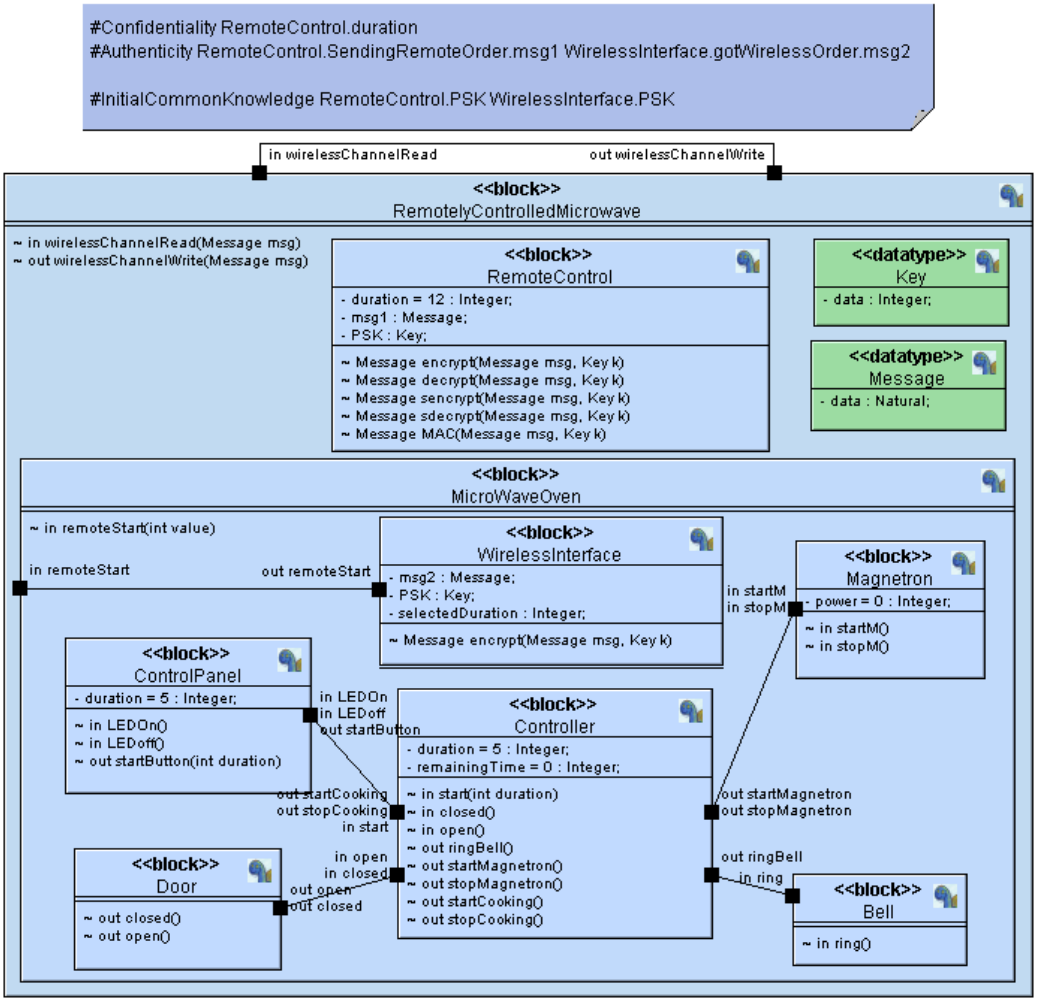


Figure 2 – Diagramme de blocs

Un diagramme machine à états AVATAR décrit une machine à états communicante avec manipulation de variables et gestion de temporisation, temps de calculs et retards sur le déclenchement de transitions. La figure 3 décrit le comportement d'un réchauffe plat à micro-ondes, robuste aux ouvertures de porte en cours de fonctionnement et respectueux de la durée de cuisson transmise par ailleurs par la télécommande ou le bouton de réglage implanté sur le four.

Les propriétés de sûreté sont décrites à l'aide de diagrammes paramétriques SysML, étendus avec de nouvelles contraintes, et renommés TEPE [6]. Ces diagrammes mettent en évidence des relations logiques entre valeurs d'attributs et occurrences de signaux dans le système. Avec cette approche, nous avons par exemple modélisé la propriété suivante : "Le magnétron ne doit pas fonctionner plus de 10 unités de temps d'affilé à pleine puissance".

Les propriétés de sécurité sont quant à elles spécifiées sous la forme de *pragmas* définis dans une note du diagramme de blocs :

- *#Confidentiality* spécifie une propriété de confidentialité. Ici, on estime que la durée de cuisson envoyée par la télécommande au four est une donnée confidentielle.

- *#Authenticity* spécifie l'authenticité d'un message envoyé par un bloc vers un autre bloc. Dans notre système, nous utilisons ce pragma pour spécifier que les messages reçus par le four doivent avoir été envoyés précédemment par la télécommande de ce four, et donc pas par un attaquant.

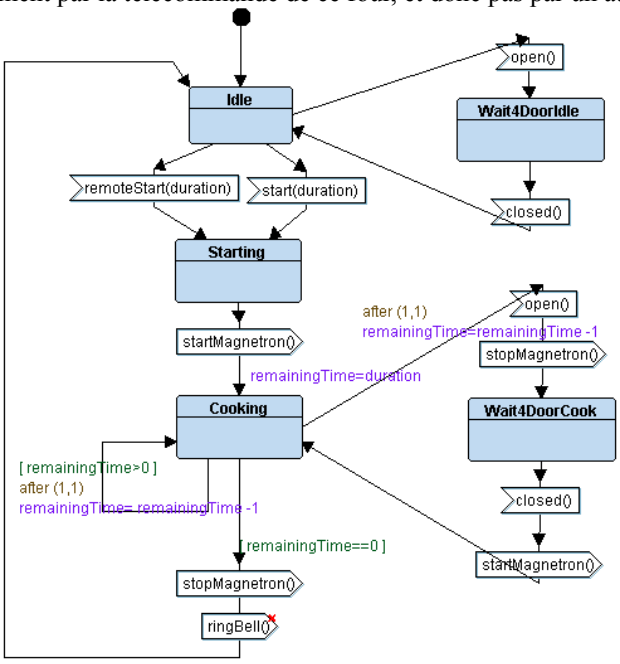


Figure 3 – Diagramme de machine à états du contrôleur

Combiner simulation et vérification formelle de modèle revient à mettre en place un premier filtre d'erreurs par analyse d'un modèle dont on peut dériver ensuite un code exécutable. Cette facilité de prototypage ne sera pas présentée dans la suite de cet article. Le paragraphe qui suit concentre la discussion sur la détection au plus tôt des erreurs de conception par simulation d'un modèle AVATAR et vérification de propriétés au travers des outils compagnons de TTool, à savoir UPPAAL [13] et ProVerif [10].

## 7. Analyse des modèles

Le simulateur intégré à TTool permet de tracer une exécution partielle du comportement du système sous forme d'un diagramme de séquences. La figure 4 en donne un exemple lié à l'ouverture de la porte du four en cours de cuisson. Une interface graphique permet de contrôler cette simulation de façon graphique, tout en animant les diagrammes SysML. Aussi, il est possible d'ajouter des points d'arrêts (*breakpoints*) directement sur les modèles SysML, en cours de simulation. Le simulateur permet aussi de revenir en arrière dans les transactions, de visualiser les valeurs courantes des variables des blocs, etc.

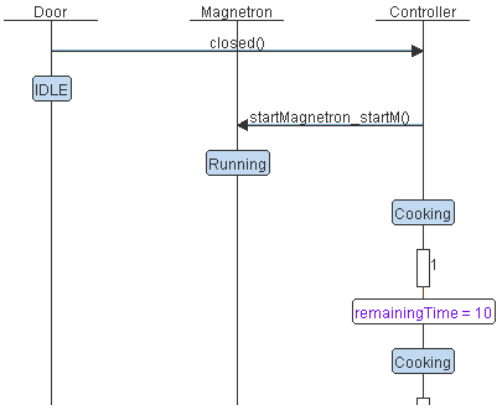


Figure 4 – Trace de simulation

L'outil TTool est capable de traduire un diagramme de blocs AVATAR et les diagrammes de machines à états associés dans un automate temporisé au format accepté par l'outil UPPAAL. Les informations liées à la sécurité sont ignorées lors de la génération de ces automates. Le vérificateur UPPAAL de modèle permet de confronter un modèle de conception aux exigences et propriétés définies dans les étapes précédentes. Sur le modèle du four à micro-ondes, la propriété de sûreté liée à puissance du magnétron a ainsi pu être vérifiée. Le même modèle AVATAR a ensuite été traduit par TTool dans le format accepté par l'outil ProVerif (Pi-calcul). Les deux propriétés de sécurité pré-citées (confidentialité, authenticité) ont été prouvées comme étant vérifiées par le modèle.

Dans les deux cas de vérification, aucune maitrise des langages ou outils de vérification formelle sous-jacents n'est nécessaire puisque TTool implante une approche presse-bouton.

## 8. Projets académiques et industriels utilisant TTool

L'étude de cas à vocation pédagogique traitée jusqu'ici n'est qu'un exemple d'utilisation d'AVATAR et TTool pour l'enseignement de modélisation de systèmes temps réel et distribués. Plusieurs expériences avec des étudiants de niveau Master en école d'ingénieur ou IUT ont montré la facilité d'apprentissage de l'outil et la facilité de détacher les étudiants de spécificités d'AVATAR ou TTool pour se concentrer sur certains fondamentaux tels que la capacité à faire des abstractions et l'application d'une méthode réutilisable dans d'autres environnements SysML temps réel. Des projets étudiants dédiés par exemple à la modélisation d'un pacemaker ou d'un drone confirment la facilité de prise en main de TTool.

Cet outil dont la première version remonte à 2003 capitalise sur les profils UML temps réel TURTLE [3] et DIPLODOCUS [4] et CTTool [1]. TURTLE a notamment été utilisé pour la preuve de protocoles de diffusion de données (projet européen Maestro, UDCast), pour la preuve de propriétés temporelles dans le cadre du projet national SAFECAST, et pour la preuve de propriétés de sûreté de gestionnaire d'énergie des plates-formes matérielles (DoceaPower). DIPLODOCUS a été développé en partenariat avec Texas Instruments et Freescale, pour lequel il a été utilisé pour la conception de systèmes-sur-puce pour terminaux mobiles, en particulier pour des applications multimédia : codage et décode vidéo et pour stations de base (couches MAC de LTE). AVATAR est actuellement utilisé dans le cadre du projet européen EVITA qui s'intéresse notamment à la preuve de propriétés de sécurité dans les systèmes embarqués automobiles.

## 9. Positionnement par rapport aux travaux du domaine

L'outil Rhapsody [11] permet la vérification formelle de propriétés en s'appuyant sur UPPAAL, sans distinction explicite entre exigences et propriétés, ni expression formelle de ces propriétés dans des diagrammes spécifiques comme c'est le cas en AVATAR. De plus, Rhapsody contraint à saisir les propriétés sous forme de formules de logique temporelle alors que TTool permet la preuve de propriétés d'accessibilité d'actions avec un simple click droit sur ces actions.

L'environnement OMEGA2 [7] possède une connexion forte avec Rhapsody car il implémente une sémantique similaire. OMEGA2 supporte de plus les diagrammes d'exigences SysML, alors qu'ARTISAN [5] les étend pour supporter les flux continus. De plus, les modèles ARTISAN supportent des probabilités et des régions interruptibles, deux concepts non supportés par AVATAR, qui supporte simplement l'interruption d'états hiérarchiques.

TTool supporte une petite dizaine de profils UML, en particulier TURTLE [3], DIPLODOCUS [4] et AVATAR. Il peut-être facilement étendu pour en supporter davantage : quelques jours de développement suffisent en général. Mais, à la différence de TOPCASED [2], TTool ne supporte pas la méta-modélisation. Il n'est pas non plus lié à Eclipse, ce qui le rend indépendant de tout autre environnement.

Finalement, TTool offre un environnement multi-profils qui met en œuvre la vérification formelle de propriétés par une approche de type presse-bouton. En particulier, le nouvel environnement AVATAR, basé sur SysML, couvre les phases de recueil d'exigence jusqu'au prototypage, tout en offrant la possibilité de vérifier formellement depuis un même modèle des propriétés de sûreté et de sécurité. Cette dualité sûreté - sécurité nous parait d'autant plus importante avec l'interconnexion omniprésente des systèmes embarqués de nouvelle génération.

## 10. Conclusion et perspectives

Le langage de modélisation AVATAR appuyé sur SysML, l'outil open-source TTool et la méthode déroulée dans l'article définissent un cadre avenant mais formel de spécification, conception et prototypage de systèmes temps réel, distribués, et sécurisés.

Le four à micro-onde qui sert d'exemple dans cet article nous permet d'illustrer les différentes phases de modélisation et de vérification, en en particulier l'expression de propriétés de sûreté de fonctionnement et de sécurité, et leurs preuves formelles.

Nos travaux futurs concernent le renforcement des propriétés de sûreté et de sécurité modélisables et prouvables avec notre environnement. A titre d'exemple, nous travaillons à la définition de preuve de propriétés d'intégrité et de *freshness*. Nous projetons par ailleurs d'inclure à TTool un assistant méthodologique qui facilité la prise en main de l'outil TTool et du langage AVATAR.

## BIBLIOGRAPHIE


[1] S. Ahumada, L. Apvrille, T. Barros, A. Cansado, E. Madelaine, and E. Salageanu. Specifying Fractal and GCM Components With UML. In Proceedings of the XXVI International Conference of the Chilean Computer Science Society (SCCC'07), Iquique, Chile, November 2007.

[2] M. Audrain and B. Marconato. Topcased 3.4 tutorial - requirement management. In http://www.topcased.org/index.php? documentsSynthesis=y&Itemid=59, 2010.

[3] L. Apvrille, J.-P. Courtiat, C. Lohr, and P. de Saqui-Sannes. TURTLE: A real-time UML profile supported by a formal validation toolkit. In IEEE transactions on Software Engineering, volume 30, pages 473–487, July 2004.

[4] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert, and R. Pacalet. A UML-based environment for system design space exploration. In Proceeedings of the 13th IEEE International Conference on Electronics, Circuits and Systems (ICECS '06), pp.1272–1275, Nice, France, December 2006.

[5] M. Hause and J. Holt. Testing solutions with UML/SysML.
In http://www.artistembedded.org/docs/Events/2010/UML AADL /slides/Session1 Matthew Hause.pdf, 2010.

[6] Daniel Knorreck, Ludovic Apvrille, and Pierre de Saqui-Sannes. Tepe: a SysML language for time-constrained property modeling and formal verification. In Proceedings of the third IEEE International workshop UML and Formal Methods - ULM&FM'2010, Shanghaï, China, November 2010.

[7] Iulian Ober and Iulia Dragomir. OMEGA2: A new version of the profile and the tools. In UML&AADL'2009 - 14th IEEE International Conference on Engineering of Complex Computer Systems, pages 373–378, Potsdam, Germany, June 2009.

[8] OMG. Systems Modeling Language (SysML). In http://www.sysml.org, 2011.

[9] OMG. Unified Modeling Language (UML). In http://www.uml.org, 2011.

[10] Proverif. In http://www.proverif.ens.fr/, 2011.

[11] E. C. da Silva and E. Villani. Integrando SysML e model checking para V&V de software crítico espacial. In Brazilian Symposium on Aerospace Engineering and Applications, São José dos Campos, SP, Brazil, September 2009 (in Portuguese).

[12] TTool, The TURTLE Toolkit. In http://labsoc.comelec.telecom-paristech.fr/ttool, 2011.

[13] UPPAAL. In http://www.uppaal.com, 2011.