Chapitre 6

Analyse des exigences

6.1. Introduction

En amont du cycle de vie d'un système complexe, le recueil des exigences joue un rôle déterminant pour les phases d'analyse, de conception, de réalisation et de maintenance qui lui font suite. La traçabilité des exigences au long du cycle de vie est ainsi devenue une préoccupation majeure des praticiens du domaine de l'ingénierie système. Ces mêmes praticiens reconnaissent l'importance d'une analyse d'exigences menée au plus tôt dans le cycle de vie d'un système.

Outre les relectures croisées qui permettent d'apprécier le degré de cohérence des exigences entre elles, il est un type d'analyse essentiel pour détecter au plus tôt les erreurs de conception et réduire le coût de développement et de test du système : la confrontation d'un modèle de conception aux exigences. En contexte SysML, ce type de confrontation met en balance une architecture d'instances de blocs et les comportements des entités qui la composent, avec les exigences exprimées dans les diagramme d'exigences et les cas d'utilisation.

Pour confronter une conception à des exigences, nous avons besoin de diagrammes d'architecture et de comportements à la fois exécutables et non ambigus. C'est une des clés de l'approche de simulation et vérification formelle mise en oeuvre dans ce chapitre.

Chapitre rédigé par Ludovic APVRILLE et Pierre DE SAQUI-SANNES.

La levée de toute ambigüité d'interprétation des diagrammes de conception (instance de blocs et machines à états) d'une part et la traduction de ces diagrammes dans un formalisme déjà outillé en matière de vérification formelle d'autre part, demande de doter ces diagrammes de conception SysML d'une sémantique formelle. C'est chose faite avec le langage AVATAR [APV 11a] que nous utilisons dans ce chapitre. AVATAR est le langage supporté par l'outil TTool [APV 11b]. Logiciel libre, TTool est interfacé avec l'outil UPPAAL [BEN 04] en vue de vérifier la logique et la temporalité de conceptions modélisées en SysML.

Ce chapitre réutilise le modèle SysML de pacemaker du chapitre précédent et l'étend en vue d'une analyse logique et temporelle. Nous étendons le diagramme d'exigences SysML original et lions formellement les exigences aux propriétés à vérifier. Le langage TEPE d'expression de propriétés et son intégration à la méthode orientée "vérification de modèle" supportée par l'outil TTool, sont au coeur de la présentation de l'environnement TTool/AVATAR décrit par la section 6.2. Ce modèle AVATAR du pacemaker comporte une partie « documentaire » formée des diagrammes d'exigences et d'analyse et une partie rendue « exécutable » que forment les diagrammes de conception (architecture et comportement). La mise au point de ces diagrammes de conception fait appel au simulateur intégré à l'outil TTool. Cette fonctionnalité de débogage est présentée à la section 6.3. et illustrée à l'aide des diagrammes les plus significatifs du modèle AVATAR du pacemaker. La section 6.4 enchaîne sur la vérification formelle. Parmi les modes de fonctionnements du pacemaker, ce chapitre retient le mode VVI pour la concision des modèles et des traces de simulation. La vérification formelle du même modèle illustre l'intérêt de l'approche "presse bouton" mise en oeuvre par TTool sans occulter le problème de la remontée d'informations d'UPPAAL vers TTool. Les diagrammes d'expressions de propriété (langage TEPE) permettent de décrire des propriétés et de les traduire en observateurs utiles pour guider la vérification formelle. L'ensemble forme un environnement de simulation et vérification formelle de modèles SysML temporisés que la section 6.5 compare à d'autres outils et approches SysML. La section 6.6 synthétise les contributions du chapitre et ouvre des perspectives sur la transition entre le modèle SysML/AVATAR et le modèle AADL du même pacemaker.

6.2. Le langage AVATAR et l'outil TTool

L'approche AVATAR comporte une méthode (section 6.2.1), un langage basé SysML (section 6.2.2), enrichi par TEPE (section 6.2.3) et supporté par l'outil TTool (section 6.2.4).

6.2.1. *Méthode*

Le langage SysML normalisé à l'OMG est une notation et en aucun cas une méthode. A l'origine, le langage AVATAR se trouvait dans la même situation. Nous lui avons adjoint une méthode applicable à une large gamme de systèmes. Les sept étapes s'enchaînent dans cet ordre avec des re-bouclages possibles :

1. Recueil des exigences,
2. Expression des hypothèses simplificatrices,
3. Analyse guidée par les cas d'utilisation documentés par des scénarios et des enchaînements d'activités,
4. Conception en termes d'architecture et de comportements,
5. Expression des propriétés à vérifier,
6. Confrontation des diagrammes de conception aux exigences et propriétés par combinaison de techniques de simulations et de vérifications formelles.
7. Expression des allocations entre éléments de modèles et construction des matrices de traçabilité.

Un rebouclage peut être effectué entre n'importe quel étape i vers une étape précédente j avec $j < i$, même si l'on préferera toujours aller le plus loin possible - c'est à dire jusqu'à la derniere étape 7 - dans chaque itération de la méthodologie.

6.2.2. *Diagrammes AVATAR et norme SysML*

Le langage AVATAR réutilise les neuf diagrammes SysML, leur syntaxe et leur sémantique dans la mesure où celle-ci est définie de manière suffisamment précise dans la norme [OMG 11]. Le langage AVATAR est de plus outillé et associé à une méthode qui répond aux besoins d'une large classe de système temps réel. Cette méthode oriente la manière dont nous allons maintenant décrire les diagrammes AVATAR et leur positionnement par rapport à la version normalisée de SysML.

En amont de la méthode, le recueil des exigences organise ces dernières dans une structure arborescente qui montre leurs attributs, leurs inter-relations et leurs liens avec d'autres éléments du modèle, y compris - en AVATAR - des références aux propriétés à vérifier formellement sur les diagrammes d'architecture fonctionnelle ou organique. Outre certains attributs - de sécurité par exemple - associés à chaque exigence, AVATAR se démarque de SysML par la possibilité de référencer des propriétés qui ont vocation à être ultérieurement décrites de façon plus précise et formelle avec des élements de la conception.

La phase d'analyse s'appuie classiquement sur un diagramme de cas d'utilisation documenté par des diagrammes de séquences et des diagrammes d'activité. AVATAR

réutilise le diagramme IOD (Interaction Overview Diagram) d'UML 2 pour structurer les scénarios que décrivent les différents diagrammes de séquences. Ces derniers supportent non seulement les opérateurs de temps relatif et absolu mais aussi la manipulation de temporisateurs au travers d'opération d'armement, test d'expiration et réinitialisation. En plus de gérer le temps, il faut représenter le caractère réactif des systèmes temps réel. Pour cela, les diagrammes de séquences AVATAR modélisent des interactions asynchrones aussi bien que synchrones. Notons cependant que les flux continus ne sont pas supportés.

Cette restriction se retrouve dans les diagrammes de conception qui définissent les entités du système tant en terme d'architecture que de comportement. En termes d'architecture, AVATAR utilise un seul et même diagramme pour représenter d'une part les blocs (et donc le typage des éléments d'architecture) et d'autre part, les instances de blocs (qui sont sur le même diagramme composées au sens de la composition entre machines à états temporisées communicantes). Au niveau des comportements, les diagrammes machines à états supportent les intervalles temporels (clause 'after[min, max]') et les opérations sur les temporisateurs déjà énumérées pour les diagrammes de séquences. AVATAR fait ici l'hypothèse d'un système "fermé", c'est à dire que les échanges de signaux avec l'environnement du système doivent être explicitement modélisés dans la conception.

Cet ensemble de diagrammes sert de support à la méthode présentée au paragraphe précédent, avec le souci de pouvoir valider l'architecture de conception et les automates de comportements. Pour cela, les diagrammes d'instances de blocs et de machines à états sont dotés d'une sémantique formelle permettant leur traduction vers des automates temporisés que supportent l'outil UPPAAL. Ceci ajoute de la formalisation à SysML.

6.2.3. *Le langage TEPE d'expression de propriétés*

La formalisation du langage AVATAR et la passerelle établie entre les outils TTool et UPPAAL, participent de la mise en oeuvre d'une approche de vérification formelle. Cette approche fait classiquement référence à la notion de "propriété à vérifier". Dérivées le plus souvent des exigences, ces propriétés peuvent s'exprimer dans un langage formel étranger à SysML. Cette approche où les propriétés se retrouvent de facto exprimées dans un formalisme autre que SysML est courante [FAR 06] [LEB 10] [SAT 10]. A contrario, l'introduction des propriétés dans le modèle SysML du système est une spécificité d'AVATAR qui étend ainsi les diagrammes paramétriques de SysML afin de définir un langage graphique d'expression de propriétés : TEPE (Temporal Property Expression languagE) [Kno 11a].

Le langage SysML utilise les diagrammes paramétriques pour établir des relations entre attributs des blocs de l'architecture et plus généralement pour définir des équations impliquant des éléments d'un ou plusieurs diagrammes d'un modèle SysML.

Ainsi, nous pensons que ces diagrammes paramétriques offrent un excellent cadre structurant pour exprimer des propriétés à vérifier. Avec TEPE, les propriétés sont décrites d'une part au travers de relations logiques et temporelles entre occurrences d'événements et d'autre part à l'aide de relations entre valeurs des attributs de blocs. De ces deux types de relations est né le langage TEPE d'expression de propriétés temporelles qui gagne en facilité d'utilisation par rapport aux formules de logiques, sans pour autant perdre en pouvoir d'expression. La sémantique formelle de TEPE est par ailleurs basée sur les logiques temporelles *Metric Temporal Logic* (MTL) [KOY 92] et *Fluent Linear Temporal Logic* (FLTL) [LET 05].

6.2.4. *TTool*

Le langage AVATAR est intégralement supporté par l'outil libre TTool citettool développé pour Linux, Windows ou MacOS. L'installation par défaut de TTool permet de disposer des éditeurs de diagrammes et du simulateur. TTool implante des passerelles vers trois outils développés par d'autres laboratoires : UPPAAL pour la vérification formelle de propriétés logiques et temporelles, Proverif pour la vérification de propriétés de sécurité [ABA 02] et SocLib pour le prototypage virtuel du logiciel et du matériel de systèmes temps-réel. L'absence de communication cryptée dans le système du pacemaker et la délégation des implantations logicielles aux approches MARTE et AADL font que nous n'utiliserons ni Proverif ni SocLib dans la suite de ce chapitre. A contrario, UPPAAL est utilisé dans ce chapitre pour la vérification de contraintes de sureté de fonctionnement.

6.3. Expression AVATAR du modèle SysML de pacemaker enrichi

La modélisation en AVATAR est présentée en deux étapes : le fonctionnement et les hypothèses du système modélisé, et la réprésentation des exigences dudit système.

6.3.1. *Fonctionnement du Pacemaker et hypothèses de modélisation*

Palliant aux insuffisances d'un coeur qui ne bat plus correctement, un pacemaker en fonctionnement peut se trouver dans un des modes suivants caractériséq par trois lettres.

- La première lettre (A, V ou D) indique qui de l'oreillette, du ventricule ou des deux est stimulé électriquement (*paced* en anglais) et doit recevoir une impulsion.
- La deuxième lettre (A, V ou D) indique quelle chambre (éventuellement les deux) sera monitorée électriquement et donc observée.
- La troisième lettre I, T ou D identifie respectivement les modes '*Inhibited*, *Self Triggering* et "*Dual Pacing*". Dans le mode I, une contraction cardiaque

détectée comme un événement provenant du coeur dans le délai imparti va empêcher le pacemaker d'envoyer une impulsion électrique. Dans le mode T, un événement provenant du coeur enclenchera immédiatement une stimulation de la part du pacemaker. Ainsi, dans le mode VVI, le ventricule est stimulé si un événement ne provient pas du ventricule. Si un événement provient du ventricule alors la stimulation est inhibée. Ce mode particulier sera utilisé dans ce chapitre pour illustrer la vérification formelle guidée par des observateurs.

Ce chapitre ajoute d'autres hypothèses simplificatrices à l'ensemble du modèle. Nous supposerons en particulier que le pacemaker a été correctement implanté et que le patient est en mode déambulatoire. Le pacemaker est réputé correctement initialisé. Ses composants logiciels et matériels ne tombent jamais en panne. Ni sa maintenance, ni son remplacement, ni sa valorisation en tant que déchet ne sont traités. De cette architecture globale, nous extrayons une architecture simplifiée, support à une présentation pédagogique des modes de simulation et de vérification formelle supportés par TTool.

6.3.2. *Diagramme d'exigences*

Le diagramme d'exigences AVATAR de la figure 6.1 reprend les diagramme d'exigences SysML du chapitre précédent. Il met en évidence à la fois les buts principaux de l'application, les hypothèses de modélisation, les relations de raffinement entre exigences, ainsi que les relations de composition. Nous avons enrichi ce diagramme avec des liens entre exigences et "propriétés à vérifier". Pour l'instant, ces propriétés sont identifiées par un nom. Elles seront détaillées lorsque les éléments d'architectures tels que les noms de signaux et les attributs de blocs auront été définis. Notons que pour les besoins de l'exposition dans ce chapitre, nous nous limitons à des propriétés liées au mode VVI. Plus précisément, la verticale gauche de la figure 6.1 raffine l'exigence fondamentale "Eviter les défaillances cardiaques" en plusieurs étapes qui conduisent à introduire le mode VVI. La verticale de droite complète le diagramme par des hypothèses sur le patient et, plus bas, sur le fonctionnement du système lui-même. Au bas de la figure, les relations «verify» lient le noeud d'exigence VVIstimulation aux propriétés que l'on vérifiera à la section 6.6.

6.4. Architecture

La clé de voûte d'une conception AVATAR est le diagramme d'instances de blocs, reflet de l'architecture du système modélisé. La simulation et la vérification formelle d'un modèle AVATAR nécessitent d'avoir un modèle "fermé". C'est pourquoi le diagramme d'instances de blocs de la figure 6.2 décrit non seulement l'architecture interne du pacemaker mais aussi les connexions de celui-ci avec le coeur qu'il observe et stimule. Notons bien que la figure 6.2 décrit une architecture organique qui, dans une démarche d'ingénierie système, vient après l'architecture fonctionnelle non décrite dans ce chapitre.

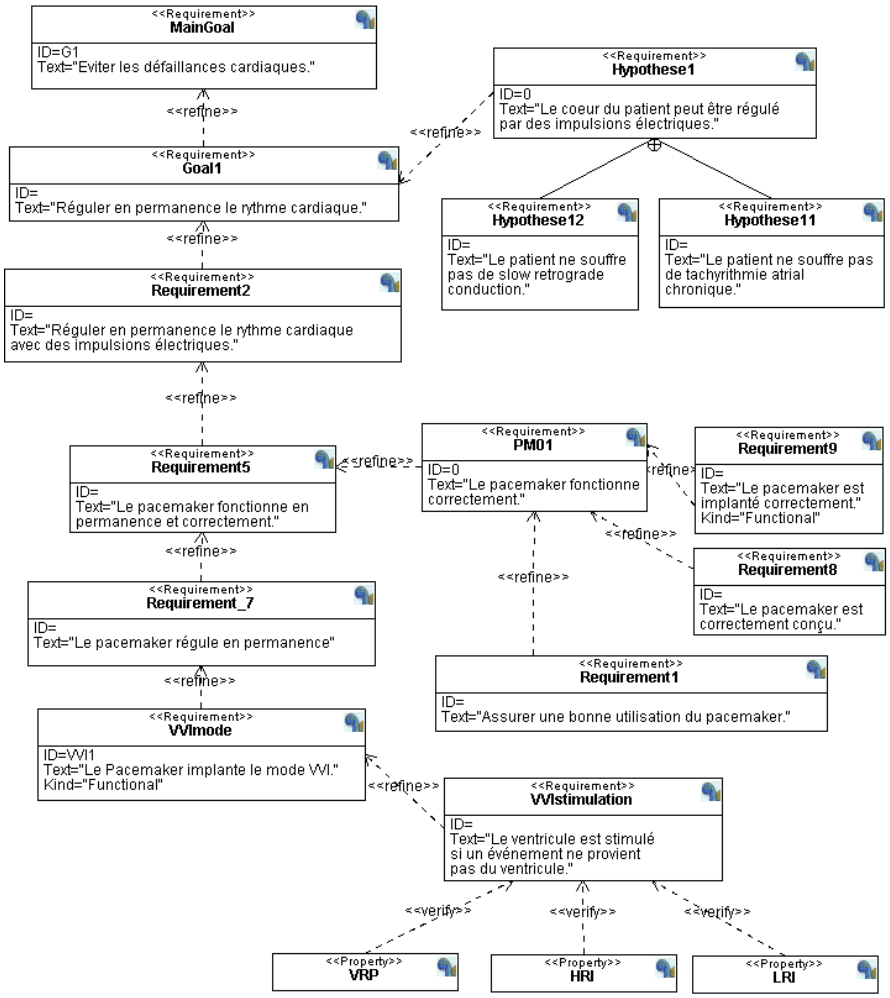

Figure 6.1 – Diagramme d'exigences : Buts, hypothèses, exigences raffinées et propriétés

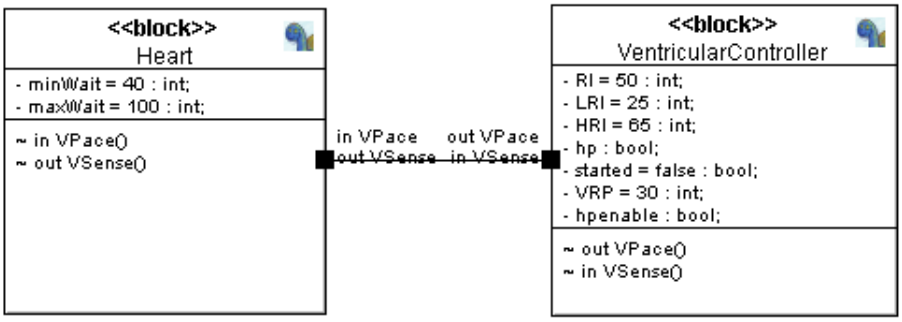

Figure 6.2 – Extrait du diagramme d'instances de blocs

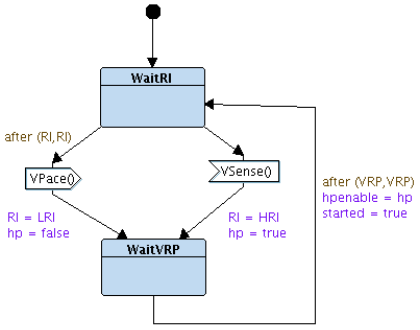

Figure 6.3 – Machine à états du controlleur de venticule

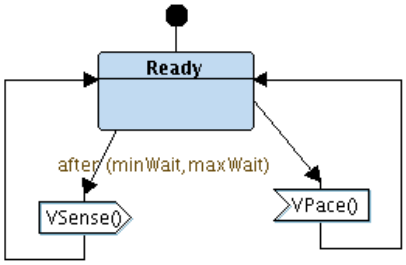

Figure 6.4 – Machine à états du controlleur du coeur

6.5. Comportement

Toutes les instances de blocs représentées sur la figure 6.2 ont un comportement décrit par une machine à états. Nous avons choisi de présenter celles du controlleur de ventricule (*VentricularController*) et du coeur dans les figures 6.3 et 6.4. Le premier diagramme (figure 6.3) met en évidence deux états principaux (*WaitRI* et *WaitVRP*) correspondant respectivement à l'exécution d'un impact sur le coeur si ce dernier n'a pas effectué son action normale, et à l'attente du prochain cycle.

Les instances de blocs étant dotées de comportements, il devient possible de les simuler avec l'outil TTool. Celui-ci permet de jouer, pas à pas ou en rafale, des scénarios d'exécution du modèle. Les portions de modèle explorées sont rapidement identifiables sur les diagrammes eux-mêmes et les traces de simulation prennent la forme de diagrammes de séquences, tels que celui présenté à la figure 6.5. Ces traces mettent

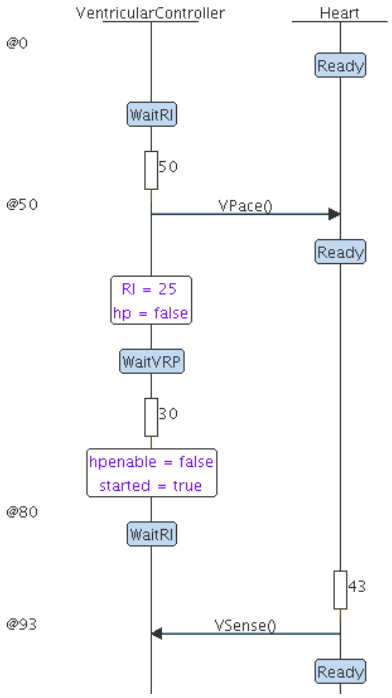

Figure 6.5 – Diagramme de séquences obtenu à partir d'une simulation du système

en évidence les communications synchrones et asynchrones entres les instances, les actions logiques effectuées par les instances (i.e., les changements de valeur des variables) et enfin les délais entre actions ou communications.

6.6. Vérification formelle du mode VVI

La pratique de la vérification formelle conduit à distinguer des propriétés qualifiées de "générales" applicables à une large classe de systèmes (section 6.6.1) et les propriétés spécifiques au système en cours d'étude (section paragraphe 6.6.2).

6.6.1. *Propriétés générales*

Comme la plupart des systèmes temps réel, le pacemaker doit satisfaire des propriétés générales telles que l'absence de blocage (*deadlock*), l'absence de cycle interne sans échappatoire (*livelock*) et la capacité à retourner à l'état initial.

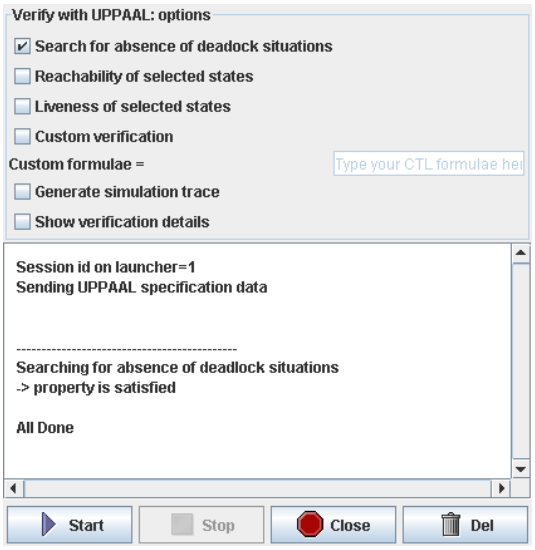

Figure 6.6 – Fenêtre de vérification : recherche de deadlock

La vérification de ces propriétés à l'aide de l'outil UPPAAL nécessite la traduction du modèle AVATAR en automates temporisés. Cette dernière est entièrement prise en charge par TTool qui propose, en natif, de vérifier l'absence de blocage. La copie d'écran de la figure 6.6 montre ainsi que le modèle AVATAR du pacemaker ne comporte pas de deadlock. Cette bonne propriété a été vérifiée sans avoir à écrire ou lire une ligne du "code automate temporisé" automatiquement généré par TTool et injecté dans l'outil UPPAAL.

6.6.2. *Expression de propriétés en TEPE*

Nous avons modélisé en TEPE les trois propriétés définies lors de la phase de modélisaton des exigences (cf. section 6.3.2) :

- **PropLRI** : Lorsque le *pace* sur l'hysteresis est désactivé, un *pace* sur le ventricule ou alors une action *sense* du coeur doit être effectuée avant LRI unités de temps
- **PropHRI** : Lorsque le *pace* sur l'hysteresis est activé, un *pace* sur le ventricule ou alors une action *sense* du coeur doit être effectuée avant HRI unités de temps.
- **PropVRP** : Une action sense ne peut être effectuée avant que VRP unités de temps ne se soient écoulées après l'action précedente *sense* ou une action *pace*.

A titre d'exemple, la figure 6.7 représente *PropLRI*. L'opérateur TC représente une contrainte de temps (*Time Constraint*) d'une valeur LRI. Ainsi, une fois le l'hystérésis

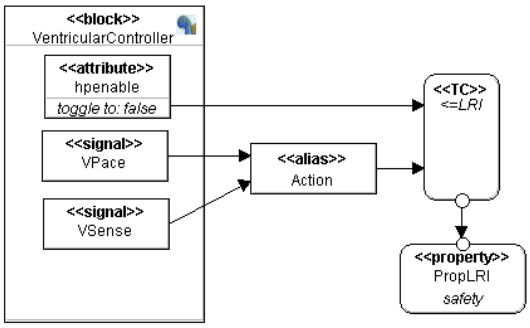

Figure 6.7 – Modélisation en TEPE d'une propriété de surêté : PropLRI

désactivé, une "Action" (c'est à dire, soit un signal *VPace* soit un signal *VSense*, cf. l'opérateur $<< Alias >>$) doit être effectuée dans le système avant LRI unités de temps. D'une façon assez similaire, il est possible de modéliser graphiquement et simplement la propriété *PropHRI*.
La propriété PropVRP est plus complexe à modéliser car elle fait référence à des états particuliers de la machine à états du controlleur. Ainsi, sa modélisation repose sur l'utilisation des signaux d'entrée dans les états *WaitVRP* et *WaitRI* (cf. figure 6.8). La propriété modélise deux sous propriétés mises en relation avec l'opérateur *OR*. Une sous-propriété exprime la cas où le système n'a pas encore effectué un premier cycle i.e., *started* vaut *false*. La deuxième sous-propriété s'applique uniquement dans le cas où un premier cycle a été effectuée : ainsi, le système a déjà atteint l'état *WaitVRP*, ce qui se traduit par le signal *enterState__WaitVRP*). Dans ce cas, le système doit attendre une durée *VRP* avant d'effectuer le cycle suivant dont le démarrage dse traduit par l'entrée dans l'état *WaitRI* (signal TEPE *enterState__WaitRI*).

La vérification des propriétés précédemment décrites repose soit sur leur expression sous la forme d'une formule en logique temporelle (section 6.6.3), soit par utilisation d'observateurs (section 6.6.4).

6.6.3. *Utilisation de logiques temporelles*

Une approche très usitée de vérification consiste à exprimer le système dans un langage formel (par exemple, LOTOS), à exprimer les propriétés à vérifier dans une logique temporelle (par exemple, CTL) puis à donner en entrée d'un *model-checker* à la fois la specification du système et les propriétés. Le système est transformé en un système de transitions étiquetées sur lequel les propriétés sont étudiées par le *model-checker*. Parmi les logiques temporelles usuelles, nous pouvons citer LTL, CTL, PSL/-Sugar.

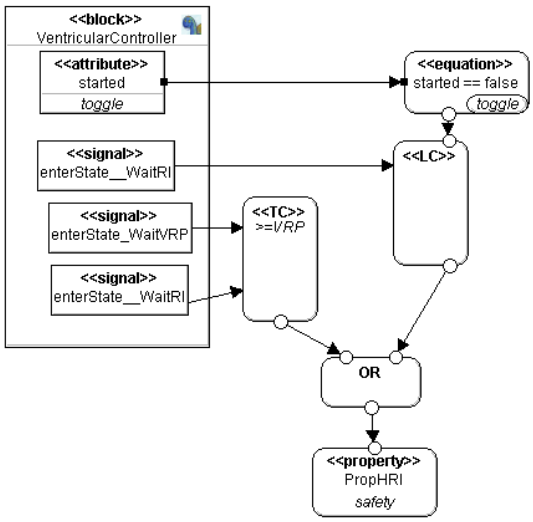

Figure 6.8 – Modélisation en TEPE d'une propriété de surêté : PropVRP

Avec TTool, il est possible de saisir une formule CTL qui est analysée par le model-checker d'UPPAAL. Cette formule fait directement référence à l'automate d'UPPAAL, et nécessite donc d'ouvrir les automates générés par TTool avec UPPAAL, et de comprendre ces derniers. Afin de montrer la différence entre une propriété saisie grpahiquement avec TEPE et une formule CTL corresponante, la formule CTL, au format UPPAAL, de la propriété PropLRI est :

```
A[] VentricularController__0.hp  imply
VentricularController__0.h <= VentricularController__0.HRI
```

et celle de la propriété PropVRP est :

```
A[] ((VentricularController__0.id0 && VentricularController__0.started) imply
VentricularController__0.h__ >= VentricularController__0.VRP
```

Aussi, la figure 6.9 montre la fenêtre de saisie d'une formule CTL, et sa vérification depuis TTool.

6.6.4. *Vérification guidée par des observateurs*

La vérification guidée par les observateurs consiste à modifier le modèle de conception avec de nouveaux blocs qui sont chargés d'observer le modèle et d'effectuer une

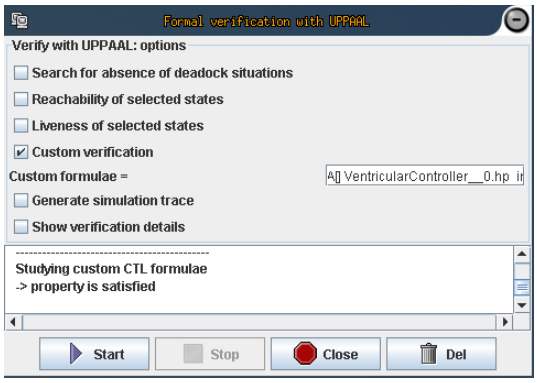

Figure 6.9 – Vérification d'une formule CTL depuis TTool : verification de PropLRI

action particulière (par exemple, aller dans un état nommé "erreur") si la ou les propriétés observées ne sont pas satisfaites. Les blocs d'observation recoivent ainsi des signaux émis par les blocs de la conception, et prennent les décisions de satisfiabilité en fonction de ces signaux. Quelques remarques :

— Les blocs d'observation sont par hypothèse non intrusif sur le système observé, c'est à dire que leur adjonction au modèle ne doit en aucun cas modifier le résultat des propriétés observées
— L'adjonction de blocs d'observation peut parfois amener à modifier la conception afin d'ajouter des signaux émis par les blocs observés. Ces adjonctions de signaux se fait toujours en respectant l'aspect non-intrusif de l'aobservateur, c'est à dire que l'observateur doit être prêt à accepter en permanence les signaux synchrones ajoutés au système pour l'observation.

A titre d'exemple, la figure 6.10 présente le diagramme d'instances de blocs auquel a été ajouté des observateurs pour les trois propriétés étudiées dans ce chapitre. Pour chaque nouveau bloc, il faut bien entendu fournir une machine à états.

La machine à états de l'observateur qui étudie les propriétés PropLRI et PropHRI st représenté à la figure 6.11. Les signaux émis par le controlleur du ventricule sont analysés afin de déterminer si un intervalle de temps clairement spécifié s'écoule entre des réceptions, ou non. Cet intervalle de temps est modélisée par le démarrage, l'expiration et l'arrêt d'un timer qui sert de chronomètre.

TTool permet de rechercher automatiquement l'atteignabilité (*reachability*) des états ou actions des marchines à l'état. La méthodologie de vérification à l'aide d'observateurs repose sur la preuve de non-atteignabilité des états d'erreur des observateurs. Cette preuve est menée automatiquement par TTool, en s'appuyant encore une fois sur la transformation automatique du modèle et des observateurs en automates temporisés, puis l'appel au *model-checker* d'UPPAAL avec des formules CTL obtenues

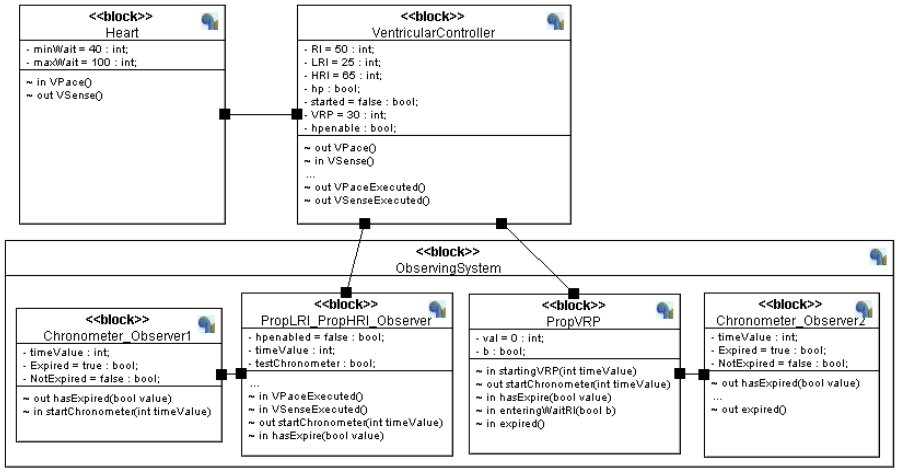

Figure 6.10 – Conception avec observateurs

par analyse da la modélisation SysML et de sa transformation en automates temporisés. La figure 6.11 montre le résultat de cette méthodologie de vérification appliquée au pacemaker, en particulier le fait que l'état d'erreur de l'oservateur des propriétés PropLRI et PropHRI n'est pas atteignable.

Les observateurs ont été ajoutés manuellement à la conception. En effet, actuellement, les propriétés TEPE ne peuvent pas être automatiquement transformées en observateurs. Nous travaillons actuellement à l'ajout à TTool de cette génération automatique, comme cela est le cas pour d'autres profils : TURTLE [FON 08] et DIPLODOCUS [KNO 11b].

6.6.5. *Remontée au modèle*

Lorsque l'on effectue des simulations ou des vérifications d'un modèle, il se pose la question de l'interprétation des résultats issus des dites simulation et vérifications par rapport au modèle fournit en entrée au simulateur ou au vérificateur.
Les simulations avec TTool de modèlisations AVATAR sont effectuées directement sur les diagrammes. En particulier, TTool anime directement les machines à états du système simulé, et produit en outre un diagramme de séquences dont les noms des entités et des actions sont exactement ceux du modèle.
Dans le cas des vérification, les preuves sont effectuées sur un modèle sous forme d'automates temporisés qui est structurellement différent du modèle AVATAR correspondant. Dans le cas d'une propriété non vérifiée, le *model-checker* d'UPPAAL exhibe une trace menant à la violation de la propriété. Actuellement, cette trace de

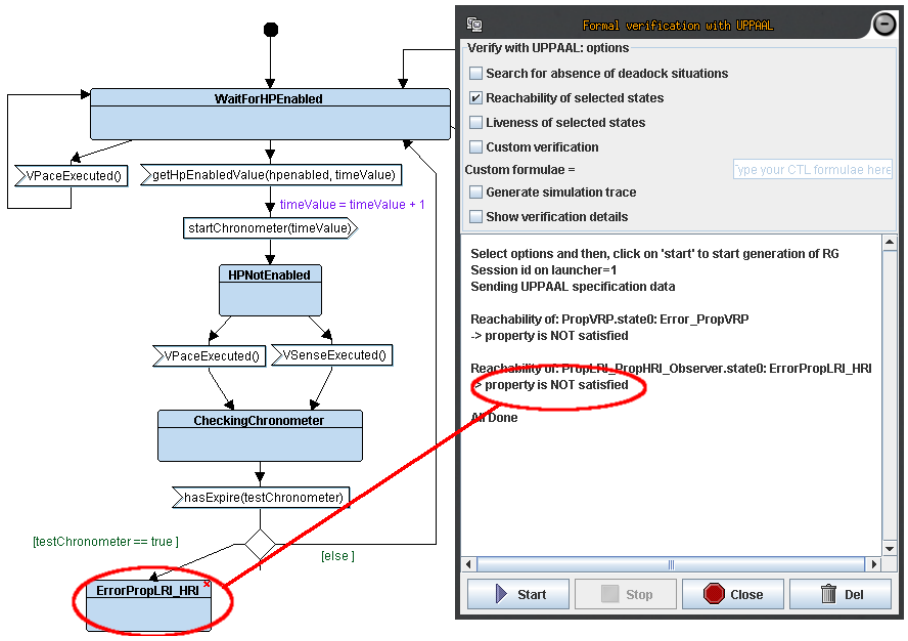

Figure 6.11 – Machine à états d'un observateur, et vérification de la non-accessibilité de l'état d'erreur de cet observateur

simulation doit être analysée sous UPPAAL directement, TTool n'étant pas capable de la transformer en une trace AVATAR. La résolution de ce problème fait partie de nos travaux futurs.

6.7. Positionnement par rapport à d'autres travaux

Notre contribution consiste en un environnement de modélisation (AVATAR) et en un outil support à ce langage (TTool). Nous discutons du positionnement de ces deux contributions dans cette section.

6.7.1. *Langages*

Dans la mouvance des travaux qui jettent des ponts entre le langage SysML et les méthodes formelles supportées par des outils de vérification de modèles, le langage AVATAR précise la sémantique des diagrammes de conception SysML et intègre l'expression de propriétés directement avec un diagramme SysML là où d'autres environnements délègue cette fonction de modélisation à un formalisme externe, logique

temporelle par exemple. Le langage d'expression de propriétés TEPE se veut plus accessible aux personnes rebutées par les langages formels que ne peut l'être le langage CCSL associé au profil UML MARTE.

Au delà des langages de la famille UML, bien d'autres environnements permettent la modélisation d'un système temps-réel et sa vérification formelle. UPPAAL [BEN 04] en est un bon exemple : il permet de modéliser un système sous la forme d'un ensemble d'automates communicants temporisés. La vérification formelle de propriétés CTL est directement intégrée à l'environnement du même nom (UPPAAL). Par contre, le langage sous forme d'automates peine àsupporté un processus d'ingénierie système, et n'est pas usité non plus pour documenter un projet. Ce constat s'étend à d'autres formalismes tels que les réseaux de Petri supportés par l'outil TINA [BER 06], ou encore l'algèbre de processus LOTOS supportée par l'outil CADP [GAR 07].

6.7.2. *Outils*

UML/SysML vers ESL [VIE 06] est un environnement de modélisation pour les systèmes sur puces, offrant de la simulation et de la vérification formelle depuis les modèles constitués de diagrammes de séquences UML. Les résultats de simulation et de vérification permettent notamment d'obtenir les pires cas d'exécution. Mais là encore, toute la phase d'ingénierie n'est pas couverte. De plus, l'environnement est très limité en termes de communications.

L'environnement basé SysML présenté dans [SIL 09] permet la vérification - avec UPPAAL - de propriétés logiques et temporelles sur des modèles réalisés avec l'outil Rhapsody. Cette approche ne supporte pas la phase de recueil d'exigences. En l'absence de langage de haut niveau pour exprimer les propriétés, il faut utiliser des formules CTL. TTool permet au contraire de vérifier l'accessibilité et la vivacité d'actions en ajoutant par de simples clics de souris des marqueurs aux actions concernées.

L'environnement OMEGA [OBE 09] supporte la phase de capture d'exigences en raison de son support de SysML via Rhapsody, mais ne permet pas d'intégrer ces exigences (et des propriétés) dans l'approche de vérification formelle. OMEGA et TTool ne supportent par les flux continus à la SysML, contrairement à l'outil Artisan [5], qui supporte de plus les probabilités sur les transitions, ainsi que les régions interruptibles.

Développée dans l'environnement Eclipse récemment enrichi avec l'éditeur Papyrus, l'atelier TopCased [FAR 06] fédère des outils d'analyse de modèle et des générateurs de code construits qui exploitent l'approche "transformation de modèle". L'outil TTool utilisé dans ce chapitre repose sur une approche différente reposant sur un environnement moins flexible mais formalisée, tant du point de vue conception qu'expression de propriétés.

La recherche de nouvelles techniques de conception en électronique a aussi motivé des travaux liant SysML et les méthodes formelles. Ainsi, des modèles SysML peuvent être traduits en VHDL-AMS [Rea 11] ou en simulink [VAN 06]. En génie mécanique, SysML est aussi utilisé conjointement avec d'autres langages spécifiques, tel que Modelica [PAR 10]. Ce langage décrit des composantes mécaniques, électriques, hydraulique et thermique d'un système sous la forme d'un ensemble d'équations qu'un simulateur peut résoudre à chaque pas temporel.

6.8. Conclusion

Le développement du langage AVATAR et du logiciel libre TTool s'inscrit dans la mouvance des travaux qui allient les langages graphiques semi-formels de type SysML aux langages formels déjà dotés d'outils de vérification. L'objectif poursuivi de détecter les erreurs de conception au plus tôt dans le cycle de vie du système.

S'appuyant sur la sémantique formelle du langage AVATAR, l'outil TTool traduit un modèle AVATAR en automates temporisés, intègre un simulateur pour la mise au point de ce modèle et pilote l'outil UPPAAL pour rendre l'usage d'un langage et d'un outil formels aussi transparents que possible à la personne qui confronte un modèle AVATAR à un ensemble d'exigences. Les résultats de simulation et de vérification reposent sur le modèle fournit par l'utilisateur dans TTool, sur l'absence d'erreurs dans le processus de traduction vers des automates temporisés, et sur l'absence d'erreurs dans le *model-checker* d'UPPAAL. L'utilisation de traducteur de d'outils de vérification certifiés ou prouvés permettrait d'asseoir sur des bases plus solides les résultats de vérification.

Par le diagramme d'exigences et l'architecture globale, ce chapitre a montré un continuum entre le modèle AVATAR et le modèle SysML du précédent chapitre. Le discours focalisé sur le mode VVI a permis d'illustrer les fonctionnalités de simulation et vérification formelles qu'offre l'outil TTool.

Le diagramme AVATAR a été analysé à un haut degré d'abstraction. Ses blocs logiciels ont vocation à servir de point de départ à une conception UML temps réel ou AADL.

6.9. Bibliographie

[ABA 02] ABADI M., BLANCHET B., « Analyzing Security Protocols with Secrecy Types and Logic Programs », *29th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL 2002)*, Portland, Oregon, ACM Press, p. 33–44, janvier 2002.

[APV 11a] APVRILLE L., SAQUI-SANNES P. D., « AVATAR/TTool : un environnement en mode libre pour SysML temps réel », *Génie Logiciel*, vol. 98, p. 22-26, septembre 2011.

[APV 11b] APVRILLE L., « webpage of TTool », *http ://ttool.telecom-paristech.fr/*, 2011.

[BEN 04] BENGTSSON J., YI. W., « Timed Automata : Semantics, Algorithms and Tools », *Lecture Notes on Concurrency and Petri Nets*, W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, p. 87-124, 2004.

[BER 06] BERTHOMIEU B., VERNADAT F., « Time Petri Nets Analysis with TINA », *3rd Int. IEEE Conf. on The Quantitative Evaluation of Systems (QEST 2006)*, p. 123-124, 2006.

[FAR 06] FARAIL P., GAUFILLET P., CANAL A., CAMUS C. L., SCIAMMA D., MICHEL P., CRÉGUT X., PANTEL M., « The topcased project : a toolkit in open source for critical aeronautic systems design », *In ERTS2006 : Embedded Real Time Software*, Toulouse, France, novembre 2006.

[FON 08] FONTAN B., Méthodologie de conception de systèmes temps réel et distribués en contexte UML/SysML, Thèse de doctorat, Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS), 2008.

[GAR 07] GARAVEL H., LANG F., MATEESCU R., SERWE W., « CADP 2006 : A Toolbox for the Construction and Analysis of Distributed Processes », *Computer Aided Verification (CAV'2007)*, vol. 4590, Berlin Germany, p. 158-163, 2007.

[Kno 11a] KNORRECK D., APVRILLE L., DE SAQUI-SANNES P., « TEPE : A SysML Language for Time-Constrained Property Modeling and Formal Verification », *ACM SIGSOFT Software Engineering Notes*, vol. 36/1, p. 1–8, janvier 2011.

[KNO 11b] KNORRECK D., UML-based Design Space Exploration, Fast Simulation and Static Analysis, PhD thesis, EDITE, Ecole Doctorale Informatique, Télécommunications et Electronique, Telecom ParisTech, 2011.

[KOY 92] KOYMANS R., *Specifying Message Passing and Time-Critical Systems with Temporal Logic*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992.

[LEB 10] LEBLANC P., « Modélisation SysML exécutable d'un système événementiel et continu », *Livre blanc IBM, https ://files.me.com/jmbruel/gblurt*, 2010.

[LET 05] LETIER E., KRAMER J., MAGEE J., UCHITEL S., « Fluent temporal logic for discrete-time event-based models », *Proceedings of the 10th European software engineering conference*, ESEC/FSE-13, New York, NY, USA, ACM, p. 70–79, 2005.

[OBE 09] OBER I., DRAGOMIR I., « OMEGA2 : A new version of the profile and the tools », *14th IEEE International Conference on Engineering of Complex Computer Systems, UML-AADL'2009*, Potsdam, p. 373-378, 2009.

[OMG 11] OMG O. M. G., « SysML », *http ://www.sysml.org/*, 2011.

[PAR 10] PAREDIS C. J., BERNARD Y., BURKHART R. M., DE KONING H.-P., FRIEDENTHAL S., FRITZSON P., ROUQUETTE N. F., SCHAMAI W., « An Overview of the SysML-Modelica Transformation Specification », *INCOSE'2010*, 2010.

[Rea 11] REALTIME-AT-WORK, « SysML-Companion : Virtual prototyping from SysML models », *http ://www.realtimeatwork.com/software/sysml-companion/*, 2011.

[SAT 10] SATURN P. F., « SATURN, SysML bAsed modeling, architecTUre exploRation, simulation and syNthesis for complex embedded systems », *http ://www.saturn-fp7.eu/*, 2010.

[SIL 09] DA SILVA E. C., VILLANI E., « Integrating SysML and model-checking techniques for the v&v of space-based embedded critical software (In Portugese) », *Brasilian Symposium on Aeropspace Engineering and Applications*, 2009.

[VAN 06] VANDERPERREN Y., DEHAENE W., « From UML/SysML to Matlab/Simulink : current state and future perspectives », *DATE '06 : Proceedings of the conference on Design, automation and test in Europe*, 3001 Leuven, Belgium, Belgium, European Design and Automation Association, p. 93–93, 2006.

[VIE 06] VIELHL A., SCHONWALD T., BRINGMANN O., ROSENSTIEL W., « Formal performance analysis and simulation of UML/SysML models for ESL design », *DATE '06 : Proceedings of the conference on Design, automation and test in Europe*, Munich Germany, p. 1-6, 2006.