Synthèse d'observateurs à partir d'exigences temporelles

Benjamin Fontan[*], Pierre de Saqui-Sannes[*], Ludovic Apvrille[**]

[*] Université de Toulouse ; ISAE ; LAAS-CNRS
10 avenue Edouard Belin, BP 5402, Toulouse Cedex 04, France
bfontan@isae.fr, pdss@isae.fr

[**] GET/ENST, Laboratoire System-on-Chip
2229 route des Crêtes, B.P. 193, 06904 Sophia-Antipolis Cedex, France
ludovic.apvrille@telecom-paris.fr

Résumé. A contrario des normes UML 2.1 et SysML, le profil UML TURTLE (Timed UML and RT-LOTOS Environment) dispose d'une sémantique formelle et d'une méthodologie. Avec les systèmes temps réel pour cible, cette méthodologie met l'accent sur la vérification formelle du comportement des objets. Le profil TURTLE est doté d'un langage graphique et formalisé d'expression d'exigences temporelles. La contribution de cet article réside dans la présentation d'algorithmes de génération d'observateurs à partir d'exigences temporelles exprimées dans ce langage. Ces observateurs sont destinés à guider la vérification formelle et en particulier à confronter le comportement des objets aux exigences temporelles tout en traçant ces dernières au long de la trajectoire de conception du système en cours d'étude. Un dispositif de charge d'une batterie de véhicule hybride sert d'étude de cas.

1 Introduction

La notation UML (*Unified Modeling Language*) (OMG, 2007) normalisée par l'Object Management Group est un langage de modélisation à large spectre dédié aux systèmes à logiciel prépondérant. Le concept de « profil UML » permet de personnaliser la notation de l'OMG pour les besoins d'un domaine d'application particulier. Un profil se doit de doter UML d'une sémantique et d'une méthodologie supportée par des outils.

Les profils « UML temps réel » (Garbi et al, 2006) apportent ainsi des réponses aux besoins spécifiques des concepteurs de systèmes temps réel. Ils enrichissent UML par des constructions (ports de communication par exemple) aptes à traiter le caractère réactif des systèmes temps réel et prennent en compte l'assujettissement de ces systèmes à des exigences temporelles. Les techniques généralement associées à ces profils sont d'une part, les générateurs de code exécutable et, d'autre part, les techniques de vérification formelle. Ces derniers permettent de confronter une conception UML à un ensemble d'exigences et ce, en préalable à toute exécution sur une cible réelle.

La pratique de la vérification formelle fait ressentir le besoin d'intégrer les exigences au modèle de conception. Or, sur ce point, la notation UML pêche par l'absence de diagramme dédié aux exigences. Une solution possible est d'emprunter au langage SysML (SysML, 2006) la notion de diagramme d'exigences. C'est la solution retenue dans le cadre du profil UML temps réel TURTLE (*Timed UML and RT-LOTOS Environment*) et qui motive le travail présenté dans cet article.

Le profil TURTLE introduit en (Apvrille *et al.*, 2004), étendu en (Apvrille *et al.*, 2006) et supporté par l'atelier Open Source TTool (TTool, 2007) permet de couvrir les phases d'analyse, conception et déploiement d'un système temps réel ou distribué. Bien que cet article s'intéresse en priorité aux diagrammes de conception, nous pouvons préciser qu'un modèle TURTLE inclut généralement trois familles de diagrammes : (1) une analyse à base de cas d'utilisation et de scénarii ; (2) des diagrammes de conception qui définissent l'architecture et le comportement du système ; (3) une architecture de déploiement qui préfigure l'implantation réelle du système.

De cette brève introduction il ressort que le profil TURTLE n'offrait à l'origine aucune facilité pour décrire les exigences. (Fontan, 2008) propose d'étendre TURTLE par des diagrammes d'exigences calqués sur ceux de SysML et enrichis pour traiter les exigences temporelles dans un cadre qui allie l'aisance d'utilisation d'un chronogramme et le formalisme requis pour vérifier formellement la satisfaction des exigences. (Fontan 2008) traite en particulier des exigences temporelles et de leur vérification formelle au moyen d'observateurs. Objets ajoutés au modèle TURTLE du système, ces observateurs sont en charge de piloter la vérification formelle.

Jusqu'à présent les observateurs étaient conçus manuellement avec le risque d'erreurs qui va de pair. La contribution principale de cet article réside dans l'automatisation de la construction des observateurs à partir d'exigences temporelles exprimées dans un langage de type « chronogrammes » inséré dans des diagrammes d'exigences à la SysML.

Le reste de l'article est structuré de la manière suivante. La section 2 rappelle comment le profil TURTLE se distingue de la norme UML 2.1 au niveau des diagrammes d'analyse et de conception. La section 3 montre comment les diagrammes d'exigences SysML 1.0 ont été intégrés à TURTLE. La section 4 propose un langage d'expression d'exigences temporelles intégré aux diagrammes d'exigences TURTLE. La section 5 expose les principes de synthèse d'observateurs à partir d'exigences temporelles. La section 6 présente les travaux du domaine. Enfin, la section 7 conclut l'article. L'étude de cas retenue pour illustrer l'approche proposée, est un dispositif de charge de batterie de véhicule hybride.

2 Diagrammes de conception

2.1 Aperçu

Le profil UML TURTLE (Apvrille et al., 2004) (Apvrille et al., 2006) est dédié à la conception de systèmes répartis dont la complexité tient en premier lieu à la satisfaction d'exigences temporelles. Fort d'une sémantique formelle obtenue par traduction vers l'algèbre de processus temporisée RT-LOTOS (Courtiat, 2000) et du support de l'outil TTool (TTool), ce profil a été appliqué maintes fois et en particulier à la vérification de protocoles de communication de groupes sécurisés (Fontan et al., 2007). En effet, bien que l'outil TTool intègre un générateur de code Java, la destination première du profil TURTLE demeure la vérification formelle d'exigences temporelles.

En l'absence de diagramme d'exigences, une modélisation TURTLE démarre par une phase d'analyse. Un diagramme de cas d'utilisation est construit et documenté par des diagrammes de séquences (DS) structurés au moyen d'un diagramme global d'interactions (DGI). Ces diagrammes de séquences gèrent des temps relatifs et absolus. De plus, un scénario exprimé dans un DS peut interrompre un autre scénario (DS).

La phase d'analyse est suivie d'une phase de conception qui vise à décrire l'architecture du système et les comportements des différents objets qui composent cette architecture. Les diagrammes de classes/objets et d'activités sont les deux diagrammes de conception supportés par TURTLE. Présentés de manière plus détaillée dans les sections 2.2 et 2.3, ces diagrammes de classes/objets et d'activités forment la conception que l'on confronte aux exigences temporelles en utilisant TTool et des interfaces vers les outils de vérification formelle RTL (Courtiat et al., 2000) et CADP (Garavel, 2007).

L'outil TTool transforme tout modèle TURTLE dans un langage pivot appelé TIF (TURTLE Intermediate Format). Le modèle TIF sert de point de départ à la génération d'une spécification RT-LOTOS qui peut être formellement vérifiée en utilisant l'outil RTL. Celui-ci implante une analyse d'accessibilité qui, à partir de l'état initial du système modélisé, consiste à construire un graphe caractéristique de tous les états stables que ce système est susceptible d'atteindre au cours de son exécution. Le graphe d'accessibilité n'est pas nécessairement constructible et l'on supposera dans cet article qu'il l'est, de surcroît dans un délai non rédhibitoire. Une fois le graphe d'accessibilité construit, reste encore à l'exploiter. Pour cela, nous préconisons de procéder à une minimisation de ce graphe dès lors que le choix de la relation d'équivalence fait qu'une telle minimisation préserve les propriétés que l'on escomptait vérifier sur le graphe d'accessibilité d'origine. En pratique, nous recommandons d'utiliser l'équivalence observationnelle de Miner. Celle-ci préserve des propriétés de blocages qu'il serait impossible de déceler en utilisant l'équivalence de traces.

La pratique de TURTLE nous a montré que les concepteurs définissent des architectures majoritairement formées d'un ensemble de « boites » qui communiquent par rendez-vous. C'est pourquoi nous avons privilégié un type de vérification centré sur les rendez-vous entre objets. Ainsi, chaque transition du graphe d'accessibilité associée à une série d'actions incluant un rendez-vous se voit étiquetée par un label qui permet d'identifier sans ambigüité ce rendez-vous. La minimisation d'un tel graphe étiqueté en utilisant une relation d'équivalence (équivalence observationnelle de Milner, par exemple), produit en sortie un automate quotient qui élimine un certain nombre de labels de rendez-vous et préserve les labels importants pour le travail de vérification en cours. La préservation (par la minimisation) des « bons » labels dépend des labels manuellement sélectionnés dans l'outil TTool et de la relation d'équivalence retenue. Compte tenu de la taille des graphes d'accessibilité, il est plus commode de rechercher des labels de rendez-vous dans l'automate quotient.

A la construction d'un graphe d'accessibilité minimisé en automate quotient, nous ajoutons des observateurs en charge de guider la vérification. Ces observateurs vont eux-aussi communiquer par rendez-vous avec les objets qui composent le modèle du système. L'ajout d'observateurs au modèle du système suppose d'intervenir dans le diagramme de classes/objets qui définit l'architecture de ce système, mais aussi dans les diagrammes d'activités qui caractérisent les comportements. Dans les deux cas, il s'agit d'ajouter les éléments de modèles permettant les rendez-vous observateur-objet. Ainsi, l'analyse d'accessibilité et la minimisation seront réalisées sur un modèle du système augmenté des observateurs. Reste donc à construire les observateurs de telle sorte que les propriétés qu'ils servent à vérifier soient exprimables en termes de synchronisations. Cela peut-être fait à la main mais aussi de manière automatique comme nous le verrons plus loin dans cet article.

2.2 Diagrammes de classes/objets

En UML, un diagramme de classes représente l'architecture statique du système modélisé. Un diagramme d'objets peut le compléter pour mettre en évidence les instances des classes. Un diagramme de classes/objets TURTLE regroupe les classes et leurs instances en confondant les deux lorsqu'une classe n'est instanciée qu'une fois.

Comme en UML, une classe TURTLE (Tclass) possède des attributs et des méthodes. Le point remarquable est que TURTLE ajoute à UML la possibilité de représenter explicitement la composition – au sens des algèbres de processus et non du losange plein d'UML – entre objets. Ainsi, la Fig.1 montre deux classes (A et B) qui peuvent communiquer par rendez-vous au travers de leurs ports de communication (*Req* et *Ind*). L'on pourrait également représenter le parallélisme, l'invocation, l'exécution en séquence ou la préemption entre ces deux objets.

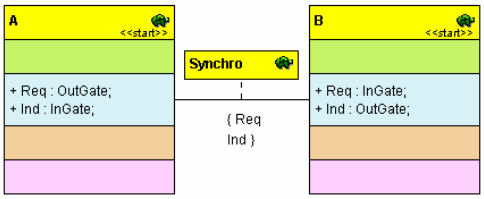

Fig.1. Exemple de diagramme de classes/objets TURTLE

2.3 Diagrammes d'activités

Une classe TURTLE possède nécessairement un comportement décrit par un diagramme d'activités. Outre les symboles autorisés par la norme UML pour ce diagramme en forme d'organigramme, un diagramme d'activités TURTLE permet d'exprimer des actions de synchronisation par rendez-vous. Trois opérateurs temporels sont également fournis pour exprimer un retard fixe, un intervalle temporel où encore une offre de rendez-vous limitée dans le temps (appelée « offre limitée dans le temps » dans la suite de l'article). La Fig.2 montre un extrait de diagramme d'activités de l'objet A de la Fig.1. L'on peut en particulier noter l'offre limitée dans le temps de T unités de temps sur l'action de synchronisation *Ind*. Si cette dernière est consommée avant T unités de temps, alors le chemin à gauche est exécuté (rebouclage de l'objet A). Dans le cas contraire, le comportement de A est terminé (symbole de fin du diagramme d'activités à droite de la Fig.2).

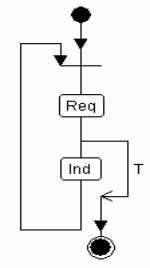

Fig.2. Exemple de diagramme d'activités TURTLE

3 Diagrammes d'exigences

Un diagramme d'exigences TURTLE (noté par la suite DE) étend le diagramme d'exigences SysML dans le but de formaliser les exigences temporelles et d'établir un lien entre ces dernières et la vérification pilotée par observateurs. Un DE permet d'exprimer : d'une part, la dérivation d'une exigence formelle temporelle à partir d'une exigence informelle ; d'autre part, le lien entre un observateur et l'exigence formelle temporelle à laquelle celui-ci se rapporte (cf. Fig.3)

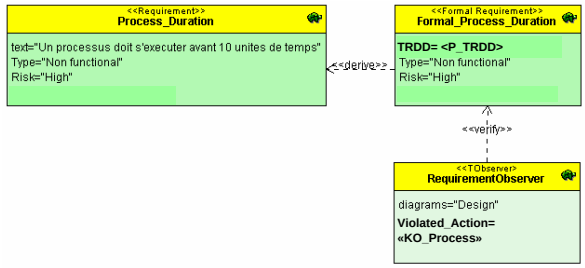

Fig.3. Diagramme d'exigences TURTLE

Comme le montre la Fig.3, une exigence informelle (stéréotypée par <<Requirement>>) possède quatre attributs : un *identifiant*, un *texte* (description informelle de l'exigence), un *type* (fonctionnelle, non-fonctionnelle) et un *niveau de criticité* (haut ou bas) provoquant respectivement, l'interruption ou non, de l'exécution du modèle TURTLE (voir section 5.2).

Une exigence formelle temporelle (stéréotypée par <<Formal Requirement>>) est également dotée d'un type, d'un niveau de criticité et d'une formalisation sous la forme d'un chronogramme exprimé dans le langage TRDD (*Timing Requirement Description Diagram*). Un TRDD étend les *Timing Diagrams* UML2 qui sont à l'origine des chronogrammes descriptifs de comportements d'objets et non un moyen d'exprimer des exigences. Le TRDD de la Fig.4 montre un processus observé entre les dates *Start_P* et *End_P*. La ligne de vie de l'exigence est composée d'un élément de début, d'éléments de description d'exigence et d'un élément de fin (Fig.4). Les éléments de description d'exigence représentent la satisfaction ou la violation de l'exigences, respectivement OK et KO sur la Fig.4. Notons que OK et KO sont séparés par une frontière temporelle de valeur 10.

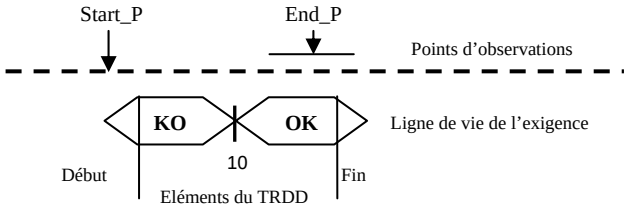

Fig.4. Diagramme de description d'exigences temporelles

Un DE peut enfin contenir un observateur (stéréotypé par <<TObserver>>). Celui-ci possède un nom, une indication sur le type de diagramme vérifié (analyse ou conception) et

un attribut *Violated_Action* qui désigne l'identificateur que l'on retrouvera dans le graphe d'accessibilité lorsque l'exigence sera violée. Notons que dans cet article, le processus de vérification d'exigences est appliqué aux diagrammes de conception.

4 Exemple : gestion de la batterie d'un véhicule hybride

Les véhicules hybrides posent un problème de charge efficace de la batterie dans la mesure où cette dernière est sollicitée par le moteur et ne peut pas se recharger en permanence. Il faut donc recharger cette batterie lorsque la voiture produit de l'énergie, c'est-à-dire dans les phases de décélération et de freinage (zones hachurés de la Fig.5). La Fig.5 montre l'évolution de la puissance électrique et mécanique au cours du temps (courbes bleue et rouge). Ti et Tf correspondent aux dates de chargement de la batterie. Le chronogramme représentant l'état de la batterie (chargée ou déchargée) apparaît également sur la Fig.5.

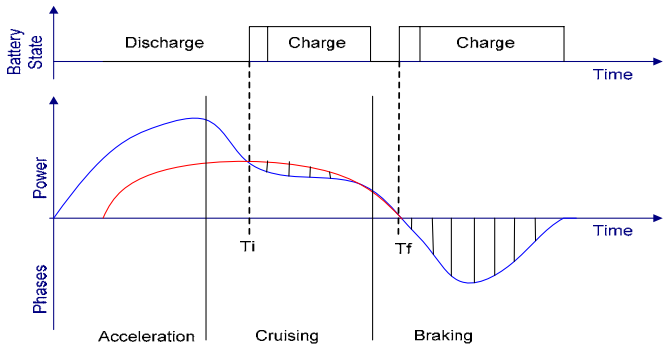

Fig.5. Gestion de l'énergie dans le véhicule hybride

Le chronogramme de la Fig.6 montre d'une part, les interactions du pilote par le biais des pédales de frein (notée B) et d'accélérateur (notée A) et, d'autre part, le moment où la batterie doit être chargée aux dates *Ti* et *Tf*, autorisant une latence *d* qui exprime la variabilité du temps réponse du système.

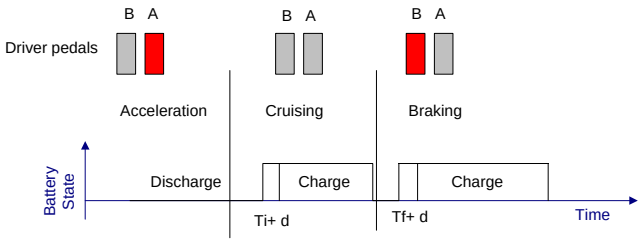

Fig.6. Exigence temporelle concernant la gestion de l'énergie

Le scénario décrit dans les Fig.5et Fig.6 correspond à un trajet en ville standard où le conducteur appuie sur la pédale d'accélérateur, la relâche, puis freine en appuyant sur la pédale de frein (par exemple en approchant d'un feu rouge).

Les Fig.5 et Fig.6 servent de référence pour la construction du diagramme d'exigences de la Fig.7. Sur cette dernière, les exigences temporelles sont documentées par des chronogrammes exprimées en TRDD.

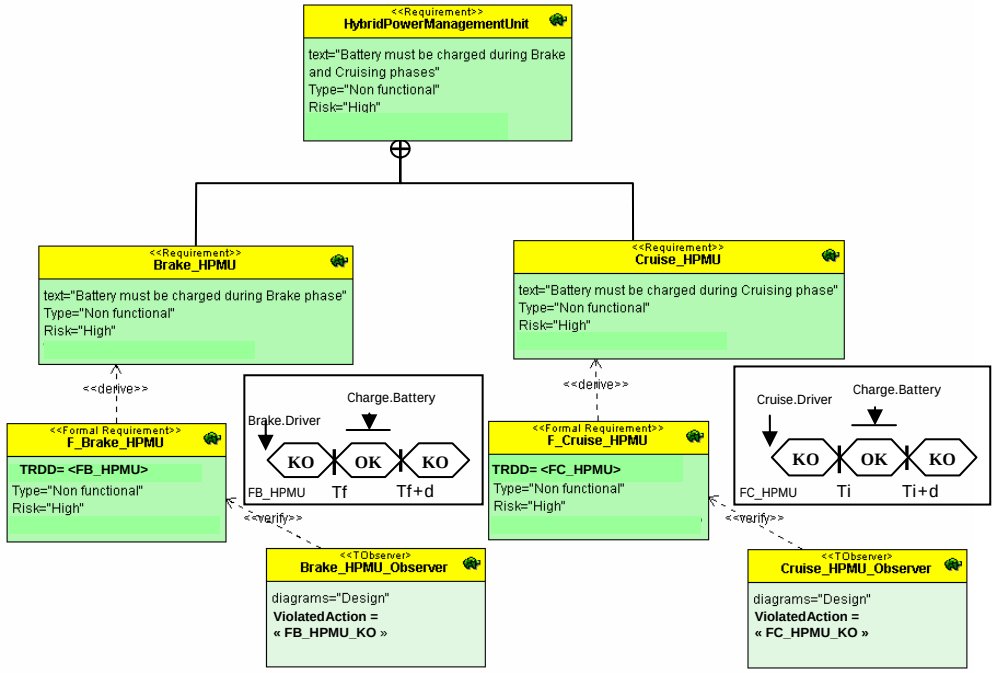

Fig.7. Diagramme d'exigence incluant les chronogrammes en TRDD

Avant d'expliquer en section 5 le principe de la synthèse des observateurs à partir des exigences temporelles, nous montrons sur la Fig.8 comment les informations contenues dans le diagramme de la Fig.7 se retrouvent dans la matrice de traçabilité que l'outil TTool construit à l'issue de l'analyse d'accessibilité. La satisfaction (respectivement non satisfaction) est exprimée par OK (respectivement KO).

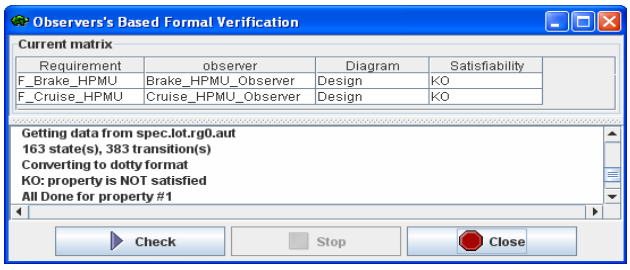

Fig.8. Matrice de traçabilité

5 Synthèse d'observateurs

5.1 Principe général

Les diagrammes TURTLE édités à l'aide de l'outil TTool sont traduits par celui-ci dans un format intermédiaire appelé TIF (*TURTLE Intermediate Format*). Le TIF sert de point de départ à la génération du code RT-LOTOS accepté par l'outil de vérification RTL.

Le modèle TIF est donc directement traduit des diagrammes de conception[1] (DC et DA). A partir des diagrammes d'exigences (DE et TRDD) des observateurs peuvent être générés pour guider et interpréter la vérification des exigences temporelles décrites par le diagramme de description d'exigences temporelles (TRDD). Les observateurs sont « greffés » au modèle du système à vérifier par construction dans la forme TIF du modèle TURTLE du système en question. Le format TIF ainsi obtenu est traduit en spécification RT-LOTOS pour générer le graphe d'accessibilité.

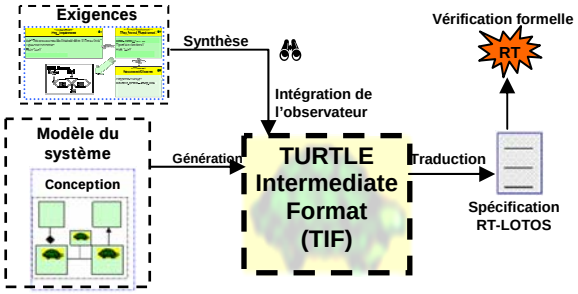

Fig.9. Méthodologie de vérification basée sur la synthèse d'observateurs

Dans la suite de cette section, nous présentons les algorithmes permettant de créer, pour chaque exigence temporelle, un observateur à ajouter au format intermédiaire TIF. Par la suite, nous considérons que ce format TIF est similaire à celui d'une conception TURTLE, c'est-à-dire que TIF est constitué des mêmes éléments structurants et comportementaux qu'une conception TURTLE, à savoir un ensemble de classes TURTLE (Tclasses) mises en relation (synchronisation, préemption, etc.). Chaque Tclass possède un diagramme d'activités.

L'idée générale de la traduction est la suivante : (1) pour chaque observateur, ajout d'une nouvelle Tclasse à la spécification TIF ; (2) construction automatique du diagramme d'activités de cette classe ; (3) intégration de cette classe au TIF en gérant ses « interférences » avec les autres classes.

[1] Ce modèle peut aussi être traduit à partir des diagrammes d'analyses (DGI et DS) (Apvrille et al., 2006)

5.2 Construction d'un observateur

5.2.1 Principe d'un observateur

Un observateur est une classe TURTLE stéréotypée par « TObserver » qui possède les mêmes caractéristiques de base qu'une classe TURTLE. Le nom de l'observateur est défini dans le diagramme d'exigences. Les attributs d'un observateur correspondent aux dates définies dans les frontières temporelles du TRDD. On distingue trois types de portes de communications :

- **Connexions au modèle des actions figurant dans le TRDD**. Conformément à la Fig.4, les portes de synchronisations sont appelées *points d'observations*. Les synchronisations entre observateurs et objets observés sont unidirectionnelles : elles vont des objets observés vers l'observateur pour prélever dans le système les différentes informations logiques correspondant aux exigences temporelles présentes dans le TRDD. Il existe deux types de points d'observations (pouvant être synchronisées avec différents objets observés) : l'action « Start » déclenchant la capture d'exigence et l'action « Capture » déclenchant le diagnostic de l'exigence par l'observateur.
- **Traçabilité des exigences**. Cette porte sert à exploiter la vérification de l'exigence décrite par le TRDD. On génère la porte de violation d'exigence par une étiquette définie dans le diagramme d'exigences (cf. l'attribut *Violated_Action* en section 3). Si l'exigence est satisfaite, l'observateur ne se manifeste pas pour ne pas perturber le comportement du système.
- **Contrôle de l'exécution du modèle.** L'arrêt du système est envisagé si une exigence de niveau de criticité haute est violée (cf. section 3). Par contre, le système n'est pas interrompu si l'exigence violée est de niveau de criticité bas. Dans le cas de violation d'une exigence haute, il faut construire des portes de communication entre l'observateur et les objets du système pour arrêter l'exécution du système. Enfin, dans le cas d'une exigence de niveau de criticité bas, il n'est pas nécessaire de construire des portes de contrôle d'exécution du modèle.

5.2.2 Principe général

La construction d'un observateur nécessite principalement d'extraire les informations du TRDD, et de les transformer dans un diagramme d'activités UML / TURTLE, comme le montre la Fig.10. L'algorithme correspondant comporte trois étapes. Tout d'abord, deux tableaux sont créés pour isoler les différentes étiquettes du TRDD (*Partie_OK* et *Partie_KO*) et les informations temporelles sur les frontières temporelles (cf. tableau de la Fig.10). Les informations collectées dans les deux tableaux permettent alors de construire le cœur du diagramme d'activités de l'observateur, c'est à dire la partie chargée de l'observation et donc du relevé des violations de propriétés (temporelles). Cet embryon de comportement (cf. Fig.10) est composé, d'une part, de l'action de synchronisation sur la porte *Capture*, de l'étiquette de violation d'exigence définie dans le diagramme d'exigences (cf. section 3) et d'offres limitées dans le temps permettant d'observer la contrainte temporelle. Ce cœur du diagramme d'activités est construit en remontant progressivement les différentes frontières. Ainsi, l'observateur est assemblé « à l'envers ».

Par la suite, le diagramme d'activités de l'observateur est étendu (cf. diagramme d'activités de l'observateur à droite de la Fig.10) en respectant deux règles qui concernent :

- D'une part, le début et la terminaison du diagramme d'activités d'un observateur (cf. système anti-blocage de l'observateur sur la Fig.10). Un observateur doit rester passif durant la phase d'observation (Jard et al, 1988) et ne doit pas bloquer les objets observés. Pour respecter cette règle, il faut envisager un mécanisme pour ne pas bloquer l'observateur lorsque les actions observées apparaissent dans le désordre (exemple si « *Capture* » arrive avant « *Start* »).
- D'autre part, l'interruption de l'exécution des objets observés lorsqu'une exigence haute est violée (cf. arrêt de l'exécution du système sur la Fig.10). L'observateur doit alors stopper l'exécution de la vérification du système lorsque le niveau de criticité de l'exigence formelle est haut. Nous traduisons ce mécanisme en TIF par synchronisation avec tous les objets du système pour leur demander de stopper leur exécution.

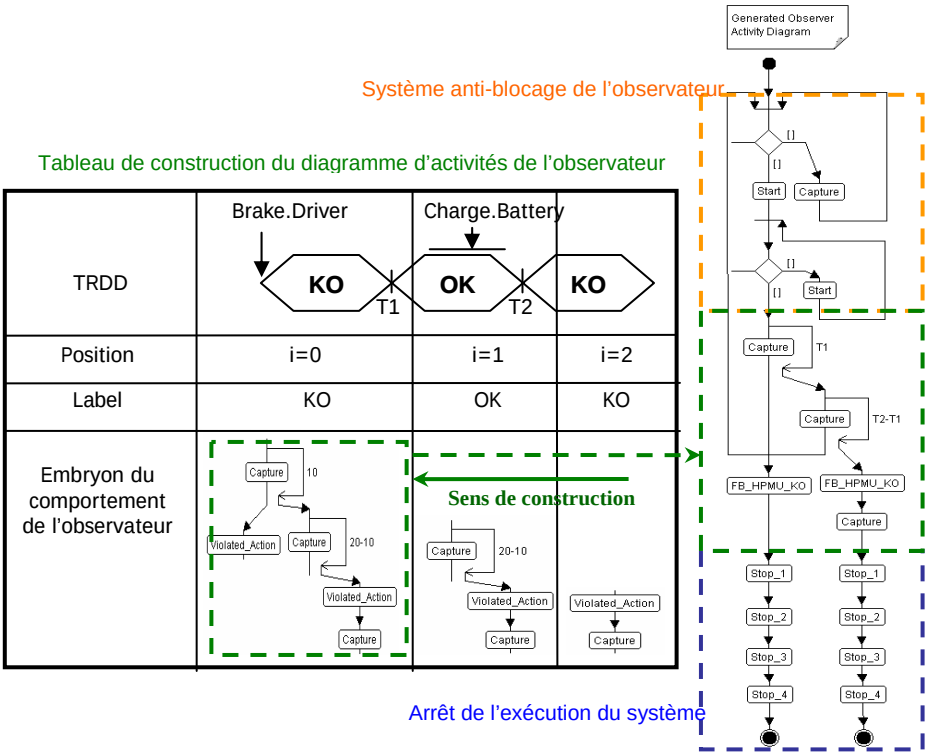

Fig.10. Génération de l'observateur Brake_HPMU_Observer à partir du TRDD

5.2.3 Métamodèle

La Fig.11 montre le métamodèle de la classe stéréotype « *TObserver* ». Un observateur est une Tclasse particulière (héritage de la classe *::metamod_Tnative ::Tclass* appartenant au meta-modèle de TURTLE natif).

L'observateur est composé, en plus des éléments présentés dans le DE (cf. section 3 et Fig.7) :

- D'un ensemble d'attributs (classe *Attributes_Obs*) qui correspondent aux valeurs des frontières temporelles du TRDD (classe *Date_of_FT*).
- D'au-moins trois portes de communication (classe stéréotypée « interface » *Gate_Obs*) composée de trois types de portes, qui sont :
  - Deux portes d'observation (classe *Observation_Gates*) qui se synchronisent avec les portes correspondantes aux points d'observations du TRDD.
  - Un ensemble de portes de contrôle d'exécution pour arrêter l'exécution du système (classe *Execution_Control*) dépendant du niveau de criticité de l'exigence temporelle formelle comme indiqué par l'association *generated_if_risk_is_high* et la formule OCL qui décrit cette association.
  - Une porte de notification de violation d'exigence (classe *Violated_Gate*) qui correspond à l'étiquette définie dans le diagramme d'exigences (voir section 3).
- D'un diagramme d'activités (classe *TAD_Obs*) construit à partir du TRDD (le métamodèle du diagramme d'activités d'un observateur est présenté dans la Fig.12).

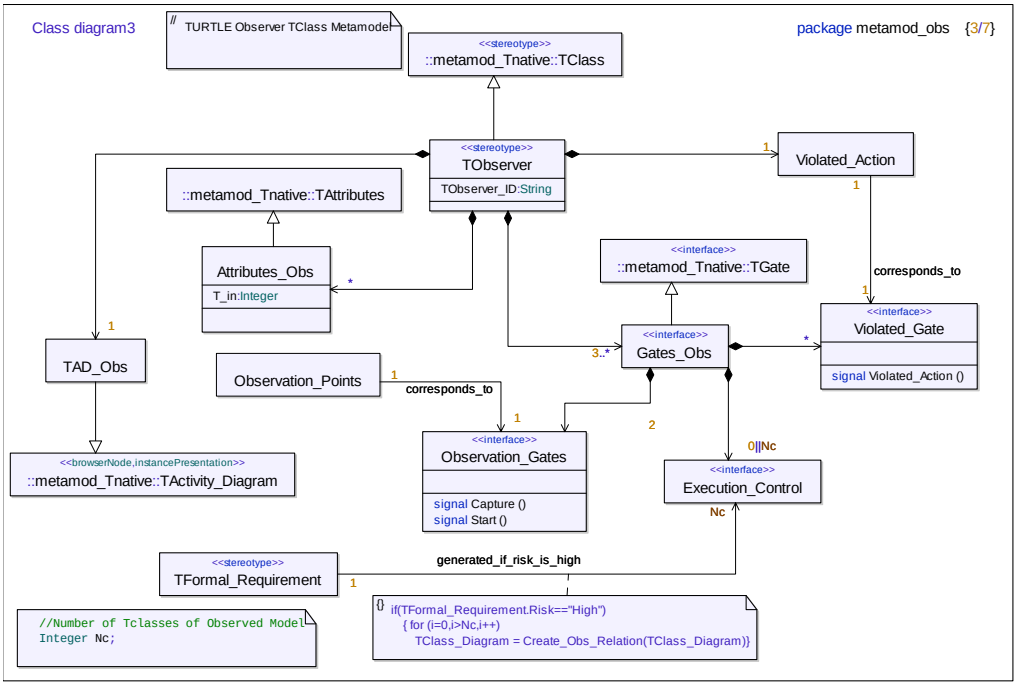

Fig.11. Métamodèle d'une classe stéréotypé « TObserver »

La Fig.12 décrit le diagramme d'activités d'un observateur (classe *TAD_Obs*) tel qu'on le construit à partir du TRDD. On notera la présence de l'association *is_built_upon* entre les classes *TAD_Obs* et *TRDD*.

Le diagramme d'activités de l'observateur est composé :

- D'une action de début (classe *::metamod_Tnative ::Start_AD*), comme un diagramme d'activités TURTLE en TIF, correspondant au symbole de début du TRDD (classe stéréotypée par « icon » *Begin*).
- D'un ensemble d'actions de fin (classe *::metamod_Tnative ::Stop_AD*) à l'identique d'un diagramme d'activités TURTLE classique.
- D'un ensemble de connecteurs (classe *::metamod_Tnative ::Connector*) de type choix (classe *::metamod_Tnative ::Choice*), jonction (classe *::metamod_Tnative ::Junction*) et boucle (classe *::metamod_Tnative ::Loop*).
- D'un ensemble d'actions de synchronisation (classe *::metamod_Tnative ::Synchronisation_Action*) qui correspondent soit aux portes d'observations (classe *Observation_Points*) qui seront synchronisées avec les portes des TObjets correspondant aux points d'observations (classes *::metamod_Tnative ::TGate*), soit à la porte de notification de violation d'exigence (classe *Violation_Gate*).
- De *1* à *N-1* (*N* correspond au nombre d'état de satisfaction/violation d'exigences voir Fig.5) offres limitées dans le temps (classe *::metamod_Tnative ::Time_Limited_Offer*) qui correspondent aux frontières temporelles du TRDD et dont l'élément de synchronisation *Capture* est synchronisé avec la porte de communication correspondante au point d'observation du symbole *Capture_Action*.

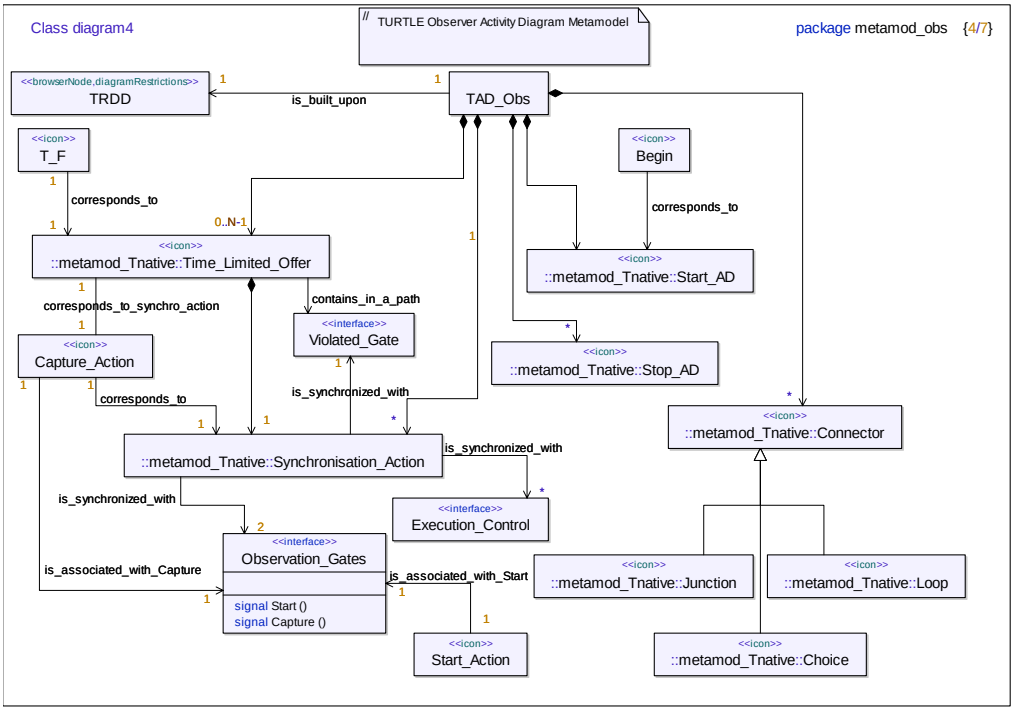

Fig.12. Métamodèle du diagramme d'activités de l'observateur

La Fig.13 décrit les règles de construction du diagramme d'activités d'un observateur. Ce diagramme commence par une action de début (classe *::metamod_Tnative ::Start_AD*) connectée avec un opérateur logique de choix (classe *::metamod_Tnative ::Choice*). Ce dernier est connecté soit à l'action de synchronisation *Start* (classe *Start_Action*) déclenchant la capture de l'exigence soit à l'action de synchronisation *Capture* (classe *Capture_Action*) notifiant la fin de capture d'exigence. Si ce dernier chemin est atteint, l'observateur reboucle au début du diagramme d'activités pour ne pas bloquer les objets observés. Dans l'autre cas, l'action de synchronisation *Start* est connectée avec un nouvel opérateur de choix pour anticiper sur une occurrence supplémentaire de la porte *Start*. Cette suite d'opérateurs de choix correspond à un dispositif anti-blocage de l'observateur applicable lorsque les actions observées apparaissent dans le désordre (exemple si *Capture* arrive avant *Start*») ou sont redondantes. Un fois appliqué ce dispositif, le diagramme d'activités de l'observateur correspond à l'embryon de comportement de l'observateur défini à partir du TRDD ; on a donc une succession d'offres limitées dans le temps (classe *::metamod_Tnative ::Time_Limited_Offer*). Enfin l'observateur interrompt le système et lui-même (classes *Execution_Control ::metamod_Tnative ::Stop_AD*) s'il observe la violation d'une exigence de criticité haute (cf formule OCL). Si l'exigence de criticité basse est violée, l'observateur reboucle au début après avoir pris soin de se synchroniser avec l'objet observé sur la porte *Capture* afin de ne pas bloquer les objets observés.

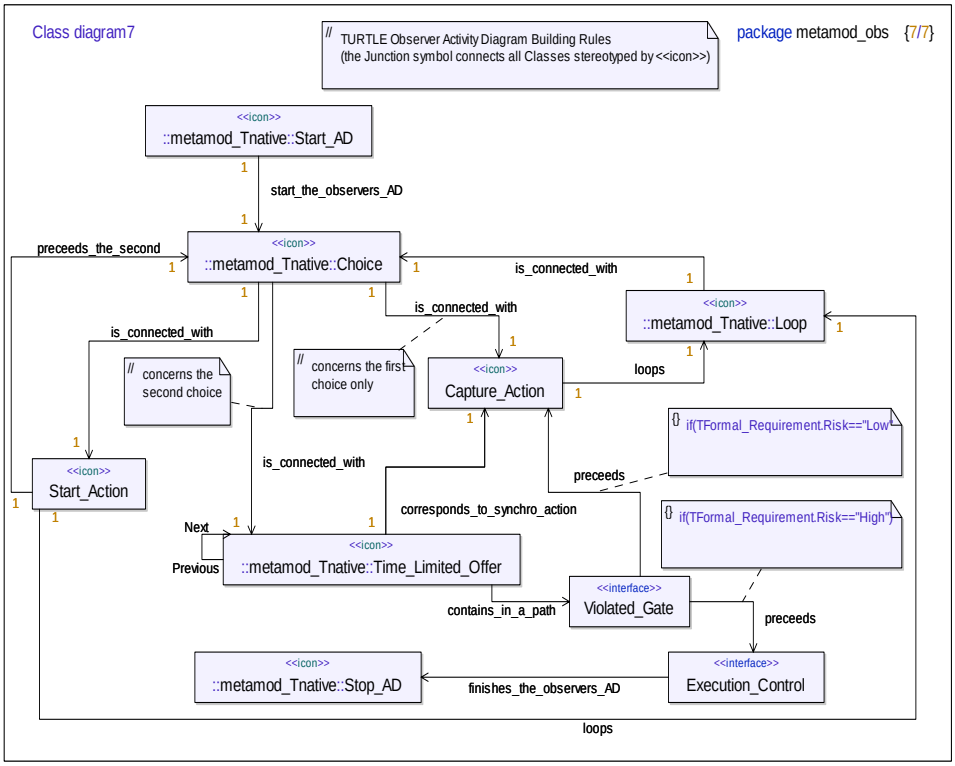

Fig.13. Métamodèle des règles de construction du diagramme d'activités d'un observateur

5.3 Intégration de l'observateur dans le modèle objet

5.3.1 Principe général

Deux étapes sont nécessaires dans la génération du nouveau modèle TURTLE comportant les observateurs :

- Connecter l'observateur aux autres classes/objets du système par des relations de synchronisations. Pour cela, les labels du TRDD (*start*, *capture*, etc.) se traduisent en des points d'observation des classes. Ainsi, ces labels servent de référence pour créer des relations de synchronisation entre l'observateur et les classes observées : pour cela, on utilise les points d'observations définis dans le TRDD qui correspondent à des synchronisations entrantes avec l'observateur qui les effectue de manière atomique (instantanée).
- Pouvoir interrompre l'exécution du système si une exigence est violée en modifiant les diagrammes d'activités des objets du système, en ajoutant un mécanisme de préemption. Ainsi, pour respecter l'interruption de l'exécution des objets observés si une exigence haute est violée il faut modifier les comportements des objets observés. Il est alors nécessaire de construire dans les objets du système (objets observés plus les autres) les mécanismes pour stopper l'exécution du système. Il faut donc construire des relations de préemption entre les différents objets du système et l'observateur déclenchant l'arrêt de l'exécution. De même les diagrammes d'activités des objets du système doivent être modifiés pour prendre en compte ce processus de préemption.

5.3.2 Métamodèle

La figure Fig.14 ci-dessous montre le métamodèle du package de conception TIF où il est possible de greffer un observateur permettant de se synchroniser avec le système observé. Le package TIF est composé d'observateurs et d'un package de conception qui est à observer (Classe *Observed_Design_Package*). Ce package est composé d'un diagramme de Tclasses composé lui-même d'un ensemble de Tclasses à observer (Classe *Observed_Tclass*) et de Tclasses qui ne sont pas observées (Classe *Non_Observed_Tclass*). Seules les classes intervenant dans les TRDDs, dont certaines portes sont définies par des points d'observations, sont observées dans le modèle (Classe *Observed_Tclass*).

Les Tclasses observées sont composées de portes observées (Classe *Observed_Gate* qui correspondent une à une aux points d'observations du TRDD (Classe *Observation_Point*) comme le montre l'association étiquetée par *corresponds_to*. Les portes observées sont ensuite synchronisées avec les portes de l'observateur (Classe *Gates_Obs* composée de la classe *Observation_Gates*). La synchronisation est indiquée par l'association étiquetée par *is_synchronized_with*.

Dans le cas d'une exigence de priorité haute, les portes de contrôle d'exécution de l'observateur (Classe *Execution_Control*) interrompent toutes les classes du système en TIF (classes observées, non observées et les observateurs). Si l'exigence est violée, l'exécution du système est donc stoppée (cf. l'association étiquetée par *interrupts* et le commentaire attaché sur la classe *Execution_Control)*.

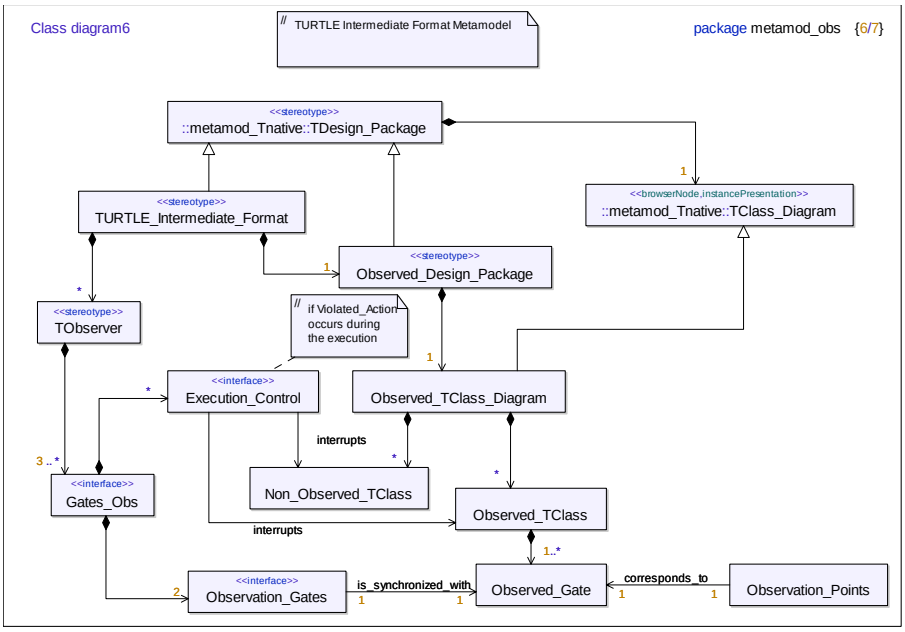

Fig.14. Métamodèle du package TIF incluant des observateurs

6 Positionnement par rapport aux travaux du domaine

En focalisant la discussion sur la vérification de systèmes temps réels à base d'analyse d'accessibilité, l'on peut distinguer deux grandes approches de vérification d'exigences temporelles ; à savoir le « contrôle de modèle » et la « vérification guidée par observateurs ».

L'approche par *contrôle de modèles* consiste à vérifier un modèle en termes d'assertions, par des formules de logique que l'on applique sur les traces d'exécutions (Clarke et al, 1999). Les formules de logique sont très souvent difficiles à mettre en œuvre quand la définition des exigences fait intervenir des « distances temporelles » entre événements (mis à part les logiques modales (Bradfield et al, 2006) très complexes à manipuler).

Pour pallier ce problème, plusieurs auteurs suggèrent de greffer des observateurs temporels au modèle du système et de vérifier l'ordre entre les événements à observer et ceux déclenchés par l'observateur. Cette technique reprise dans (Mallet et al 2006) fut utilisée pour Estelle (Jard et al, 1988) et SDL (Doldi, 2003) avant d'être transposée à des profils UML (Graff et al, 2005) (Dhaussy et al, 2007). Les travaux exposés en (Jard et al, 1988) (Doldi, 2003) (Graff et al, 2005) ne proposent pas de méthodologie automatisable pour la construction des observateurs, ces derniers étant bâtis à la « main ». En dehors de notre approche, seul (Dhaussy et al, 2007) établit un lien entre un langage graphique de description des exigences - CxUCC pour (Dhaussy et al, 2007) et TRDD pour ce qui nous concerne - et la synthèse d'observateurs. Par ailleurs, comme en (Hull et al, 2004), la méthodologie TURTLE (Fontan, 2008) place les exigences en amont de l'analyse alors que (Dhaussy et al, 2007) place des automates de contexte (*use case charts*) à l'intérieur de la phase d'analyse. Cette différence provient de l'intégration à TURTLE du fait que les diagrammes SysML

autorisent l'expression d'exigences fonctionnelles et non-fonctionnelles par opposition à UML dont les cas d'utilisation traitent uniquement l'aspect fonctionnel.

7 Conclusion

Limité jusqu'ici aux diagrammes d'analyse et de conception, le profil UML temps réel TURTLE a été récemment étendu par des diagrammes d'exigences. Inspirés de SysML, (SysML, 2007) ceux-ci permettent de dériver des exigences formelles à partir d'exigences informelles. En particulier, les exigences formelles temporelles sont exprimables dans un langage graphique appelé TRDD. Introduits par l'exemple dans cet article, le langage TRDD et les diagrammes d'exigences TURTLE n'en sont pas moins syntaxiquement définis par des métamodèles (Fontan, 2008).

La contribution de l'article porte sur le lien entre « exigences temporelles » et « vérification formelle » au travers d'algorithmes de synthèse automatique d'observateurs à partir de ces exigences.

Dans cet article, la synthèse d'observateurs prend pour point de départ une seule exigence temporelle. Le langage TRDD a été étendu en (Fontan, 2008) pour traiter la composition d'exigences temporelles. Les algorithmes de synthèse prenant en compte ce TRDD étendu restent à définir. Ce travail sera poursuivi dès lors que la validation expérimentale des algorithmes proposés dans cet article sera menée à terme dans l'environnement TTool.

Références

L. Apvrille, J.P. Courtiat, C. Lohr, et P de Saqui-Sannes (2004). TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit. *IEEE Transactions on Software Engineering*, 30:7:473-487.

L. Apvrille, P. de Saqui-Sannes, R. Pacalet, et A. Apvrille. Un environnement de conception de systèmes distribués basé sur UML. *Annales des Télécommunications*. 61:11/12:1347-1368.

J. Bradfield, et C. Stirling (2006). *Handbook of Modal Logic*, chapter Modal Mu Calculi. Elsevier.

E.M. Clarke, O. Grumberg, et D. Peled (1999), *Model Checking*. MIT Press, Cambridge, Mass.

J.P. Courtiat, C.A.S. Santos, C. Lohr, et B. Outtaj (2000). Experience with RT-LOTOS, a Temporal Extension of the LOTOS Formal Description Technique. *Computer Communications*. 23:12:1104-1123,

P. Dhaussy P., J.C. Roger, et F. Boniol (2007). Mise en œuvre d'unités de preuve pour la vérification formelle de modèles. *Ingénierie Dirigée par les Modèles* (IDM'07). Toulouse, France.

L. Doldi. (2003). *Validation of Communications Systems with SDL*. Wiley.

B. Fontan, S. Mota, P. de Saqui-Sannes, et T. Villemur (2007). Temporal Verification in Secure Group Communication System Design *International Conference on Emerging Security Information, Systems and Technologies* (SECURWARE 2007). Valencia, Spain.

B. Fontan, (2008). Méthodologie de conception de systèmes temps réel et distribués. Thèse de Doctorat (en préparation).

H. Garavel, R. Mateescu, F. Lang, et W. Serwe (2007). CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. Tool presentation at CAV'07 (19th *International Conference on Computer Aided Verification*). Berlin, Germany.

A. Gherbi, et F. Khendek (2006). UML Profiles for Real-Time Systems and their Applications, *Journal of Object Technology*. 5:3:149 – 169.

S. Graff, I. Ober, et I. Ober (2006). Validating Timed UML Models by Simulation and Verification. *Int. Journal On Software Tools for Technology Transfer*. 8:2:128-145.

E. Hull, K. Jackson, et J. Dick (2004). *Requirement Engineering*, 2nd Edition. Springer. ISBN 1-85233-879-2.

C. Jard, J.F. Monin et R. Groz (1988). Development of Veda, a Prototyping Tool for Distributed Algorithms. *IEEE Transactions on Software Engineering*.14:3:339-352.

F. Mallet, M. Peraldi-Frati, et C. André C (2006). From UML to Petri Nets for non functional Property Verification. *First IEEE Symposium on Industrial Embedded Systems* (IES'06). Antibes, France.

OMG (2007) Object Management Group. Unified Modeling Language. http://www.omg.org/docs/formal/07-02-05.pdf

SysML (2006) http://www.SysML.org/docs/specs/SysML-v1-Draft-06-03-01.pdf

TTool (2007) http://labsoc.comelec.enst.fr/turtle/

Summary

Unlike the UML 2.1 and SysML standards, the real-time UML profile TURTLE (Time UML and RT-LOTOS Environment) has a formal semantics and a methodology. The latter was developed with formal verification of real-time distributed system behavioural models in mind. TURLE was recently extended with a graphic and formal language dedicated to temporal requirement expression. The paper's contribution lies in the set of algorithms that we propose for synthesizing observers from temporal requirements expressed in that language. Such observers guide formal verification. In particular, the behaviour of objects is checked against temporal requirements. How the traceability of temporal requirements may be achieved throughout the system's life cycle is also an issue discussed in the paper. A battery load system serves as running example throughout the paper.