UML pour le temps réel :

l'exemple du régulateur d'allure.

Par François Terrier

Docteur ès Sciences en Informatique, Directeur de Recherche au CEA, Professeur à l'INSTN
Responsable du département d'ingénierie des logiciels et des systèmes (DILS) du CEA LIST de Saclay

et Sébastien Gérard

Ingénieur de l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique, Docteur ès Sciences en Informatique, Directeur de Recherche au CEA
Responsable du Laboratoire Ingénierie dirigée par les modèles pour les Systèmes Embarqués (LISE) du CEA LIST de Saclay

Avec sa richesse conceptuelle et notationnelle, UML offre une palette d'outil pour modéliser un système logiciel qui répond à la plus grande partie des besoins. Cette richesse est sa force, mais elle peut aussi parfois laisser l'utilisateur dans le doute sur le choix et le processus d'usage du langage. On pourrait définir autant de méthodologies qu'ils y a d'utilisateurs d'UML, et il est déraisonnable de prétendre en proposer une qui réponde à tous et se décrive en quelques pages. Cet article se contentera donc d'illustrer les capacités d'UML pour la modélisation des systèmes logiciels embarqués, sans prétendre fournir une description complète d'une méthodologie de développement. Toutefois, la structure linéaire naturelle des textes écrits amène à proposer un ordre de présentation. Celui adopté ici, est aussi simple que possible : il consiste à partir d'un cahier des charges simple et d'élaborer progressivement un modèle du système spécifié en UML aussi complet que possible pour pouvoir ensuite passer à la phase de codage avec le minimum d'ambiguïté.
La présentation s'appuie pour cela sur la méthode Accord/UML développée au LISE, Laboratoire d'ingénierie dirigée par les modèles pour les systèmes embarqués du CEA LIST.

De même, l'article est focalisé et limité à l'usage de la norme UML2, même si plusieurs extensions normalisées pourraient augmenter les capacités de modélisation, comme en particulier l'utilisation de SysML pour la traçabilité des exigences et l'utilisation de la nouvelle norme MARTE dédiées aux aspects temps réel embarqués. Ces point pourront faire l'objet d'articles complémentaires.

1 L'ingénierie dirigée par les modèles, un exemple d'étude

Certains véhicules sont équipés d'un système capable de réguler la vitesse courante de la voiture autour d'une vitesse, dite vitesse de consigne. Cette vitesse est fixée par l'utilisateur et représente la vitesse à laquelle il désire rouler automatiquement. Cet exemple se veut simple mais cependant suffisamment représentatif des problèmes que l'on peut rencontrer dans la modélisation d'applications embarquées pour une automobile. C'est un système qui doit être réactif, qui présente des comportements cycliques et qui doit être en forte interaction avec son environnement.
Le système que l'on veut réaliser est donc un régulateur d'allure capable de maintenir la vitesse d'un véhicule à une vitesse consigne cible fixée par le conducteur. Le maintien de la vitesse s'effectuera par envoi d'une variation de couple au système de contrôle du moteur. La loi de commande que le régulateur d'allure utilise pour calculer la variation du couple est la suivante :

$$\delta C = \frac{\arctan(k \times (Vcible - Vvéhicule))}{2}$$

Ainsi, si *Vcible>Vvéhicule* alors $\delta C>0$ et le véhicule accélère. Sinon, *Vcible*$\leq$*Vvéhicule* alors $\delta C \leq 0$ et le véhicule ralentit.
La mesure de la vitesse du véhicule s'effectue à l'aide d'un compteur de vitesse pourvu de son propre système d'affichage. La mesure est réalisée de façon cyclique à la fréquence de *2 Hz*. La mise à jour de l'affichage de la vitesse s'effectue au même rythme. On suppose que ce compteur est capable de fournir la valeur courante de la vitesse du véhicule en m/s sous la forme d'un entier.
La mise en marche de la régulation nécessite l'allumage préalable du système régulateur d'allure, puis l'activation de la régulation. L'allumage ou l'extinction du régulateur d'allure s'effectuent à l'aide d'un signal émis par un bouton poussoir *Allumer/Éteindre* actionné par le conducteur. Si le régulateur d'allure est allumé, il s'éteint. Si le régulateur d'allure est éteint, il s'allume. L'action d'allumage ou d'extinction du régulateur d'allure doit être effectuée en moins de *1s* suite à une action sur le bouton poussoir *Allumer/Éteindre* du régulateur d'allure.
L'activation (mise en marche) de la régulation se fait ensuite via un autre bouton poussoir *Marche/Arrêt*. La régulation de la vitesse doit démarrer au plus tard *0,5 s* après la demande d'activation de la régulation. La régulation du véhicule ne peut être mise en route que si la vitesse du véhicule est au moins égale à *50 km/h* et si le système régulateur d'allure a été préalablement allumé. Lors de l'activation de la régulation, la vitesse de consigne est définie comme égale à la vitesse courante du véhicule au moment où la régulation est activée.
La désactivation (arrêt) de la régulation peut se faire de quatre façons différentes :
- soit implicitement, lorsque le conducteur appuie sur la pédale de frein ;
- soit explicitement par action du conducteur sur le bouton *Marche/Arrêt* du régulateur d'allure ;
- soit si la vitesse du véhicule devient inférieure à *50 km/h*.
- soit par extinction du régulateur d'allure via action du conducteur sur le bouton *Allumer/Eteindre* du régulateur d'allure ;

Les temps de réponse attendus pour l'arrêt de la régulation sont : de *0,5 s* par appui sur le frein ou par action sur le bouton *Marche/Arrêt*; ou le bouton Allumer/Eteindre ; et de *100 ms* si la vitesse du véhicule devient inférieure à *50 km/h*.
De plus, lorsque le conducteur appuie sur la pédale d'accélération alors que la régulation est activée, la régulation de vitesse est suspendue jusqu'à ce que le conducteur relâche l'accélérateur. La suspension doit être réalisée en au plus *200 ms* et la reprise en au plus 250 ms. Le système reprend alors le contrôle de la vitesse du véhicule et l'amène progressivement à la vitesse de consigne fixée avant l'accélération.
En cas de simultanéité des actions de décélération et d'accélération, c'est l'action de décélération qui doit être prise en compte prioritairement. La régulation doit alors être arrêtée et non suspendue avec le délai d'un arrêt par appui sur la pédale de frein.
Le régulateur d'allure est connecté à un écran qui affiche la vitesse de consigne et l'état d'activation de la régulation (désactivée, activée ou suspendue). Il fonctionne à la demande avec un temps de réponse attendu de 0,5 s.
Afin de visualiser fonctionnement du système on peut représenter le système et son interaction avec le véhicule par le schéma (simpliste) de la Figure 1.

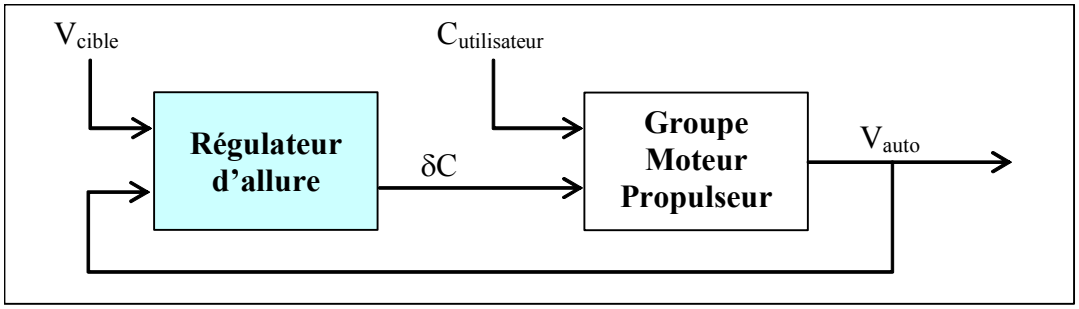

Figure 1 : Schéma de la boucle de commande du système régulateur d'allure.

2 Le modèle d'analyse préliminaire

Cette première étape d'analyse préliminaire occupe une place importante dans le cycle de développement d'un projet. En effet, elle peut être vue comme étant une étape de reformatage et formalisation du cahier des charges du projet à l'aide d'un langage graphique et textuel facilement compréhensible par tous. De plus, cette étape est l'occasion pour le futur développeur du système de se familiariser avec le vocabulaire et les concepts spécifiques au domaine des applications visées.
Durant la phase de spécification préliminaire d'un système, le développeur met en œuvre trois types de diagramme UML pour décrire son application : le diagramme de cas d'utilisation, le diagramme de communication et le diagramme de séquences. Par ailleurs, afin de répondre entre autres à des besoins de traçabilité, nous proposons de définir au préalable un dictionnaire des concepts-clés du domaine d'application auquel le système à développer appartient.

2.1 Définition du dictionnaire

Un dictionnaire a pour objectif de collecter l'ensemble des concepts-clé du domaine et de l'application visée. Il contient des noms, des verbes et des adjectifs de la terminologie employée par les acteurs du domaine d'application [2].
Le dictionnaire va servir de support à l'analyse préliminaire dans l'identification des différents concepts de modélisation de l'application, comme par exemple les classes, les attributs ou les relations. Pour le décrire, on répertorie l'ensemble des termes servant à décrire l'application dans le cahier des charges suivant trois critères :

- Les noms - ils vont permettre de définir des concepts de type classe ou acteur ;
- les qualificatifs - ils peuvent être associés aux concepts d'attribut ou de relation ;
- les verbes – ils sont souvent traduits sous la forme d'opération et imposent souvent une relation entre deux concepts.

Enfin, on ajoute une colonne supplémentaire contenant la description des contraintes et des propriétés temps-réel éventuellement associées aux concepts du cahier des charges. Pour un même nom du dictionnaire, on associe les qualificatifs et verbes qui lui sont liés dans le cahier des charges. Pour l'exemple du régulateur d'allure, l'étude du cahier des charges nous permet d'extraire le dictionnaire de notre système régulateur d'allure tel que décrit dans le Tableau 1.

Tableau 1 - Dictionnaire de l'application de régulation de la vitesse

Nom (classe ou	Qualitatif (attribut ou	Verbe (opération)	Contrainte temps-réel

acteur)	**relation)**		
Régulateur d'allure		allumer, éteindre (le système régulateur d'allure)	1 s
		activer/démarrer (la régulation)	0,5 s
		désactiver/arrêter (la régulation)	0,1 s si vitesse < 50 km/h 0,5 s si frein ou bouton Marche/Arret ou Allumer/Eteindre
	(relation avec l'affichage)	mettre à jour (l'affichage)	
		suspendre (la régulation)	200 ms
		reprendre (la régulation)	250 ms
	Vitesse de consigne	fixer (la vitesse de consigne)	
Compteur de vitesse	Vitesse courante	acquérir (la vitesse courante)	2 Hz
		afficher (la vitesse courante)	2 Hz
Pédale de frein		presser, relâcher	
Pédale d'accélération		presser, relâcher	
Bouton Allumer / Eteindre		allumer, éteindre (le système régulateur)	
Bouton Marche/Arrêt		activer/démarrer, désactiver/arrêter (la régulation)	
Ecran d'affichage		afficher (état et vitesse consigne)	0,5 s
Loi de commande		calculer	

La description du dictionnaire peut également servir de phase d'apprentissage ou d'introduction du domaine d'application pour le développeur. En effet, le développeur va entre autres relever tous les termes relatifs au domaine d'application visé et par conséquent acquérir la signification de chacun d'eux. Ainsi, il s'imprègne d'une part du vocabulaire du domaine et d'autre part de la philosophie de travail des experts du domaine. Cette étape permet d'accroître la synergie entre les développeurs et les experts du domaine, synergie qui est nécessaire au bon déroulement du développement et surtout qui permet de se placer dans les meilleures conditions pour que le futur produit du développement réponde au mieux aux besoins de l'utilisateur.
Enfin, le dictionnaire peut servir de base pour alimenter un outil spécifique chargé de tracer les exigences à travers les différentes phases et les différents modèles d'une application.

2.2 Le diagramme des cas d'utilisation

Le diagramme de cas d'utilisation d'UML a pour but de décrire les usages requis (les exigences) d'un système à développer. Ainsi, la plupart des méthodes d'ingénierie dirigée par les modèles et utilisant UML débutent par une description des cas d'utilisation du système à développer.
Les concepts principaux manipulés dans ces diagrammes sont les sujets (« subject »), les acteurs (« Actor ») et les cas d'utilisation (« UseCase »).
Définitions UML :

- Le sujet référence en fait le système considéré et dont on cherche à décrire les cas d'utilisation ;
- un cas d'utilisation décrit le comportement requis d'un système ou toute autre entité sémantique sans en dévoiler la structure interne. Un diagramme de cas d'utilisation possèdera en général au moins un cas d'utilisation ;
- un acteur modélise toute autre entité différente du sujet (c.a.d. le système considéré), et interagissant avec celui-ci. Un acteur peut ainsi représenter les utilisateurs du système ou les autres systèmes interagissant avec le système sujet.

Comme décrit dans [3] et [4], la construction d'un diagramme de cas d'utilisation est intéressante pour trois raisons. Premièrement, elle permet à un expert d'un domaine de spécifier un système avec son point de vue métier et ceci de façon suffisamment précise pour qu'un développeur puisse transformer cette spécification en une réalité, un système opérationnel. Deuxièmement, les éléments constituant les diagrammes de cas d'utilisation (essentiellement acteurs et cas d'utilisation) forment un moyen d'expression simple et commun aux différentes personnes intervenant dans la vie d'un système (experts du domaine d'application, développeurs de systèmes informatiques et utilisateurs). Ainsi, des personnes possédant un point de vue différent d'un système peuvent échanger des idées dans le but d'améliorer le développement de ce système. Enfin, les cas d'utilisation peuvent servir de base pour valider l'implantation d'un système.
A ce stade du développement d'un système, l'application que l'on veut modéliser est regardée comme une boîte noire pour laquelle seule la description des principales fonctions fournis par le système sujet, son environnement et ses interactions avec celui-ci nous intéresse.
La description des cas d'utilisation (« use case ») peut se dérouler autour de trois tâches principales :

- Identifier les acteurs et les relations entre eux ;
- identifier les cas d'utilisation et les relations entre eux ;
- identifier les connexions entre acteurs et cas d'utilisation.

2.2.1 Identification des acteurs

Puisqu'un système ne fonctionne jamais seul, le premier travail du développeur est d'identifier les « objets » externes au système interagissant avec lui. Ces objets sont appelés des acteurs.
Les acteurs peuvent être de nature différente : vivant (humain ou non), électronique (par exemple des capteurs) ou encore logiciel. Leur représentation prend la forme schématique d'un personnage.
La recherche des concepts d'une application qui vont être des acteurs peut se faire depuis trois sources d'information : l'expert du domaine, le cahier des charges

(normalement rédigé par l'expert du domaine) et, dans notre cas, le dictionnaire qui est un premier filtre du cahier des charges.
Pour notre exemple, on prend la colonne des noms du dictionnaire (Tableau 1) qui contient les concepts les plus significatifs de l'application, issus du cahier des charges, et on isole les concepts extérieurs au système que l'on désire modéliser.

Identification des concepts du cahier des charges candidat au statut d'acteur.

« pour calculer la commande à envoyer au **moteur** »
« Ensuite, ce couple moteur est transmis à la chaîne de **traction du véhicule** »
« on pourra regrouper **le contrôle moteur** et **la chaîne de traction** sous un même organe, le **groupe moteur-propulseur** »
« L'allumage ou l'extinction du régulateur d'allure s'effectuent à l'aide d'un signal émis par un **bouton poussoir *Allumer/Éteindre*** actionné par le conducteur »
« lorsque le conducteur appuie sur **la pédale de frein** »
« De plus, lorsque le conducteur appuie sur **la pédale d'accélération** »
« Le régulateur d'allure est doté d'un **écran** permettant **l'affichage** »

Ces concepts (en gras ci-dessus) sont ensuite renommés de manière à leur associer un nom informatique qui pourra être exploité lors des phases ultérieures de modélisation et d'implantation. Le nom « informatique » est un nom ne contenant ni accent, ni espace.
Par exemple, pour l'expression « compteur de vitesse », on associe la chaine de caractère « CompteurVitesse ». Cette règle permet de faciliter une analyse de traçabilité de l'application et favorise le raffinement itératif du modèle depuis la description de ses exigences jusqu'à son implantation.
Le tableau suivant (Tableau 2) contient la liste des sept concepts principaux de l'application qui sont considérés comme externes au système de contrôle/commande de la vitesse et interagissant directement avec lui et qui, par conséquent, sont identifiés comme des acteurs du système. La deuxième colonne de ce même tableau contient le nom « informatique » associé à chaque acteur.

Tableau 3 – Noms des acteurs du régulateur d'allure

Acteurs du système	Nom « informatique » des acteurs
le groupe moteur/propulseur[1]	GroupeMoteurPropulseur
le compteur de vitesse	CompteurVitesse
la pédale d'accélération	PedaleAcceleration
la pédale de frein	PedaleFrein
le bouton Allumer/Eteindre du régulateur d'allure	AllumerEteindreRegulateurAllure
Le bouton de Marche/Arrêt de la régulation	MarcheArretRegulation
l'écran d'affichage	Afficheur

A partir du tableau des acteurs du système, le premier diagramme UML peut commencer à être construit, le diagramme des cas d'utilisation (Figure 1). Le système y occupe la place centrale sous la forme d'un rectangle blanc contenant le

[1] Comme le suggérait le cahier des charges, le moteur et la chaîne de traction ont été regroupés en une seule entité, le groupe moteur-propulseur.

nom de l'application développée. Les acteurs identifiés sont alors disposés autour du rectangle modélisant le système avec lequel ils interagissent dans le cadre de l'application développée.

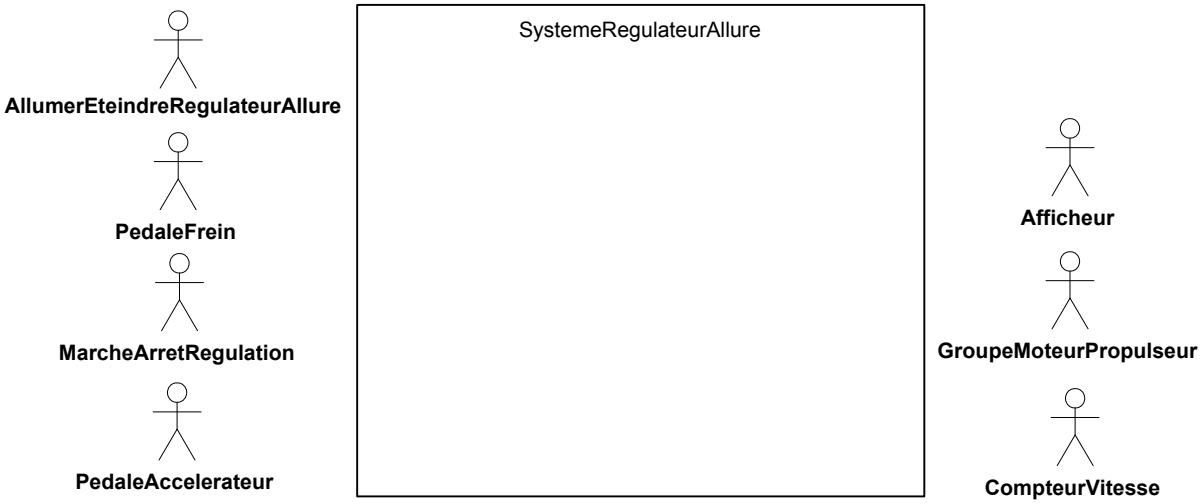

Figure 1 - Les acteurs du diagramme de cas d'utilisation du système régulateur d'allure

A l'issue de cette première étape de description du diagramme des cas d'utilisation, la frontière délimitant l'intérieur du système et l'extérieur du système est totalement et clairement identifiée. Ceci fait, puisque le système est clairement identifié, il est maintenant possible d'identifier les cas d'utilisation du système, c.a.d. les fonctions principales que le système devra réaliser.

2.2.2 Identification des cas d'utilisation

L'identification des fonctionnalités principales d'un système se fait depuis les mêmes sources d'information que précédemment : l'expert du domaine, le cahier des charges et le dictionnaire.
Les grandes fonctions du système que l'on identifie dans le cahier des charges sont en général associées aux verbes listés dans le dictionnaire pour le concept clé du système, ici le régulateur. Pour notre exemple, l'analyse du cahier des charges nous permet d'identifier les six grandes fonctionnalités que le système doit offrir (cf. encadré ci-après) : allumer/éteindre le régulateur d'allure, activer/désactiver/suspendre/reprendre la régulation de la vitesse.

Identification des concepts du cahier des charges candidat au statut de cas d'utilisation.

« nécessite l'**allumage** préalable du système régulateur d'allure »
« ... L'action d'allumage ou d'extinction du régulateur d'allure »
« L'activation (mise en marche) de la régulation »
« La désactivation (arrêt) de la régulation »
« la régulation de vitesse est **suspendue** jusqu'à »
« Le système **reprend** alors le contrôle »
« Le régulateur d'allure est connecté à un écran qui **affiche** »

Ces fonctions principales nous permettent alors d'identifier les cas d'utilisation de notre système. On aboutit alors au diagramme de la figure 2 décrivant les cas d'utilisation de notre exemple.

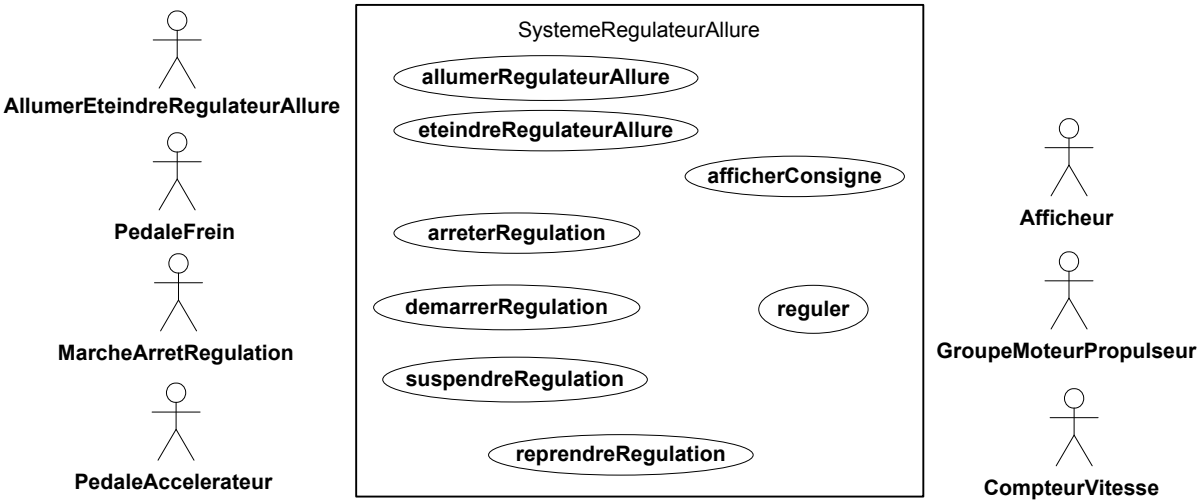

Figure 2 - Les cas d'utilisation du diagramme de cas d'utilisation du système régulateur d'allure

Remarque : en UML, les cas d'utilisation peuvent être reliés par des relations de généralisation, d'inclusion (« include ») ou d'extension (« extend ») :

- La relation de généralisation entre deux cas d'utilisation a la même signification que celle entre deux classes.
- La relation d'inclusion est un moyen de réaliser une factorisation comportementale d'un système. Elle signifie que le cas d'utilisation initial contient explicitement le comportement d'un autre cas d'utilisation à un point donné de sa séquence. Ici, elle servira à indiquer que les différents cas d'utilisation font appel à l'utilisation pour affichage de la consigne du régulateur et que les deux opérations d'activation de la régulation (demarrerRegulation et reprendreRegulation) font appel à une même utilisation du système pour réguler..
- la relation d'extension est utilisée pour modéliser les comportements optionnels d'un cas d'utilisation. Le cas d'utilisation possède alors un point d'extension. Un cas d'utilisation qui possède un point d'extension présente alors deux situations. Dans la première situation, l'extension n'est pas prise en compte, alors que dans la deuxième situation, le cas d'utilisation décrit par

l'extension est inclus dans le premier. Ici, elle servira à indiquer que le cas d'extinction du régulateur d'allure peut faire appel au cas d'arrêt de la régulation (en l'occurrence, si celle est encore active au moment de l'extinction du régulateur d'allure).

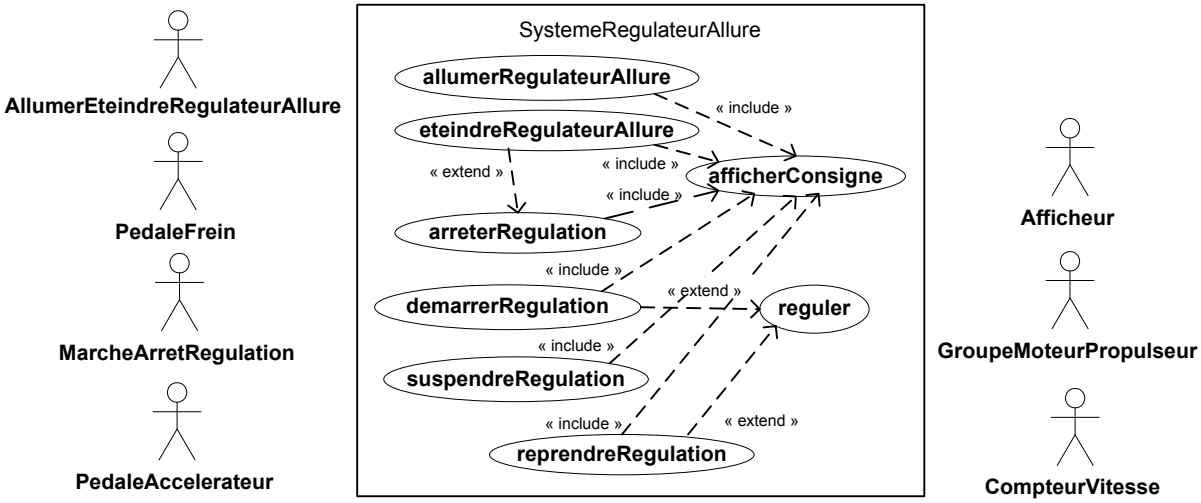

Figure 3 - Les relations entre cas d'utilisation du diagramme de cas d'utilisation du régulateur d'allure

2.2.3 Identification des relations entre acteurs et cas d'utilisation

La troisième et dernière étape de la construction du diagramme des cas d'utilisation consiste à identifier les relations entre les acteurs et les cas d'utilisation du système. Une relation entre un acteur et un cas d'utilisation marque la présence de communication(s) entre les entités en relation pour réaliser la fonction supportée par le cas d'utilisation. Il faut noter que cette relation est structurelle. Elle ne spécifie donc pas la nature de la communication elle même, c.a.d. cette relation ne décrit pas quelles sont les données ou commandes échangées par exemple, mais uniquement la présence de communication entre les deux entités mis en relation.
Comme décrit dans la figure suivante, une analyse du cahier des charges et des verbes identifiés dans le dictionnaire nous permet de compléter le précédant diagramme de cas d'utilisation du régulateur d'allure.

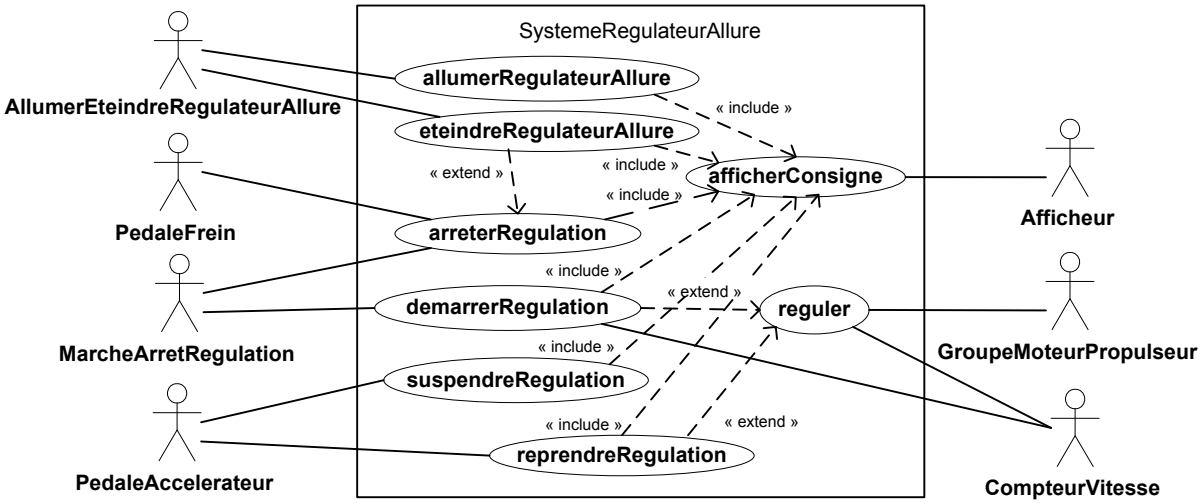

Figure 4 - Diagramme complet de cas d'utilisation du régulateur d'allure.

Dans le diagramme des cas d'utilisation, on a identifié les points de communication entre le système et son environnement, mais les détails de ces communications, essentiellement le sens et la nature de la communication, restent inconnus à ce stade. La prochaine section consiste donc à répondre à ces questions.

2.3 Détails des communications entre le système et son environnement

Après avoir construit le diagramme de cas d'utilisation (Figure 4), on construit un diagramme de communication impliquant les acteurs du système (c.a.d. l'environnement) et le système lui-même. Par ce diagramme, on cherche à fournir une vue d'ensemble des flux d'information échangés entre le système et son environnement.
Le diagramme de communication fait partie de la famille des diagrammes d'interactions de UML. Comme son nom l'indique, le diagramme de communication a pour objectif de mettre en valeur les échanges d'information entre les entités d'un système.
Dans notre exemple, le diagramme de communication décrit en figure 4 précise les interactions entre le régulateur d'allure et son environnement :

- Le régulateur d'allure reçoit les informations d'allumage ou d'extinction du régulateur d'allure via un signal *AERegSyst* ;
- les informations d'activation ou de désactivation de la régulation de vitesse via un signal *MARegulation* ;
- les messages d'enclenchement et de relâche de la pédale d'accélération, ainsi que l'enclenchement du frein sont pour l'instant nommés *PAEnc*/*PARel* et *PFE* ;
- l'acquisition de la vitesse courante du véhicule apparaît comme un flot entrant du compteur de vitesse vers le système ;
- les actions sur le groupe moteur propulseur et sur l'écran d'affichage font apparaître deux flots sortants, respectivement variation de couple (*coupleReg*) et vitesse de consigne/état d'activation (*consigneReg*).

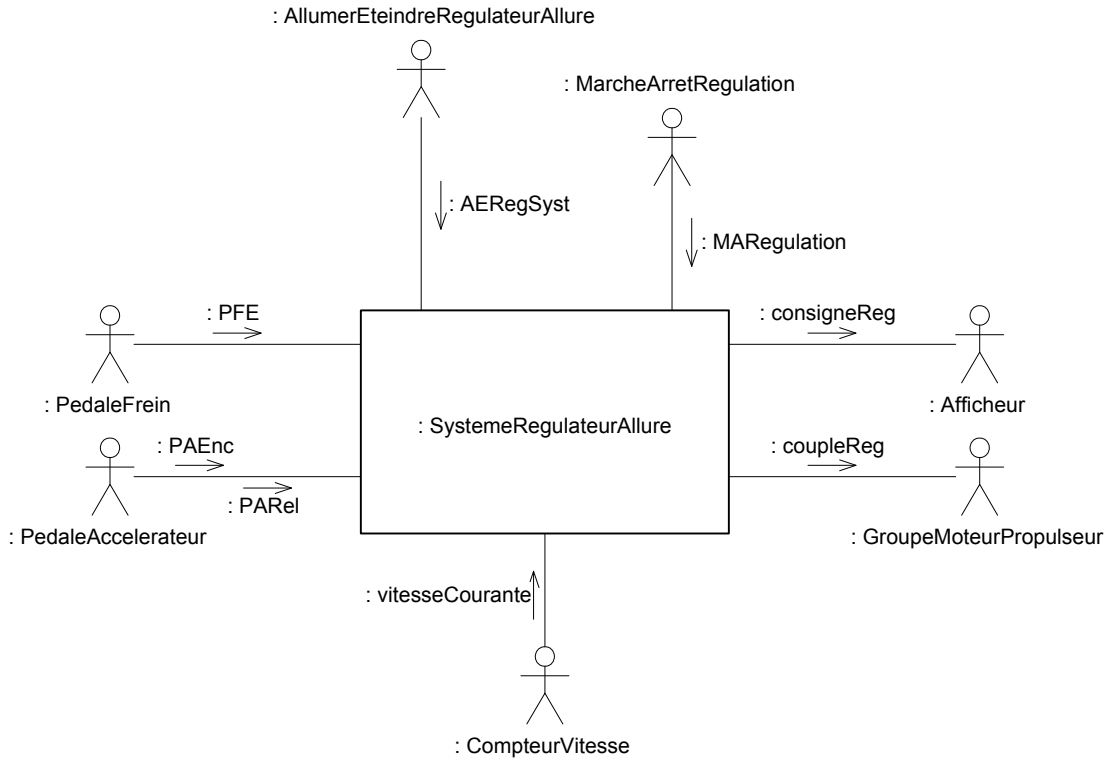

Figure 5 - Diagramme d'échange de flux du régulateur d'allure

2.4 Scénarii d'analyse préliminaire

Pour chaque cas d'utilisation identifié, on répertorie puis on détaille les différents scénarios possibles pour chaque cas d'utilisation. Les scénarii sont à voir comme des instances des cas d'utilisation. La description des scénarii se fait en deux étapes. On commence par décrire de façon textuelle les besoins. L'objectif de cette étape est de rassembler toute l'information, relative à la fonction représentée par le cas d'utilisation, et disponible, soit auprès des spécialistes du domaine, soit dans les documents constituant le cahier des charges. Ensuite, on traduit cette forme textuelle de représentation d'un scénario en un diagramme de séquence. Cette étape se focalise sur la description des séquences de messages échangés entre le système et les acteurs constituant son environnement pour un scénario donné d'un cas d'utilisation donné.
Par exemple pour le cas d'utilisation *arreterRegulation*, on identifie trois scénarii possibles :

- Scénario 1, arrêter la régulation de la vitesse par l'intermédiaire du bouton Marche/Arrêt du régulateur d'allure ;
- scénario 2, arrêter la régulation de la vitesse suite à un freinage ;
- et scénario 3, arrêter la régulation de la vitesse suite à l'extinction du régulateur.

Une analyse du cahier des charges de notre régulateur d'allure, nous donne pour le premier scénario la description textuelle décrite dans l'encadré ci-après.

Cas d'utilisation *arreterRegulation* – scénario *arrêter la régulation de la vitesse par l'intermédiaire du bouton Marche/Arrêt du régulateur d'allure*.

Dans le scénario 1, l'environnement envoie un message au système pour l'informer de l'appui sur le bouton Marche/Arrêt du régulateur. Le système procède alors à l'arrêt de la régulation et met à jour son afficheur en moins d'une demi-seconde.

Le diagramme de séquence correspondant à ce scénario est alors décrit en figure 5.

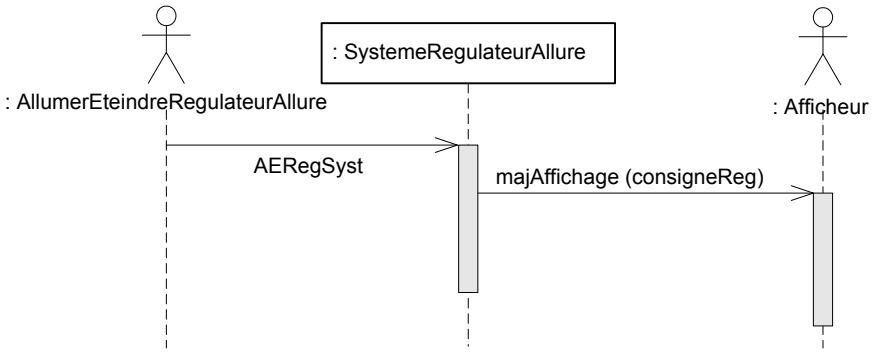

Figure 6 - Diagramme d'interaction du scénario *arrêter la régulation de la vitesse par l'intermédiaire du bouton Marche/Arrêt du régulateur d'allure* du cas d'utilisation *arreterRegulation*

On notera que c'est dès ce niveau de modélisation que sont décrites les contraintes temps-réel présentes dans le cahier des charges. Ici, la notation utilisée est celle des contraintes préconisées par UML, c'est-à-dire une expression textuelle entre accolades. En effet, pour exprimer une telle contrainte de temps sur un diagramme de séquence, UML propose deux mécanismes particuliers permettant de déclarer des observations temporelles : un instant du temps (comme par l'exemple l'instant d'émission d'un message ou de démarrage d'une exécution) ou une durée (comme par exemple la durée pour un message d'aller de son émetteur à son récepteur ou la durée d'exécution liée à une action). L'observation d'un instant est ainsi décrite par une chaine de caractère décrivant un nom de variable attaché à l'élément de modèle annoté par un trait fin en pointillé. Dans l'exemple de la figure 6, l'observation temporelle *@t1* dénote l'instant auquel le régulateur d'allure reçoit le message nommé *AERegSyst. @t2* marque l'instant de fin de l'exécution de l'action effectuée par l'afficheur du régulateur à la réception du message nommé *majAffichage*.
La contrainte de temps spécifiée entre accolades sur l'exemple suivant spécifie ainsi que l'ensemble des traitements exécutés par la tâche initiée par le message *arreterRegulation* doit s'exécuter en moins de 500 millisecondes.

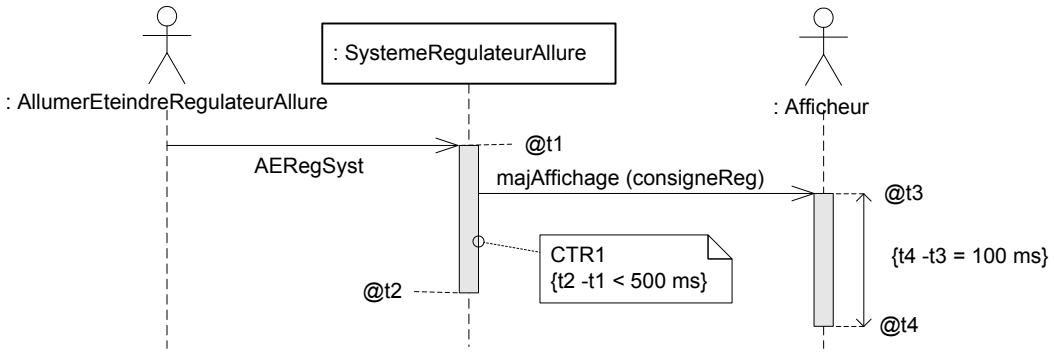

Figure 7 - Spécification d'une contrainte de temps via des marques temporelles

2.5 Résumé de l'analyse préliminaire

A la fin de cette étape d'analyse préliminaire, le modèle global de notre application comporte donc:

- Un dictionnaire ;
- un diagramme de cas d'utilisation ;
- un diagramme de communication global ;
- des diagrammes de séquences.

La construction du dictionnaire a permis au concepteur du système de se familiariser avec le vocabulaire du domaine d'application pour lequel il va travailler. De plus, ce dictionnaire sera également une source importante d'information qui alimentera le processus de découverte des concepts à introduire dans les modèles ultérieurement.
Dans le diagramme de cas d'utilisation décrit en section 2.3, on a spécifié les acteurs, les principales fonctions du système (appelées cas d'utilisation) et les relations structurelles entre ces différents éléments. La découverte des acteurs (qui vont constituer l'environnement du système) a permis de clairement identifier la frontière du système. La délimitation claire et explicite de cette frontière favorise l'identification des fonctions que le système doit offrir à son environnement, en particulier à ses utilisateurs. Ensuite, on peut associer le système à son environnement en spécifiant les relations entre les acteurs et les cas d'utilisation.
Dans le diagramme de communication décrit en section 2.4, on retrouve les acteurs et le système (vu alors comme un objet unique). On spécifie dans ce diagramme de communication les flux de messages échangés entre le système et son environnement afin de fournir une vue globale des échanges réalisés aux frontières de notre application.
Enfin, les différents scénarios possibles pour chaque cas d'utilisation du système sont décrits sous forme textuelle, puis sous forme graphique par le biais de diagrammes de séquence. Ces derniers permettent de spécifier les séquences de messages échangés entre le système et son environnement pour réaliser une fonction (c.a.d. cas d'utilisation) donnée dans un contexte donné. A ce niveau de la modélisation et ceci chaque fois que c'est possible, les diagrammes de séquence doivent posséder des contraintes temps-réel.
La phase d'analyse préliminaire nous a donc permis de réécrire sous une forme plus formelle (ou parfois, juste d'écrire) le cahier des charges de l'application. Elle a permis de mettre en place des modèles à la frontière entre une vue fonctionnelle (souvent celle des experts du domaine) et une vue objet. Le point de vue objet est l'orientation adoptée pour le reste du développement. La phase d'analyse préliminaire a mis en œuvre très peu de concepts, et surtout, elle reste très intuitive.

De ce fait, elle peut être réalisée aussi bien par un expert du domaine d'application (le temps d'apprentissage aux concepts et diagrammes manipulés est court), que par un développeur spécialisé. Cette étape de modélisation aura également permis d'établir des passerelles de communication entre les différents acteurs participant au développement du système, ce qui est en faveur d'une meilleure qualité de développement et surtout améliore l'adéquation entre le produit réalisé et le produit imaginé.
Enfin, elle aura permis de poser des bases formelles nécessaires au reste de la modélisation de l'application. En effet, on a maintenant un modèle clair et non ambiguë des besoins de l'utilisateur sous un point de vue plutôt fonctionnel. L'objectif de la prochaine étape est de construire le modèle détaillé d'analyse de l'application à partir des modèles issus de cette étape d'analyse préliminaire. Durant cette prochaine étape, il faut maintenant travailler pour répondre à la question : « Que doit faire mon système du point de vue de son fonctionnement interne ? »
Pour cela, on propose d'organiser la constitution du modèle global d'une application autour de trois modèles complémentaires et cohérents : le modèle structurel, le modèle des comportements et le modèle des interactions détaillées. La prochaine section consiste ainsi en la description de ces trois modèles (respectivement sections 3.2, 3.3 et 3.4). On insistera plus particulièrement sur les points d'interaction entre chacun des 3 sous-modèles qui, ensemble, constituent la spécification détaillée complète d'une application.

3 L'architecture d'une application

La description du modèle d'architecture d'une application repose sur une architecture générique proposée ci-après en section 3.1 et s'effectue en six étapes :

a. Extraction des concepts-clés de l'application qui définiront les classes.
b. Définition des dépendances entre ces concepts : relations d'héritage et d'association. Et pour ces dernières, précision des cardinalités et orientations.
c. Classement entre signaux et appels d'opération via la définition et le typage des signaux existant dans l'application.
d. Définition pour chaque classe de la liste des opérations qu'elle supporte, des signaux auxquels elle est sensible et des signaux qu'elle est susceptible d'émettre.. Pour chaque opération: nom, type de la valeur de retour, liste des paramètres avec leur type et mode de passage respectifs.
e. Déclaration de classes comme « modules autonomes » sources potentielles de parallélisme. Les appels aux opérations de ces classes pourront alors être considérés comme étant traités en parallèles. Pour ces classes, déclaration des contraintes de concurrence entre les différents traitements supportés par leurs instances.
f. Description des attributs de chaque classe (uniquement des types simples).

Hormis l'étape initiale de définition des classes de l'application, l'ordre relatif des autres étapes n'est pas fondamental et s'inscrit dans une démarche itérative de raffinement du modèle de l'application. Le passage des activités d'analyse aux activités de conception reste lui-même progressif puisqu'il s'agit toujours de préciser de manière continue un même modèle.
La section suivante présente un patron d'architecture générique basée sur un modèle en couches et visant à structurer le système de manière à orienter le développement du système sous la forme d'un composant réutilisable. Les sections

qui suivent décrivent alors une à une les étapes de construction du modèle structurel d'une application en les appliquant à notre exemple du régulateur d'allure. Tout au long de la présentation de la construction du sous-modèle structurel seront présentées des règles de modélisation, permettant de formaliser l'approche.

3.1 Un patron d'architecture générique

Afin de favoriser la construction d'une application en termes de composant réutilisable, on préconise une architecture logicielle générique mettant l'accent sur la séparation entre le cœur du système et l'interface avec son environnement. Cette architecture en couches contient trois parties distinctes (Figure 8 -). La première est définie par le paquetage nommé *InterfacesFournies*. Cette couche décrit les points d'interaction fournis par le système à son environnement, c.a.d. où et comment l'environnement peut stimuler le système. La couche inférieure représentée par le paquetage nommé *InterfacesRequises* rassemble tous les points d'interaction au travers desquelles le système peut agir sur son environnement, par exemple par le biais d'actionneurs connectés à ces points d'interaction requis. Cette couche se concentre donc sur la description de l'interface nécessaire à un environnement pour que le système puisse fonctionner. La couche du milieu décrit le cœur du système. En plus de ces trois couches, nous avons introduit deux paquetages spécifiques supplémentaires : le paquetage qui servira à regrouper toutes les déclarations de types métiers particuliers à l'application en cours de développement. Par exemple, on pourra y déclarer le type Vitesse pour notre régulateur d'allure. Le second paquetage spécifique introduit (à gauche sur la figure 8) est nommé *DeclarationsSignaux*. Comme son nom l'indique il est là pour collecter l'ensemble des déclarations des types de signaux qui vont être utilisés dans l'application. Pour notre application de régulation de vitesse, on retrouvera ici entre autre la déclaration des types de signaux *MARegulation* et *AERegSyst*. La raison de l'introduction de ces deux paquetages spécifiques est purement organisationnelle, afin de ne pas mélanger les concepts différents. C'est un résultat de l'application du fameux principe de séparation des préoccupations [5].

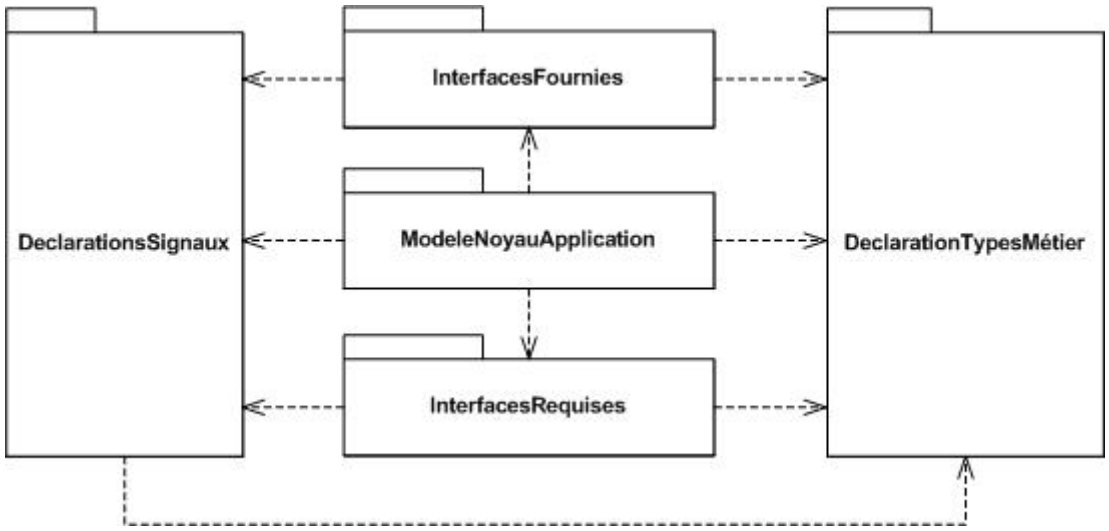

Figure 8 - Un patron d'architecture générique favorisant la réutilisation

Les deux paquetages spécifiques d'interface, *InterfacesFournies* et

InterfacesRequises, ainsi que l'orientation des différentes dépendances entre les paquetages d'une application se justifient par une volonté de structurer et d'identifier clairement les dépendances entre le système développé et son environnement afin, d'une part, de favoriser son intégration dans un contexte existant, et, d'autre part, de manière à accroître la propriété de réutilisation des modèles construits.
En accompagnement de ce patron d'architecture générique, on définit les deux règles de modélisation suivantes :

- Le paquetage contenant les signaux ne contient que des éléments de type signal de UML, et contient tous les signaux de l'application. De plus, ce paquetage ne peut utiliser aucun autre paquetage à part le paquetage de déclaration des types ;
- les paquetages d'interface, *InterfacesFournies* et *InterfacesRequises*, contiennent en phase d'analyse uniquement des éléments de type interface d'UML. En phase ultérieure de conception, chaque interface devra être réalisée par une classe. On préconise de nommer cette classe de réalisation du nom de l'interface préfixée par « R_ ». De plus, le paquetage des interfaces fournies ne pourra être utilisé par aucun paquetage du système.

3.2 Les détails de la structure du système

Dans cette section, nous allons aborder uniquement des problèmes de modélisation orientée objet d'une application avec UML. Il s'agit des deux premières étapes citées précédemment en introduction de la section 3. Les points particuliers liés aux aspects temps-réel de notre application seront traités dans la section suivante.
Une partie des classes de l'application, les classes d'interface du système avec son environnement, peut se déduire du modèle d'analyse préliminaire tel que décrit en section 2.
Ainsi, chaque acteur identifié dans le modèle de cas d'utilisation en phase d'analyse préliminaire peut donner lieu à l'identification d'une classe d'interface correspondante dans le modèle d'analyse détaillée. Si l'acteur avait un rôle actif vis-à-vis du système, c'est-à-dire s'il est à l'origine d'un stimulus reçu par le système, alors l'interface correspondante sera positionnée dans le paquetage des interfaces fournies. En revanche, s'il occupe un rôle passif dans sa relation avec le système, c'est-à-dire qu'il reçoit des stimuli du système, l'interface correspondante est introduite dans le paquetage des interfaces requises. S'il est supposé interagir dans les deux sens, on lui attribuera alors une interface distincte dans chacun des deux paquetages.
Pour le paquetage des interfaces fournies de notre exemple, les concepts-clés sont les notions suivantes :

- *AllumerEteindreRegulateurAllure*, interface la mise sous tension du régulateur d'allure ;
- *MarcheArretRegulation* interface avec le bouton Marche/Arret de la régulation de la vitesse ;
- *PedaleAcceleration*, interface entre la pédale d'accélération et le système ;
- *PedaleFrein*, interface entre la pédale de frein et le système.

Pour le paquetage des interfaces requises, les éléments d'interface déduits du diagramme de cas d'utilisation sont :

- *GroupeMoteurPropulseur*, interface entre le système et le groupe moteur propulseur ;
- *Afficheur*, interface entre le système et l'écran d'affichage du régulateur d'allure ;

- *CompteurVitesse*, interface entre le système et le compteur de vitesse.

Enfin, on identifie depuis le dictionnaire les trois classes principales suivantes pour le cœur du système : *ControleurRegulateur*, *LoiDeRegulation* et *ControleurCompteur*. Les classes *LoiDeRegulation* et *ControleurCompteur* sont des classes qui feront l'objet d'une attention particulière. En effet, il s'agit de concepts très courants, pour la première, dans le domaine de l'automatique, et pour la seconde, dans les applications automobiles. De ce fait, il est intéressant de s'efforcer à en faire des entités réutilisables pour d'autres applications.

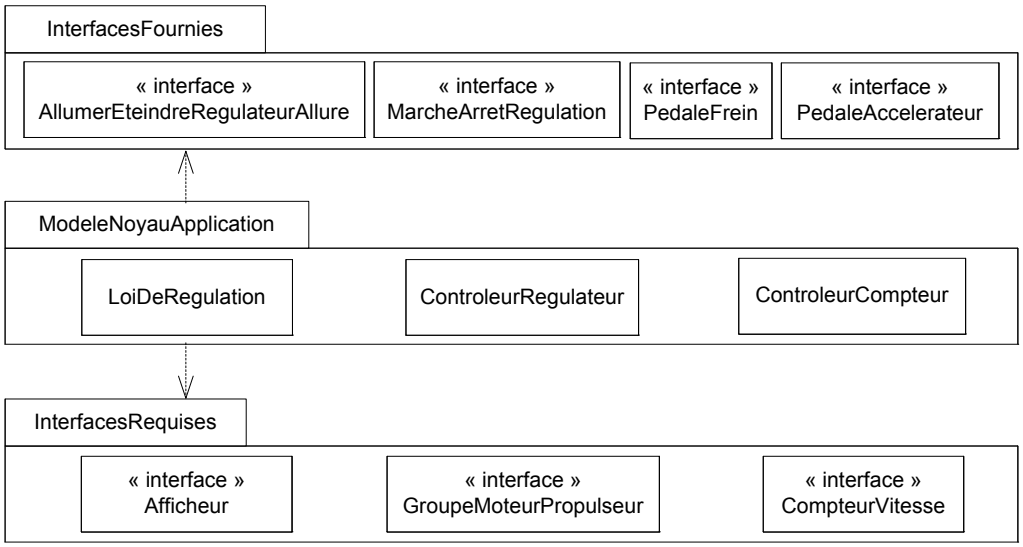

Figure 9 - Identification des classes et éléments d'interfaces du régulateur d'allure

Après avoir identifié les classes d'interface de notre système et quelques premières classes de son cœur, il faut procéder à la connexion de ces classes les unes avec les autres. Pour se faire, on procède en trois temps :

I. Identifier les relations de réalisation entre les interfaces fournies et les classes contenues dans le paquetage qui forme le cœur du système ;
II. identifier les relations (associations ou dépendances) entre les classes internes du système et les interfaces requises ;
III. identifier les relations (associations ou dépendances) entre les classes du cœur du système entre elles.

Pour notre exemple, la figure 10 présente un modèle structurel complété sur ces aspects.

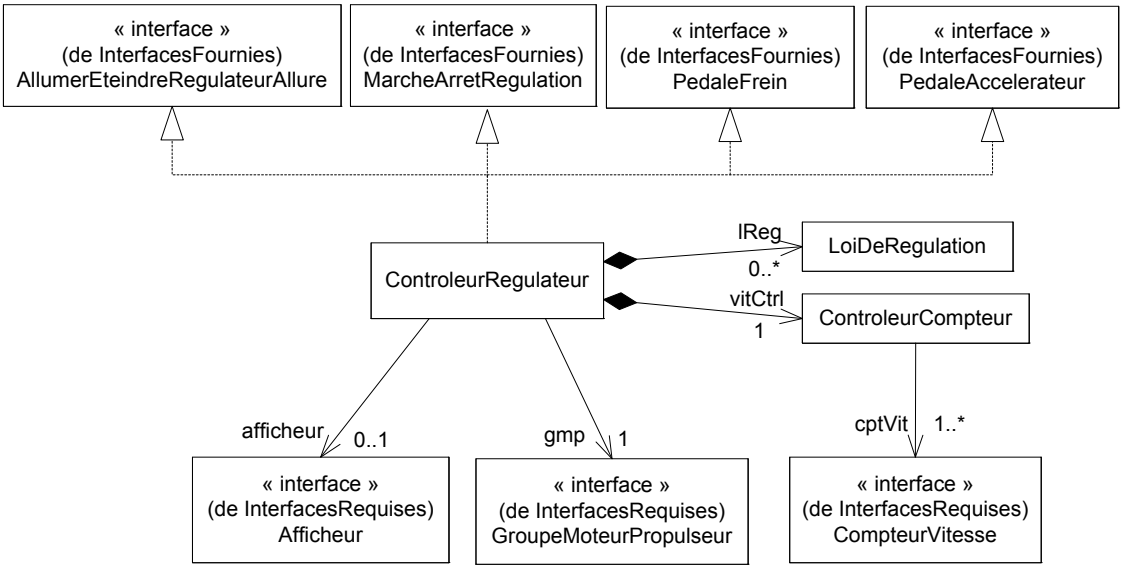

Figure 10 - Identification des relations structurelles

3.3 Modèle d'architecture et aspects temps-réel

3.3.1 La communication par signal

La description des signaux gérés par l'application est un aspect essentiel de la modélisation, en particulier, pour les systèmes temps-réel multitâches. En effet, ils fixent une part importante de la dynamique de l'application et, plus particulièrement, leur spécification concerne sa réactivité.
Un signal est la spécification d'une communication asynchrone entre deux instances d'une application. Une classe déclare qu'elle peut recevoir un type de signal via une "réception" et spécifie son comportement à la réception d'un signal dans une machine à états-transitions. Un signal est un élément généralisable dont la définition est indépendante de la définition des classes les traitant. Ils ont ainsi un impact important sur la modularité des systèmes par la possibilité qu'ils offrent de communication sans lien structurel explicite.
La prise en compte des communications par signal dans une application passe par deux étapes de modélisation au niveau du modèle structurel :

- La liste et le type des signaux présents dans l'application ;
- la spécification des classes concernées par les différents signaux, en particulier, les classes émettrices de signaux et les classes réceptrices de signaux (sensibles à la présence d'un signal dans l'application).

La communication par signal repose sur le principe de non connaissance du destinataire en cas d'envoi d'un signal et de non connaissance de l'émetteur en cas de réception d'un signal. Ainsi, l'émetteur comme le récepteur d'un signal n'ont pas besoin de se connaître a priori (i.e. il n'est pas nécessaire qu'il y ait un lien structurel ou opérationnel de l'émetteur vers le récepteur).
Par ailleurs, la définition *UML* d'un signal spécifie que le comportement d'un objet suite à la réception d'un signal est spécifié dans une machine à états-transitions. La réception d'un signal peut donc, si l'état de l'objet récepteur le permet, déclencher le tirage d'une transition de la machine à états-transitions, ce qui se traduit par l'exécution d'une séquence d'actions associée à cette transition.

Pratiquement, la communication par signal est particulièrement utile dans trois types de situation :

- Lorsqu'on ne veut pas introduire de liens de dépendance structurels entre des objets mais que cependant on désire qu'ils puissent communiquer ;
- lorsqu'un objet O1 doit réagir à un changement d'état particulier d'un objet O2, mais que l'on ne veut pas que l'activité de O1 en soit réduite à scruter l'état de O2 (a fortiori si le changement d'état de O2 attendu est sporadique) ;
- lorsque l'on veut que plusieurs objets (ou un même objet) puissent réagir, peut-être différemment, à un même événement.

Les signaux sont ainsi particulièrement intéressants pour modéliser des parties d'un système dont le caractère réactif est important.
Conformément à la démarche proposée par UML, les signaux seront tous typés par une classe caractérisant leurs propriétés. Une classe spécifiant un signal porte l'annotation « signal » en partie supérieure et peut posséder des attributs définissant les paramètres possibles du signal.
Un signal peut être émis par un objet sans préciser une valeur pour chaque attribut du signal émis. Or, un objet qui reçoit un signal peut être amené à utiliser n'importe lequel des paramètres du signal reçu quel que soit son émetteur. Le récepteur d'un signal n'ayant aucune connaissance de son émetteur, il n'a a fortiori aucune connaissance de la façon avec laquelle le signal est émis. Il est donc nécessaire que chaque signal reçu par un objet ait une valeur pour chacun de ses paramètres et que le récepteur en connaisse leur interprétation et usage. La règle proposée est d'imposer que lors de la création d'un signal, c'est-à-dire aussi lors de son émission, il lui soit passé une liste de paramètres précisant pour chaque attribut du signal la valeur à lui donner.
Un signal ne possède, ni opérations, ni relations d'association autre qu'une possible relation d'héritage avec un autre signal parent. Chaque classe de signal peut ainsi elle-même être placée dans une hiérarchie de signaux plus ou moins génériques et affiner ses caractéristiques par héritage.
Pour l'exemple du régulateur d'allure, deux types de signal peuvent être mis en évidence :

- La demande d'arrêt ou de mise en marche de la régulation qui sera rattachée au signal *MARegulation* ;
- et la demande d'arrêt ou de mise en marche du système de régulation via le signal *AERegSyst*.

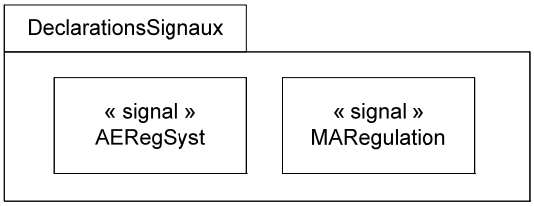

Figure 11 - Déclaration des signaux pour le régulateur d'allure

Une fois que les signaux de l'application ont été déclarés, il est possible de définir quels seront les éléments structurels du modèle qui seront sensibles à la réception d'instances de ces types de signal (figure 12).

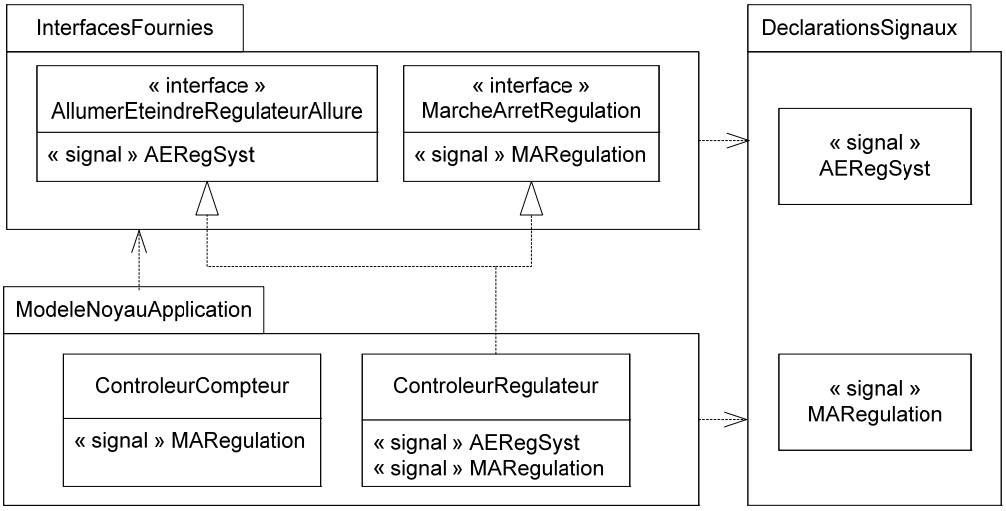

Figure 12 - Déclaration des réceptions aux signaux pour le régulateur d'allure

3.3.2 Les supports au parallélisme

Les systèmes temps-réel, dont le but majeur est de contrôler des systèmes du monde réel/physique, doivent être capables de prendre en compte le parallélisme inhérent à ce même monde réel (ou concurrence) et ainsi proposer des concepts de modélisation appropriés à ce type de situation.
Dans UML, c'est le concept d'objet actif qui est introduit comme support du parallélisme dans un système. La notion d'objet actif est introduite dans une extension de la notion de classe. Lorsqu'une classe est déclarée active, cela signifie qu'elle dispose d'une ressource de traitement propre à son fonctionnement. Dans le cas contraire, les opérations d'une classe passive s'exécutent dans le contexte et sous le contrôle de la ressource d'une autre classe qui doit être active.

Pour choisir les candidats à la qualité d'objet actif, on peut s'appuyer sur les critères de modélisation suivants :

- Les objets déclencheurs de traitements autonomes (tâches) ;
- les objets récepteurs de signaux ;
- les objets récepteurs d'appels d'opération asynchrones ;
- les objets chargés d'acquisition de données depuis l'environnement ;
- les objets chargés d'observer des comportements (sporadiques ou périodiques) de l'environnement.

Dans l'analyse structurelle de notre exemple de régulateur d'allure, deux classes apparaissent maintenant comme des candidats naturels au statut de classe d'objets actifs : le contrôleur du régulateur (*ControleurRegulateur*), et le controleur du compteur de vitesse (*ControleurCompteur*). La classe *ControleurRegulateur* répond à deux des critères précédents. D'une part, elle est déclarée sensible à des signaux (*AERegSyst* et *MARegulation*), et d'autre part, d'après le cahier des charges, elle a besoin de pouvoir déclencher des traitements autonomes (principalement, la tâche de calcul de la consigne) tout en restant attentive au changement d'état de l'environnement contrôlé (détection de freinage, d'accélération,). La classe *ControleurCompteur* répond également au critère numéro 2 (sensibilité aux signaux) mais elle répond également au quatrième critère. En effet, cette classe est en charge d'acquérir à un rythme de 2 Hz la vitesse courante du véhicule et ceci

indépendamment de toute autre action dans le système. On en fait donc également une classe d'objets actifs. Par contre, la classe *LoiDeRegulation* ne répond à aucun de ces critères. Elle restera donc, a priori, un objet passif. D'un point de vue notationnel, une classe active est représenter avec une double barre sur son contour latéral (Figure 13).

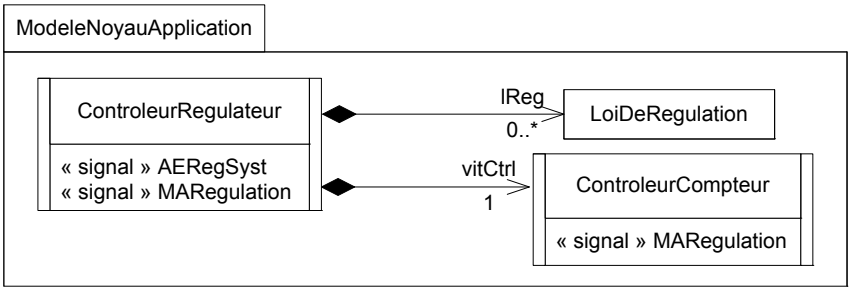

Figure 13 - Identification des sources de parallélisme pour le régulateur d'allure

Il faut souligner à ce stade que UML ne permet pas de préciser pas la manière dont l'exécution sera effectuée, dans quel ordre les messages seront réalisés, si un même objet pourra exécuter simultanément plusieurs traitements ou s'ils seront sérialisés, ni comment les communications ou accès aux attributs seront synchronisés. Pour cela, il faut soit définir sa propre spécialisation des classes actives, comme proposée par la méthode ACCORD/UML à travers la notion de *RealTimeObject* ou en utilisant le tout nouveau profil UML dédié au domaine de l'embarqué : MARTE (www.omgmarte.org).

4 Le comportement d'un système

En termes de modélisation du comportement d'une application, UML propose trois famille de langage : les langages de type automate (aussi appelés machines à états-transitions), les langages de type flux de donnée/contrôle (les diagrammes d'activité de UML) et les langages orientés description des interactions (par ex., les diagrammes de séquence). L'usage de ces derniers ayant déjà été illustré en section 2.4, la suite de cette section sera consacré à donner des exemples d'utilisation des machines à états-transitions et des diagrammes d'activité dans le contexte du régulateur d'allure. La section suivante présentera ainsi une approche générale favorisant une utilisation conjointe de ces deux sous-langages de UML pour spécifier complètement le comportement des entités structurelles de nos modèles UML. Les deux dernières sections donneront des exemples de machine à états-transitions et de diagrammes d'activité issus de la modélisation comportementale du régulateur d'allure.

4.1 Combiner machines à états-transitions et activités

Comme il avait été largement expliqué dans [1], la modélisation du comportement des classes d'une application va se faire en appliquant une fois encore le principe de la séparation des préoccupations afin de favoriser les aspects réutilisabilité et maintenabilité de notre système. Pour se faire, nous allons modéliser le comportement dit de contrôle de la classe dans un diagramme de machine à états-transitions et les aspects algorithmiques dans un digramme d'activité de UML [6].

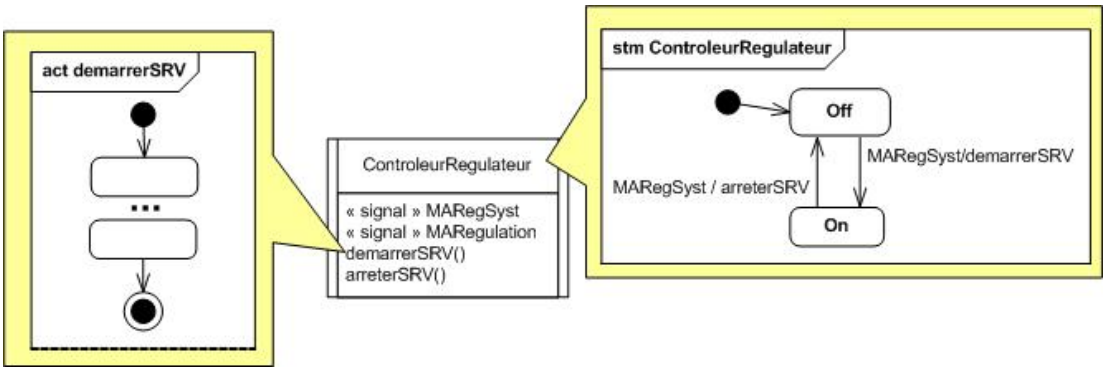

Figure 14 - Une bonne structuration du comportement des classes

Les deux sections qui suivent s'attachent donc à décrire respectivement des exemples de machines à états-transitions et d'activité pour le régulateur d'allure.

4.2 Modéliser le contrôle d'une application

Pour modéliser les aspects contrôle d'une application, nous préconisons l'utilisation des automates de protocole de UML. L'automate de protocole correspond à une utilisation particulière des machines à états-transitions de UML, la spécification des protocoles d'appels d'une classe encore appelée « cycle de vie » de la classe. Les automates de protocole définissent le contexte et l'ordre dans lequel les opérations d'une classe peuvent être appelées. Ils constituent en quelque sorte des modes d'emploi des classes dont ils décrivent le comportement. Le comportement des opérations est alors contenu dans la spécification de méthodes décrivant l'implantation des opérations et non dans des séquences d'actions élémentaires disséminées sur les transitions de la machine à états-transitions. Les transitions d'une machine à états-transitions de protocole sont une restriction du concept de transition standard d'UML. Elles sont nommées transition-protocole et elles sont composées uniquement d'un événement déclencheur dont le type est restreint aux appels d'opération et d'une éventuelle garde. La partie droite de sa spécification est vide. En fait, la séquence d'actions est implicitement spécifiée. Elle est contenue dans la spécification de la méthode-associée à l'opération invoquée par l'événement reçu. Par la suite, la méthode exécutée suite au tirage d'une transition de type transition-protocole sera nommée méthode-associée. Elle désignera la spécification d'implantation de l'opération qui est associée à l'événement déclencheur d'une transition de type transition-protocole.
Le comportement des classes *ControleurCompteur* et *ControleurRegulateur* de notre exemple d'application, le régulateur d'allure, servent d'illustration à cette étape de la modélisation d'une application.
L'analyse du cahier des charges de l'application nous amène à définir pour la classe *ControleurCompteur* les deux états suivants :

- Un état où l'objet est opérationnel et peut effectuer les opérations d'acquisition et de mise à jour de ses attributs. On l'appellera *EnAcquisition* ;
- et un état où l'objet attend une demande de démarrage des opérations d'acquisition. On l'appellera *EnMarche*.

A partir de cette base, il faut ensuite affiner l'analyse en précisant quelles sont les opérations qui peuvent être appelées dans les différents états et quelles transitions leurs appels peuvent déclencher.

On peut d'abord s'intéresser au passage d'un état à l'autre. Conformément aux règles d'UML concernant les automates de protocole, la transition entre deux états ne peut s'effectuer que par l'exécution d'une méthode. Dans notre exemple, la transition entre *EnMarche* et *EnAcquisition* sera réalisée à la suite de l'exécution de la méthode *demarrerAcquisition* de la classe *ControleurCompteur*. La transition inverse sera associée à la méthode *arreterAcquisition*.

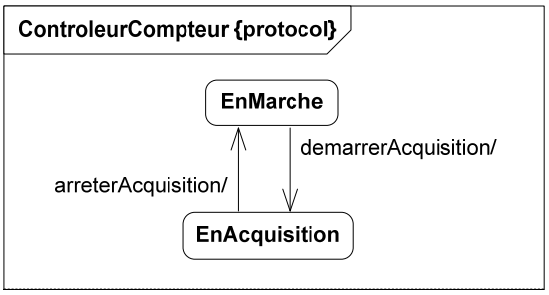

Figure 15 - Machine à états-transitions initiale de la classe *ControleurCompteur*

Un objet possède deux familles d'opérations particulières, les constructeurs et les destructeurs. Les premières sont appelées pour instancier des objets d'une classe et les secondes pour provoquer la destruction des instances cibles.
Les machines à états-transitions attachées à une classe ont pour but de décrire le comportement global des instances de cette classe. Donc, en particulier les étapes de création et de destruction des instances. Pour se faire, UML a introduit deux types d'états particuliers, les états initiaux et les états finaux.

Une étape importante de la modélisation d'une machine à états-transitions est la description de son initialisation. Pour la classe *RegulateurCompteur*, on considère que son état initial est *EnMarche*. On complète ainsi le diagramme précédant en ajoutant un état initial (rond plein noir) et une transition initiale allant de cet état à l'état *EnMarche* (Figure 17).

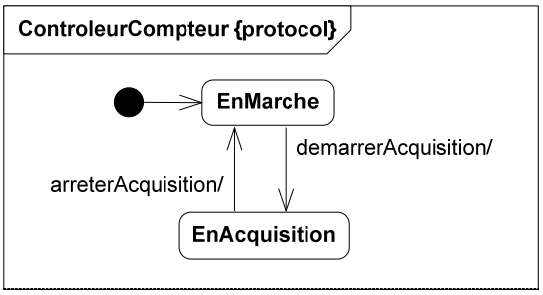

Figure 16 - Initialisation du comportement de la classe *ControleurCompteur*

Dans un second temps, il faut spécifier état par état les opérations qui peuvent être appelées. Ici, dans l'état *EnAcquisition*, les opérations qui peuvent être exécutées de manière cohérente et légitime sont : *acquerirVitesse* et *donnerVitesse* (Figure 18).

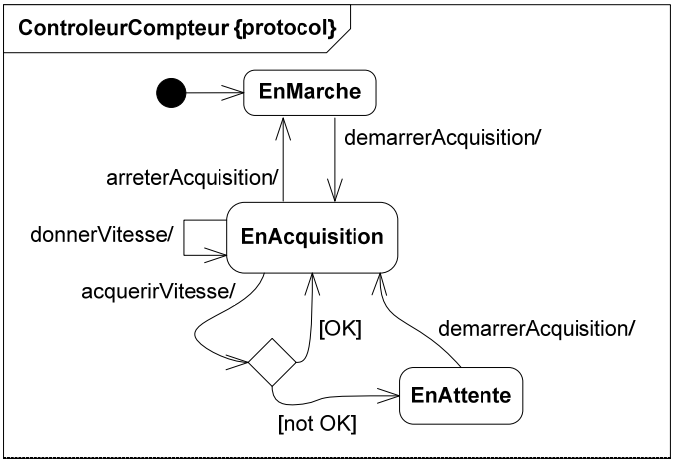

Figure 17 - Description complète du comportement de contrôle de la classe *ControleurCompteur*

Enfin, on notera que la transition déclenchée par la réception d'un l'appel à l'opération *acquerirVitesse* présente deux chemins possibles dans l'automate de protocole : une transition de l'état *EnAcquisition* sur lui-même et une transition de l'état *EnAcquisition* vers l'état *EnAttente*. La transition est une transition composée constituée de trois transitions simples connectées par un pseudo-état de type « choice ». Les gardes des deux transitions sortantes du pseudo état « choice » étant exclusives, le point de choix est déterministe. Le choix sera fait en dynamique lors de l'exécution de la méthode-associée à l'opération *acquerirVitesse* en fonction de la valeur de l'expression booléenne Ok. Si elle est calculée à vrai à la fin de la première transition simple (la transition entrante dans le pseudo état « choice »), alors l'état cible de la transition déclenchée par *acquerirVitesse* sera *EnAcquisition*. En revanche si l'expression booléenne Ok, est évaluée à faux, l'objet se retrouvera dans l'état *EnAttente* à la fin de la transition. Ce second chemin correspond au cas où l'opération d'acquisition de la vitesse détecte un problème de communication avec le compteur de vitesse. Dans ce cas, elle place l'objet dans l'état *EnAttente* (en effectuant tous les traitements éventuellement nécessaires). En effet, la reprise de l'acquisition peut nécessiter des opérations spécifiques de reconfiguration du capteur par exemple. On pourra revenir à l'état *EnAcquisition* en appelant l'opération *demarrerAcquisition* une fois que d'éventuelles actions de correction du problème auront été exécutées.

Suite à cette étape de modélisation, la classe ControleCompteur aura évoluée dans sa spécification en proposant au moins l'ensemble des opérations mentionnées dans l'automate de protocole (Figure 18 -) :

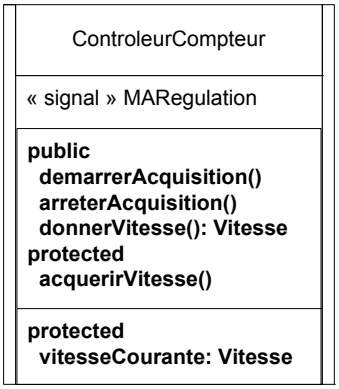

Figure 18 - Description détaillée de la classe *ControleurCompteur* en conformité avec son automate de protocole.

4.3 Modéliser la réactivité d'une application

L"option choisie par UML pour les automates de protocole limite leur usage à la définition des règles d'appel des opérations d'une classe. On note que du coup, un type de communication important est laissé de coté : la communication par signal.
Dans le but de préciser le comportement de l'objet suite à la réception de signaux, la méthode ACCORD/UML propose de compléter les automates de protocole par une vue supplémentaire dédiée à la spécification de la réaction d'un objet à la réception des signaux auxquels il s'est déclaré sensible. Les règles de modélisation proposées consistent à étendre l'automate de protocole en autorisant la définition de transitions déclenchées par un événement du type réception de signal. Dans ce cas, on demande la spécification en partie droite de l'action réalisée par l'identification d'une opération de l'objet à exécuter pour réaliser cette transition.
Pour alléger le modèle, la vue "automate de déclenchement est réduite à ce type de transitions. Dans notre exemple, on obtient alors le modèle suivant (Figure 19 -) :

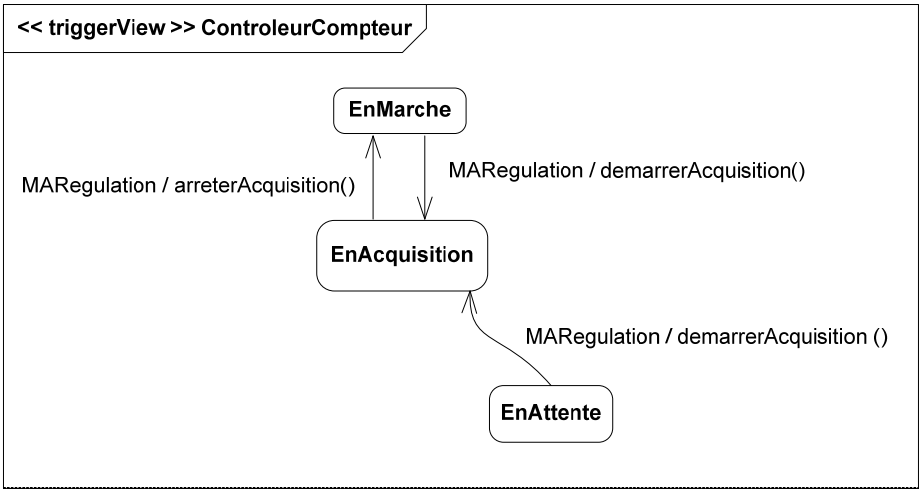

Figure 19 - Description du comportement réactif de la classe *ControleurCompteur*

On notera deux points importants :

- - Afin de garantir la cohérence avec l'automate de protocole, l'opération exécutée doit être une opération valide dans l'état de l'objet conformément

aux spécifications de l'automate de protocole ;
- Un même signal peut déclencher des traitements différents en fonction du contexte de l'objet (c'est d'ailleurs un des intérêts de ce type de communication).

Enfin pour synthétiser la vision automate de comportement, il faut bien avoir en tête que l'ensemble des automates d'une classe ne sont ici que des vues partielles d'un unique automate. Dans notre exemple, on a implicitement définit l'automate suivant (Figure 20 -) :

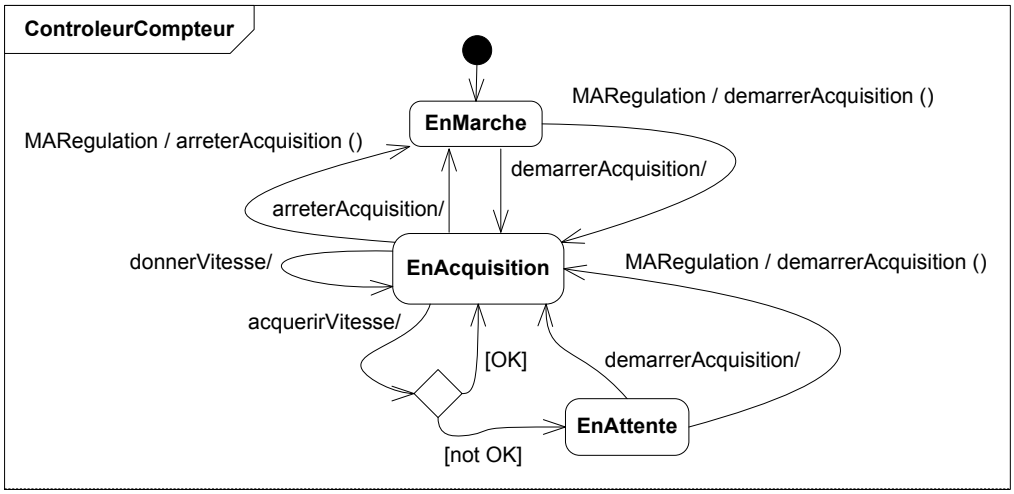

Figure 20 - Description du comportement réactif de la classe *ControleurCompteur*

4.4 Modéliser les aspects algorithmiques

La définition du comportement sur la base d'automate de protocole a l'avantage de simplifier les modèles en focalisant la logique de fonctionnement sur la granularité des classes. En repoussant la description des détails algorithmiques on facilite la maintenance des modèles et augmente leur modularité. Il reste maintenant à préciser les différentes séquences d'actions réalisées par les opérations des objets.

Pour cela on utilisera des diagrammes d'activité afin de mettre en avant l'enchainement des traitements. Prenons, par exemple, le traitement effectué de manière cyclique pour calculer la régulation de la vitesse par le ControleurRegulateur. On nommera ce traitement maintenirVitesse et on le décomposera en plusieurs étapes :
- obtenir la valeur courante de la vitesse
- obtenir la valeur du couple à transmettre au bloc moteur
- envoyer cette valeur au bloc moteur

On peut également, intégrer à ce traitement une action de mise à jour de l'afficheur d'état du système par l'envoi de la vitesse courante prise en compte.
On obtient alors le diagramme suivant :

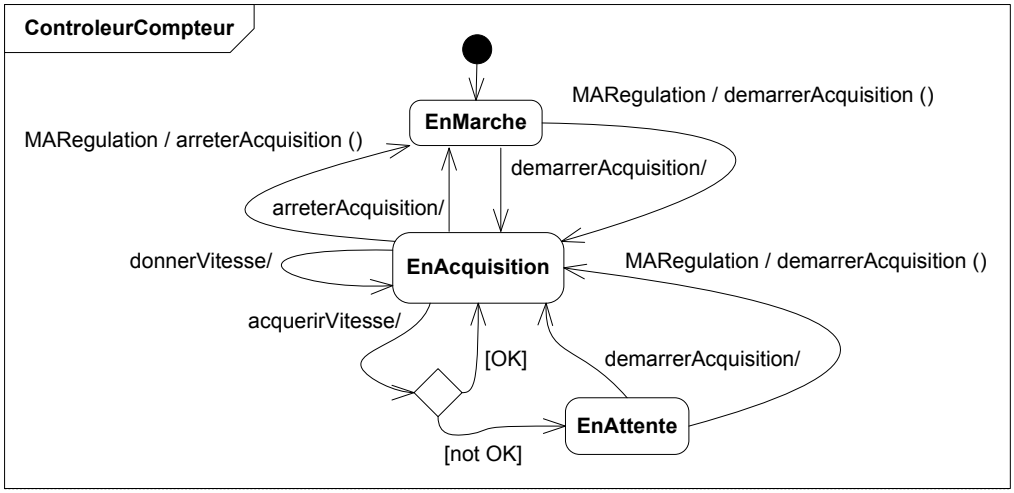

Figure 21 - Description du comportement réactif de la classe *ControleurCompteur*

$$\delta C = \frac{\arctan(k \times (Vcible - Vvéhicule))}{2}$$

Bibliographie

[1] F. Terrier et S. Gérard, UML pour le temps-réel : le langage et les méthodes, Techniques de l'ingénieur, dossier S8070, Juin 2005.

[2] P. Desfray, Modélisation par objets : la fin de la programmation, Masson Ed., 1996.

[3] I. Jacobson, Formalizing use-case modeling, in JOOP, 1995.

[4] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley Pub., Object Technology Series, 1999

[5] W. Hürsch and C. Lopes, Separation of Concerns, College of Computer Science, Northeastern University, 1995.

[6] C. Mraidha, S. Gérard, F. Terrie et J. Benzakki, A Two-Aspect Approach for a Clearer Behavior Model, dans les actes du 6ieme IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'2003), ISBN 0-7695-1928-8, 2003.