Modélisation UML d'un régulateur de vitesse automobile

Auteurs: Luka Le Roux, Jérôme Delatour, Philippe Dhaussy

Résumé:

Dans le développement des systèmes embarqués temps réel, le couplage des approches semi-formelles et formelles reste complexe, tant par la diversité des langages utilisables que par les approches méthodologiques possibles. En outre, il est encore reproché aux approches formelles d'être difficiles d'utilisation par des industriels.

Afin d'illustrer les progrès de ces approches et leur relative facilité d'emplois par des industriels, un même cas d'étude sera traité par différentes approches formelles. Cet article, premier d'une série, introduit le cas d'étude d'un système de régulation de vitesse automobile. Cette présentation s'appuie sur une modélisation semi-formelle. Cette modélisation, de type SysML/UML nous semble assez représentative des pratiques industrielles du domaine. L'objectif est ainsi de fournir un premier modèle commun en entrée des différentes approches formelles qui seront présentées dans les articles suivants.

Mots-clés: Modélisation, UML, régulateur de vitesse, cas d'étude

1 Introduction

La complexité des systèmes embarqués temps réel s'accroît de manière exponentielle avec l'apparition de nouvelles technologies. De nombreux efforts, tant techniques que méthodologiques, sont investis pour augmenter la confiance dans ces systèmes. La modélisation est une étape clé du processus de développement.

Cette modélisation fait souvent appel à l'utilisation conjointe de notations semi-formelles et formelles. En effet, les approches dites semi-formelles simplifient notamment la capture des besoins, le dialogue entre les différents spécialistes impliqués dans le développement ainsi que le passage des fossés sémantiques entre les phases d'analyse de besoin, de spécification et de conception. L'utilisation de modèles formels et d'outils de vérifications associés permet de s'assurer, tout au long du développement, la cohérence et la faisabilité du futur système.

Toutefois l'utilisation conjointe des approches formelles et semi-formelles reste complexe. En outre, il existe de très nombreuses approches, tant par les notations semi-formelles utilisées (SA-RT, HOOD, UML, AADL, SysML, …) que par les langages formels employés (Promela, TLA, Fiacre, Altarica, UPPAAL, SMV, …).

Nous présentons section 2 les spécifications d'un régulateur de vitesse (*CCS – Cruise Control System*) dans le but d'illustrer certaines contraintes sur la modélisation introduites par l'utilisation de techniques pour la vérification de propriétés. Nous présentons les fonctionnalités du système, son architecture et des exigences à vérifier pour valider le modèle.

La section 3 explicite les choix préliminaires à la modélisation, en particulier les implications de la méthodologie mise en œuvre, avant de présenter différentes vues du modèle UML obtenu.

2 Spécifications du Cruise-Control System (CCS)

Cette section présente les spécifications du système CCS embarqué sur une voiture en vue de la régulation automatique de la vitesse du véhicule. Ce système est considéré critique et doit faire l'objet d'une validation formelle. Dans une première phase, nous présentons les fonctionnalités du système. Ensuite, nous décrivons son environnement et des exigences qui seront soumises aux vérifications.

2.1 Fonctionnalités principales

La fonction principale du CCS est de contrôler automatiquement la vitesse du véhicule. Un scénario nominal d'utilisation est présenté figure 2. Après la mise en marche du système (message **PowerOn**), le conducteur doit fournir une vitesse de consigne dite *vitesse de croisière (*message ***Set***). Le système peut alors être engagé (message **Resume**), c'est à dire qu'il prend en charge le maintien de cette consigne de vitesse. A tout moment, le conducteur peut diminuer ou augmenter la vitesse de croisière.

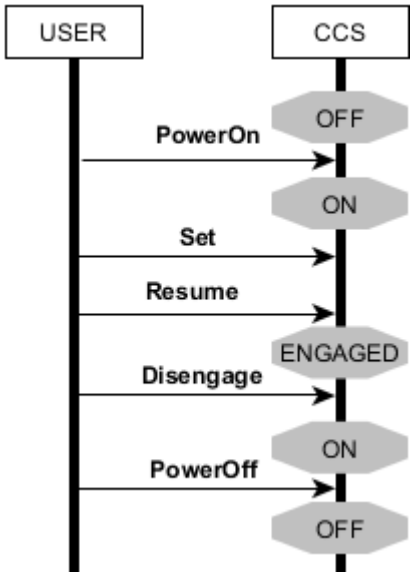

Figure 1 : *Scénario nominal d'utilisation.*

Le système doit garantir la sécurité pour une conduite sûre. En particulier il doit se désengager automatiquement lors des conditions suivantes : (i) une pression est appliquée sur la pédale de frein, (ii) le conducteur débraye ou (iii) la vitesse courante du véhicule s dépasse les bornes autorisées ($40 < s < 180 km/h$). Le système doit aussi se mettre en pause lors d'une pression sur l'accélérateur et ce jusqu'à ce qu'il soit relâché.

2.2 Environnement du système

Figure 2 : *Cas d'utilisation simplifié du CCS.*

Le CCS interagit avec 2 entités de l'environnement (figure 3).

D'un coté le conducteur (*User*) possède les capacités suivantes:

- Changement de la vitesse courante du véhicule;
- Pression sur la pédale de frein, l'embrayage, l'accélérateur;
- Interaction avec le système via l'un des boutons de contrôle.

De l'autre, la voiture (*Car*) est responsable de la mise à jour de la vitesse courante. Elle informera régulièrement le CCS de la vitesse de la voiture.

2.3 Exemples d'exigences

Nous présentons ici quelques exigences qui doivent être vérifiées sur le modèle du CCS.

- ***Req1*** : ***Après*** *la détection d'un événement induisant un désengagement et* ***tant que*** *le système n'est pas réengagé, le CCS* ***ne doit pas*** *tenter d'ajuster la vitesse du véhicule.*
- ***Req2*** : *La vitesse de croisière* ***ne doit pas*** *être inférieure à 40km/h ou supérieure à 180km/h.*
- ***Req3*** : ***Tant que*** *le système est engagé, la vitesse cible* ***doit être*** *considérée comme définie.*

3 Modélisation du cas d'étude

Cette section présente la modélisation du cas d'étude que nous venons de présenter. Le langage utilisé correspond à un sous-ensemble d'UML qui nous semble représentatif des pratiques habituelles. Les aspects structurels sont représentés soit par un diagramme de communication, soit par un diagramme composite ou par un diagramme de classe. Le comportement des objets identifiés comme actifs est modélisé par des diagrammes état-transition. La notation SysML pourrait tout aussi bien être utilisée (à des variantes prés, par exemple un IBD serait utilisé pour la représentation des aspects structurels).

3.1 Architecture candidate

Le CCS est composé de quatre entités : *control panel*, *actuation*, *health monitoring* et *system center* (figure 3). Cette dernière est l'entité centrale de l'architecture. C'est elle qui assure le contrôle du système.

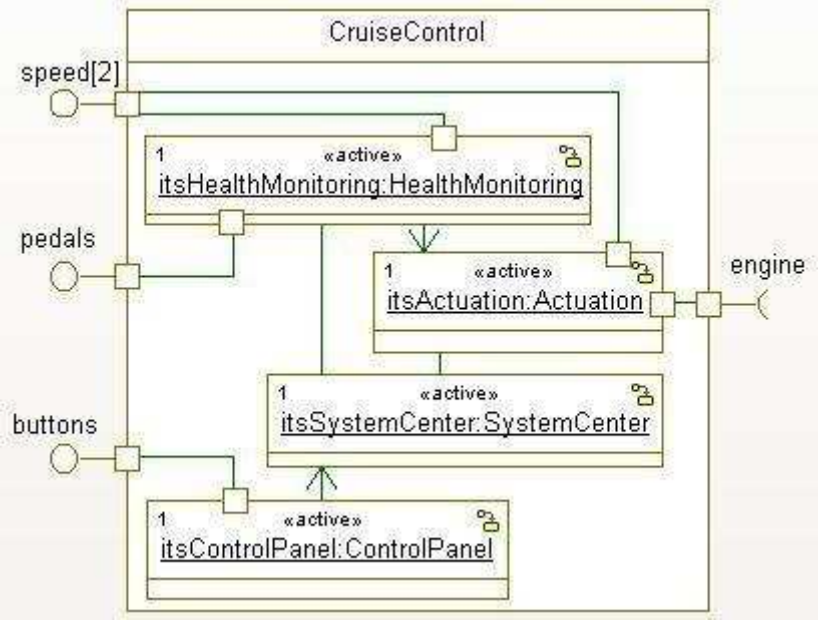

***Figure 3** : Architecture du système CCS.*

Le composant ***control panel*** capture les consignes du conducteur par le biais des entrées suivantes :

- **PowerOn**: mise en marche du système;
- **PowerOff**: extinction du système;
- **Set**: définie la vitesse courante du véhicule comme *vitesse cible* ;
- **Resume**: engage le système ;
- **Disengage**: désengage le système;
- **Inc**: incrémente la *vitesse cible* de 5*km/h*;
- **Dec**: décrémente la *vitesse cible* de 5*km/h*.

Le composant ***actuation*** permet au système d'interagir avec le moteur du véhicule via une interface de contrôle et donc d'en ajuster la vitesse. Elle capture la vitesse courante comme *vitesse de croisière* et, lorsque le système est engagé, elle est responsable d'ajuster la vitesse du véhicule.

Le composant ***health monitoring*** surveille le comportement du système, et relaye au contrôleur les événements suivants :

- Pression sur la pédale de frein, l'embrayage, l'accélérateur;
- Dépassement des bornes de sécurité de la vitesse courante.

L'appui sur l'accélérateur induit une pause du système tant que la pédale est enfoncée. Les autres événements induisent un désengagement du système.

Le composant ***system center*** est le cœur du CCS. Il est chargé de traiter les événements détectés par le *control panel* et l'*health monitoring*. Pour ce faire il doit être capable d'impacter le comportement des autres entités.

3.2 Vue haut niveau du système

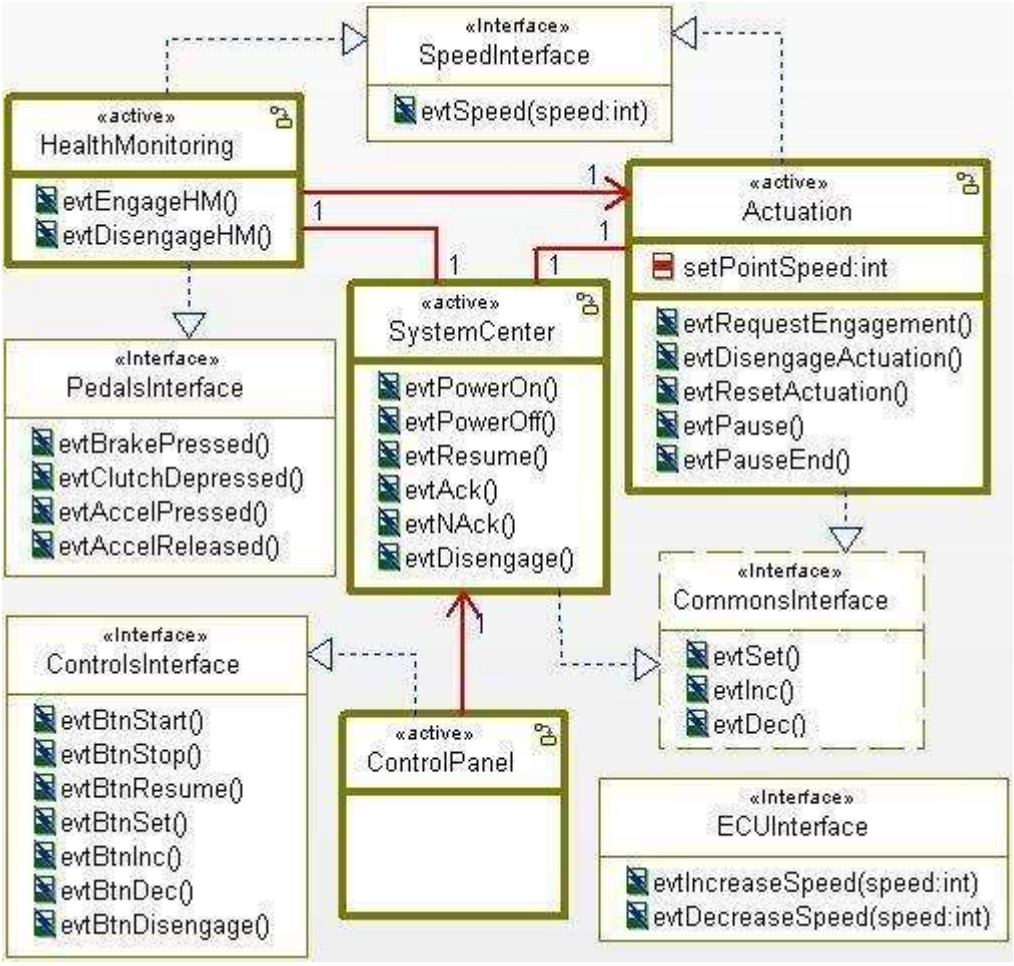

Figure 4 : *Diagramme de classe du CCS*

Le diagramme Figure 4 présente les classes qui composent notre système. On y retrouve les différents composants du système. Apparaissent aussi les différentes relations entre eux et les événements qui peuvent être échangés.

Les instances des classes définies ici sont considérées comme des objets actifs (représenté par le stéréotype *<<active>>*). Ces objets actifs possèdent leur propre fil d'exécution et leur propre file d'événements (principe de modélisation dite par acteur). Les objets actifs communiquent entre eux par envois d'événements asynchrones.

Les stimuli et réponses de l'environnement correspondent aux événements des interfaces *PedalsInterface*, *ControlsInterface*, *SpeedInterface* et *ECUInterface*.

3.3 Les machines à états

Cette section présente le comportement des composants dit actifs. Ces comportements sont décrits par des diagrammes état-transitions UML.

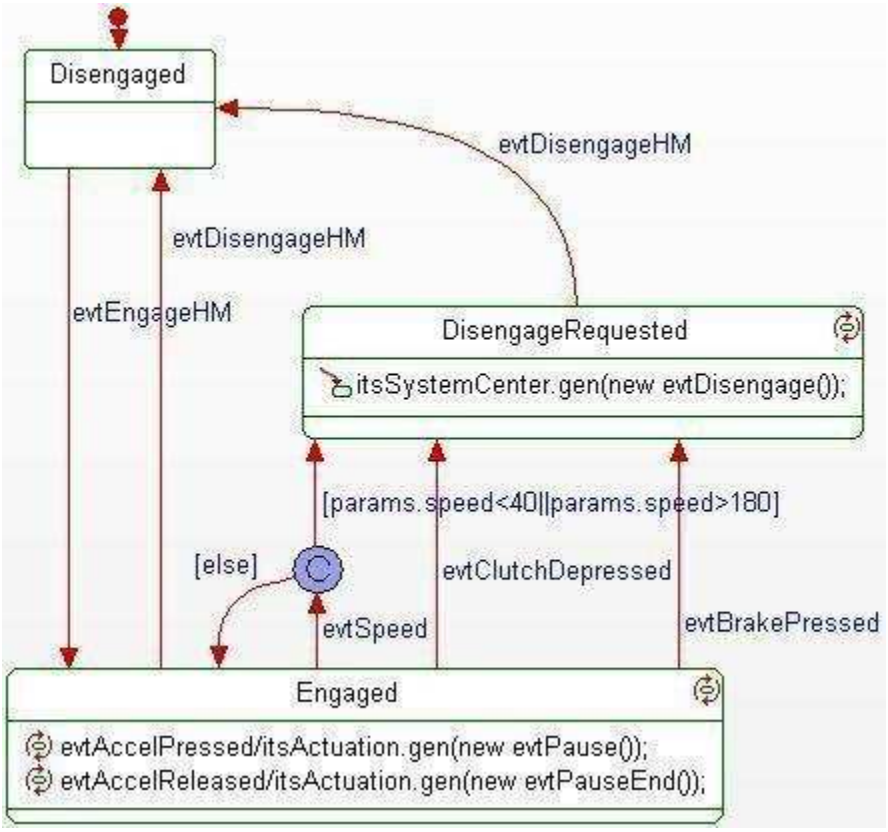

Figure 5 : *Machine à états de l'entité health monitoring*

La machine à états de l'entité *health monitoring* (figure 5) est composée de trois états: *Disengaged*, *Engaged* et *DisengageRequested*. Le passage du premier au second se fait sur réception d'un événement envoyé par le *system center*. Le passage dans le dernier dénote la détection d'un événement critique qui doit induire un désengagement du système et de ce fait provoque l'envoi d'une notification au *system center*.

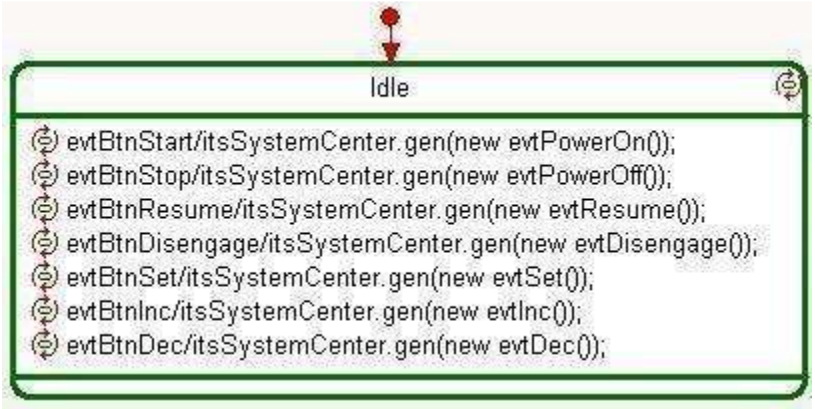

Figure 6 *: Machine à états de l'entité control panel*

L'entité *control panel* se contente de relayer les pressions sur les boutons en événements compréhensibles par l'entité *system center*. Cette entité a été simplifiée et ne prend pas en charge, en particulier, la gestion de l'affichage. Toutefois il a été décidé de la représenter par un objet actif en respect des fonctionnalités manquantes. Par conséquent son comportement, représenté par la machine à états donné en figure 6, se résume à un seul état ayant des transitions internes. Par exemple, la première de ces

transitions internes envoie l'événement *evtPowerOn* au *system center* sur réception de *evtBtnStart*.

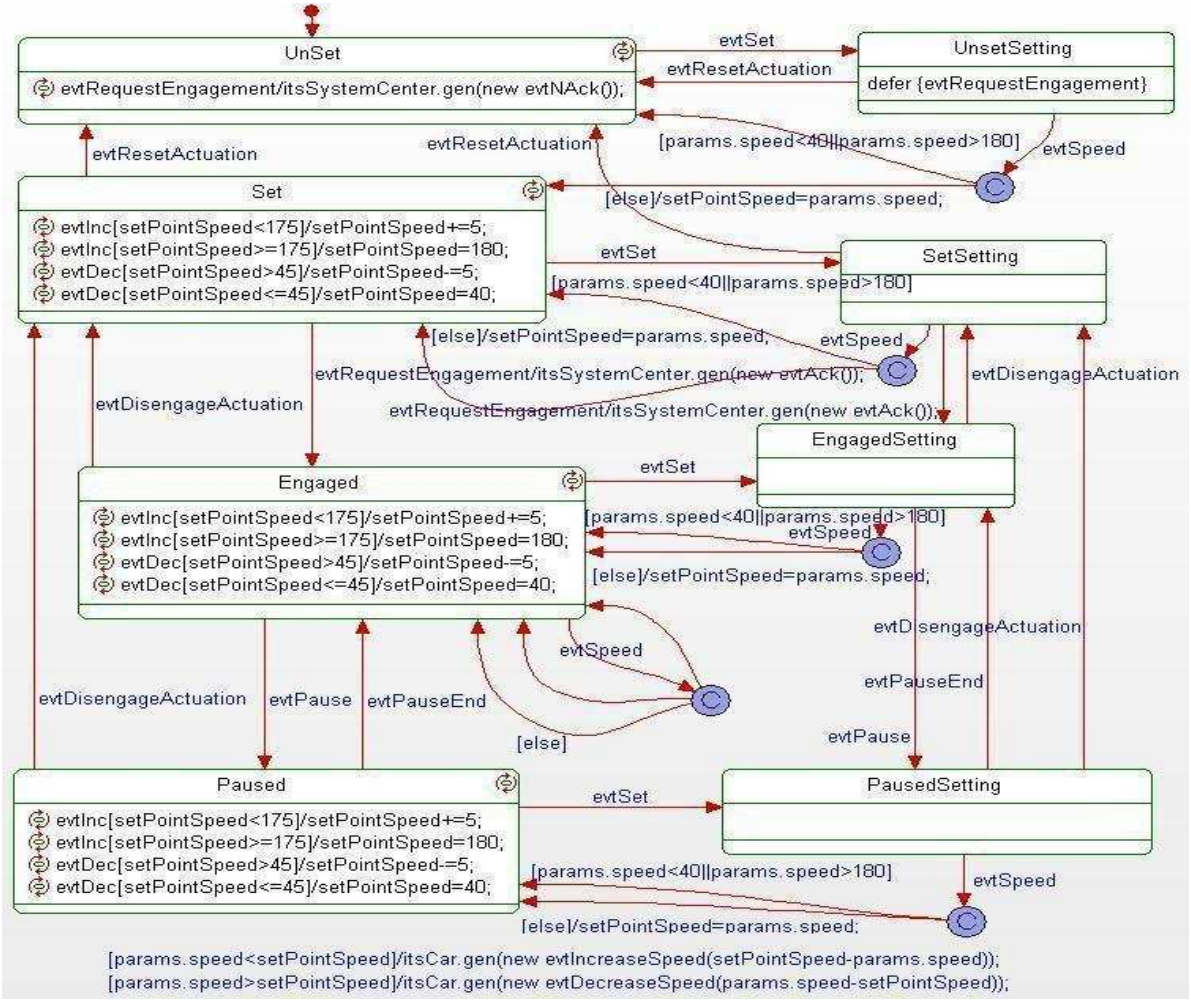

***Figure 7** : Machine à états de l'entité actuation*

L'apparente complexité de l'entité *actuation* (figure 7) est due au fait que sa machine à états est composée de paires *X* (partie gauche) et *XSetting* (partie droite), dénotant respectivement le mode nominal et la capture de vitesse.

L'état initial *UnSet* dénote le fait que la *vitesse de croisière* est indéfinie. L'automate passe dans l'état *Set* une fois celle-ci capturée. Il est alors possible d'engager le système (état *Engaged*) puis de le mettre en pause (état *Paused*).

Les autres états correspondent aux attentes respectives lors de la capture de la *vitesse de croisière*. Après réception d'une requête *evtSet* venant du *system center*, le composant bascule dans l'état *Setting* correspondant et attend la prochaine réception de la vitesse courante (*evtSpeed(int)*). Suivant la valeur reçue l'état de retour (après la capture) peut être différent, par exemple *UnSet -> UnSetSetting -> Set*.

Si l'entité n'est pas en attente de capture (etats *XSetting*) et que la *vitesse de croisière* est définie il est possible d'incrémenter (*evtInc*) ou de décrémenter (*evtDec*) la *vitesse de croisière*. Le traitement se fait dans les transitions internes aux états *Set, Engaged* et *Paused*.

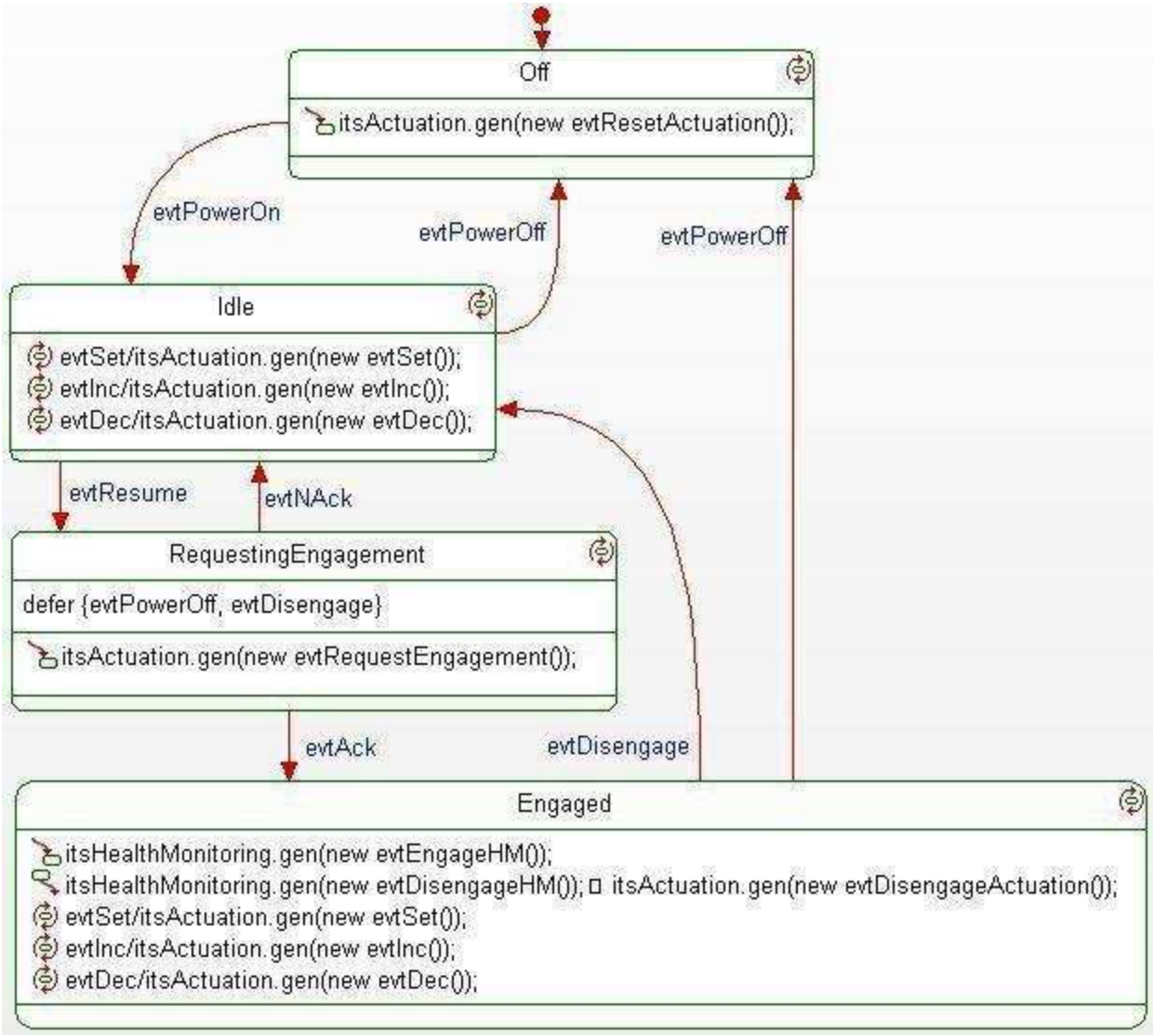

***Figure 8**: Machine à états de l'entité system center*

L'entité *system center* (figure 8) doit s'assurer que tous les composants se comportent de façon cohérente. Ses états correspondent au fonctionnement global du système.

4 Conclusion

Nous avons présenté les spécifications d'un régulateur de vitesse (CCS). Nous avons ensuite décrit une modélisation UML. Cette modélisation semi-formelle du CCS a été volontairement complexifiée sur certains aspects (complexité de certains comportements) et simplifiée sur d'autres. Elle comporte des erreurs. Le bon fonctionnement du système n'est donc pas garanti. Dans [2], nous mettons en œuvre une technique de vérification formelle [1] qui est censé permettre de valider le modèle. Dans un processus itératif, la technique doit de permettre d'aboutir à un modèle correct respectant les exigences données.

5 Bibliographie

[1] L. Le Roux, P. Dhaussy et F. Boniol : Vérification formelle de propriétés basée sur une réduction de l'espace d'exploration de modèles; Revue Génie Logiciel, n° 107, décembre 2013.

[2] P. Dhaussy, L. Le Roux, et C. Teodorov ; Vérification formelle de propriétés : Application de l'outil OBP au cas d'étude CCS; Revue Génie Logiciel, n° 109, juin 2014.

Luka Leroux est ingénieur de recherche. Après un Master recherche à l'université de Rennes I, il a rejoint en 2011 l'équipe du pôle STIC de l'ENSTA-Bretagne. Ses recherches sont orientées dans le domaine de modélisation des logiciels embarqués et les techniques de vérifications formelles d'exigences.

Jérôme Delatour est enseignant-chercheur à l'ESEO (école d'ingénieur, http://www.eseo.fr/). Il est le responsable de l'équipe de recherche TRAME (TRAnsformations de Modèles pour l'Embarqué, http://trame.eseo.fr/). Ses activités de recherche sont axées sur la modélisation, ainsi que sur le développement d'applications temps réels à l'aide de l'Ingénierie Dirigée par les Modèles. Dans ce contexte, il a notamment participé à la normalisation du profil UML MARTE (*http://omgmarte.org/*).

Philippe Dhaussy est directeur du pôle STIC de l'ENSTA-Bretagne et enseignant-chercheur HDR. Son expertise et ses recherches sont orientées vers l'Ingénierie Dirigées par les Modèles et les techniques de validation formelle pour le développement de logiciels embarqués et temps réel. Diplômé ingénieur (1978) de l'Institut Supérieur d'Électronique du Nord (ISEN), il a obtenu le titre de docteur (1994) à Telecom Bretagne. Après un parcours industriel (1980-1991), il a rejoint l'ENSTA-Bretagne et a contribué à la création du groupe de recherche en modélisation de systèmes et logiciels embarqués, intégré au laboratoire Lab-STICC UMR CNRS 6285