Démarche de conception sure de la supervision de la fonction conduite autonome

Romain Cuer[1,2], Laurent Pietrac[1], Eric Niel[1], Saidou Diallo[2], Nicoleta Minoiu-Enache[2], Christophe Dang-Van-Nhan[2]
[1] INSA Lyon, 20 avenue Albert Einstein, Villeurbanne, France
[2] Renault, 2 avenue du Golf, Guyancourt, France

Résumé

Le véhicule autonome est conçu pour se conduire sans aucune intervention du conducteur. Ce véhicule comprend une nouvelle fonction, appelée fonction AD (pour *Autonomous Driving*), en charge de la conduite lorsque ceci est autorisé. Cette fonction peut se trouver dans différents états (*off*, *disponible* par exemple) qui sont gérés par une sous-fonction nommée supervision. Le principal objet de cette étude est de garantir que la supervision d'une fonction, réalisée par un système automobile embarqué critique du point de vue de la sûreté, respecte les exigences fonctionnelles et de sécurité qui lui sont allouées. Puisque deux aspects différents sont étudiés, la mise en cohérence de l'ensemble de ces exigences, dès le début de la conception, est un défi majeur. Dans cet article, une méthode est proposée pour contribuer à résoudre ce problème. Les exigences traitées sont progressivement consolidées en utilisant des modèles comportementaux formels. Les potentielles ambiguïtés, incohérences ou incomplétudes sont ainsi graduellement traitées. De plus, l'application de la méthode à la conception de la fonction AD souligne son efficacité et son intérêt dans un contexte industriel.

1 Introduction

Ces dernières années, des systèmes d'aide à la conduite ont été progressivement introduits dans les véhicules. Ces systèmes embarqués électroniques sont critiques du point de vue de la sûreté (Kaiser et al., 2010; Weissnegger et al., 2015) car certaines de leurs défaillances peuvent avoir des conséquences graves pour les personnes et/ou l'environnement. Néanmoins, il est toujours possible de compter, en dernier recours, sur le conducteur afin de garantir la sûreté du véhicule. Ceci se traduit par le critère de contrôlabilité fourni par la norme ISO 26262 (N. Becker, 2011) : certains événements redoutés sont déterminés comme étant contrôlables par le conducteur. En conséquence, les défaillances potentielles des systèmes d'aide à la conduite qui en sont à l'origine peuvent, dans une certaine mesure, être acceptables. Ce n'est plus le cas pour le véhicule autonome. Le défi posé par sa conception est d'une autre ampleur étant donné l'absence de conducteur, donc de contrôlabilité (Koopman and Wagner,

2017). De plus, garantir la sûreté du véhicule autonome uniquement par des tests de validation semble presque impossible (Kalra and Paddock, 2016). Le véhicule autonome sera précisément équipé d'un système embarqué (appelé système AD pour *Autonomous Driving*) chargé d'assurer la fonction de conduite autonome. Au vu des éléments présentés précédemment, il est primordial de garantir la sûreté de la conception du système AD. Néanmoins, l'intégration du processus de Sûreté de Fonctionnement (SdF) à celui de l'Ingénierie Systèmes (IS) demeure problématique étant donné les différences en termes d'objectifs, de planning, de contraintes et d'équipes de travail (Mauborgne et al., 2016; Taofifenua, 2012).

Ce sujet est aussi central pour les autres types de moyens de transport. Des méthodes spécifiques, comme la *Safety Driven Design Methodology* (Owens et al., 2008), outillée par un environnement logiciel tel que SCADE (Märtin et al., 2013) sont mises en œuvre dans le domaine aéronautique. Concernant le secteur ferroviaire, la méthode B (Behm et al., 1999; Boulanger, 2014) a prouvé son efficacité dans un contexte industriel. Toutefois, les contraintes propres aux applications automobiles, telles que le temps très court de mise sur le marché, les conditions extrêmement variables (pays, capacités des conducteurs, conditions climatiques, réglementations…), les contraintes fortes de l'existant, la réduction des coûts, l'espace limité ou encore l'organisation des équipes de travail (Broy, 2006; Chalé et al., 2011; Maurer and Winner, 2013), rendent l'adaptation de ces méthodes difficile. Des moyens, démarches et outils adaptés doivent donc être développés spécifiquement pour le secteur automobile.

Cet article est structuré de la manière suivante. La seconde section présente les études connexes et donne le positionnement des travaux réalisés. La méthode proposée, permettant de concevoir de manière sûre une fonction exécutée par un système embarqué automobile, est exposée dans la troisième section. Plus précisément, la démarche déployée se focalise sur le comportement de la fonction conçue: l'objectif principal est d'assurer qu'elle reste toujours dans un état sûr, en toutes circonstances. Cette section contient les principales contributions de ce papier : l'approche en elle-même, permettant de mettre en cohérence les points de vue fonctionnel et de SdF en phase amont de conception; la consolidation des exigences traitées par la mise en exergue d'ambiguïtés, incomplétudes, incohérences ; et l'exploitation de certaines ambiguïtés dans le but de faire émerger des conceptions alternatives. La quatrième section illustre l'intérêt pratique de la démarche et les résultats obtenus dans le cadre d'une application industrielle : la conception de la fonction AD. Enfin, les limites des travaux présentés ainsi que les principales perspectives sont exposées dans la dernière partie.

2 Etat de l'art

La prise en compte du résultat des analyses de SdF dès le début du cycle de conception est un problème bien connu pour la mise au point de systèmes embarqués critiques du point de vue de la sûreté (Leveson, 2002), et notamment dans le domaine automobile (Chen et al., 2011; Kaiser et al., 2010; Kang et al., 2013; Weissnegger et al., 2015). Ainsi, de nombreux travaux abordent cette question.

La modélisation de systèmes embarqués critiques afin d'intégrer les activités de l'Ingénierie Systèmes aux analyses de SdF associées dans un modèle unique formel, ou semi-formel, a fait l'objet de multiples publications (Chen et al., 2011; Cressent et al., 2011; David et al., 2010; Gudemann and Ortmeier, 2010; Jean-François Pétin et al., 2010; Kang et al., 2013; Liu and Zhu, 2011). Cependant, dans le cadre de cet article, nous nous focalisons sur les méthodes de vérification de ces modèles. Les trois principales méthodes de vérification mises en œuvre (Evrot, 2008), que ce soit dans l'industrie ou du point de vue de la recherche sont :

- *La simulation* : cette technique répandue (Cressent et al., 2011; Nouacer et al., 2016) repose sur l'exécution symbolique des modèles et la réalisation de tests de conformité correspondant aux besoins exprimés par l'utilisateur. L'exécution symbolique nécessite une sémantique

opérationnelle définissant de manière déterministe le comportement d'un modèle en réaction à des stimuli d'entrée. La **principale limite** est l'exhaustivité des scénarios testés. La simulation ne donne donc qu'une présomption de bon comportement et est fortement corrélée à l'expérience et l'expertise des personnes qui la pratiquent ;

- *Le theorem proving* : la vérification est vue comme un théorème à prouver à partir d'un ensemble d'axiomes. Le programme et les propriétés à vérifier doivent être transformés en objets mathématiques. Des conclusions sont inférées directement à partir d'une description d'événements ou d'opérations permettant de faire évoluer le système (Roussel and Denis, 2002). Les **principales limites** sont que l'automatisation complète est rarement possible car l'utilisateur doit souvent interagir avec l'assistant de preuve et que la préparation de la preuve requiert la spécification d'éléments sortant du cadre direct de l'expression des besoins du cahier des charges ;

- *Le Model Checking*: technique automatisée qui, étant donné un modèle d'état fini d'un système et une propriété formelle, vérifie systématiquement si cette propriété est valable pour ce modèle (Baier and Katoen, 2008). Cette méthode se décompose en trois phases : la *modélisation* du système étudié et la formalisation des propriétés à vérifier, l'*exécution* de l'algorithme de *model checking* et l'*analyse* des propriétés (satisfaites, non satisfaites ou mémoire saturée). Différents outils existent pour mettre en œuvre cette méthode :UPPAAL[*] (Apvrille and Becoulet, 2012; Jean-François Pétin et al., 2010; Liu and Zhu, 2011), UPPAAL-Port[†] (Kang et al., 2013) ou encore NuSMV[‡] (Sharvia and Papadopoulos, 2015).

  Les **principales limites** sont :
  1) Le modèle du système et non le système lui-même est vérifié. Cette méthode n'est donc que complémentaire avec les tests sur le système réel et ne s'y substitue pas ;
  2) Le problème de l'explosion combinatoire ;
  3) La nécessité d'une expertise pour trouver les abstractions appropriées permettant d'obtenir un modèle du système correct et d'exprimer les propriétés correctement dans une logique formelle.

La technique du *model checking* est retenue pour la méthodologie déployée. En effet, si la simulation est déjà une pratique largement reconnue et éprouvée dans le domaine automobile, elle reste subordonnée à une démarche essai-erreur. En outre, la sûreté du véhicule autonome ne peut être raisonnablement démontrer uniquement par des tests de validation et des simulations (Kalra and Paddock, 2016). Par ailleurs, les limites du *theorem proving* mentionnées empêchent, pour le moment, son application dans un contexte opérationnel automobile. La démarche proposée contribue à réduire la 3^{e} limitation du *model checking* indiquée plus haut.

Ainsi cet article vise plus particulièrement à analyser le problème de l'expressivité des propriétés à vérifier. Des travaux abordent l'analyse des exigences, leur décomposition et leur formalisation (Bitsch, 2000; Evrot, 2008; Nebut and Fleurey, 2005). Néanmoins les hypothèses nécessaires à la formalisation ne sont pas explicitées et les impacts de ces hypothèses analysés. (Ghazel et al., 2015) propose bien un processus de formalisation progressive d'exigences initialement exprimées en langage naturel. Cependant l'accent est porté sur le raffinement des exigences (passage d'une exigence brute à une exigence dite *formalisable*, voir section 3.2) mais la formalisation en elle-même est moins détaillée. Par ailleurs, la méthode de la *synthèse algébrique* (Roussel and Lesage, 2014, 2012) permet de formaliser (en expressions booléennes) des exigences exprimées en langage naturel, tout en traitant les

[*] http://www.uppaal.org/
[†] http://www.it.uu.se/research/group/darts/uppaal/port/
[‡] http://nusmv.fbk.eu/

incohérences qu'elles contiennent initialement. Toutefois, les expressions booléennes employées rendent difficilement compte des notions temporelles.

Cet état de l'art montre que peu d'approches améliorent conjointement la modélisation d'un système et l'expression des exigences qui lui sont allouées pour les systèmes embarqués critiques. La théorie du contrôle par supervision (Ramadge and Wonham, 1982) permet bien de construire des modèles formels, corrects par construction à partir d'une description du système libre et des spécifications qu'il doit respecter. Elle a déjà été appliquée dans le domaine de l'Ingénierie Systèmes (Markovski and van de Mortel-Fronczak, 2012) mais n'est pas commune pour le secteur industriel automobile. Qui plus est, la phase de formalisation des exigences, initialement informelles en général, demeure problématique dans le cadre de cette approche (Zaytoon and Riera, 2017).

La démarche proposée dans cet article est inspirée des travaux présentés. Elle permet de construire itérativement des modèles comportementaux formels à partir d'exigences exprimées en langage naturel, améliorant ainsi simultanément la modélisation du système et la formulation des exigences. A l'instar de la méthode B et de la synthèse algébrique, les ambiguïtés, incomplétudes et incohérences des exigences sont progressivement identifiées, puis traitées. Toutefois, contrairement à ces méthodes, le but de la démarche est de fournir des spécifications claires et non ambigües au fournisseur (en charge de la réalisation du système) et non de générer du code exécutable. La démarche utilise les méthodes formelles du *model checking* et le principe de composition d'automates mis en œuvre dans le contexte de la théorie du contrôle par supervision ; le tout appuyé par les avis d'experts. Les exigences initiales sont modifiées et complétées suite à la confrontation avec les experts. Par ailleurs, des solutions alternatives, implicitement contenues dans les exigences initiales, sont mises en évidence.

3 Démarche proposée

3.1 Aperçu général de la démarche

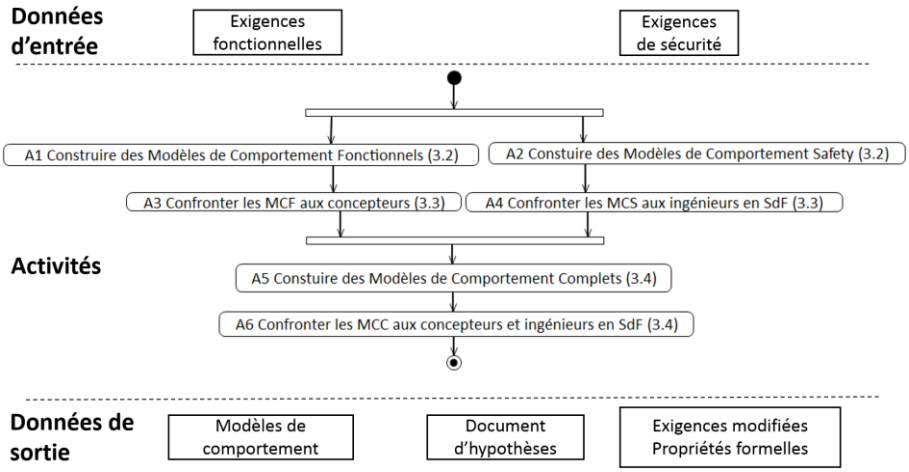

Figure 1: Aperçu de la démarche déployée

La Figure 1 donne un aperçu de la méthode adoptée. Les deux données d'entrée principales sont les exigences fonctionnelles d'une part, et les exigences de sécurité, d'autre part. Les deux types d'exigences sont alloués à la fonction conçue. Deux activités (A1 et A2) sont ensuite menées en parallèle : la construction de Modèles de Comportement Fonctionnel (MCF) et celle de Modèles de Comportement Safety (MCS). Ces modèles sont assortis d'hypothèses, nécessaires à leurs

constructions, et ces hypothèses sont listées dans le *Document d'hypothèses*. Afin de réaliser ces deux premières activités, l'outil UPPAAL est utilisé. Dans le contexte de ce travail, seuls le langage de modélisation et le *model checker* sont utilisés. En effet, le langage de modélisation offre une vue graphique des modèles comportementaux, ce qui est commun pour les ingénieurs. Par ailleurs, le *model checking* fournit une preuve formelle qu'un modèle comportemental est conforme à des propriétés données. Outre ses fonctionnalités et sa prise en main aisée, UPPAAL est également un outil reconnu (Berezin et al., 1998; David and Yi, 2000; Lindahl et al., 1998). Formellement, l'outil UPPAAL manipule des *Automates temporisés communiquant*, c'est-à-dire des tuplets *(L, l_0, C, A , E, I)* où *L* est l'ensemble des places, $l_0 \in L$ la place initiale, *C* l'ensemble des horloges, *A* l'ensemble des actions, *E* l'ensemble des arcs entre les places, ces arcs sont dotés d'une action, d'une garde et d'un ensemble d'horloges, et *I* assigne des invariants aux places (Behrmann et al., 2004). Notons que dans le cadre de la démarche proposée, l'aspect temporisé et les invariants ne sont pas considérés. Ainsi, les modèles construits sont des *Automates à états finis* de la forme *(L, l_0, A, E)* où les arcs de *E* peuvent être simplement pourvus d'une action et d'une garde.

Les deux activités suivantes (A3 et A4) consistent à confronter les modèles obtenus avec les avis d'experts. Etant donné que deux aspects de la fonction ont été étudiés, deux activités sont suivies. D'une part, les MCF sont confrontés aux concepteurs (A3) et, d'autre part, les MCS sont analysés avec les ingénieurs en SdF (A4). A l'issue de ces discussions, les modèles obtenus (nommés MCF_1 et MCS_1) sont effectivement conformes au comportement attendu par les experts (voir section 3.3). Ces activités montrent un des intérêts principaux de la méthode proposée. En effet, deux types d'hypothèses ont été pris durant les activités précédentes : des *hypothèses d'interprétation*, qui mènent à la modification des exigences initiales, et des *hypothèses de modélisation*, qui restent des hypothèses (voir section 3.2). Ces différentes hypothèses sont alors passées en revue. Le nouvel ensemble d'exigences obtenu est stocké dans le document *Exigences modifiées*. Les activités A3 et A4 produisent donc des modèles comportementaux, nommés MCF_1, MCS_1 et leurs variantes, les hypothèses associées, les exigences complètes, non ambigües, claires et vérifiables (qualités usuellement requises pour la formulation d'exigence (Holt et al., 2012; Institute of Electrical and Electronics Engineers, 2011)) ainsi que les propriétés formelles déduites. Ces qualités sont requises pour les exigences car, s'ils subsistent par exemple des ambigüités dans leur formulation, ces dernières peuvent être interprétées de plusieurs façons différentes. Ceci peut occasionner des problèmes lors de la réalisation effective du système (exigences contradictoires, faisabilité des exigences, possibilité de vérifier les exigences…).

Cependant, les exigences fonctionnelles ne peuvent être vérifiées sur MCS_1 et les exigences de sécurité ne peuvent l'être sur MCF_1. En outre, les exigences de sécurité adressent des sous-fonctions redondantes tandis que les exigences fonctionnelles sont allouées à la fonction globale (voir section 3.2, Figure 3). Par conséquent, il est nécessaire de construire un modèle complet, sur lequel toutes les exigences traitées puissent être vérifiées. Ceci est précisément l'objectif des deux dernières activités (A5 et A6) de la démarche. Etant donné que la construction du modèle complet repose sur la composition d'automates, l'outil *Supremica*, également reconnu (Benoit Rohée et al., 2006; Markovski and van de Mortel-Fronczak, 2012; Mohajerani et al., 2017), est utilisé. Les objets manipulés dans cet outil sont des *Automates à états finis déterministes* définis par les tuplets $(Q, q_i, Q_x, Q_m, \Sigma, \delta)$ où Q est l'ensemble fini des états, q_i l'état initial, Q_x l'ensemble des états interdits, Q_m l'ensemble des états marqués, Σ l'alphabet, ensemble fini des événements, et δ la fonction de transition définie telle que $\delta(q_k, \sigma) = q_l$ où σ est un événement (Akesson et al., 2003). A noter que les notions d'états marqués et interdits ne sont pas utilisées pour la méthode déployée. Ceci revient donc à manipuler les tuplets (Q, q_i, Σ, δ). Dans le contexte de la démarche proposée (les difficultés de transposition d'un automate décrit dans l'outil UPPAAL à *Supremica*, évoquées dans (Koolmees et al., 2011) ne sont ici pas rencontrées), on retrouve ainsi les mêmes types d'objets que ceux manipulés précédemment, avec *L* et *Q*, l'ensemble des places (UPPAAL) et des états (*Supremica*), l_0 et q_i, la place (UPPAAL) et l'état (*Supremica*) initial(e), *A* et Σ, l'ensemble des actions (UPPAAL) et événements (*Supremica*), *E* et δ,

l'ensemble des arcs (UPPAAL) et fonctions de transition (*Supremica*). Enfin, de la même manière que les activités précédentes, le modèle résultant est associé à des hypothèses et mène à la modification et création d'exigences (activité A6).

3.2 Construction des modèles de comportement

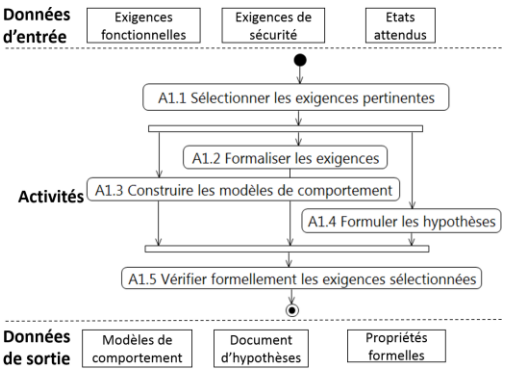

Figure 2: Construction des modèles de comportement

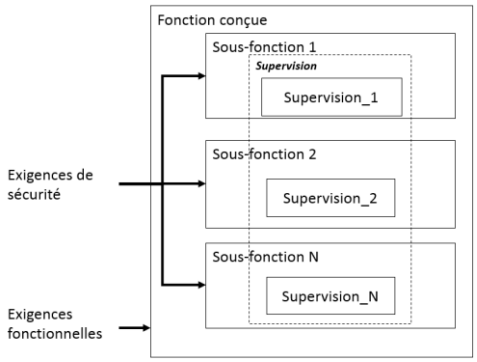

Figure 3: Architecture interne de la fonction conçue

La Figure 2 illustre le détail des activités A1 et A2 (ces deux activités sont menées de manière similaire : les sous-activités de A2 sont nommées A2.1, A2.2, A2.3, A2.4 et A2.5). Les exigences fonctionnelles et celles de sécurité sont toutes exprimées en langage naturel. La Figure 3 représente l'architecture simplifiée de la fonction conçue. Elle est composée de *N* sous-fonctions redondantes (pour des raisons de disponibilité et de sécurité) qui réalisent ensemble la fonction globale. Chaque sous-fonction intègre, entre autres fonctions, un bloc fonctionnel, nommé *Supervision_i*, chargé de gérer les états de la sous-fonction. La fonction développée comprend également un bloc fonctionnel *Supervision* qui gère les états de la fonction globale à partir des supervisions au niveau local (des sous-fonctions). Les exigences fonctionnelles décrivent le comportement de la fonction globale étudiée, du point de vue de l'utilisateur. Elles déterminent les actions à réaliser dans des états particuliers de la fonction ainsi que les conditions pour passer d'un état à un autre. Les exigences de sécurité résultent des analyses de risques. Elles spécifient les réactions des sous-fonctions en cas de défaillance d'un organe (ECU, alimentation de l'ECU, réseau de communication…) causant la perte d'une sous-fonction. Le cas de double défaillance indépendante simultanée est exclu de cette étude. La différence fondamentale avec les exigences fonctionnelles est qu'elles adressent les sous-fonctions redondantes.

La seconde donnée d'entrée disponible contient deux listes d'états attendus (*Etats attendus* sur la Figure 2): l'une du point de vue fonctionnel et l'autre donnée par le métier Sûreté de Fonctionnement. Elles sont déterminées après discussion avec les concepteurs, d'une part, et les ingénieurs en SdF, d'autre part.

Afin de procéder à la sélection des exigences pertinentes (A1.1, Figure 2) un critère, directement tiré de l'objectif principal de ce travail (garantir que la fonction développée soit toujours dans un état sûr) a été défini. Ce critère stipule que le **contenu de l'exigence** doit être relatif à un **changement d'état**. Ceci peut être déterminé grâce aux listes d'états attendus.

A partir des exigences sélectionnées, il est ensuite nécessaire de formuler des hypothèses supplémentaires (A1.4, Figure 2**)** pour être en mesure de construire les modèles de comportement (A1.3, Figure 2). Deux types d'hypothèses sont pris :

Hypothèse d'interprétation : il s'agit d'une hypothèse émise pour compléter une exigence qui n'est pas complète. Une exigence est dite *complète* si elle définit un ou plusieurs état(s) initial(ux), une

ou plusieurs condition(s) et un état final. Cette hypothèse implique la modification ou la création d'exigences. La confrontation avec les experts permet de trancher entre les hypothèses et seules les interprétations les plus crédibles sont retenues. Ce qui donne lieu aux conceptions alternatives.

Hypothèse de modélisation : c'est une hypothèse portant sur le cadre plus global de modélisation. Elle consiste, par exemple, à modéliser le système étudié comme un Système à Evénements Discrets. Ces hypothèses cadrent les interprétations de la modélisation.

Ces définitions font appel aux notions suivantes :

- ***Etat initial*** : sans notion de ***contexte***, il s'agit du premier état parcouru de l'ensemble des états d'un automate (nommé l_0 ou q_i). Il est unique pour les automates dans le cadre de cette étude (automates déterministes). Néanmoins, pour un ***contexte donné*** par une exigence, il peut y avoir un ensemble d'états, également nommés états ***initiaux***, comprenant ou non l'état l_0 ou q_i.
- ***Contexte*** : ensemble d'état(s) du système pour lequel une exigence donnée s'applique.

Chaque exigence sélectionnée est exprimée sous la forme d'une ou plusieurs exigence(s) complète(s), ce qui permet de construire les modèles de comportement selon les deux points de vue (A1.3, Figure 2 et A2.3). Parallèlement à la construction de ces modèles comportementaux l'activité de formalisation des exigences est menée (A1.2, Figure 2). La méthode appliquée est inspirée des travaux (Bitsch, 2000; Ghazel et al., 2015). (Ghazel et al., 2015) propose un processus de modification itératif d'exigences (Figure 4) initialement exprimées en langage naturel afin d'en déduire des propriétés formelles. L'idée est d'appliquer trois opérations : *clarification*, pour supprimer les ambiguïtés et erreurs contenues dans les exigences initiales, *fractionnement*, pour exhiber les exigences élémentaires implicitement comprises dans les exigences initiales et, *modification*, relative aux informations contenues dans les exigences. Une fois ces différentes opérations réalisées, l'exigence initiale est transformée en une ou plusieurs *exigence(s) formalisable(s).* La notion d'exigence formalisable est définie ainsi : « une exigence qui peut être directement formalisée comme une propriété », *i.e.* une formule logique en *CTL* (Full Computation Tree Logic)*.

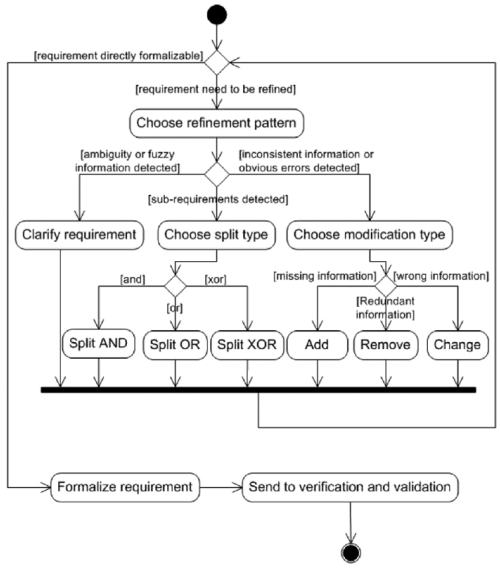

Figure 4: Processus de formalisation des exigences (Ghazel et al., 2015)

La démarche proposée dans le cadre de cet article consiste à décliner différentes étapes jusqu'à obtenir des *exigences complètes*, au sens défini précédemment. Cette forme d'exigence constitue le point d'arrêt du processus de formalisation. Puis, à partir de ces exigences complètes, il est possible de générer différentes propriétés formelles selon les nuances temporelles que l'on souhaite exprimer. Pour ce faire, nous suivons la classification selon le type d'exigence (exigence statique, dynamique, portant sur une séquence d'événements…) fournie par (Bitsch, 2000) . Pour chaque type d'exigence, un énoncé est donné, ainsi que sa traduction en *CTL*. Dans le cadre de ces travaux, la formulation suivante a été adoptée :

A [ ] (a imply b) (1)

Où **A** est l'opérateur *All* (tous les chemins possibles), **[ ]** signifie *toujours, dans le futur*, **a** est un état et/ou une/des condition(s) et **b** est un état. L'expression **(1)** signifie donc : dans un état **a** donné et/ou si une/des condition(s) est/sont remplie(s), alors pour tous les états futurs de tous les chemins possibles, l'état **b** est atteint. Lors de cette activité, des variables et des paramètres ont dû être déterminés ou identifiés. Les ***variables*** représentent l'évolution des données mesurables, telles que la vitesse ou l'accélération, tandis que les ***paramètres*** sont des valeurs particulières (seuils) prises par les variables.

Une fois l'ensemble des exigences sélectionnées traduites, il est possible de procéder aux activités de vérification formelle à l'aide d'UPPAAL (A1.5, Figure 2). Ceci permet d'obtenir deux ensembles {modèles comportementaux, exigences, propriétés formelles, hypothèses} cohérents.

3.3 Confrontation avec les compétences des experts

Des hypothèses ont été prises afin de construire les modèles de comportement MCF et MCS. Il est ensuite nécessaire de les confronter aux avis d'experts, ce qui est le but des activités A3 et A4 (Figure 1). Les hypothèses d'interprétation et de modélisation sont alors passées en revue, lors de séances de travail avec les concepteurs et les ingénieurs en SdF étant à l'origine des exigences traitées.

Les *hypothèses d'interprétation* sont vérifiées en examinant les différents modèles de comportement auxquels elles donnent lieu (voir 4.4). A l'issue de cette analyse, certaines hypothèses sont retenues et d'autres rejetées, en conservant les justifications de ces choix dans le *Document d'hypothèses*.

Par ailleurs, le langage graphique utilisé (modèle d'état) facilite grandement la détection d'erreurs dans les exigences initialement contenues dans des tableaux, parfois volumineux. Il arrive en effet que le comportement d'une fonction ne soit pas défini correctement par certaines exigences. Ceci peut se détecter par la relecture de l'ensemble des exigences, mais la représentation graphique sous forme de modèles d'état facilite ce travail (voir 4.4).

Deux ensembles d'exigences complètes, cohérentes entre elles, vérifiables et non ambigües sont obtenus à l'issu de ce travail. Toutefois, les deux points de vue ne sont pas nécessairement cohérents. C'est pourquoi un modèle comportemental complet, sur lequel l'ensemble des exigences sélectionnées peuvent être vérifiées, doit être construit. Ceci est l'objectif des activités A5 et A6 (Figure 1).

3.4 Construction d'un modèle complet

La validation des modèles de comportement obtenus résultant des activités précédentes est assurée par la vérification formelle, d'une part, et la consultation des experts, d'autre part. Afin de rendre les points de vue fonctionnel et de SdF cohérents, une approche fondée sur la composition parallèle d'automates à états finis est proposée. C'est la raison pour laquelle *Supremica* est utilisé pour mener à bien l'activité A5.

Comme le montre la Figure 5, la première étape consiste à déterminer $MCG_{attendu} = (Q_{at}, \Sigma_{at}, q_{iat}, \delta_{at})$ à partir de $MCF_1 = (Q_{f1}, \Sigma_{f1}, q_{if1}, \delta_{f1})$ de telle sorte que $Q_{at} = Q_{f1} \cup Q_{sûr}$, $\Sigma_{at} = \Sigma_{f1} \cup \Sigma_{déf}$, $q_{iat} = q_{if1}$, $\delta_{at}(q_{f1}, \sigma_{déf}) = q_{sûr}$ où $Q_{sûr}$ est l'ensemble des états sûrs, $\Sigma_{déf}$, l'ensemble des défaillances envisagées, $q_{sûr} \in Q_{sûr}$ et $\sigma_{déf} \in \Sigma_{déf}$. Cette activité est réalisée conjointement avec les concepteurs et les ingénieurs en SdF pour définir ensemble $Q_{sûr}, \Sigma_{déf}$ et δ_{at}.

La seconde étape est la modification itérative des automates du MCS_1 jusqu'à ce que l'automate résultant de leur *produit synchronisé* (MCG_{obtenu}) soit identique au $MCG_{attendu}$ défini lors de l'étape précédente. Plus formellement, on a, pour *N* sous-fonctions redondantes (Figure 3) :
$MCS_1 = \{A_{s1}^1, A_{s1}^2, \dots, A_{s1}^N\}$ où $A_{s1}^j = (Q_{s1}^j, \Sigma_{s1}^j, q_{is1}^j, \delta_{s1}^j)$ et $MCG_{attendu} = (Q_{at}, \Sigma_{at}, q_{iat}, \delta_{at})$ et on cherche $MCS_3 = \{A_{s3}^1, A_{s3}^2, \dots, A_{s3}^n\}$ où $A_{s3}^j = (Q_{s3}^j, \Sigma_{s3}^j, q_{is3}^j, \delta_{s3}^j)$ tel que
$\boldsymbol{MCG_{attendu} = A_{s3}^1 \parallel A_{s3}^2 \parallel \cdots \parallel A_{s3}^N}$, c'est-à-dire tel que, pour deux automates A_{s3}^1 et A_{s3}^2 (Faraut, 2010):

$Q_{at} = Q_{s3}^1 \; x \; Q_{s3}^2$; $\Sigma_{at} = \Sigma_{s3}^1 \cup \Sigma_{s3}^2$; $q_{iat} = (q_{is3}^1, q_{is3}^2)$;

$$\delta_{at}((q_{s3}^1, q_{s3}^2), \sigma) = \begin{cases} \left(\delta_{s3}^1(q_{s3}^1, \sigma), \delta_{s3}^2(q_{s3}^2, \sigma) \right) si \; \delta_{s3}^1(q_{s3}^1, \sigma) \; et \; \delta_{s3}^2(q_{s3}^2, \sigma) \; sont \; définies \\ \left(\delta_{s3}^1(q_{s3}^1, \sigma), q_{s3}^2 \right) si \; \delta_{s3}^1(q_{s3}^1, \sigma) \; est \; définie \; et \; \sigma \notin \Sigma_{s3}^2 \\ \left(q_{s3}^1, \delta_{s3}^2(q_{s3}^2, \sigma) \right) si \; \delta_{s3}^2(q_{s3}^2, \sigma) \; est \; définie \; et \; et \; \sigma \notin \Sigma_{s3}^1 \\ non \; définie \; sinon \end{cases}$$

Les états des automates A_{s3}^j sont dans un premier temps définis tels que $q_{at} = (q_{s3}^1, q_{s3}^2, \dots, q_{s3}^N)$. Ceci est, à l'instar de la détermination de $MCG_{attendu}$, effectué en collaboration avec les concepteurs et les ingénieurs en SdF. Puis, dans un second temps, les transitions des automates sont modifiées jusqu'à l'obtention de l'égalité recherchée ($MCG_{attendu} = A_{s3}^1 \parallel A_{s3}^2 \parallel \cdots \parallel A_{s3}^N$). Actuellement, cette étape reste un processus de modélisation mené en collaboration avec les concepteurs et les ingénieurs en SdF. Il vise à obtenir l'adéquation des machines à états finis aux perceptions de l'ingénierie et de la SdF. Afin d'apporter une meilleure cohérence de ce processus, cette étape est intégrée dans une démarche d'ingénierie des systèmes.

Enfin, un Modèle de Comportement Complet (MCC) est déterminé, il est composé de $MCG_{attendu}$ et de MCS_3, ainsi que des exigences modifiées, leurs propriétés associées et les hypothèses prises.

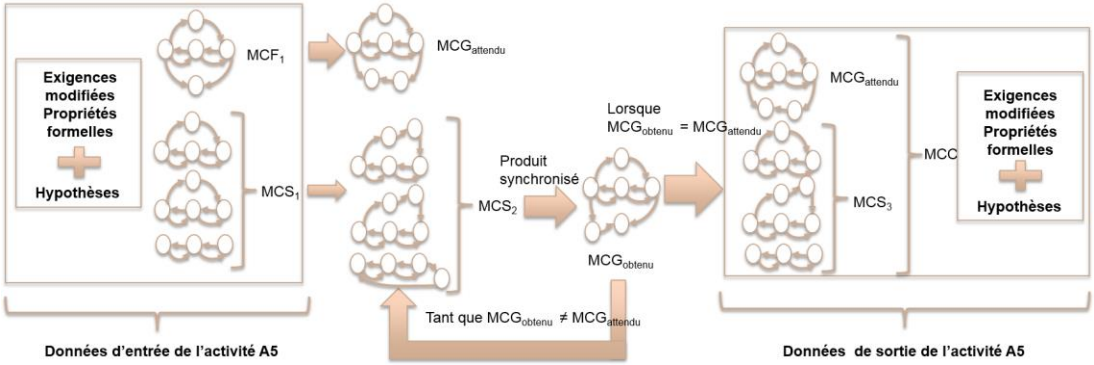

Figure 5: Représentation graphique de l'activité A5

4 Application à la fonction AD

4.1 Conception de la fonction AD

La Figure 6 représente une architecture fonctionnelle statique simplifiée d'un véhicule récent équipé de systèmes d'aide à la conduite nécessaires à la mise en œuvre de la fonction AD. Les trois principaux ensembles fonctionnels sont celui affectant à la fonction AD ses données d'entrée (relatives aux conditions environnementales, au comportement du conducteur et à l'état du véhicule) : blocs *Localization*, *Perception* et *Enabling systems* ; celui représentant la fonction AD dont le rôle principal est de générer continuellement une commande de trajectoire (sous-fonctions *Fusion*, *Target selection* et *Trajectory planning*) au troisième bloc fonctionnel, *Acting*, chargé d'appliquer cette commande. La fonction AD peut être dans différents états (*Active*, *Available* par exemple) gérés par le bloc fonctionnel *AD management level*. Plus précisément le sous-bloc *Supervision* détermine en permanence dans quel état la fonction AD doit être. Cette supervision est au cœur de l'étude menée. Dans un état spécifique de la fonction AD, appelé MRM pour *Minimal Risk Manoeuver*, le véhicule engage des manœuvres prédéterminées dans le but de garantir la sûreté dans les circonstances autorisées. Ce sont les exigences de sécurité qui spécifient cet aspect.

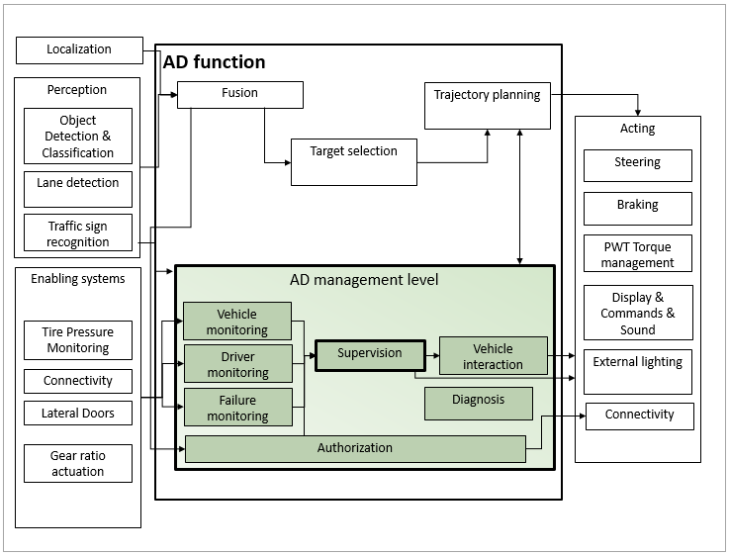

Figure 6: Architecture fonctionnelle simplifiée d'un véhicule équipé de systèmes d'aide à la conduite

Ceci met donc en lumière l'importance cruciale de la considération des résultats des analyses de risques en phase amont de conception. L'architecture interne de la fonction AD est semblable à celle présentée sur la Figure 3. La fonction AD est donc composée de 3 sous-fonctions redondantes, nommées *main_AD, sub_AD* et *AD-3,* qui réalisent ensemble la fonction globale. Chaque sous-fonction intègre, entre autres fonctions, un bloc fonctionnel chargé de gérer les états de la sous-fonction (*Supervision_main, Supervision_sub* et *Supervision_3*). La fonction AD comprend également un bloc fonctionnel, nommé *Supervision*, qui gère les états de la fonction globale.

4.2 Sélection des exigences pertinentes

Dans le cadre de l'étude menée, environ 175 exigences fonctionnelles ont été fournies. En voici deux exemples (FR signifie *Functional Requirement*):

FR1: « AD function shall not be available within a construction area »
FR2: « If AD function is active, then the HMI shall always give information of the autonomous vehicle status. »

Environ 350 exigences de sécurité ont été exprimées, dont les deux exemples suivants (SR signifie *Safety Requirement*):

SR1: « In case of an excessive deceleration due to a failure of sub_AD, AD-3 shall switch itself on»
SR2: « SPF (Single Point Fault) Metric of AD-3 shall comply with target value 99 percent »

Les deux listes d'états attendus (*Etats attendus* sur la Figure 2) sont, dans le cas de la fonction AD, les suivantes :
Point de vue fonctionnel : *Off, Not_available, Available, Activatable, Active*
Point de vue SdF :
Main_AD : *Off, Active, MRM* Sub_AD : *Off, Standby, On* AD-3 : *Standby, On*

En appliquant le critère défini dans la section 3.2, **FR1** et **SR1** sont sélectionnées car elles spécifient bien un changement d'état. Au contraire, **FR2** n'est pas retenue car elle définit une action à réaliser dans un état donné. De même, **SR2** est rejetée car elle concerne la fiabilité du composant réalisant la fonction AD-3. Au final, environ 40% des exigences fonctionnelles initiales (70) et 20% des exigences de sécurité initiales (73) ont été retenues.

4.3 Construction des modèles d'état

Une fois les exigences pertinentes sélectionnées, la construction des modèles d'état (A1.3, Figure 2) peut débuter. Cette activité nécessite de formuler des hypothèses.

Pour l'exigence **FR1**, *15 hypothèses d'interprétation* ont été prises. En effet, le(s) état(s) initial(ux) n'est(ne sont) pas clairement indiqué(s) dans la formulation de l'exigence. Ainsi, les quatre cas suivants doivent être *a priori* considérés :

1) L'état initial est un des 4 états (4 possibilités). 4 modèles d'état peuvent donc être construits : dans le premier modèle, l'état initial est *Off*, dans le second : *Available*, dans le troisième : *Activatable* et dans le dernier : *Active*
2) Les états initiaux sont deux états parmi les quatre possibles (6 possibilités)
3) Les états initiaux sont trois états parmi les quatre possibles (4 possibilités)
4) Les états initiaux sont les quatre possibles (1 possibilité)

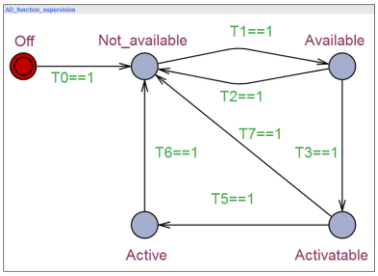

Figure 7: Modèle de Comportement Fonctionnel (MCF)

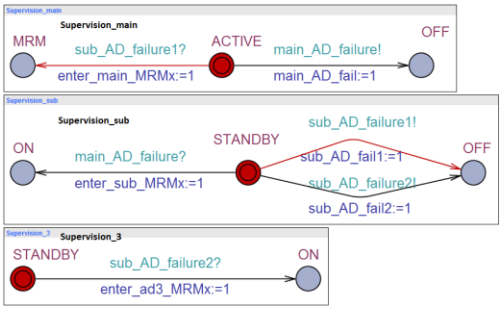

Figure 8: Modèle de Comportement Safety (MCS)

La Figure 7 représente un modèle d'état possible, construit à partir des exigences fonctionnelles sélectionnées et complétées. Le seul état perceptible par le conducteur est l'état *Active*, les autres états sont des états d'initialisation de la fonction AD. Les gardes des transitions *Ti* contiennent les conditions définies dans les exigences fonctionnelles sélectionnées.

Le même travail a été réalisé du point de vue de la SdF. La Figure 8 illustre un modèle d'état safety obtenu. Ce modèle est en réalité une version agrégée du véritable modèle d'état issu des exigences de sécurité sélectionnées. D'autres défaillances sont considérées mais ont toutes les mêmes conséquences sur le comportement des sous-fonctions. Ainsi, l'événement *main_AD_failure!* regroupe toutes les défaillances ayant pour effet le passage de la sous-fonction *main_AD* à l'état *OFF*.

Parallèlement à l'activité de construction des modèles d'état, les exigences retenues sont formalisées (A1.2, Figure 2). Pour **FR1**, une variable Booléenne, nommée *construction_area*, a été définie telle que *construction_area==0* signifie que le véhicule autonome ne se trouve pas dans une zone de travaux. Or, comme indiqué précédemment, **FR1** est sujette à 15 interprétations, donc 15 formalisations sont possibles. Il est considéré que chaque interprétation donne lieu à une propriété logique spécifique. Ainsi

chaque interprétation correspond à une propriété formelle, attachée à une *Hypothèse d'Interprétation* (***HI***). Ci-dessous est reportée la 1ère interprétation, notée **FP1-1** (FP pour *Functional Property*) :

FP1-1: A [ ] ((construction area==1 && AD function.Available) imply (AD_function.Not_Available))

HI1: Etat initial: *Available*

Une démarche similaire a été adoptée pour les exigences de sécurité. La différence principale est que les paramètres définis sont des *paramètres d'observation*, permettant de procéder à la vérification formelle, ce qui signifie que ces paramètres ne sont utilisés que dans le cadre de cette modélisation afin d'observer l'occurrence d'événements particuliers. Par exemple, le paramètre *main_AD_fail* (Figure 8) indique que la fonction *main_AD* a subi une défaillance. De nouveaux paramètres ont donc dû être déterminés et constituent des *hypothèses de modélisation*, reportées dans le *Document d'hypothèses* (Figure 1). Il s'agit bien d'hypothèses car il est supposé que les événements considérés sont observables par le changement de valeur de ces paramètres. Or il est possible que l'observation de ces événements nécessitent plusieurs sources d'informations et non seulement un paramètre. Pour les exigences de sécurité traitées, il n'a pas été nécessaire de formuler des hypothèses d'interprétation. Une fois que l'ensemble des exigences retenues a été traduit, il est possible de les vérifier formellement à l'aide d'UPPAAL. Cette vérification est effectuée au cours des activités A1.5 (voir Figure 2) et A2.5 et clôt la construction des modèles comportementaux MCF et MCS.

4.4 Confrontation avec les avis d'experts

Toutes les hypothèses prises pour construire les modèles d'état doivent être confrontées aux experts. C'est le but des activités A3 et A4 (Figure 1).

Concernant les hypothèses d'interprétation, chaque exigence incomplète a dû être traitée. Pour **FR1**, l'hypothèse d'interprétation 1 (**HI1**; état initial : *Available*) a été considérée comme correcte. En effet, l'état *Off* ne peut être l'état initial car la condition pour entrer dans l'état *Not_available* est que l'alimentation électrique soit enclenchée. Par ailleurs si le véhicule autonome entre dans une *construction area* alors que la fonction AD est dans l'état *Active*, des conditions plus spécifiques sont détaillées dans d'autres exigences. Ces deux solutions sont donc éliminées. En revanche, pour le cas où le véhicule autonome entre dans une *construction area* alors que la fonction AD est dans l'état *Activatable*, deux solutions sont alors envisageables :

a) La fonction AD doit passer par l'état *Available* avant d'atteindre l'état *Not_available*

b) La fonction AD entre directement dans l'état *Not_available* depuis l'état *Activatable*

Les deux solutions ont été retenues et donnent lieu à deux conceptions alternatives, qui seront confrontées par la suite. Pour les deux cas, une nouvelle formulation de **FR1** correspondant à sa première interprétation (**HI1**) est écrite:

FR1-1: « If AD function is in the state *Available* and the autonomous vehicle stands in a construction area, then AD function shall switch to the state *Not_available* »

En complément, deux nouvelles exigences, nommées **FR1a** et **FR1b**, correspondant aux deux solutions concurrentes sont formulées :

FR1a: « If AD function is in the state *Activatable* and the autonomous vehicle stands in a construction area, then AD function shall switch to the state *Available* »

FR1b: « If AD function is in the state *Activatable* and the autonomous vehicle stands in a construction area, then AD function shall switch to the state *Not_available* »

Le modèle d'état correspondant à la solution **b** est en fait celui représenté sur la Figure 7 tandis que le modèle d'état de la solution **a** est illustré par la Figure 9.

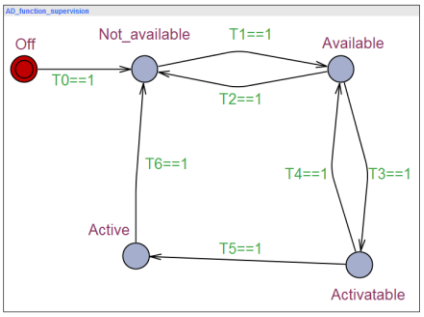

Figure 9: Modèle de Comportement Fonctionnel$_1$, solution a (MCF$_1$a)

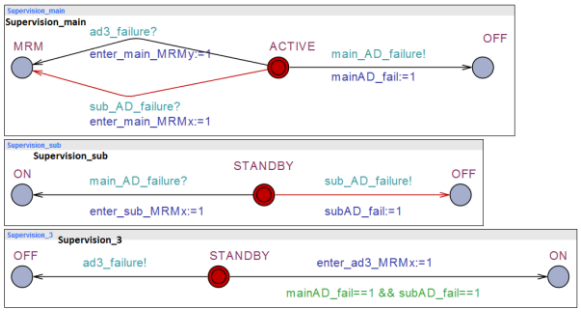

Figure 10: Modèle de Comportement Safety$_1$ (MCS$_1$)

Du point de vue de la SdF, l'analyse avec les ingénieurs en SdF a mis en évidence que la sous-fonction *AD-3* n'était pas spécifiée conformément à leurs attentes. Il apparaît (Figure 8) que l'*AD-3* intervient en cas de certaines défaillances affectant la fonction *sub_AD*. Or l'*AD-3* ne doit en réalité changer d'état que si une défaillance de cause commune, touchant simultanément les fonctions *main_AD* et *sub_AD*, survient. La Figure 10 restitue le modèle comportemental désiré pour les trois sous-fonctions. Etant donné qu'aucune interprétation n'a dû être faite pour formaliser les exigences de sécurité sélectionnées, il n'y a qu'un seul modèle d'état solution.

4.5 Construction du modèle complet

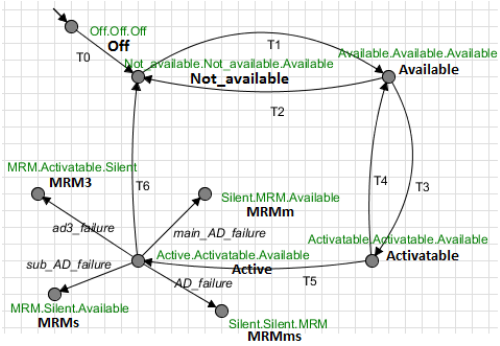

Figure 11: MCG$_{attendu}$

Pour l'illustration de la méthode, la solution **a** est retenue, mais le travail est similaire pour la solution **b**. Afin de déterminer MCG$_{attendu}$ (Figure 11), $Q_{sûr}$, $\Sigma_{déf}$ et δ_{at} sont définis comme suit :

- $Q_{sûr} = \{MRMm, MRMs, MRM3, MRMms\}$
- $\Sigma_{déf} = \{main\_AD\_failure, sub\_AD\_failure, ad3\_failure, AD\_failure\}$
- $\delta_{at}(Active, main\_AD\_failure) = MRM_m$, $\delta_{at}(Active, sub\_AD\_failure) = MRM_s$, $\delta_{at}(Active, ad3\_failure) = MRM3$, $\delta_{at}(Active, AD\_failure) = MRM_{ms}$

MCG$_{attendu}$ (q_{at})	Supervision_main (q^1_{s3})	Supervision_sub (q^2_{s3})	Supervision_3 (q^3_{s3})
Off	Off	Off	Off
Not_available	Not_available	Not_available	Available
Available	Available	Available	Available
Activatable	Activatable	Activatable	Available
Active	Active	Activatable	Available
MRMm	Silent	MRM	Available
MRMs	MRM	Silent	Available
MRMms	Silent	Silent	MRM
MRM3	MRM	Activatable	Silent

Tableau 1: Correspondance états globaux/états locaux

Dans un second temps, les états des automates MCS_3 sont déterminés tels que $q_{at} = (q_{s3}^1, q_{s3}^2, q_{s3}^3)$. Cette opération est faite, également en concertation avec les ingénieurs en SdF et les concepteurs, et le résultat est donné dans le Tableau 1.

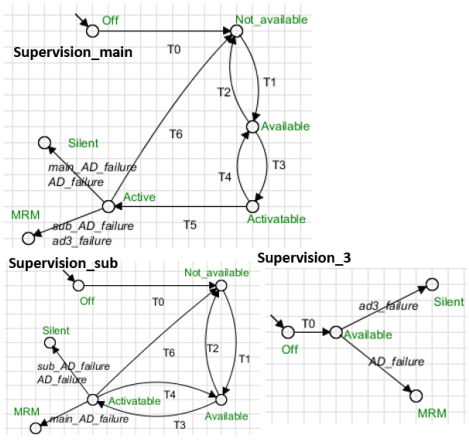

Figure 13: MCS_3

Enfin, les automates du MCS_1 (Figure 10) sont progressivement modifiés jusqu'à obtenir MCS_3 (Figure 13) de la manière suivante :

- *Supervision_main* : ajout des transitions *T0* à *T6*. Ajout des états *Off* (au sens non alimenté électriquement), *Not_available*, *Available*, *Activatable*. Remplacement de l'état *Off* initial (au sens réaction suite à l'occurrence d'une défaillance) par l'état *Silent.*

- *Supervision_sub* : ajout des transitions *T0*, *T1*, *T2*, *T3*, *T4*, *T6*. Ajout de l'état *Off* (au sens non alimenté électriquement). Remplacement de l'état *Standby* par les états *Not_available*, *Available*, *Activatable*. Remplacement de l'état *Off* initial (au sens réaction suite à l'occurrence d'une défaillance) par l'état *Silent*.

- *Supervision_3* : ajout de la transition *T0*. Ajout de l'état *Off* (au sens non alimenté électriquement). Remplacement de l'état *On* par l'état *Available*. Remplacement de l'état *Off* initial (au sens réaction suite à l'occurrence d'une défaillance) par l'état *Silent*.

Les automates corrects obtenus (formant MCS_3) sont représentés sur la Figure 13. La construction des automates du modèle MCS_3 nécessite de nouvelles hypothèses, relatives aux modifications effectuées. Enfin, le produit synchronisé des trois automates du MCS_3 (MCG_{obtenu}) est bien identique au $MCG_{attendu}$ (au sens défini section 3.4), ce qui prouve l'exactitude des trois automates du MCS_3.

Cependant, des hypothèses supplémentaires ont été prises pour la construction d'un modèle complet. Certaines, comme la modification des modèles d'état des sous-fonctions, sont réalistes et mènent à la formulation de nouvelles exigences. Ainsi, 70 nouvelles exigences ont été allouées à la fonction *main_AD*, 35 à la fonction *sub_AD* et une à la fonction *AD-3*. En revanche, l'hypothèse de la synchronisation parfaite entre les transitions *Ti* n'est pas acceptable. Ceci met en évidence la nécessité de prendre en compte certaines contraintes de l'implémentation dans la modélisation proposée. Cet aspect constitue justement une perspective des travaux présentés dans cet article.

5 Conclusion

L'introduction de la fonction de conduite autonome accentue les contraintes de SdF pesant sur les systèmes embarqués en charge de sa réalisation. Des approches appropriées permettant de prendre en compte le plus d'exigences possibles dès le début de la conception et d'éviter la plupart des erreurs commises durant cette phase doivent être élaborées. Cet article contribue précisément à cet important travail en proposant une méthode améliorant conjointement la modélisation comportementale d'un système et la formulation des exigences qui lui sont allouées.

La démarche déployée a effectivement permis de construire des modèles de comportement formels à partir d'exigences initialement exprimées en langage naturel. De plus, les exigences associées ont été

consolidées par deux moyens principaux. Le premier est la formalisation en elle-même, qui permet de mettre en évidence certaines lacunes, telles que des ambiguïtés, des incomplétudes ou incohérences. Le second est l'aspect graphique des modèles d'état qui facilite grandement la détection d'erreurs par rapport à l'expression textuelle des exigences, sans perte de rigueur. Enfin, il doit être noté que cette démarche repose également sur les compétences des experts.

Cet article se focalise sur le processus de vérification des exigences et la formulation de ces exigences, sans apporter de précisions sur l'intégration de la méthode proposée dans un processus plus global d'Ingénierie Systèmes. Enfin, le comportement des composants assurant les fonctions n'est pas pris en compte de manière systématique et structurée.

En conséquence, deux principales perspectives sont envisagées. La première est d'utiliser les langages de modélisation de l'IS (SysML, EAST-ADL) et les outils associés (*Rational Doors*[§], modeleurs SysML) pour intégrer la démarche dans un contexte MBSE[**] plus global et faciliter sa réutilisabilité. Par ailleurs les contraintes d'implémentation doivent être formalisées afin d'être prises en compte de manière homogène et structurée dans la méthode.

Références

Akesson, K., Fabian, M., Flordal, H., Vahidi, A., 2003. Supremica—a tool for verification and synthesis of discrete event supervisors, in: 11th Mediterranean Conference on Control and Automation.

Apvrille, L., Becoulet, A., 2012. Prototyping an embedded automotive system from its UML/SysML models. Proc. Embed. Real Time Syst. Softw. 87–124.

Baier, C., Katoen, J.-P., 2008. Principles of model checking. The MIT Press, Cambridge, Mass.

Behm, P., Benoit, P., Faivre, A., Meynadier, J.-M., 1999. Météor: A Successful Application of B in a Large Project. Presented at the International Symposium on Formal Methods, Springer Berlin Heidelberg, pp. 369–387.

Behrmann, G., David, A., Larsen, K., 2004. A tutorial on uppaal. Form. Methods Des. Real-Time Syst. 33–35.

Benoit Rohée, Bernard Riera, Véronique Carré-Ménétrier, Jean-Marc Roussel, 2006. A methodology to design and check a plant model, in: 3rd IFAC Workshop on Discrete-Event System Design (DESDes'06). Rydzyna, Poland, pp. 246–250.

Berezin, S., Campos, S., Clarke, E.M., 1998. Compositional reasoning in model checking. Springer.

Bitsch, F., 2000. Classification of Safety Requirements for Formal Verification of Software Models of Industrial Automation Systems, in: Proceedings of the 13th Conference on Software and Systems Engineering and Their Applications. Citeseer.

Boulanger, J.-L., 2014. Formal Methods Applied to Industrial Complex Systems: Implementation of the B Method. John Wiley & Sons.

Broy, M., 2006. Challenges in Automotive Software Engineering, in: Proceedings of the 28th International Conference on Software Engineering, ICSE '06. ACM, New York, NY, USA, pp. 33–42.

Chalé, H., Taofifenua, O., Gaudré, T., Topa, A., Lévy, N., Boulanger, J.-L., 2011. 11.2. 2 Reducing the Gap Between Formal and Informal Worlds in Automotive Safety-Critical Systems, in: INCOSE International Symposium. Wiley Online Library, pp. 1306–1320.

Chen, D., Johansson, R., Lönn, H., Blom, H., Walker, M., Papadopoulos, Y., Torchiaro, S., Tagliabo, F., Sandberg, A., 2011. Integrated safety and architecture modeling for automotive embedded systems*. E Elektrotechnik Informationstechnik 128, 196–202. doi:10.1007/s00502-011-0007-7

Cressent, R., Idasiak, V., Kratz, F., David, P., 2011. Mastering safety and reliability in a model based process, in: Proceedings - Annual Reliability and Maintainability Symposium. pp. 1–6.

David, A., Yi, W., 2000. Modelling and analysis of a commercial field bus protocol. Presented at the Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000, pp. 165–172.

David, P., Idasiak, V., Kratz, F., 2010. Reliability study of complex physical systems using SysML. Reliab. Eng. Syst. Saf. 95, 431–450. doi:10.1016/j.ress.2009.11.015

[§] http://www-03.ibm.com/software/products/fr/ratidoor

[**] *Model Based Systems Engineering*

Evrot, D., 2008. Contribution à la vérification d'exigences de sécurité: application au domaine de la machine industrielle (thèse). Université Henri Poincaré-Nancy I.
Faraut, G., 2010. Commutations sûres de mode pour les systèmes à événements discrets. INSA de Lyon.
Ghazel, M., Yang, J., El-Koursi, E.-M., 2015. A pattern-based method for refining and formalizing informal specifications in critical control systems. J. Innov. Digit. Ecosyst. 2, 32–44.
Gudemann, M., Ortmeier, F., 2010. A Framework for Qualitative and Quantitative Formal Model-Based Safety Analysis. Presented at the 2010 IEEE 12th International Symposium on High Assurance Systems Engineering, pp. 132–141.
Holt, J., Perry, S.A., Brownsword, M., 2012. Model-Based Requirements Engineering. Institution of Engineering and Technology.
Institute of Electrical and Electronics Engineers, 2011. ISO/IEC/IEEE 29148:2011(E) Systems and software engineering — Life cycle processes — Requirements engineering.
Jean-François Pétin, Dominique Evrot, Gérard Morel, Pascal Lamy, 2010. Combining SysML and formal methods for safety requirements verification. Presented at the 22nd International Conference on Software & Systems Engineering and their Applications, Paris, France.
Kaiser, B., Klaas, V., Schulz, S., Herbst, C., Lascych, P., 2010. Integrating system modelling with safety activities, in: International Conference on Computer Safety, Reliability, and Security. Springer, pp. 452–465.
Kalra, N., Paddock, S.M., 2016. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? Transp. Res. Part Policy Pract. 94, 182–193.
Kang, E.-Y., Enoiu, E.P., Marinescu, R., Seceleanu, C., Schobbens, P.-Y., Pettersson, P., 2013. A methodology for formal analysis and verification of EAST-ADL models. Reliab. Eng. Syst. Saf. 120, 127–138.
Koolmees, B., Reniers, M., Markovski, J., 2011. Validation of modeled behavior using UPPAAL. Bachelor Final Proj. Eindh. Univ. Technol. 43.
Koopman, P., Wagner, M., 2017. Autonomous Vehicle Safety: An Interdisciplinary Challenge. IEEE Intell. Transp. Syst. Mag. 9, 90–96.
Leveson, N.G., 2002. An approach to designing safe embedded software, in: International Workshop on Embedded Software. Springer, pp. 15–29.
Lindahl, M., Pettersson, P., Yi, W., 1998. Formal design and analysis of a gear controller, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, pp. 281–297.
Liu, X., Zhu, Z., 2011. Construct Aspectual Models from Requirement Documents for Model-driven Development of Automotive Software. Electron. Notes Theor. Comput. Sci. 274, 33–50.
Markovski, J., van de Mortel-Fronczak, J.M., 2012. Modeling for safety in a synthesis-centric systems engineering framework, in: International Conference on Computer Safety, Reliability, and Security. Springer, pp. 36–49.
Märtin, L., Schatalov, M., Hagner, M., Goltz, U., Maibaum, O., 2013. A methodology for model-based development and automated verification of software for aerospace systems, in: Aerospace Conference, 2013 IEEE. IEEE, pp. 1–19.
Mauborgne, P., Deniaud, S., Levrat, E., Bonjour, E., Micaëlli, J.-P., Loise, D., 2016. Operational and System Hazard Analysis in a Safe Systems Requirement Engineering Process – Application to automotive industry. Saf. Sci. 87, 256–268.
Maurer, M., Winner, H., 2013. Automotive Systems Engineering. Springer Science & Business Media.
Mohajerani, S., Malik, R., Fabian, M., 2017. Compositional synthesis of supervisors in the form of state machines and state maps. Automatica 76, 277–281.
N. Becker, 2011. ISO 26262 :2011(E) Road vehicles - functional safety.
Nebut, C., Fleurey, F., 2005. Une méthode de formalisation progressive des exigences basée sur un modèle simulable. L'OBJET 11, 145–158.
Nouacer, R., Djemal, M., Niar, S., Mouchard, G., Rapin, N., Gallois, J.-P., Fiani, P., Chastrette, F., Lapitre, A., Adriano, T., Mac-Eachen, B., 2016. EQUITAS: A tool-chain for functional safety and reliability improvement in automotive systems. Microprocess. Microsyst. 47, Part B, 252–261.
Owens, B.D., Herring, M.S., Dulac, N., Leveson, N.G., Ingham, M.D., Weiss, K.A., 2008. Application of a Safety-Driven Design Methodology to an Outer Planet Exploration Mission, in: 2008 IEEE Aerospace Conference. Presented at the 2008 IEEE Aerospace Conference, pp. 1–24.
Ramadge, P.J., Wonham, W.M., 1982. Supervision of discrete event processes. Presented at the 1982 21st IEEE Conference on Decision and Control, pp. 1228–1229.

Roussel, J.-M., Denis, B., 2002. Safety Properties Verification of ladder programs.pdf. J. Eur. Systèmes Autom. JESA 36, 905–917.

Roussel, J.-M., Lesage, J.-J., 2014. Design of Logic Controllers Thanks to Symbolic Computation of Simultaneously Asserted Boolean Equations. Math. Probl. Eng. 2014, 1–15.

Roussel, J.-M., Lesage, J.-J., 2012. Algebraic synthesis of logical controllers despite inconsistencies in specifications. IFAC Proc. Vol. 45, 307–314.

Sharvia, S., Papadopoulos, Y., 2015. Integrating model checking with HiP-HOPS in model-based safety analysis. Reliab. Eng. Syst. Saf. 135, 64–80.

Taofifenua, O., 2012. Ontology centric design process: Sharing a conceptualization (PhD. dissertation). Conservatoire national des arts et metiers-CNAM.

Weissnegger, R., Pistauer, M., Kreiner, C., Römer, K., Steger, C., 2015. A novel method to speed-up the evaluation of cyber-physical systems (ISO 26262), in: 2015 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES). Presented at the 2015 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES), pp. 109–114.

Zaytoon, J., Riera, B., 2017. Synthesis and implementation of logic controllers – A review. Annu. Rev. Control 43, 152–168.