L'ingénierie dirigée par les modèles pour assurer la sécurité des logiciels embarqués en automobile

Yandika Sirgabsou, Claude Baron, Lorenzo Grenier, Laurent Pahun, Philippe Esteban

L'ingénierie dirigée par les modèles pour assurer la sécurité des logiciels embarqués en automobile.

Proposition méthodologique et étude de cas.

YANDIKA SIRGABSOU[1,2,4,5], CLAUDE BARON[2, 3,4,5], LORENZO GRENIER[4,5], LAURENT PAHUN[1], PHILIPPE ESTEBAN[2,4,6]

[1] RENAULT SOFTWARE FACTORY
Toulouse, France
laurent.pahun@renault.com

[3] ISAE-SUPAERO
Toulouse, France
claude.baron@laas.fr

[5] INSA-TOULOUSE
Toulouse, France
grenier@etud.insa-toulouse.fr

[2] LAAS-CNRS, CNRS
Toulouse, France
ysirgabsou@laas.fr

[4] UNIVERSITE DE TOULOUSE
Toulouse, France

[6] UNIVERSITE TOULOUSE 3
Toulouse, France
esteban@laas.fr

***Résumé* –L'avènement des véhicules autonomes a été longtemps perçu comme un événement qui allait révolutionner le monde du transport, améliorer le confort et la sécurité des usagers. Bien que cet optimisme soit partagé par la plupart et que d'importants progrès technologiques aient été fait dans ce sens, le chemin ne reste pas pour le moins jonché de nombreux défis, notamment celui de la sécurité, liée à l'utilisation croissante de logiciels embarqués devenus très complexes. Face à cette complexité grandissante, les méthodes manuelles traditionnelles d'analyse de la sécurité basées sur des documents ont atteint leur limite et il est temps de repenser les approches. Nous proposons pour cela de nous appuyer sur l'ingénierie dirigée par les modèles pour mener les analyses de sécurité non plus directement à partir des documents de conception mais d'un modèle de l'architecture du logiciel embarqué. Nous faisons en ce sens une proposition méthodologique, qui, validée sur l'étude d'un composant logiciel automobile, améliore les pratiques actuelles en temps, en qualité des analyses et en réutilisation.**

***Abstract* – The advent of autonomous vehicles was long seen as an event that would revolutionize the world of transportation and improve the comfort and safety of users. Although this optimism is shared by most people and significant technological progress has been made in this direction, the road ahead offers many challenges, particularly in the area of safety, due to the increasing use of embedded software that has become very complex. Faced with this growing complexity, traditional manual, document-based methods of safety analysis have reached their limits and it is time to rethink approaches. We propose to rely on model-driven engineering to conduct safety analyses no longer directly from design documents but from a model of the embedded software architecture. To this goal, we make a methodological proposal, which, validated on the study of an automotive software component, improves current practices in time, analysis quality and reuse.**

***Mots clés* – ISO 26262, Logiciel Embarqué, MBSA, IDM, Sécurité**

***Keywords* –ISO 26262, Embedded Software, MBSA, MBSE, Safety**

1 INTRODUCTION

Avec l'avènement des véhicules autonomes, l'industrie automobile a de plus en plus recours aux logiciels embarqués pour assurer diverses fonctionnalités. Cela n'est pas sans conséquence pour la sécurité (au sens de safety, capacité du système à se prévenir de défaillances pouvant causer blessures et dommages) (DoD, 2012), qui doit être assurée avec rigueur. Être en mesure de fournir des garanties de la sécurité sera un facteur déterminant pour l'acceptation des véhicules autonomes par la société. Chez Renault, les pratiques en matière d'ingénierie logicielle sont bien encadrées par divers standards dont les principaux sont l'ASPICE (VDA QMC, 2015) qui définit l'utilisation et l'évaluation des processus d'ingénierie, et l'ISO 26262 (ISO 26262-1, 2018) qui adresse les aspects liés à la sécurité dans le développement du système et des logiciels. Le processus de développement suit un cycle en V classique, qui couvre l'élicitation des exigences, l'analyse, la conception d'architecture, jusqu'à l'implémentation, sur la partie dite descendante, ainsi que les tests et l'intégration, sur la partie remontante.

Conformément aux normes citées, la traçabilité verticale (des exigences vers les composants les implémentant) et horizontale (des tests vers les exigences) doivent être maintenues entre les artefacts produits aux différentes étapes du développement. Or, celles-ci reposent principalement sur des documents textuels, même s'ils sont supportés par des outils comme Excel ou DOORS. Avec la complexité croissante des logiciels, et l'augmentation du volume de documents associés, il n'est plus possible aujourd'hui de garantir la qualité, de maintenir la traçabilité et être en conformité avec les standards en utilisant les méthodes actuelles.

Par ailleurs, la norme ISO 26262 (ISO 26262-1, 2018) requiert déjà d'effectuer des analyses de sécurité et d'en apporter des preuves. Or, la complexité actuelle des applications embarquées dans un véhicule automobile rend extrêmement difficile d'assurer la safety. De plus, la complexité des architectures automobiles nécessite aujourd'hui un travail fortement collaboratif entre plusieurs équipes, constituées de spécialistes de domaines métiers différents, qui ont besoin d'utiliser de méthodes et de moyens efficaces pour travailler ensemble. En outre, il devient très difficile de maintenir à jour les analyses de sécurité par rapport aux évolutions, parfois rapides en cas de développement agile, des artefacts d'ingénierie.

Face à cette utilisation croissante de logiciel dans les véhicules, à la difficulté d'en évaluer et garantir la safety, et au vu du besoin accru d'échanger, au sein des équipes et entre équipes, des données d'ingénierie consolidées, simulables, vérifiables, il est aujourd'hui nécessaire de changer de paradigme, et de passer

des approches basées-documents à des méthodes basées-modèles pour les activités de développement de systèmes et de logiciels, ainsi que celles d'évaluation de leur sécurité. Ces approches favorisent la communication entre experts et équipes, et améliorent la capitalisation à travers la réutilisation. N'échappant pas à cette tendance, l'industrie des systèmes embarqués critiques, incluant l'automobile, considère également avec beaucoup d'intérêt ces approches basées modèles.

La proposition, dans cet article, est de nous appuyer sur l'ingénierie dirigée par les modèles pour définir une méthode pour l'évaluation de la safety des architectures logicielles en automobile, et de faire une proposition d'outillage. Les travaux présentés dans cet article résultent de notre retour d'expérience sur des études de cas menées dans l'entreprise Renault Software Labs dans l'objectif de déployer et de faire évaluer plus largement la méthodologie et son outillage par les experts de l'entreprise.

En section 2, l'article commence par explorer les méthodes basées sur l'ingénierie dirigée par les modèles (IDM) en conception système et logicielle, ainsi que pour l'évaluation de la safety. La Section 3 fait ensuite une proposition méthodologique visant à aider les experts safety à construire, à partir des artefacts d'ingénierie, des modèles que l'on appelle dysfonctionnels, à partir desquels mener des analyses de safety; celle-ci étend une première proposition, décrite dans (Sirgabsou et al., 2020). La Section 4 applique pas à pas la méthodologie sur l'une des études de cas, le contrôle longitudinal, une application embarquée d'aide à la gestion de la vitesse du véhicule compte tenu de la signalisation et des conditions de circulation. La Section 5 discute les résultats obtenus. L'article conclut et indique comment il serait intéressant de faire évoluer l'outillage de la méthodologie compte tenu des limitations observées, ainsi que des options alternatives d'outillage.

2 DEVELOPPEMENT DES APPROCHES BASEES MODELES

On observe depuis quelques temps cette évolution des pratiques industrielles vers des approches basées modèles en ingénierie système et logicielle ; on parle de MBSSE (Model Based System and Safety Engineering). Il s'agit d'un ensemble de pratiques fondées sur le concept de modèle conceptuel de domaine qui ont entre autres pour but d'automatiser la production de logiciels (NDIA, 2011). Le processus de conception peut alors être vu comme un ensemble ordonné de transformations de modèles, jusqu'à obtention d'artéfacts exploitables, ce qui favorise notamment réutilisation et les vérifications précoces. Ce processus consiste à tisser différents modèles entre eux (fonctionnel, organique, ou adressant différentes exigences extra-fonctionnelles à garantir, comme la sécurité, la fiabilité, la performance …). Plus récemment, et toujours à l'état de recherche, on considère l'adoption des approches basées modèles pour mener des analyses de safety, pour le moment limitées au niveau système ; on parle de MBSA (Model Based Safety Assessment).

2.1 Model Based System and Software Engineering

Avant d'être adoptées en ingénierie système, les approches basées modèles sont apparues en ingénierie du logiciel (Estefan, 2008). Elles permettent d'assurer une certaine continuité entre les différentes étapes du développement de systèmes et de logiciels (modélisation des exigences, de l'architecture logique, de l'architecture physique, jusqu'à l'implémentation), en assurant une traçabilité durant les transitions entre modèles. Elles reposent souvent sur l'utilisation de langages et d'outils.

Ainsi, les langage UML (Unified Modeling Language) et SysML (Systems Modeling Language) sont couramment utilisés pour modéliser les architectures fonctionnelles et organiques, et le comportement des logiciels et systèmes. Les diagrammes structuraux (diagramme de classe, de paquetage) sont par exemple utilisés pour modéliser l'architecture organique ; les diagrammes de cas d'utilisation, de séquence, d'activité, sont utilisés pour modéliser des scénarii comportementaux.

Il existe également des langages de description d'architecture, qui supportent spécifiquement l'étape de conception. Citons par exemple le langage AADL (Architecture Analysis and Description Langage), un standard de la SAE (Society of Automotive Engineers) initialement conçu pour l'avionique (Feiler et al., 2003), qui permet de concevoir et d'analyser l'architecture des systèmes embarqués. Certains langages de description d'architecture sont spécifiques à un domaine. En automobile, c'est par exemple le cas de EAST-ADL (Cuenot et al., 2007), un méta-modèle qui permet de modéliser l'environnement du système, le système lui-même à 5 différents niveaux de détails, et qui offre des extensions permettant de modéliser les propriétés du système à chaque niveau.

Toujours en conception, des sémantiques plus proches du code telles que celles des modèles SIMULINK et SCADE sont utilisées pour le prototypage rapide des architectures détaillées du logiciel et pour la génération de code. SCADE, beaucoup utilisé en avionique (Colaco et al., 2017), permet de modéliser des systèmes synchrones tels que ceux de commandes de vol ayant des contraintes de temps réel et ainsi d'anticiper certaines problématiques relatives à la safety propres aux systèmes concurrents ; il permet également l'automatisation des tests. Dans d'autres secteurs, comme c'est le cas pour l'automobile, ce sont les modèles Simulink qui sont de plus en plus utilisés.

Il apparait donc que différents modèles/langages sont utilisés à différentes étapes du cycle de développement d'un système ou d'un logiciel, selon le point de vue que l'on souhaite représenter. Une conception supportée par des modèles offre plusieurs avantages, tels qu'une meilleure communication, efficacité et réutilisation. La difficulté reste d'assurer une continuité entre ces différents artefacts de conception, par des transformations de modèles, dans un objectif de maintien de la cohérence globale. Une autre difficulté, qui devient majeure dans le contexte actuel de systèmes critiques de plus en plus autonomes, consiste à savoir tisser ces modèles de conception avec des modèles d'analyses spécifiques, comme les analyses de safety.

2.2 Model-Based Safety Assesment

Comme le MBSE, le MBSA résulte de la tendance en ingénierie de l'abandon des approches basées-documents au profit de celles basées-modèles. Il ne vient pas remplacer les modèles traditionnels sur lesquels reposent les analyses de la safety (telles que les arbres de fautes (Vesely et al., 1981), coupes minimales (Center for Chemical Process Safety, 2010) et les AMDE(C) (Analyse des Modes de Défaillance et de leurs Effets (Critiques) (SAE, 2015), mais vise plutôt à établir des ponts entre ces modèles et les modèles d'ingénierie (résultant du MBSE), favorisant ainsi une meilleure cohérence et traçabilité entre modèles, ainsi qu'une justesse des analyses. Une autre motivation du MBSA est également d'unifier les modèles classiques d'analyses de la safety, considérée au sens large (incluant fiabilité et maintenabilité) dans un seul modèle dysfonctionnel (Lisagor et al., 2011).

Plusieurs approches de MBSA existent, telles que le FPTN (Fenelon & McDermid, 1993), HiP-HOPS (Papadopoulos &

McDermid, 1999), FSAP/NuSVM (Bozzano & Villafiorita, 2007) ou encore AltaRica (Point & Rauzy, 1999). Sans rentrer dans les détails, ces approches se distinguent principalement par la façon dont le modèle dysfonctionnel est construit : celui-ci peut être étendu à partir du modèle fonctionnel (dit nominal) issu de la conception, ou être distinct (Lisagor et al., 2011). Dans le cas d'un modèle étendu, le modèle du système sera enrichi par des informations de défaillance ; un modèle dédié sera construit manuellement par l'expert safety.

Les approches offrent chacune des avantages et des inconvénients. Utiliser un modèle étendu permet de garantir naturellement la cohérence entre le modèle du système et le modèle de safety dérivé, sans avoir besoin de recourir à des mécanismes supplémentaires. De plus, le concepteur et l'expert safety peuvent utiliser le même environnement et outil de modélisation. Cependant, si l'idéal est d'avoir ainsi un seul modèle capable d'intégrer les éléments fonctionnels et dysfonctionnels, suffisamment structuré pour en déduire les analyses de safety, en pratique, les sémantiques des modèles de conception sont souvent encore peu formelles et peu structurées. Une alternative satisfaisante consiste alors dans ce cas à utiliser un modèle dédié. L'un des intérêts de cette approche est qu'elle permet d'avoir une indépendance entre les activités de conception et d'analyse de la safety, ce qui peut constituer un atout important, par exemple dans un contexte de certification. Cependant, utiliser un modèle dédié pose la question du temps conséquent nécessaire pour construire ce modèle. Plusieurs travaux portent sur des passerelles de construction d'un modèle MBSA à partir d'un modèle MBSE (David et al., 2009), (Nguyen et al., 2018) et souligne l'effort à faire pour établir celles-ci. D'autres travaux tels que ceux de (Batteux et al., 2019) au (Legendre et al., 2016) s'intéressent au maintien de la cohérence des modèles MBSE et MBSA à travers la synchronisation.

Parmi les approches mentionnées plus tôt, nous nous intéressons à AltaRica (Point & Rauzy, 1999), qui aujourd'hui est une approche pionnière dans l'écosystème des systèmes critiques Européen. AltaRica est un langage de haut niveau conçu pour la modélisation des systèmes. Un modèle AltaRica décrit un système à travers des domaines, des variables et une hiérarchie de nœuds où chaque composant peut incorporer plusieurs sous-nœuds. Un domaine décrit un ensemble de valeurs booléenne, entières, énumérées ou abstraites, qu'une variable peut prendre. Les nœuds décrivent les comportements des composants du système à travers des variables d'état, des événements, des transitions et des assertions.

Les transitions décrivent les changements d'état. Un nœud interagit avec l'environnement de deux manières : 1) à travers des événements et 2) à travers des variables de flux. À l'exemple d'une fonction de transfert, les assertions décrivent les relations entre les flux d'entrée, les variables d'état et les flux de sortie. L'intérêt d'AltaRica, réside surtout dans sa sémantique à la fois formelle et proche des systèmes qu'elle décrit. Cette dualité, difficile à obtenir avec les autres approches, justifie le choix d'AltaRica pour notre démarche.

3 PROPOSITION METHODOLOGIQUE

Notre ambition et l'originalité de notre contribution est de proposer une méthodologie adaptant les concepts, principes et méthodes du MBSA, généralement appliqués au niveau d'un système, afin d'améliorer les pratiques d'analyse de la sécurité des logiciels. Cette proposition est notamment importante dans le contexte des logiciels embarqués en automobile, qui sont critiques du point de vue de la sécurité.

Parmi les approches possibles, nous avons choisi dans cet article d'explorer celle d'un modèle dysfonctionnel dédié, basé sur une méthode de modélisation des logiques de propagation de défaillance et utilisant le langage AltaRica.

Comme spécifié plus tôt, cette méthodologie étend et adapte une première proposition, décrite dans (Sirgabsou et al., 2020), qui permettait de construire un modèle dysfonctionnel de l'architecture logicielle et de l'utiliser comme base pour conduire les analyses safety en trois étapes, modélisation dysfonctionnelle, traduction des logiques et analyse de safety. La nouvelle proposition introduit une étape préliminaire de choix des données d'entrée, qui adresse spécifiquement la difficulté d'avoir de bonnes entrées pour la construction du modèle, La méthodologie procède donc en trois nouvelles étapes, comme illustré en Figure 1: 1) le choix des données d'entrée à partir desquelles sera construit le modèle dysfonctionnel, 2) la construction du modèle dysfonctionnel du logiciel, à partir duquel 3) les coupes minimales, les arbres de fautes et les AMDE pourront être générés automatiquement.

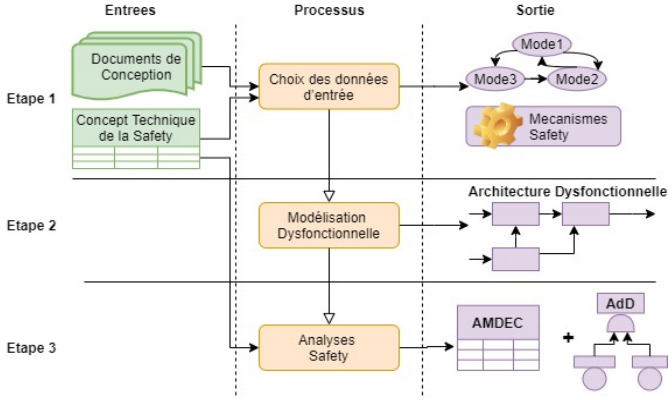

Figure 1. Les 3 grandes étapes de la méthode

3.1 Étape 1 : Choix des données d'entrée

Ayant opté pour une approche reposant sur un modèle dédié, nous devons tout d'abord identifier les informations nécessaires pour sa construction : nous partons des documents de conception (architecture) et d'un 'Technical Safety Concept' (TSC). Le TSC, défini dans l'ISO 26262 (ISO 26262-1, 2018), est un document qui contient les spécifications des exigences de safety issues du système tels que les safety goals (exigences de safety de haut niveau résultant de l'analyse préliminaire des risques au niveau véhicule) (cf. ISO 26262-1 3.139), leur degré de criticité ou ASIL (Automotive Safety Integrity Level) ainsi que leurs allocations aux composants matériels ou logiciels.

Le TSC contient aussi des informations qui précisent quelles mesures (actions préventives) doivent être prises durant le développement et quels mécanismes de sécurité (actions curatives) doivent être implémentés dans l'architecture. Ainsi, grâce au TSC nous pouvons identifier les composants et interfaces à modéliser ainsi que les exigences et mécanismes à évaluer dans le cadre de l'architecture dysfonctionnelle, et ainsi éviter de surcharger le modèle dysfonctionnel avec des éléments inutiles à la safety.

A partir des documents d'architecture, nous identifions le périmètre du système, ses interactions, ses modes de fonctionnement (états, normaux ou anormaux) et les conditions de changement d'état de ses composants ; nous pouvons ainsi avoir une compréhension de son fonctionnement. La démarche suivie à l'étape 1 est illustrée sur la Figure 2

Cette compréhension peut être complétée, si nécessaire, en consultant le concepteur (architecte logiciel) ou encore en exploitant des modèles de design plus détaillés s'ils existent. Par exemple, des modèles fonctionnels Simulink peuvent être

exploités pour une compréhension de la logique fonctionnelle et pour déterminer comment les mécanismes de safety sont implémentés.

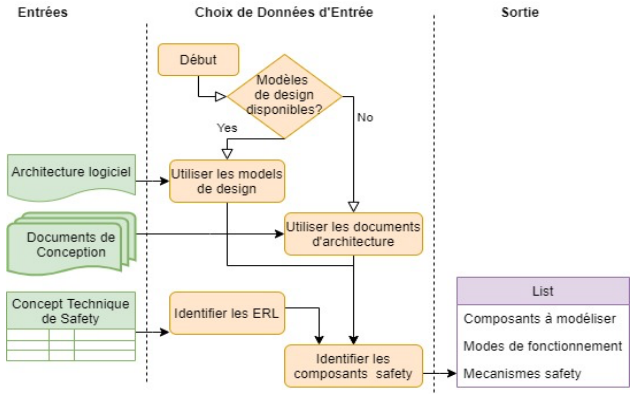

Figure 2. Étape 1, Choix des données d'entrée

3.2 Étape 2 : Modélisation dysfonctionnelle

Elle consiste en la modélisation du comportement dysfonctionnel (en présence de défaillance) des composants logiciels ainsi que leurs interactions. Comme illustrée sur la Figure 3, elle se déroule en 2 sous-étapes : 1) la modélisation par des machines à états du comportement des composants, et 2) la modélisation des propagations des défaillances. Nous les décrirons dans les 2 sous-sections suivantes.

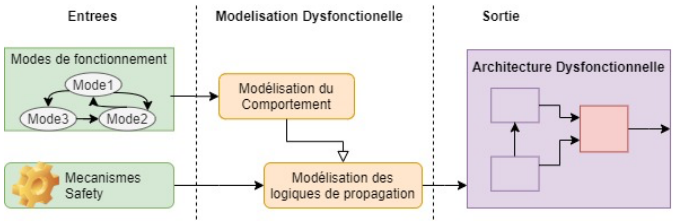

Figure 3. Étape 2, Modélisation dysfonctionnelle

3.2.1 Modélisation du comportement des composants

Elle consiste en la modélisation du comportement dysfonctionnel interne des composants du système grâce aux des données collectées à l'étape 1 (modes de fonctionnement, mécanismes de safety). Elle se focalise sur les composants et signaux en lien avec la safety. Nous modélisons le comportement interne des composants grâce à des machines à états (états et transitions). Les états ne sont rien d'autre que les modes de fonctionnement (que nous avions identifiés), qu'ils soient nominaux ou défaillants (ON, OFF, défaillant, dégradé...). Au travers des transitions, nous exprimons les conditions logiques conditionnant le passage d'un état à un autre. Ces conditions peuvent être liées à des événements aléatoires et guidées par une loi de probabilité (stochastique), ou dépendantes de l'état des entrées (déterministes).
De cette modélisation, il résulte des composants logiciels ramenés à des machines d'états.

3.2.2 Modélisation des interfaces

La modélisation des interfaces complète celle du comportement des composants avec des logiques de propagation de défaillance. Il s'agit de modéliser les interactions dysfonctionnelles entre les composants (comment les défaillances se propagent d'un composant à un autre).
Dans un premier temps, nous modélisons les signaux d'entrée et de sortie de chaque composant en leur attribuant des états discrets représentant les classes de valeurs qu'ils ne peuvent pas prendre. Par exemple, d'un point de vue dysfonctionnel, une donnée d'entrée peut être correcte, erronée, en retard ou absente (voir l'annexe E de la partie 6 de ISO 26262, qui liste les erreurs possibles relatives à l'exécution et à l'échange d'information entre composants logiciels).
Dans un deuxième temps, nous écrivons des logiques nous permettant de déduire les états de sortie en fonction de l'état interne du composant et de ses entrées. Par exemple, prenons une simple fonction ADD qui additionne deux variables (entree1 et entree2). Pour cette fonction nous définissons une variable d'état 'state' pouvant prendre trois valeurs : ok, erronée ou perdue. Pour les 2 entrées et la sortie, nous définissons également 3 états : correcte, erronée, absente). Nous pouvons alors écrire l'expression de la sortie en fonction des entrées et de l'état de la fonction :

```
if (state = ok) then
        if ((entree1 = correcte) and (entree2 = correcte)) then
            sortie : = correcte
        Else if ((entree1 = absente) or (entree2 = absente)) then
        sortie : = absent
        else
            sortie: = erronée
        end if
else //state not ok
        sortie : = erronée
end if
```

Il existe aujourd'hui des formalismes supportant une telle approche, comme AltaRica ou celui de l'annexe d'erreur d'AADL.
En réitérant cette démarche pour tous les composants, nous aboutissons à une architecture dysfonctionnelle bien définie, dans une sémantique formelle ; ceci constitue une base nécessaire pour la génération des analyses de safety.

3.3 Étape 3 : Analyses de Safety

Le but de l'étape 3 est de dériver des modèles de safety classiques à partir du modèle dysfonctionnel construit. Pour cela, il est avant tout nécessaire d'ajouter des Événements Redoutés Logiciels (ERL) au modèle dysfonctionnel. Pour ce faire, il est nécessaire d'exprimer les ERL par des combinaisons booléennes et de les associer aux composants concernés. Après cela, il sera alors possible d'utiliser le modèle résultant pour produire les AMDE et arbres de fautes pour faire des analyses grâce à la vérification de modèle. Cela est rendu possible au travers de l'ensemble du comportement dysfonctionnel capturé grâce aux modes de défaillances individuels ainsi qu'aux événements associés modélisés au niveau de chaque composant. De plus, l'ajout des événement redoutés à l'architecture permet, au travers de la logique de propagation modélisée, de les associer ou non à certains composants. Ainsi plusieurs ERL peuvent être ajoutés, évalués sur la base de la même architecture dysfonctionnelle.

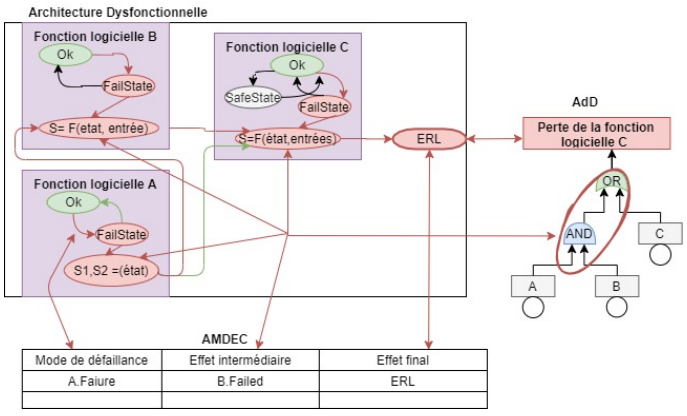

Figure 4. Étape 3 : Génération des AMDE et arbres des fautes à partir de l'architecture dysfonctionnelle

Une illustration faisant apparaitre les correspondances entres les éléments du modèle dysfonctionnel et ceux des AMDE et arbre des fautes est présentée sur la Figure 4. Sur cette base, on peut donc générer, à partir de l'architecture dysfonctionnelle, des arbres des fautes ayant comme sommet l'un des événements redoutés modélisés, et dont les branches découlent (en fonction de la logique de propagation) des transitions vers des états défaillants tels que modélisés individuellement par les machines à états au niveau des composants.
Cette même logique peut s'appliquer aux AMDE, où les transitions vers des états défaillants deviendront des modes de défaillances, alors que les événements redoutés deviendront des effets finaux.

4 Étude de Cas

Nous proposons d'outiller la méthode décrite précédemment avec le logiciel SimfiaNeo et de l'appliquer sur une étude de cas, l'analyse d'un composant logiciel assurant le contrôle longitudinal d'un véhicule. Elle est complémentaire à l'étude sur le contrôle latéral, décrite dans (Sirgabsou et al., 2020). L'objectif est d'évaluer l'architecture du logiciel, grâce à un modèle dysfonctionnel, pour vérifier si elle satisfait un certain nombre d'exigences de safety. Nous évaluerons par la même occasion les performances de l'outil SimfiaNeo. Le périmètre de la nouvelle étude inclut un safety concept incorporant des modèles informels expliquant les mécanismes safety proposés. Ceci nous permettra de simplifier le processus de modélisation de l'architecture dysfonctionnelle.

4.1 Présentation de SimfiaNeo

SimfiaNeo est un outil de MBSA basé sur le langage AltaRica développé par APSYS-Airbus. Il offre une interface de modélisation graphique basé sur Eclipse qui est plus conviviale comparée aux autres éditeurs AltaRica présents sur le marché tels que OCAS, Open AltaRica ou AltaRica Studio. SimfiaNeo permet de construire graphiquement le modèle AltaRica d'un système et de générer directement des coupes, des arbres de fautes et des AMDE à partir du modèle AltaRica. Il est donc compatible avec notre démarche. Pour notre étude nous utiliserons la version 2.0.

4.2 Présentation du système étudié

Le contrôle longitudinal est une fonction du système ADAS (Advanced Driver Assistance Systems) d'aide à la conduite. Concrètement, il s'agit d'un composant logiciel dont le but est d'assurer le contrôle de vitesse et du freinage en mode de conduite autonome. Il est construit autour de l'ACC (Adaptive Cruise Control), un système de régulation de la vitesse et de la distance. L'ACC calcule la vitesse que peut avoir le véhicule tout en restant en sécurité par rapport à certains événements prédéfinis (virages, bouchons, stop…). Un certain nombre d'exigences de safety lui sont associées. Parmi elles figurent celles liés aux ERL, telles que le freinage intempestif et l'accélération non contrôlée émanant de l'ACC. Il existe également d'autres contraintes ; par exemple, l'utilisateur doit toujours pouvoir désactiver l'ACC à tout moment.
Nous allons modéliser ce système en ayant à l'esprit ces exigences, tout en accordant une attention particulière aux mécanismes de safety proposés dans l'architecture du logiciel pour leur implémentation (limites de l'accélération, conditions d'activation de l'ACC). Ainsi, pour modéliser l'ACC, il faut non seulement en considérer les éléments internes mais aussi les autres éléments avec qui l'ACC est en interaction. Cela nous permettra de délimiter le périmètre du système et favorisera une meilleure interprétation des interfaces. Dans la section suivante, nous identifierons les composants en lien avec le contrôleur longitudinal en partant du TSC et de la documentation générale de l'ADAS.

4.3 Application des trois étapes de la méthodologie

4.3.1 Choix de données d'entrée

Ayant opté dans notre démarche pour la construction d'un modèle dysfonctionnel manuellement, il fallait tout d'abord recueillir un ensemble d'informations de conception. Nous avons donc consulté les livrables du projet : documents de conception, planches des présentations officielles, tableaux Excel des exigences, safety concept. Devant le manque de clarté dans l'expression des modes de haut niveau, nous avons dû compléter notre compréhension du système en interprétant les modèles Simulink de l'ACC. Grâce aux éléments contenus dans le safety concept, nous avons pu identifier les safety goals, les composants (et sous-composants) à modéliser, ainsi que les signaux d'entrée associés à prendre en considération. De même, à partir du document de conception, nous avons identifié les modes de fonctionnement des composants ainsi que leurs conditions d'activation. Nous avons ainsi déterminé les domaines nécessaires à la modélisation du comportement des composants. Après avoir étudié les composants, nous avons créé 5 domaines particuliers avec différents états. Grâce à ces domaines, nous avons constitué des machines à états, à partir desquelles nous allons pouvoir modéliser le comportement dysfonctionnel des composants dans la partie suivante.

4.3.2 Modélisation

Le but est de modéliser le comportement dysfonctionnel des composants prenant part dans la fonction de l'ACC. Pour ce faire, dans l'outil SimfiaNeo, nous avions commencé par déclarer les domaines AltaRica (ensemble des états possibles à attribuer aux composants et aux interfaces) en nous appuyant sur les modes de fonctionnement retenu à l'étape précédente. Ainsi, nous avons créé les domaines à la fois pour représenter l'état des composant mais aussi ceux de leurs entrées et sorties. Ces informations nous permettent de modéliser facilement le comportement et les interfaces de chaque composant.

Modélisation du comportement des composant

Pour modéliser les composants et leurs états dans l'outil, nous utilisons les briques de modèle disponibles, puis nous leur attribuons les domaines préalablement créés. Il faut ensuite modéliser les conditions de franchissement des transitions en utilisant les équations identifiées dans les machines d'états. Cela se fait dans SimfiaNeo à travers la création des événements AltaRica. Un événement AltaRica est caractérisé par une garde (condition à remplir avant déclenchant du changement d'état), un effet (action résultant du changement d'état), et une loi (exponentielle, Dirac etc.). Cependant, si l'utilisation de taux de défaillances dans la modélisation des transitions est bien maîtrisée dans le cas des composants matériels, ce n'est pas le cas pour les logiciels qui sont de nature déterministe et pour lesquels il n'existe pas aujourd'hui de modèle de défaillance probabiliste assez mature. Ainsi notre démarche se différencie d'une utilisation classique d'AltaRica par la manière dont les gardes des transitions sont construites ; au lieu de nous limiter à une modélisation des transitions par les lois de probabilistes citées ci haut, nous utilisons aussi des combinaisons des états d'entrée. Traditionnellement, en AltaRica, les gardes ne font apparaître que l'état dans lequel le composant doit être pour que la transition soit franchissable ; l'événement est alors déclenché suivant la valeur d'une loi de probabilité spécifiée. Cependant, en associant les entrées aux gardes, nous pouvons aller au-delà de ce paradigme et faire en sorte que les gardes puissent

dépendre non seulement des lois de probabilité (si applicable) mais également des entrées. Cela nous permet d'avoir un changement d'état déterministe, qui ne dépend plus non seulement des lois de probabilité mais aussi de l'état des entrées. L'approche convient donc bien pour la modélisation des défaillances déterministes comme ceux des logiciels.

Modélisation des interfaces

Pour chacun des composants, nous avons écrit des logiques de propagation des défaillances liant les entrées et les sorties correspondantes. Pour ce faire, nous avions étudié des fichiers Simulink qui précisent les logiques fonctionnelles implémentées ; nous avons également trié et choisi les éléments à prendre en considération car toutes les entrées et sorties n'étaient pas utiles pour notre modèle dysfonctionnel, car n'affectant pas les mécanismes safety associés. Ensuite, nous avons décrit les états des entrées et sortie en utilisant le domaine AltaRica que nous avons nommé « Generic Data » et qui incorpore 4 états : Ok (la donnée est bonne), Loss (la donnée est perdue), Delay (la donnée arrive tardivement), Out of Range (la valeur est hors des limites acceptable). Grâce à ces états, nous étions alors en mesure de modéliser les flux d'information dysfonctionnelle entre les composants.

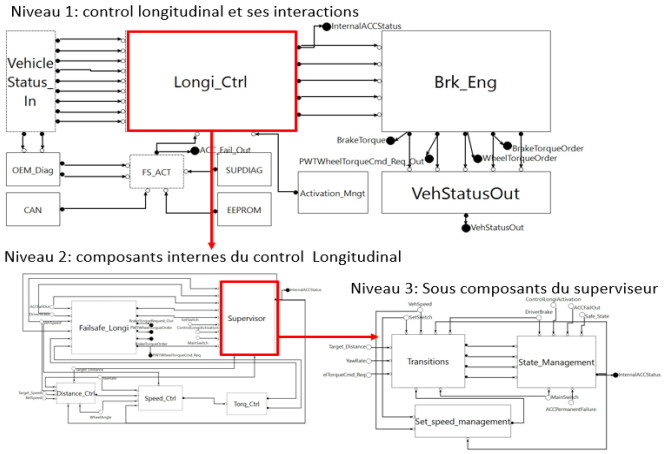

Figure 5. Trois niveaux dans la modélisation

Du fait du niveau de détail auquel il était nécessaire de descendre pour modéliser le comportement du système, nous avons obtenu un modèle en 3 niveaux tel qu'illustré en Figure 5. Ainsi, si nous nous focalisons sur l'ACC, le niveau 1 représente le contrôleur longitudinal et les composants entrant en interaction avec lui ; le niveau 2, les composants internes du contrôleur longitudinal et le niveau 3, les sous-composants des éléments du contrôleur longitudinal.

safety à partir du modèle dysfonctionnel. Pour cela, nous avons mis en place des observateurs AltaRica sur les sorties qui nous intéressaient. Un observateur AltaRica est un indicateur qui peut être associé à une défaillance que l'on souhaite observer. En guise d'exemple, intéressons-nous à l'envoi d'une commande de gestion du moteur (PWTWheelTorqueCmd). Il s'agit d'un scénario qui peut affecter plusieurs safety goals. Il peut conduire à : 1) une accélération ou un freinage moteur intempestifs, 2) l'application d'une commande moteur alors que le statut de l'ACC ou celui du PWTWheelTorqueOrder ne le permettent pas et 3) l'application d'une commande conflictuelle avec la commande de freinage. L'objectif va donc être de vérifier, grâce à la simulation et à partir des AMDE, coupes minimales et arbres de fautes, si l'on peut déterminer les événements ou défaillances pouvant mener à l'envoi de cette commande erronée.

4.4.1 Analyse des Modes de Défaillance et leurs Effets

Nous avons ensuite utilisé SimfiaNeo pour générer des tables d'AMDE à partir du modèle dysfonctionnel construit. Les AMDE permettent de lister tous les événements provoquant la violation d'un safety goal (ou d'un observateur créé), et cela pour chaque composant du modèle. Ainsi, on retrouve en Figure 6 un certain nombre d'éléments de base d'une AMDE.

En première colonne (Scenario) figurent les événements provoquant la violation. Dans les colonnes suivantes, nous retrouvons : Local Effet (effet de l'événement sur la sortie du composant initial), Intermediate Effect (effet de l'événement sur tous les composants intermédiaires entre le composant initial et l'observateur final) et enfin Final Effect (effet sur la sortie du modèle - ici il s'agit de l'effet sur certains des observateurs décrits dans les parties précédentes).

SimfiaNeo permet d'exporter ce document sous la forme d'un tableur Excel, permettant de mieux traiter les données et de les partager avec d'autres personnes. Cela constitue un atout de l'outil, quand on sait que les outils MBSA ne sont pas forcément utilisés par un grand nombre. Cependant la Figure 6 ne représente qu'un petit extrait de l'AMDE initiale, qui présente plus de 18000 lignes. De ce fait, si le but est de d'avoir des détails exploitables ou de faire une synthèse, cette représentation n'est certes pas idéale.

4.4.2 Coupes minimales

La génération des coupes minimales passe avant tout par la configuration et la génération des séquences de calcul qui proposent plusieurs possibilités pour un même observateur. Ainsi il est possible de choisir l'ordre de la coupe, correspondant au nombre maximal de combinaisons possibles entre

Scenario	Local effect	Local effect value	Intermediate effect	Intermediate effect value	Final effect	Final effect value
SELECT_ALL	SELECT_ALL	SELECT_ALL	SELECT_ALL	SELECT_ALL	SELECT_ALL	SELECT_ALL
	VehicleStatus_In.YawRate.YawRate	Loss	VehicleStatus_In.YawRate.YawRate	Loss	InternalACCStatus	Loss
			VehicleStatus_In.YawRate	Loss	VehStatusOut	Loss
					BrakeTorqueOrder	Loss
					PWTWheelTorqueCmd_Req_Out	Loss
					WheelTorqueOrder	Loss
					BrakeTorque	Loss
Longi_Ctrl.Supervisor.State_Management.failure	VehicleStatus_In.YawRate.YawRate	Loss	VehicleStatus_In.YawRate.YawRate	Loss	InternalACCStatus	Loss
			VehicleStatus_In.YawRate	Loss	VehStatusOut	Loss
					BrakeTorqueOrder	Loss
					PWTWheelTorqueCmd_Req_Out	Loss
					WheelTorqueOrder	Loss
					BrakeTorque	Loss
Longi_Ctrl.Supervisor.State_Management.failure	VehicleStatus_In.YawRate.YawRate	Loss	VehicleStatus_In.YawRate.YawRate	Loss	InternalACCStatus	Loss
Longi_Ctrl.Supervisor.State_Management.failure			VehicleStatus_In.YawRate	Loss	VehStatusOut	Loss
					BrakeTorqueOrder	Loss

Figure 6. AMDE

4.4 Analyses de safety

Après la modélisation complète de l'ACC et des composants qui interagissent avec lui, l'objectif était de faire des analyses de

événements élémentaires amenant à l'événement redouté. Pour notre étude de cas, nous avons choisi l'ordre 3 (choix lié à la performance de l'outil), car plus l'ordre est grand, plus la génération de la coupe prendra du temps et plus elle nécessitera

une puissance élevée. Un ordre de 3 constitue donc un bon compromis entre temps de calcul et précision. Le choix du type de filtre (coupes minimales ou séquences minimales) aussi est important. Tandis que les séquences donnent les combinaisons d'événements dans un ordre précis, les coupes minimales renvoient les plus petites combinaisons possibles pouvant mener à l'événement redouté. Nous avons choisi l'option « coupes minimale » car grâce à elle il sera ensuite possible de générer des arbres de fautes.

PWT_Torque_Order (erroneous)

	Elements	Order	Probability
1	Longi_Ctrl.Failsafe_Longi.SafeState.TempFailure	1	1.0E-5
2	Longi_Ctrl.Torq_Ctrl.Brake_Torque.error	1	1.0E-5
3	Activation_Mngt.error	1	1.0E-5
4	FS_ACT.ACC_OutputGen.error	1	1.0E-5
5	Longi_Ctrl.Distance_Ctrl.Rate_Limiters.error & Longi_Ctrl.Torq_Ctrl.DynamicSaturation.error	2	1.0E-10
6	Longi_Ctrl.Torq_Ctrl.DynamicSaturation.error & VehicleStatus_In.V_kph_VehicleSpeed_Est_comp.error	2	1.0E-10
7	Longi_Ctrl.Torq_Ctrl.DynamicSaturation.error & Longi_Ctrl.Speed_Ctrl.FFD_Loop_FBK_Loop_Corrections.error	2	1.0E-10
8	Longi_Ctrl.Torq_Ctrl.DynamicSaturation.error & Longi_Ctrl.Speed_Ctrl.Internal_Model_Accelerator_Ctrl.error	2	1.0E-10
9	Longi_Ctrl.Torq_Ctrl.DynamicSaturation.error & VehicleStatus_In.Target_Distance.error	2	1.0E-10
10	Longi_Ctrl.Torq_Ctrl.DynamicSaturation.error & Longi_Ctrl.Torq_Ctrl.Preprocessing.error	2	1.0E-10
11	Longi_Ctrl.Torq_Ctrl.DynamicSaturation.error & VehicleStatus_In.RelSpeed.error	2	1.0E-10
12	Longi_Ctrl.Torq_Ctrl.DynamicSaturation.error & VehicleStatus_In.Target_Speed.error	2	1.0E-10
13	Longi_Ctrl.Torq_Ctrl.DynamicSaturation.error & VehicleStatus_In.WheelAngle.error	2	1.0E-10
14	Longi_Ctrl.Distance_Ctrl.Feedback_Module.error & Longi_Ctrl.Torq_Ctrl.DynamicSaturation.error	2	1.0E-10
15	Longi_Ctrl.Distance_Ctrl.Vstop_Vmin_Strategies.error & Longi_Ctrl.Torq_Ctrl.DynamicSaturation.error	2	1.0E-10
16	Longi_Ctrl.Torq_Ctrl.DynamicSaturation.error & VehicleStatus_In.YawRate.error	2	1.0E-10
17	Longi_Ctrl.Distance_Ctrl.Cut_In_Cut_Out.error & Longi_Ctrl.Distance_Ctrl.Following_Module.error & Longi_C...	3	1.0E-15

Figure 7. Coupe minimale

A titre d'exemple, une coupe minimale pour l'ERL « commande erronée de la gestion du moteur » est présentée en Figure 7. Elle montre des combinaisons (d'ordre 1, 2, 3) d'événements de base pouvant causer l'événement redouté spécifié (PWT_Torque_Order(erroneous)) ainsi que des probabilités associées.

Par ailleurs, on peut voir que la coupe met en avant les événements et les composants hiérarchiques, permettant d'avoir la traçabilité des composants à haut niveau comme montré en Figure 7. Pour des modèles dysfonctionnels ayant plusieurs sous-composants à nomenclature identiques, cela permet d'identifier clairement d'où provient chaque événement.

Cependant, plusieurs erreurs de compilation sont survenues, dont certaines du fait de la présence de boucles dans la chaîne de propagation des défaillances. Pour résoudre ce problème, nous avons modifié les assertions des observateurs.

La large palette de choix qu'offre SimfiaNeo à travers différentes options de configuration permet d'affiner les calculs et les simulations en fonction du besoin du projet.

4.4.3 Arbres de fautes à partir des coupes minimales

Nous avons pu générer des arbres de fautes à partir des coupes minimales présentées plus tôt. Comme le montre la Figure 8, les arbres de fautes permettent de voir, à travers une arborescence graphique, la chaîne de causalité entre les événements de base au niveau des composants et l'événement redouté de haut niveau.

En modifiant les assertions des observateurs, il est possible de combiner les événements pour chercher à reproduire des événements redoutés liants plusieurs composants. La Figure 8 représente les événements pouvant amener à un ordre de gestion du moteur **erroné** et pourtant à une commande du moteur **Ok**. Ainsi, en fonction de l'analyse safety nécessaire, SimfiaNeo et AltaRica permettent de développer des assertions pour représenter des situations très diverses.

Toutefois il est parfois difficile de traduire certaines exigence safety (comme celles ayant des contraintes de temps) en expression AltaRica. Il s'agit d'un problème spécifique de la version 2 d'AltaRica implantée dans l'outil SimfiaNeo. Il s'agit donc là d'une limite de l'approche, cependant liée à la version utilisée du langage.

5 DISCUSSION DES RESULTATS

L'application de la proposition méthodologique à l'étude de cas montre qu'il est possible d'appliquer au logiciel une approche de type MBSA, jusqu'à présent orientée système. A travers un bon choix des données d'entrée et un effort de modélisation, notre approche permet d'obtenir un modèle dysfonctionnel représentatif du système réel, ce qui améliore la qualité des analyses de safety. Grâce à cette démarche, il est donc possible d'unifier différents modèles de safety classiques dans un modèle dysfonctionnel unique. De plus la démarche offre un énorme atout pour la réutilisation. En effet, pour évaluer de nouveaux ERL, il suffira de les décomposer en des combinaisons logiques et de les associer à l'architecture dysfonctionnelle déjà modélisée. Une fois le modèle dysfonctionnel construit, celui-ci permettra de mener des analyses avec des paramètres différents et ce pour un grand nombre d'ERL car il suffira d'exprimer ces derniers en utilisant des combinaisons booléennes appropriées.

La proposition méthodologique améliore ainsi les pratiques actuelles. Grâce à l'approche décrite et à l'utilisation d'outils comme SimfiaNeo, il est possible de déplacer la tâche de l'expert safety, de la construction manuelle des AMDE et arbres de fautes, à la construction d'une architecture dysfonctionnelle. En focalisant son énergie sur la modélisation dysfonctionnelle, la méthode rapproche les modèles safety le plus proche possible des modèles du design, ce qui améliore de ce fait la qualité des analyses.

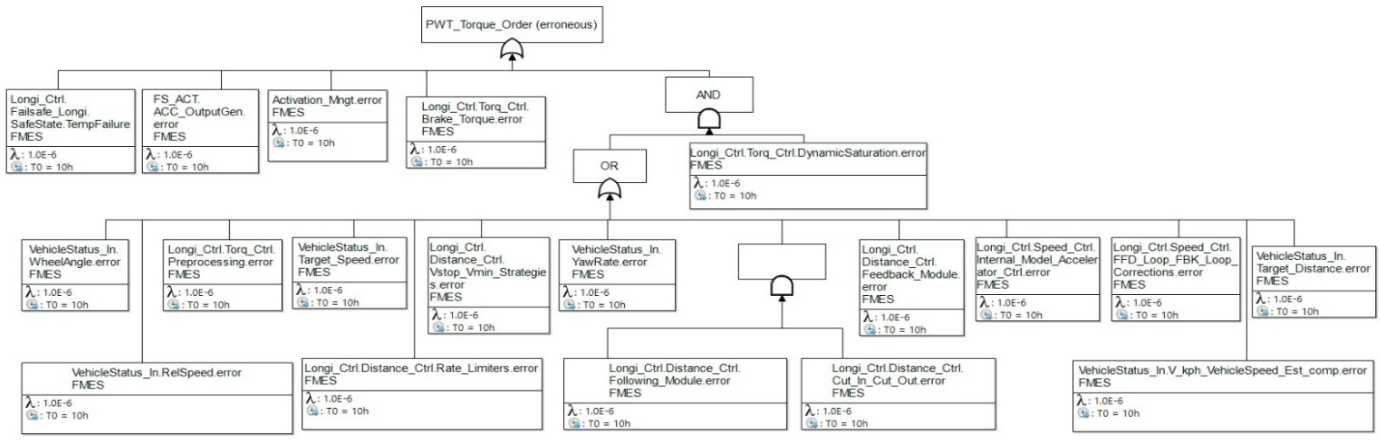

Figure 8. Arbre de faute

Ainsi, le gain se mesure en temps, en qualité des analyses et en possibilités de réutilisation des modèles.

En outre, l'utilisation de la sémantique formelle d'AltaRica permet de représenter le comportement du système sans

ambigüité, qui en fait un candidat possible dans un contexte de certification.

L'ensemble de ces éléments permettent d'améliorer significativement les pratiques actuelles en matière d'analyse de la safety. Cependant, l'une des difficultés avec un modèle dysfonctionnel dédié, est la problématique du maintien en cohérence de celui-ci avec le modèle de conception lorsque ce dernier évolue. Il est donc nécessaire de mettre en place des mesures supplémentaires pour garantir la cohérence. Ainsi nous avons relevé certaines limitations durant l'étude de cas. La modélisation peut prendre du temps ; il faut se placer au juste niveau de détail dans la représentation du système. Nous proposons de clairement définir les éléments d'entrée (composants, signaux, mécanismes safety) à étudier à l'étape 1. Grace à cette étape préliminaire et en nous basant sur les informations du TSC (mécanismes safety, criticités), nous sommes en mesure de nous assurer que les éléments liés à la safety ne sont pas ignorés Une autre limitation constatée est relative à l'incapacité de AltaRica 2.0 à gérer les boucles ; notre solution a été de modifier certaines assertions de manière à supprimer ces boucles. Une dernière limitation, relative aux performances de SimfiaNeo, est son besoin important en ressources (mémoire et processeur) ; ainsi, l'un des problèmes rencontrés a été la perte inexpliquée d'une partie du fichier source de notre modèle et nous avions dû le reconstruire manuellement. C'est ce qui explique notre choix de nous limiter à l'ordre 3 pour les coupes. Cela soulève des questions de fiabilité de l'outil. Enfin, l'étude de cas concernant un système de complexité réduite, le passage à l'échelle reste à évaluer.

6 Conclusion

Nous avons fait une proposition méthodologique reposant sur l'ingénierie dirigée par les modèles qui permet, en trois étapes, de construire un modèle dysfonctionnel à partir duquel il est possible de dériver des modèles classiques safety.

Grâce à l'outil SimfiaNeo et en utilisant le langage AltaRica, nous avons appliqué la méthodologie sur une étude de cas, en construisant un modèle dysfonctionnel d'un logiciel à partir duquel nous avons pu générer des AMDE, coupes minimales et arbres de fautes. Ces résultats sont encourageants. Ils montrent qu'il est possible d'appliquer une approche MBSA pour évaluer la safety des logiciels, notamment en automobile. Ils démontrent également les bénéfices d'une génération des analyses de safety à partir d'un modèle dysfonctionnel (gain en temps, qualité des analyses, réutilisation).

Forts de ces résultats, l'étude doit maintenant se poursuivre, afin d'évaluer la complexité des systèmes pour lesquels la méthode et l'outillage peuvent être raisonnablement appliqués. De plus, dans la mesure où la proposition de cet article s'appuie sur un modèle dysfonctionnel dédié, il faudra également adjoindre à la méthode un mécanisme de mise en cohérence entre les modèles de conception et les modèles de safety.

7 References

Batteux, M. B., Prosvirnova, T., & Rauzy, A. (2019, October). Model synchronization: A formal framework for the management of heterogeneous models. International Symposium on Model Based Safety Assessment, IMBSA 2019. https://doi.org/10.1007/978-3-030-32872-6_11

Bozzano, M., & Villafiorita, A. (2007). The FSAP/NuSMV-SA Safety Analysis Platform. International Journal on Software Tools for Technology Transfer, 9(1), 5–24. https://doi.org/10.1007/s10009-006-0001-2

Center for Chemical Process Safety. (2010). Appendix D: Minimal Cut Set Analysis. In Guidelines for Chemical Process Quantitative Risk Analysis (pp. 661–670). John Wiley & Sons, Ltd. https://doi.org/10.1002/9780470935422.app4

Colaco, J.-L., Pagano, B., Pouzet, M., & Pouzet, M. (2017). Scade 6: A Formal Language for Embedded Critical Software Development. 10.

Cuenot, P., Frey, P., Johansson, R., Lönn, H., Papadopoulos, Y., Reiser, M.-O., Sandberg, A., Servat, D., Kolagari, R. T., Törngren, M., & Weber, M. (2007). The EAST-ADL architecture description language for automotive embedded software. Proceedings of the 2007 International Dagstuhl Conference on Model-Based Engineering of Embedded Real-Time Systems, 297–307.

David, P., Idasiak, V., & Kratz, F. (2009). Automating the synthesis of AltaRica Data-Flow models from SysML. In M. & G. S. Briš (Ed.), ESREL 2009 (p. 8). Taylor & Francis Group. https://hal.archives-ouvertes.fr/hal-00579532

DoD. (2012). MIL-STD-882E "Department of Defense Standard Practice System Safety". https://www.dau.edu/cop/armyesoh/DAU%20Sponsored%20Documents/MIL-STD-882E.pdf

Estefan, J. A. (2008). Survey of Model-Based Systems Engineering (MBSE) Methodologies. 70.

Feiler, P. H., Lewis, B., & Vestal, S. (2003). The SAE Avionics Architecture Description Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering: Defense Technical Information Center. https://doi.org/10.21236/ADA612735

Fenelon, P., & McDermid, J. A. (1993). An integrated tool set for software safety analysis. Journal of Systems and Software, 21(3), 279–290. https://doi.org/10.1016/0164-1212(93)90029-W

ISO 26262-1_2018_Ed2. (n.d.).

Legendre, A., Lanusse, A., & Rauzy, A. (2016, December). Synchronisation des modèles d'architecture et d'analyse de risques: Quel gain, comment et pourquoi ? Congrès Lambda Mu 20 de Maîtrise des Risques et de Sûreté de Fonctionnement. Congrès Lambda Mu 20 de Maîtrise des Risques et de Sûreté de Fonctionnement, 11-13 Octobre 2016, Saint Malo, France. https://doi.org/10.4267/2042/61813

Lisagor, O., Kelly, T., & Niu, R. (2011). Model-based safety assessment: Review of the discipline and its challenges. The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety, 625–632. https://doi.org/10.1109/ICRMS.2011.5979344

Nguyen, N., Mhenni, F., & Choley, J.-Y. (2018). AltaRica 3.0 code generation from SysML models (pp. 2435–2440). https://doi.org/10.1201/9781351174664-306

Papadopoulos, Y., & McDermid, J. A. (1999). Hierarchically Performed Hazard Origin and Propagation Studies. Computer Safety, Reliability and Security, 139–152. https://doi.org/10.1007/3-540-48249-0_13

Point, G., & Rauzy, A. (1999). AltaRica: Constraint automata as a description language. http://www.altarica-association.org/ressources/ARBib/PointRauzy1999-AltaRicaConstraintLanguage.pdf

SAE. (2015). Potential Failure Mode and Effect Analysis Reference Manual. https://www.lehigh.edu/~intribos/Resources/SAE_FMEA.pdf

Sirgabsou, Y., Pahun, L., Baron, C., Grenier, L., Bonnard, C., & Esteban, P. (2020). Investigating the use of a model-based approach to assess automotive embedded software safety. 9.

VDA QMC. (2015). Automotive SPICE Process Assessment / Reference Model. http://www.automotivespice.com/fileadmin/software-download/Automotive_SPICE_PAM_30.pdf

Vesely, W. E., Goldberg, F. F., Roberts, N. H., & Haasl, D. F. (1981). NUREG-0492, "Fault Tree Handbook". 209.