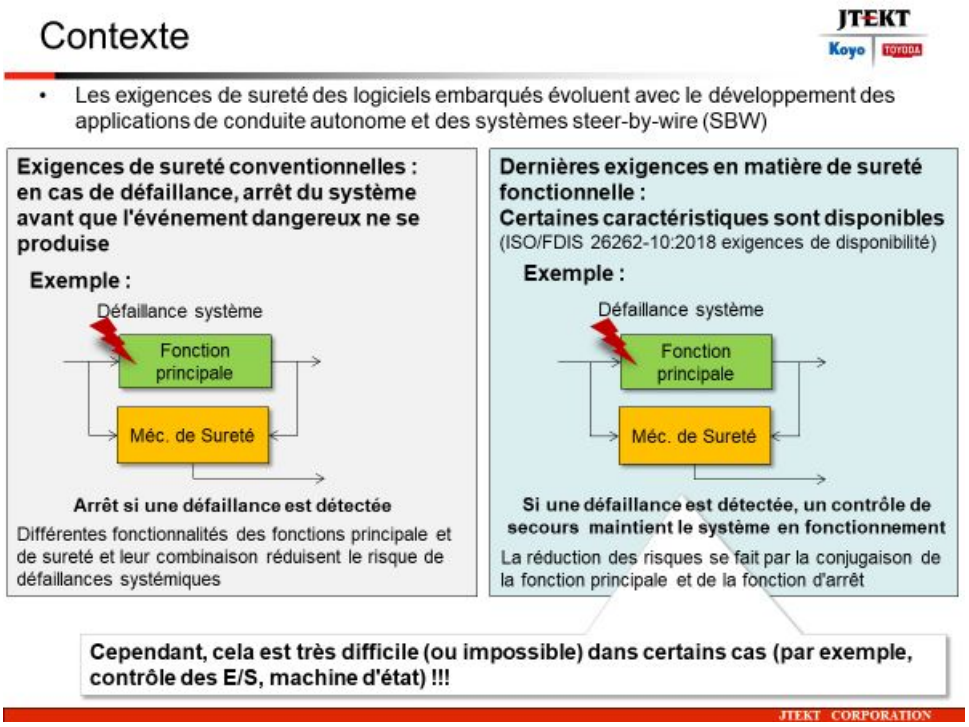
18.09.20

Toyota et NVIDIA se focalisent sur le développement de logiciels zéro défaut prouvé mathématiquement

Hidemasa Kimura　Nikkei Xtech

Le modèle de développement zéro défaut des logiciels embarqués pour véhicules attire beaucoup d'attention. Cela tient au besoin accru de sécurité et de sûreté dans le cadre de la conduite autonome. JTEKT, un important fabricant de systèmes de direction à assistance électrique (EPS pour Electric Power Steering system), et filiale de Toyota, est en train d'introduire cette démarche dans sa production de masse. Aux États-Unis, NVIDIA, qui produit des systèmes embarqués sur puce (SoC) pour la conduite autonome, envisage également d'utiliser une approche similaire.

JTEKT et NVIDIA examinent la méthode dite de la « preuve déductive » (un type de méthode formelle) pour prouver mathématiquement qu'un logiciel n'a pas de défauts. Avec l'utilisation opérationnelle de la conduite autonome et des systèmes steer-by-wire (SBW), les exigences de sûreté des logiciels embarqués augmentent rapidement. Dans le passé, lorsqu'une défaillance survenait dans la fonction principale d'un système, il suffisait d'arrêter le système à l'aide de mécanismes de sûreté. En revanche, avec la conduite autonome et le SBW, il est risqué d'arrêter le système, de sorte qu'il est nécessaire de maintenir un fonctionnement minimal (le fonctionnement spécifié) assuré par le mécanisme de sûreté.



Les exigences de sûreté évoluent dans le contexte de la conduite autonome et du SBW (Source : JTEKT)

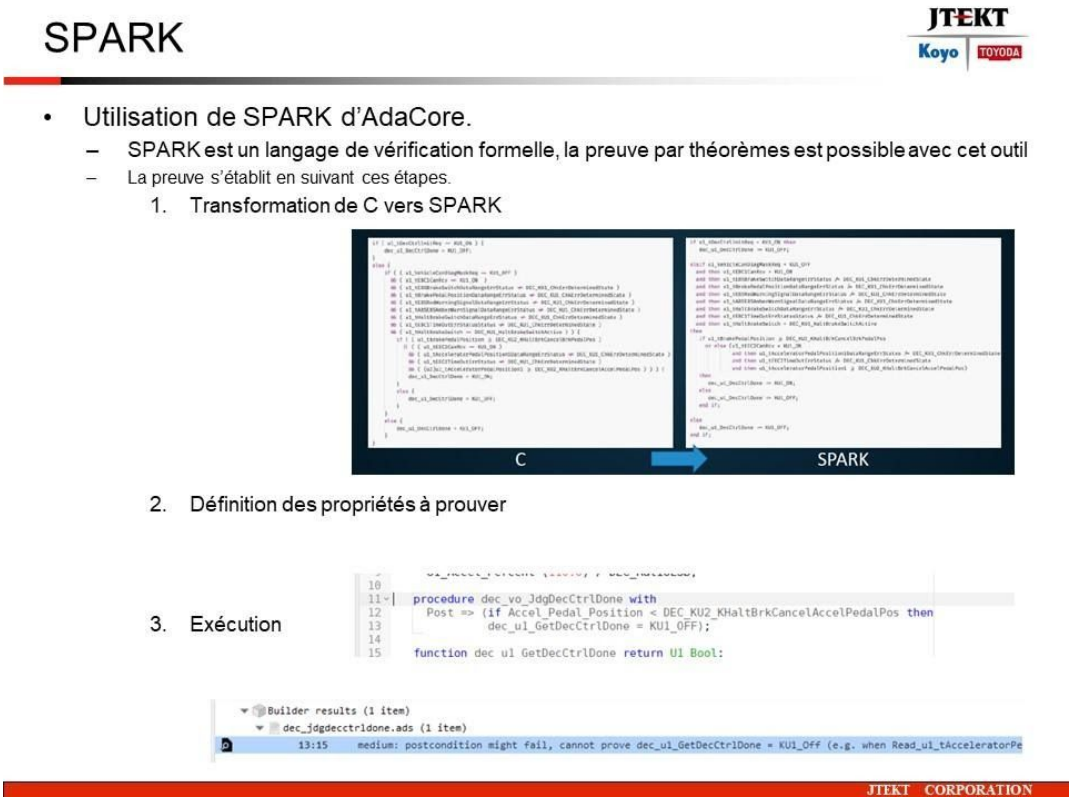
Cependant, la mise en œuvre d'un tel mécanisme de sûreté est « parfois difficile », souligne Shinya Yoneki, du département Advanced System Development de la division Steering Business de JTEKT. Normalement, un mécanisme de sûreté est mis en œuvre avec un logiciel qui est différent de la fonction principale. Avec l'EPS, la fonction principale et le mécanisme de sûreté doivent tous deux renvoyer la même sortie de données à la même entrée, ce qui rend difficile la modification de

l'implémentation du logiciel. Par conséquent, JTEKT cherchait une méthode pour prouver que la fonction principale en soi ne causerait pas de problèmes, même sans le mécanisme de sûreté.

Normalement, la sûreté des logiciels embarqués est vérifiée au cours d'un processus de tests. Mais, il est difficile de prouver qu'il n'y a pas de défauts. Il existe un nombre infini de conditions de tests, et quelle que soit l'étendue du processus de tests, il est impossible d'affirmer qu'il n'y a pas de défauts. Il appartient donc au client de décider en dernier ressort si le produit est sûr ou non. En revanche, la preuve déductive peut mathématiquement prouver que le logiciel ne présente aucun défaut. « Cela pourrait servir de base à une décision sur laquelle tout le monde peut s'entendre », explique Shinya Yoneki.

Le choix du langage SPARK

L'outil utilisé est SPARK Pro, fourni par AdaCore. JTEKT étant actuellement en phase d'implémentation, ses développeurs convertissent d'abord le code source, du langage C au langage à preuve déductive SPARK, puis procèdent à sa vérification. Après avoir confirmé l'absence d'erreurs, ils reconvertissent le code source en C. A terme, ils prévoient d'utiliser SPARK dès le début du développement des nouveaux logiciels, et de convertir le code en C après la vérification par preuve déductive.



Utilisation de SPARK (Source : JTEKT)

L'usage de SPARK est « similaire à la revue de code » explique Shinya Yoneki. Mais, le langage est SPARK, et non pas C. En plus du programme, SPARK permet de décrire les attributs que l'on veut vérifier (comme les plages de sortie/entrée de données dans les spécifications). Lorsqu'on exécute l'outil de vérification, on peut immédiatement voir si le programme comporte des défauts ou si la spécification est erronée. « C'est comme si on remplaçait la revue de code par une machine », considère Shinya Yoneki.

Par exemple, si un programme est jugé incompatible avec les spécifications, alors que l'on sait par expérience que le programme est correct, « nous pouvons découvrir qu'il y a eu une omission dans les spécifications », déclare Shinya Yoneki. De tels défauts pourraient aussi bien être trouvés lors du processus de tests, mais « si l'ingénieur logiciel pouvait vérifier le programme par lui-même lors de son écriture, cela permettrait un travail plus rapide », remarque-t-il.

Dans les produits sur lesquels JTEKT a travaillé par le passé, la majorité des défauts constatés lors des processus de tests étaient dus à des erreurs dans les spécifications ou dans l'implémentation du logiciel lui-même. C'est donc un grand avantage de pouvoir réduire les erreurs dans les deux causes de défauts qui sont les plus courantes au stade du développement.

Pas pour tous les usages

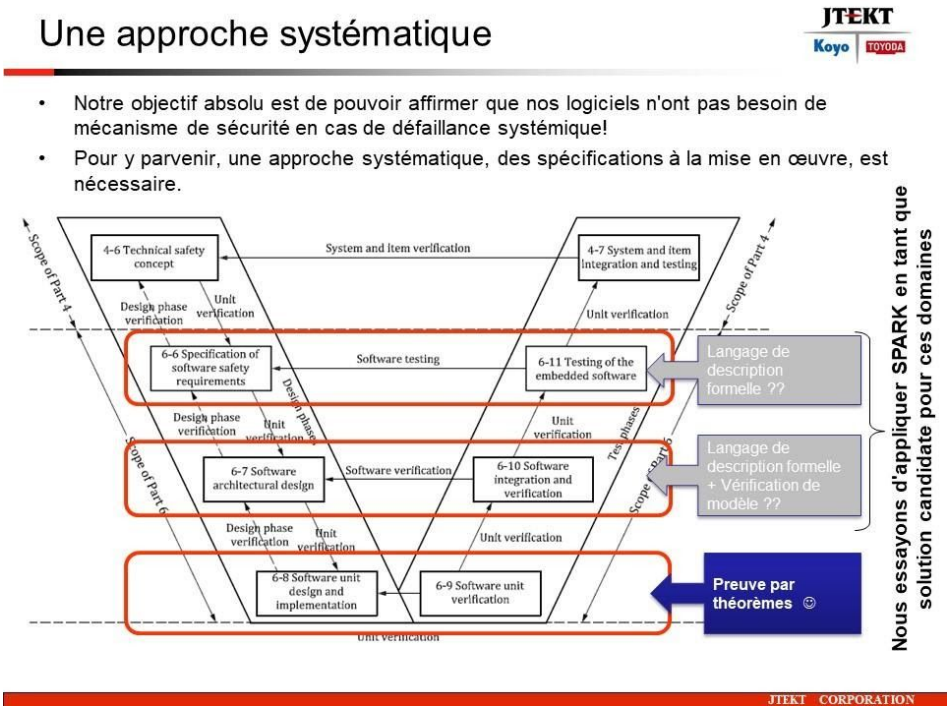
Toutefois, « cette méthode n'est pas universelle », souligne Shinya Yoneki. Bien que JTEKT ait confirmé que la méthode de la preuve déductive peut être utilisée, dans une certaine mesure, comme une alternative aux tests, il est nécessaire de procéder à une évaluation pour savoir si les défauts trouvés dans SPARK peuvent être identifiés avec les outils existants. « Nous travaillons actuellement sur ce point », dit-il.

M. Yoneki Shinya Développement de systèmes avancés à JTEKT, (Source : JTEKT)

Le plus grand défi est que les programmes auxquels la méthode de la preuve déductive peut être appliquée sont aujourd'hui en nombre limités. « Il est difficile d'appliquer la méthode à moins que le programme ait une grande affinité avec la logique de Hoare, qui est la base de la preuve déductive », précise Yoneki Shinya. Par exemple, « les programmes qui peuvent clairement décrire les limites supérieures et inférieures des plages de sortie/entrée de données sont adaptés », dit-il. D'autre part, « Dans le cas de programmes tels que le contrôle de rétroaction, où la sortie change en fonction de l'état précédent, la

plage de sortie/entrée est difficile à décrire et difficile à appliquer », ajoute-t-il. Il est donc nécessaire de déterminer la compatibilité de chaque programme.

JTEKT a l'intention d'utiliser la méthode en premier lieu pour les composants logiciels qui devront communiquer directement avec le matériel et pour les programmes de contrôle de base. Ces logiciels n'ont pas de commandes très complexes, mais leur mauvais fonctionnement peut avoir des conséquences importantes pour la sûreté.



Appliqué aujourd'hui au bas du processus en V (Source : JTEKT)

Les logiciels sur lesquels est appliquée la méthode de la preuve déductive sont de petite taille (en nombre de lignes de code), moins de quelques pour cent du système EPS dans son ensemble. Néanmoins, il est important de pouvoir dire que cette partie du programme atteint l'objectif de zéro défaut. JTEKT veut ainsi augmenter la sûreté globale du système en commençant par prouver que les petits composants logiciels sont à zéro défaut puis en développant à partir de cette base les fonctions qu'ils souhaitent atteindre.

NVIDIA étudie également le langage SPARK d'AdaCore, tout comme le langage Ada qui est à la base de SPARK. NVIDIA envisage de réécrire certains firmwares critiques pour la sécurité, du langage C vers Ada et SPARK. Dans le mesure où les voitures à conduite autonome sont connectées au monde extérieur par des fonctions de communication, les vulnérabilités du firmware SoC embarqué en font une cible pour les cyber-attaques. Si les vulnérabilités peuvent être éliminées en utilisant la méthode de la preuve déductive, cela pourrait augmenter la sécurité.

Publié dans Nikkei xTECH, le 18 septembre 2020
https://xtech.nikkei.com/atcl/nxt/column/18/00001/04594/
Traduit par AdaCore avec l'autorisation des détenteurs de droits.