Les logiciels de sécurité de Trainguard MT CBTC

La Canarsie Line à New York, une application du Trainguard CBTC

© Siemens TS

■ Daniel DOLLE, Siemens TS

Cet article présente de manière synthétique les technologies qu'utilise STS pour garantir le fonctionnement en toute sécurité de ses automatismes ferroviaires, essentiellement basés sur des logiciels aujourd'hui. Le processeur sécuritaire codé permet de se prémunir contre les erreurs de calcul, la méthode B contre les erreurs de conception du programme.

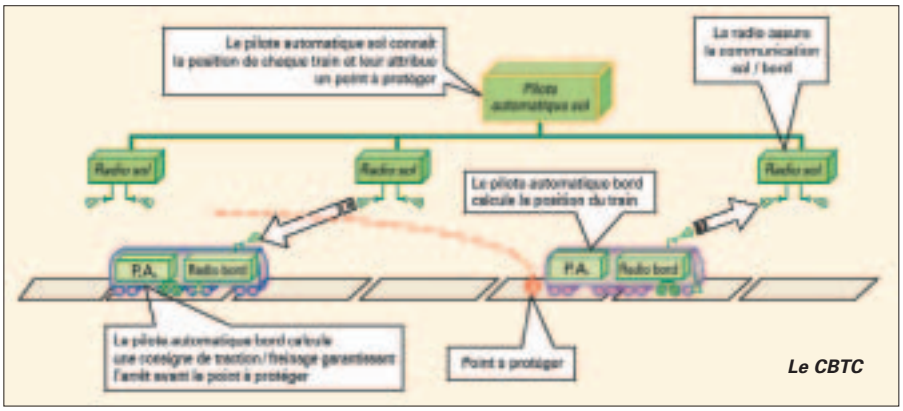
PÔLE international d'expertise pour les automatismes de transport urbain, Siemens Transportation Systems est le centre de compétences du groupe Siemens pour les systèmes de transport sans conducteur. La gamme des produits développés par STS va de l'aide à la conduite jusqu'à l'automatisme intégral. Elle comprend notamment le VAL, un système entièrement automatique de métro sur pneu qui a plus de vingt ans d'existence et dont la plus récente mise en service a eu lieu en 2006 à Turin. Cette même année, le produit phare de la gamme d'automatismes – le Trainguard MT CBTC – a été déployé par le New York City Transit sur sa ligne Canarsie.

La rénovation de cette ligne, rendue difficile par son exploitation 24 heures sur 24 et 7 jours sur 7, a permis de répondre aux exigences du client en matière de sécurité, de fréquence des trains et de réduction des coûts de maintenance.

Qu'est-ce que le CBTC?

Définie par la norme IEEE 1474.1, la dernière génération des automatismes de transport urbain dite CBTC (Communications-Based Train Control, ou "contrôle-commande des trains par télécommunications") est fondée sur trois principes:

- le contrôle des trains est continu sur la portion de voie équipée en automatismes;
- les trains sont localisés avec précision et indépendamment des circuits de voie;
- les automatismes bord et sol échangent continûment des informations via un système de communication disponible en tout lieu et à tout moment;



Le CBTC

● les calculateurs à bord et au sol effectuent des traitements de sécurité.
Cette définition normative se décline chez Siemens TS par le produit Trainguard MT CBTC. Ce produit est adapté non seulement à l'automatisation de lignes nouvelles mais aussi aux besoins de rénovation de lignes en exploitation. Il répond également aux exigences d'interopérabilité exprimées par le métro de New York et d'interchangeabilité exprimées par la RATP.
Avec Trainguard MT CBTC, la communication entre le sol et le bord est assurée par une radio à propagation libre basée sur de la modulation à étalement de spectre. Dans ce système, c'est le train qui calcule sa propre position sur la ligne et la transmet aux automatismes "sol". Ceux-ci tiennent à jour en continu la cartographie de la portion de voie qu'ils gèrent et assignent à chaque train un point à protéger. Ce point à protéger correspond en fait à l'arrière du train qui le précède. Les automatismes "bord" élaborent ensuite le profil optimal de vitesse de manière à assurer l'arrêt en sécurité au niveau du point à protéger.
Les trains sont donc espacés au minimum d'une distance correspondant à leur distance de freinage plus une distance de sécurité. Cette distance peut être assimilée à un canton qui se déplace avec le train d'où le nom de "canton mobile".

Les logiciels de sécurité: contexte historique

Les automatismes "sol" et "bord", réalisés en logiciel, assurent des fonctions de sécurité. Il se pose alors le problème de la confiance que l'on peut leur accorder. STS utilise deux techniques complémentaires pour répondre à ce problème: le processeur sécuritaire codé garantit que l'application s'exécute conformément à son code source et la méthode B protège le code source contre les erreurs de conception et de codage.
Le processeur sécuritaire codé (PSC) a été développé dans les années 1980 par le consortium chargé de réaliser le SACEM (Système d'Aide à la Conduite, à l'Exploitation et à la Maintenance), un automatisme ferroviaire destiné à la ligne A du RER parisien. Le PSC est basé sur une technique de codage arithmétique et garantit que le logiciel s'exécute conformément à son code source. Il est particulièrement adapté aux environnements soumis à de fortes perturbations électromagnétiques que l'on rencontre dans le ferroviaire.
Par nature, le PSC ne protège pas contre les erreurs de conception. La RATP a donc souhaité renforcer significativement les moyens de validation de la conception. Comme une validation classique par test semblait insuffisante, une approche dite "formelle" a été préférée:
● le code source du logiciel, écrit en Modula 2, a été complété de pré- et postconditions vérifiées formellement de façon semi-automatique;
● la spécification du logiciel a été réécrite en langage formel (c'est à dire mathématique);
● la conformité entre la spécification formelle et le code muni de ses pré- et postconditions, a été vérifié manuellement au moyen de démonstrations

Développement B: un petit exemple

LA spécification du logiciel - Le point de départ d'un développement B est une écriture textuelle de ce que l'on attend du système. Ce document, organisé selon les principes de l'analyse fonctionnelle (SADT), explicite rigoureusement ce que doit réaliser le logiciel et les notations utilisées sont celles qui contribuent le mieux à clarifier l'exposé: français, automates d'état, équations ou notations B selon le cas. C'est un document qui doit faciliter l'étape de formalisation qui suit sa rédaction.
Nous allons maintenant examiner le processus de développement sur un exemple simpliste mais représentatif de la démarche: *«Un train situé en zone d'interdiction de marche doit déclencher son freinage d'urgence.»*
La formalisation des exigences - La première étape du développement formel va consister à traduire en B le document de spécification. Ce travail se termine quand toutes les exigences ont été reportées dans le **modèle B abstrait**. Le développement du modèle abstrait vise à atteindre les objectifs suivants:
● faciliter la vérification de conformité entre le modèle B et sa spécification textuelle;
● faciliter la preuve;
● laisser le plus de liberté possible à la réalisation.
Dans notre exemple, la formalisation se fait en modélisant le train et la zone interdite par l'ensemble des abscisses qu'ils recouvrent.

Modèle de spécification

position_train ∩ zone_interdite ≠ Ø <=>
freinage_d_urgence = VRAI

Le raffinement - L'étape de raffinement consiste ensuite à remplacer les notions abstraites par des notions informatiques pour aboutir au modèle B concret. C'est un travail qui demande des compétences plus "informatiques" que "ferroviaires". Manuel à l'époque de Météor, ce travail est aujourd'hui largement automatisé: 75% du modèle concret de la Canarsie Line a été déduit du modèle abstrait grâce à un outil informatique.
Dans l'exemple, les ensembles et l'intersection sont remplacés par des entiers et des comparaisons.
Le résultat auquel on aboutit est au niveau d'abstraction de n'importe quel langage de programmation. La traduction vers le langage cible - Ada dans notre cas - est entièrement réalisé par un outil.

Modèle de réalisation

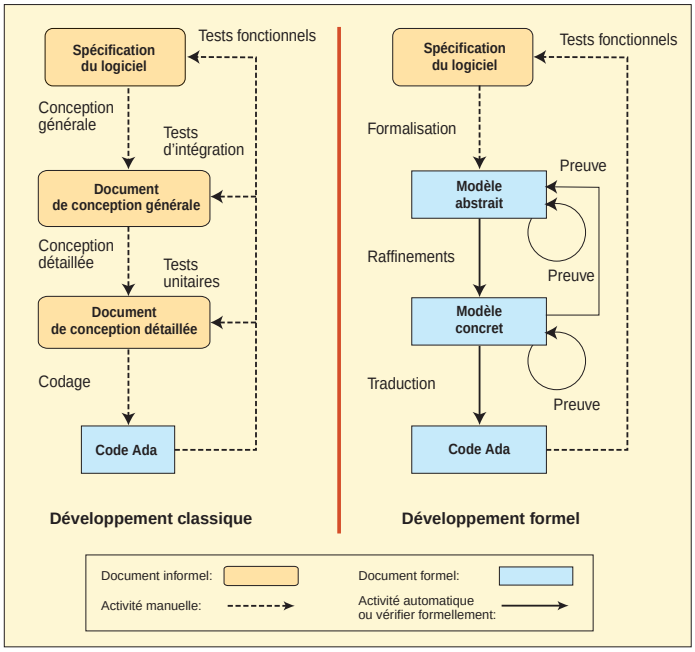
position train = a .. b & zone_interdite
= c .. d

IF c > d THEN
freinage_d_urgence := FAUX
ELSIF b <= c THEN
freinage_d_urgence := FAUX
ELSIF a <= c & b >= c THEN
freinage_d_urgence := VRAI
ELSIF a >= c & b <= d THEN
freinage_d_urgence := VRAI
ELSIF a <= c & b >= d THEN
freinage_d_urgence := VRAI
ELSIF a <= d & b >= d THEN
freinage_d_urgence := VRAI
ELSIF a > d THEN
freinage_d_urgence := FAUX
END

La preuve - L'activité de preuve se déroule au fur et à mesure de l'écriture des modèles B. Elle vise à démontrer que le modèle mathématique est cohérent et que chaque étape de raffinement préserve les propriétés de l'étape précédente.
Dans le cas de l'exemple on doit démontrer 8 lemmes (un pour chacune des branches du IF et un lemme supplémentaire pour la branche implicite du cas ou aucune des conditions n'est vraie). La démonstration de la branche 2, en gras, est laissée en exercice au lecteur.

not(c > d) & b <= c
([a, b] ∩ [c, d] ≠ Ø ∫ FAUX = VRAI)

mathématiques. La validation formelle du SACEM, réalisée a posteriori sur des logiciels développés classiquement, permit de trouver une vingtaine d'erreurs de conception mais fut très coûteuse pour la RATP.
Le projet suivant de la RATP était Météor, un métro sans conducteur qui permet l'exploitation simultanée de rames en conduite automatique intégrale et de rames non automatisées en conduite manuelle. Ce projet fut confié à Siemens TS et la RATP exigea l'utilisation de méthodes formelles pour concevoir les logiciels de sécurité. Ce fut la première utilisation industrielle de la méthode B. Le processus de développement basé sur la preuve a donné des logiciels de qualité et de sécurité supérieures à celle d'un développement classique. Claude Hennebert, délégué auprès du Directeur Général Adjoint de la RATP en 1998, a même affirmé: *«Je n'avais jamais vu cela. Le logiciel a été quasi parfait du premier coup.»*
Depuis sa première mise en œuvre, le processeur codé a été industrialisé par Siemens TS. Le codage arithmétique est maintenant calculé sur un coprocesseur puissant qui laisse toute sa puissance à l'unité centrale, un Pentium. Le PSC a été certifié sous le nom de Digisafe par l'INRETS et par TÜV InterTraffic. De même, la compétence en développement formel a été capitalisée sous la forme d'un *Guide de développement B* de 200 pages et de nombreux outils facilitant le développement formel et la preuve ont été réalisés.



Comparaison entre un cycle de développement classique et un cycle B. Les flèches signifient "sert à produire" ou, pour les tests, "validé vis-à-vis de"

La méthode B

Inventée par Jean-Raymond Abrial, la méthode B est une manière formelle – mathématique – de développer des systèmes sûrs. Son objectif est de garantir par construction qu'ils fonctionnent correctement. La méthode B permet, dans un même langage, de modéliser un système ou un logiciel depuis sa spécification jusqu'à sa réalisation. Les modèles décrivent des fonctions et aussi des propriétés générales. Ils s'écrivent dans un langage mathématique simple mais expressif qui permet aux différents intervenants de se comprendre sans risque d'ambiguïté.
La principale différence entre un développement B et un développement classique tient à la preuve: en B, toute propriété doit être prouvée mathématiquement. On peut ainsi démontrer que le système modélisé a bien les propriétés attendues. La validation démarre donc au plus tôt dans le cycle

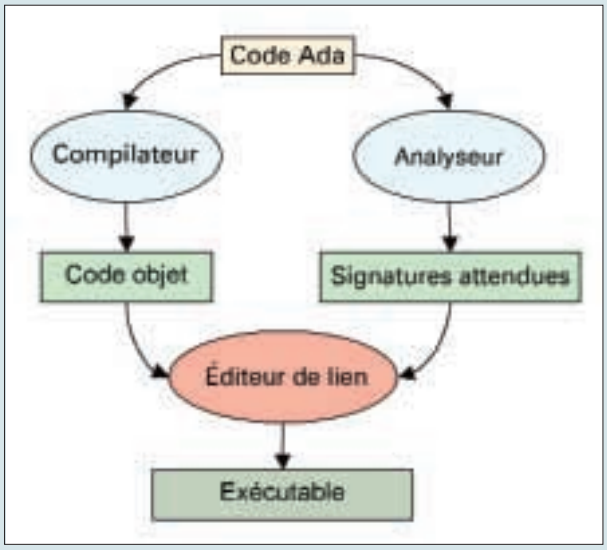
de vie du logiciel, dès sa spécification.
La validation constitue aussi une "vérification" plus complète: il est généralement impossible de tester toutes les valeurs d'entrée d'une fonction alors qu'une preuve mathématique peut être valide pour un grand éventail d'entrées.
Actuellement tous les logiciels sécuritaires des pilotes automatiques de Siemens TS sont développés en B, depuis la formalisation de la spécification jusqu'au code.
Les modèles obtenus sont prouvés à 100% et la traduction du code B en langage compilable (Ada) est entièrement produite par un outil. L'emploi de la preuve a permis de supprimer totalement les tests unitaires et certains tests d'intégration.
Le principal outil de développement est l'Atelier B, commercialisé par la société ClearSy. Cet outil vérifie que les modèles sont correctement écrits et calcule automatiquement les lemmes (c'est-à-dire les propriétés logiques) qu'il faut démontrer mathématiquement pour valider un modèle. L'Atelier B contient deux outils de démonstration de lemmes. L'un est automatique et arrive à démontrer entre 80 et 90% des lemmes; l'autre, semi-automatique, permet aux développeurs de préciser de nouvelles consignes pour démontrer les lemmes restants. L'Atelier B a atteint un remarquable degré de maturité: les logiciels de la Canarsie Line représentent près de 270 000 lignes de B et leur validation a nécessité la démonstration de plus de 80 000 lemmes.

La validation d'un développement formel

La validation consiste à s'assurer que le code compilé va s'exécuter conformément aux besoins identifiés pour le logiciel. Ce travail est fait par étapes:
• ***validation du modèle vis-à-vis de la spécification*** - La conformité entre la spécification informelle du logiciel (voir encadré) et le modèle est vérifiée par relecture et par des tests fonctionnels.
Puisque le modèle B abstrait formalise toute la spécification, l'analyse de conformité ne porte que sur lui (ensuite les modèles sont déjà prouvés corrects par rapport aux spécifications). Dans le cas de la Canarsie Line, c'est moins de 50% du modèle qui a dû être analysé par l'équipe sûreté de fonctionnement, ce qui constitue un allègement notable

Les principes du processeur codé

LE PSC associe à chaque donnée qu'il faut sécuriser, un "code" arithmétique qui la protège contre les erreurs de la chaîne de compilation et contre les erreurs d'exécution. Le niveau de sécurité ne dépend que de la taille du code. Il est indépendant de la fiabilité et de la technologie du matériel utilisé. Il est également indépendant de la chaîne de compilation, ce qui autorise l'emploi de compilateurs non certifiés ainsi que de systèmes d'exploitation ordinaires (Trainguard MT CBTC utilise par exemple Apex ou Gnat Pro avec VxWorks).



Le processeur sécuritaire codé

Dans un logiciel de sécurité réalisé avec le PSC, chaque variable X est représentée par un couple d'entiers (X_F, X_C). La donnée "fonctionnelle" X_F est protégée par la partie "contrôle" X_C qui est la somme de trois termes:

$$X_C = -(2^k.X_F \bmod A) + B_X + D$$

Le principe du PSC est que la partie contrôle et la partie fonctionnelle d'une variable vont évoluer de manière indépendante. A la fin de chaque cycle de calcul, les deux termes de l'égalité sont comparés. S'ils diffèrent, on sait qu'il y au eu une erreur, s'ils sont égaux on sait borner la probabilité qu'il y a eu une erreur de calcul.

La partie contrôle se décompose comme suit:
• le premier terme est un codage arithmétique qui détecte les altérations des données en mémoire ou lors de leurs transferts. A et 2^k caractérisent le niveau de protection que l'on cherche à obtenir. Dans nos applications leur ordre de grandeur est de 10^{14}, cela permet d'atteindre un taux de panne dangereuse inférieur à 10^{-9} par heure sur l'ensemble du système CBTC de la Canarsie Line (New York);
• B_X est une signature statique qui est fonction de la variable et des traitements qu'elle a subis;
• D est une signature dynamique qui évolue à chaque cycle de calcul et sert à détecter les erreurs de rafraîchissement des données.
Lors des calculs réalisés par l'application, le terme $B_X + D$ de la partie contrôle (X_C) évolue indépendamment de la partie fonctionnelle (X_F). Cette évolution peut être prévue avant l'exécution par analyse du code source. Il est alors possible de vérifier que les sorties de l'application sont correctes en comparant leur signature prévue avec celle obtenue lors de l'exécution. La comparaison est réalisée à la fin de chaque cycle de calcul par un dispositif matériel réalisé en sécurité intrinsèque, le "contrôleur dynamique". En cas d'erreur, il positionne toutes les sorties à l'état restrictif. On notera que la comparaison se fait modulo A et qu'elle a une probabilité faible, de l'ordre de 10^{-14}, de conclure que tout va bien alors qu'une variable n'a pas la valeur correcte.

La sécurité offerte par le PSC repose sur l'absence de lien entre les erreurs issues de la chaîne de compilation et les erreurs de l'analyseur: si l'un de ces outils a un défaut, le défaut ne sera pas présent dans l'autre et les signatures attendues différeront donc des signatures calculées à l'exécution. Cette hypothèse se justifie par le fait que les outils ont été réalisés indépendamment: le compilateur est un compilateur Ada commercial, l'analyseur a été développé spécifiquement pour analyser le sous-ensemble Ada que supporte le PSC.

Réalisations de Siemens TS avec la méthode B et le processeur codé

de l'activité de validation et traçabilité.
Pour se prémunir des erreurs survenant lors de la modélisation, les tests se font à partir de la spécification informelle du logiciel (jamais à partir de la spécification formelle) et sont tous des tests fonctionnels. Siemens TS n'effectue jamais de tests unitaires. Cela reviendrait à vérifier la correction du code vis-à-vis de la conception détaillée, une étape déjà "sécurisée" par la preuve et le double transcodage (deux outils indépendants sont utilisés pour transformer le B concret en langage informatique Ada);

- ***validation du modèle B concret vis-à-vis du modèle abstrait*** - La conformité est vérifiée par preuve;
- ***validation du modèle B concret vis-à-vis du code Ada*** - La conformité est garantie par l'utilisation de deux transcodeurs redondants;
- ***validation de l'exécution vis-à-vis du code Ada*** - La conformité est vérifiée pendant le fonctionnement du logiciel par le processeur codé.

Les limites de la formalisation

Les limites sont doubles. D'abord, le langage B ne contient pas de construction permettant d'exprimer explicitement des contraintes temps réel. On peut toutefois les formaliser dans le modèle abstrait en s'appuyant sur l'hypothèse que le logiciel est activé à des intervalles de temps constants, une hypothèse garantie par le matériel.
Ensuite, les interfaces avec la partie non formelle de l'application ne possèdent pas de modèles B concrets. Ces modèles gèrent les entrées/sorties de l'application ainsi que celles qui contiennent les données de configuration de la voie.

Conclusion

La première exigence des exploitants et de leurs clients est d'assurer le transport de manière sûre. Siemens TS répond à cette attente par des processus de développement rigoureux (placés au niveau 3 du CMM) et par l'utilisation des techniques les plus avancées. Utilisés conjointement depuis plus de dix ans, le processeur codé et la méthode B permettent de produire des systèmes de qualité inégalée, qualifiés au niveau SIL4 selon les normes du ferroviaire (50126, 50128, 50129) et qui ont fait leur preuve partout dans le monde. Depuis la mise en service de Météor, aucune anomalie n'a été constatée dans les logiciels développés selon ce principe. ■