Back-to-Back Testing : méthodes formelles pour maîtriser la conformité des implémentations logicielles

Guignard Anaïs, Alain Ourghanlian, Niguez Julien, Nicolas Systerel Breton, Vincent Pouzol

Institut pour la Maîtrise des Risques
Sûreté de Fonctionnement - Management - Cindyniques

Back-to-Back Testing : méthodes formelles pour maîtriser la conformité des implémentations logicielles

Back-to-Back Testing: formal methods for the control of software implementations compliance

GUIGNARD Anaïs
Systerel Paris
3 rue Danton, 92240 Malakoff
anais.guignard@systerel.fr

NIGUEZ Julien
Systerel Aix-en-Provence
1090 rue René Descartes, 13100 Aix-en-Provence
julien.niguez@systerel.fr

POUZOL Vincent
Systerel Aix-en-Provence
1090 rue René Descartes, 13100 Aix-en-Provence
vincent.pouzol@systerel.fr

OURGHANLIAN Alain
EDF R&D PRISME
6 quai Watier, 78400 Chatou
alain-1.ourghanlian@edf.fr

BRETON Nicolas
Systerel Toulouse
6 impasse Michel Labrousse, 31100 Toulouse
nicolas.breton@systerel.fr

***Résumé* —** Le Back-to-Back Testing (B2BT) décrit un procédé qui permet de générer automatiquement à l'aide de méthodes formelles un ensemble de séquences de test à partir d'un modèle. Ces séquences caractérisent le comportement observable du modèle. Pour chaque pas de test, elles décrivent les entrées et sorties du modèle, les sorties étant calculées en fonction des entrées et de l'état du programme. Ces séquences sont ensuite mises en œuvre sur un banc afin de confirmer qu'une implémentation matérielle et/ou logicielle est conforme au modèle initial. Ces séquences sont construites de manière à détecter les mutations qui modifieraient le comportement observable du modèle. Comme pour toute stratégie de tests, un certain nombre d'indicateurs de couverture sont définis par l'approche B2BT. Des premiers essais industriels, en partenariat avec EDF, sont en train d'être menés.

Mots-clefs — preuve formelle, vérification, validation, systèmes critiques

***Abstract* —** B2BT is a process of automatic test sequences generation based on formal methods. These sequences describes the observable behavior of system: for every step, inputs and outputs are given. Outputs values are computed from inputs and memories values. Hence, they are used on a test bench in order to control the compliance of the implementation with the initial model used to generate the sequences. The sequences are designed in order to detect mutations of the implementation that would change the observable behavior of the system. Coverage criteria are given along the B2BT method. Industrial applications based on EDF use case and a railway signaling use case are presented.

Keywords —Formal proof, verification, validation, critical systems

I. Introduction

Les techniques de Back-to-Back Testing (B2BT) permettent de vérifier que deux produits des étapes d'un processus de développement logiciels sont équivalents. En exécutant « dos à dos » (« back to back » en anglais) les deux produits sur un ensemble de scénarios de test, on vérifie qu'ils produisent les mêmes sorties. Si les scénarios sont choisis avec soin, et que les deux exécutions donnent les mêmes résultats, il est possible d'en déduire que les deux produits sont équivalents avec un certain niveau de confiance.

Généralement, ces techniques sont utilisées pour s'assurer qu'un modèle d'un logiciel critique est une représentation fiable d'un logiciel exécuté sur sa cible matérielle. Cela permet de reporter des résultats de techniques de validation obtenus à partir du modèle, par exemple des revues, des tests fonctionnels, ou des preuves formelles, directement sur l'objet réel. Ainsi, si une propriété de sécurité est prouvée sur un modèle, et que l'on a une confiance suffisante dans les tests effectués en B2BT, alors il est possible d'en déduire avec un certain niveau de confiance que la propriété tient sur le logiciel exécuté sur sa cible matérielle.

Le modèle utilisé peut être lui-même le résultat de la validation. En diversifiant les artefacts, et en montrant qu'ils sont équivalents, on peut en déduire que les deux processus diversifiés permettant d'obtenir ces produits n'ont probablement pas introduit d'erreur.

Une autre utilisation possible de ces techniques est de valider une transformation effectuée sur un premier produit

pour obtenir le second. Par exemple un générateur automatique de code, un compilateur, un linker, un optimisateur. S'il est possible d'exécuter (ou de simuler) le premier produit et de vérifier qu'il se comporte de façon identique au second par des méthodes de B2BT, alors il est possible de conclure que l'outil utilisé pour effectuer la transformation n'a probablement pas introduit d'erreur.

Cet article est organisé de la manière suivante : la partie II présente le principe du B2BT et de la génération des séquences de test, illustré autour d'une application. La stratégie employée pour générer les séquences est expliquée dans la partie III. La mise en œuvre des tests autour du système à tester est détaillée dans la partie IV. Enfin, les parties V, VI et VII présentent les cas d'utilisation et deux projets pour lesquels notre solution de B2BT a été utilisée.

II. Presentation de la solution de B2BT de Systerel

A. *Problématique posée*

Dans les systèmes critiques, les normes prévoient des activités pour maîtriser les défauts qui séparent la spécification d'un système de son implémentation.

Parmi ces activités, on trouve la vérification, les tests, la simulation, la preuve, la validation, la qualification des outils utilisés. Une des problématiques industrielles concerne la difficulté à valoriser dans la politique de validation tous les travaux menés sur des modèles ou des codes qui ne sont pas encore le logiciel exécuté sur sa plateforme d'exécution.

L'approche B2BT proposée par Systerel s'inscrit dans les techniques de génération automatique de test par Model Checking ([1],[2],[3]) mais propose d'apporter de nouveaux éléments de conviction pour utiliser plus massivement les travaux menés sur des modèles ou du code source par une génération automatique de tests qualifiant la conformité de logiciel exécutable à la sémantique du modèle / code source.

B. *Spécificités de l'outil B2BT*

La solution de B2BT développée par Systerel s'appuie sur nos technologies existantes. Nous avons donc fait le choix de réaliser nos modèles en High Level Language (HLL) [4], le langage utilisé par nos solutions Systerel Smart Solver (S3) [5]. Un modèle peut être le résultat de différentes activités : réalisation spécifique aux activités B2BT, réutilisation des données de validation et de vérification, traduction automatique à partir d'un autre langage, etc.

Enfin, ce modèle est utilisé par l'outil B2BT afin de générer des séquences de test.

Le B2BT s'inscrit dans la catégorie du test par comparaison du comportement observable et à ce titre, les séquences générées décrivent pour chaque pas d'évolution du système les sorties attendues en fonction de ses entrées et de l'état de ses mémoires. Le choix de la stratégie utilisée pour la génération des séquences, ainsi que le niveau de confiance accordé à ces séquences sont décrites dans la partie III.

Bien que cela ne soit pas l'objet de cet article, il est important de noter que le même modèle HLL, peut être utilisé par notre moteur d'analyse S3-core afin d'effectuer des activités de preuve formelle.

C. *Application sur un modèle simple*

Afin d'illustrer le processus de génération des séquences, une application logiciel volontairement simple, mais compréhensible et représentative est détaillée dans cette partie.

Considérons que cette application dispose de 2 entrées, représentées par deux variables booléennes I1 et I2, et de 2 sorties, représentées par deux variables booléennes O1 et O2.

Le comportement observable des sorties en fonction des entrées est décrit par les équations booléennes suivantes.

$$O1 = I1 + I2 \quad (1)$$

$$O2 = I1 * I2 \quad (2)$$

Considérons de plus que nous disposons d'un équipement testable, c'est à dire, dont il est possible de commander les entrées afin d'en contrôler les sorties, qui embarque une implémentation logicielle basé sur le comportement décrit en (1) et (2).

1) *Génération des séquences*

On peut construire le modèle HLL correspondant à ce système. Ce modèle est explicité dans la Fig. 1.

```
Inputs:
    bool I1, I2;

Declarations:
    bool O1, O2;

Definitions:
    O1  :=  I1 # I2;
    O2  :=  I1 & O2;

Outputs:
    O1;
    O2;
```

Fig. 1. Modèle HLL de l'application

Le modèle HLL est ensuite utilisé pour générer les séquences de test. A l'issue de la génération, on obtient un scénario de test de longueur 4 cycles, tel que présenté dans la Fig. 2 où chaque ligne représente les états successifs (f pour False, t pour True) de chaque variable d'entrée (I1 et I2) et de chaque sortie (O1 et O2).

```
<models>
<model name="CEX for PO -1" type="cex" depth="3" length="4">
<variable name="I1" type="bool" kind="input">
f t t f
</variable>
<variable name="I2" type="bool" kind="input">
f t f t
</variable>
<output expression="O1" type="bool" kind="combinational" output-index="0">
f t t t
</output>
<output expression="O2" type="bool" kind="combinational" output-index="1">
f t f f
</output>
</model>
</models>
```

Fig. 2. Séquence de test B2BT générée

2) Détection des divergences dans le comportement

Supposons que le comportement décrit dans le modèle HLL présenté dans la partie précédente ait été implémenté dans un système. Considérons les deux erreurs suivantes :

1) Erreur d'implémentation dans le calcul de la variable 02 : utilisation d'un OU à la place d'un ET,

2) Erreur dans la capture de la variable I1 à chaque cycle : I1 bloqué à VRAI.

Le tableau I présente la table de vérité du système, en détaillant le comportement attendu et le comportement obtenu, sur la séquence de test obtenue à l'aide de l'outil B2BT.

TABLE I. TABLEAU DE VÉRITÉ DU SYSTÈME

		Attendu		**Erreur calcul O2**		**I1 bloqué à VRAI**	
I1	***I2***	***O1***	***O2***	***O1***	***O2***	***O1***	***O2***
FAUX	FAUX	FAUX	FAUX	FAUX	FAUX	**VRAI**	FAUX
VRAI	VRAI	VRAI	VRAI	VRAI	VRAI	VRAI	VRAI
VRAI	FAUX	VRAI	FAUX	VRAI	**VRAI**	**VRAI**	FAUX
FAUX	VRAI	VRAI	FAUX	VRAI	**VRAI**	**VRAI**	**VRAI**

On remarque que cette unique séquence de test permet de détecter les deux erreurs introduites :

- Pour l'erreur de calcul de 02, une différence dans le comportement observable est détectée au 3ème pas de la séquence,
- Pour le blocage de la variable I1, une différence dans le comportement observable est détectée dès le 1er pas de la séquence.

Plus généralement, la stratégie de génération des tests va permettre d'établir un maillage détaillé du comportement de référence, à partir duquel il est possible d'observer les conséquences de défaillances dans l'élaboration de l'implémentation du modèle.

III. STRATEGIE DE GENERATION DES SEQUENCES DE TEST B2BT

Cette partie décrit la stratégie qui a été choisie pour la génération des séquences de test B2BT. L'objectif de l'outil B2BT est d'effectuer des mutations sur le modèle, puis de déterminer un ensemble de séquences qui permettraient de détecter une différence entre le comportement observable du modèle d'origine et le modèle muté.

Afin de déterminer les classes de mutations, nous nous sommes appuyés sur la capacité d'observer une mutation sur les sorties et sur la sémantique du HLL dont notre technologie S3 peut fournir une vision sémantiquement équivalente plus fine. C'est à ce niveau plus fin que les mutations sont opérées, sur les entrées, les mémoires, les opérations. L'inconvénient de cette approche réside dans la quantité surabondante de mutations possibles par rapport à la réalité. Mais c'est aussi sa force car la maille ainsi tressée est d'une extrême finesse.

L'outil B2BT s'appuie donc sur ces classes de mutation pour diriger la génération des séquences de test. En utilisant le moteur d'analyse S3-core, nous cherchons à générer des séquences de test permettant de détecter chacune des mutations possibles du modèle, c'est-à-dire, des séquences d'entrées du système qui produiraient des sorties différentes selon si elles sont évaluées sur le modèle d'origine ou le modèle muté.

Bien que ce ne soit pas l'objectif de cette stratégie, nos expérimentations ont montré qu'elle peut couvrir partiellement l'évolution temporelle d'un signal (voir parties VI et VII). Pour une telle couverture, il conviendrait d'étudier la notion de couverture associée et la possibilité de définir une stratégie dédiée.

IV. MISE EN ŒUVRE DES TESTS

Une fois les séquences de test générées, il est nécessaire de les mettre en œuvre sur le système cible. Cette mise en œuvre peut s'avérer plus ou moins complexe selon les caractéristiques du système à tester et de l'environnement de test disponible.

En effet, les séquences de test B2BT fournissent, pour chaque cycle du système, un ensemble de valeurs d'entrée et les valeurs des sorties attendues. Il est donc nécessaire de synchroniser le moyen de test et le système à tester, afin de s'assurer que :

- le système à tester ait le temps d'acquérir les entrées et de calculer les sorties,
- le moyen de test contrôle des valeurs des sorties attendues et calculées cohérentes (qui correspondent au même cycle de la séquence de test).

Les sous parties suivantes présentent les 4 cas de mises en œuvre que nous considérons.

A. Tests sur machine hôte / simulateur

Le test sur machine hôte / simulateur permet de mettre en place un moyen de test avec les moyens de contrôle les plus fins :

- La synchronisation de l'exécution du système à tester,
- L'observation maximale (les variables internes peuvent devenir accessibles).

Cette souplesse maximale assure le bon cadencement des entrées ainsi que la synchronisation fine des sorties.

B. Tests en "boîte blanche"

On parle de test en « boîte blanche » quand le moyen de test et la machine à tester rassemblent les conditions suivantes :

- Le moyen de test s'interface à la machine cible,
- Le moyen de test dispose d'un accès en lecture et en écriture aux variables internes du système à tester,
- Le moyen de test permet de piloter le démarrage d'un cycle sur la machine cible.

Il est alors possible de commander les différents cycles de fonctionnement du système à tester ou d'en forcer les variables.

C. Tests en "boîte grise"

On parle de test en "boîte grise" quand le moyen de test peut uniquement lire tout ou partie des variables du système à tester. Dans cette configuration, il est toujours possible de synchroniser le moyen de test à l'aide de variables internes du système à tester. Il faut au préalable que le système dispose de ce genre de variables (compteur de cycle d'exécution, état des acquisitions / émissions, etc.).

D. Tests en "boîte noire"

On parle de test en "boîte noire" quand le moyen de test n'a aucun accès aux variables internes du système à tester. Les

tests sont particulièrement complexes à mettre en œuvre dans ce type de configuration. Il est possible d'essayer de synchroniser le moyen de test et le système en dimensionnant leur temps de cycle à des valeurs proches (idéalement égales) et en les démarrant de manière synchrone.

Ce genre de configuration est néanmoins nettement plus sensible aux divergences sur les temps d'exécution.

La maitrise de l'exécution des tests sur une configuration "boîte noire" peut nécessiter une stratégie de génération des tests particulière afin de rendre le scénario compatible au glissement d'un cycle sur l'échantillonnage. Une telle stratégie est généralement plus couteuse.

Une alternative consiste à construire une surcouche au système à tester qui embarque les vecteurs de test et les sorties attendues dans le système à tester si ses ressources en offrent la capacité. On se retrouve alors dans une configuration "boîte blanche" mais cela nécessite de modifier le logiciel afin de le rendre compatible avec l'encapsulation.

V. Cas d'utilisation

Les cas d'emplois de cette technologie sont variés pour le domaine du logiciel critique.

Comme il s'agit de test par comparaison entre les sorties réelles d'un objet sous test à celles attendues dérivées d'un modèle, examinons les cas d'emploi selon l'origine de ce modèle.

A. Optimisation des compilateurs

Considérons d'abord que le modèle soit une image sémantiquement équivalente de celle d'un code source. Les différences observables à l'exécution des tests peuvent provenir de la plateforme d'exécution de ce code (ensemble de ressources matérielles, logiciel de base, système d'exploitation) ou du compilateur utilisé pour le transformer en logiciel exécutable. Les défauts rencontrés peuvent ensuite être reproduits et analysés pour identifier la cause racine. Dans le domaine du développement de logiciels critiques, (e.g. IEC 61508-3, IEC 62138), les normes exigent que les outils contribuant à la production du logiciel exécutable soient évalués vis-à-vis du niveau de confiance accordé par rapport à l'existence de défaillance affectant ce logiciel exécutable. Les compilateurs appartiennent à cette classe d'outils. On peut considérer que cette application du B2BT constitue une solution de vérification des sorties du compilateur à l'exécution sur la cible, capturant les défaillances ayant un impact sur les sorties du logiciel exécutable. Cette analogie vaut également vis-à-vis de la plateforme d'exécution mais aussi vis-à-vis d'outils de traduction entre une sémantique de plus haut niveau et un langage compilable.

En s'appuyant sur ce garde-fou, il est désormais envisageable de considérer le B2BT comme un moyen de maitrise d'outils du commerce. Une application pourra être de s'autoriser à exploiter plus largement la puissance des compilateurs, en particulier les options d'optimisation lorsqu'elles sont accompagnées de restrictions telles que "l'activation de l'optimisation lève la garantie sur l'ordre d'évaluation des paramètres d'une fonction" lorsque les tests B2BT sont déroulés avec succès.

B. Continuité des activités de vérification formelle

Considérons ensuite le cas où le modèle qui sert à générer les tests de B2BT n'est pas dérivé du code mais est élaboré dans le cadre d'activités de vérification formelle de propriétés. L'intérêt de la vérification formelle de propriétés est de démontrer mathématiquement que des exigences de sûreté de fonctionnement tiennent sur un modèle, quelle que soit la séquence d'entrées considérée et à quelque profondeur temporelle que ce soit dans la vie du système. Dans ce cas, l'apport du B2BT est de pouvoir démontrer que la machine sous test ne comporte aucune mutation observable par rapport à la sémantique d'exécution du modèle qui a supporté les activités de preuve. Quand les obligations de preuves du modèle formel sont démontrées comme prouvées, cette preuve tient sur la sémantique d'exécution du modèle et par transitivité sur celle de la machine sous test puisqu'elle ne contient pas de déviation observable. Dans ce cas, B2BT complète les stratégies de validation traditionnelles en permettant de démontrer le respect d'exigences de sûreté de fonctionnement par preuve sur un modèle formel associé à la démonstration de conformité à la sémantique sous-tendant la preuve sur la machine sous test par le bon déroulement des séquences B2BT. Une telle approche permet de considérer des approches de validation hybrides où la preuve prend à sa charge la démonstration du respect des exigences de sûreté de fonctionnement. Une approche par preuve apporte une couverture complète vis-à-vis de la sûreté de fonctionnement dans le sens où la preuve apporte une démonstration que quelle que soit la séquence d'évènements sur les entrées d'un modèle, une obligation de preuve tient. Par comparaison, une approche par test fonctionnel est fondée sur une approche par scénario. A l'exception de quelques situations rares où l'exhaustivité des combinaisons est possible, une approche par test assure une couverture par échantillonnage. La stratégie est donc d'élaborer une politique de validation composant une preuve face aux exigences de sûreté de fonctionnement avec des tests de B2BT générés automatiquement et quelques tests fonctionnels vis-à-vis d'exigences non critiques pour la sûreté de fonctionnement (ergonomie, performances, vivacité, etc.). La preuve apporte une couverture totale face aux exigences de sûreté de fonctionnement. Le B2BT consolide que les exigences de sûreté de fonctionnement tiennent à l'exécution par échantillonnage fin du respect de la sémantique d'exécution. Il est notable que l'élaboration d'une preuve est moins coûteuse en charge de travail que l'élaboration d'un test en charge d'ingénierie. En effet, l'écriture d'une obligation de preuve consiste à formaliser le « QUOI » d'une exigence. Prenons l'exemple d'une exigence de réaction sur la désactivation d'une sortie S sur dépassement par une grandeur G d'un seuil critique Sc en présence de certaines conditions C. Avec un modèle manipulant S, Sc et C, l'écriture formelle de l'obligation de preuve est immédiate et les outils de preuve sont chargés de trouver la preuve (la charge humaine complémentaire éventuelle étant consacrée à baliser le chemin de preuve par des lemmes intermédiaires lorsque le prouveur ne parvient pas à trouver seul la démonstration). Dans le cas d'une approche par test, il faut, par ingénierie de test et une connaissance approfondie du fonctionnel, créer les conditions préalables au test pour amener le système sous test dans un état où le test aura un sens. Dans le cas jouet proposé, il s'agit de décrire pour l'environnement de validation chaque étape du scénario qui amènera le système dans la situation où G dépassera Sc tout en respectant C. Ce simple exemple montre que la preuve propose une couverture totale (quelles que soient les séquences de valeur sur les entrées, alors la propriété de sûreté de fonctionnement est respectée) là où le test échantillonne (dans l'approche par scénario, on ne teste qu'une trace du système par rapport à une quantité énorme, si

ce n'est infinie). Il démontre également que l'effort d'ingénierie associé à la preuve est plus efficace et plus robuste aux évolutions que l'effort d'ingénierie au service des tests (la preuve ne s'occupe pas de chercher les scénarios mais d'établir un invariant en toute circonstance et la recherche est déléguée à la machine).

C. *Traitement de l'obsolescence*

Considérons enfin le cas où le modèle est dérivé d'une version de logiciel d'un équipement obsolète. Il est courant dans l'industrie qu'un équipement qualifié dans un passé plus ou moins récent ne puisse plus être maintenu, faute de pièces de rechange disponibles voire même d'environnement de test encore opérationnel (serveurs de test sous des OS caduques que les politiques SI bannissent ou banniront à raison des réseaux en raison des vulnérabilités qu'elles présentent). Dans le cadre du traitement d'obsolescence de tels équipements, il est possible de générer par l'approche B2BT des séquences de test de conformité à la sémantique d'exécution du code obsolète. Ces tests proposent une couverture plus fine que des tests de validation par rapport à cette gestion d'obsolescence dont on sait que l'équipement obsolète les a passés avec succès : le bon déroulement de ces tests démontre que la nouvelle génération de l'équipement n'a pas d'altération sémantique par rapport à la sémantique d'exécution de la version qualifiée mais obsolète. Dans l'hypothèse où son propriétaire ne dispose pas ou plus des moyens de test de validation de l'époque, les tests B2BT apportent une solution automatique pour reconstruire un référentiel de qualification de la nouvelle génération. Cette approche est à pondérer par rapport à une gestion d'obsolescence assortie d'un ensemble d'évolutions fonctionnelles. Il faudra dans ce cas procéder en 2 étapes : qualifier le traitement d'obsolescence par le B2BT puis ajouter un ensemble de tests ou de preuves dédiés aux évolutions fonctionnelles.

Cette approche de traitement d'obsolescence peut se généraliser plus globalement à la maîtrise de portages vers de nouvelles plateformes d'exécution d'un logiciel antérieur.

D. *Validation d'un système critique*

La perspective d'emploi la plus ambitieuse sera d'utiliser la technologie B2BT couplée à un modèle réalisé par une équipe de validation pour un système critique. En effet, dans cette configuration, ce sera désormais l'équipe de validation à travers son modèle qui impose la sémantique d'exécution de l'objet conforme au besoin du client sans pour autant descendre dans la complexité des moyens de production du logiciel exécutable ou de la plateforme d'exécution de sécurité. Ce dernier cas d'emploi propose une stratégie vis-à-vis de l'intégration de composants du commerce au sein de contrôle commande critiques. En effet, en s'assurant que la sémantique d'exécution proposée par un calculateur de sécurité du commerce et son environnement de programmation reste sans mutation par rapport à celle d'un modèle du besoin, le B2BT apporte des éléments de réponse sur l'aptitude à l'emploi d'une solution calculateur de sécurité + environnement de programmation fourni en boîte noire avec des conditions d'utilisation sûres.

VI. APPLICATION INDUSTRIELLE : PARTENARIAT AVEC EDF

A. *Mise en œuvre sur les modèles EDF*

En partenariat avec EDF, Systerel a mis en œuvre l'outil B2BT sur la base d'un modèle Lustre fourni par EDF. Ce modèle a été développé pour correspondre aux applications-type utilisées dans certains systèmes de commande et contrôle d'une tranche nucléaire. L'objectif est ainsi d'évaluer l'applicabilité de la méthode de B2BT aux applications nucléaires. Plusieurs étapes successives ont été mises en œuvre afin de permettre l'utilisation de ce modèle pour effectuer la génération des séquences de test B2BT.

En termes de sanction, Systerel et EDF ont travaillé à la définition de critères de couvertures haut niveau. Trois critères de couverture haut niveau ont été retenus pour vérifier que les tests générés par la solution B2BT couvrent bien les comportements particulièrement sensibles et critiques de l'application. Bien que ces critères ne soient pas détaillés pour des raisons de confidentialité, on peut néanmoins noter qu'ils ont été spécifiquement définis pour ce projet. Le parcours des séquences de test, au regard de ces critères haut niveau, permettra alors d'augmenter le niveau de confiance quant à la couverture des tests par B2BT : non seulement, toutes les mutations sont testées, mais tous les chemins critiques ont bien été parcourus.

Ces étapes sont brièvement décrites ci-dessous :

1) *Prétraitement des fichiers d'entrée*

Les fichiers d'entrée, au format Lustre V6 pour les besoins de l'étude, nécessitent une traduction vers le langage HLL,

2) *Analyse de la bonne définition du modèle*

Avant de lancer la génération des séquences de test, une passe de vérification de bonne conception du modèle est réalisée à l'aide du moteur d'analyse S3. Cette vérification consiste principalement à s'assurer que le modèle initial est sain en confirmant l'absence, par exemple, d'accès aux variables en dehors des bornes définies ou encore de division par zéro.

3) *Génération des séquences de test*

Une fois le modèle traduit et vérifié, il peut être transmis à l'outil de génération de séquence de test B2BT qui va appliquer les mutations et produire les séquences de test permettant de détecter les éventuels défauts.

4) *Mesure du taux de couverture selon les critères définis*

La séquence de test générée est appliquée au modèle de l'application afin de mesurer la valeur du taux de couverture atteinte pour chacun des critères de couvertures définis.

5) *Exécution de la séquence de test*

Cette dernière étape consiste à jouer la séquence de test sur l'équipement cible. Il est à noter que dans le cas où l'étape 1 introduirait des fautes sur la sémantique, alors celles-ci seront détectées lors de l'exécution du test, sauf cas de fautes combinées.

Du fait de l'existence de temporisations longues (plusieurs centaines de cycles) dans l'application étudiée et de l'explosion de l'espace d'état inhérente à l'utilisation d'un moteur de preuve à des profondeurs importantes, le modèle a été outillé afin de permettre le pilotage des mémoires et des temporisations.

A ces deux types de génération de test B2B s'ajoute une génération de séquence de test orientée sur des objectifs de test de plus haut niveau que ceux de la stratégie par défaut.

B. *Résultats*

1) *Génération B2BT du modèle standard*

L'analyse B2BT du modèle standard a été menée suivant deux phases complémentaires. La première d'une durée de

12h, consiste à la définition de pas de test aléatoires et permet d'obtenir la majeure partie de la couverture. Une seconde, d'une durée de 3 jours, utilise le moteur d'analyse S3 pour aller chercher tous les objectifs de test non encore couverts.

A l'issue de la génération, nous avons obtenu 77 scénarios totalisant 21606 pas de temps et couvrant 5928 objectifs de test B2BT sur les 6719 que compte le système. En tenant compte des 437 objectifs supposés non atteignables (voir partie suivante), on obtient un taux de couverture sur la détection des opérateurs mutés de 94,6%.

Concernant les critères de couverture de haut-niveau, 561 objectifs sur les 566 que comporte le modèle de l'application ont été couverts. Soit un taux de couverture de 99.1 %. Il est notable que les séquences générées selon le critère de couverture B2BT, foncièrement différent de ceux haut niveau, assure un taux de couverture très satisfaisant également sur ces critères haut-niveau.

2) Génération B2BT du modèle outillé

L'analyse B2BT du modèle outillé (mémoire et temporisation pilotables) prend environ 8 min et produit 10 scénarios totalisant 87 pas de temps et couvrant 5782 objectifs de couverture B2BT sur les 6219 que compte le système (93%). Par ailleurs, une exploration jusqu'à une profondeur de 50 cycles n'a pas permis de couvrir les 437 objectifs non couverts ce qui laisse à supposer qu'ils sont, tout ou partie inatteignables.

L'analyse des critères de couverture de haut-niveau sur les scénarios obtenus indique que 544 objectifs sur les 566 ont été couverts. Soit un taux de couverture de 96.1 %. Cette différence par rapport au modèle B2B standard est due au fait que certains des critères haut-niveau sont séquentiels. En effet, le fait que les mémoires et temporisations soient pilotables réduit très sensiblement le nombre de pas de test successifs nécessaires pour atteindre un objectif et donc les chances de produire les séquences attendues pour ces critères.

Cet outillage réduit donc drastiquement le temps de calcul et le nombre de pas de test tout en améliorant sensiblement la couverture des tests. Mais cette solution nécessite cependant de pouvoir piloter ces mémoire et temporisations pendant l'exécution de la séquence de test.

3) Génération de séquences selon les critères de couverture haut-niveau seuls

Les trois critères de couverture de haut-niveau, appliqués à l'application étudiée, donnent un ensemble de 566 objectifs de test. Le moteur d'analyse S3 trouve 86 scénarios totalisant 3230 pas de temps en 1,5 min, couvrant 563 des objectifs de test sur les 566 existants. Il a par ailleurs été possible de prouver que les 3 objectifs de test non couverts n'étaient pas atteignables.

Bien que cette solution offre le meilleur résultat en terme de temps de génération, de longueur de séquence de test et de couverture des critères haut niveau, cette solution ne teste que les chemins critiques et pas chacun des opérateurs. Elle ne garantit donc pas le même niveau de complétude du test réalisé et ne permet donc pas de détecter les implémentations dont le comportement observable non-critique pourrait avoir été altéré lors du codage ou de la compilation.

C. Conclusion

Ces différentes expérimentations ont montré que cette méthode de génération automatique de séquences de test permet d'adresser efficacement des problèmes de validation de systèmes industriels et de prendre en compte les contraintes matérielles (banc de test standard ou outillé) et de complexité (génération par mutation des opérateurs ou par critères haut niveau uniquement) pour l'exécution des séquences de test.

VII. Application industrielle : postes d'aiguillage sur automates HIMA

Dans le cadre d'un projet de conception de postes d'aiguillage pour tramway, nous avons cherché à réaliser une implémentation logicielle d'un système d'enclenchement pour tram sur un automate HIMAtrix F30 03, développé par HIMA. Ces automates ont la particularité d'être certifiés SIL4 par rapport aux normes EN 50126, EN 50128 et EN 50129 pour des applications de sureté.

Le système, une ligne complète de tramway, est divisé en stations / zones indépendantes. Pour cette raison, une architecture décentralisée avec un automate par zone a été sélectionnée. Enfin, dans le but de faciliter les activités d'évaluation de ces sous-systèmes décentralisés, la stratégie de développement suivante a été envisagée :

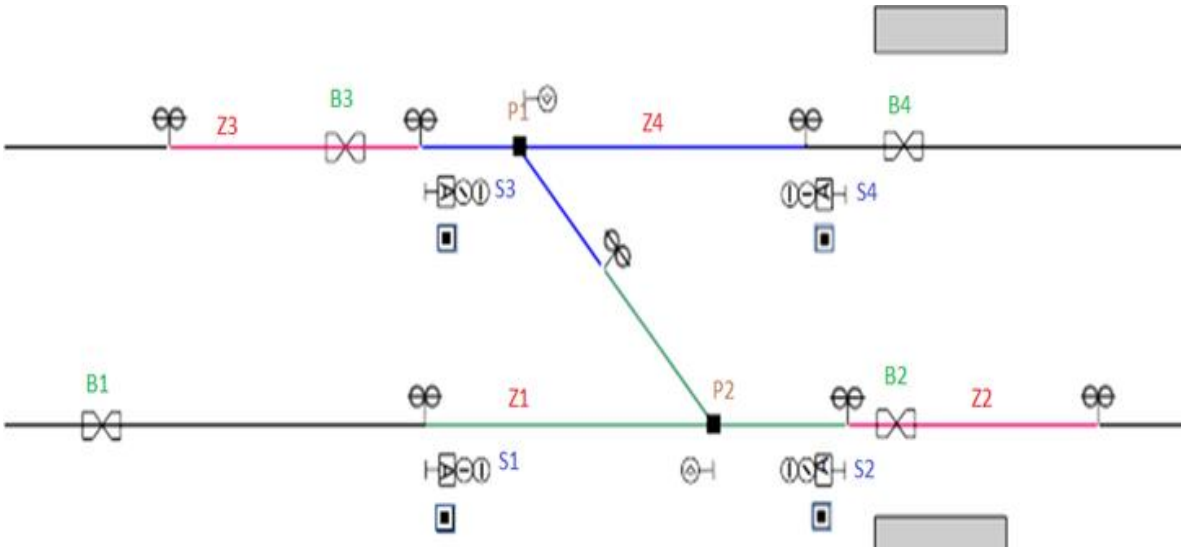

Fig. 3. Plan de voie

- Développement d'une bibliothèque générique de principes de signalisation pour tramway sous forme de Sequential Function Chart (SFC), un langage graphique de programmation des Automates Programmables Industriels (API) défini dans la norme IEC 61131-3,
- Développement des applications spécifiques à chaque station / zone à l'aide de Function Block Diagram (FBD), un autre langage graphique pour API défini dans la norme IEC 61131-3.

Afin de tester ces applications spécifiques, nous avons choisi d'utiliser les techniques de B2BT. Les premiers essais ont porté sur le développement de l'application spécifique pour une zone de signalisation. Cette zone, composée de 4 signaux et de 2 appareils de voie, permet la création de 4 itinéraires (dont 2 pour le changement de voie). Le plan de cette zone est décrit dans la Fig. 3.

A partir des principes de signalisation sous forme de SFC, le comportement attendu de cette zone a été implémenté dans l'automate HIMA.

En parallèle, un modèle HLL de cette application a été réalisé, dans le but de :

1) Prouver la tenue des propriétés de sûreté sur le modèle,

2) Générer les séquences de test B2BT à partir du modèle,

3) Exécuter ces séquences de test sur l'application embarquée.

Lors de l'exécution des séquences de test, une erreur d'implémentation a été rapidement détectée. Le séquencement de l'exécution des principes instanciés ne s'effectuait pas dans le même ordre sur la cible et sur le modèle. Cette différence a modifié le comportement observable du système, puisque l'ordre dans les priorités d'exécutions des principes n'était plus respecté et a bien été détectée lors de l'exécution des tests.

Cette expérimentation a donc permis de mettre en avant une mauvaise maîtrise de l'environnement de développement de l'automate.

VIII. CONCLUSIONS ET PERSPECTIVES

Les résultats que nous avons obtenus lors de la mise en œuvre de notre solution de B2BT sont particulièrement satisfaisants. Ils nous ont permis :

- De consolider la confiance que nous avons dans notre stratégie de génération des séquences de test, dont les taux de couverture sont très élevés, y compris lorsqu'on cherche à vérifier des critères spécifiques à une implémentation,
- De détecter des erreurs d'implémentations pour lesquelles le comportement observable du système à tester était différent du comportement attendu,
- De reporter les résultats de preuves formelles obtenus à partir des modèles sur les implémentations logicielles des systèmes.

Il reste des limitations dans les cas d'usage de notre solution. En particulier, il serait intéressant d'enrichir la stratégie de génération des séquences de test pour pallier le manque de couverture sur les modèles contenant des temporisations. Concernant la mise en œuvre, le test des systèmes en "boite grise" et "noire" reste complexe.

Il existe des pistes prometteuses face à ces limitations :

- Définir des stratégies de génération spécialisées autour des temporisations ; à noter que cet axe sera intéressant à construire conjointement avec la capacité à abstraire le temps au niveau des plateformes d'exécution pour gagner également sur le temps d'exécution des tests,
- Travailler les moyens de mise en œuvre par rapport au pilotage fin de l'exécution des séquences ou proposer des alternatives selon les capacités des machines sous test,
- Travailler la traduction des systèmes pour isoler plus précisément le contexte mémoire qui permet de rendre l'approche plus massivement combinatoire pour les machines qui permettent de piloter la mémoire d'un cycle sur l'autre.

REFERENCES

[1] Simon H., Friedrich N., Biallas S., Hauck-Stattelmann S., Schlich B. & Kowalewski S, « Automatic test case generation for PLC programs using coverage metrics », 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA) (2015), p.1-4

[2] Enoiu E. P., Čaušević A., Ostrand T. J., Weyuker E. J., Sundmark D. & Pettersson P, « Automated test generation using model checking: an industrial evaluation », International Journal on Software Tools for Technology Transfer 18(3) (2016), p.335-353

[3] Enoiu E. P., Sundmark D. & Pettersson P., « Model-based test suite generation for function block diagrams using the uppaal model checker », 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (2013), p.158-167

[4] Nicolas Breton. High Level Language - Syntax and Semantics - Logical Foundation Document. [Technical Report] C672 pr4.0 rc1 A, Systerel. 2021. ⟨hal-03356342⟩

[5] Breton, N., Fonteneau, Y. (2016). S3: Proving the Safety of Critical Systems. In: Lecomte, T., Pinger, R., Romanovsky, A. (eds) Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification. RSSRail 2016. Lecture Notes in Computer Science(), vol 9707. Springer, Cham.