Vérification formelle des programmes automates de la SNCF par Model-Checking

Mohamed Niang, Philippot Alexandre, François Gellot, Raphaël Coupat, Bernard Riera, Sébastien Lefebvre

Vérification formelle des programmes automates de la SNCF par Model-Checking

Mohamed NIANG [1&2], Alexandre PHILIPPOT[1], François GELLOT[1], Raphaël COUPAT[2], Bernard RIERA[1] et Sébastien LEFEBVRE[2]

[1] CReSTIC (EA3804), Université de Reims Champagne Ardenne, Moulin de la Housse, BP 1039, 51687 Reims CEDEX 2, France

[2] IP.TE (CES), Direction de l'ingénierie, Société Nationale des Chemins de fer Français, 6 avenue François Mitterrand – 93574 La plaine Saint Denis CEDEX, France

mohamed.niang@etudiant.univ-reims.fr,
{francois.gellot, alexandre.philippot, bernard.riera}@univ-reims.fr,
{raphael.coupat, s.lefebvre}@reseau.sncf.fr

Résumé

Suite à l'ouverture du marché, le Réseau Ferré National (RFN) français doit faire face à la concurrence du transport ferroviaire européen. Afin de garder sa position de leader du marché ferroviaire français, la Société Nationale des Chemins de Fer français (SNCF) cherche à mettre en place des solutions innovantes permettant d'améliorer la sécurité et les conditions de travail de ses chargés d'études lors des travaux d'automatisation. Partant de l'étude théorique d'un projet jusqu'à sa validation sur site, les chargés d'études doivent effectuer un certain nombre de tâches qui s'avèrent souvent être longues, complexes et répétitives, ce qui a pour effet d'augmenter leurs charges de travail. Ce projet de recherche se focalise principalement sur une des tâches les plus importantes, qui est la vérification des programmes automates (aspects fonctionnels et sécuritaires). L'objectif est d'améliorer la méthodologie de vérification des programmes automates en mettant en place un outil qui permettra aux chargés d'études d'effectuer cette tâche de manière efficace et automatique.

1 Introduction

Pour un système automatisé, la vérification des programmes automates demeure une étape incontournable dans la validation du système de contrôle commande de l'installation. Nombreux sont les accidents qui se produisent en industrie suite à une mauvaise qualité des programmes, ce qui conduit donc à la nécessité de les vérifier avant leur implémentation sur le système réel. Cette vérification concerne aussi bien l'aspect fonctionnel du programme automate (ou programme API)

que son aspect sécurité. La vérification de l'aspect fonctionnel du programme permet de s'assurer que le programme automate est en accord avec les spécifications fonctionnelles décrites dans le cahier des charges, et celle de l'aspect sécurité consiste à vérifier si le système commandé est susceptible ou non d'atteindre un état dangereux qui pourrait entrainer une détérioration du matériel, voire pire.

A la SNCF, les chargés d'études ont actuellement recours à la méthode des tests et simulation pour valider le contrôle commande des EALE (Equipements d'Alimentation des Lignes Electrifiées). Pour cela, ils disposent d'un cahier de recette qui contient un ensemble d'instructions séquentielles qu'ils doivent manuellement exécuter sur le système en usine. Ce travail de vérification est effectué une fois que le contrôle commande de la sous-station a été conçu (programmes automates et câblage des armoires électriques) et avant son implémentation sur site. Il leur permet de vérifier et de valider en même temps l'aspect fonctionnel et sécurité du contrôle commande.

Toutefois cette méthode de vérification n'est pas automatique et prend beaucoup de temps pour être effectuée par les chargés d'études (1 à 2 semaines), car le cahier de recette renferme en général plus de 80 pages d'instructions à réaliser. Une telle charge de travail a pour effet d'augmenter le stress et la fatigue chez les chargés d'études, qui ne sont donc pas à l'abri d'erreurs pouvant compromettre la validité des tests réalisés. De plus, rien ne semble prouver que le cahier de recette est exhaustif, bien qu'il ait servi à valider le contrôle commande des installations de la SNCF depuis plusieurs décennies et que l'expérience ait montré qu'il semblait suffisant pour garantir la sureté et le bon fonctionnement de l'installation. A travers l'utilisation du Model-Checking, l'objectif de ce projet de recherche est d'optimiser et d'automatiser la méthodologie actuelle de vérification des programmes automates de la SNCF, à l'occasion des études de conception du système de contrôle/commande/protection des projets d'électrification menés par la section Conception et Expertise des Systèmes (CES2) de la Direction de l'Ingénierie de la SNCF.

Après une présentation du contexte général en section 2, nous détaillons le principe de notre approche puis nous l'appliquons sur un exemple en section 3. L'exhaustivité du cahier de recette et la génération automatique des modèles sont étudiées en section 4, puis nous terminons par une conclusion en section 5.

2 Contexte général

Les EALE sont les points d'alimentation en énergie électrique des lignes électrifiées, appelées caténaires. Leur rôle est de transformer, de distribuer et de redresser dans le cas d'une alimentation continue, la tension du réseau Haute Tension (HT) en tension compatible avec les engins de traction (1500 V continu, 25 kV ou 2 x 25 kV alternatif). Ces systèmes électriques sous (très) Haute Tension sont soumis à des normes strictes de sécurité ferroviaire (EN 50126, 2012) (IEC 60870-4, 2013).

Les EALE sont des systèmes automatisés distribués dont le contrôle et la commande peuvent se faire soit localement, soit à distance dans un poste de commande centralisé appelé Central Sous-Station (CSS). Les opérateurs d'exploitation peuvent manœuvrer les appareils HT depuis cette salle de contrôle. Ils ont la responsabilité d'assurer l'alimentation des EALE en régime nominal et dégradé (maintien de la tension caténaire), d'assurer la sécurité en cas de travaux sur les EALE sous réglementation nationale (UTE C 18510, 1988) ou particulière et la coupure d'urgence en cas de danger électrique pour les personnes et les biens.

Les EALE sont principalement composés de trois éléments de parties opératives (Haute Tension HT, Groupe Traction GT, Départ Traction DT) appelés également ElPO, chacun étant commandé par un automate programmable industriel (API). Chaque ElPO est également constitué de sous-ensembles de parties opératives (disjoncteurs, sectionneurs, transformateurs, etc…) appelés SePO. La figure 1 présente la composition d'un exemple d'EALE (sous-station électrique à courant continu).

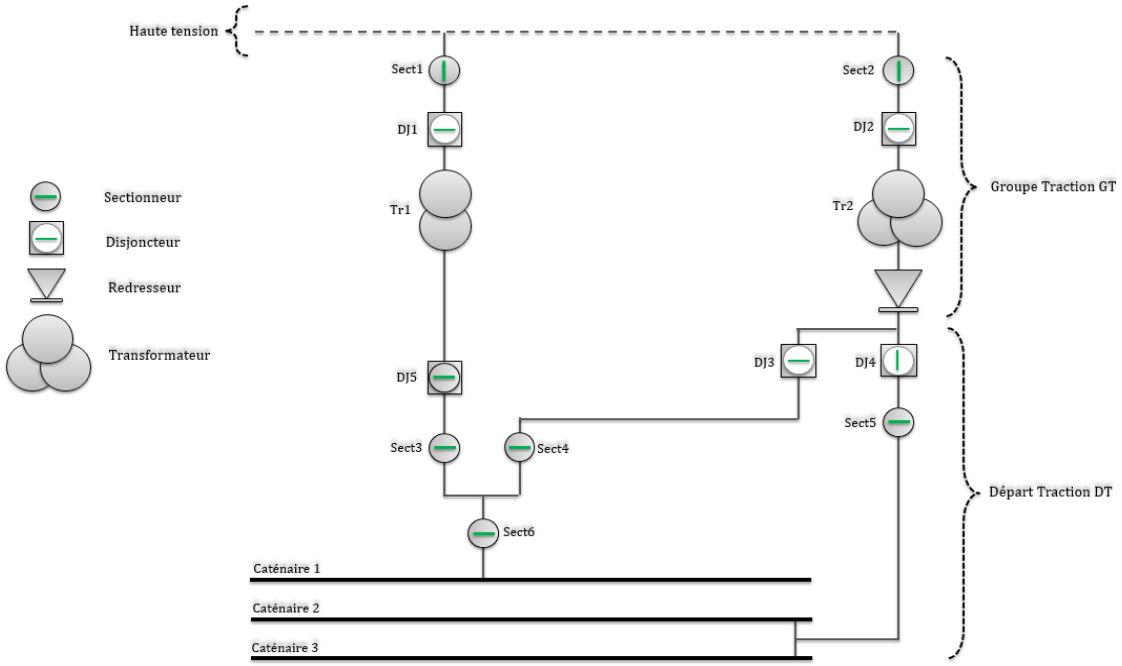

Figure 1 : Composition d'une sous-station électrique

Depuis plusieurs décennies, les cahiers de recette ont toujours été considérés comme des documents de références fiables permettant de vérifier toute la partie contrôle/commande/protection des installations électriques de la SNCF. Ce sont des documents qui ont été conçus par les experts grâce à des retours d'expériences sur le fonctionnement (en mode normal et en mode dégradé) des installations, et qui contiennent un ensemble d'instructions réparties sur plusieurs dizaines de pages. Chacune d'elles permet d'exécuter un scénario de tests (voir exemple dans le tableau 1) sur le système

Réaction du disjoncteur après un court-circuit	
Conditions initiales	
Sectionneur Sect1 et Disjoncteur DJ1 fermés	
Présence tension	
Absence de défauts	
Tests	
Instructions	**Résultats attendus**
1. Créer un défaut "court-circuit"	**1**. Ouverture de DJ1, entrée "Imax" de l'API activée
2. Refermer le disjoncteur DJ1	**2**. Néant (aucune réaction)
3. Arrêter le défaut puis refermer DJ1	**3**. Entrée "Imax" desactivée, fermeture de DJ1

Tableau 1 : Exemple de scénario de tests

de contrôle/commande/protection, et de vérifier si le comportement de celui-ci est conforme à leurs attentes (au niveau fonctionnel et sécuritaire). Par conséquent, tout système de contrôle/commande/protection qui satisfait entièrement à l'ensemble des scénarios du cahier de recette sera considéré comme valide.

Durant les études d'automatisation des EALE, les livrables métiers (cahier de recette, programmes automates,…) peuvent être automatiquement générés par un logiciel nommé ODIL (Coupat, 2014). Pour cela, le chargé d'études renseigne toutes les données d'entrées de l'installation étudiée (telles

que le schéma unifilaire de l'installation, la configuration matérielle des équipements, la répartition de la commande par les automates, les entrées/sorties, …) et ensuite, ODIL génère automatiquement les programmes automates et le cahier de recette associé. L'objectif du projet de recherche de (Coupat, 2014) consistait donc à écourter la durée des projets d'automatisation en accélérant le processus de définition des livrables métiers, mais n'avait aucun apport quant à l'optimisation du temps de vérification des programmes API et du câblage des armoires de contrôle/commande/protection.

Après une relecture des livrables (générées ou non) pour vérifier leurs contenus, les chargés d'études ne pourront tester réellement la partie commande qu'en usine. En effet, lorsqu'ils vont réceptionner les armoires de contrôle/commande/protection qu'ils auront préalablement commandés chez l'intégrateur, ils vont transférer tous les programmes API dans les automates correspondants, ainsi que les paramétrages des appareils de protections numériques. Grace à des simulateurs d'appareils (disjoncteurs, sectionneurs, …) connectés aux armoires électriques, ils pourront effectuer les essais du cahier de recette. La partie contrôle/commande/protection ne peut donc être vérifiée et validée qu'en usine, et cela prend en général une à deux semaines au(x) chargé(s) d'études en charge du projet, pour en vérifier l'intégralité. De plus pendant les essais, ils subissent certaines contraintes qui ont pour effet d'affaiblir leur rendement :

- la complexité du debuggage des problèmes,
- le stress lié au temps imparti pour terminer les essais,
- les essais non automatisés,
- la fatigue,
- les erreurs humaines qu'ils peuvent faire pendant les essais.

Ces contraintes sont celles que rencontrent souvent les chargés d'études pendant la recette usine, particulièrement la première qui consiste à trouver l'origine d'une anomalie détectée pendant les essais. Elle représente une des principales raisons qui ralentit les tests en usine. En effet, lorsqu'une instruction d'un scénario n'est pas satisfaite lors des essais en usine, ils doivent l'analyser pour en trouver la cause (liée aux programmes API ou au câblage des armoires). Par exemple pour le scénario de tests décrit dans le tableau 1, lorsque le disjoncteur DJ1 ne s'ouvre pas suite à l'apparition du défaut de court-circuit (instruction N°1 de la fiche), deux raisons peuvent justifier ce problème :

- l'entrée « Imax » associé à la présence du court-circuit, n'est pas câblée sur l'automate (ou bien elle n'est pas câblée sur la bonne entrée) : donc l'automate ne reçoit pas l'information. On en déduit que le problème est lié à un mauvais câblage.
- L'entrée « Imax » a été correctement câblée sur l'automate, mais le programme du disjoncteur DJ1 n'a pas pu bien exploiter cette information pour faire ouvrir le disjoncteur associé : par conséquent le problème est lié à une mauvaise programmation de la commande du disjoncteur.

Malgré l'expérience acquise au fil des recettes usines effectuées, ce diagnostic peut s'avérer complexe et peut prendre énormément de temps dans certains cas. Toutefois, il pourrait être moins complexe si les programmes API pouvaient être vérifiés bien avant la recette usine, et de ce fait tout problème rencontré pendant les essais en usine proviendrait logiquement d'un mauvais câblage des armoires. Actuellement les chargés d'études de la SNCF ne disposent pas de méthodes pour vérifier les programmes API avant la recette usine.

La démarche que nous proposons dans ce travail de recherche a pour objectif de diviser l'étape de vérification du contrôle/commande/protection des EALE en deux phases :

- Vérification hors ligne des programmes API des EALE par model-checking (Katoen et al., 2008);

- Vérification en ligne du câblage des armoires de contrôle/commande/protection des EALE.

Le principe de la vérification hors ligne des programmes API des EALE reste le même que celui utilisé actuellement par les chargés d'études, c'est-à-dire que tout programme API qui satisfait à l'intégralité des tests du cahier de recette sera considéré comme valide. Le model-checker UPPAAL (Behrmann et al., 2002) est utilisé pour vérifier si le système (modèle EALE + modèle programmes API) satisfait à la propriété suivante : peut-on exécuter sur le système toutes les instructions du cahier de recette (dans l'ordre) jusqu'à atteindre la dernière, sans trouver aucune instruction non satisfaite (ou bloquante) ? Pour cela, nous devons partir d'une modélisation dans UPPAAL de la partie opérative (EALE) des programmes API et du cahier de recette, et faire en sorte que pendant la vérification formelle toutes les instructions du cahier de recette soient automatiquement exécutées sur le système. Toujours pendant la vérification, UPPAAL va contrôler l'instruction en cours d'exécution et s'assurer que le résultat attendu (associé à cette instruction) soit obtenu avant un certain délai, au-delà duquel l'instruction sera considérée comme bloquante. Dans ce cas, la simulation s'arrête (ainsi que l'évolution du cahier de recette) et le logiciel nous retourne la fiche et le numéro de l'instruction bloquante. Le chargé d'études connaitra non seulement l'instruction n'ayant pas été satisfaite, mais pourra également s'aider de la trace retournée par le model-checker pour analyser l'état de la partie opérative, des programmes, et du cahier de recette, à n'importe quel stade de la vérification (de la première jusqu'à la dernière instruction exécutée). Une fois que le problème est corrigé, il pourra relancer la vérification sous UPPAAL pour rechercher d'autres blocages s'il en existe. Cette méthode lui permet en définitive de vérifier les programmes API suivant les mêmes principes actuels, mais de manière automatique et rapide, et lui permet également de faciliter la recette usine à venir.

La deuxième phase de vérification du contrôle/commande/protection repose sur une vérification en ligne du câblage des armoires électriques lors de la recette usine. Une fois que les programmes API ont été validés, le chargé d'études devra utiliser un logiciel simulateur de partie opérative, connecté aux armoires de contrôle/commande/protection, et contenant une version numérisée du cahier de recette. A la différence de la première phase, la vérification (toujours automatique) sera effectuée en ligne avec tous les équipements des armoires pour tester : le câblage, les paramétrages et les réglages des protections numériques. Ce document traite uniquement la première phase de vérification du contrôle/commande/protection des EALE.

Pour mettre en place la méthode de vérification avec UPPAAL, il est d'abord nécessaire de modéliser la partie opérative, les programmes, et les fiches de recette. La section suivante présente la méthode de vérification des programmes mise en place.

3 Vérification formelle des programmes API

Avant d'obtenir les modèles, une analyse structurelle et fonctionnelle du système (Partie Opérative + programmes + cahier de recettes) a été minutieusement effectuée avec l'aide des chargés d'études. Les modèles doivent avoir le même comportement que le système réel, sans quoi l'efficacité de la méthode serait remise en cause. Les modèles ont donc été vérifiés et testés maintes fois sur des programmes simples. Dans cette section, nous présentons uniquement les modèles du groupe traction d'une sous-station nommée « Invalides », que nous utilisons pour illustrer la méthode de vérification.

3.1 Modélisation de la partie opérative : le groupe traction 1500V

Un groupe traction 1500V est principalement constitué des sous-ensembles de parties opératives (SePO) suivants : un disjoncteur, un sectionneur, un transformateur et un redresseur. Ces objets ne

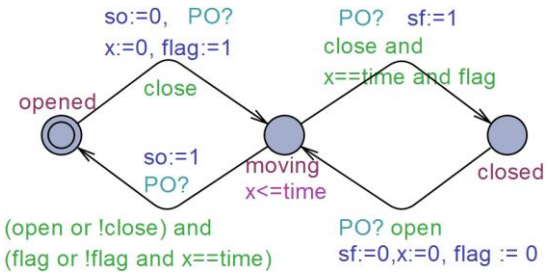

Figure 2 : Modèle d'un sectionneur à commande permanente

sont pas propres au groupe traction, mais peuvent également être trouvés dans d'autres parties de l'installation. L'idée est donc de modéliser tous les sous-ensembles de parties opératives que l'on peut trouver dans une sous-station électrique, puis d'instancier les objets dont on a besoin pour représenter le modèle du groupe traction : seuls les modèles utilisés dans cet exemple sont donc présentés.

Le modèle présenté en figure 2 représente le sectionneur du groupe traction étudié. Il est du type « à commande permanente » car pour fermer l'appareil il est nécessaire de maintenir la commande de fermeture tant que l'appareil n'est pas fermé (à l'opposé du sectionneur à commande impulsionnelle pour lequel un front descendant de la commande de fermeture est nécessaire pour fermer l'appareil).

Dans le modèle du sectionneur, les variables « ***so*** » et « ***sf*** » représentent respectivement les capteurs de position « ouvert » et « fermé ». La variable « ***close*** » (respectivement « ***open*** ») représente la commande de fermeture (respectivement d'ouverture) provenant du programme automate. La variable « ***flag*** » est égal à « **1** » pendant la fermeture et « **0** » pendant l'ouverture. Le modèle est constitué de 3 états, représentant les 3 états possibles de l'appareil :

- sectionneur ouvert ("***opened***") : (**so=true** et **sf=false**),
- sectionneur en mouvement ("***moving***") : (**so=false** et **sf=false**),
- sectionneur fermé ("***closed***") : (**so=false** et **sf=true**).

Initialement ouvert, l'appareil commence à se fermer dès qu'il reçoit la commande fermeture « ***close*** » provenant du programme. Le modèle passe alors à l'état « ***moving*** » tout en remettant à zéro l'information « ***so*** » et en initialisant la temporisation « ***x*** » à zéro. Une fois que celle-ci atteint la valeur de « ***time*** » (qui représente ici le temps d'ouverture ou de fermeture de l'appareil modélisé) et que l'ordre de fermeture est toujours maintenu, le modèle passe à l'état fermé tout en mettant à 1 le capteur « ***sf*** ». Par contre si depuis l'état « ***moving*** », la commande de fermeture a été désactivée (ou que l'ouverture a été activée), l'appareil reviendra à l'état ouvert. Lorsque l'appareil est fermé et qu'il reçoit la commande d'ouverture, il s'ouvre au bout d'un certain temps tant que la commande est maintenue. L'invariant « ***x<=time*** » a été rajoutée à l'état « ***moving*** » pour éviter que le sectionneur ne reste indéfiniment dans une position intermédiaire.

Pour instancier un sectionneur dans le modèle global, il est nécessaire de fournir en arguments : les capteurs de position de l'appareil (***so*** et ***sf***), les ordres d'ouverture et de fermeture (***open*** et ***close***), ainsi que le temps d'ouverture ou de fermeture du sectionneur (***time***).

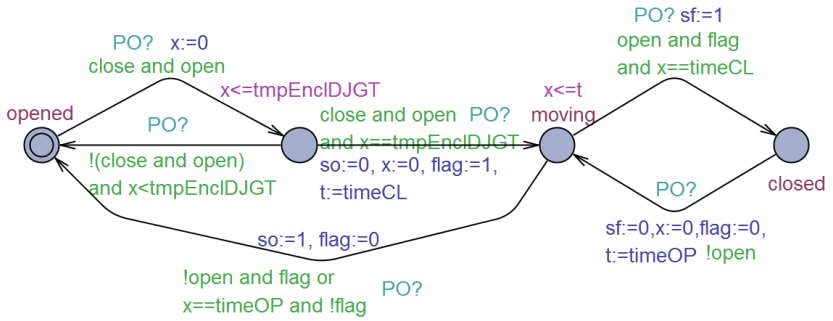

Figure 3 : Modèle d'un disjoncteur à commande impulsionnelle

Le modèle du disjoncteur à commande impulsionnelle est présenté à la figure 3. Il comporte quelques particularités par rapport au modèle du sectionneur, notamment :

- Pour fermer l'appareil, il faut d'abord activer les ordres ***open*** et ***close*** pendant un certain temps (=***tmpEnclDJGT***) pour provoquer l'enclenchement du disjoncteur. Ensuite il suffira juste d'assurer l'auto maintien de la commande « ***open*** » pour que l'appareil s'ouvre après un certain temps;
- Pour ouvrir l'appareil, il suffit de désactiver l'auto maintien de la commande « ***open*** » (même si la commande « ***close*** » était activée) ;
- Le temps d'ouverture de l'appareil (***timeOP***) est différent du temps de fermeture (***timeCL***)

En ce qui concerne le transformateur et le redresseur, même si ces derniers ont des comportements magnétiques et thermiques, nous ne prenons pas en compte ces données dans les modèles associés. Nous les considérons comme des objets non animés, mais dans lesquels différents types de défauts peuvent survenir (par exemple des défauts court-circuit, absence tension, surcharge, fusion fusible, augmentation température,…). Ils sont donc représentés sous forme de structure de variables booléennes dans UPPAAL.

A ce stade, tous les objets nécessaires pour composer le modèle de la partie opérative sont définis. Il nous reste à les instancier pour obtenir le modèle global du groupe traction.

3.2 Modélisation des programmes automates

Le groupe traction est commandé par un automate programmable du type ACS25 (Automate de Conduite Sécurisée) qui se programme avec le logiciel STRATON (www.copalp.com). Les langages de programmation principalement utilisés sont le LD (Ladder Diagram), et le SFC (Sequential Function Chart) (IEC 61131-3, 2003). Dans notre modélisation des programmes, il est nécessaire de convertir les programmes LD et SFC en langage ST (Structured Text), étant donné que celui-ci est interprétable par UPPAAL. Ce changement de format n'entrainera en rien une modification des données, car celle-ci est basée sur la norme de représentation algébrique des programmes SFC (Machado et al., 2006).

Pour simuler l'exécution d'un programme automate dans le model-checker UPPAAL, il est nécessaire d'avoir un modèle qui assure l'exécution des tâches classiques durant un cycle automate (lecture des entrées, exécution du programme, et mise à jour des sorties). Au démarrage de la simulation, le modèle de la figure 4 effectue une initialisation du système (initialisation des paramètres, positionnement initial des appareils, programmes SFC à l'étape initiale, …) pendant le premier cycle automate. Ensuite, il effectue de manière cyclique et dans l'ordre les tâches suivantes :

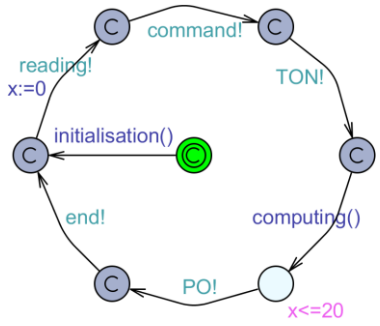

Figure 4 : Modélisation du cycle automate

- lecture des entrées (défauts transformateur et redresseur, position ouvert/fermé des appareils,…),
- lecture des commandes (envoyées par l'opérateur humain),
- évolution des temporisations (voir figure 5),
- exécution du programme principal et calcul des sorties,
- écriture des sorties calculées.

Le modèle dispose également d'une horloge ***x*** (limité à 20 unités de temps par cycle) qui permet de mesurer le temps écoulé durant chaque cycle automate. Pour réduire l'espace d'états afin d'éviter l'explosion combinatoire, la plupart des états du modèle précédent sont déclarés comme étant « ***committed*** », de manière à ce que le temps ne puisse évoluer que pendant l'exécution du programme principal.

Le programme automate utilise des temporisateurs du type TON (Timer On-delay) dont le principe de fonctionnement est détaillé dans la norme IEC 61131-3. Ce temporisateur comporte 2 entrées (une variable booléenne « ***in*** » pour démarrer ou arrêter le comptage, et une variable entière « ***PT*** » qui représente la base de temps) et 2 sorties (une variable booléenne « Q » qui passe à 1 lorsque le comptage est terminé, et une variable « ***x*** » qui donne la valeur en cours du compteur de temps). Depuis l'état initial « ***idle*** », le modèle de comptage passe à l'état « ***running*** » lorsque l'entrée « ***in*** » est activée. Lorsque celle-ci repasse à 0 avant la fin du comptage, le modèle revient alors à son état initial. Dans le cas contraire, la sortie « ***Q*** » passe à 1 et le modèle évolue vers l'état « ***timeout*** ».

Cet état est alors maintenu jusqu'à ce que l'entrée « ***in*** » soit désactivée. Le modèle de ce temporisateur est inspiré de (Mokadem et al., 2010).

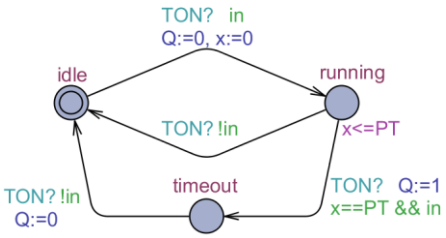

Figure 5 : Modélisation d'un temporisateur TON

Il arrive souvent qu'un programme SFC utilise 3 à 4 temporisateurs dans ses différentes étapes. De ce fait en instanciant autant de modèles de temporisateurs qu'il y en a dans le programme réel, cela risque d'alourdir le modèle général et de l'exposer à de fréquentes explosions combinatoires. Pour éviter cela, toutes les étapes temporisées d'un même programme SFC devront utiliser sans conflit une seule et même instance d'un modèle de temporisateur. Aucun des programmes SFC avec lesquels nous travaillons ne renferme de « divergence en ET » au niveau des transitions, ce qui signifie qu'aucun de ces programmes ne peut avoir plus d'une étape activée en même temps. Toutefois lorsque deux étapes successives du programme SFC utilisent la même instance de temporisateur, il faudra s'assurer qu'entre la désactivation de la première étape et l'activation de la suivante, le temporisateur est bien réinitialisé. Dans le cas contraire, il n'y aura pas de comptage de temps à la deuxième étape, car la sortie « Q » (maintenue activée à l'issu du premier comptage) validerait automatiquement la transition de la deuxième étape.

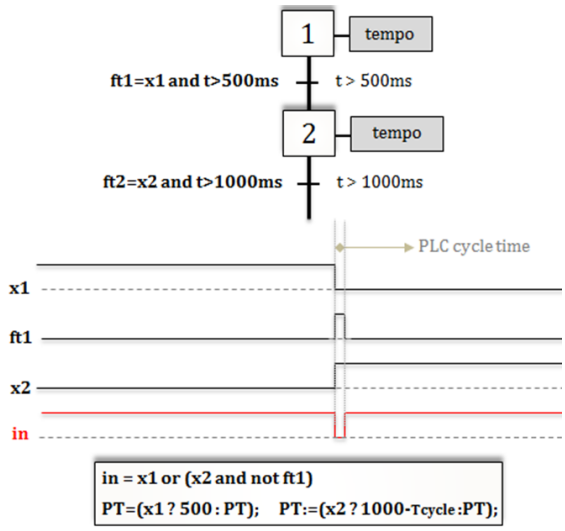

Figure 6 : Exemple d'utilisation du bloc TON dans un programme SFC

La figure 6 illustre le cas ci-dessus : les étapes **X1** et **X2** sont suivies de transitions ***ft1*** et ***ft2*** temporisées respectivement de 500ms et de 1000ms. En affectant à l'entrée « ***in*** » du temporisateur, la valeur « ***in*** = ***x1*** *or* ***x2*** » (ou ***x1*** et ***x2*** sont les valeurs booléennes représentant les états activés/désactivés des étapes **X1** et **X2**), elle ne repassera pas à 0 lors du passage de l'étape **X1** à l'étape **X2** (car la permutation d'étapes se fait instantanément). Pour remédier à cela, nous utilisons l'information ***ft1*** dans l'expression de l'entrée booléenne du temporisateur pour aboutir à « ***in*** = ***x1*** *or (****x2*** *and not* ***ft1)*** ». De ce fait lorsque l'étape **X1** est activée (➔ PT=500ms, voir figure 6), la temporisation démarre jusqu'à atteindre les 500ms puis la sortie Q et donc la transition ***ft1*** passent de 0 à 1 : l'étape **X1** devient alors inactive, au moment où l'étape **X2** s'active (➔ PT=1000ms). Par conséquent, l'entrée « ***in*** » et donc la sortie « **Q** » passent de 1 à 0, ce qui ne pouvait pas être possible avec l'ancienne expression du paramètre d'entrée « ***in*** ». Au cycle automate suivant, la transition ***ft1*** repasse à 0 (car l'étape **X1** n'est plus active) ce qui entraine le passage de l'entrée « ***in*** » de 0 à 1 ; la temporisation redémarre alors à l'étape **X2**.

Avec cette approche le conflit d'appel entre les deux étapes **X1** et **X2** a été évité, mais cela a également eu pour effet de retarder d'un cycle automate le démarrage de la temporisation à l'étape **X2** (voir chronogramme figure 6). Pour y remédier, nous retranchons la durée du cycle automate (**Tcycle**) à la base de temps de l'étape **X2** (1000ms-T_{cycle}).

Les programmes automates des EALE sont organisés sous formes de blocs fonctionnels (en langage SFC) instanciés dans le programme principal. Ils renferment également des sections en langage LD dans lesquelles les observateurs et les défauts observés sont reconstruits. Cette organisation a été conservée dans les modèles de programmes. Les blocs fonctionnels ont été traduits en langage ST en utilisant la représentation algébrique des programmes SFC (Machado et al., 2006) (voir exemple en figure 8), et les programmes en LD ont été traduits en ST (voir exemple figure 7).

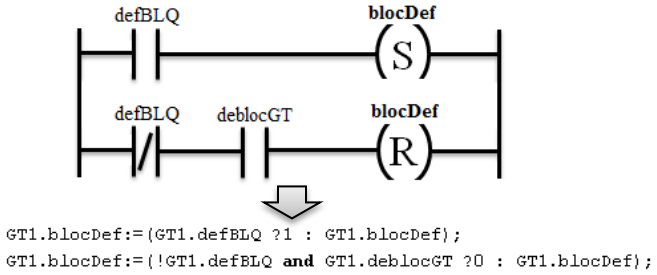

Figure 7 : Exemple de traduction d'un programme LD en ST

En raison de leur longueur, les programmes automates du groupe traction ne sont pas présentés dans ce papier.

3.3 Modélisation du cahier de recette

Après une analyse structurelle du cahier de recette, nous en avons déduit qu'il pouvait être traduit sous forme de programme SFC (par fiche de recette) car il est constitué d'une suite séquentielle d'actions et de transitions. La figure 8 présente l'équivalent en programme SFC (puis en programme ST) de la fiche de recette du tableau 1.

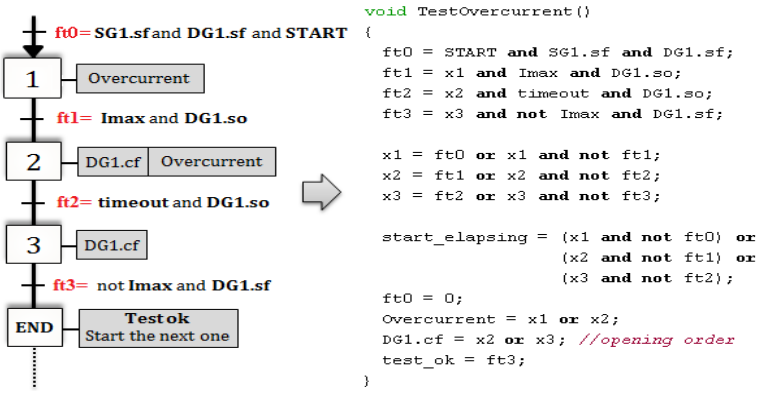

Figure 8 : équivalent fiche de recette (tableau 1) en programme SFC et ST

La variable « **start_elapsing** » (à droite de la figure 8) permet de démarrer une temporisation qui mesure pour chacune des étapes X1, X2, et X3 du scénario de tests, le temps écoulé entre l'exécution de l'action et l'obtention du résultat attendu. De ce fait, s'il existe une étape à laquelle le temps mesuré dépasse une certaine valeur (qui ne peut être atteinte si le programme est valide), cela signifie alors que le programme est défaillant car le résultat attendu n'a pas été obtenu.

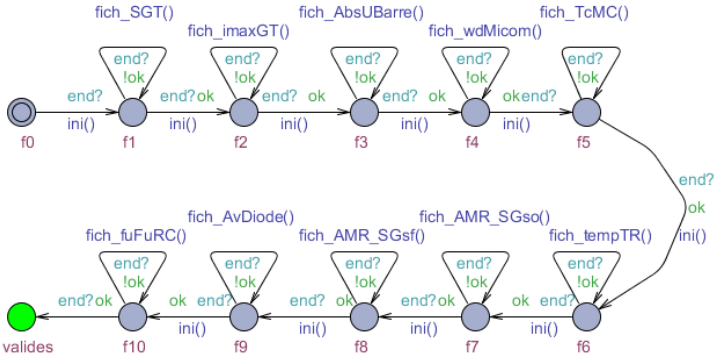

Figure 9 : Modèle du cahier de recette

Après avoir modélisé toutes les fiches de recette sous formes de programmes ST, le modèle de la figure 9 permet d'appeler et d'exécuter dans l'ordre tous les scénarios de tests. Dans cet exemple, 10 scénarios de tests doivent être exécutés dans l'ordre pour vérifier les programmes du Groupe Traction. Chaque état du modèle (à l'exception du 1er et du dernier) représente un scénario de tests. Durant la simulation, le modèle reste dans un état (entre *f1* et *f10*) tant que le scénario de tests qui lui est associé

n'est pas exécuté jusqu'à la fin. Un nouveau scénario ne peut être exécuté que lorsque le précédent (s'il existe) a été validé.

3.4 Résultats de la vérification

La figure 10 présente l'ensemble des modèles réunis dans l'interface du logiciel UPPAAL. Ceux-ci ont été synchronisés de manière à ce que les scénarios du cahier de recette soient exécutés dans l'ordre sur le système. Pour vérifier si toutes les fiches sont valides, il suffit alors de vérifier si l'état nommé « ***valides*** » du modèle du cahier de recette est atteignable via la propriété :

E <> fiches_test.valides (1)

Toutefois lorsque la propriété (1) n'est pas atteignable, UPPAAL ne retourne aucune trace qui permet de détecter l'origine du blocage. Dans ce cas nous vérifions la propriété (2) qui permet de rechercher la première instruction bloquante :

E<>time_step.timeout (2)

Dans cette dernière propriété, « **time_step** » est une temporisation (figure 5) déclenchée par la variable « **start_elapsing** » (définie plus haut). En vérifiant la propriété (2), UPPAAL nous retourne alors directement s'il existe, le numéro de l'instruction qui n'a pas été satisfaite. Ainsi, la trace retournée par UPPAAL après la vérification facilite le diagnostic au chargé d'études.

	Propriétés vérifiés	résultats	temps	mémoire
1er programme sans erreurs	*(1)* E < > fiches_tests.valides	satisfait	0.064 s	31Mb
	(2) E< > time_step.timeout	non satisfait	0.067 s	34Mb
2ième programme (avec des erreurs)	*(1)* E < > fiches_tests.valides	non satisfait	0.037 s	36Mb
	(2) E< > time_step.timeout	satisfait	0.037 s	36Mb

Tableau 2 : résultats de la vérification

Après la vérification du 1er programme, nous réalisons que la propriété (1) est satisfaite alors que la (2) ne l'est pas, ce qui est normal car les programmes vérifiés dans cet exemple l'ont déjà été en usine. Dans un second exemple, nous avons utilisé un programme qui a été modifié à notre insu, pour vérifier si notre méthode est en mesure de détecter tous les bugs rajoutés. Le tableau 2 présente les résultats de la vérification, ainsi que la durée et la mémoire utilisée (simulation faite avec un ordinateur de 4Gb de RAM, Core i5).

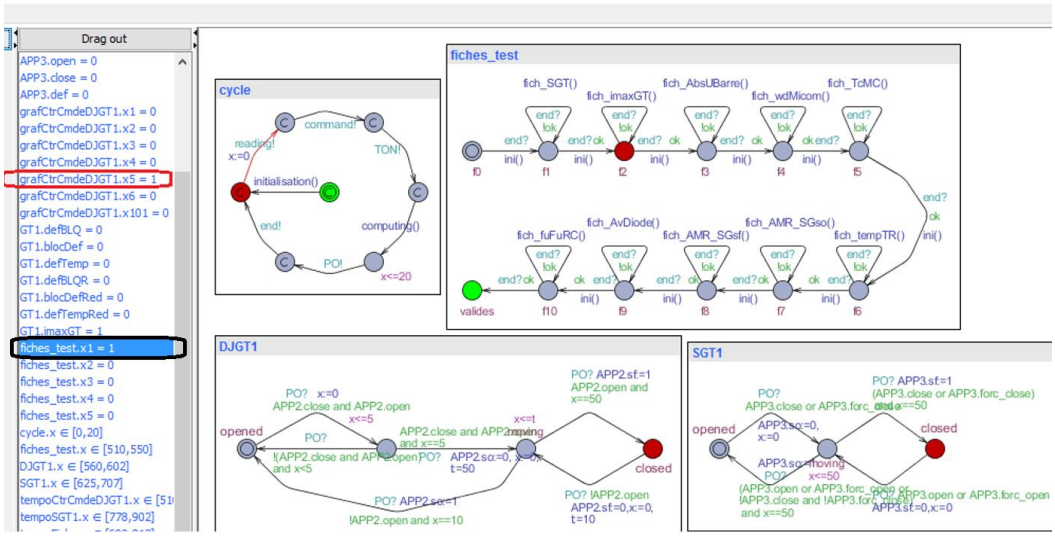

Figure 10 : interface de la simulation sous UPPAAL

La propriété (1) n'est pas satisfaite par le 2ième programme, ce qui veut dire qu'elle présente effectivement des erreurs. Mais pour pouvoir les diagnostiquer, nous devons vérifier la propriété (2). Après une première vérification, le model-checker nous retourne (visuellement, voir figure 10) l'instruction N°1 de la fiche de test du défaut court-circuit (instruction N°1 du scénario de tests du tableau 1, ou étape 1 de son équivalent en programme SFC à la figure 8). En effet, le résultat attendu : « ouverture du disjoncteur » ne s'est pas produit, ce qui a donc bloqué l'évolution de la fiche de tests. Après analyse de la trace (figure 10) retournée par UPPAAL, nous avons constaté que le programme de commande (**grafCtrCmdeDJGT1**, en langage SFC) du disjoncteur n'a pas évolué après l'apparition du défaut « Imax » (il est resté à l'étape X5 au lieu d'évoluer vers l'étape X6). L'expression « ***or Imax*** » a été en effet supprimée de la transition reliant les deux étapes X5 et X6, ce qui impliquait que le programme ne voyait pas l'information.

Après correction, suivie d'une deuxième vérification, la deuxième instruction de la même fiche de test a également été jugée bloquante. En effet, le disjoncteur devait rester ouvert tant que le défaut «court-circuit » n'a pas disparu, et cela même si l'opérateur essayait de refermer le disjoncteur, ce qui n'a pas été respecté par le programme. Après analyse, nous avons réalisé que l'expression ***« and not Imax*** » qui devait conditionner la refermeture de l'appareil a été supprimée du programme, le problème a donc été corrigé.

Avec cette approche, le chargé d'études peut corriger rapidement les problèmes détectés dans les programmes, d'autant plus qu'une vérification ne dure que quelques millisecondes. Cette méthode a été également utilisée pour vérifier et corriger de nouveaux types de programmes qui ont été établies pour remplacer les anciens. En effet, lors d'une récente recette usine, un chargé d'études avait pour mission d'utiliser les nouveaux programmes à la place des anciens pour mettre l'installation en service. Toutefois ces nouveaux programmes présentaient encore beaucoup d'anomalies, et n'ayant pas eu le temps pour les résoudre, il les a finalement abandonnés et a utilisé les anciens programmes. Grace à notre approche, nous avons pu détecter et corriger toutes les erreurs contenues dans ces nouveaux programmes (détaillés dans un rapport). Cela leur a évité de perdre à nouveaux du temps en essayant de vérifier ces nouveaux programmes en usine. Ces corrections ont été par la suite approuvées et validées par les chargés d'études.

4 Discussion et perspectives

4.1 Exhaustivité du cahier de recette

Bien que le cahier de recette ait servi à vérifier les programmes automates des EALE depuis plusieurs décennies, rien ne semble prouver que les tests garantissent formellement la sureté de l'installation. En effet, il ne permet d'étudier que quelques états atteignables par le système et pas la totalité (c'est-à-dire les 2^{Ne} états atteignables au maximum, avec Ne=nombre de capteurs de la partie opérative + nombre de commandes émises par l'opérateur). Le principe de la vérification de l'exhaustivité est le suivant : en faisant évoluer le système de manière aléatoire dans tous ses états atteignables (grâce à des modèles générateurs aléatoires de défauts et de commandes, voir figure 11), nous définissons un ensemble de propriétés de sécurité représentant tous les états dangereux du système, puis l'atteignabilité de celles-ci doit être vérifiée pour chacun des 2^{Ne} états du système. De ce fait, si nous arrivons à trouver un programme (déjà vérifié avec le cahier de recette) qui viole les propriétés de sécurité, nous pourrons conclure que les tests du cahier de recette ne sont pas exhaustifs.

Pour vérifier l'exhaustivité avec UPPAAL, nous supprimons le modèle du cahier de recette, puis nous instancions respectivement pour chaque défaut observé et commande opérateur, un modèle générateur aléatoire de défaut et un générateur aléatoire de commande. Avec le modèle de gauche (figure 11), tout défaut est aléatoirement mis à 1 ou à 0 à chaque cycle automate, et avec le modèle de

droite, toute commande envoyée par l'opérateur sur un appareil peut être du type ouverture (co=1 et cf=0), fermeture (co=0 et cf=1), ou aucune des deux (co=cf=0).

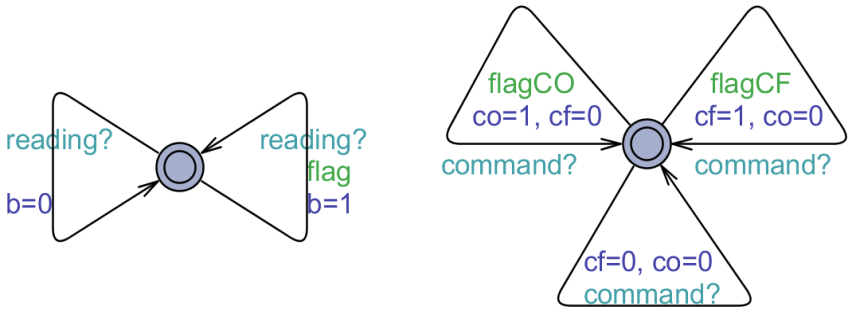

Figure 11 : Générateurs aléatoires de défaut (à gauche) et de commande opérateur (à droite)

Cette approche de vérification exhaustive est très efficace car elle permet d'étudier tous les états atteignables par le système. Toutefois elle requiert plus de temps et de mémoire durant la vérification en raison de l'augmentation de l'espace d'états, elle est donc plus exposée aux explosions combinatoires. Elle n'a pas encore été testée sur l'intégralité des installations électriques et toutes les propriétés de sécurité ne sont pas encore définies. Avec les installations que nous avons déjà testées avec cette méthode, nous avons trouvé des combinaisons (de commandes et de défauts) sur certaines installations, qui violent les propriétés de sécurité. Toutes ces combinaisons trouvées ont été jugées par les experts de la SNCF comme étant improbables, voire irréalisables sur site. Ils considèrent alors (comme depuis plusieurs décennies) que les tests du cahier de recette leur suffisent pour garantir la sûreté de l'installation électrique, jusqu'à preuve du contraire. L'étude de l'exhaustivité du cahier de recette est toujours en cours.

4.2 Génération automatique des modèles UPPAAL

La méthode de vérification automatique des programmes automates est destinée principalement aux nouveaux programmes API pour lesquels le fonctionnement n'a jamais été testé en usine. Toutefois ces programmes doivent être nécessairement traduits pour être interprétables et testés dans UPPAAL. Le problème est que non seulement le chargé d'études ne maîtrise pas actuellement la modélisation sous UPPAAL, mais que même dans le cas contraire il pourrait faire des erreurs de modélisation. La solution proposée consiste alors à générer automatiquement tous les modèles nécessaires pour effectuer la vérification en utilisant le logiciel ODIL. En effet, celui-ci renferme pour toute installation électrique, les données telles que la composition de la partie opérative, les programmes, et les fiches de tests, et est capable de générer n'importe quel fichier texte ou XML à partir de ces données. Les modèles UPPAAL étant sous formes de fichiers XML, pourront donc être automatiquement générés. La méthodologie de génération des modèles UPPAAL n'est pas détaillée dans ce document, mais présentée dans (Coupat, 2014). Après génération des modèles, le chargé d'études aura juste à les importer depuis UPPAAL et démarrer la vérification des programmes.

5 Conclusion

L'objectif de ce travail de recherche est d'optimiser la phase de vérification des programmes automates des EALE pour les chargés d'études de la SNCF. La nouvelle méthode mise en place est basée sur des approches formelles, et repose sur une modélisation de la partie opérative, des

programmes et du cahier de recette dans le model-checker UPPAAL. La vérification n'entraine pas d'explosion combinatoire en raison de la simplicité des modèles, et les résultats de la simulation s'obtiennent en quelques millisecondes.

L'avantage de cette méthode est qu'elle permet au chargé d'études de vérifier rapidement les programmes automates afin de mieux préparer (faciliter) la recette usine à venir. Elle lui permet également de diminuer le temps, la complexité du debbugage (réduite à présent aux problèmes de câblage), et le stress liés aux tests en usine. La prochaine étape de ce projet de recherche est de mettre en place une approche qui permettra aux chargés d'études de vérifier en ligne le câblage des armoires électriques lors de la recette usine, grâce à un banc de tests connecté à un simulateur de partie opérative.

Cette méthode de vérification des programmes API a également été exposée aux chargés d'études de la SNCF, qui l'ont jugée intéressante suite aux résultats obtenus sur certaines applications. L'étude de son intégration au sein du métier des chargés d'études est actuellement en cours, afin d'optimiser la phase de vérification des programmes automates et de gagner plus de temps durant les projets d'automatisation.

Références

Behrmann, G., Bengtsson, J., David, A., Larsen, K.-G., Pettersson, P., Yi, W., 2002. UPPAAL implementation secrets. *7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*. In Springer, *Verlag London, UK 2002*: 3-22.

Canet, G., Couffin, S., Lesage, J.-J., Petit, A., and Schnoebelen, P. (2000). *Towards the automatic verification of PLC programs written in instruction list*, Proc. of the IEEE SMC, 2000.

Coupat, R., 2014. *Méthodologie pour les études d'automatisation et la génération automatique de programmes Automates programmables Industriels sûrs de fonctionnement – Application aux Equipements d'Alimentation des Lignes Electrifiées*, doctorat, Université de Reims Champagne-Ardenne, décembre 2014.

Coupat, R., Meslay, M., Burette, M.-A., Philippot, A., Annebicque, D., Riera, B., 2014. Standardization and Safety Control Generation for SNCF Systems Engineer. *19th IFAC World Congress 2014 (IFAC WC 2014)*, Cape Town, South Africa, 2014.

EN 50126, 2012. Applications ferroviaires - Spécification et démonstration de la fiabilité, de la disponibilité, de la maintenabilité et de la sécurité (FDMS).

Katoen, J-P., Baier, C., 2008. Principles of Model Checking. The MIT Press, Cambridge, Massachusetts, London, England.

Mokadem, H.-B., Bérard, B., Gourcuff, V., De Smet, O., Roussel, J.-M., 2010. Verification of a timed multitask system with UPPAAL. *IEEE Transactions on Automation Science and Engineering, Institute of Electrical and Electronics Engineers*, 7 (4), pp.921 - 932.<10.1109/TASE.2010.2050199>.<hal- 0527736>.

IEC 60870-4 , 2013. Telecontrol equipment and systems. Part 4: Performance requirements Ed. 1.

IEC (International Electrotechnical Commission). IEC Standard 61131: Programmable controllers - Part 3, 1993.

Machado, J., Denis, B., Lesage, J.-J., Faure, J.-M., Ferreira Da Silva, J., *2006.* Logic controllers dependability verification using a plant model In *Proc. 3rd IFAC Workshop on Discrete-Event System Design, (DESDes'06), Rydzyna (Poland), Sept. 2006*, pages 37-42.

UTE C 18510. Operations on electrical work and installations and in an electrical environment - Electrical hazard prevention.