Vérification formelle et simulation pour la validation des systèmes de contrôle commande des EALE

Mohamed Niang

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

ÉCOLE DOCTORALE SCIENCES DU NUMÉRIQUE ET DE L'INGÉNIEUR

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Discipline : ***AUTOMATIQUE, SIGNAL, PRODUCTIQUE, ROBOTIQUE***

Spécialité : Automatique et Traitement du Signal

Présentée et soutenue publiquement par

Mohamed NIANG

Le 20 décembre 2018

Vérification formelle et simulation pour la validation des systèmes de contrôle commande des EALE

Thèse dirigée par **Bernard RIERA**

JURY

M. Sébastien LAHAYE	Professeur des Universités	LISA – Université d'Angers	Rapporteur
M. Jean-Jacques LESAGE	Professeur des Universités	LURPA – Ecole Normale Supérieure Paris-Saclay	Rapporteur
Mme Sylvie CHARBONNIER	Professeur des Universités	INP – Université de Grenoble	Examinatrice
M. Damien TRENTESAUX	Professeur des Universités	UPHF – Université de Valenciennes	Examinateur
M. Alexandre PHILIPPOT	Maître de conférences	CReSTIC – Université de Reims Champagne-Ardenne	Examinateur Co-encadrant
M. François GELLOT	Maître de conférences	CReSTIC – Université de Reims Champagne-Ardenne	Examinateur Co-encadrant
M. Raphaël COUPAT	Docteur / Chef de projet Recherche	SNCF Réseau	Examinateur Co-encadrant
M. Bernard RIERA	Professeur des Universités	CReSTIC – Université de Reims Champagne-Ardenne	Examinateur Directeur de thèse

Remerciements

Mes remerciements s'adressent dans un premier temps aux rapporteurs, Monsieur Sébastien LAHAYE et Monsieur Jean-Jacques LESAGE pour le temps et l'énergie consacrés à la lecture de ce mémoire de thèse, ainsi qu'aux examinateurs Monsieur Damien TRENTESAUX et Madame Sylvie CHARBONNIER pour l'intérêt qu'ils ont porté à ce travail.
Je tiens à remercier Monsieur Bernard RIERA mon directeur de thèse, qui m'a accordé sa confiance en prenant la direction de cette thèse, qui m'a guidé, corrigé et appris énormément tout au long de ces trois années de travail. Dans cette même perspective je remercie également Monsieur Alexandre PHILIPPOT et Monsieur François GELLOT qui m'ont supporté tous les jours et qui m'ont appris énormément durant ces trois années. Sans oublier Monsieur Janan ZAYTOON qui m'a beaucoup aidé bien structurer les idées de la thèse, la rédaction du manuscrit, et à valoriser mes travaux de recherches à travers la publication d'une revue. Je tiens également à remercier tous les membres du CReSTIC, qui m'ont accueilli avec beaucoup de sympathie, ont soutenu mon travail, et avec qui j'ai partagé de très bon moments.
Je tiens aussi à remercier la direction du département DGII-TE (Direction Générale Industrielle et Ingénierie - Traction Électrique), notamment Monsieur Christian COURTOIS, Monsieur Stéphane MARIE, et Monsieur Marc MESLAY mon ancien chef de section CES 2 (Conception et Expertise des Systèmes), qui ont soutenu et encouragé ce projet de recherche.
Je remercie également tous les membres de la section CES2 pour leur soutien, leurs explications, leur implication dans le but d'aider à la réussite de ce projet de recherche. En particulier je remercie mes encadrants industriels Monsieur Raphaël COUPAT et Monsieur Sébastien LEFEBVRE, qui ont activement participé à la mise en place de ce projet et à sa bonne évolution durant ces trois années. Je n'oublierai pas de nommer Monsieur Gal Blaszczyk mon actuel chef de section CES 2, Monsieur Alexandre BOUAZIZ, Monsieur Alexandre Mendeiros, et Monsieur Romain BARTHE, qui également contribué à la réussite de ce projet de recherche.
Je ne terminerai pas sans mentionner ma famille bien sûr, qui a su m'accompagner durant les étapes difficiles de cette thèse. Papa, Maman, votre éducation sans failles et vos enseignements m'ont été d'une très grande utilité durant cette thèse, sans oublier bien sûr vos prières qui ne cessent de m'accompagner. Je n'aurais jamais pu arriver jusqu'à ce stade sans vous, et je ne vous en remercierai jamais assez. Un grand merci également à mon grand-frère et mes grandes sœurs, Ibrahima, Yacine Diouf, et Diarra NIANG, pour leur soutien. Je n'oublierai pas de remercier ma petite femme chérie, Anne Marie DIENA, d'avoir été là pour s'occuper de moi, et de rendre ma vie simplement parfaite.
Enfin, merci à mes amis que je ne vais pas énumérer, pour leur joie de vivre et l'équilibre qu'ils m'apportent.

Table des matières

Table des figures

Liste des tableaux

Glossaire, abréviations et symboles

AA - Abonné Automate

API - Automate Programmable Industriel

BT - Basse Tension

CES - Conception et Expertise des Systèmes

CF - Commande Fermeture

CIFRE - Convention Industrielle de Formation à la Recherche en Entreprise

CLE - Commande Locale Ecran

CO - Commande Ouverture

CReSTIC - Centre de Recherche en STIC

CSS - Central Sous-Station

CTL - Computation Tree Logic

DJ - DisJoncteur

DT - Départ Traction

EALE - Équipement d'Alimentation des Lignes Électrifiées

EN - Norme Européenne

FBD - Function Block Diagram

FG - Fournié Grospaud

FIP - Factory Information Protocol

FP - Fil pilote

HIL - Hardware-In-the-Loop

HT - Haute Tension

I&P.TE - Ingénierie & Projets de Traction Électrique

IEC - International Electrotechnical Commission

IHM - Interface Homme Machine

LDI - mode de commande «Local Individuel»

LD - Ladder Diagram (langage de programmation API)

LTL - Linear Temporal Logic

ODIL - Outil Description d'ILots

PO - Partie Opérative

RFF - Réseau Ferré de France

RFN - Réseau Ferré National

RLI - Réseau Local Industriel

RSS - Régulateur Sous-Station

RTE - Réseau de Transport d'Electricité

RTU - Remote Terminal Unit

SePO - Sous-ensemble de Partie Opérative

SFC - Sequential Function Chart (langage de programmation API)

SF - Signalisation position Fermée d'un appareil

SIL - Software-In-the-Loop

SMV - Symbolic Model Verifier

SNCF - Société Nationale des Chemins de Fer français

SO - Signalisation position Ouverte d'un appareil

STIC - Sciences et Technologies de l'Information et de la Communication

ST - Structured Text (langage de programmation API)

TA - Timed Automata

TCO - Tableau de Contrôle Optique

TCTL - Timed Computational Tree Logic

TON - Timer On-Delay

URCA - Université de Reims Champagne-Ardenne

V&V - Vérification et Validation

XML - eXtensible Markup Language

Introduction générale

Les systèmes automatisés sont de plus en plus répandus et omniprésents dans plusieurs domaines tels que le transport, l'informatique, l'industrie pharmaceutique, l'aéronautique, et le nucléaire. Ces systèmes souvent critiques mettent en jeu des coûts importants, voire des vies humaines. Ils sont de plus en plus autonomes, complexes et interagissent directement avec leur environnement, leur fiabilité est donc primordiale. Des défaillances peuvent entraîner de graves conséquences allant à l'encontre de la sécurité des personnes et des biens, comme l'illustrent certains bugs célèbres (Mokadem, 2006) à l'instar de l'explosion de la fusée **Ariane-5** et le bug du processeur **Pentium II** d'Intel qui ont causé des pertes économiques très importantes. De plus, le bug célèbre de l'appareil médical **Therac-25** en 1985 a provoqué la mort de plusieurs patients. Ces illustrations malheureuses sont toutes dues à des défaillances, qui proviennent généralement de nombreux facteurs tels que :

— l'influence de l'environnement,

— des erreurs humaines lors de la conception,

— des pannes matérielles aléatoires,

— etc.

Ces facteurs sont souvent incontrôlables voire inévitables, à l'exception des erreurs humaines qui peuvent survenir lors de la phase de conception des systèmes. Par exemple, une étude menée sur des systèmes complexes a montré que 70% des erreurs des produits logiciels trouvent leur origine dans les phases de spécification et de conception (Selby and Selby, 2007) (contre 79% dans Sourisse and Boudillon (1997)), et que le coût relatif aux corrections augmente considérablement au fil du temps (Boehm, 1979). La détection tardive des erreurs entraîne une explosion des délais et des coûts de correction. Ces études montrent ainsi l'importance d'une bonne méthodologie de conception et de vérification des systèmes.

Face à ces exigences en milieu industriel, plus précisément dans le domaine des systèmes automatisés, les ingénieurs disposent aujourd'hui de plusieurs méthodes pour concevoir

des systèmes de contrôle commande, mais également pour les vérifier et les valider avant la mise en service du système automatisé, afin de s'assurer que les exigences du cahier des charges ont été respectées. Ces exigences sont liées non seulement au respect des spécifications fonctionnelles, mais aussi à la sûreté de fonctionnement tout au long du cycle de vie de l'installation. À cette fin, plusieurs méthodes de vérification et de validation (V&V) existent et s'appliquent en industrie, mais elles différent de par leur type, leur efficacité, leur fiabilité et leur rapidité d'exécution.

Cependant, avec l'augmentation de la complexité et des exigences des systèmes automatisés, la V&V devient de plus en plus complexe. Telle est la difficulté que rencontrent aujourd'hui les chargés d'études automatisme de la SNCF, qui sont chargés de la conception et de la V&V des systèmes de contrôle commande des « Équipements d'Alimentation des Lignes Électrifiées » (**EALE**).

Les EALE sont les points d'alimentation en énergie électrique des lignes électrifiées ou caténaires, à partir desquelles les engins de traction puisent leur énergie électrique. Elles ont pour rôle de transformer, de distribuer et de redresser dans le cas d'une alimentation continue, la tension du réseau HT en des valeurs de tension compatibles avec les engins de traction. Ces EALE sont des systèmes automatisés complexes et soumises à des normes strictes de sécurité ferroviaire (EN5, 2012) (IEC-60870-4, 2013).

Durant les projets d'automatisation des EALE, les chargés d'études de la SNCF sont responsables de la conception (phase 1) et de la V&V (phase 2) des systèmes de contrôle commande (figure 1). La phase 1 consiste à concevoir les programmes automates et le cahier de recettes, et à réaliser les schémas électriques de l'EALE et de la partie commande. Le cahier de recettes est un document contenant un ensemble de « procédures de tests » (appelées également « scénarios de tests », « fiches de tests », ou « fiches de recettes ») nécessaires pour valider un système de contrôle commande. Après relecture de l'ensemble des livrables établis, la phase 2 consiste à vérifier et à valider le système de contrôle commande en usine. Cette phase permet de s'assurer que les programmes automates ont été conçus sans erreurs (i.e. ils respectent le cahier des charges) et que les armoires de contrôle commande ont été correctement réalisées sans erreur de câblage des entrées/sorties. Le principe de cette phase 2 consiste à exécuter manuellement les procédures de tests du cahier de recettes sur le système de contrôle commande. Les résultats obtenus durant les essais sont ainsi comparés aux résultats attendus, afin de permettre aux chargés d'études de valider ou non la partie commande. L'expérience et le savoir-faire qu'ils ont acquis au fil

des années leur ont permis de considérer qu'un système de contrôle commande est validé lorsque tous les tests de validation du cahier de recettes ont été exécutés avec succès.

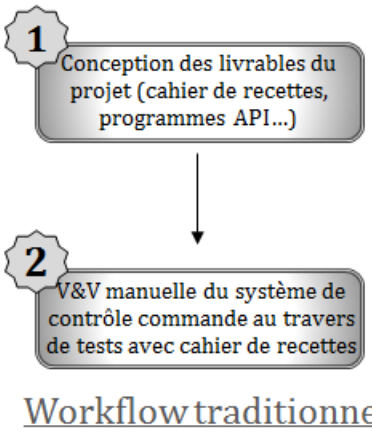


FIGURE 1 – Phases 1 et 2 du workflow d'un projet d'automatisation des EALE

Cependant, ces deux phases qui constituent le workflow des chargés d'études sont très complexes et exigent beaucoup de temps et de ressources. Les livrables d'un projet (programmes API, schémas électriques...) sont longs et fastidieux à établir, sans compter la présence de tâches répétitives lors de la phase 1. La phase de V&V, précédée d'une simple relecture des livrables, nécessite la présence du chargé d'études en usine (d'une à deux semaines). Cette phase 2 requiert surtout beaucoup d'expertise et de concentration de la part du chargé d'études en usine, où les conditions de travail sont souvent difficiles. La fatigue générée par cette surcharge de travail, ainsi que le stress lié aux deadlines imposées pour des raisons de productivité, exposent le chargé d'études à une surcharge mentale durant les projets d'automatisation. Par conséquent, ces mauvaises conditions de travail ne le mettent pas à l'abri d'erreurs humaines durant les phases de conception et de V&V, ce qui peut compromettre la fiabilité des systèmes de contrôle commande.

Pour remédier à ces inconvénients, un projet de recherche mené par Coupat (2014) (dans le cadre d'une convention CIFRE entre le laboratoire CReSTIC de l'Université de Reims et la SNCF) avait pour objectif d'améliorer la phase 1 du workflow des chargés d'études (conception des livrables du système de contrôle commande). Cette étude a abouti au développement d'un outil nommé « ODIL GREMLINS » (qu'on appellera « ODIL » dans le reste du document), qui permet de générer automatiquement les livrables d'un projet à partir de la description du projet dans l'interface du logiciel ODIL. Cette nouvelle méthodologie a été adoptée depuis lors par les chargés d'études, et leur a permis de réduire la charge de travail, de supprimer les tâches répétitives, et de gagner

deux fois plus de temps durant la conception des livrables tout en minimisant les erreurs. Néanmoins, la phase de V&V du système de contrôle commande n'a fait l'objet d'aucune amélioration malgré les nombreux inconvénients qu'elle présente, tels que la longueur des essais manuels sujets à des erreurs humaines et le manque d'exhaustivité de l'approche de V&V. Leur méthodologie ne permet pas toujours de détecter toutes les erreurs de conception de la partie commande. La preuve en est que près de 5% des pannes électriques qui se produisent après la mise en service des EALE de la SNCF sont dues à des erreurs de conception du système de contrôle commande qui auraient pu être détectées et corrigées avec de meilleures techniques de V&V beaucoup plus fiables.

Ce travail de recherche est réalisé dans le cadre d'une Convention Industrielle de Formation à la Recherche en Entreprise (CIFRE), signée entre la Direction de l'Ingénierie SNCF et le Centre de Recherche en Sciences et Technologies de l'Information et de la Communication (CReSTIC) de l'Université de Reims Champagne-Ardenne (URCA). L'objectif est de contribuer à l'amélioration de la phase 2 du workflow des chargés d'études automatisme, en proposant une approche méthodologique de V&V des systèmes de contrôle commande des EALE, plus rapide et plus fiable que leur approche traditionnelle, de manière à supprimer les contraintes qu'ils rencontrent durant cette phase.
Le manuscrit de thèse est décomposé en quatre chapitres :

Le premier chapitre a pour but de présenter le groupe SNCF et le domaine des Équipements d'Alimentation des Lignes Électrifiées. Les EALE peuvent être de plusieurs types, et peuvent délivrer plusieurs niveaux d'alimentations électriques différentes. Nous décrivons également l'architecture globale des systèmes de contrôle commande, et la structure des programmes automates. Ensuite, nous présentons dans ce chapitre le workflow des chargés d'études durant les projets d'automatisation. Nous commençons par décrire le workflow traditionnel, puis le nouveau workflow des chargés d'études suite aux améliorations apportées par Coupat (2014) à travers ses travaux de recherche. Les difficultés rencontrées dans ce nouveau workflow sont également décrites. Pour finir, la contribution de ce travail de recherche portant sur la V&V des systèmes de contrôle commande des EALE est présentée.

Le deuxième chapitre de ce manuscrit présente un état de l'art sur les techniques de vérification et de validation des systèmes de contrôle commande. Il existe aujourd'hui plusieurs techniques de V&V utilisées en industrie ou dans le milieu académique, mais elles diffèrent de par leur approche, leur périmètre d'application, leur facilité et rapidité

d'exécution, et surtout leur fiabilité. Les principales techniques utilisées sont décrites dans ce chapitre 2, puis elles sont comparées entre elles de manière à faire ressortir les avantages et les manquements de chacune d'elles. Ensuite, après avoir positionné la méthodologie de V&V des chargés d'études, et mis en évidence ses inconvénients, nous présentons à la fin de ce chapitre 2 la contribution méthodologique de ce projet de recherche, dédiée à la vérification et à la validation des systèmes de contrôle commande des EALE. Cette nouvelle méthodologie devra répondre aux attentes des chargés d'études.

La contribution de ce travail de recherche est présentée dans les chapitres 3 et 4. Cette nouvelle méthodologie est constituée de deux étapes principales : une étape de **vérification**, et une étape de **validation** du système de contrôle commande. La première étape, détaillée dans le chapitre 3, concerne la vérification des programmes automates résultant de la phase 1 du workflow des chargés d'études (génération automatique des livrables). La méthodologie est d'abord décrite dans ce chapitre 3, avant d'être appliquée sur un cas concret. Ensuite, la deuxième étape, détaillée dans le chapitre 4, présente la solution retenue pour la validation du système de contrôle commande (programmes automates et câblage des armoires électriques).

La contribution de ce travail de recherche doit répondre à trois objectifs principaux :

- **Sécurité :** L'exécution manuelle des essais en usine est sujette aux erreurs humaines pouvant compromettre la sécurité de l'installation. De plus, le manque d'exhaustivité des tests basés sur le cahier de recettes peut compromettre la sécurité des personnes et des biens. Ces inconvénients doivent par conséquent être résolus ;
- **Humain :** La phase de V&V en usine nécessite beaucoup de temps, de concentration, et de ressources de la part des chargés d'études, sans compter les conditions difficiles dans lesquelles ils travaillent. La méthodologie proposée doit permettre d'améliorer leur condition de travail durant la phase de V&V ;
- **Économique :** Cette solution doit permettre de diminuer les coûts liés aux prestations pour les tests de V&V, aux pertes matérielles en cas d'erreurs de conception, ou aux précautions de sécurité prises durant les essais.

Chapitre I

Le métier des Équipements d'Alimentation des Lignes Électrifiées (EALE) de la SNCF

1 Introduction

Ce premier chapitre a pour but de décrire le domaine spécifique lié à ce travail de recherche. Il s'agit du métier des Équipements d'Alimentation des Lignes Électrifiées (EALE). Après avoir présenté le groupe SNCF, la structure des EALE ainsi que les types d'alimentations électriques existantes sont décrits. Ensuite, l'architecture des systèmes de contrôle commande des EALE ainsi que les exigences liées à leur sûreté de fonctionnement seront présentées. Durant les projets d'automatisation des EALE, les chargés d'études sont responsables de la conception (dimensionnement et programmation), et de la Vérification et Validation (V&V) des systèmes de contrôle commande. La vérification et la validation sont des étapes complémentaires et incontournables pour un système automatisé. Elles permettent de s'assurer, avant la mise en service du système, que les exigences décrites dans le cahier des charges ont été satisfaites.

Cependant, en raison de la complexité de ces systèmes, le workflow traditionnel suivi par ces chargés d'études présente beaucoup d'inconvénients, notamment la présence de tâches longues, manuelles, répétitives, et sources d'erreurs. Pour pallier ces inconvénients, Coupat (2014) a proposé une méthodologie originale visant à améliorer les conditions de travail des chargés d'études. Les résultats de ces travaux sont également présentés dans ce chapitre. Cependant, ces études d'amélioration n'ont porté que sur les tâches liées à la

conception des livrables (programmes API, cahier de recettes...). La phase de V&V n'a donc fait l'objet d'aucune amélioration malgré tous les inconvénients qu'elle présente (voir section 5.2.2). Ces problématiques liées à la phase de V&V sont justement au coeur des préoccupations de ce projet de recherche. L'objectif est de mettre en place une approche méthodologique permettant de vérifier et de valider les systèmes de contrôle commande des EALE. Cette nouvelle méthode de V&V des systèmes de contrôle commande devra être plus rapide et plus efficace que l'ancienne méthode utilisée par les chargés d'études, tout en améliorant leurs conditions de travail.

2 Présentation du groupe SNCF

La SNCF (Société Nationale des Chemins de fer Français) est une entreprise publique industrielle et commerciale, officiellement créée le 1er janvier 1938 en application du décret-loi du 31 aout 1937. C'est une entreprise ferroviaire « intégrée », c'est-à-dire qu'elle exerce à la fois le métier d'exploitant et celui de gestionnaire d'infrastructure ferroviaire. Elle est détenue par l'Etat comme actionnaire. La SNCF est présente dans les domaines du transport de voyageurs, du transport de marchandises ainsi que la gestion, l'exploitation et la maintenance du réseau ferré national.
Depuis le 1er Janvier 2015, la SNCF s'est réorganisée en 3 ÉPIC (Établissements Publics à caractère Industriel et Commercial) :

- **SNCF Réseau** : qui se charge de la gestion, de l'exploitation et du développement de l'infrastructure du Réseau Ferré National (RFN) ;
- **SNCF Mobilités** : qui gère les activités d'exploitation des services de transport de voyageurs et de marchandises ;
- **SNCF (Direction)** : qui prend en charge le pilotage global du groupe.

L'ÉPIC **SNCF Réseau** est constitué de 4 entités :

- l'entité « Production Industrielle »,
- l'entité « Maintenance du Réseau »,
- l'entité « Ingénierie & Projets »,
- l'entité « Services Transverses ».

Ce projet de recherche est effectué au sein de l'entité « Ingénierie & Projets », et plus précisément au sein du pôle « Conception et Expertise EALE » du département « Ingénierie & Projets de Traction Électrique (I&P.TE) ». Cette section travaille pour le compte des

chargés d'études en fournissant les équipements d'automatisme des Équipements d'Alimentation des Lignes Électrifiées (EALE). Elle assure également les études, le suivi et la mise en service des systèmes de téléconduite, d'automatismes et de protection destinés au pilotage des EALE.

3 Présentation des EALE

Les EALE sont les points d'alimentation en énergie électrique des lignes électrifiées appelées caténaires. Elles ont pour rôle de transformer, de distribuer et de redresser dans le cas d'une alimentation continue, la tension du réseau HT en des valeurs de tension compatibles avec les engins de traction. Ainsi pour assurer la traction électrique, les niveaux de tension du réseau de lignes électrifiées sont de deux types :

— l'un à courant continu, avec une tension de 1.5 kV,

— l'autre à courant alternatif d'une tension de 25 kV - 50 Hz

En 2014, les lignes exploitées sont estimées à une longueur de 30 000 km, dont 15375 km électrifiées et alimentés par 560 sous-stations. Parmi ces lignes électrifiées sont compris 2 600 km de Lignes à Grande Vitesse (LGV).

Les EALE sont constituées d'installations fixes comprenant :

— les sous-stations électriques réparties le long de la ligne ferroviaire avec des distances allant généralement de 8 à 20 km pour le 1.5 kV et de 40 à 90 km pour le 25 kV,

— les lignes de contact ou caténaires,

— les postes assurant le sectionnement ou la mise en parallèle des caténaires,

— les rails de roulement assurant le retour du courant vers la sous-station.

3.1 Les sous-stations électriques

Les sous-stations sont la propriété de RFF (Réseau Ferré de France), elles sont raccordées au réseau haute tension de distribution, propriété de RTE (Réseau de Transport d'Électricité). Elles représentent les points d'alimentation en énergie électrique des caténaires, et alimentent des tronçons de lignes appelés secteurs (figures 2 et 3).

L'alimentation par secteur a été choisie pour des besoins de souplesse d'exploitation et de maintenance du réseau ferré. Chaque secteur est électriquement indépendant et est alimenté par une ou deux sous-stations. En cas de panne électrique d'un secteur, l'architecture du réseau est conçue de sorte que le secteur en panne puisse être alimenté

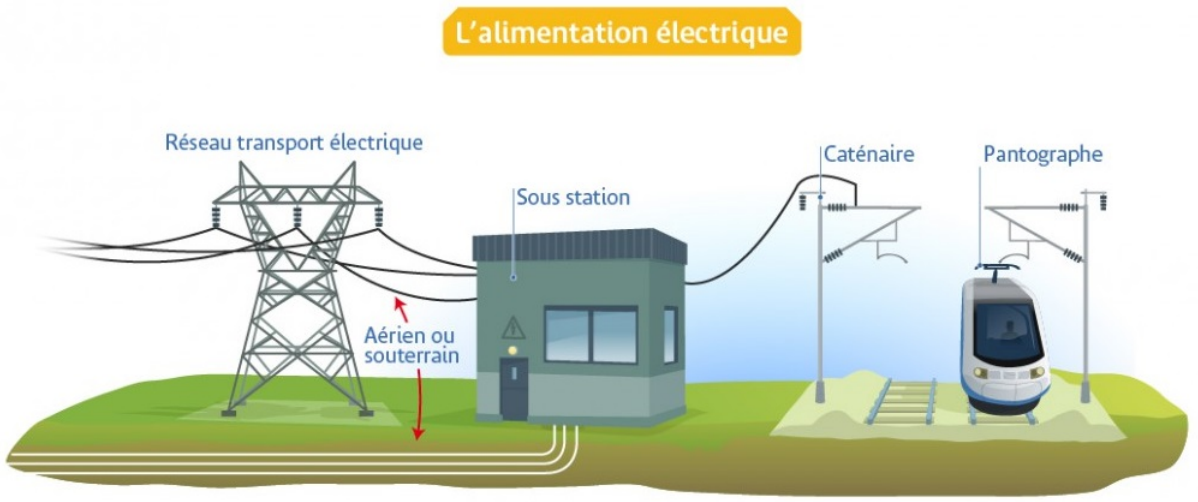


FIGURE 2 – Schéma d'alimentation d'un train par une sous-station électrique

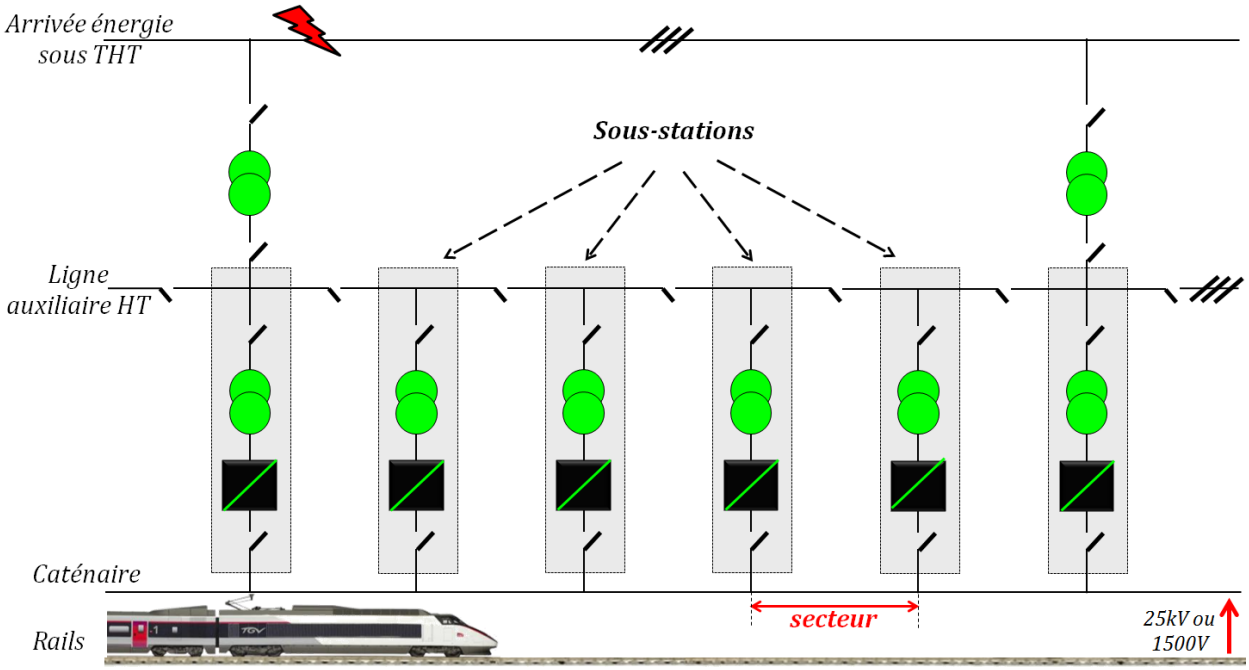


FIGURE 3 – Schéma d'alimentation d'une ligne par plusieurs sous-stations

(ou secouru) par une sous-station voisine au moment où la sous-station en panne est en maintenance. Ce système permet d'assurer la continuité de l'alimentation électrique sur l'ensemble de la ligne afin d'éviter l'interruption de la circulation des trains.

La puissance délivrée par les sous-stations dépend du trafic de la ligne qu'elles alimentent. En effet, plusieurs trains peuvent emprunter un même secteur, d'où la nécessité de calculer préalablement la puissance à fournir lors du dimensionnement de la sous-station. Les lignes de contact ou caténaires permettent à l'engin moteur de capter l'énergie électrique via le pantographe pour la transformer en énergie mécanique afin d'assurer la traction du convoi.

Il existe deux types de sous-stations : l'une à courant continu (sous-station 1,5 kV continu) et l'autre à courant alternatif (sous-station 25 kV alternatif). Quelle que soit sa nature, une sous-station est composée de 4 sous-ensembles de partie opérative - SePO (Figure 4) :

— La partie alimentation ou poste haute tension (HT) : raccordée au réseau RTE, constituée de jeux de barres, de sectionneurs et de disjoncteurs.

— La partie conversion (Groupe Traction GT), constituée principalement d'un redresseur (dans le cas de l'alimentation 1,5 kV) et d'un transformateur.

— La partie distribution (Départ Traction DT), pour laquelle les disjoncteurs de départ alimentent et protègent les caténaires.

— La partie énergie auxiliaire (non visible sur la figure 4) qui fournit l'énergie à l'appareillage de l'installation : Courants BT 220V et 48 ou 120 V continu obtenus par les transformateurs de services auxiliaires.

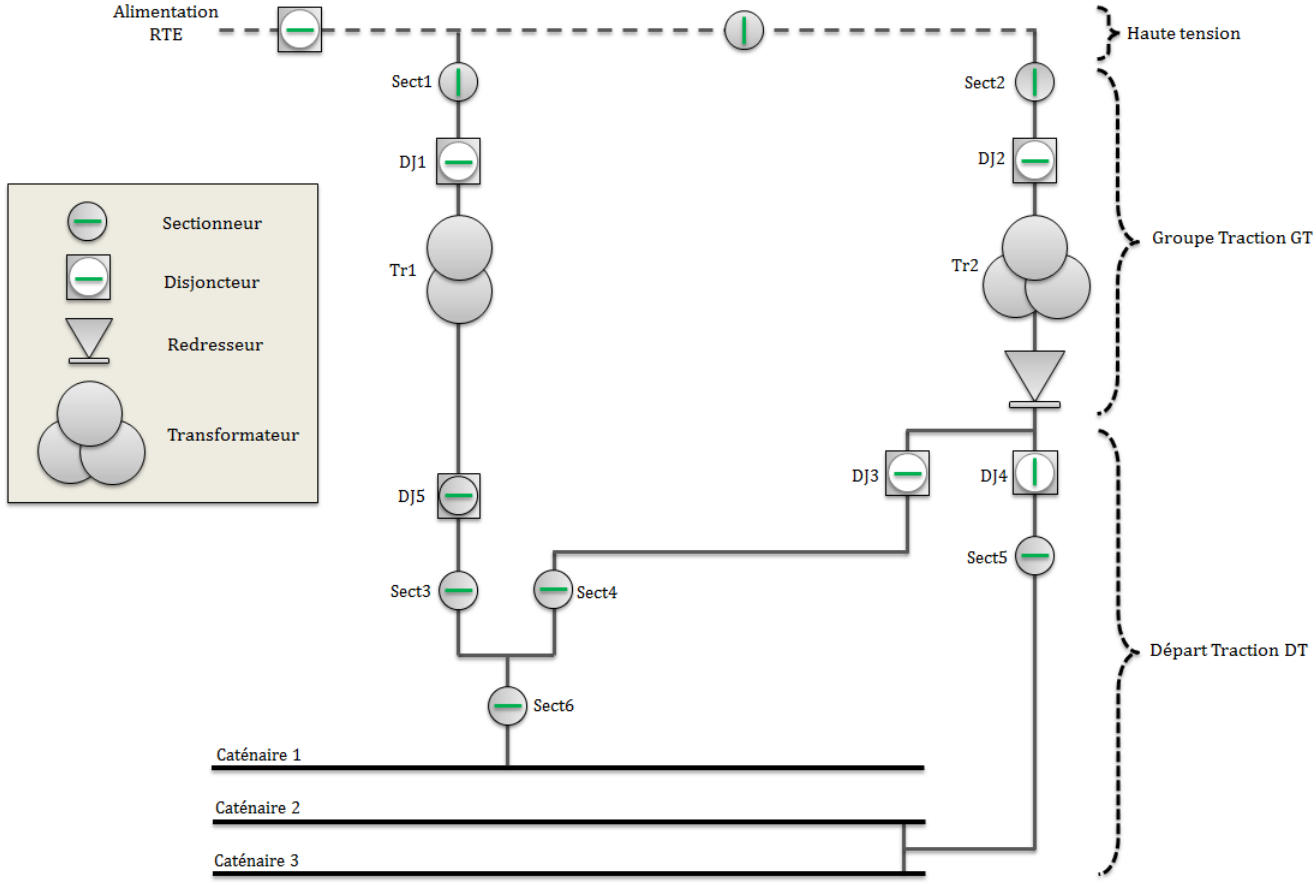


FIGURE 4 – Constitution d'une sous-station

3.2 L'alimentation électrique

A la SNCF, les toutes premières électrifications des caténaires ont été réalisées en 1500V continu à partir de 1920. Cependant pour répondre à l'augmentation des puissances appelées par les engins de traction, l'augmentation du niveau de tension de la caténaire fut retenue. A partir de 1950, les électrifications nouvelles sont réalisées en 25 kV alternatif. Le courant est redressé dans l'engin de traction pour alimenter les moteurs séries à courant continu. De ce fait, la distance séparant deux sous-stations s'est accrue considérablement, passant de 15 km en 1500V continue à 65 km en moyenne pour l'alimentation 25 kV alternative. Le réseau ferroviaire de la SNCF comporte trois systèmes principaux d'alimentation des caténaires :

— Le courant continu 1500 V

— Le courant alternatif monophasé 25 kV 50 Hz

— Le courant alternatif 2 x 25 kV 50 Hz

3.2.1 L'alimentation en courant continu 1500V

L'alimentation en courant continu 1500V (figure 5) est la plus ancienne des tensions d'électrifications, elle est utilisée principalement dans le sud de la France sur le réseau ferré. Le courant 1500V est obtenu par transformation et redressement d'une tension triphasée (15 kV, 20 kV, 30 kV, 63 kV ou 90 kV).

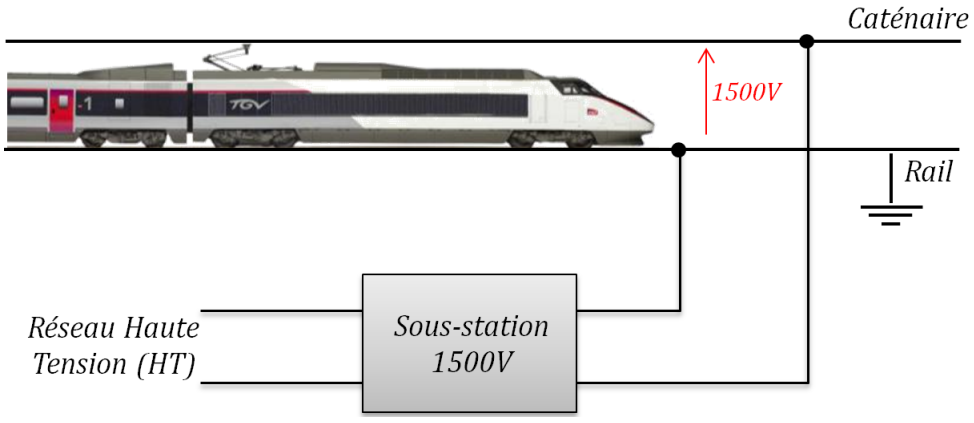


FIGURE 5 – Schéma simplifié du principe de l'alimentation en 1500V

Ce niveau de tension est plus faible que celui du 25 kV, à puissance équivalente il y aura donc plus de pertes par effet joule. Par conséquent, la répartition des sous-stations varie de 8 à 20 km maximum sur le long de la ligne électrifiée, pour combler les chutes de tension et réduire l'échauffement de la caténaire. Entre deux sous-stations, des postes de

sectionnement et des postes de mise en parallèle sont installés :

— Les postes de sectionnement - Poste S (figure 6) : en cas de défaut sur une portion de secteur (sous-secteur), les postes S permettent d'isoler uniquement le sous-secteur touché par le défaut, tout en permettant l'alimentation de l'autre partie du secteur.

— Les postes de mise en parallèle - Poste PMP (figure 7) ont pour rôle de diminuer la chute de tension entre la sous-station et l'engin moteur.

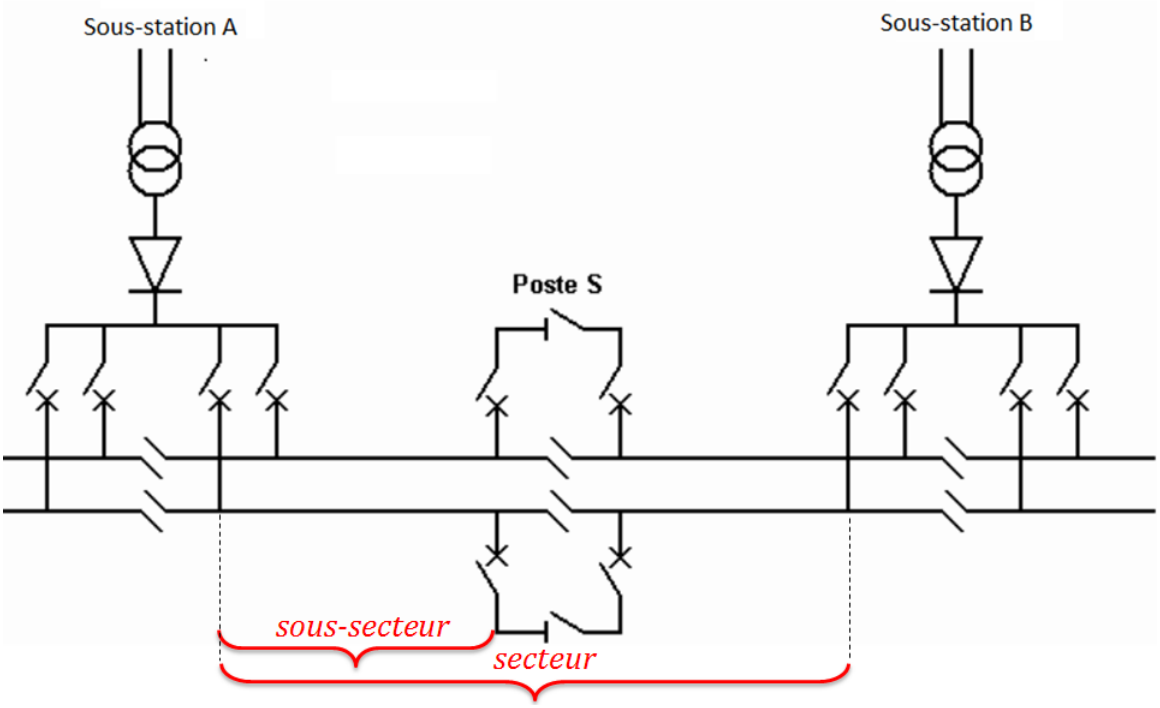


FIGURE 6 – Poste de Sectionnement

Il existe également des types de postes qui assurent les fonctions de sectionnement (S) et de mise en parallèle (PMP) qu'on appelle « Poste de Sectionnement et de Mise en parallèle (PMPS) ».

3.2.2 L'alimentation en courant alternatif monophasé 25 kV - 50 Hz

L'électrification monophasée 25 kV - 50 Hz (figure 8) est apparue dans les années 1950. Du fait de son infrastructure plus légère donc moins onéreuse, ce système d'alimentation est installé au détriment de l'alimentation 1500 V. L'alimentation HT des sous-stations se fait en 90 kV, 225 kV ou 400 kV. La tension 25 kV alternatif est obtenue par des transformateurs monophasés reliés à 2 phases d'un réseau haute tension.

Le long de la voie, la répartition des sous-stations varie de 40 à 90 km maximum. Entre deux sous-stations, des postes de sectionnement et des postes de mise en parallèle sont installés pour assurer le sectionnement et la mise en parallèle des caténaires. Par abus

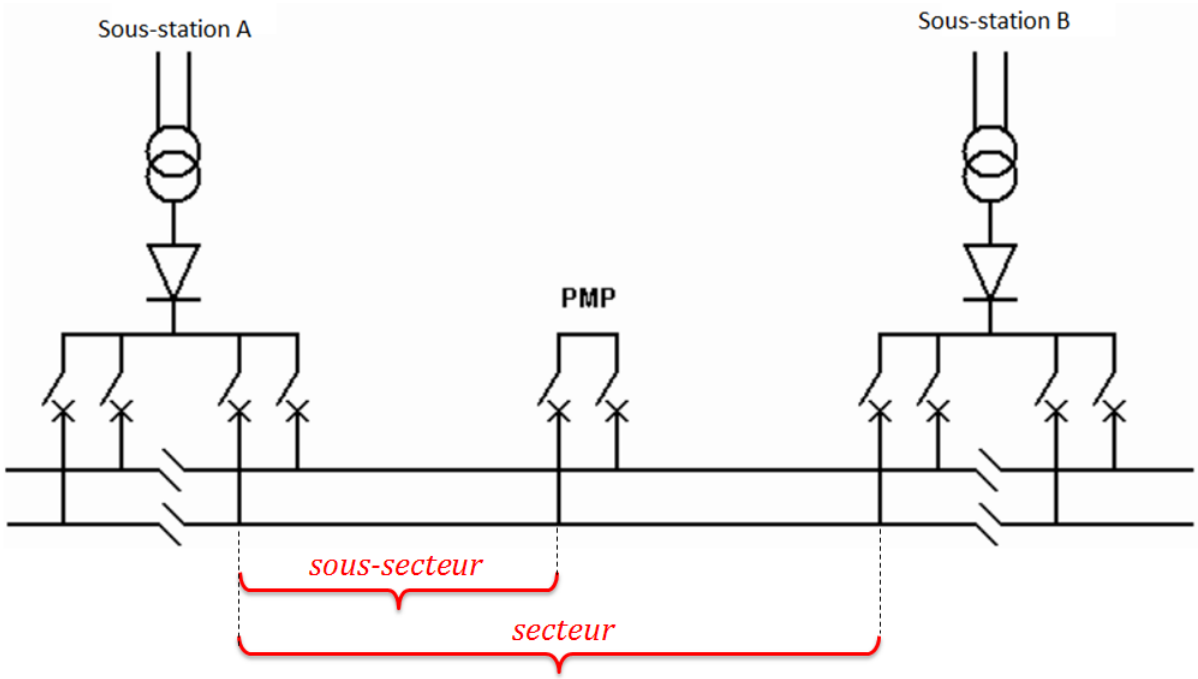


FIGURE 7 – Poste de Mise en Parallèle

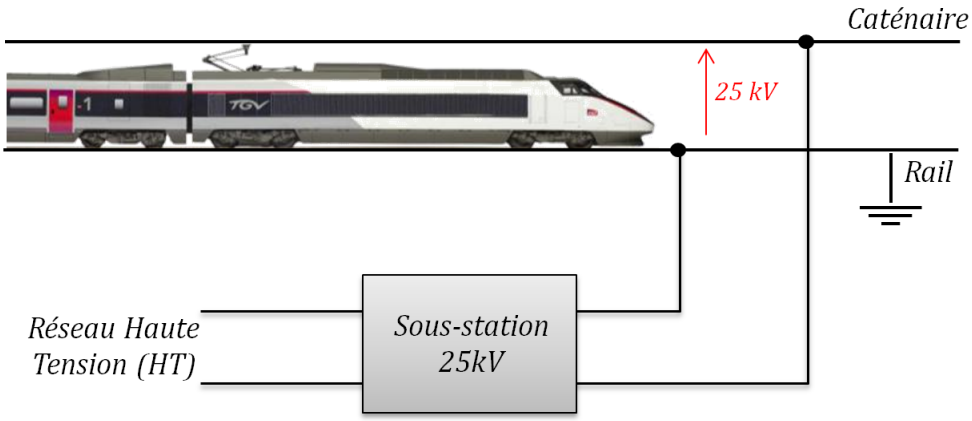


FIGURE 8 – Schéma simplifié du principe de l'alimentation en 25 kV

de langage, on parle d'une alimentation 25 kV mais pour assurer des niveaux de tensions acceptables, la tension au secondaire du transformateur de la sous-station est supérieure de 10% à la valeur voulue : c'est-à-dire qu'elle est en réalité de 27,5 kV.

3.2.3 L'alimentation en courant alternatif 2x25 kV - 50 Hz

Avec ce type d'électrification (figure 9), l'engin de traction est toujours alimenté en 25 kV (valeur efficace). Par contre l'énergie est transportée sous une tension de 50 kV, ce qui permet de diminuer les chutes de tension et d'augmenter les distances entre sous-stations (c'est-à-dire la longueur des secteurs). Les sous-stations sont équipées de transformateurs délivrant à leurs secondaires deux tensions alternatives efficaces nominales égales à 25

kV. Une des bornes est reliée à la caténaire. Le point milieu est connecté aux rails et sert de chaîne de retour traction. L'autre borne est reliée à un conducteur installé le long de la voie. Ce conducteur est par abus de langage appelé « feeder négatif » ou « feeder -25 kV ». La tension entre le « feeder -25 kV » et les rails est également de 25 kV efficace mais en opposition de phase. Des autotransformateurs connectent l'ensemble caténaire feeder pour récupérer une tension de 50 kV.

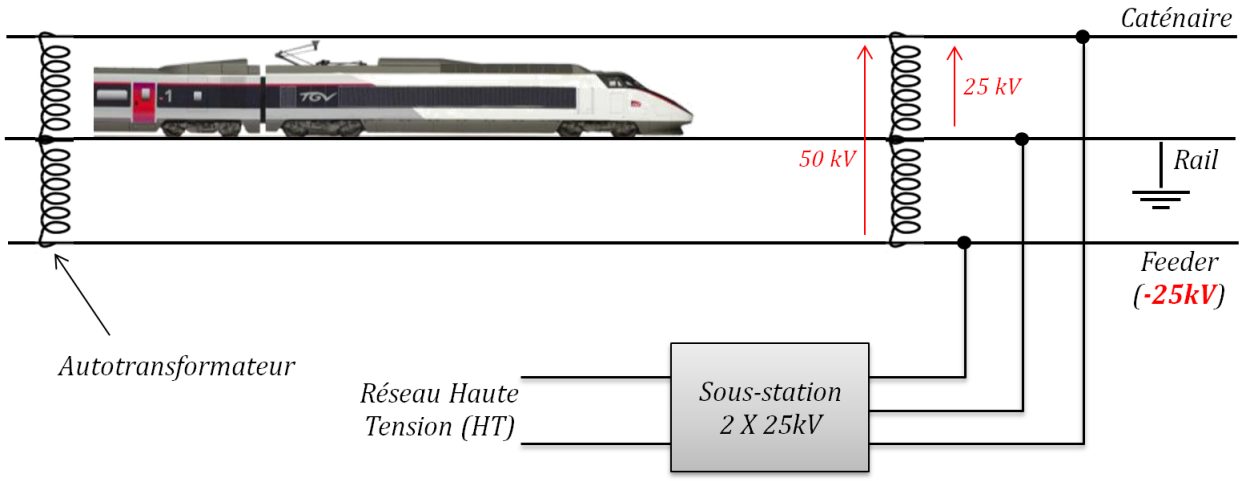


FIGURE 9 – Schéma simplifié du principe de l'alimentation en 2x25kV

3.2.4 Comparaison entre les alimentations 1500 V, 25 kV, et 2x25 kV

Historiquement, l'alimentation 1500V est apparue avant le 25 kV et tend à être démodé. En 25 kV, le courant circulant dans les caténaires est plus faible qu'en 1500V, ce qui réduit la section des caténaires. L'augmentation de la tension permet d'augmenter la distance entre chaque sous-station et de réduire les pertes d'énergie dans le transport.
En ce qui concerne l'alimentation 2x25 kV, elle entraîne des dépenses supplémentaires sur la ligne. En effet, elle nécessite l'utilisation d'un conducteur supplémentaire (le feeder), d'autotransformateurs, et de supports caténaires spécifiques. Sur une zone réduite, ce choix d'électrification est plus coûteux que le 25 kV classique. En revanche sur un secteur plus large, le 2x25 kV est moins coûteux. La ligne électrifiée nécessiterait moins de sous-stations (car l'espacement entre sous-stations est plus large). Dans ce cas, le coût d'implémentation des sous-stations (ou d'électrification de la ligne) sera réduit. La symétrie du 2x25 kV présente aussi l'avantage de réduire l'influence des perturbations électromagnétiques engendrées par la ligne. Cela permet ainsi de diminuer les dépenses liées aux protections des circuits de télécommunication proches des voies. Ce dernier aspect est souvent le fac-

teur qui favorise le 2x25 kV pour l'électrification des lignes classiques et celles à grande vitesse.

Un descriptif plus détaillé des types d'EALE et des types d'alimentations électriques des caténaires est présenté dans Coupat (2014).

4 Présentation du système de contrôle commande des EALE

4.1 Définition d'un système de contrôle commande

Un système de contrôle commande permet le pilotage d'un procédé industriel physique au travers des fonctions de commande, de surveillance et de supervision (figure 10) (Kesraoui, 2017). La commande a un rôle opérationnel qui consiste à faire exécuter un ensemble d'opérations pour agir sur le procédé physique. La surveillance a un rôle informationnel qui consiste à recueillir les signaux provenant du procédé et de la commande, reconstituer l'état du système, dresser les historiques et traiter les défaillances. La supervision a un rôle décisionnel qui permet d'optimiser, en présence ou non de défaillances, le fonctionnement du système.

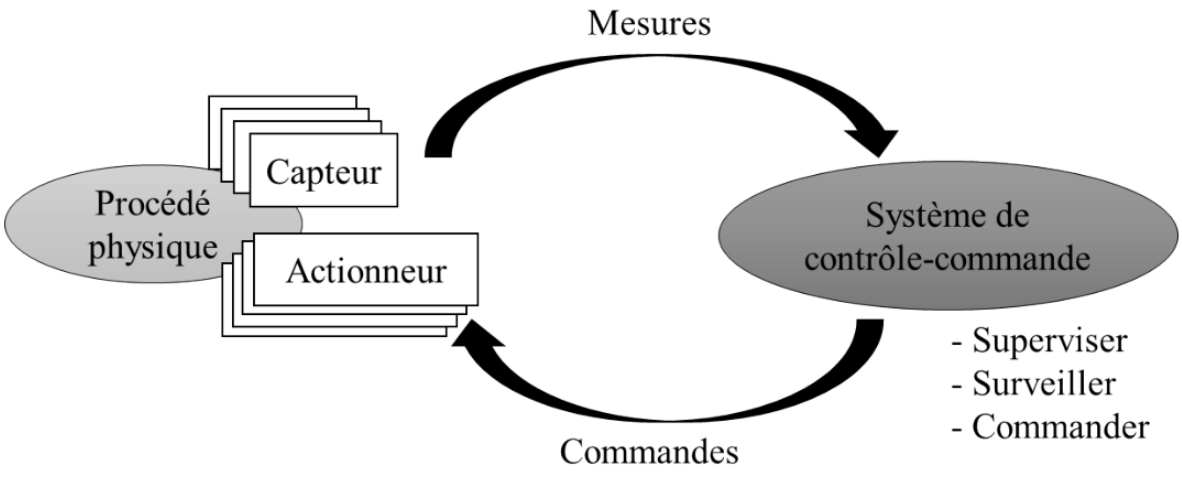


FIGURE 10 – Principe d'un système de contrôle commande

L'interaction du système de contrôle commande avec le procédé physique est réalisée d'une part par des observations sur le procédé à travers des capteurs (mesures), et d'autre part par des commandes envoyées par l'intermédiaire d'actionneurs (figure 10). Les capteurs permettent de transformer les grandeurs physiques du procédé matériel en signaux

électriques interprétables par le système de contrôle commande. Les actionneurs (moteurs, transformateurs...) permettent de transformer les commandes électriques du système de contrôle commande en ordres qui permettent d'agir sur le procédé physique en changeant son état.

4.2 Le système de contrôle commande des EALE

4.2.1 Architecture du système de contrôle commande

Une EALE est contrôlée localement par un système de contrôle/commande distribué et hiérarchisé (figure 11).

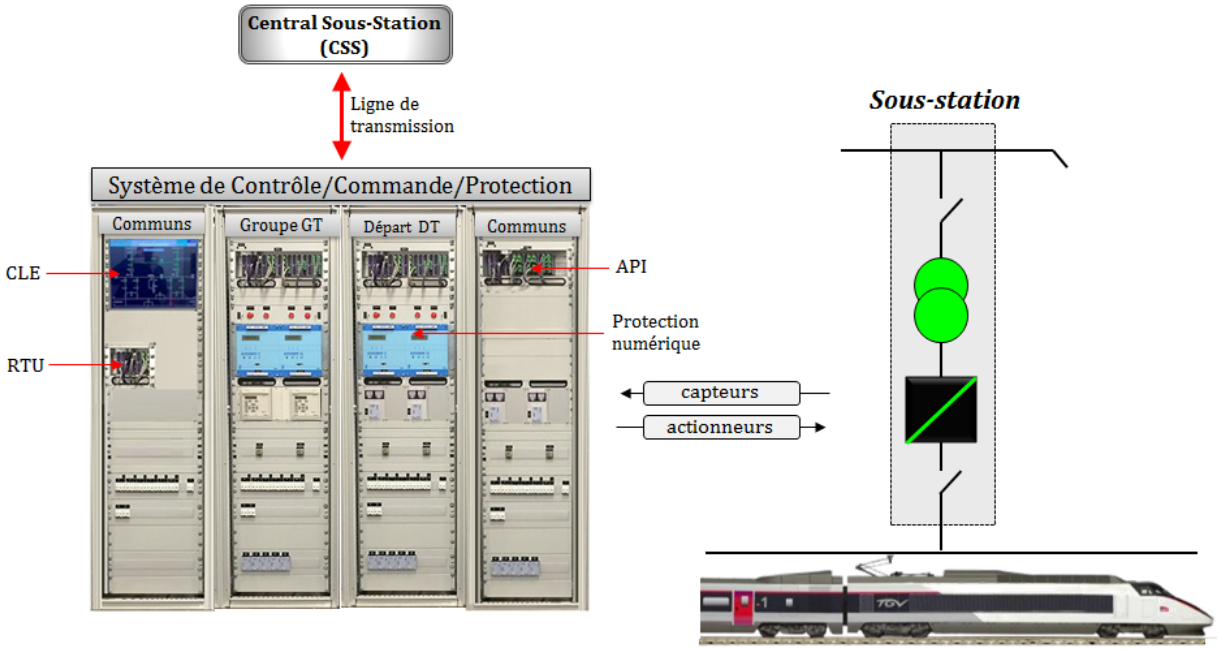


FIGURE 11 – Architecture du système de contrôle commande des EALE

Les systèmes de contrôle commande des EALE renferment globalement :

— Des Automates Programmables Industriels (APIs) auxquels sont assignés des tâches spécifiques. Chaque API, appelé également Abonné Automate (AA), est chargé de piloter un sous-ensemble de l'installation électrique (Groupe Traction GT, Départ Traction DT...) et communique avec les autres abonnés automates à travers un Réseau Local Industriel (réseau FIP - Factory Information Protocol) ;

— Des borniers à travers lesquels les entrées/sorties de l'EALE sont connectées aux APIs ou abonnés automates associés ;

— Une interface CLE (Commande Local Ecran) qui représente l'IHM local (Interface Homme Machine) permettant à l'opérateur de piloter l'installation électrique en

mode local. Ce CLE communique également avec les APIs à travers le Réseau Local Industriel ;

— Des protections numériques qui communiquent avec les APIs et protègent l'EALE contre les accidents électriques (défauts court-circuit, surcharge...). Cette fonction « protection » est indispensable pour la commande d'une EALE, d'où le terme « système de contrôle/commande/protection » utilisé par les chargés d'études automatisme de la SNCF ;

— Le RTU, qui est un automate programmable qui sert d'intermédiaire entre les APIs du système de contrôle commande et le CSS.

En effet, le système de contrôle commande des EALE échange également des informations avec un « Central Sous-Station » (CSS) à travers le protocole de communication HNZ (protocole propriétaire de la SNCF). Le CSS (figure 12) est chargé de superviser plusieurs EALE situées dans une zone d'action variant de 200 à 400km. Ils sont au nombre de 18 en France, et fonctionnent en permanence. Des Régulateurs Sous-Stations (RSS) sont chargés de surveiller l'état des EALE supervisées par le CSS et de donner les consignes aux agents chargés de la maintenance des installations.

FIGURE 12 – Exemple d'un Central Sous-Station

Un CSS est constitué :

— D'un Tableau de Contrôle Optique (TCO) qui permet la visualisation globale de la zone de contrôle avec l'état des appareils (disjoncteurs, sectionneurs. . .). Les TCO « classiques » ont des boutons de commande intégrés qui permettent la manipulation des équipements. Dans les CSS plus récents (figure 12), le TCO est un mur d'image et les commandes se font depuis un poste informatique.

— Des postes de régulation à partir desquels les Régulateurs Sous-Stations pilotent les installations à distance.

— Un local technique dans lequel des armoires permettent la communication avec les postes distants et les sous-stations (Télécommande).

Grâce au système de contrôle commande, l'opérateur peut commander en ligne l'installation électrique suivant 3 modes différents : le mode local (en envoyant des commandes depuis le CLE), le mode distant (depuis le CSS), et le mode local-individuel (depuis un Poste de Commande Local - PCL connecté au système de contrôle commande). Il existe plusieurs types d'abonnés automates dans un système de contrôle commande :

— l'Abonné « **AAGT** » pour la commande d'un Groupe Traction GT,
— l'Abonné « **AADT** » pour la commande d'un Départ Traction DT,
— l'Abonné « **AACom** » pour la commande de la partie HT et/ou des équipements auxiliaires (batteries de condensateurs, appareils de pontage des voies...),
— l'Abonné « **AARTU** » qui sert d'intermédiaire entre les APIs du système de contrôle commande et le CSS,
— l'Abonné « **AATCO** » qui permet d'archiver les données de télégestion et d'afficher les informations de l'EALE sur l'interface de supervision locale de l'installation (CLE).

Le système de contrôle commande respecte des normes spécifiques au domaine des EALE (Coupat, 2014), notamment celles liées à la sûreté de fonctionnement (IEC-60870-4, 2013) : la Fiabilité, la Maintenabilité, la Disponibilité et la Sécurité (FMDS).

La Fiabilité : Cet indicateur caractérise l'aptitude d'un système à accomplir une fonction requise, dans des conditions données, pendant un intervalle de temps déterminé. Une mesure pratique de la fiabilité est le MTBF (Mean Time Between Failure) qui est la durée moyenne entre deux défaillances consécutives d'une entité réparée. Selon la norme (IEC-60870-4, 2013), les équipements des EALE doivent répondre à la classe de fiabilité R3, soit un MTBF supérieur à 10 000 h. Cette valeur se réfère à la fiabilité d'un équipement en mode simple pour une configuration maximale à une température externe de 55^{o}C.

La Maintenabilité : Le MTTR (Mean Time To Repair) est le temps de dépannage total mesuré pour un groupe d'unités divisé par le nombre total de défaillances. Ce critère quantifie la maintenabilité d'un système. Il est l'addition des composantes suivantes :

— le temps administratif : période de temps entre la détection d'une panne et la notification au service de maintenance,

— le temps de transport : période de temps entre la notification au service de maintenance et l'arrivée sur place du personnel de maintenance muni du matériel nécessaire,
— le temps moyen de réparation MRT (Mean Repairing Time).

Les équipements des EALE doivent répondre aux exigences de la classe de maintenabilité M3 pour laquelle le temps de réparation est ramené à 15 minutes.

La Disponibilité : La disponibilité caractérise l'aptitude d'un système à être en état d'accomplir une fonction requise, dans des conditions données, à un instant donné ou pendant un intervalle de temps déterminé, en supposant que la fourniture des moyens extérieurs soit assurée. La disponibilité intrinsèque est exprimée numériquement à l'aide de la formule :

$$A = \frac{MTBF}{MTBF + MTTR} \times 100 \tag{I.1}$$

L'ensemble de l'équipement de télécommande doit répondre à la classe A3 (Disponibilité supérieure à 99,95%).

La Sécurité des abonnés automates : Ce facteur concerne les abonnés automates et est intrinsèquement lié à la conception de l'équipement. Il devra répondre à la norme (NFC-63850, 1988) relative aux automates programmables industriels. En cas de défaut interne, l'équipement doit réagir comme suit :

— aucun ordre intempestif ne devra être émis,
— l'organe en défaut devra être bloqué jusqu'à la prochaine réinitialisation du système.

Le comportement de l'équipement doit correspondre au type 3 prévu dans la norme (NFC-63850, 1988), c'est-à-dire qu'en cas de défaut, celui-ci doit être détecté et signalé, et les sorties doivent être réinitialisées.

4.2.2 Structure des programmes automates des EALE

Les abonnés automates du système de contrôle commande des EALE sont des Automates Programmables Industriels de type ACS25 (Automate à Conduite Sécurisée). Ces automates sont fabriqués par le groupe LEROY automation, constructeur français d'équipements d'acquisition, d'entrées/sorties déportées et automates durcis, pour les environnements sévères. Ils sont programmés avec le logiciel STRATON (www.copalp.com).

Les langages de programmation utilisés pour la commande des EALE sont décrits par la norme IEC 61131-3 (2013) :

- Le **FBD (Function Block Diagram)** : est un langage graphique qui permet de réaliser des équations complexes à partir de fonctions élémentaires représentées par des blocs fonctionnels. Les variables d'entrées et de sorties sont connectées aux blocs par des arcs de liaison. Une sortie d'un bloc peut être connectée à une entrée d'un autre bloc.
- Le **LD (Ladder Diagram)** : ce langage graphique est essentiellement dédié à la programmation d'équations booléennes, il est proche du schéma électrique et inspiré des schémas à relais.
- Le **ST (Structured Text)** : est un langage textuel de haut niveau (type Pascal) dédié à la description de procédures complexes, difficilement modélisables avec les langages graphiques. C'est le langage par défaut pour la programmation des actions et des réceptivités du langage SFC.
- L'**IL (Instruction List)** : ce langage textuel de bas niveau est un langage à instruction comparable au langage assembleur. Il n'est cependant pas utilisé pour la programmation des EALE.
- Le **SFC (Sequential Function Chart)** : qui est une représentation graphique issue du langage GRAFCET (IEC-60848, 2002). Ce langage de haut niveau permet la programmation des procédés types séquentiels, en s'appuyant sur l'un des quatre autres langages pour la programmation des étapes et des transitions.

Les programmes API sont globalement structurés en plusieurs tâches, exécutées dans l'ordre et de manière cyclique durant chaque cycle automate. Ces tâches peuvent être classées (pour chaque API) en 5 catégories :

Lecture des entrées (langages FBD et ST)

- « Acquisition des entrées physiques » : cette section permet de récupérer toutes les entrées physiques telles que les états ouverts et fermés des appareils, les défauts observés ... et de les affecter à des variables de l'automate ;
- « Lecture des paramètres du PCL » : récupère les paramètres saisis (ou modifiés) en ligne par l'opérateur, tels que les temps d'attente d'ouverture ou de fermeture d'un appareil ;

— « Acquisition des commandes opérateurs » : récupère à travers le RLI, les commandes opérateurs destinés à l'EALE (commandes distantes, locales, et local-individuelles).

Traitement des entrées (langage LD)

— Construction des observateurs à partir des données récupérées lors de la tâche précédente ;
— Calcul des positions des appareils (disjoncteurs, sectionneurs...) ;
— Calcul des défauts.

Aiguillage des commandes (langage SFC)

— Selon le mode de commande actuel de l'EALE (mode distant, local, ou local-individuel), cette tâche permet d'aiguiller les ordres envoyés par l'opérateur vers l'installation. Par exemple, lorsque l'EALE est pilotée en mode local, les commandes distantes (ou local-individuelles) ne seront pas prises en compte par le programme.

Mise à jour des programmes (langage SFC)

— Évolution de tous les programmes de commande des appareils

Traitement des sorties (langages FBD et LD)

— Remise à zéro des sorties en cas de défaut API ;
— Mise à jour et écriture des sorties ;
— Envoi des informations à travers le RLI pour la télégestion.

Mises à part ces tâches cycliques, il existe une autre section renfermant un ensemble de blocs fonctionnels écrits en langage SFC (figure 13). Ces blocs fonctionnels sont ensuite instanciés dans la section « Mise à jour des programmes » pour former le programme principal.

5 Workflow d'un projet d'automatisation des EALE

5.1 Présentation du workflow traditionnel

Le workflow traditionnel d'un projet d'automatisation d'EALE est globalement constitué de deux phases (figure 14) :

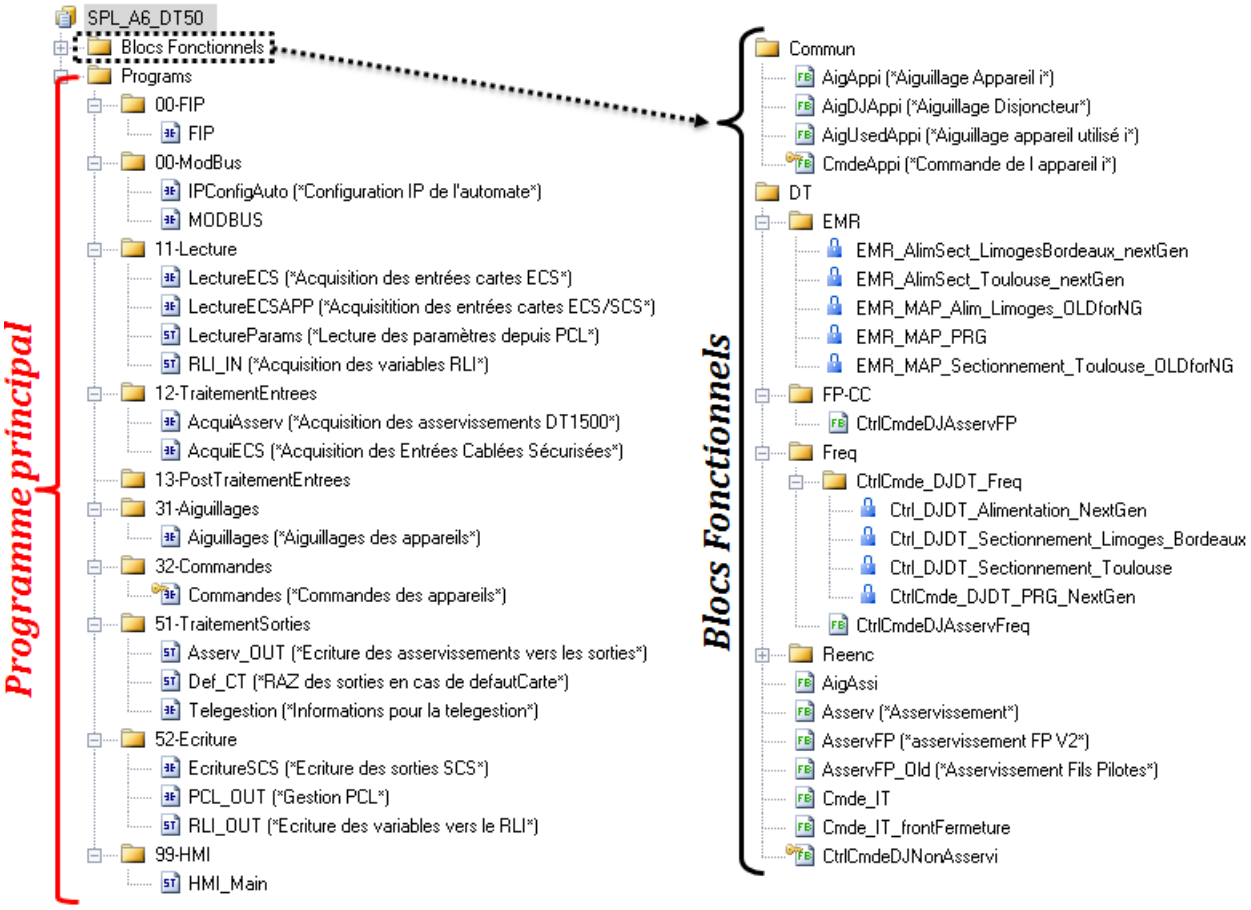


FIGURE 13 – Exemple de programme automate édité avec Straton

— La Conception des livrables du projet (programmes, cahier de recettes, schémas électriques...) ;

— La Vérification et la Validation (V&V) du système de contrôle commande en usine et sur site.

Àprès l'étude du cahier des charges du projet (spécifications fonctionnelles et contraintes de l'installation), les chargés d'études ont pour mission de concevoir les livrables du projet. Cette phase consiste dans un premier temps à réaliser les schémas électriques décrivant la structure de l'EALE et l'architecture du système de contrôle commande associé. Ces schémas électriques seront transmis à des intégrateurs pour la conception et le câblage des armoires de contrôle commande des EALE. Ensuite, les chargés d'études procèdent à la conception des programmes automates destinés à la commande de l'installation. Pour cela, ils réutilisent des programmes automates dont l'EALE associée est similaire à celle en cours d'étude, et les adaptent en fonction du nouveau cahier des charges. En effet, l'adaptation manuelle des livrables d'un projet similaire requiert moins d'effort pour les chargés d'études. Enfin, la dernière étape consiste à établir (tou-

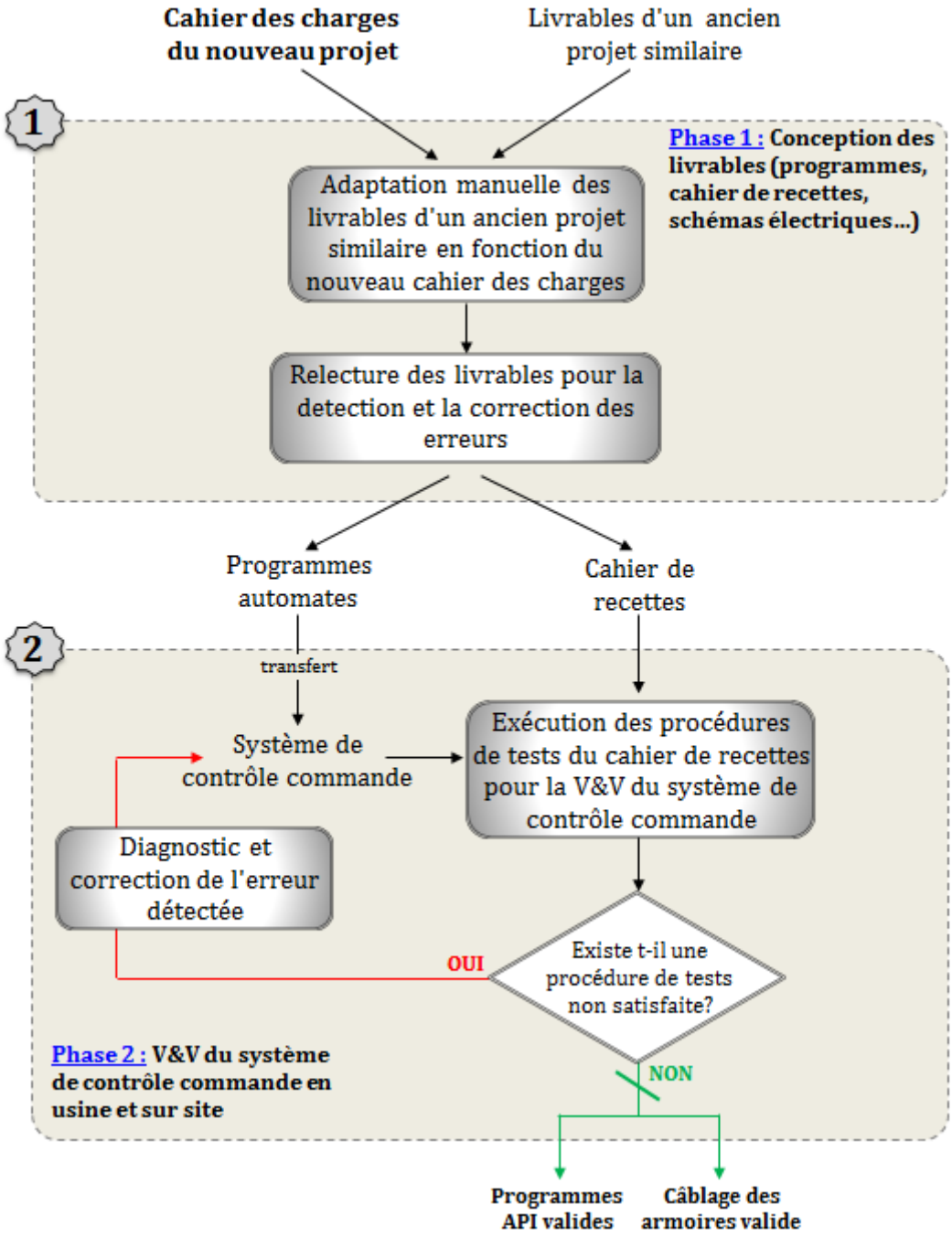


FIGURE 14 – Workflow traditionnel d'un projet d'automation d'EALE

jours par adaptation manuelle d'un projet similaire) un ensemble de procédures d'essais regroupées au sein d'un cahier de recettes, qui permettra de vérifier et de valider le bon fonctionnement du système de contrôle commande. Après la conception des livrables du projet (programmes, schémas électriques...), les chargés d'études procèdent à une relecture des documents pour détecter et corriger d'éventuelles erreurs avant la phase de V&V du système de contrôle commande.

La deuxième phase du projet consiste à vérifier et valider le système de contrôle commande en y exécutant manuellement les procédures de tests du cahier de recettes. Les

chargés d'études ne peuvent valider un système de contrôle commande que lorsque toutes les procédures de tests du cahier de recettes y ont été exécutées avec succès. Dans le cas contraire, cela signifierait que le système de contrôle commande contient des erreurs, et que celles-ci doivent être diagnostiquées et corrigées en usine. Ces essais ont donc lieu dans un premier temps en usine pour valider les programmes API et le câblage des armoires. Le programme API d'une EALE n'est considéré comme valide que si le comportement de l'EALE commandée est conforme aux attentes du cahier des charges. Quant au câblage des armoires de contrôle commande des EALE, la validation consiste à s'assurer que :

— les entrées/sorties de l'EALE ont été correctement connectées aux borniers du système de contrôle commande,

— les protections numériques ont été convenablement paramétrées de sorte qu'elles puissent protéger l'EALE en cas d'incident électrique,

— le Réseau Local Industriel (RLI) a été correctement établi entre les différents abonnés automates du système.

Après une première phase de correction, les essais ont lieu sur site pour valider le système de contrôle commande en présence de l'installation électrique réelle. La phase de V&V sera davantage détaillée dans le chapitre 2.

Workflow traditionnel

Phase 1 : Conception des livrables du projet	o Réalisation des schémas électriques------- 60h o Conception des programmes API---------- 50h o Rédaction du cahier de recettes---------- 40h o Relecture et correction des livrables------- 10h **Total des heures---------- 160h**
Phase 2 : V&V du système de contrôle commande	o Vérification du système de contrôle commande en usine ---------- 40h o correction---------- 20h o Validation du système de contrôle commande sur site---------- 40h **Total des heures---------- 100h**

TABLEAU 1 – Les étapes et durées du workflow traditionnel

Durant un projet d'automatisation d'EALE, le chargé d'études doit respecter des deadlines. Il consacre en moyenne 160 heures pour la première phase du projet, et 100 heures pour la deuxième phase (tableau 1). Une étude menée par Coupat (2014), à travers un questionnaire anonyme, a montré que la charge de travail des chargés d'études de la

SNCF est très élevée. De plus, les deadlines imposées sont souvent sources de stress supplémentaires. Ces mauvaises conditions de travail peuvent par conséquent compromettre la qualité des livrables du projet, pour les raisons suivantes :

- Des livrables longs et fastidieux à établir (cahier de recettes, programmes API...)
- La fatigue générée par les efforts et le stress lié à la deadline de fin du projet qui exposent le chargé d'études à une surcharge mentale, ce qui ne le met pas à l'abri d'erreurs humaines durant les phases de conception et de V&V ;
- La gestion en parallèle de plusieurs projets par le chargé d'études ;
- L'existence des tâches répétitives, qui peut être source d'erreurs et peut entraîner un sentiment de monotonie chez le chargé d'études ;

L'ingénierie SNCF est aujourd'hui préoccupée par trois grandes problématiques :

- L'amélioration des conditions de travail des chargés d'études automatismes ;
- L'optimisation de la production ;
- La sûreté du fonctionnement.

À cette fin, un projet de recherche (Coupat, 2014) a été lancé en partenariat avec l'ingénierie SNCF et le laboratoire CReSTIC de l'université de Reims Champagne Ardenne. Ce projet avait principalement pour objectif d'optimiser la première phase du workflow d'un projet d'automatisation d'EALE, à savoir la conception des livrables (figure 15). Cette étude a abouti au développement d'un outil nommé ODIL permettant de générer automatiquement des livrables d'un projet (programmes API et cahier de recettes), à partir de la description graphique de l'EALE sur l'interface du logiciel. La première phase du workflow traditionnel (conception manuelle des livrables) est alors remplacée par une nouvelle méthodologie (génération automatique des livrables) présentée dans la section 5.2.

5.2 Nouveau workflow d'un projet d'automatisation

5.2.1 Génération automatique des livrables avec ODIL (phase 1)

À l'issue du projet de recherche de Coupat (2014), une nouvelle méthodologie de conception des livrables a été développée pour les chargés d'études à l'occasion des projets d'automatisation d'EALE. Cette méthodologie (figure 16) basée sur un outil nommé ODIL, permet de générer automatiquement les livrables à partir d'une description graphique de l'installation et du système de contrôle commande dans l'interface du logiciel

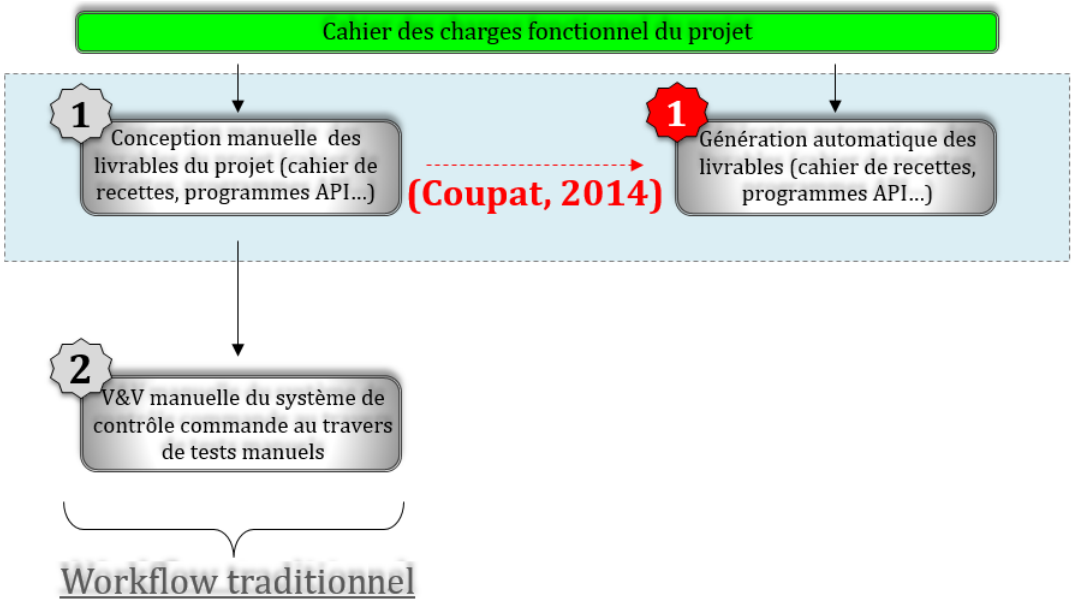


FIGURE 15 – Optimisation de la première phase du workflow traditionnel (Coupat, 2014)

ODIL.

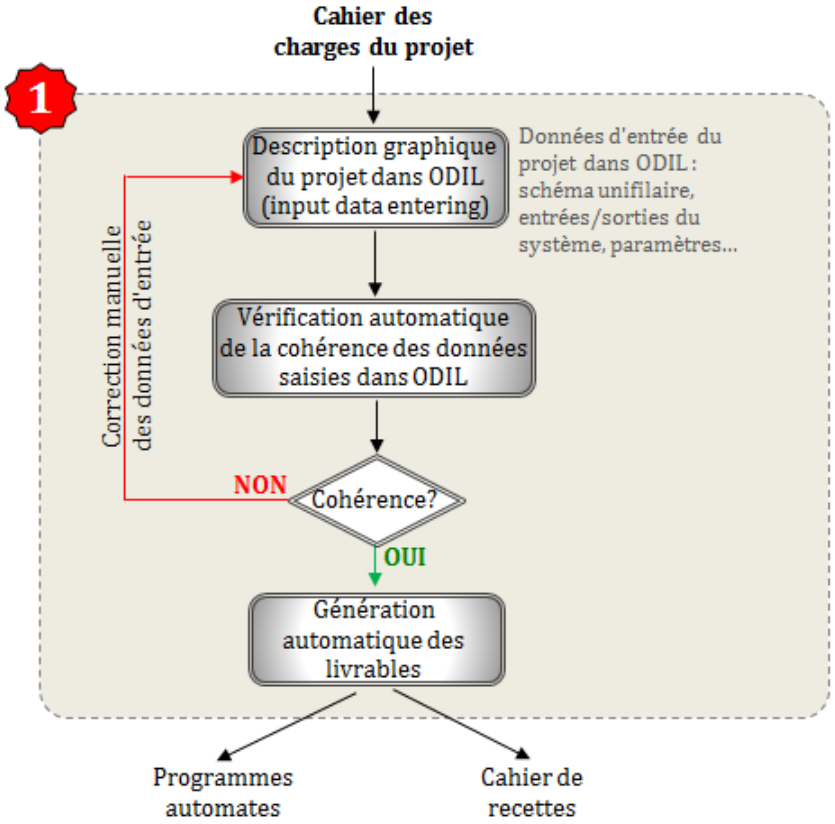


FIGURE 16 – Méthodologie de génération automatique des livrables

Pour générer les livrables d'un projet avec l'outil ODIL, le chargé d'études doit respecter une procédure qui se compose globalement de 5 étapes :

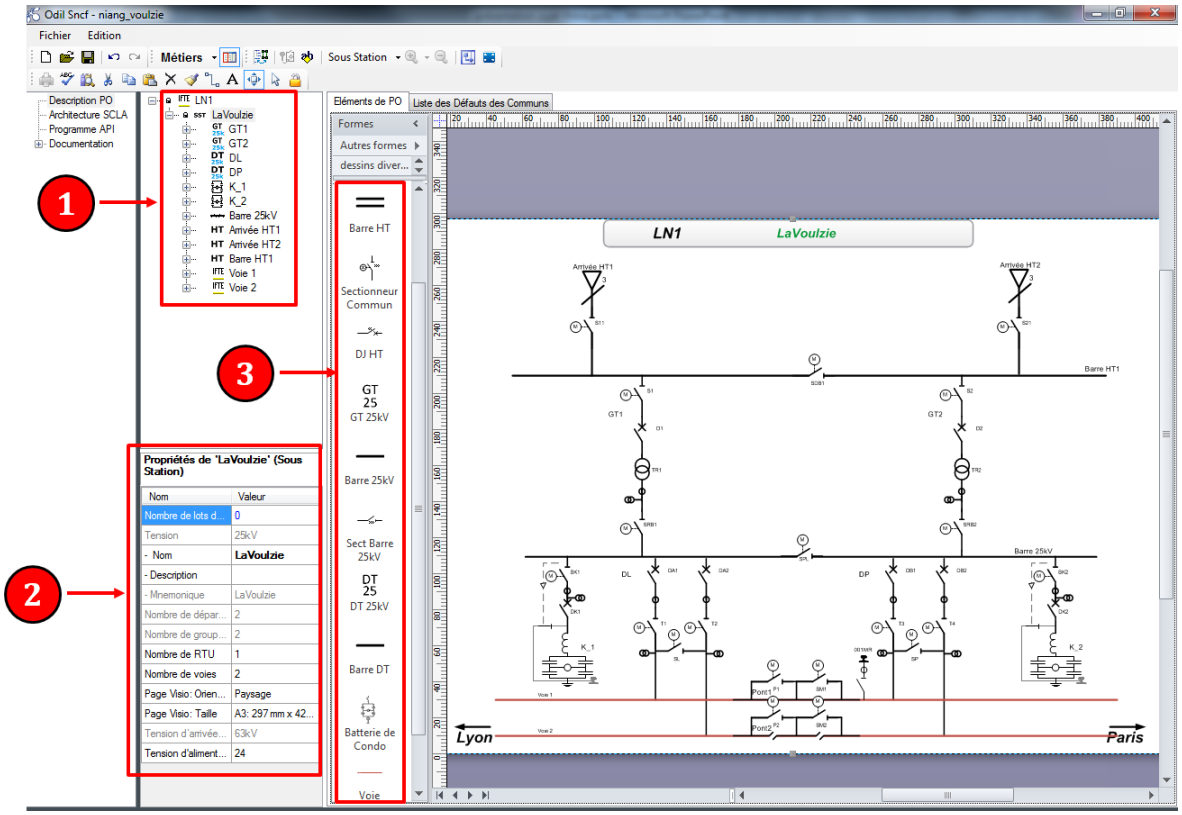


FIGURE 17 – Exemple de représentation d'un schéma unifilaire sous ODIL

Réaliser le schéma unifilaire de l'installation électrique (figure 17) Cette étape consiste à reconstituer intégralement dans l'interface du logiciel, le schéma unifilaire de l'EALE cible, en instanciant les différents sous-ensembles de partie opérative (Groupe Traction, Départ Traction...) nécessaires et prédéfinis dans ODIL (dans la zone 3 de la figure 17). Prenons en exemple le schéma unifilaire décrit à la figure 17. D'après le cahier des charges du projet, cette sous-station est principalement constituée de deux arrivées HT, deux batteries de condensateurs, deux groupes traction (GT), deux départs traction (DT), deux voies munies d'appareils de pontage de type série, une barre HT et une Barre 25 kV. La liste des SePO instanciés s'affiche également dans la zone 1 de la figure 17. Il est à noter que le nombre de chacun de ces SePO peut changer d'une sous-station à l'autre, avec cependant des normes à respecter. Toutefois la force de ce logiciel de génération réside dans le fait qu'il peut s'adapter à n'importe quelle configuration d'installation normée, et générer les livrables adéquats.

Définir les propriétés de l'EALE conformément au cahier des charges (zone 2 figure 17). Les sous-ensembles de partie opérative renferment des propriétés spécifiques qui peuvent varier d'une sous-station à l'autre en fonction du cahier des charges. Quelques

exemples de propriétés sont présentés ci-dessous :

- la tension d'alimentation (25 kV, 2 x 25 kV, ou 1500 V),
- la présence ou non de disjoncteurs et de sectionneurs dans l'arrivée HT,
- la nature des appareils de pontage au niveau des voies,
- le type de commande des appareils,
- ...

Pour renseigner les propriétés de l'installation, le chargé d'études sélectionne les objets instanciés dans ODIL et les paramètre convenablement par l'intermédiaire de la zone 2 de la figure 17.

Définir les défauts à observer. Pour assurer la sécurité de l'installation, il est nécessaire de faire l'acquisition des états des défauts afin que le système de contrôle commande puisse réagir en cas d'incident. Ainsi à chaque SePO, est associé une liste de défauts à surveiller, qui peuvent être ou non activés sous ODIL en fonction du cahier des charges. En cas de désactivation du défaut, celui-ci ne sera pas lu par l'automate associé. La figure 18 montre un exemple d'une liste des défauts actifs (en vert) et inactifs (en rouge) pour un des Départs Tractions de la sous-station.

Définir l'architecture du système de contrôle commande. Cette étape consiste à définir le nombre d'abonnés automates qui composent le système de contrôle commande. Ensuite chaque abonné automate sera virtuellement connecté au sous-ensemble de partie opérative associé, conformément au cahier des charges. Le nombre de cartes d'entrées/sorties doit également être défini pour chaque API déclaré.

Câblage des entrées/sorties de l'EALE sur le système de contrôle commande. La dernière étape de la description du projet dans ODIL consiste à câbler toutes les entrées/sorties de l'EALE aux différents APIs du système de contrôle commande (à travers les cartes d'entrées/sorties virtuelles).

À l'issue de la saisie des données d'entrée dans ODIL, la génération automatique des livrables peut débuter. Toutefois, il est nécessaire qu'ODIL vérifie la cohérence des données saisies par l'utilisateur, sans quoi la génération des livrables est impossible. Par définition, les données d'entrée d'un projet sont cohérentes seulement si elles respectent un ensemble de règles définies dans une structure de données définie dans ODIL (nommée « standard EALE »). Parmi ces règles, nous pouvons citer quelques exemples :

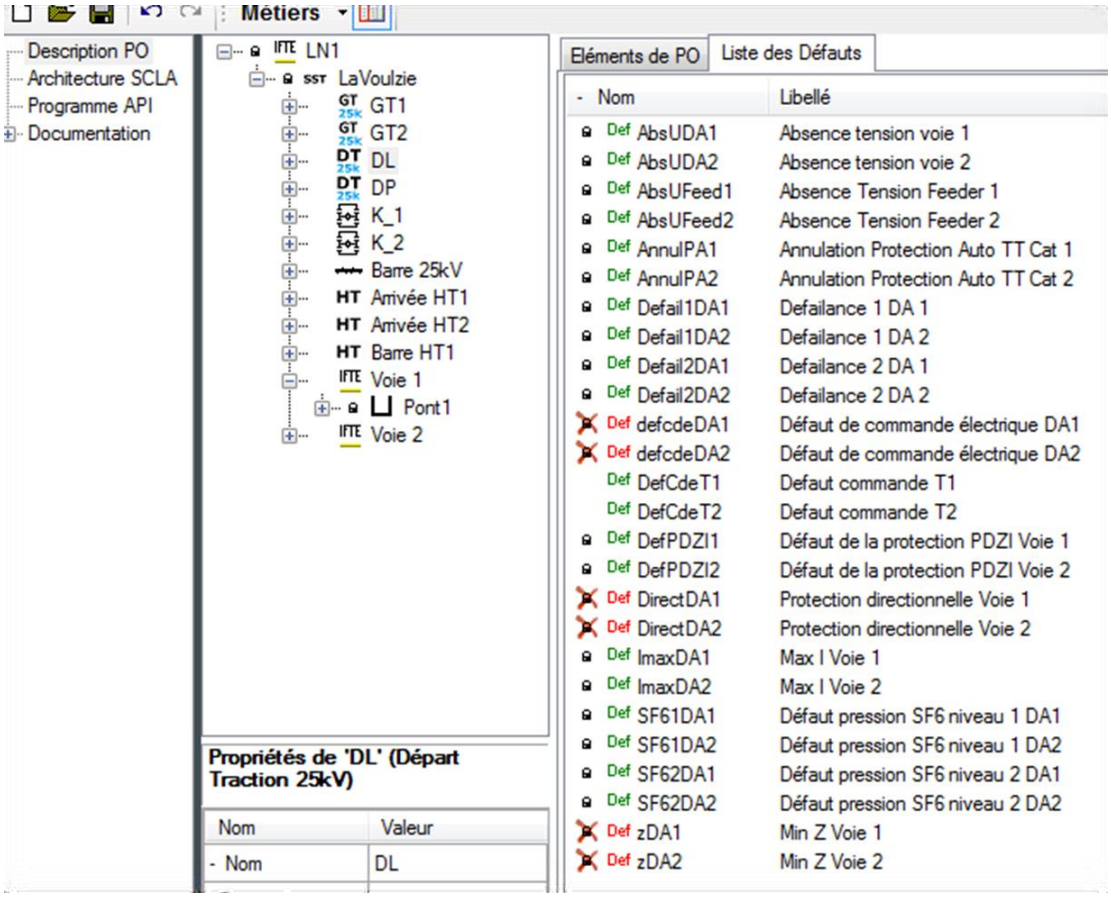


FIGURE 18 – Exemple de paramétrage d'une liste de défauts

— chaque EALE décrit dans ODIL doit obligatoirement contenir au moins un Groupe Traction, un Départ Traction, et une arrivée Haute Tension,
— un Départ Traction doit toujours être connecté à une caténaire,
— chaque sous-ensemble de partie opérative doit être associé à un abonné automate,
— toutes les entrées/sorties de l'EALE doivent être connectées au système de contrôle commande ;
— ...

Lorsque les données d'entrée du projet ne respectent pas ces règles de syntaxe, ODIL en informe l'utilisateur à travers des messages d'erreurs afin que celles-ci puissent être corrigées (figure 16). Dans le cas contraire, la génération automatique des livrables peut débuter. Pour cela, la structure de données d'ODIL (« standard EALE ») renferme des *templates* de génération (Coupat, 2014). Ces *templates* sont des codes sources développés en langage XML qui permettent de générer les fiches de tests et les programmes API associés à chaque SePO. Elles renferment des requêtes de base de données exécutées par ODIL au moment de la génération des livrables. Un exemple de *template* (générant un bout de programme en LADDER) est présenté à la figure 19. Ce *template* de génération

permet de rajouter sur le programme API de l'automate associé à ce *template*, une section permettant de récupérer le défaut nommé « ImaxDK » (s'il était activé dans ODIL avant la génération) et de l'associer à un observateur.

```
1  <OdilExportXml Fonction="SetVarEvt" VarEvtName="CPT" Value="1">
2  </OdilExportXml>
3  <OdilExportXml Fonction="Enumeration" Requete="SELECT_FROM(TypeCarteTSS::ModeleES;oui)" Tri="NumAuto">
4  <OdilExportXml Fonction="CreateNode">
5  <Condition>[TypeConnexion]!=[EMPTY()]</Condition>
6  <Node>
7  <OdilExportXml Fonction="SetVarEvt" VarEvtName="CNX" Value="0">
8  </OdilExportXml>
9  <OdilExportXml Fonction="CreateNode">
10 <Condition>[FROM_HANDLE([LIST_ID_AT([Connexion];0)])::RoleID]==ImaxDK</Condition>
11 <Node>
12 (*I Max K*)
13
14 SOR  [0,[VAR_EVT(CPT)]] (**) (**)
15  BST  XIC  [1,0] (*ImaxDK_1*) (*I max (1) DK (16 TSS)*)
16      NXB  XIO  [1,1] (*ImaxDK_2*) (*I max (2) DK (16 TSS)*)
17  BND
18
19  BST  OTS  [2,0] (*ImaxDK*) (*I max DK (FIP)*)
20      NXB  OTE  [2,1] (*ImaxDKF*) (*I max DK fugitif*)
21  BND
22
23 EOR [3,0]
24
25 <OdilExportXml Fonction="SetVarEvt" VarEvtName="CNX" Value="1">
26 </OdilExportXml>
27 </Node>
28 </OdilExportXml>
```

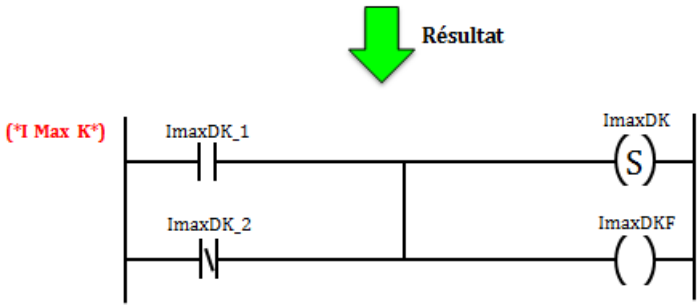


FIGURE 19 – Exemple de *template* de génération

Au moment de la génération des livrables, ce *template* est exécuté par ODIL de la manière suivante :

- sélection de toutes les entrées de l'automate sur lesquelles sont câblées un défaut, en les triant dans l'ordre croissant selon leur numéro,
- parmi les entrées sélectionnées, extraire (si elle existe) l'entrée sur laquelle est câblée un défaut nommé « ImaxDK »,
- Si ce défaut existe, alors rajouter dans le programme automate, le code en LD (décrit par les lignes 12 à 23 du code XML de la figure 19) qui permet au programme automate de lire le défaut « ImaxDK ».

Le résultat du code API généré est également présenté en bas de la figure 19.

Lors de la génération des livrables, ODIL sélectionne dans la structure de données « Stan-

dard EALE », les *templates* requis pour reconstituer intégralement les programmes API (et le cahier de recettes) de chaque abonné automate du système de contrôle commande. Le principe de la génération automatique des livrables est détaillé dans Coupat (2014). Après la génération des livrables, les chargés d'études procèdent à une relecture des programmes et du cahier de recettes pour valider ces documents avant la V&V en usine. Cette phase de relecture permet également aux chargés d'études d'avoir un regard critique sur les livrables générés, dont le retour permet de corriger et d'améliorer les *templates* de génération.

Actuellement, le logiciel ODIL est activement utilisé pour la génération des livrables d'un projet au sein du département « ingénierie et projets » de la SNCF. Toutefois, du fait de l'existence de quelques rares types d'installations non standardisables (environs 5% des types d'EALE existants), l'outil ODIL ne peut être utilisé que pour environ 95% des types de projets. Néanmoins cela n'enlève en rien les améliorations que cet outil a apportées sur le workflow traditionnel des chargés d'études. Le tableau 2 présente une comparaison entre l'ancien et le nouveau workflow basé sur la génération automatique.

	Workflow traditionnel	**Nouveau workflow**
Phase 1 : Conception des livrables du projet	○ Réalisation des schémas électriques------ 60h ○ Conception des programmes API----------50h ○ Rédaction du cahier de recettes---------- 40h ○ Relecture et correction des livrables------ 10h **Total des heures----------160h**	○ Réalisation des schémas électriques----- 60h ○ Génération des programmes API et du cahier de recettes avec ODIL---------- 5h ○ Relecture des livrables générés---------- 5h **Total des heures ---------- 70h**
Phase 2 : V&V du système de contrôle commande	○ Vérification du système de contrôle commande en usine ---------- 40h ○ correction---------- 20h ○ Validation du système de contrôle commande sur site---------- 40h **Total des heures----------100h**	○ Vérification du système de contrôle commande en usine ---------- 40h ○ correction---------- 20h ○ Validation du système de contrôle commande sur site---------- 40h **Total des heures----------100h**

TABLEAU 2 – Comparaison entre l'ancien et le nouveau workflow

Grâce à la génération automatique, les tâches de rédaction manuelle des programmes et du cahier de recettes sont remplacées par la description graphique du projet dans ODIL suivie d'une relecture des livrables générés. Étant donné que les tâches manuelles et répétitives sont sources d'erreurs humaines, la génération automatique permet de pallier ces inconvénients. De plus, elle supprime les phases de sous-charge mentale en les remplaçant par des phases cognitives dont l'enchaînement permet d'éviter une surcharge mentale. La saisie unique dans un environnement logiciel unique permet aux chargés d'études de rester concentrés et de ne pas perdre en performance (Coupat, 2014).

Comme le montre le tableau 2, la durée de la phase 1 est quasiment réduite de moitié (de 160 heures à 70 heures), ce qui montre que cette approche réduit également le temps d'études nécessaire pour un projet d'automatisation.

Ainsi, les attentes de la solution ODIL ont été atteintes :

— l'homogénéité des projets,

— l'augmentation de la productivité,

— la réduction des erreurs de conception,

— l'allègement de la charge de travail,

— la réduction du temps requis pour un projet,

— ...

Néanmoins comme le montre le tableau 1, cette solution n'a d'impact que sur la phase 1 du workflow traditionnel. La phase 2 du workflow (100 heures) n'a donc fait l'objet d'aucune amélioration. De plus, cette deuxième phase présente beaucoup d'inconvénients pour les chargés d'études, elles sont résumées à la section suivante.

5.2.2 Vérification & Validation (V&V) du système de contrôle commande (phase 2) : problématiques

Les étapes de la V&V du système de contrôle commande sont résumées dans le tableau 1, et la procédure est décrite par la figure 14. Cette étape permet aux chargés d'études de s'assurer, avant la mise en service sur site, que le système de contrôle commande a été correctement conçu sans erreurs, et qu'il respecte le cahier des charges. La moindre erreur de V&V pourrait compromettre la validité des tests réalisés, et par conséquent le bon fonctionnement et la sécurité des personnes et des biens, d'où la nécessité d'avoir une méthodologie fiable.

Malheureusement, la méthodologie de V&V utilisée par les chargés d'études de la SNCF présente toujours quelques inconvénients. De plus, suite à l'allègement de la phase 1 du workflow qui se résume aujourd'hui à de la génération automatique, les chargés d'études ont davantage manifesté leur besoin d'améliorer la phase 2 du workflow, qui est réalisée aujourd'hui avec beaucoup de difficultés. Tout d'abord, les programmes automates générés lors de la phase 1 sont vérifiés à travers une simple relecture, ce qui est loin d'être suffisant pour détecter toutes les erreurs que renferment les programmes API. Ensuite, la V&V du système de contrôle commande (programmes API + câblage des armoires électriques) est réalisée manuellement au travers des essais manuels basés sur un cahier

de recettes. Avec un système aussi complexe que l'EALE, ces essais prennent énormément de temps aux chargés d'études. Cette tâche épuisante combinée au stress de deadline du projet expose le chargé d'études à une surcharge mentale qui ne le met pas à l'abri d'erreurs humaines pendant les essais. À cela s'ajoute la complexité du diagnostic des erreurs présentes sur le système de contrôle commande (erreur de câblage des armoires, ou de programmation des automates). L'autre inconvénient majeur de la méthode de V&V actuelle est son manque d'exhaustivité. Depuis plusieurs décennies, le cahier de recettes a toujours été considéré à la SNCF comme un document de référence fiable, permettant de vérifier et de valider tout système de contrôle commande. Malheureusement cette méthode ne permet d'explorer qu'une partie des scénarios possibles du système décrits par le cahier de recettes. Elle n'est donc pas assez exhaustive, et ne garantit pas formellement la sûreté de fonctionnement. Ces inconvénients seront davantage détaillés dans le chapitre 2.

6 Contribution de la thèse

Pour supprimer toutes les contraintes de la phase de V&V (section 5.2.2), les chargés d'études ont aujourd'hui besoin d'une méthodologie de V&V automatisée et formelle, et surtout fiable. L'objectif de ce projet de recherche est justement d'optimiser la phase 2 du workflow en mettant en place une approche méthodologique et originale, qui permet aux chargés d'études de vérifier formellement et automatiquement les systèmes de contrôle commande des EALE. La phase 2 du workflow traditionnel intitulée **« V&V du système de contrôle commande »** va désormais être remplacée par une nouvelle phase intitulée **« Vérification formelle des programmes API et validation automatique du système de contrôle commande »** (figure 20).

Comme le montre son nouvel intitulé, la deuxième phase se réalise en deux étapes :

Étape 1 : Vérification formelle des programmes API. *(phase 2-a figure 21)*
Mise à part une simple relecture, les chargés d'études ne disposent aujourd'hui d'aucune méthode pour vérifier « uniquement » les programmes API générés par ODIL (aspect fonctionnel et sûreté de fonctionnement). Cette première étape permet de vérifier formellement que les programmes automates respectent les spécifications fonctionnelles tout en étant sûrs de fonctionnement. Dans le cas contraire, les erreurs pourront être diagnostiquées et corrigées de deux manières possibles :

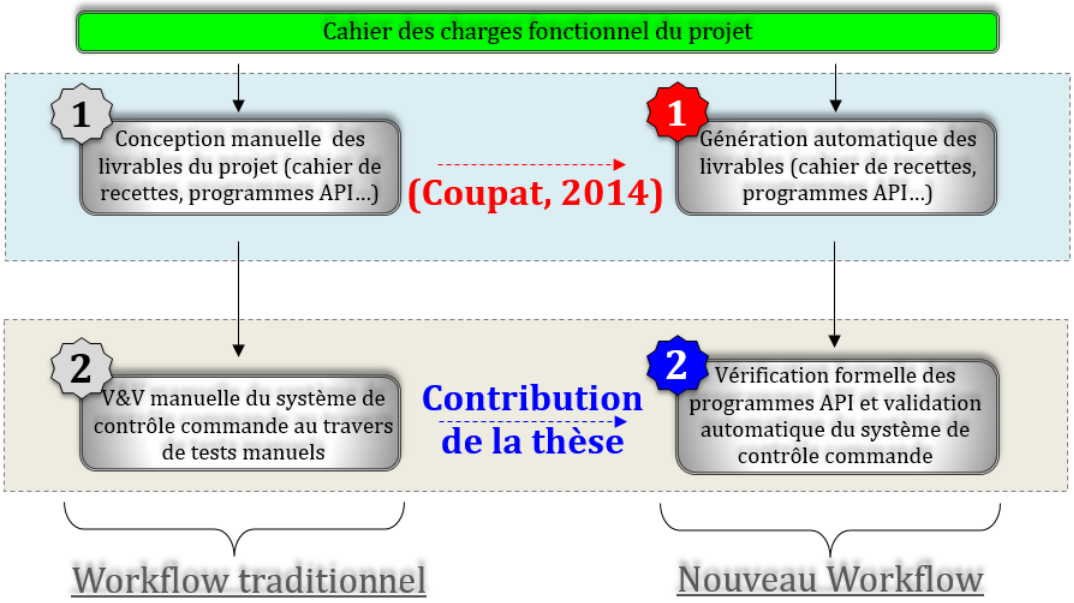


FIGURE 20 – Contribution de la thèse

— soit par correction manuelle, lorsque les spécifications fonctionnelles ne sont pas respectées,

— soit en rajoutant un filtre logique de commandes par contraintes (Pichard et al., 2018) dans les programmes API, de manière à le rendre formellement sûr de fonctionnement.

Cette méthode permet de remédier au manque d'exhaustivité des tests du cahier de recettes. Elle sera détaillée dans le chapitre 3. Après la vérification formelle et la correction des programmes automates, ou plutôt des « modèles » des programmes automates, ces corrections sont ensuite reportées sur les programmes API d'origine (générés par ODIL). L'étape suivante permet de valider ces programmes API (avec les corrections apportées).

Étape 2 : Validation automatique du système de contrôle commande en usine. *(phase 2-b figure 21)*

Une fois que le programme a été formellement vérifié, cette deuxième étape permet de valider automatiquement les programmes automates et le câblage des armoires du système de contrôle commande. La technique du Virtual Commissioning sera utilisée à cet effet en mode **Software-In-the-Loop** (phase ***i***, figure 21) puis en mode **Hardware-In-the-Loop** (phase ***ii***, figure 21). Le premier mode permet aux chargés d'études de valider les programmes automates (depuis leur bureau), et le deuxième sert à valider le câblage des armoires en usine. Le principe de la validation reste le même pour les deux architectures, et consiste à programmer l'exécution automatique des scénarios du cahier de recettes sur

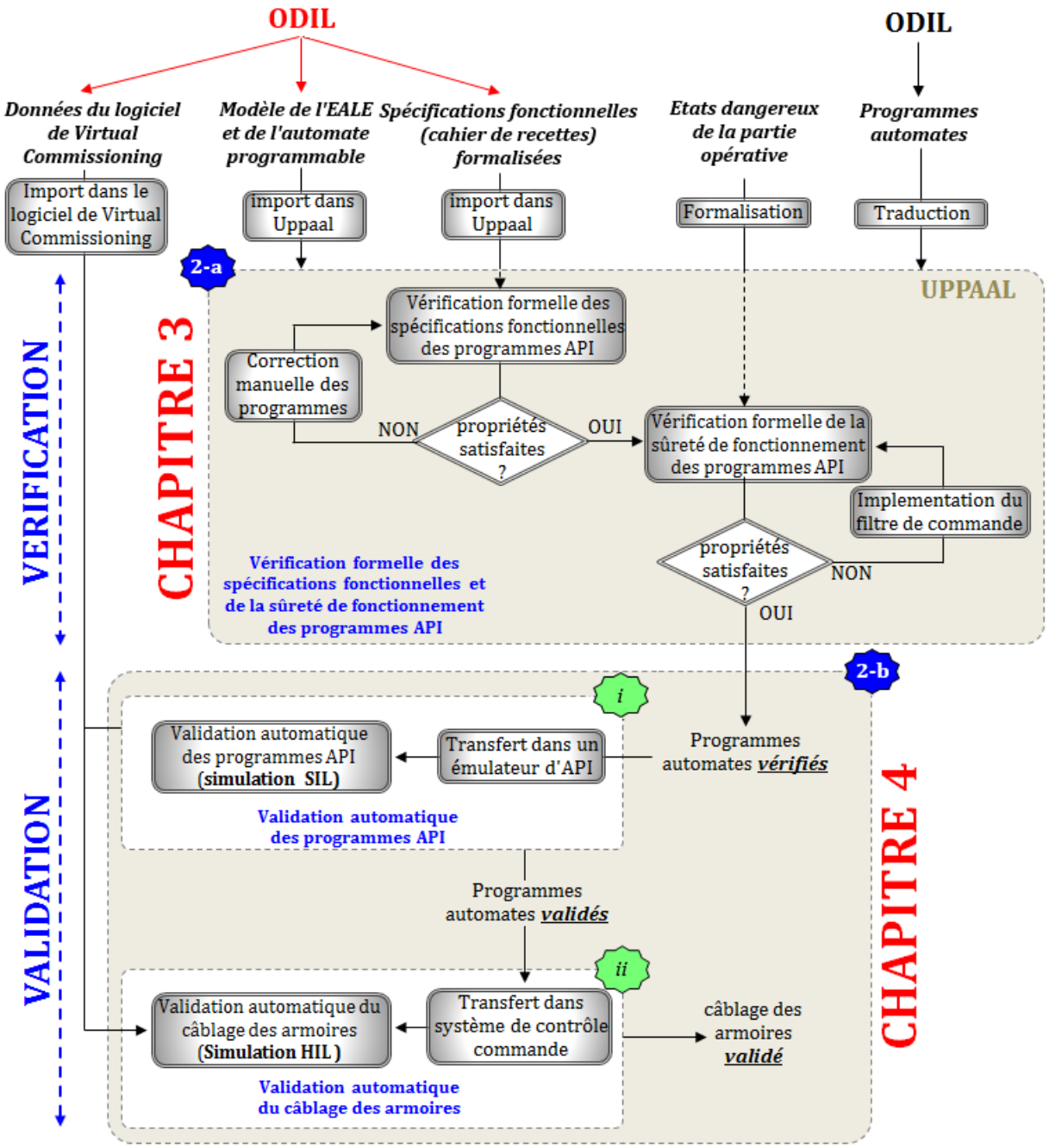


FIGURE 21 – Présentation de la nouvelle méthodologie de V&V

la partie commande connectée à une EALE virtuelle, et à générer un compte rendu à la fin des essais. Ce compte rendu devra mentionner l'ensemble des anomalies que présente le système de contrôle commande (ainsi que leurs causes), afin de faciliter aux chargés d'études la correction des erreurs.

Cette nouvelle méthodologie (figure 21) permet de simplifier le diagnostic et la correction d'éventuelles erreurs, d'abord sur le programme (phase 2-a), et ensuite sur le câblage des armoires de contrôle commande (phase 2-b). Ces deux phases sont respectivement développées dans les chapitres 3 et 4. Pour faciliter l'application de ces deux méthodes, le

logiciel ODIL sera réadapté de sorte qu'il puisse également générer les données d'entrées nécessaires aux deux méthodes (Modèles de simulation pour la vérification formelle des programmes, et modèles de simulation pour le Virtual Commissioning). Cela évitera des pertes de temps supplémentaires dues à la saisie manuelle des données.
Les avantages de cette nouvelle méthodologie de V&V sont les suivantes :

- un gain de temps lors de la V&V étant donné que celle-ci est automatisée,
- une vérification plus exhaustive et donc plus fiable grâce à la vérification formelle,
- une augmentation de la robustesse du système face aux erreurs de commandes (grâce au filtre logique de commande par contraintes),
- une augmentation du rendement et une réduction des pannes qui minimisera les pertes économiques.

7 Conclusion

Dans ce chapitre, nous avons présenté la structure des Équipements d'Alimentation des Lignes Électrifiées ainsi que l'architecture des systèmes de contrôle commande. Le workflow traditionnel permettant aux chargés d'études de concevoir ces systèmes a également été présenté. Suite au projet de recherche de Coupat (2014) visant à optimiser la phase de conception manuelle des livrables d'un projet, un nouveau workflow a été établi pour les chargés d'études. L'outil ODIL a été développé pour générer automatiquement les livrables (programmes API et cahier de recettes) à partir d'une description graphique du projet sur l'interface du logiciel. Cette nouvelle solution a permis d'alléger la charge de travail des ingénieurs lors de la phase 1, mais également de réduire les erreurs humaines lors de la conception ainsi que le temps requis pour cette étape. Cependant la phase 2 du workflow présente aujourd'hui quelques inconvénients (section 5.2.2) que les chargés d'études souhaitent éradiquer. Pour répondre à leurs attentes, ce projet de recherche a pour objectif de développer une approche méthodologique de V&V formelle et automatisée, qui sera utilisée par les chargés d'études à la place de l'ancienne méthode. Cette contribution fait l'objet des chapitres 3 et 4. Le chapitre 2 présente un état de l'art sur les méthodes de Vérification et Validation des systèmes de contrôle commande, ainsi que les verrous scientifiques associés.

Chapitre II

Vérification et Validation des systèmes de contrôle commande : état de l'art, verrous scientifiques, et proposition d'une méthodologie pour la SNCF

1 Introduction

La vérification et la validation des systèmes de contrôle commande sont des étapes complémentaires et incontournables avant la mise en service d'un système automatisé. Elles permettent aux ingénieurs de s'assurer que les spécifications décrites dans le cahier des charges ont été respectées. Lorsqu'elles ne sont pas bien exécutées, cela peut avoir de lourdes conséquences sur le système et compromettre la sécurité des personnes et des biens, d'où l'importance d'avoir une méthodologie. À cette fin, il existe plusieurs techniques de vérification et de validation des systèmes de contrôle commande, mais elles différent de par leur type, leur efficacité, leur fiabilité et leur rapidité d'exécution.

Ce chapitre a pour but de présenter dans un premier temps les différentes techniques utilisées pour la vérification et la validation des systèmes de contrôle commande, en mettant en évidence les avantages et les manquements de chacune d'elles. Puis, après avoir présenté l'approche de V&V utilisée par les chargés d'études de la SNCF, nous proposons

une nouvelle approche méthodologique de V&V automatique de manière à augmenter la fiabilité et la robustesse des systèmes de contrôle commande des EALE.

2 État de l'art des méthodes de Vérification et Validation existantes

2.1 Généralités

Avant de faire le point sur les méthodes de V&V existant dans la littérature, il est important de bien différencier ces deux notions. Selon la norme ISO-9000 (2015), la vérification est la confirmation par examen et apport de preuves tangibles que les exigences spécifiées ont été satisfaites. Elle répond à la question « Construisons-nous correctement le produit ? ». La validation est la confirmation par examen et apport de preuves tangibles que les exigences pour une utilisation spécifique ou une application prévue ont été satisfaites. Elle répond à la question « Avons-nous construit le bon produit ?».
Ces deux étapes en plus d'être complémentaires, sont incontournables durant un projet d'automatisation. Ce processus de vérification et de validation (V&V) permet de s'assurer que non seulement le cahier des charges fonctionnel a été respecté, mais aussi que la sûreté de fonctionnement sera garantie après la mise en service. Les méthodes de vérification et/ou validation des systèmes de contrôle commande peuvent être regroupées en trois catégories : les tests, le Virtual Commissioning, et les méthodes formelles.

2.2 Les tests

La méthode des tests contribue depuis longtemps à la vérification et la validation des systèmes de contrôle commande. C'est d'ailleurs l'approche la plus utilisée dans le monde industriel (Bjørner and Havelund, 2014). Selon le standard (ANSI, 1983), les tests sont définis comme un processus d'exercices ou d'évaluations d'un système ou d'un composant du système par des méthodes manuelles ou automatisées, pour vérifier qu'il satisfait aux exigences spécifiées ou pour identifier les différences entre les résultats attendus et réels. Une autre définition apparaît dans (Hetzel, 1984), et considère les tests comme toutes activités visant à évaluer un attribut ou une capacité d'un programme ou d'un système et à déterminer s'il répond aux résultats requis. Et enfin selon (Myers, 1979), les tests représentent un processus d'exécution d'un programme dans le but de trouver des erreurs.

En résumé, les tests sont basés sur l'exécution de scénarios de tests prédéfinis ou aléatoires, et permettent de vérifier que le système répond aux besoins du cahier des charges. Le schéma de principe du test est décrit par la figure 22.

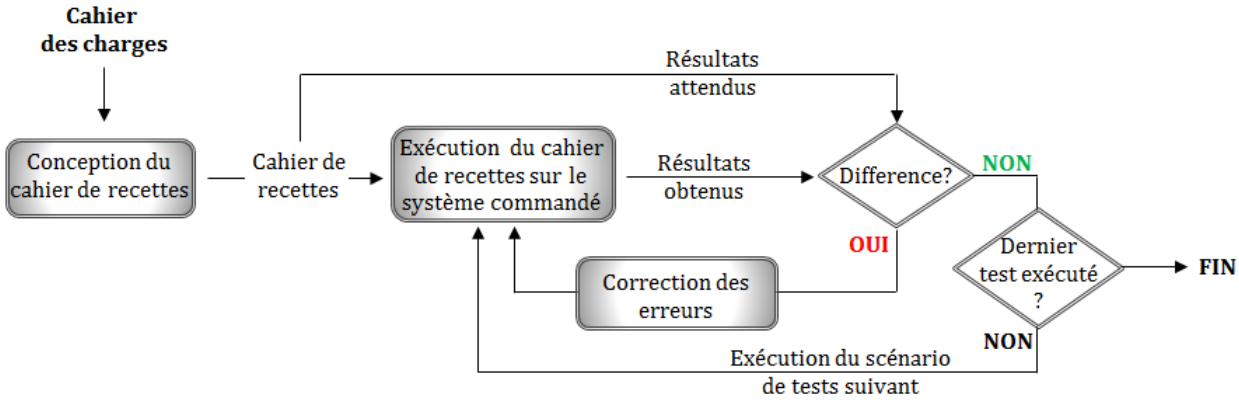


FIGURE 22 – Principe du déroulement du test

Les tests sont en général réalisés grâce à un document contenant l'ensemble des scénarios à exécuter sur le système de contrôle commande ainsi que les résultats attendus après chaque essai. Ce document nommé cahier de recettes (voir structure dans figure 23) est défini en même temps que les spécifications et sert de référence au testeur. Chaque scénario de tests est constitué de 3 champs :

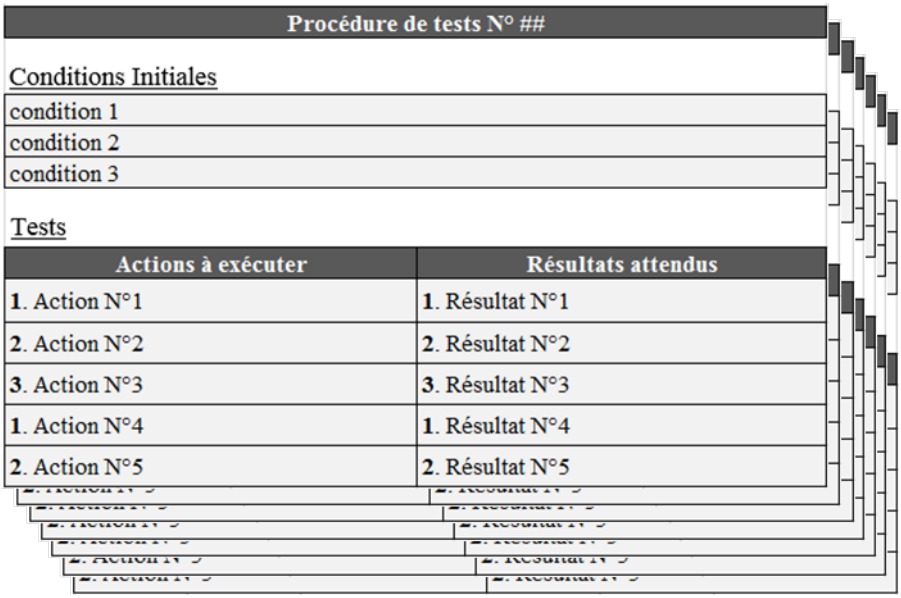
— Les conditions initiales : qui déterminent l'état dans lequel le système doit nécessairement demeurer avant le début des essais, sans quoi les résultats des tests seraient erronés ;

— Les actions à réaliser ;

— Les résultats attendus.

Un cahier de recettes regroupe l'ensemble des scénarios de tests (ou procédures de tests) nécessaires pour vérifier et valider le système de contrôle commande. Le principe du test avec cahier de recettes consiste à exécuter un à un les différents scénarios de tests en partant d'un état initial donné, et à vérifier que le comportement du système (i.e. résultat obtenu) à la suite de ces actions correspond bien au résultat attendu figurant dans le cahier de recettes (Chériaux et al., 2010). L'opérateur ne pourra valider le système de contrôle commande que lorsque tous les scénarios du cahier de recettes ont été exécutés avec succès.

Les tests peuvent être classés en fonction de leur mode d'exécution (manuel ou automatique, voir section 2.2.1 ci-dessous) et hiérarchisés en fonction de leur périmètre d'application (section 2.2.2).



Procédure de tests N° ##

Conditions Initiales

condition 1
condition 2
condition 3

Tests

Actions à exécuter	Résultats attendus
1. Action N°1	**1**. Résultat N°1
2. Action N°2	**2**. Résultat N°2
3. Action N°3	**3**. Résultat N°3
1. Action N°4	**1**. Résultat N°4
2. Action N°5	**2**. Résultat N°5

FIGURE 23 – Structure d'un cahier de recettes

2.2.1 Classification des tests en fonction de leur mode d'exécution

D'une manière générale, les tests peuvent être exécutés en mode manuel ou en mode automatique (Leitner et al., 2007). Dans le premier cas, les tests sont effectués pas à pas par l'opérateur sans aucun support d'outil ou de script, contrairement au second cas où les tests sont automatisés à l'aide d'outils, de scripts, ou de logiciels spéciaux. Ces deux modes présentent chacun des avantages et des inconvénients. De ce fait, le choix du mode de test pour un projet dépend des facteurs essentiels tels que le temps alloué au projet, le coût, et la qualité.

	Avantages	Inconvénients
LES TESTS MANUELS	○ Sont mieux adaptés si les essais nécessitent à la fois l'expérience, la capacité d'analyse, la créativité et l'intuition du testeur ○ Sont mieux adaptés s'il y a peu d'essais à exécuter ou si les tests ne sont pas répétitifs ○ Permettent à l'opérateur de mieux suivre en temps réel l'évolution des essais	○ Ne sont pas totalement fiables en raison des possibilités d'erreurs humaines ○ Peuvent prendre beaucoup de temps, surtout pour des systèmes complexes
LES TESTS AUTOMA-TIQUES	○ Sont plus fiables étant donné qu'ils sont automatisés ○ Peuvent effectuer un grand nombre de tests en peu de temps ○ Sont mieux adaptés lorsque les scénarios de tests sont nombreux ou répétitifs, et peuvent être automatisés	○ Ne sont pas adaptés si les tests ne peuvent pas être automatisés ○ Ne sont pas directement exécutés par l'opérateur, celui-ci n'aura donc pas un aussi bon suivi qu'avec les tests manuels

TABLEAU 3 – Tests manuels et automatiques : avantages et inconvénients

Le tableau 3 (inspiré de Admin (2014), Leitner et al. (2007), et Fernandez et al. (2014))

résume les avantages et les inconvénients de chacun des modes d'exécution des tests.

Il est important de noter que ces deux modes de tests ne s'excluent pas forcément et peuvent être utilisés ensemble dans certains cas, de manière à combiner leurs avantages respectifs (Leitner et al., 2007).

2.2.2 Hiérarchisation des tests

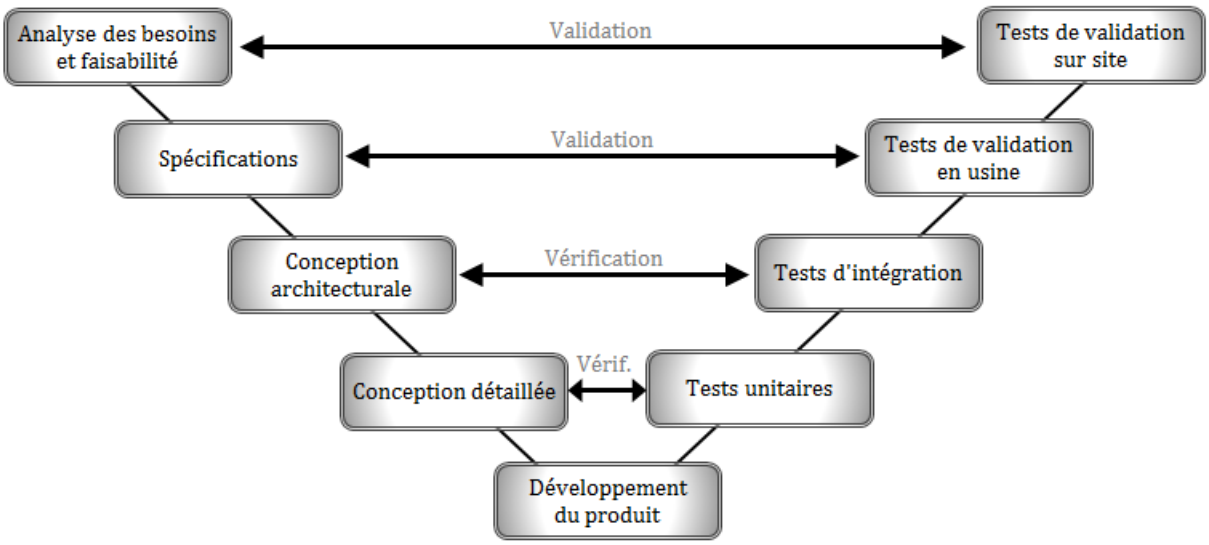


FIGURE 24 – Cycle en V

Comme le montre la deuxième partie du **cycle en V** (partie droite de la figure 24), les tests peuvent s'appliquer de manière structurée sur un produit, c'est-à-dire qu'ils peuvent s'appliquer progressivement sur plusieurs sous-parties avant le test de validation final. Cela permet de faciliter le diagnostic d'éventuelles erreurs sur le produit fini. Selon ce critère, ils peuvent être classés dans l'ordre en 4 catégories (Forsberg and Mooz, 1991) :

— Test unitaire : consiste à tester séparément chaque bloc du système (fonction, programme ...) pour s'assurer qu'il est conforme aux spécifications.

— Test d'intégration : une fois que les blocs ont été séparément testés, cette étape consiste à regrouper plusieurs blocs du système pour vérifier qu'ils échangent bien entre eux et que l'ensemble respecte les spécifications. Cette approche est plus complète que celle du test unitaire car son périmètre d'application est plus vaste.

— Test de validation en usine : la conception du système étant achevée, cette étape consiste à le tester intégralement (avec des procédures de tests, établies en même temps que le cahier des charges) pour s'assurer qu'il est conforme aux spécifications. Selon la norme ISA-62381 (2012), cette étape correspond au « Factory Acceptance

Test - FAT » qui a lieu en usine et idéalement en présence du client qui pourra approuver les résultats des essais avant l'intégration du système sur site.

— Test de validation sur site : le système est vérifié une dernière fois en pré-production sur site, avant d'être mis en production. Toujours selon la norme ISA-62381 (2012), cette étape correspond au « Site Acceptance Test - SAT ». Les procédures de tests (ou la recette) sont exécutées par l'intégrateur en présence du client, pour vérifier que l'expression du besoin a été respectée.

Certaines étapes telles que les tests unitaires et les tests d'intégration ne sont pas obligatoires, mais peuvent être très utiles pour des systèmes complexes ou distribués. Cette démarche méthodologique permet en cas d'anomalies de limiter un retour aux étapes précédentes.

Bien que le test soit la méthode la plus utilisée aujourd'hui en industrie, il nécessite la présence du système réel (machine réelle + système de contrôle commande) car il s'effectue sur le produit fini. La partie commande ne peut donc être définitivement vérifiée et validée qu'en présence de la machine réelle (« Real Commissioning ») et sans aucun modèle de simulation intermédiaire. Cette technique a certes l'avantage d'être fidèle à la réalité du terrain, mais elle peut s'avérer très longue, onéreuse, voire dangereuse. Elle diffère en cela de la méthode du « Virtual Commissioning » (mise en service virtuelle), qui permet de vérifier le système de contrôle commande grâce à un simulateur de partie opérative (PO).

2.3 Le Virtual Commissioning

A la différence du « Real commissioning » qui nécessite la présence de la machine réelle et du système de contrôle commande réel, le principe du « Virtual Commissioning » (Drath et al., 2008) se base sur la connexion entre le système de contrôle commande et un simulateur de partie opérative (modèle virtuel de simulation de la machine cible). Comme son nom l'indique, un simulateur de PO est un modèle informatique chargé de reproduire le comportement du procédé physique à commander (machine, convoyeur ...). Les premiers simulateurs de PO ne datent pas d'aujourd'hui. En effet, les toutes premières méthodes de simulation de PO sont basées sur l'utilisation de boites à boutons câblées sur les entrées/sorties du système de contrôle commande cible pour exciter les entrées et émuler les sorties. Depuis, un pas décisif a été franchi avec l'apparition des premiers logiciels simulateurs de PO implantés dans une machine cible (un Automate Programmable industriel ou un ordinateur).

Basé sur la connexion entre la commande réelle et un simulateur de PO, le Virtual Commissioning permet la vérification du système de contrôle commande au travers d'une simulation. Ce principe est également connu sous le nom de **Simulation Hardware-In-the-Loop (Simulation HIL)** qui signifie littéralement « simulation avec matériel dans la boucle ».

Cette méthode de simulation HIL est aujourd'hui utilisée pour les essais et la mise au point des applications critiques de commande, de protection et de surveillance. Sa montée en puissance est la résultante de deux grands facteurs influant sur le cycle de développement produit de tous les secteurs industriels : le délai de mise sur le marché et la complexité des systèmes mis en œuvre. Traditionnellement, les essais en contrôle commande industriel se font directement sur l'équipement physique (une chaîne de production par exemple), sur l'intégralité du système ou sur un banc de laboratoire. Cette pratique a certes l'avantage d'être fidèle à la réalité du terrain, mais elle peut s'avérer très longue, onéreuse, voire dangereuse.

La simulation HIL (figure 25) remédie parfaitement à ces inconvénients. Dans ce cas, l'installation physique à l'essai est remplacée par un modèle informatique, fidèle réplique de l'existant, exécuté en temps réel sur un simulateur équipé d'entrées/sorties et communiquant avec le système de contrôle commande par l'intermédiaire d'une interface physique. Ce simulateur peut ainsi reproduire avec précision le système piloté et sa dynamique, ainsi que son instrumentation (capteurs/actionneurs), pour tester leurs interactions en boucle fermée, sans passer par un système réel.

Depuis plusieurs décennies, le Virtual Commissioning a été appliquée dans certains domaines du transport tels que l'aérospatiale et l'automobile. Cette technique s'applique également dans le domaine du transport ferroviaire (Baccari et al., 2012), y compris sur les systèmes de contrôle commande (Di Tommaso et al., 2005). D'autres exemples d'application du Virtual Commissioning en industrie sont présentés dans (Kleijn, 2014).

Le Virtual Commissioning peut également se présenter sous une autre forme, reliant un procédé simulé à un système de contrôle commande également simulé (Lee and Park, 2014). Cette configuration utilisée également pour la vérification de la partie commande est connue sous le nom de **Simulation Software-In-the-Loop (Simulation SIL)** qui signifie littéralement « Simulation avec logiciel dans la boucle ». Elle s'utilise en général en amont et en complément de la simulation HIL. Il s'agit d'un mode 100% virtuel qui permet de vérifier uniquement la partie logicielle du système de contrôle commande (i.e. le

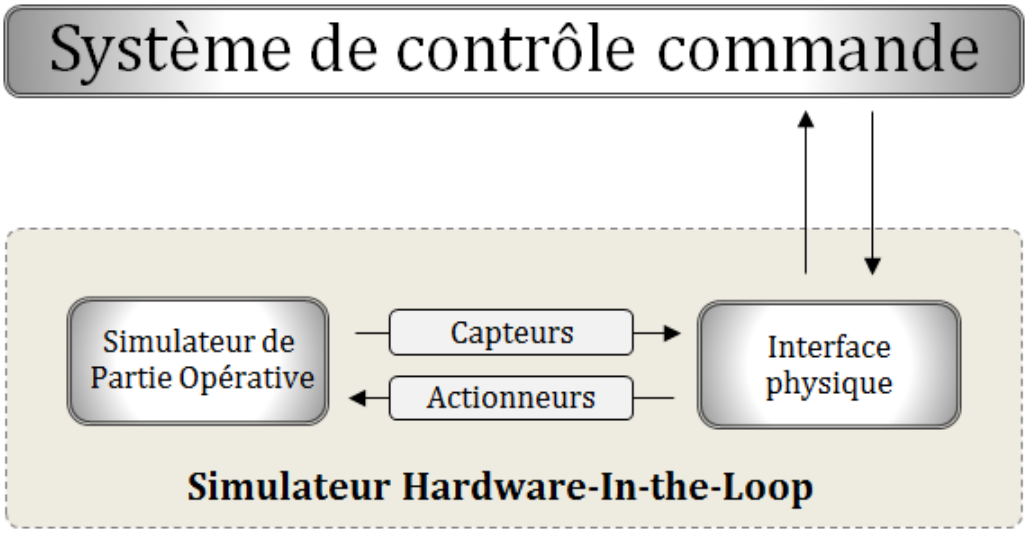


FIGURE 25 – Architecture de la Simulation Hardware-In-the-Loop

programme API). Pour ce faire, l'exécution du programme est simulé dans un émulateur d'API qui échange directement avec le simulateur de partie opérative (figure 26).

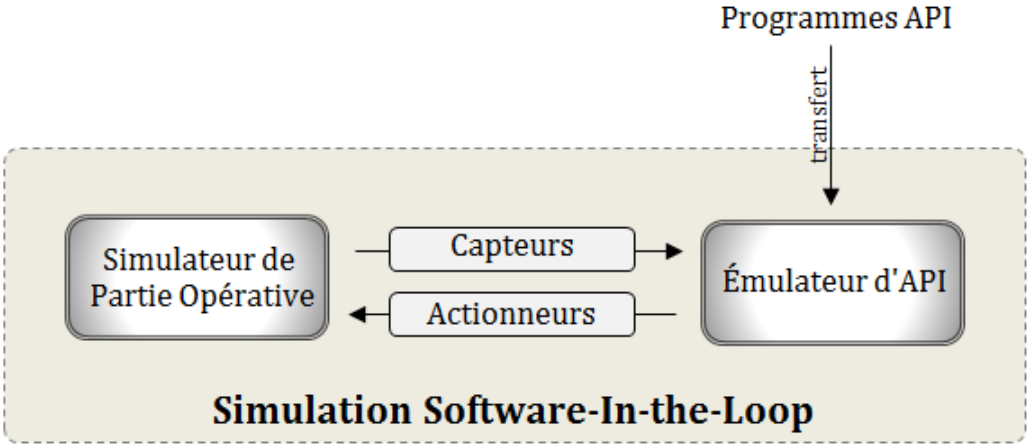


FIGURE 26 – Architecture de la Simulation Software-In-the-Loop

Ces deux méthodes de simulation sont complémentaires et peuvent vérifier intégralement un système de contrôle commande : la première (simulation SIL) vérifie uniquement la partie logiciel de la commande (i.e. le programme API) et la deuxième (simulation HIL) vérifie l'ensemble, y compris la partie matérielle du système de contrôle commande.

Au-delà de ces fonctionnalités, ces techniques de simulation HIL et SIL présentent beaucoup d'avantages, comparées aux méthodes traditionnelles.

2.3.1 L'augmentation de la sécurité

Pour des machines potentiellement dangereuses, l'utilisation de la simulation HIL garantit la sécurité des personnes car l'opérateur travaille sur un système simulé sur lequel

il peut tester plusieurs scénarios sans risque, même si le programme API implémenté présente des erreurs. Elle assure également la sécurité de l'installation, car elle permet de détecter à l'avance des scénarios qui pourraient potentiellement endommager ou détruire le système réel.

2.3.2 L'amélioration de la qualité

De nombreuses études ont montré que la détection et la correction précoces des erreurs sur un système entraînent une augmentation de sa qualité (Kleijn, 2014). Grâce au Virtual Commissioning (simulations HIL et SIL), la vérification de la partie commande peut être effectuée dès les premières phases de sa conception. L'opérateur peut recourir à des essais progressifs sur la partie commande connectée à la PO simulée, ce qui lui permet de détecter et de corriger d'éventuelles erreurs bien avant les phases de développement et d'intégration du matériel afin d'accélérer le cycle de développement. Koo et al. (2011) ont d'ailleurs montré que le Virtual Commissioning réduisait considérablement le taux d'erreurs détectées lors des tests de validation sur site. De plus, avec cette technique, les tests peuvent être automatisés de manière à accélérer les essais et diminuer les risques d'erreurs humaines survenant lors des tests manuels.

2.3.3 Gain de temps sur le projet

Généralement la conception de l'installation réelle et celle du système de contrôle commande se font en parallèle. Par conséquent, certaines erreurs de conception ne peuvent être détectées que pendant les tests de validation sur site en présence de l'installation réelle. Durant cette étape, les ingénieurs passent beaucoup de temps sur la correction des erreurs du système de contrôle commande. Grâce au Virtual Commissioning, les erreurs peuvent être détectées et corrigées bien assez tôt, ce qui réduit considérablement le temps nécessaire pour les tests de validation sur site. Koo et al. (2011) ont confirmé ce résultat à travers leur étude qui a montré que la durée des tests de validation sur site (avant la mise en service) est réduite de 75% grâce au Virtual Commissioning.

2.3.4 Diminution du coût

Le coût relatif aux erreurs augmente considérablement au fil du temps (Boehm, 1979), et les tests de validation sur l'installation réelle peuvent être très coûteux. Cela peut être dû à d'éventuelles pertes matérielles en cas d'erreurs de conception de la commande, des

précautions de sécurité coûteuses et indispensables pour les testeurs lors des essais, ou tout simplement les coûts de fonctionnement. Le Virtual Commissioning remédie parfaitement à ces inconvénients. De plus, il permet de gagner plus de temps lors de la phase de validation sur site, ce qui réduit les coûts de validation et de mise en service.

2.3.5 Formation des futurs opérateurs

Le Virtual Commissioning est très bénéfique pour la formation des futurs opérateurs d'une machine afin de les y préparer. Il leur permet :

- de mieux comprendre le fonctionnement de la machine en mode nominal ou dégradé sans être exposés aux dangers du système réel (avec une simulation 3D par exemple),
- de tester librement des scénarios sans risques,
- d'apprendre à résoudre des problèmes sur le système réel qu'ils auront déjà rencontrés et résolus en mode simulation,
- etc.

Par exemple, des logiciels comme FACTORY I/O (usine virtuelle) et HOME I/O (maison virtuelle) (Riera et al., 2018), développés dans le cadre d'un partenariat scientifique et technique entre le laboratoire **CReSTIC** et la société portugaise **Real Games** (*www.realgames.co*) ont été développés pour la formation à la technologie de l'automatisation et aux sciences de l'ingénieur.

Hormis son manque d'exhaustivité, le Virtual Commissioning présente toutefois deux inconvénients majeurs :

- La conception des modèles de simulation nécessite beaucoup de temps et d'effort pour les ingénieurs qui n'en ont pas forcément les compétences ;
- Le système simulé peut ne pas correspondre au système réel pour des raisons de manque de granularité, de manque de réalisme, ou de présence d'erreurs sur les modèles.

Les tests et la simulation permettent aujourd'hui de vérifier que le fonctionnement d'un système est conforme aux attentes du cahier des charges. Cependant, ces techniques basées sur l'exécution des procédures ou scénarios de tests sur la PO réelle ou simulée ne permettent d'explorer qu'une partie des comportements possibles du système. Cela peut servir à montrer que les spécifications fonctionnelles de la commande ont été respectées, mais reste insuffisant pour garantir formellement la sûreté de fonctionnement du système.

Ces techniques sont donc loin d'être exhaustives. Elles diffèrent en cela des méthodes formelles qui garantissent qu'une propriété est formellement vérifiée par la totalité des exécutions possibles du système.

2.4 Les méthodes formelles

Basées sur des notations mathématiques, les méthodes formelles (Clarke and Wing, 1996) offrent un moyen robuste pour la détection et la correction des erreurs de conception. Elles permettent de vérifier formellement que des propriétés écrites au préalable en langages formels sont vérifiées par un système (décrit également en langages formels) (Bjørner and Havelund, 2014). Leur caractère mathématique les rend plus précises et exhaustives comparées au test et à la simulation.

Au cours des deux dernières décennies, les recherches sur les méthodes formelles ont conduit au développement de certaines techniques de vérification très prometteuses facilitant la détection précoce des défauts d'un système. Ces techniques sont accompagnées de puissants outils logiciels pouvant être utilisés pour automatiser diverses étapes du processus de vérification. Des études ont montré que ces méthodes auraient permis de diagnostiquer des défauts dans des systèmes tels que la fusée **Ariane-5**, la sonde spatiale **Mars Pathfinder**, le processeur **Pentium II** d'Intel et la machine de radiothérapie **Therac-25** (Baier and Katoen, 2008). Ces études ont permis de démontrer leur efficacité dans la détection des erreurs et la garantie de la qualité de ces systèmes. Dans ce sens, plusieurs standard (à l'instar de la norme CENELEC (2012) dans le domaine du ferroviaire) recommandent fortement l'application de ces techniques dans le cadre du développement et de la vérification des systèmes complexes exigeant un haut niveau de sécurité.

L'application des méthodes formelles pour la V&V des systèmes de contrôle commande dans le domaine du ferroviaire constitue un sujet de recherche très actif depuis quelques années (Vu, 2015). Un aperçu général des travaux menés à ce sujet est présenté dans (Fantechi et al., 2012), (Ferrari et al., 2013), (Fantechi, 2014), et (Fantechi et al., 2014).

Les méthodes formelles les plus utilisées sont le Model-Checking, le Theorem-Proving, et l'analyse d'atteignabilité.

2.4.1 Le Model-Checking (vérification de modèles)

Le Model-Checking (Clarke et al., 1999) (Baier and Katoen, 2008) est une méthode formelle qui consiste à construire un modèle fini d'un système et vérifier automatiquement

que celui-ci satisfait à un ensemble de propriétés ou spécifications. Le système est généralement modélisé en réseau de Petri (Peterson, 1981) (Reisig, 2013) ou en automates finis ou temporisés (Moor et al., 1998), et les propriétés sont exprimées en logique temporelle LTL, CTL, TCTL... (Thistle and Wonham, 1986) ou en automates. Il existe globalement deux formes de Model-Checking :

— Le Model-Checking temporel développé par Clarke and Emerson (1982) et Queille and Sifakis (1982) : le système est modélisé sous forme de système de transition à état fini et les propriétés sont exprimées en logique temporelle. Une procédure de recherche exhaustive permet ensuite de vérifier si les propriétés sont satisfaites.

— L'autre forme du Model-Checking consiste à modéliser le système et les spécifications en automates, afin de les comparer entre eux pour vérifier si le comportement du système est conforme aux attentes des spécifications.

Lors de la vérification, le Model-Checker (logiciel de Model-Checking) explore de manière exhaustive la totalité des exécutions possibles du système. De cette manière, il vérifie pour chaque état si les propriétés sont satisfaites, et retourne dans ce cas la trace (ou succession d'événements) qui permet d'arriver à cet état (figure 27).

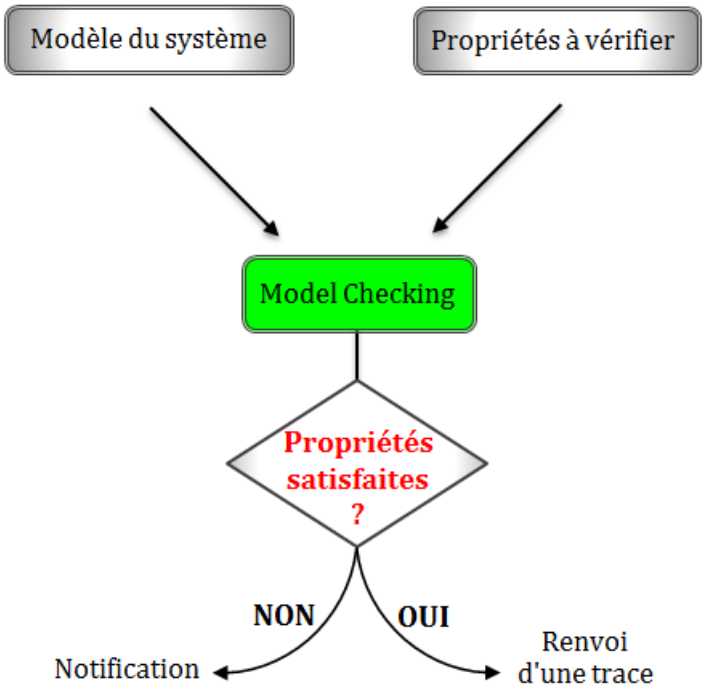


FIGURE 27 – Principe du Model Checking

Cependant pour des systèmes de taille importante, cette exploration exhaustive d'états peut dépasser la capacité mémoire des ordinateurs avec comme conséquence l'interruption de la vérification. Ce problème est connu sous le nom d'« **explosion com-**

binatoire » ou « **explosion de l'espace d'états** » et reste à ce jour un vrai obstacle pour l'adoption de cette technique en industrie. Des techniques ont été développées à ce sujet (abstraction, symbolic model checking, partial order reduction, SMT solvers, Binary Decision Diagrams BDD...) pour minimiser voire éviter ce problème d'explosion combinatoire. Ces travaux sont présentés dans Xin-feng et al. (2009). Le Model-Checking est également sujet à des erreurs humaines lors de la modélisation et la formalisation des propriétés à vérifier. Il reste malgré tout l'une des méthodes formelles les plus utilisées et les plus efficaces (Bérard and Schnoebelen, 1999).

2.4.2 Le Theorem-Proving (preuve de théorèmes)

Dans le cas du Theorem-Proving (Duffy, 1991) (Fitting, 1996), le système et les propriétés à vérifier sont exprimés sous formes de formules dans une logique mathématique. Cette logique est décrite par un système formel définissant un ensemble d'axiomes et de règles d'inférence (Lampérière-Couffin et al., 1999). Le Theorem-Proving est un processus de recherche de la preuve d'une propriété à partir des axiomes du système. Contrairement au Model-Checking, l'explosion combinatoire n'existe pas avec le Theorem-Proving car il peut s'utiliser avec des espaces d'états infinis. Son principal inconvénient est que la vérification n'est pas automatique, car elle nécessite souvent l'intervention humaine pour la manipulation des formules ((Clarke and Wing, 1996)). De plus elle est sujette à l'erreur, et demande beaucoup d'expertise.

2.4.3 Reachability Analysis (analyse d'atteignabilité)

L'analyse d'atteignabilité est une technique qui consiste à construire l'espace d'états complet du système modélisé, à la suite de quoi les propriétés sont vérifiées en examinant la structure et les composants de cet espace d'états. Toutefois, le problème majeur avec cette technique est l'explosion combinatoire avec les Systèmes à Événements Discrets. Le nombre d'états du système augmente de manière exponentielle en fonction du nombre de variables discrètes du système. Dans la littérature, cette technique est largement moins utilisée que le Model-Checking et le Theorem-Proving.

Les méthodes formelles ont chacune des avantages et des inconvénients. Contrairement au Theorem-Proving, le Model-Checking est automatique et peut donner des résultats très rapidement. De plus, un de ses points forts est la génération de contre-exemples en cas de

violation d'une contrainte, ce qui peut être très utile pour diagnostiquer et corriger des erreurs. Une étude menée par Kesraoui (2017) a d'ailleurs montré que parmi les méthodes formelles, le Model-Checking est presque cinq fois plus utilisé que les autres méthodes formelles en industrie, et quatre fois plus dans le milieu académique, ce qui confirme qu'il présente plus d'avantages d'une manière générale.

Des études ont permis de démontrer l'efficacité des méthodes formelles dans la détection des erreurs et la garantie de la qualité de ces systèmes. Cependant leur utilisation en milieu industriel est très rare, contrairement au domaine académique. D'autres études menées par Kesraoui (2017) confirment d'ailleurs ce résultat : seulement en moyenne 27% d'utilisation de la vérification formelle en industrie, contre 73% dans le domaine académique. Cette faible utilisation en milieu industriel peut s'expliquer pour diverses raisons : les langages formels devant être mis en œuvre (pour la modélisation du système et la formalisation des propriétés à vérifier) sont délicats à maîtriser, et les ingénieurs ne sont généralement pas familiers avec ces techniques. Les résultats fournis en cas de preuve négative sont souvent difficiles à interpréter, et par-dessus tout, ces méthodes d'analyse exhaustive sont souvent sujettes à des problèmes d'explosion combinatoire, même pour des systèmes de taille relativement modeste (Marangé, 2008). Ainsi pour faciliter l'intégration de ces techniques formelles en industrie, la méthode de vérification formelle doit pallier ces inconvénients en répondant aux critères suivants :

— langage formel accessible, et non complexe,
— facilité d'utilisation (lors de la vérification),
— génération automatique des modèles de vérification (pour éviter les erreurs de modélisation entraînant des pertes d'informations sur le système),
— absence d'explosion combinatoire.

Plusieurs travaux ont été réalisés dans le cadre de la vérification formelle des programmes API. Les approches existantes dans la littérature diffèrent au niveau du formalisme utilisé pour la modélisation et la spécification des propriétés, mais également au niveau de la technique de vérification formelle utilisée. La section suivante présente quelques travaux qui ont été menés à ce sujet.

2.4.4 État de l'art sur la vérification formelle des programmes API

La vérification formelle des programmes API n'est pas une pratique usuelle en industrie. Les concepteurs des programmes de commande les développent directement en

langages API à partir des exigences décrites dans le cahier des charges en se basant sur leurs expériences et leur savoir-faire. Leurs compétences ne leur permettent pas de maîtriser les langages formels, encore moins de formaliser les programmes API et la partie opérative, ce qui est nécessaire pour la vérification formelle. Dans la littérature, les travaux sur la vérification formelle des programmes API ont été effectués pour chacun des langages automates (SFC, LD, ST, FBD, IL). Ces travaux peuvent être classés en fonction des 3 catégories suivantes (Frey and Litz, 2000b) :

La méthode formelle utilisée

- Model-Checking ;
- Theorem-Proving ;
- Analyse d'atteignabilité ;

Le formalisme

- Les réseaux de Petri (Reisig, 2013) ;
- Les automates à états finis (Chow, 1978) ;
- Les automates temporisés (Alur and Dill, 1994) ;
- Les automates hybrides (Raskin, 2005) ;
- Les équations logiques ;
- Les systèmes Condition/Événement (Sreenivas and Krogh, 1991).

L'approche de vérification

- **Model-based** : en plus du programme et des spécifications, la partie opérative est modélisée et incluse dans la vérification. De ce fait, le programme API est vérifié en présence de la partie opérative, ce qui réduit l'espace d'états des évolutions du système durant la vérification. En effet, la présence de la partie opérative dans le modèle global restreint les évolutions possibles de celui-ci en termes de combinaisons possibles d'entrées sorties. Néanmoins elle rajoute de la complexité sur la modélisation.
- **Non-model-based** : A l'opposé de la précédente approche, le programme API est vérifié sans présence de la partie opérative. Toutes les combinaisons d'entrées sorties sont envisageables lors de la vérification. L'espace d'états augmente et expose la vérification à l'explosion combinatoire. Toutefois, la modélisation est moins

complexe que pour la première approche, car la partie opérative n'est pas incluse dans le modèle global.

— **Constrained-based** : La partie opérative n'est pas entièrement modélisée, mais des contraintes structurelles sont rajoutées au modèle global sous forme d'équations booléennes. Ces contraintes structurelles sont définies par le comportement et la structure de la partie opérative (exemple : deux entrées binaires toujours disjointes), et permettent de réduire l'espace d'états pendant la vérification.

Les propriétés les plus souvent vérifiées étant, d'après une étude et dans l'ordre, la sûreté, la vivacité, l'absence de blocage dans le programme et l'atteignabilité (Kesraoui, 2017).

Quelque soit la méthode de vérification formelle utilisée, les programmes API (et éventuellement la partie opérative dans le cas de l'approche model-based) sont généralement modélisés en réseau de Petri (Loborg and Törne, 1996) (Jiang and Holding, 1996), en automates (Système Condition/Événement temporisé Kowalewski and Preußig (1996), automate hybride (Hassapis et al., 1998) ...). Les spécifications sont formalisées généralement en logique temporelle ou en automate. Enfin la méthode de vérification formelle est appliquée pour vérifier que les spécifications ont été respectées par le programme. Citons quelques exemples de travaux :

Vérification formelle par Model-Checking.

Dans ce cadre, les auteurs Weng and Litz (2000) utilisent le réseau de Petri pour la formalisation du programme et de la partie opérative (approche model-based), et la logique temporelle LTL pour la formalisation des propriétés à vérifier. Hassapis et al. (1998) modélisent la partie opérative et les programmes SFC en automates hybrides, puis le Model-Checking est utilisé avec la logique CTL. Quant à Bauer et al. (2004), ils proposent une modélisation des programmes SFC en automates à états finis ou en automates temporisés, vérifiés respectivement avec les outils Cadence SMV (McMillan, 1999) et Uppaal (Larsen et al., 1997). Dans Moon (1994) et Moon et al. (1991), les programmes LD sont traduits en utilisant le formalisme des systèmes de transitions et les propriétés sont exprimées en logique CTL. La technique du Model-Checking est utilisée pour la vérification formelle des propriétés. Les auteurs Canet et al. (2000) présentent une approche de validation par Model-Checking des programmes API en langage IL avec le Model-Checker SMV (McMillan, 1993). Ceux-ci sont formalisés en systèmes de transitions et les propriétés sont exprimées en logique LTL. Cette étude montre d'ailleurs que les résultats de la vérifica-

tion sont plus intéressants en présence du modèle de PO (approche model-based). Dans (Soliman and Frey, 2011), les auteurs proposent une démarche qui permet de transformer des programmes de commande écrits en FBD en automates temporisés sous le Model-Checker UPPAAL (Larsen et al., 1997), afin d'en vérifier des propriétés de sûreté. Enfin, Mokadem et al. (2010) proposent une méthode basée sur la réécriture des programmes LD en automates temporisés communicants sous UPPAAL, afin de vérifier des propriétés de sûreté et de vivacité.
Ces approches proposent toutes des solutions pour traduire les programmes API en langages formelles, mais les modèles de programmes obtenus sont souvent complexes et provoquent facilement l'explosion combinatoire de l'espace d'états. C'est le cas notamment des programmes SFC traduits en automates temporisés, et des programmes LD en réseau de Petri ou en automates.

Vérification formelle par Theorem-Proving.
La vérification par Theorem-Proving est basée sur une modélisation mathématique des programmes API et des propriétés. Des auteurs comme Völker and Krämer (1999) l'ont utilisée pour vérifier des programmes ST en formalisant les propriétés en logique LTL. L'avantage de cette approche est que son caractère mathématique fait qu'il ne s'expose pas à l'explosion combinatoire de l'espace d'états. Malheureusement cette technique ne peut pas être facilement automatisée car elle nécessite l'interaction humaine pour la manipulation des formules mathématiques (Clarke and Wing, 1996).

Vérification formelle par Reachabilty Analysis.
Les auteurs (Frey and Litz, 1998) et (Frey and Litz, 2000a) utilisent le formalisme du réseau de Petri pour la modélisation des programmes et de la Partie Opérative, puis la technique de l'analyse d'atteignabilité est utilisée pour la vérification. Quant à Kowalewski and Preußig (1996), ils utilisent le formalisme du système Condition/Événement pour la modélisation des programmes SFC et de la partie opérative. L'analyse d'atteignabilité permet ensuite de vérifier si les spécifications (définis sous formes d'états interdits) sont respectées.

Il existe donc dans la littérature différentes approches de formalisation et de vérification formelle des programmes API pour les langages normalisés (SFC, LD, ST, IL, FBD). Bien que ces méthodes reposent sur des approches (model-based, non-model-based, constraint

based), techniques formelles (Model Checking, Theorem-Proving, et Reachability Analysis), et langages de formalisation différents (automate, réseau de Petri...), la méthodologie de vérification est identique et consiste à exprimer un système sous forme de modèles et d'en vérifier des propriétés. Malheureusement, en cas d'erreurs humaines ou de manque d'expressivité du langage formel utilisé, cette phase de modélisation du système peut engendrer beaucoup de pertes d'informations sur celui-ci et compromettre les résultats de la vérification. Étant donné que la vérification se fait sur le modèle du système et non pas le système réel, il est important de vérifier les modèles avant leur implémentation. Ces techniques souffrent également de problèmes bien connus qui limitent leur utilisation, notamment :

— l'explosion combinatoire (sauf pour le Theorem-Proving),
— la complexité des langages formels,
— le manque d'automatisme (cas du Theorem-Proving),
— le manque d'expressivité des langages formels pour décrire fidèlement le comportement d'un système réel complexe.

2.5 Comparaison des méthodes de V&V

Le test, le Virtual Commissioning et les méthodes formelles représentent les principales méthodes utilisées en industrie pour la vérification et la validation des systèmes de contrôle commande. Les tests servent à vérifier et valider partiellement ou intégralement un système de contrôle commande en mode manuel ou automatique. Ces deux modes présentent chacun des avantages et des inconvénients, mais peuvent être combinés entre eux lors des tests, de manière à combiner leurs avantages respectifs. Les tests souffrent toutefois d'un manque d'exhaustivité. L'exécution des procédures de tests sur le système réel (procédé réel + système de contrôle commande) n'est pas suffisante pour garantir que la commande respecte formellement les spécifications tout en étant sûr de fonctionnement, comme le confirme d'ailleurs la célèbre déclaration d'Edgar W. Dijkstra (Buxton and Randell, 1970) : « Les tests montrent la présence et non l'absence de défauts ». Ils restent malgré tout l'approche la plus utilisée aujourd'hui dans le monde industriel.

Comme pour le test, le Virtual Commissioning peut être utilisé pour vérifier et valider des systèmes de contrôle commande complexes, ce qui justifie son importante utilisation dans l'industrie. Mais contrairement au test, le Virtual Commissioning peut être effectué dès les premières phases de la conception de la commande et ne nécessite pas la présence

de la PO réelle car celle-ci est remplacée par un système simulé pendant les essais. Cela permet de détecter les erreurs sur le système de contrôle commande avant qu'il ne soit implémenté et ainsi réduire les coûts de correction. En revanche, la simulation souffre de plusieurs inconvénients comme le manque de réalisme ou de granularité du modèle. De plus les essais de la simulation ne peuvent pas couvrir la totalité des évolutions possibles de la partie opérative, d'où son manque d'exhaustivité.

Cette exhaustivité est assurée par les méthodes formelles, car celles-ci prouvent la validité d'un système en vérifiant formellement que l'ensemble des exigences est satisfait par la totalité des évolutions possibles du modèle du système. Toutefois, pour un système de contrôle commande, elles ne servent qu'à vérifier et valider les programmes automates et non la partie matérielle (i.e. le paramétrage automates, le câblage de l'armoire électrique...). Malgré leurs avantages, plusieurs obstacles limitent l'utilisation de ces techniques dans le monde industriel (Bjørner and Havelund, 2014). Le manque d'automatisme dans le cas du Theorem-Proving par exemple, l'explosion combinatoire des états dans le cas du Model-Checking ou de l'analyse d'atteignabilité, et la difficulté ou le manque d'expressivité des langages formels pour des concepteurs métier, restent les plus récurrentes.

Ces trois méthodes ont un même objectif qui est de garantir que les spécifications ont été respectées par le système de contrôle commande. Elles ont toutes des avantages et des inconvénients, mais peuvent également être associées entre elles de manière à combiner leurs avantages respectifs (Fernandez et al., 2014) (Constant et al., 2007) (Rusu et al., 2004) (Tretmans, 1999). C'est justement cette association de méthodes que nous allons exploiter afin de proposer une approche méthodologique adaptée aux chargés d'études de la SNCF pour la vérification et la validation des systèmes de contrôle commande des EALE. Mais avant cela, il est important de faire l'état des lieux sur les méthodes actuellement utilisées par les chargés d'études de la SNCF, de les positionner par rapport à l'existant, et de mettre en évidence leurs manques afin de proposer une solution répondant au mieux à leurs besoins.

2.6 Méthodologie de V&V des systèmes de contrôle commande à la SNCF

Après les phases d'étude et de conception du système de contrôle commande des EALE, les chargés d'études de la SNCF procèdent à la vérification et la validation du système au moyen de tests de validation en usine (Factory Acceptance Test - FAT) puis sur site (Site Acceptance Test - SAT) à partir d'un cahier de recettes. Lorsqu'ils réceptionnent les armoires de contrôle commande chez l'intégrateur en usine, ils y transfèrent (après relecture du code API) la totalité des programmes automates qu'ils ont développés. Les procédures de tests vont ensuite être exécutées sur l'ensemble (suivant un certain ordre défini par les chargés d'études), pour vérifier et valider simultanément les programmes automates et le câblage des armoires (figure 28).

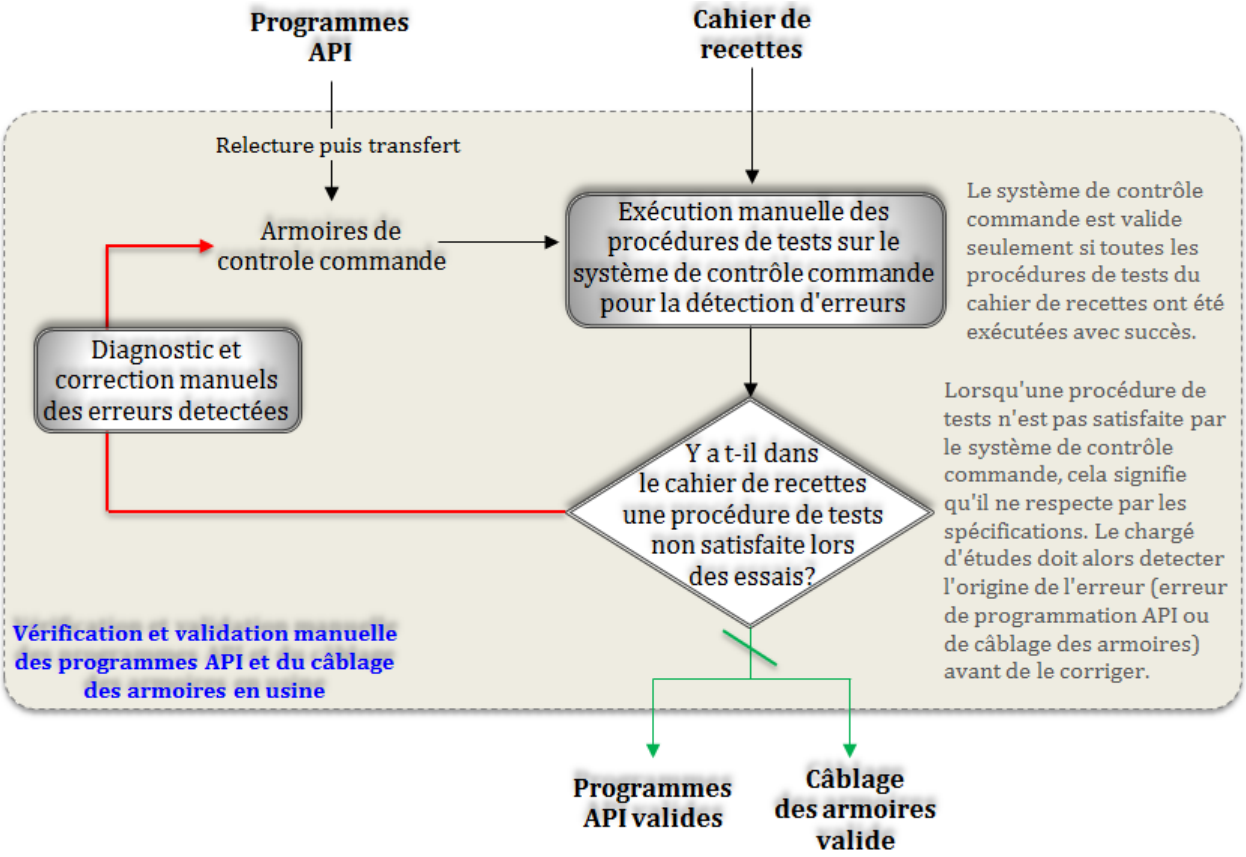


FIGURE 28 – Méthode de V&V des systèmes de contrôle commande à la SNCF

Étant donné que les essais usine se font sans présence de l'installation électrique réelle, les chargés d'études ont besoin de simuler son fonctionnement durant les essais en procédant comme suit :

— pour simuler des défauts : ils shuntent les entrées booléennes de l'API, et dans

certains cas ils procèdent à des injections de valeurs analogiques (courant/tension) sur les borniers de l'armoire.

— pour simuler le comportement des appareils électriques présents sur l'installation réelle (disjoncteurs, sectionneurs, interrupteurs...) : ils utilisent des relais électromagnétiques qu'ils connectent aux entrées/sorties du système de contrôle commande. Le principe de cette simulation au moyen de relais est défini par la figure 29.

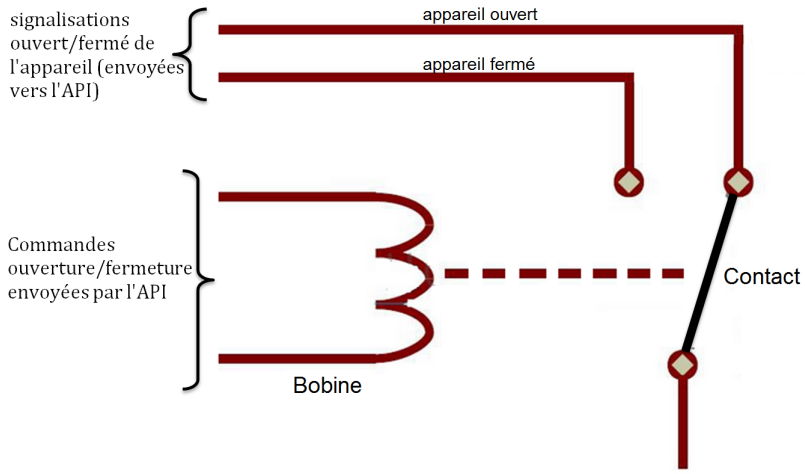


FIGURE 29 – Principe de la simulation d'un appareil électrique

Le relais électromagnétique simulant un appareil est connecté à la carte d'entrées/sorties de l'automate. Celui-ci envoie par conséquent des ordres d'ouverture et/ou de fermeture qui viendront exciter directement la bobine du relais. Ce dernier réagit alors en ouvrant/fermant instantanément ses contacts pour simuler les états ouvert/fermé de l'appareil concerné. Après avoir connecté tous les relais électromagnétiques nécessaires au système de contrôle commande, ils démarrent les essais en exécutant manuellement les scénarios de tests du cahier de recettes sur le système. Celui-ci ne peut être validé que lorsque tous les scénarios de tests ont été exécutés avec succès (i.e. il n'existe plus aucun scénario de tests bloquant). Puis après une phase de correction en usine, les mêmes essais ont lieu sur site en pré-production pour valider l'ensemble du système de contrôle commande avec le même cahier de recettes en présence de l'installation électrique réelle. Toutefois, comme nous l'avons mentionné dans l'introduction de ce chapitre, leur méthodologie de V&V présente plusieurs inconvénients que nous pouvons résumer en 4 points :

2.6.1 Manque d'automatisme de l'approche de V&V

Les scénarios de tests nécessaires pour vérifier et valider intégralement un système de contrôle commande sont très souvent répétitifs et se dénombrent en centaines. Avec des systèmes automatisés aussi complexes que les EALE, ces essais manuels prennent beaucoup de temps aux chargés d'études (1 à 2 semaines). La fatigue générée par ces efforts et le stress lié à la deadline de fin du projet exposent le chargé d'études à une surcharge mentale qui ne le met pas à l'abri d'erreurs humaines pendant les essais, ce qui peut compromettre la validité des tests réalisés (Coupat, 2014). De plus, ces tests peuvent être automatisés car ils consistent principalement à :

— ouvrir/fermer des appareils ou activer/désactiver des défauts (en ce qui concerne les actions à réaliser),
— observer les états des entrées/sorties booléennes et des variables réseaux (pour définir les conditions initiales et les résultats attendus de chaque scénario de tests).

Dans ces circonstances, le tableau 3 (chapitre 2) montre clairement que l'approche manuelle n'est pas adaptée, et que les tests automatiques présentent plus d'avantages. D'ailleurs selon Vu (2015), la V&V des systèmes de contrôle commande dans le domaine du ferroviaire se fait en général de manière informelle et manuelle, ce qui entraîne des pertes de temps, de coûts, et des erreurs humaines. Ainsi, l'automatisation de ce processus de V&V dans le domaine ferroviaire constitue un sujet de recherche actif (Haxthausen et al., 2010) (Winter, 2012) (Ferrari et al., 2011).

2.6.2 Test simultané des programmes et du câblage des armoires.

En usine, le cahier de recettes sert à valider à la fois les programmes API et le câblage des armoires. Cela signifie que pendant les essais, lorsqu'un scénario de tests devient bloquant (c'est-à-dire que la commande ne respecte pas une des spécifications), l'origine de cette anomalie peut provenir :

— soit d'erreurs dans les programmes automates,
— soit d'un mauvais câblage des armoires de contrôle commande.

Prenons en exemple le scénario de tests du tableau 4, dont l'objectif est de tester la réactivité d'un disjoncteur nommé DJ1 suite à l'apparition d'un court-circuit.

Supposons que lors de son exécution la première instruction du scénario de tests n'est pas satisfaite, c'est-à-dire que le disjoncteur DJ1 ne s'est pas ouvert suite à l'apparition

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 4 – Exemple d'un scénario de tests

d'un défaut court-circuit. On dit dans ce cas que le scénario de tests est bloquant. Deux explications peuvent justifier ce problème :

— l'entrée booléenne « Imax » associée 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 booléenne « Imax » a été correctement câblée sur l'automate, mais le programme du disjoncteur DJ1 n'a pas pu bien exploiter cette information pour ouvrir le disjoncteur associé : par conséquent le problème est lié à une mauvaise programmation de la commande du disjoncteur DJ1.

Dans ces circonstances, le chargé d'études doit diagnostiquer et corriger l'anomalie détectée avant de poursuivre les essais. Mais même avec l'expérience acquise au fil des recettes usines effectuées, ce diagnostic peut s'avérer complexe et peut prendre énormément de temps d'après le retour d'expérience des chargés d'études. Toutefois, il pourrait être moins complexe si les programmes automates pouvaient être séparément vérifiés et validés bien avant les tests de validation en usine, ce qui est tout à fait faisable car les programmes automates sont générés dès les premières phases du projet d'automatisation. De ce fait tout problème rencontré pendant les essais en usine proviendrait logiquement d'un mauvais câblage des armoires. Cela leur permettrait en même temps de réduire le nombre éventuels d'erreurs à corriger sur le système de contrôle commande durant la phase de validation en usine. Cette solution pourrait grandement faciliter et accélérer la phase de V&V en usine, mais ne peut pas être appliquée aujourd'hui par les chargés d'études car

ils ne disposent pas de méthodes (à part une simple relecture du code) leur permettant de vérifier et de valider séparément les programmes API avant la recette usine.

2.6.3 Manque d'exhaustivité du cahier de recettes

Depuis plusieurs décennies, les cahiers de recettes ont toujours été considérés comme des documents de références fiables permettant de vérifier toute la partie contrôle commande des installations électriques de la SNCF (programmes automates et câblage armoires). Ce sont des documents qui ont été conçus par les experts grâce à des retours d'expériences sur le fonctionnement nominal et dégradé des installations. Bien que le cahier de recettes ait servi à vérifier et valider les systèmes de contrôle commande des EALE depuis plusieurs années, et que l'expérience ait montré qu'il semblait suffisant pour garantir la sûreté et le bon fonctionnement de l'installation, rien ne prouve que les scénarios de tests sont exhaustifs. En effet, comme l'illustre la figure 30, les tests ne permettent pas d'explorer la totalité de l'espace d'état d'un procédé commandé. Les scénarios du cahier de recettes permettent de vérifier que le système de contrôle commande « fait bien ce qu'on lui demande de faire » (i.e. il respecte les spécifications fonctionnelles), mais ils ne permettent pas de prouver formellement que le système « ne fera jamais ce qu'on lui a interdit de faire » (i.e. il garantit la sûreté de fonctionnement du système).

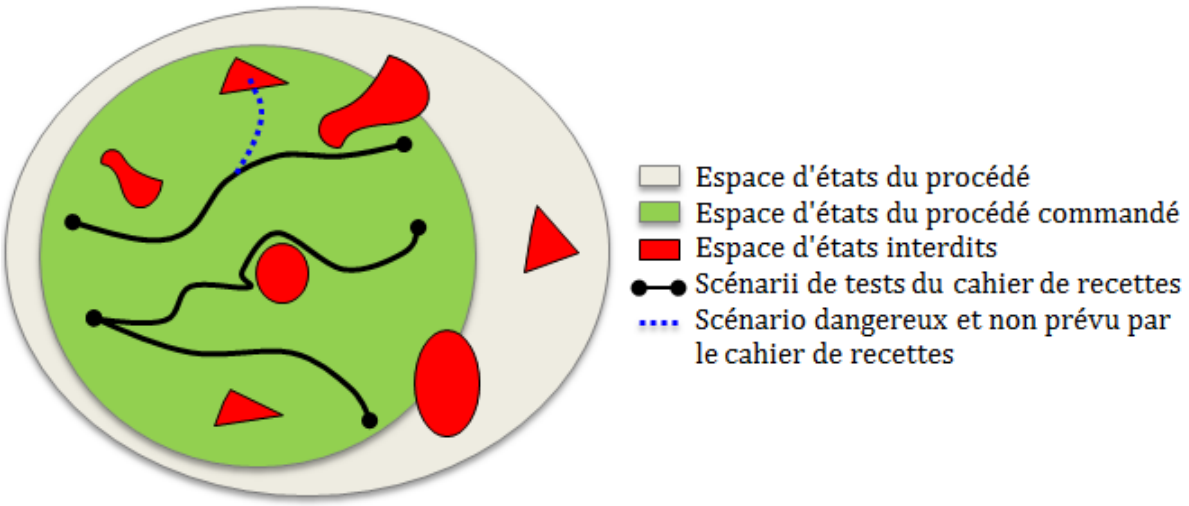


FIGURE 30 – Espace d'états d'un système

2.6.4 Manque de réalisme de la simulation des appareils

Lors des essais en usine, les chargés d'études utilisent des systèmes de relais électromagnétiques pour simuler le comportement des appareils présents dans l'installation, et

procèdent à des shunts sur les bornes de l'armoire pour simuler l'apparition de défauts sur l'installation. Leur technique de simulation est loin d'être moderne et ressemble fort aux toutes premières méthodes de simulation basées sur l'utilisation de boites à boutons. De plus, cette simulation manque de réalisme et ne reflète pas le vrai comportement des appareils présents dans l'installation électrique (disjoncteurs, sectionneurs, interrupteurs...). En effet, bien qu'ils servent globalement à isoler ou à connecter des circuits électriques, ces appareils n'ont pas la même structure ni le même fonctionnement. Ils se caractérisent par :

- des temps d'ouverture et de fermeture : ce que les relais électromagnétiques ne prennent pas en compte car ils s'ouvrent et se ferment instantanément,
- des nombres d'entrées variables (2 à 3 entrées pour commander l'ouverture et la fermeture de l'appareil) : ce qui n'est pas le cas non plus pour les relais,
- des types de commandes variables (commande ouverture/fermeture sur front montant, commande ouverture/fermeture sur front descendant, commande permanente...) : contrairement aux relais qui ne marchent qu'avec une commande permanente.

Ce manque de réalisme de la simulation est souvent à l'origine de la non-détection de quelques erreurs sur les programmes automates et le câblage des armoires des EALE, erreurs qui ne sont aujourd'hui détectables que pendant la phase de validation sur site en présence de l'installation réelle. On peut retenir de ce qui précède que les chargés d'études ont un réel besoin de simuler la partie opérative lors des essais en usine, mais leur approche manque d'efficacité et de réalisme.

Il apparaît clairement que la méthodologie de V&V des systèmes de contrôle commande à la SNCF présente des inconvénients qui peuvent compromettre la fiabilité des systèmes de contrôle commande. Par conséquent, il n'est pas à exclure que l'installation électrique soit mise en service avec son système de contrôle commande alors que celui-ci présente toujours des erreurs non détectées pendant les tests de validation. La preuve, près de 5% des pannes électriques qui se produisent aujourd'hui sur les EALE sont dues à des erreurs de conception du système de contrôle commande qui auraient pu être détectées et corrigées bien avant lors de la phase de V&V, ce qui confirme nos propos.

Pour remédier à ces inconvénients, nous proposons dans la section 3 une nouvelle approche méthodologique de V&V des systèmes de contrôle commande adaptée aux chargés d'études. Cette approche est une combinaison des différentes techniques présentées à la

section 2 de ce chapitre, de manière à bénéficier des avantages de chacune.

3 Proposition d'une nouvelle méthodologie de V&V des systèmes de contrôle commande des EALE

Notre première contribution à travers ce projet de recherche vise à réduire la complexité du diagnostic et de la correction des erreurs lors de la phase de V&V en usine. Comme nous l'avons montré à la section 2.6.2, la solution consiste à mettre en place une méthode qui permet de vérifier et de valider automatiquement les programmes automates dès leur conception.

3.1 V&V automatique des programmes API

Le système de contrôle commande des EALE est constitué de plusieurs APIs communiquant à travers un réseau industriel. Chaque API renferme un programme automate organisé sous forme de blocs fonctionnels instanciés dans le programme principal. La nouvelle approche de V&V automatique des programmes API se déroule en deux étapes :

— Étape 1 (phase 2-a du nouveau workflow) : Vérification formelle des blocs fonctionnels des programmes API (hors ligne) ;

— Étape 2 (phase 2-b) : Validation automatique des programmes API finaux par Simulation Software-In-the-Loop (en ligne).

3.1.1 Vérification formelle des blocs fonctionnels des programmes API

La première étape de cette approche consiste à vérifier hors ligne, séparément et formellement les blocs fonctionnels qui constituent le code principal du programme automate. Le choix d'une approche de vérification par bloc (à l'image des tests unitaires, section 2.2.2) facilite grandement le diagnostic et la correction des erreurs pour des programmes API aussi complexes. De plus, le choix d'une méthode formelle s'impose pour garantir l'exhaustivité de la vérification.

Parmi les méthodes formelles présentées à la section 2.4, le Model-Checking a été choisi car il répond le mieux à nos critères de sélection (vérification automatique, langage formel peu complexe comparé aux autres langages, possibilité de génération automatique des modèles...). Son atout majeur réside dans la génération de traces ou contre-exemples

après la vérification des programmes, car elle permet après détection d'une anomalie, d'en comprendre l'origine et de la corriger.

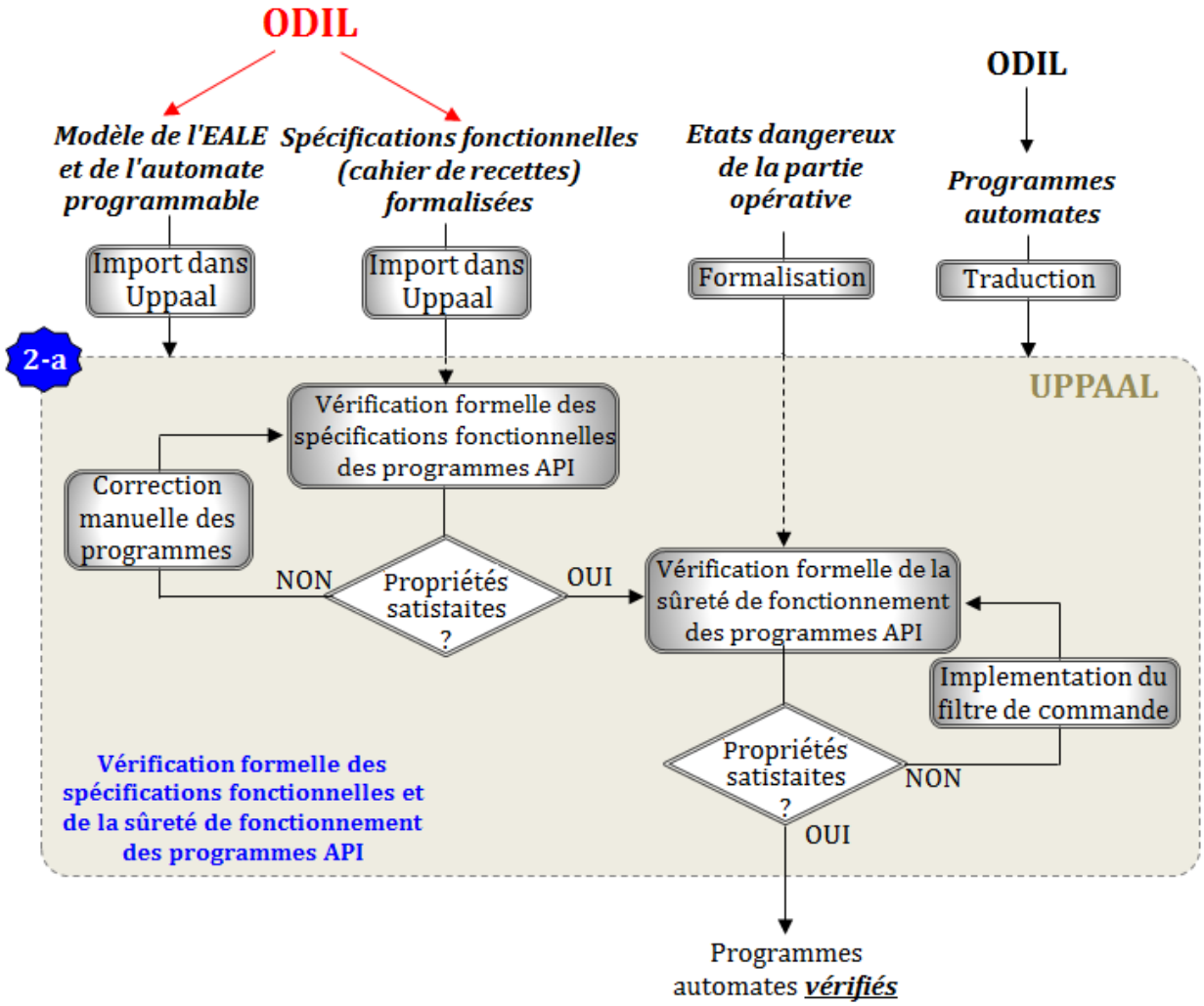


FIGURE 31 – Principe de la vérification formelle des programmes API

Notre approche de vérification par Model-Checking permet de prouver formellement que les programmes automates respectent les spécifications fonctionnelles tout en garantissant la sûreté de fonctionnement de la partie opérative (figure 31). Elle nécessite :

- une modélisation de la partie opérative (approche model-based) dans le Model-Checker (modèles qui pourront être générés automatiquement par ODIL, voir chapitre 3),
- une formalisation des propriétés à vérifier (spécifications fonctionnelles et états interdits),
- une traduction des programmes automates en un langage compréhensible par le Model-Checking. Cette traduction sera également automatisée par la suite pour éviter les erreurs humaines lors de la traduction.

Certaines données d'entrées (modèle de PO et de l'API, cahier de recettes formalisé) pourront être automatiquement générées par ODIL, afin d'éviter les erreurs de modélisation. La méthode de génération des modèles est présentée à la section 4 du chapitre 3.
Après vérification formelle, lorsque les programmes automates ne respectent pas les spécifications fonctionnelles, ils feront l'objet de correction manuelle de la part du chargé d'études. Par contre, lorsque la sûreté de fonctionnement n'est pas garantie, nous proposons de rajouter un filtre logique de commande par contraintes (Pichard et al., 2018) de manière à les rendre formellement sûrs de fonctionnement. Cette approche sera davantage détaillée dans le chapitre 3.

Après la vérification formelle et la correction des blocs fonctionnels nécessaires à la programmation des APIs, ceux-ci peuvent être instanciés et assemblés pour constituer le programme principal. La suite de notre approche méthodologique de V&V consiste à valider automatiquement le programme API final (assemblage des blocs fonctionnels) grâce à la simulation Software-In-the-Loop.

3.1.2 Validation automatique des programmes API par Simulation Software-In-the-Loop

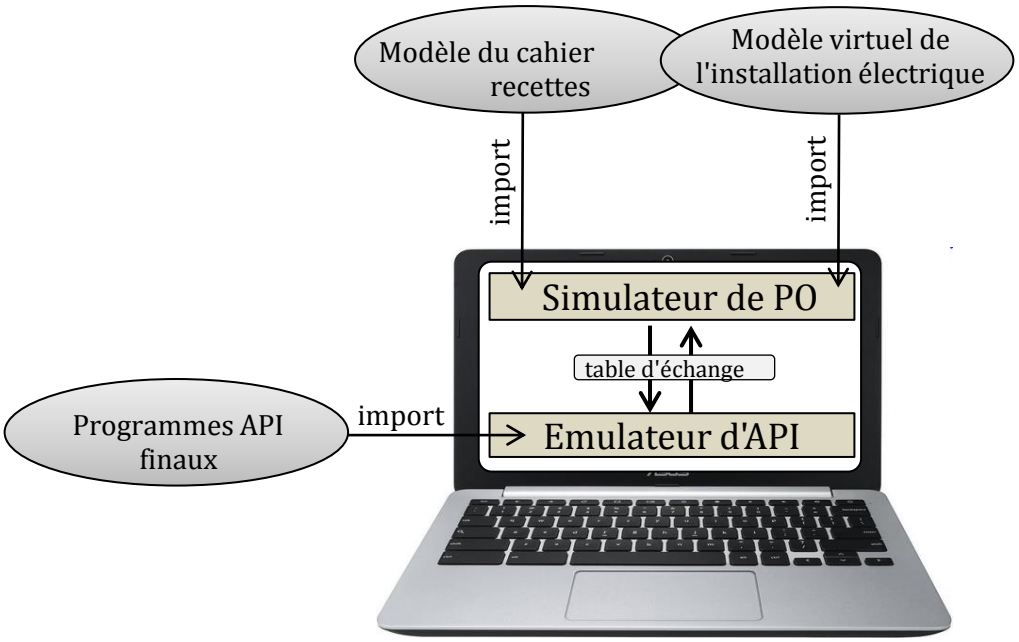


FIGURE 32 – Principe de la validation du programme par Virtual Commissioning type SIL

A la différence de la première étape qui nécessite une traduction des blocs fonctionnels en un langage intermédiaire, la deuxième étape consiste à valider directement les pro-

grammes automates (formellement vérifiés et corrigés) grâce à la simulation Software-In-the-Loop. Le principe de cette technique (figure 32) consiste à transférer le programme API dans un émulateur d'automate, puis à faire communiquer celui-ci via une table d'échange avec un logiciel de Virtual Commissioning contenant :

— le modèle virtuel de l'installation électrique,
— le cahier de recette numérisé contenant toutes les procédures de tests nécessaires à la validation des programmes API.

Ces données seront également générées automatiquement par ODIL, pour éviter les erreurs de modélisation. La validation des programmes API finaux se déroule alors en deux étapes :

— Étape 1 : exécuter librement des scénarios sur l'installation, en rejouant par exemple les mêmes scénarios dangereux retournés par le Model-Checking pour s'assurer qu'ils ne se reproduiront plus. Cette étape permet également de vérifier les modèles de PO et du cahier de recettes générés par ODIL.
— Étape 2 : exécuter automatiquement les procédures de tests sur les programmes API communiquant avec la PO simulée, pour valider les spécifications fonctionnelles.

Cette technique de simulation est beaucoup plus performante que celle utilisée par les chargés d'études, et présente beaucoup d'avantages (sections 2.3.1 à 2.3.5). En effet elle permet de valider automatiquement les programmes API en toute sécurité dès les premières phases du projet, car elle ne nécessite pas la présence de l'installation réelle. De plus les chargés d'études passeront moins de temps en usine lors des tests de validation, car seul le câblage des armoires sera testé (section 2.6.2).

3.2 Validation automatique du câblage des armoires par Simulation Hardware-In-the-Loop

A ce stade les programmes automates sont déjà validés et transférés dans les APIs des armoires de contrôle commande. Mais avant la mise en service, le câblage des armoires doit être validé. Pour cela le chargé d'études réutilise le même logiciel de simulation que précédemment (renfermant la PO simulée et le cahier de recettes), sauf que celui-ci sera exploité dans une configuration type Hardware-In-the-Loop, c'est-à-dire connecté au système de contrôle commande réel par l'intermédiaire d'une interface physique (figure 33).

La validation consiste à exécuter automatiquement toutes les procédures de tests du

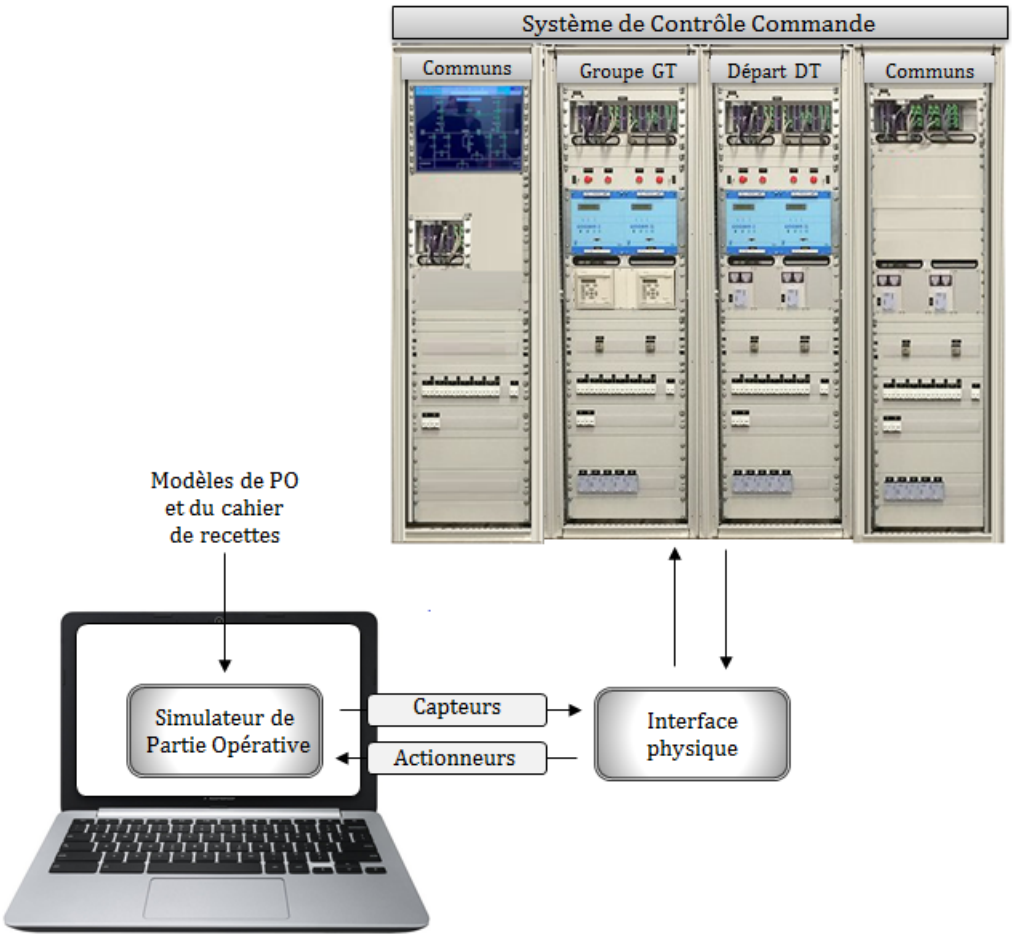


FIGURE 33 – Principe de la validation du câblage des armoires par Virtual Commissioning type HIL

cahier de recettes sur le système de contrôle commande, ce qui est suffisant pour vérifier que le câblage ne présente pas d'erreurs. Puis après les corrections, le système de contrôle commande est prêt pour la mise en service de l'installation sur site.

4 Conclusion

Dans ce chapitre, nous avons présenté les trois principales méthodes de V&V des systèmes de contrôle commande utilisées en industrie. Nous avons vu que le test qui représente la méthode la plus utilisée en industrie présente beaucoup d'avantages, mais souffre d'un manque d'exhaustivité pour garantir formellement la sûreté de fonctionnement d'un système. Il en est de même pour le Virtual commissioning qui se base également sur des essais non exhaustifs, mais remédie aux inconvénients du Real Commissioning (sections 2.3.1 à 2.3.5). Cette exhaustivité est assurée par les méthodes formelles, car celles-ci prouvent la validité d'un système en vérifiant formellement que l'ensemble des exigences est satisfait

par la totalité des évolutions possibles du modèle du système. Cependant l'utilisation de cette technique en industrie est limitée par les inconvénients qu'elle présente (complexité des langages formels pour la modélisation, explosion combinatoire, vérification du modèle du système et pas du système réel...). Nous avons ensuite montré qu'en associant ces techniques, nous pouvons aboutir à une approche méthodologique qui bénéficie des avantages de chacune d'entre elles.

Pour remédier aux problèmes que rencontrent les chargés d'études lors des phases de validation, nous proposons une approche méthodologique combinant la vérification formelle et le Virtual Commissioning (HIL et SIL) pour la V&V du système de contrôle commande des EALE. Cette approche présente l'avantage d'être plus rapide, plus exhaustive et plus fiable que celle utilisée par les chargés d'études. Les chapitres 3 et 4 présentent respectivement la méthode de vérification hors ligne des programmes API par Model-Checking et la validation en ligne du système de contrôle commande par Virtual Commissioning (SIL et HIL). Ces contributions, ainsi que celles de Coupat (2014), ont été présentées dans ZAYTOON (2018) en session plénière, à l'occasion de la $18^{ième}$ conférence international ICCAS (*International Conference on Control, Automation and Systems*) en Corée du Sud.

Chapitre III

Vérification formelle des blocs fonctionnels des programmes API

1 Introduction

La vérification est une phase incontournable qui précède et complète la validation d'un système de contrôle commande. Tandis que la première phase répond à la question « Construisons-nous correctement le produit ? », c'est-à-dire si les exigences spécifiées sont satisfaites, la deuxième répond à la question « Avons-nous construit le bon produit ? » (ISO-9000, 2015). Dans ce chapitre, nous nous intéressons particulièrement à la vérification des programmes automates, et plus précisément à la vérification formelle. Cette méthode formelle a été retenue pour pallier le manque d'exhaustivité des méthodes de vérification traditionnellement utilisées dans le milieu industriel, à l'instar des tests et de la simulation.

L'objectif de ce chapitre est de présenter la méthodologie de vérification formelle des spécifications fonctionnelles et de la sûreté de fonctionnement des programmes automates dédiés à la commande des EALE. Pour cela, la technique du Model-Checking a été retenue grâce à ses nombreux avantages. La vérification formelle des programmes automates nécessite globalement 3 étapes : la modélisation de la partie opérative, la traduction des programmes automates, ainsi que la formalisation des propriétés à vérifier. Cette méthodologie est détaillée dans la section 2 de ce chapitre. Ensuite, la section 3 présente un exemple concret d'application de la méthodologie pour la vérification formelle des « programmes d'asservissement Fil Pilote (FP) entre disjoncteurs de Départ Traction 1500 V ». Ces programmes utilisés ont été fournis par les chargés d'études dès leur conception, afin qu'ils soient formellement vérifiés. L'efficacité de notre approche pourra ainsi être

évaluée par les chargés d'études. Nous proposons également dans la section 3, une technique qui permet de rendre un programme automate formellement sûr de fonctionnement sans modifier le contenu du programme, grâce à l'implémentation d'un filtre logique de commandes par contraintes ou filtre de sécurité.

2 Principe de la méthode

Le principe de la méthodologie de vérification formelle des blocs fonctionnels (Niang et al., 2017) est résumé par la figure 31. Comme indiqué à la section 3.1.1 du chapitre 2, la vérification formelle des blocs fonctionnels concerne aussi bien l'aspect fonctionnel que l'aspect sécurité. La vérification de l'aspect fonctionnel du programme automate (appelé propriété 1) permet de s'assurer que celui-ci est en accord avec les spécifications fonctionnelles décrites dans le cahier des charges, et celle de l'aspect sécurité (propriété 2) consiste à vérifier si le système commandé est susceptible ou non d'atteindre des états dangereux ou interdits qui pourraient entraîner une détérioration du matériel.

Vérification des spécifications fonctionnelles (propriété 1). Pour vérifier cette propriété, notre approche méthodologique s'inspire de la technique de vérification actuellement utilisée par les chargés d'études en usine. Celle-ci se base sur l'exécution du cahier de recettes sur le système de contrôle commande. En effet, ce document renferme l'ensemble des scénarios de tests nécessaires pour vérifier que le programme automate « fait bien ce qu'on lui demande de faire » (i.e. il respecte les spécifications fonctionnelles). La vérification de la propriété 1 par Model-Checking consiste alors à prouver formellement qu'en exécutant tous les scénarios de tests sur le système « PO + programme API », il n'existe aucune instruction ou action bloquante. En d'autres mots, elle consiste à vérifier que toute instruction exécutée sur le programme API mènera forcément le système vers le résultat attendu par le cahier de recettes.

En cas de non-respect de la propriété 1, le Model-Checker retourne sous forme de trace le scénario de test ainsi que le numéro de l'instruction ayant entraîné le blocage de la simulation. Pour cela, il est nécessaire de :

— modéliser les programmes API et la partie opérative dans le Model-Checker,

— modéliser les scénarios de tests,

— programmer l'exécution des scénarios de tests (dans un certain ordre défini par les

chargés d'études) sur le système « programme API + Partie Opérative »

— surveiller l'état de chaque instruction après son exécution, pour vérifier si elle est bloquante ou non.

Vérification de la sûreté de fonctionnement (propriété 2). Le cahier de recettes n'est pas assez exhaustif pour prouver formellement que le programme automate garantit formellement la sûreté de fonctionnement du système, c'est-à-dire que ce document ne permet pas de prouver que la partie opérative n'atteindra jamais un état dangereux ou interdit quelque soit la commande envoyée par l'opérateur. En effet, le cahier des recettes ne permet d'explorer qu'une partie des comportements possibles (espace d'états) du système (figure 30). La solution serait de parcourir intégralement l'ensemble des évolutions possibles du système et de vérifier l'atteignabilité ou non des états interdits. La vérification de la propriété 2 consiste alors à prouver de manière exhaustive par Model-Checking que quels que soient l'état de la partie opérative et la combinaison de commandes envoyée au système par un opérateur, l'installation n'atteindra jamais les états interdits définis dans le cahier des charges. Ces états interdits devront nécessairement être traduits en langage formel.

En cas de violation de la propriété 2, le Model-Checker retourne la trace renfermant la séquence d'événements ayant abouti à cet état. Cette approche sera également détaillée dans la section 2.5.2.

Parmi les outils de Model-Checking existants, le Model-Checker **Uppaal** (Larsen et al., 1997) a été choisi pour la vérification de ces propriétés. Son aspect graphique ainsi que les traces retournées après vérification des propriétés exprimées en logique temporelle CTL, facilitent grandement le suivi du déroulement de la vérification. La section suivante présente ce Model-Checker.

2.1 Présentation du Model-Checker Uppaal

Uppaal est un Model-Checker qui permet d'analyser des systèmes formés d'une collection d'automates temporisés communiquants. Il a été développé en 1995 par les universités **Uppsala** (de la Suède) et **Aalborg** (Danemark). C'est un outil qui propose un mode de vérification graphique par model-checking pour des systèmes temps réel. Son intérêt est qu'il permet de simuler le fonctionnement d'un système modélisé en automates temporisés et de revoir le déroulement d'une vérification. Ainsi la trace retournée après la vérifica-

tion d'une propriété peut être analysée afin de parcourir la succession d'événements qui a conduit le système vers cet état.

Les automates temporisés de ce Model-Checker sont principalement constituées de locations (états) et de transitions permettant de passer d'un état à un autre. Ce sont des structures finies manipulant deux types de variables : des horloges (qui évoluent de manière synchrone avec le temps) et des variables entières (ou discrètes) bornées. Un état (ou location) de l'automate peut comporter une condition sur les horloges, appelée invariant, qui doit être satisfaite pendant toute la durée passée dans cet état. Les états peuvent être étiquetés de deux manières possibles :

— L'étiquette **U** pour **Urgent** : Lorsqu'un modèle évolue vers un état type **Urgent**, l'horloge du modèle n'évolue pas ;

— L'étiquette **C** pour **Committed** : Cette notion est plus restrictive que la notion d'état **Urgent**. Lorsque le modèle évolue vers un état marqué comme **Committed**, il doit immédiatement le quitter après son arrivée en empruntant une des transitions sortantes.

Une transition de l'automate temporisé est étiquetée par :

— une garde qui exprime la condition de franchissement de la transition,

— une synchronisation avec les autres modèles sous forme de message,

— une mise à jour des variables.

Notons qu'il existe 2 types de synchronisation (Mokadem, 2006) :

— la synchronisation binaire : un émetteur envoie un signal (ou **channel**) et un unique récepteur se synchronise avec lui par le signal complémentaire. Les transitions de ces deux composants respectivement étiquetées par « m ! » et « m ? » (avec **m** le nom du **channel**) sont franchies en même temps. Dans le cas où, à partir d'une configuration, plusieurs couples de transitions peuvent être synchronisées, le choix est non déterministe. Ce mode de synchronisation peut conduire à des blocages si un automate envoie un signal m ! et qu'aucun composant ne peut le recevoir (par exemple si la garde de la transition étiquetée par m ? est fausse).

— La synchronisation multiple (broadcast channel) : les transitions étiquetées par m ? et m ! sont synchronisées, mais cette fois avec un émetteur et plusieurs récepteurs.

La vérification formelle des programmes automates nécessite trois activités (figure 34) :

— La modélisation (en automates temporisés) des sous-ensembles de l'EALE qui représente la partie opérative ;

— La modélisation de l'automate et des programmes API : les programmes doivent être traduits dans un langage compatible avec le Model-Checker, et l'API doit également être modélisé pour simuler l'exécution cyclique des programmes ;

— La formalisation des propriétés à vérifier, à savoir les spécifications fonctionnelles et les états interdits, qui sont respectivement formalisés en automates temporisés et en logique temporelle CTL.

Ces modèles sont par la suite regroupés pour constituer le modèle global.

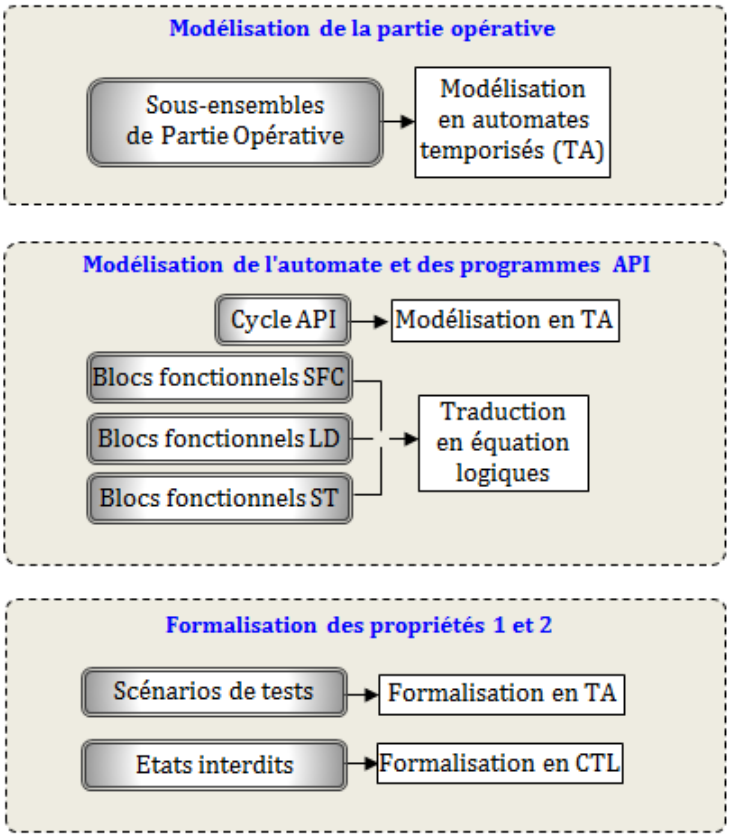


FIGURE 34 – Les étapes de la modélisation pour la vérification des programmes API

Pour vérifier formellement ces programmes automates, il est nécessaire que les conditions de simulation se rapprochent de la réalité, c'est-à-dire comme pour un vrai système automatisé. Les modèles de programmes automates doivent donc être exécutés de manière cyclique dans un modèle d'API communiquant avec le modèle de la partie opérative. Pour cela, les différents modèles présentés à la figure 34 doivent être connectés et synchronisés entre eux. Cette synchronisation est assurée par le modèle du cycle automate présenté à la figure 35.

2.2 Modélisation du cycle automate (Mokadem, 2006)

Les programmes automates sont exécutés selon un cycle constitué globalement de 3 phases (hormis la phase d'initialisation durant le premier cycle automate) :

— lecture des entrées,

— exécution du programme principal,

— mise à jour des sorties.

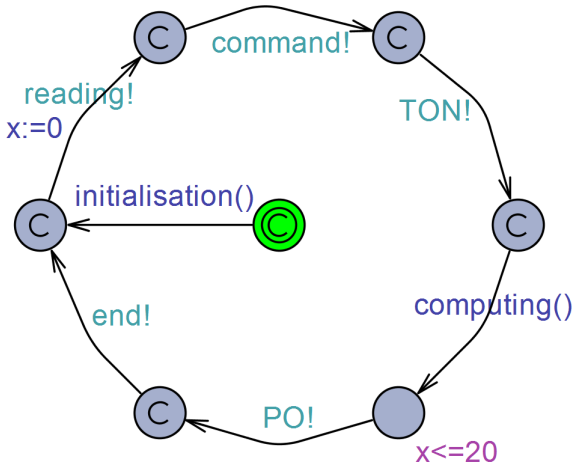


FIGURE 35 – Modélisation du cycle automate

Le modèle du cycle automate (figure 35) permet de synchroniser l'ensemble des modèles du système tout en respectant l'ordre d'exécution séquentielle des tâches cycliques du programme automate. Lors du tout premier cycle API, une phase d'initialisation est nécessaire pour réaliser certaines opérations telles que l'initialisation des programmes SFC, des variables internes, des états des appareils de la partie opérative (ouvert ou fermé) et des paramètres (temporisations d'ouverture et de fermeture des appareils). Les synchronisations sont du type **broadcast**, le cycle normal commence par une remise à zéro de la durée du cycle automate (horloge interne **x**) et une lecture des entrées (message **reading !**) telles que les défauts surveillés sur la partie opérative et les positions des appareils. Ensuite, les demandes d'ouverture et de fermeture envoyées par l'opérateur à destination des appareils sont récupérées (signal **command !**). Les messages **reading !** et **command !**, émis par le modèle du cycle automate, sont également reçus par les modèles utilisés seulement lors de la vérification de la sûreté de fonctionnement des programmes (voir section 2.5.2).

Après la lecture des entrées du système, il y aura l'évolution des temporisateurs de la figure 46 (synchronisées avec le signal **TON !**), suivie de l'exécution du programme principal **computing()** (construction des observateurs, évolution des programmes SFC,

... voir figure 51). Il s'ensuit l'étape d'évolution de la partie opérative grâce au signal **PO !** envoyé après exécution du programme principal. Ce signal **PO !** sera reçu par chacun des modèles constituant la partie opérative afin de faire évoluer leur état (figures 40, 41, 42 et 43). La fin du cycle courant est signalée par le message **end !**. Ce signal synchronise le modèle de la figure 54 permettant d'exécuter les instructions du cahier de recettes sur l'ensemble « Programmes API + EALE ». Cette partie sera détaillée à la section 2.5.1.

Certaines tâches telles que l'acquisition des entrées ou l'exécution des programmes sont instantanées. Ceci se traduit en Uppaal par le fait que les états concernés sont étiquetés par **c** (pour **committed**). La présence de l'invariant ***x<=20*** représente la durée du cycle API. Cette durée est estimée de sorte que l'automate simulé puisse détecter les moindres changements des valeurs des entrées de la partie opérative (telles que les états ouverts/fermés des appareils par exemple).

2.3 Modélisation de la partie opérative

La modélisation de la partie opérative dans le Model-Checker nécessite tout d'abord une analyse structurelle et fonctionnelle des installations électriques. Celle-ci permet d'étudier et de classifier les différents sous-ensembles qui constituent la partie opérative. L'idée consiste à modéliser la totalité des sous-ensembles de partie opérative qu'on retrouve en général dans une installation électrique, et de stocker ces modèles dans une base de données. De cette manière, toute nouvelle installation électrique peut être modélisée grâce à une instanciation directe et un assemblage des modèles de sous-ensembles de l'installation électrique cible. Les EALE sont principalement constitués de disjoncteurs, sectionneurs, interrupteurs, jeux de barres, transformateurs, et de redresseurs.

2.3.1 Modélisation des disjoncteurs-sectionneurs-interrupteurs

Ces 3 types appareils servent globalement à isoler ou à connecter des circuits électriques. En fonction de sa nature et du rôle qu'il joue sur l'installation, chaque type d'appareil peut également avoir plusieurs sous-types. Au total, une vingtaine de types d'appareils électriques a été dénombrée. Ces appareils peuvent toutefois être classifiés en 4 catégories selon leur comportement :

Appareil type 1. La figure 36 décrit l'ensemble des entrées/sorties et paramètres pour ce type d'appareil :

— Les commandes **open** et **close** correspondent aux ordres d'ouverture et de fermeture envoyés par le programme API vers l'appareil.

— Les signalisations **so** et **sf** correspondent respectivement aux retours de positions « **ouvert** » et « **fermé** » de l'appareil envoyés vers l'API.

— Les paramètres **TimeOP** et **TimeCL** correspondent aux temps nécessaires à l'appareil pour effectuer respectivement une ouverture et une fermeture complète. Ces durées peuvent varier selon le type et la fonction de l'appareil, elles sont donc modifiables lors de l'instanciation de l'appareil.

— Les paramètres de temps **Tmin** et **Tmax**, dont les valeurs dépendent des appareils. Leur rôle sur le fonctionnement de l'appareil sera décrit dans l'analyse fonctionnelle.

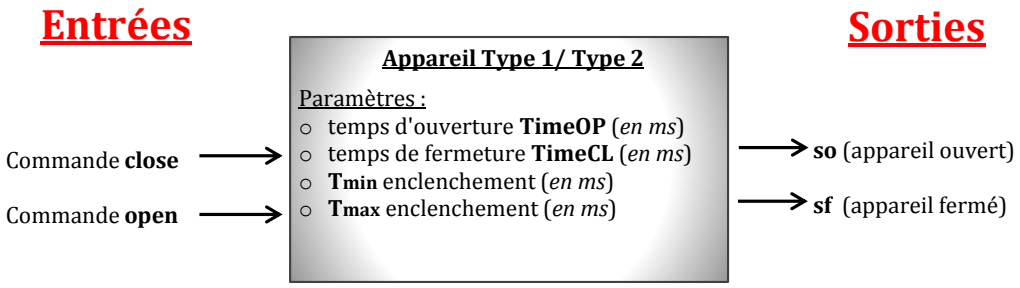


FIGURE 36 – Analyse structurelle appareil type 1 et type 2

Initialement, l'appareil est toujours en position ouverte (so=1 et sf=0). Le chronogramme de la figure 37 résume le fonctionnement de l'appareil.

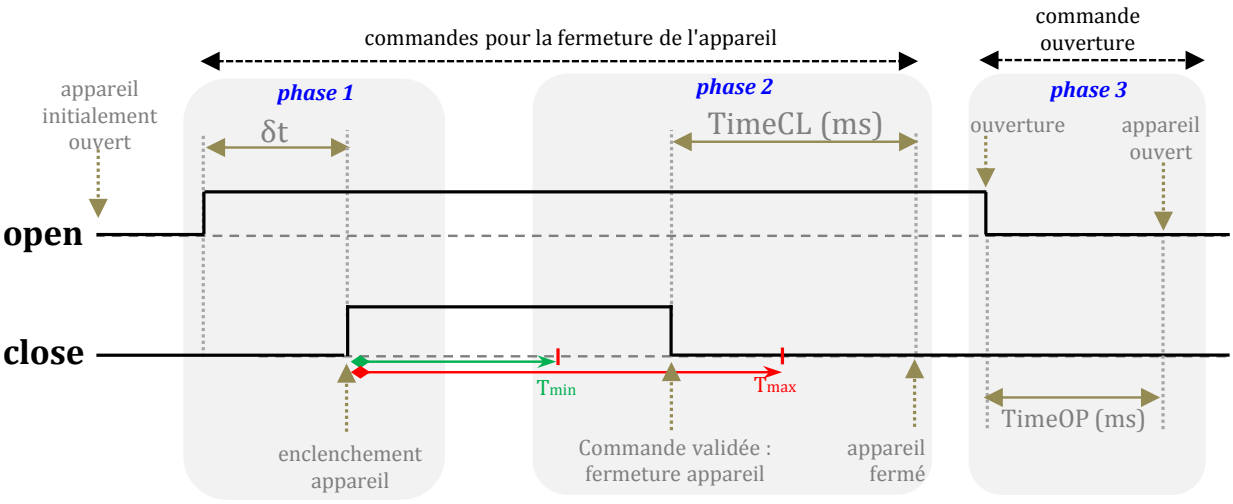


FIGURE 37 – Analyse fonctionnelle appareil type 1

Les entrées **open** et **close** sont initialement inactives. Les conditions de fermeture de l'appareil sont définies par la séquence suivante : l'entrée **open** devra d'abord être activée et maintenue, puis ce sera le tour de l'entrée **close** (voir phase 1 de la figure 37).

L'activation de la commande **open** avant celle de la commande **close** est nécessaire, sans quoi la commande n'est pas acceptée. La présence du δ**t** sur le chronogramme représente le décalage entre les deux activations. A partir de ce moment précis (c'est-à-dire au front montant de la commande **close**), la commande de fermeture de l'appareil ne sera valide que si la durée du signal **close** (c'est-à-dire la durée mesurée entre le front montant et le front descendant de la commande **close**) est comprise entre **Tmin** et **Tmax** (phase 2), autrement la commande ne sera pas valide et l'appareil restera ouvert. Lorsque la commande est valide, l'appareil entame sa fermeture (so = 0 et sf = 0) avant de se fermer complètement (so = 0 et sf = 1) au bout d'un certain temps (paramètre **TimeCL** en ms). Une commande de fermeture sur un appareil déjà fermé n'a aucun effet sur celui-ci.
En ce qui concerne la commande ouverture (phase 3), la désactivation de la commande **open** provoque automatiquement l'ouverture complète du disjoncteur. La commande d'ouverture est toujours prioritaire sur la commande de fermeture (valable également pour les 3 autres types d'appareil). De plus, une commande d'ouverture sur un appareil déjà ouvert n'a aucun effet sur celui-ci.

Appareil type 2. La structure de l'appareil type 2 est la même que celle du type 1, c'est-à-dire qu'ils ont les mêmes entrées/sorties et paramètres internes. Ils diffèrent toutefois au niveau fonctionnel. De plus, contrairement au 1er type d'appareil, celui-ci peut-être initialement ouvert (so=1 et sf=0) ou fermé (so=0 et sf=1). La position initiale par défaut étant la position ouverte, l'utilisateur peut la modifier si nécessaire avant le début de la simulation. Le chronogramme de la figure 38 décrit le comportement de l'appareil type 2.

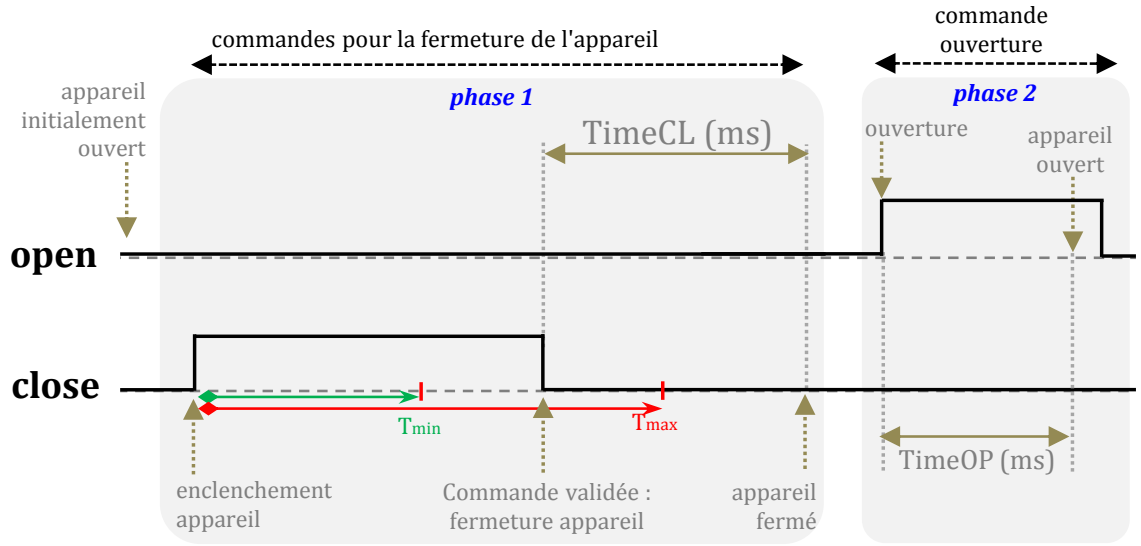


FIGURE 38 – Analyse fonctionnelle appareil type 2

En position ouverte, le principe de fermeture pour ce type d'appareil est à peu près identique à celui de l'appareil type 1, c'est-à-dire que la fermeture s'enclenche au front descendant de la commande **close** (toujours entre **Tmin** et **Tmax**). La seule différence est que la commande **open** doit nécessairement rester inactive tout au long de la manœuvre, sans quoi la commande ne serait pas valide. Lorsque la commande de fermeture est acceptée, l'appareil entame sa fermeture (so = 0 et sf = 0) avant de se fermer complètement (so=0 et sf = 1) au bout d'un certain temps (paramètre **TimeCL** en ms). Une commande de fermeture sur un appareil déjà fermé n'a aucun effet sur celui-ci.
En position fermée, l'appareil entame son ouverture au front montant de la commande **open**. De plus, une commande d'ouverture sur un appareil déjà ouvert n'a aucun effet sur celui-ci.

Appareil type 3. La figure 39 décrit l'ensemble des entrées/sorties et paramètres pour ce type d'appareil. Ces modèles ne renferment pas de paramètres **Tmin** et **Tmax**.

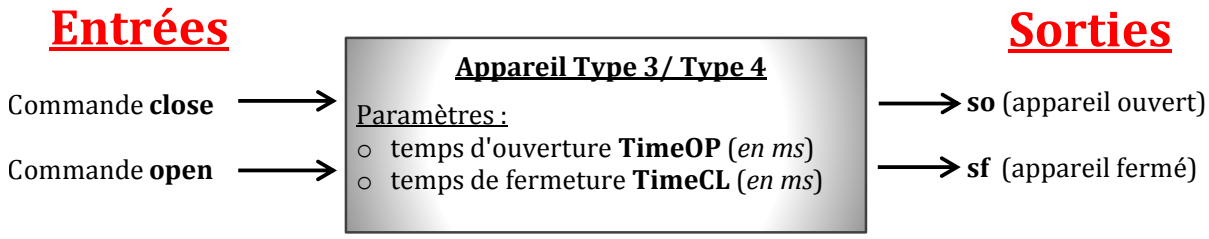


FIGURE 39 – Analyse structurelle appareil type 3 et type 4

À la différence des appareils types 1 et 2, ce type d'appareil a un comportement assez simple. En effet, l'ouverture de l'appareil (respectivement la fermeture) s'enclenche dès qu'il détecte un front montant de la commande **open** (respectivement **close**). Lors de la fermeture, la commande **close** doit être maintenue tout au long de la manoeuvre, sans quoi l'appareil retourne à sa position ouverte. Il peut être initialement ouvert ou fermé à l'instar de l'appareil type 2.

Appareil type 4. La structure de l'appareil type 4 est la même que celle du type 3. Ils ont également un même fonctionnement, sauf que la commande d'ouverture **open** est inversée pour ce type d'appareil. C'est-à-dire que l'ouverture s'enclenche à la désactivation de la commande **open**, à l'instar de l'appareil type 1.

Ces analyses structurelles et fonctionnelles ont permis d'établir les modèles Uppaal (à base d'automates temporisés) des 4 types d'appareils électriques.

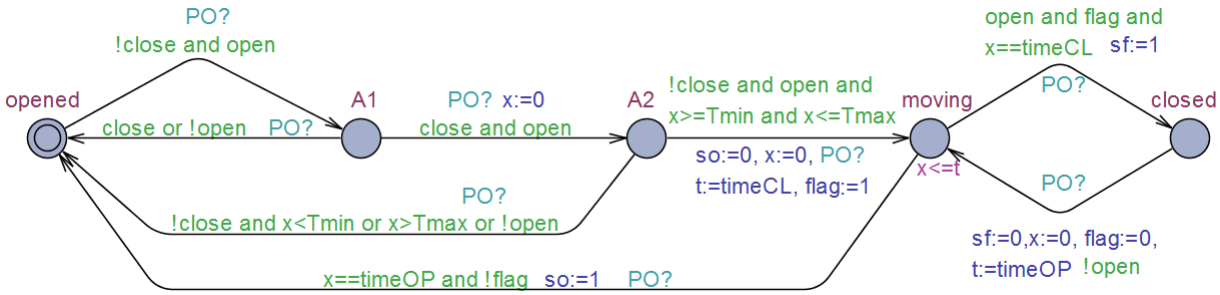


FIGURE 40 – Modèle Uppaal de l'appareil type 1

La figure 40 présente le modèle Uppaal du comportement de l'appareil type 1 décrit précédemment. Ce modèle renferme 5 états (ou locations), on y retrouve les mêmes variables et paramètres décrits lors de l'analyse structurelle, à l'exception des suivants :

- le paramètre **x** qui représente l'horloge interne du modèle (variable type **clock**),
- l'observateur booléen **flag** qui indique si l'appareil est en ouverture ou en fermeture,
- le paramètre **t** qui est un entier stockant les durées **TimeOP** lors de l'ouverture de l'appareil et **TimeCL** lors de la fermeture,
- le message **PO ?** qui indique que ce modèle est synchronisé avec le modèle du cycle API de la figure 35. Grâce à cette synchronisation, l'état de la partie opérative est rafraîchi à la fin de chaque cycle automate.

Pour respecter la position initialement ouverte pour ce type d'appareil, la signalisation **so** est initialisée à **true** (non visible sur la figure 40) et la location **opened** est marquée « initial ». En exécutant correctement la séquence décrite par la phase 1 de la figure 37, le modèle Uppaal de l'appareil type 1 évolue de l'état **opened** à l'état **A1**, puis de **A1** vers **A2** si la séquence d'activation est respectée. Le passage à l'état **A2** réinitialise l'horloge interne **x** du modèle pour mesurer le temps écoulé entre le front montant de la commande **close** et son front descendant. Lorsque cette durée est comprise entre **Tmin** et **Tmax** (phase 2 de la figure 37), le modèle évolue vers l'état **moving** et la signalisation **so** est désactivée, ce qui signifie que l'appareil entame sa fermeture. En cas de non-respect de la séquence de fermeture, le modèle retourne à l'état initial **opened** pour interrompre la demande de fermeture.

A l'état **moving**, l'horloge interne **x** est réinitialisée à nouveau pour mesurer cette fois-ci le temps écoulé pendant la fermeture de l'appareil. Lorsque cette durée atteint la valeur **TimeCL** et que la commande **open** est toujours maintenue, le modèle évolue vers l'état **closed** : l'appareil s'est complètement refermé (**sf** = true). A partir de cet état,

une simple désactivation de la commande **open** entraîne l'ouverture de l'appareil au bout d'un temps **TimeOP**. A noter que pendant la fermeture de l'appareil (à l'état **moving**), la désactivation de la commande **open** entraîne également l'ouverture instantanée de l'appareil comme le décrit son analyse fonctionnelle.

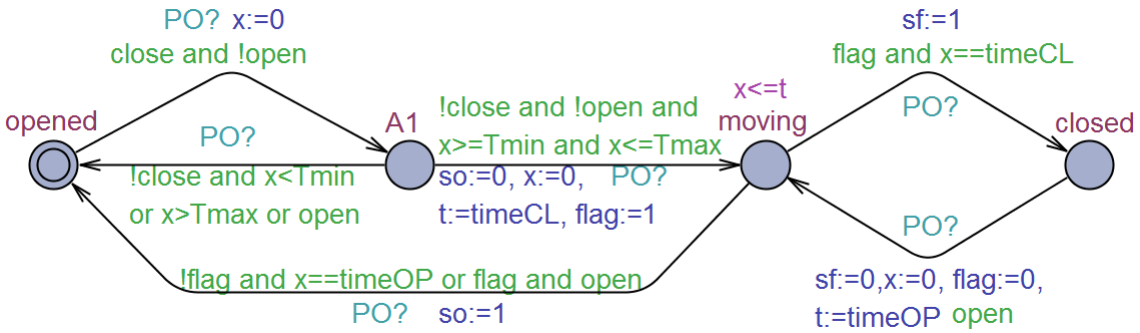


FIGURE 41 – Modèle Uppaal de l'appareil type 2

Le modèle Uppaal de l'appareil type 2 (figure 41) se présente avec 4 états. Pour choisir la position initial de l'appareil, il suffit d'étiqueter la location **opened** ou **closed** avec le label « initial ». Le comportement du modèle respecte le fonctionnement décrit par le chronogramme de la figure 38.

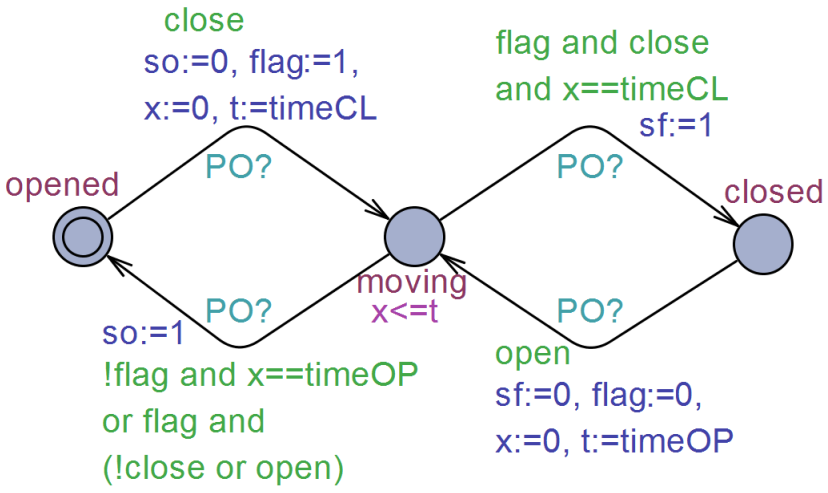


FIGURE 42 – Modèle Uppaal de l'appareil type 3

Les modèles Uppaal des appareils types 3 (figure 42) et 4 (figure 43) ne renferment que 3 états, car leur fonctionnement est plus simple.

Ces 4 types d'appareil constituent l'ensemble des appareils électriques (disjoncteurs, sectionneurs, et interrupteurs) qu'on retrouve dans une installation électrique. Il suffira d'instancier ces appareils pour constituer une partie du modèle de l'installation électrique, en renseignant les arguments suivants :

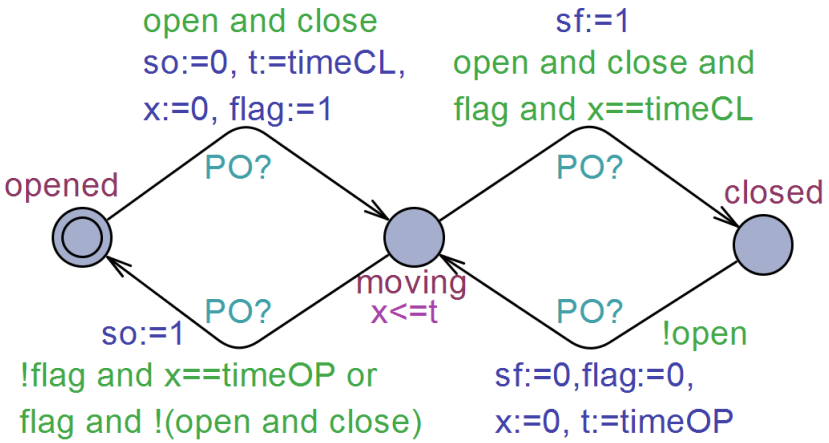


FIGURE 43 – Modèle Uppaal de l'appareil type 4

— son nom,

— ses paramètres **TimeOP**, **TimeCL**, et éventuellement **Tmin** et **Tmax**,

— le programme de commande (bloc fonctionnel) auquel il est rattaché.

En plus de ces 4 types d'appareils, une installation électrique renferme également des transformateurs et redresseurs.

2.3.2 Modélisation des transformateurs et redresseurs

Ces équipements font partie des sous-ensembles de partie opérative les plus fréquemment utilisés dans une installation électrique. Ils permettent respectivement de transformer et de redresser la tension qu'ils reçoivent en entrée. Le système de contrôle commande surveille en temps réel les informations booléennes qu'ils lui transmettent, telles que les défauts de types surintensité, absence de tension, hausse de température...

Contrairement aux 4 types d'appareils précédents, ces équipements ont un comportement d'ordre physique (thermique, magnétique...) qui ne nécessite pas une modélisation, car seuls les données booléennes échangées avec le système de contrôle commande nous intéressent. Les transformateurs et redresseurs sont alors considérés comme des structures (ou tableaux) de défauts pouvant être activés ou désactivés lors de la vérification. La figure 44 présente un exemple de modélisation et d'instanciation de deux transformateurs **Transfo1** et **Transfo2** surveillant les défauts « nfDA », « defBLQ », « blocDef »... Le nombre de défauts observés peut varier d'une installation à l'autre.

Il existe également quelques rares équipements (non présentés) qu'on peut retrouver dans une installation électrique. L'ensemble des éléments de partie opérative a été mo-

```
struct {bool nfDA, defBLQ, blocDef, defTemp, wdMicom, TcMC, temp2TR, AMR,
        fuFuRC, AvDiode, imaxGT, OoAbsUHT, OoDjAbsUHT, deblocGT,
        absUBarre, CDU, presenceSGT, pres_Hexa, defauti;} Transfo1, Transfo2;
```

FIGURE 44 – Exemple de modélisation et d'instanciation de 2 transformateurs

délisé et stocké dans une base de données. Celle-ci est réutilisable pour toute nouvelle installation.

2.4 Modélisation des programmes API

Les 3 principaux langages utilisés par les chargés d'études de la SNCF pour la programmation des EALE sont le LD, le SFC, et le ST (61131-3, 2013). Une traduction de ces programmes en langage formel compréhensible par Uppaal est nécessaire pour la vérification. De plus, ces programmes de commande utilisent des temporisateurs normalisés du type TON (Timer On-Delay). Ces temporisateurs doivent également être modélisés sous Uppaal. La section 2.4.1 présente la modélisation des temporisateurs. Ensuite la modélisation des programmes SFC est détaillée à la section 2.4.2, puis les modèles des programmes ST et LD sont décrits à la section 2.4.3.

2.4.1 Modélisation des timers (temporisateurs)

Ces blocs temporisateurs normalisés (TON) sont principalement constitués de :

— deux paramètres d'entrées **in** et **PT** : le premier étant un booléen qui démarre ou arrête la temporisation, et le second paramètre indique la durée de la temporisation prédéfinie,

— deux paramètres de sorties **Q** et **x** : le premier est mis à 1 par le bloc lorsque la temporisation est terminée, et le second retourne la valeur en cours du timer.

Le comportement d'un timer TON est donné par le diagramme de la figure 45 (norme IEC 61131-3 (2013)). La temporisation démarre lorsque la variable **in** passe de 0 à 1. Ensuite, la valeur du timer évolue jusqu'à atteindre la durée **PT** prédéfinie dans le timer. A ce moment, la sortie **Q** passe de 0 à 1 et y demeure tant que l'entrée **in** reste maintenue à 1.

Afin de modéliser le comportement du timer TON dans le Model-Checker Uppaal, nous utilisons un automate à trois états dont l'évolution est synchronisée avec le message **TON !** envoyé par le modèle du cycle automate (figure 35). L'état **Idle** correspond à l'état d'inactivité du timer dans lequel aucune variable n'évolue. Le deuxième état **running** cor-

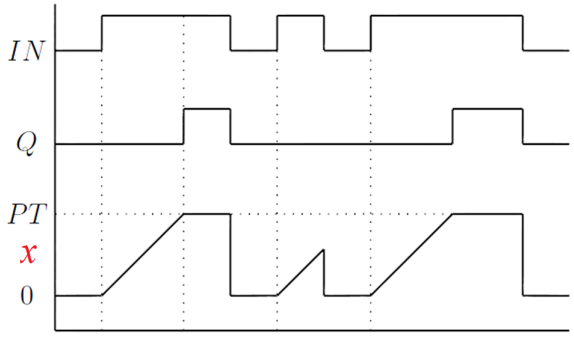


FIGURE 45 – Chronogramme du comportement d'un timer TON

respond à l'instant qui suit le déclenchement du Timer et avant la fin de la temporisation. Lorsque l'entrée **in** est activée, la sortie Q et l'horloge x sont initialisées à 0 avant que le timer ne passe à l'état **running**. Lorsque la valeur de la temporisation est atteinte tout en ayant l'entrée **in** activée, le timer passe à l'état **timeout** qui correspond à la fin de la temporisation, et y restera tant que l'entrée **in** n'est pas réinitialisée à 0. Le modèle de ce temporisateur est inspiré de Mokadem et al. (2010).

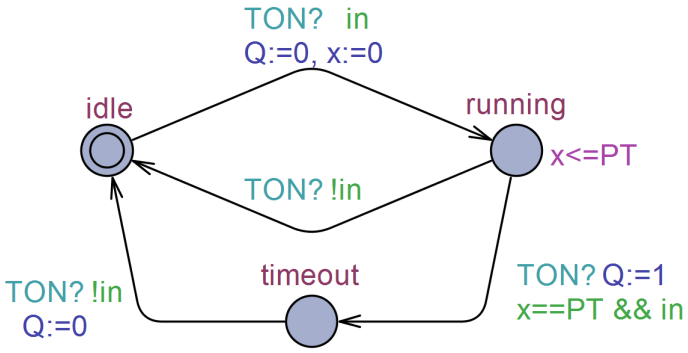


FIGURE 46 – Modélisation d'un Timer TON

Il arrive souvent qu'un programme SFC utilise plus de deux 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 de complexifier le modèle global et de l'exposer à des explosions combinatoires lors de la vérification des propriétés. 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. Cela est possible lorsque le programme SFC ne renferme pas de « divergence en ET » au niveau de ses transitions, car cela signifierait que ce programme ne peut pas avoir plus d'une étape activée en même temps. Toutefois lorsque deux étapes successives du programme SFC utilisent la même instance du 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 temporisation à la deuxième étape, car la sortie « **Q** » (maintenue activée à l'issue de la première temporisation) validerait automatiquement la transition de la deuxième étape.

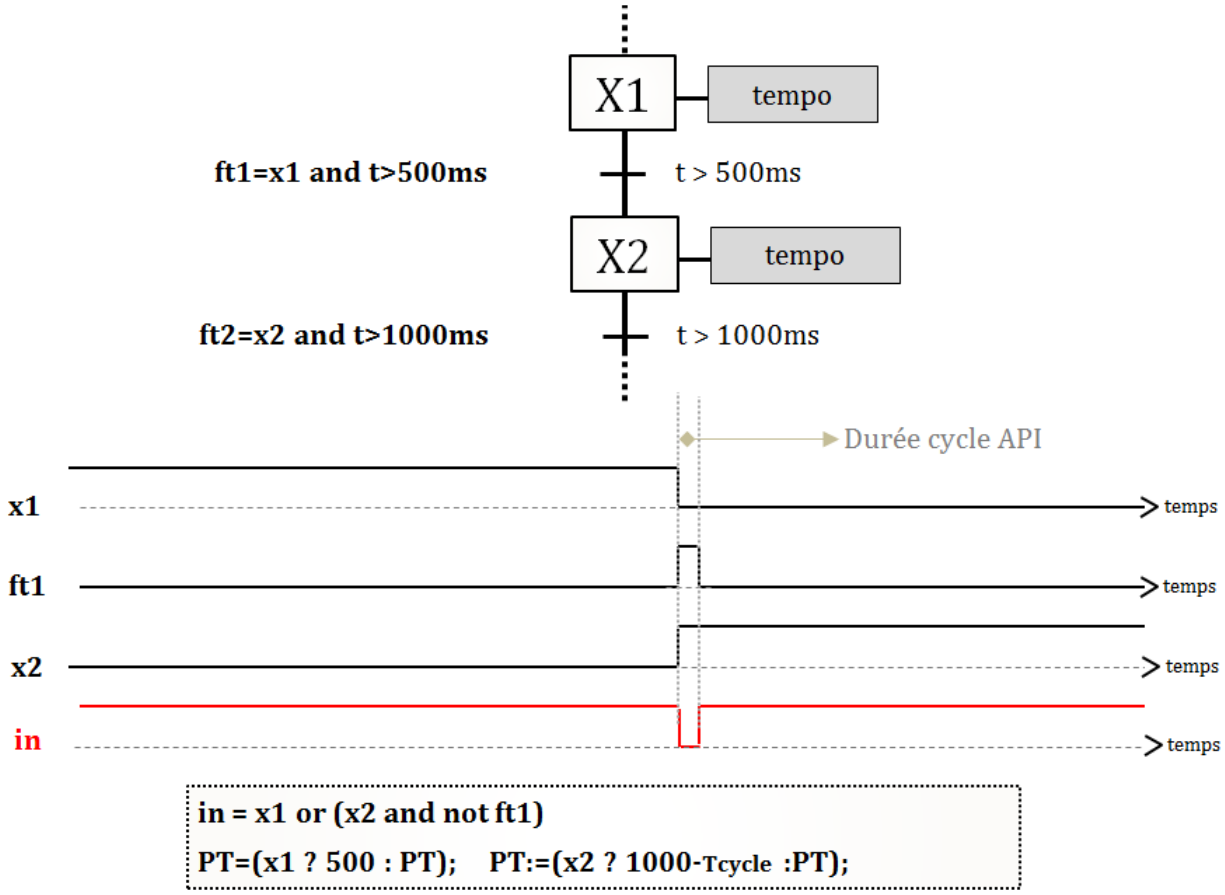


FIGURE 47 – Exemple d'utilisation du bloc TON dans un programme SFC

La figure 47 illustre le cas précédent : 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**), cette entrée reste égale à **true** 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 du paramètre **in** pour obtenir « **in** = **x1** or (**x2 and not ft1**) ». De ce fait lorsque l'étape **X1** est activée (PT=500ms, voir figure 47), la temporisation démarre jusqu'à atteindre les 500 unités de temps 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=1000 unités de temps). 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 entraîne 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 du temporisateur 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 47). Pour y remédier, il suffit de retrancher la durée du cycle automate (**Tcycle**) à la temporisation associée à l'étape **X2** (1000-Tcycle).

2.4.2 Modélisation des blocs fonctionnels SFC

Pour respecter la structure des programmes API originaux, les blocs fonctionnels doivent être créés puis instanciés dans le programme principal. En ce qui concerne le langage de formalisation de ces programmes, le choix des automates temporisés (à l'instar de Bauer et al. (2004)) présente quelques inconvénients :

— complexité de la représentation des divergences/convergences en OU/ET au niveau des transitions, modélisation des actions ...,
— modèles complexes pour des programmes SFC de taille importante,
— difficulté d'instanciation de plusieurs programmes SFC dans le programme principal.

Nous proposons dans ce cas une autre approche qui consiste à traduire les blocs fonctionnels SFC dans un formalisme d'équations logiques, grâce à la norme de représentation algébrique des programmes SFC (Machado et al., 2006) basée sur l'auto-maintien. Cette technique consiste à écrire sous forme d'équations logiques, les étapes, les transitions, et les actions du programme SFC, puis à assembler le tout dans une seule et même fonction.

La figure 48 présente un exemple assez simple d'un bloc fonctionnel SFC dédié à la commande d'un appareil type 3 (figure 42), et sa traduction en équations logiques sous Uppaal est donnée par la figure 49.

Ce programme SFC sous Uppaal est rattaché à une instance d'un temporisateur TON (figure 46) pour gérer la temporisation des étapes du programme SFC. La version traduite de ce programme sous Uppaal renferme 4 parties :

— calcul des fonctions de transition,
— mise à jour des étapes du programme SFC,
— calcul des entrées/sorties du bloc TON,
— calcul des sorties (actionneurs).

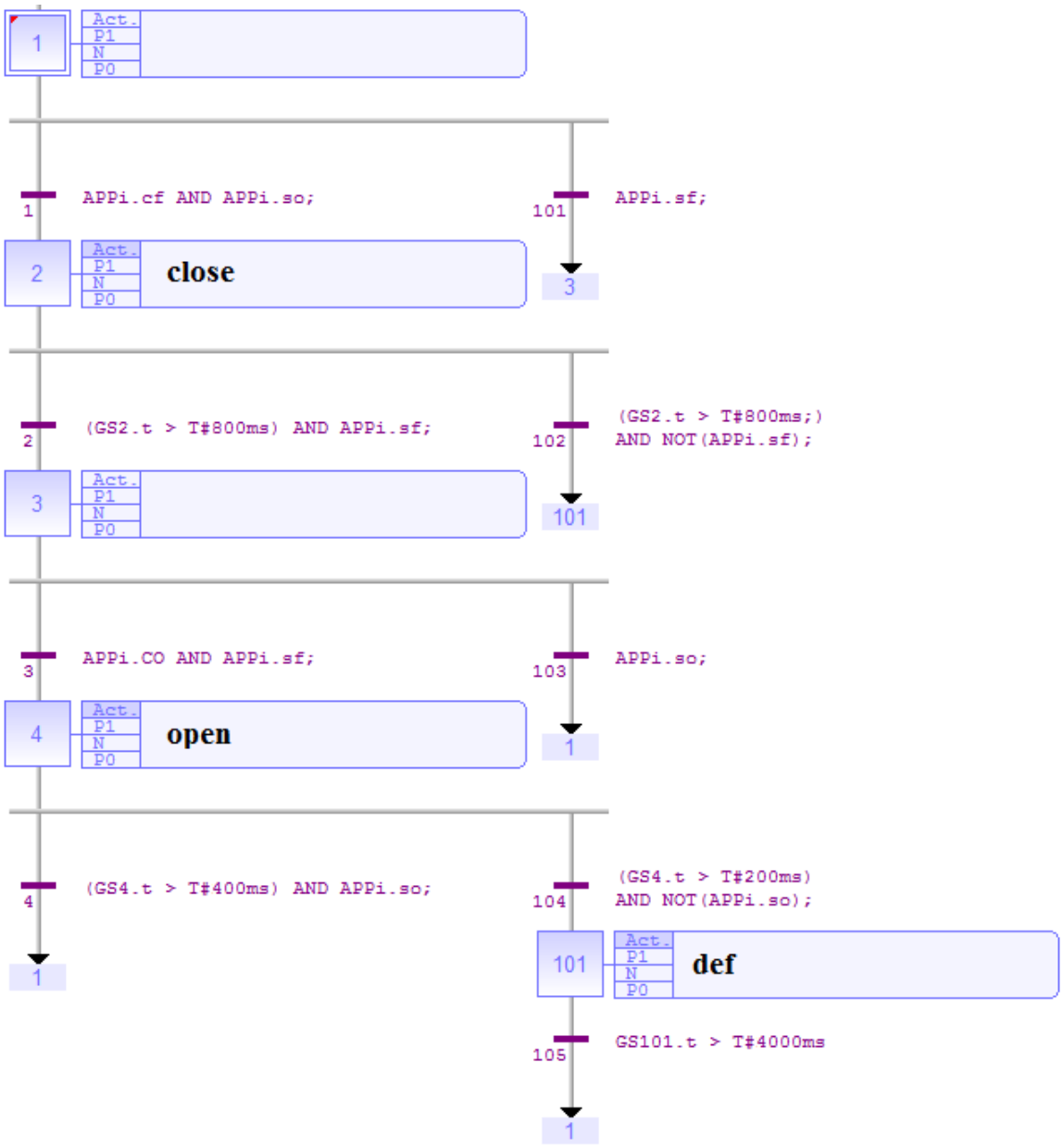


FIGURE 48 – Programme de commande d'un appareil type 3

L'avantage supplémentaire du Model-Checker Uppaal, à la différence des autres outils, est qu'il permet de créer et d'instancier des fonctions, ce qui convient parfaitement à notre approche de modélisation des programmes SFC.

2.4.3 Modélisation des programmes LD et ST

En complément des blocs fonctionnels SFC, il peut exister dans le programme principal des sections de code en LD ou ST. Le langage de traduction pour ces types de programmes reste le même que pour les blocs fonctionnels SFC.

La figure 50 présente un exemple de programme écrit en LD, et sa traduction sous

```
void CmdAPPi( bool &x1, bool &x2, bool &x3, bool &x4, // déclaration des arguments
              bool &x101, bool &so, bool &sf, bool &CO, bool &CF,
              bool &open, bool &close, bool &def, bool &intmp, bool &Qtmp, int &PT)
{
//déclaration des variables ft (fonctions de transition)
     bool ft1, ft2, ft3, ft4, ft101, ft102, ft103, ft104, ft105;

//équations des fonctions de transition
     ft1   = x1 and so and CF;
     ft2   = x2 and sf and Qtmp;
     ft3   = x3 and sf and CO;
     ft4   = x4 and so and Qtmp;
     ft101 = x1 and sf;
     ft102 = x2 and !sf and Qtmp;
     ft103 = x3 and so;
     ft104 = x4 and !so and Qtmp;
     ft105 = x101 and Qtmp;

//équations des étapes du programme SFC
     x1   = ft4 or ft105 or ft103 or x1 and !ft1 and !ft101;
     x2   = ft1 or x2 and !ft2 and !ft102;
     x3   = ft2 or x3 and !ft3 and !ft103;
     x4   = ft3 or x4 and !ft4 and !ft104;
     x101 = ft104 or ft102 or x101 and !ft105;

//Appel bloc Timer pour la temporisation des étapes X2, X4 et X101
//intmp et PT représentent les paramètres d'entrées du temporisateur
//Qtmp représente la sortie du temporisateur
     intmp = x2 or x4 or (x101 and !ft104);
     PT=(x2 ?80:PT);  PT:=(x4 ?40:PT); PT:=(x101 ? 400-20:PT);
//équations des actions
     close  = x2;
     open  = x4;
     def = x101;
}
```

FIGURE 49 – Modélisation du programme de commande d'un appareil type 3

Uppaal. Chaque rung est traduit en équation logique, l'ordre d'exécution de ces équations devra respecter celui des rungs du programme original (de haut en bas). Pour cela il suffit d'écrire les équations dans l'ordre, sans nécessité de programmer cet ordre d'exécution (à l'instar de Bender et al. (2008) pour des programmes LD modélisés en réseau de Petri). Pour les bobines Set et Reset, le formalisme suivant permet de convertir les rungs 2 et 3 en équations : « $\mathbf{A := (B\,?\,C : D)}$ » qui signifie : si **B** = true, alors **A** = **C**, sinon **A** = **D**. En ce qui concerne les programmes ST, ce langage est très proche du formalisme utilisé dans Uppaal à quelques syntaxes près.

Après la traduction des blocs fonctionnels à vérifier, il ne reste plus qu'à les instancier.

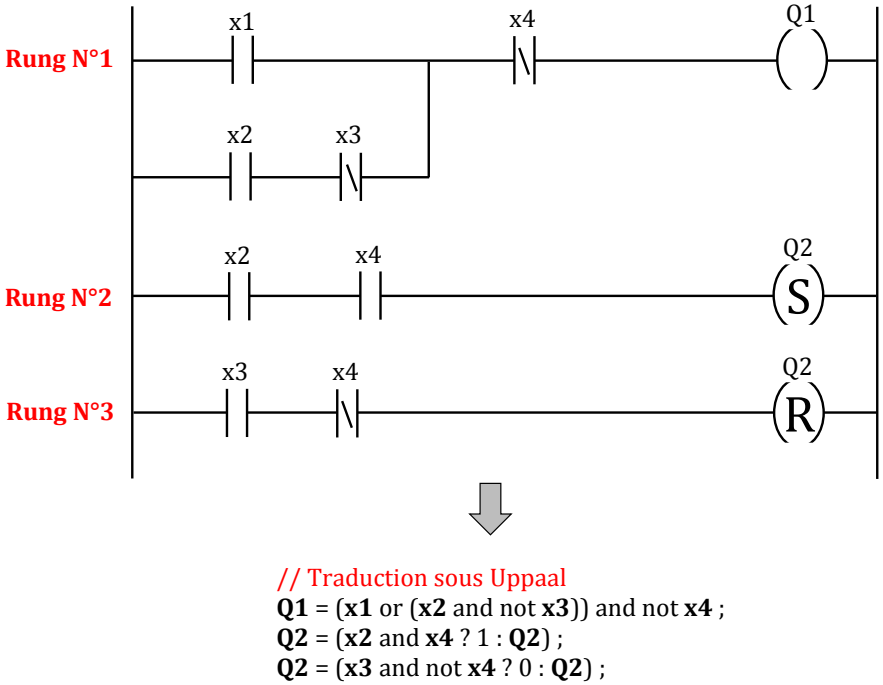


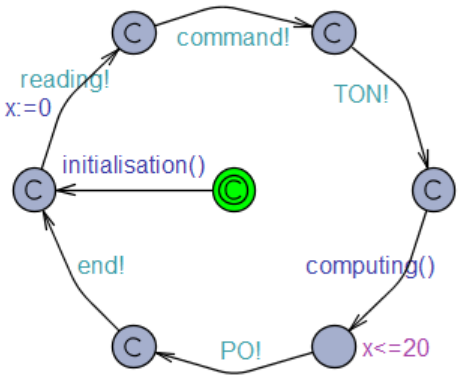
FIGURE 50 – Exemple de programme LD et sa traduction sous Uppaal

Un exemple de programme principal dans Uppaal est présenté à la figure 51. La fonction **initialisation()** est appelée une seule fois par le modèle du cycle API pour l'initialisation. Quant à la fonction **computing()**, elle est exécutée pendant chaque cycle API après la lecture des entrées et avant la mise à jour des sorties et de la partie opérative. Cette fonction **computing()** renferme dans cet exemple un programme LD et deux blocs fonctionnels SFC instanciés (dont le deuxième correspond à celui de la figure 49). L'instanciation permet en même temps de relier tout bloc fonctionnel SFC à l'appareil qu'il doit commander, à travers les arguments qui lui sont communiqués.

2.5 Formalisation des propriétés à vérifier

Après la modélisation de la partie opérative, du cycle automate et des programmes, il reste à vérifier formellement si le modèle global respecte certaines propriétés relatives au bon fonctionnement (cahier de recettes) et à la sûreté (états interdits) des programmes automates. Celles-ci doivent par conséquent être formalisées dans le Model-Checker Uppaal. Les sections 2.5.1 et 2.5.2 présentent respectivement la formalisation des spécifications fonctionnelles et des états interdits.

```
void initialisation()
{
  grafSGT1.tmpCO=60; grafSGT1.tmpCF=60;
  grafCtrCmdeDJGT1.tmpEnclDJGT=7;  GT1.tmpReencGT=5;
  grafCtrCmdeDJGT1.x1=grafSGT1.x1=1;  // initialisation des grafcets
  APP2.so=1; APP3.so=1; // initialisation des positions des appareils
  GT1.presenceSGT=1; GT1.pres_Hexa=0; GT1.OoDjAbsUHT=1;
}

void computing()
{
 // Programmes LADDER
    GT1.imaxGT :=(imaxGT ?1: GT1.imaxGT);
    GT1.imaxGT :=(!imaxGT and APP2.cfd ?0: GT1.imaxGT);

    GT1.blocDef:=(GT1.defBLQ ?1 : GT1.blocDef);
    GT1.blocDef:=(!GT1.defBLQ and GT1.deblocGT ?0 : GT1.blocDef);

    GT1.blocDefRed:=(GT1.defBLQR ?1 : GT1.blocDefRed);
    GT1.blocDefRed:=(!GT1.defBLQR and GT1.deblocGT ?0 : GT1.blocDefRed);

    GT1.OoAbsUHT = APP2.sf and GT1.absUBarre and GT1.OoDjAbsUHT;

    GT1.defBLQ := GT1.wdMicom or GT1.TcMC;
    GT1.defBLQR := GT1.AMR or GT1.fuFuRC or GT1.AvDiode;
    GT1.defTemp := GT1.temp2TR and false;

 // Programmes SFC instanciés
    CtrlCmdeDJGT (grafCtrCmdeDJGT1.x1, grafCtrCmdeDJGT1.x2, grafCtrCmdeDJGT1.x3,
                  grafCtrCmdeDJGT1.x4, grafCtrCmdeDJGT1.x5, grafCtrCmdeDJGT1.x6,
                  grafCtrCmdeDJGT1.x101, APP2.so, APP2.sf, APP2.cod, APP2.cfd,
                  APP2.open, APP2.close, GT1.blocDef, GT1.defTemp, GT1.blocDefRed,
                  GT1.defTempRed, faux, faux, GT1.pres_Hexa, GT1.imaxGT, APP3.so,
                  APP3.sf, GT1.presenceSGT, GT1.pres_Hexa,GT1.OoAbsUHT, GT1.CDU,
                  grafCtrCmdeDJGT1.OF, grafCtrCmdeDJGT1.Oo, APP2.def,in_TON2,
                  Q_TON2, PT2, grafCtrCmdeDJGT1.tmpEnclDJGT);

    CmdAPPi (grafSGT1.x1, grafSGT1.x2, grafSGT1.x3, grafSGT1.x4, grafSGT1.x101,
             APP3.so, APP3.sf, APP3.cod, APP3.cfd, APP3.open, APP3.close,APP3.def,
             in_TON3, Q_TON3, PT3);
}
```



Modèle du cycle API

FIGURE 51 – Exemple d'un programme principal dans Uppaal

2.5.1 Modélisation des fiches de recettes pour la vérification des spécifications fonctionnelles

Les fiches de recettes (ou fiches de tests) permettent de vérifier que le système de contrôle commande respecte les spécifications fonctionnelles. Chaque fiche de tests sert à vérifier et valider une partie du fonctionnement. Par exemple celle du tableau 4 est utilisée pour vérifier qu'un disjoncteur de Groupe Traction s'ouvre toujours instantanément suite à l'apparition d'un défaut court-circuit, et ne peut être refermé qu'après disparition et acquittement du défaut.

Il existe en moyenne plus d'une centaine de fiches de recettes, et leur contenu ne

varie pas d'une installation à l'autre. Selon la structure et la fonction d'une installation, le chargé d'études sélectionne depuis une base de données l'ensemble des fiches de tests nécessaires pour vérifier et valider l'aspect fonctionnel de la commande.

Afin d'utiliser ces mêmes fiches de tests pour la vérification formelle des programmes, il est nécessaire de les formaliser en un langage compréhensible par le Model-Checker Uppaal, ce qui nécessite une analyse structurelle.

Comme indiqué dans le chapitre 2 (section 2.2), une fiche de recettes est constituée d'une condition initiale (état initial du système), d'actions à réaliser et de résultats attendus (pour chaque action exécutée). Son mode d'exécution est séquentiel. Le chargé d'études vérifie d'abord si la condition initiale de la fiche est satisfaite, si tel est le cas, il exécute la première action et vérifie que le résultat attendu a été obtenu avant de pouvoir passer à l'action suivante. Il continue ainsi jusqu'à atteindre la dernière action de la fiche de tests. Ce principe de fonctionnement ressemble fort à celui d'un Grafcet, car il se base sur l'exécution séquentielle d'actions suivies de transitions. Partant de ce principe, chaque fiche de tests de la base de données peut être traduite sous forme de programme SFC. La figure 52 présente la traduction de la fiche de tests du tableau 4 en programme SFC.

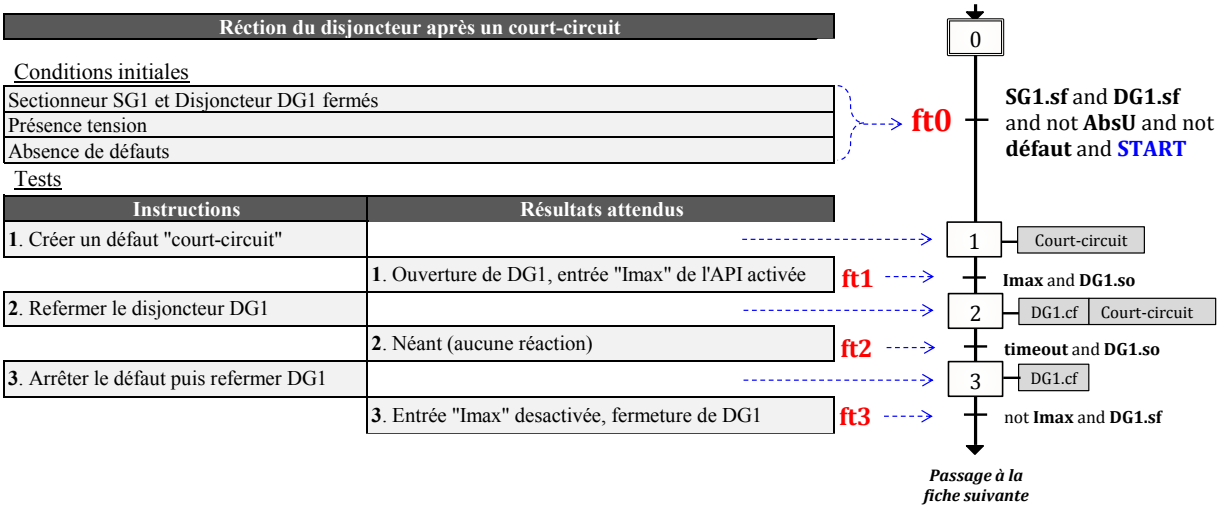


FIGURE 52 – Traduction d'une fiche de tests en programme SFC

Chaque instruction à exécuter est associée à une étape du programme SFC, et les résultats attendus sont associés aux transitions. L'exécution de la fiche de tests dans le Model-Checker se déroule alors de la manière suivante :

— Le programme SFC est à l'état initial (étape 0), l'exécution de la fiche de tests

ne peut débuter que si la condition initiale (i.e. la transition **ft0**) de la fiche est satisfaite et si la variable **START** est activée pour démarrer l'exécution de la fiche de tests ;

- À l'étape 1 du programme SFC, la variable « **Court-circuit** » est activée (instruction N^o1) pour créer le défaut court-circuit. Elle sera lue par le programme automate au cycle API suivant lors de la lecture des entrées.
- Le résultat attendu N^o1 est représenté par la transition **ft1** du programme SFC. L'apparition du défaut doit entraîner l'activation de la variable **Imax** et l'ouverture de l'appareil **DG1** (représenté par **DG1.so** = true).
- À l'étape 2, on tente de refermer le disjoncteur DG1 (**DG1.cf** = true) tout en maintenant le défaut **Court-circuit** activé.
- Le résultat attendu N^o2 (transition **ft2**) consiste à observer si au bout d'un certain temps, le disjoncteur DG1 reste toujours ouvert (représenté par **ft2** = **timeout** and **DG1.so**).
- À l'étape 3, on tente de refermer le disjoncteur DG1 en désactivant le défaut court-circuit.
- Cette dernière instruction doit entraîner (transition **ft3**) la désactivation de la variable **Imax** et la fermeture du disjoncteur DG1 (**DG1.sf)**.

Cette traduction d'une fiche de tests en programme SFC est applicable à l'ensemble des fiches de la base de données. Il ne reste alors qu'à convertir ce programme SFC en fonction à base d'équations logiques (figure 53) en utilisant la même règle décrite à la section 2.4.2.

Après avoir traduit toutes les fiches de tests en fonctions sous Uppaal, deux questions se posent :

- **Question 1** : Comment programmer l'exécution automatique d'une ou de plusieurs fiches de tests sur le système « Programme API + Partie Opérative » ?
- **Question 2** : Comment détecter automatiquement, si elle existe, l'instruction d'une fiche de tests quelconque qui n'a pas été satisfaite ?

Pour répondre à la question 1, il suffit de rajouter un nouveau modèle à base d'automates temporisés dans le modèle global. Ce modèle sélectionne les fiches de tests choisies par l'utilisateur, et les exécute dans l'ordre sur le système. Il est synchronisé avec le modèle de la figure 35 (grâce au message **end ?**), de manière à rafraîchir l'exécution des fiches de tests durant chaque cycle API. La figure 54 présente un exemple de modèle de cahier

```
void fich_imaxGT()
{
     ft0 = START and SG1.sf and DG1.sf     // condition initiale
           and not absU and not défaut;
     ft1 = x1 and Imax and DG1.so;         // résulat attendu N°1
     ft2 = x2 and timeout and DG1.so;      // résulat attendu N°2
     ft3 = x3 and not Imax and DG1.sf;     // résulat attendu N°3

     x1 = ft0 or x1 and not ft1;           // instruction N°1
     x2 = ft1 or x2 and not ft2;           // instruction N°2
     x3 = ft2 or x3 and not ft3;           // instruction N°3

  // Comptage du temps écoulé dans chaque étape
     start_elapsing = (x1 and not ft0) or
                      (x2 and not ft1) or
                      (x3 and not ft2);

  // programmation des actions
     Court-circuit = x1 or x2;
     DG1.cf = x2 or x3;

  // indicateur de l'état de la fiche (validée/boquante)
     ok = ft3;
}
```

FIGURE 53 – Traduction d'une fiche de tests en équations logiques

de recettes contenant 10 fiches de tests utilisées pour la vérification des spécifications fonctionnelles d'un ensemble de programmes SFC.

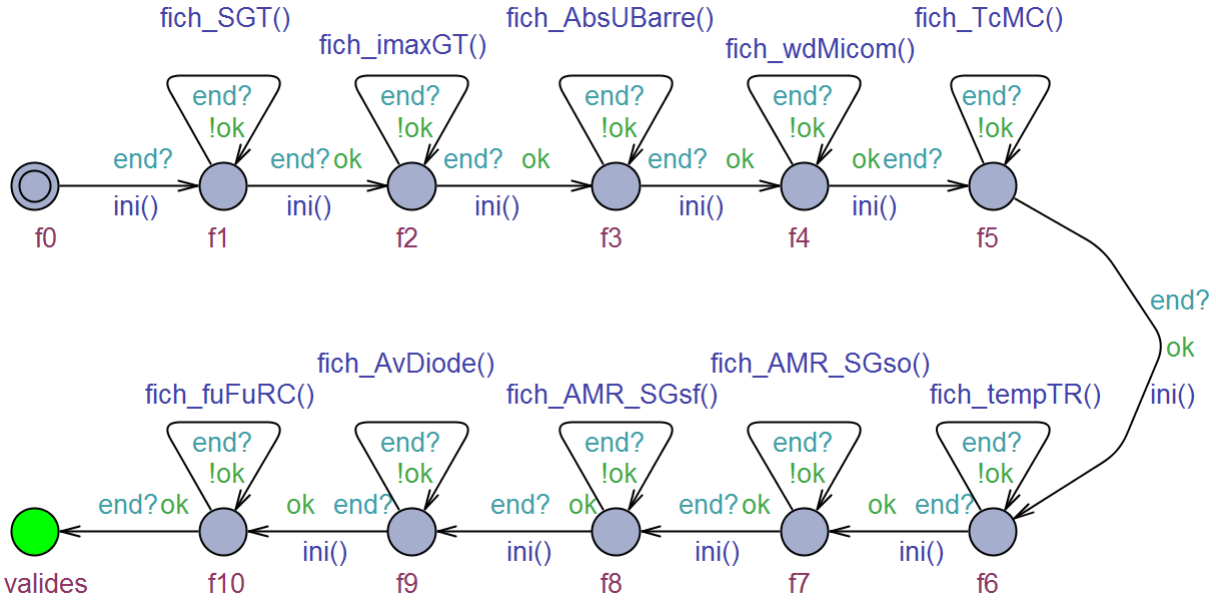


FIGURE 54 – Modèle synchronisant l'exécution des fiches de tests dans Uppaal

Ce modèle est constitué de 12 états : un état initial « **f0** », un état final « **valides** », et 10 états numérotés de « **f1** » à « **f10** ». Chacun de ces 10 états appelle et exécute dans l'ordre une fonction correspondant à une fiche de tests bien définie. Par exemple, la fiche

de tests de la figure 53 est exécutée à l'état **f2** du modèle du cahier de recettes. Ce modèle renferme également :

- une variable globale nommée **ok** qui indique à chaque instant l'état de la fiche en cours d'exécution (**ok = true** si fiche validée / **ok = false** si fiche pas encore validée),
- une fonction nommée **ini()** qui initialise le paramètre **ok** à **false** (avant chaque exécution d'une fiche) et les étapes du programme SFC de la fiche.

L'exécution séquentielle de ces fiches de recettes sur le système se déroule alors de la manière suivante :

- Initialement à l'état **f0**, le modèle du cahier de recettes (figure 54) évolue vers l'état **f1** à l'issue du 1er cycle API. Tant que le paramètre global **ok** est égal à **false**, la fonction associée à l'état **f1** (nommée « **fich_SGT()** ») est exécutée en boucle et rafraîchie à chaque cycle API.
- Lorsque le dernier résultat attendu de la fiche de tests « **fich_SGT()** » est satisfait, cela signifie que toutes les instructions de cette fiche ont été exécutées avec succès. Le paramètre **ok** prend la valeur **true**, et par conséquent le modèle du cahier de recettes évolue vers l'état **f2** (après avoir réinitialisé le paramètre **ok** à **false**) pour exécuter la fiche de tests suivante (test **fich_imaxGT()**).
- L'exécution séquentielle des fiches de tests se déroule alors de cette manière jusqu'à atteindre le dernier état du modèle du cahier de recettes (état « **valides** »), à condition qu'il n'y ait eu aucune instruction bloquante parmi toutes les fiches de tests.

Le modèle du cahier de recettes permet ainsi de répondre à la **Question 1** : « comment programmer l'exécution d'une ou de plusieurs fiches de tests sur l'ensemble Programme API + Partie Opérative ? ». À présent intéressons-nous à la **Question 2** : « comment détecter, si elle existe, l'instruction d'une fiche de tests quelconque qui n'a pas été satisfaite ? ». En effet, rien ne permet pour le moment à l'utilisateur de détecter en cas de problème, le nom de la fiche de tests et le numéro de l'instruction bloquante (i.e. qui n'a pas été satisfaite par le programme), sans quoi il ne peut faire de corrections nécessaires pour corriger l'anomalie. S'il se contente de vérifier l'atteignabilité de l'état « **valides** » du modèle du cahier de recettes dans Uppaal (figure 54) à travers la propriété en langage CTL :

$$E \prec\succ CahierDeRecettes.valides \tag{III.1}$$

Cela pourrait certes suffir à montrer que toutes les instructions ont été satisfaites, mais dans le cas contraire, cela reste insuffisant pour déterminer la fiche de tests et le numéro de l'instruction bloquante. Cette information doit nécessairement être retournée par le Model-Checker sous forme de trace à l'issue de la vérification.

Par définition, une instruction est dite bloquante si après son exécution sur le système par le modèle du cahier de recettes, le résultat attendu de cette instruction n'a pas été obtenu au bout d'un certain temps (temps qui ne peut être dépassé si le programme ne présente pas d'erreurs). Afin d'extraire cette information pour chaque instruction du cahier de recettes, il suffit alors de chronométrer le temps écoulé entre le début de l'exécution de cette instruction et l'obtention du résultat attendu. À cette fin, le cahier de recettes est munie d'un temporisateur (instance du timer TON de la figure 46) utilisé tour à tour par chacune des fiches de tests, pour mesurer ce temps écoulé (figure 55).

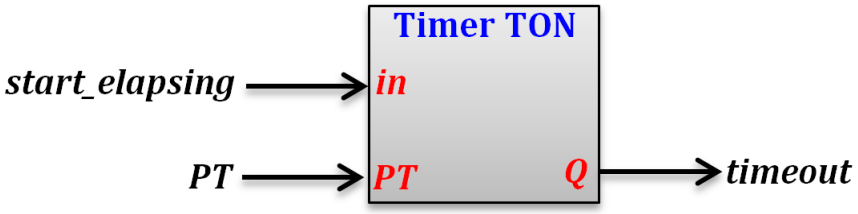


FIGURE 55 – Temporisation des instructions du cahier de recettes

La valeur du paramètre **PT** est choisie de telle sorte que ce temps de temporisation ne sera jamais dépassé par le temporisateur s'il n'y a pas d'erreurs dans le programme API. L'entrée « **start_elapsing** » du temporisateur du cahier de recettes (figure 55) correspond à une variable globale utilisée tour à tour par chaque fiche de tests en cours d'exécution. Par exemple, la fiche de tests de la figure 53 utilise (pendant son exécution) le temporisateur du cahier de recettes pour mesurer tour à tour le temps écoulé dans chacune de ses étapes **x1**, **x2**, et **x3**. Pour résumer, un seul temporisateur permet de mesurer le temps écoulé dans chacune des instructions du cahier de recettes. Cette optimisation permet de minimiser le nombre d'instances de temporisateurs afin d'éviter l'explosion combinatoire pendant la vérification.

La réponse à la **question 2** s'obtient en vérifiant dans le Model-Checker, la propriété suivante : « *Existe-t-il parmi ces fiches de tests, une instruction exécutée n'ayant pas abouti après un certain temps (PT) au résultat attendu ?* », ce qui se traduit en langage CTL

dans Uppaal par la propriété suivante :

$$E \prec\succ TempoCDR.timeout \tag{III.2}$$

Avec **TempoCDR**, l'instance du modèle de temporisation du cahier de recettes, et **timeout** l'état qui indique la fin de la temporisation (figure 46). Ainsi lorsque la propriété III.2 est satisfaite, le Model-Checker renverra sous forme de trace l'ensemble des fiches de tests qui ont été exécutées avec succès, jusqu'à la dernière instruction qui a été bloquante (si elle existe).

À noter que cette approche de vérification formelle ne consiste pas à vérifier « *s'il existe un chemin permettant d'exécuter toutes les instructions du cahier de recettes sans blocages* » (i.e. la propriété III.1), car dans ce cas rien ne prouverait qu'il n'existe pas d'autres chemins bloquants. Elle consiste plutôt à vérifier « *qu'il n'existe aucun chemin entraînant le blocage d'au moins une instruction d'une fiche de recettes quelconque* ». Cela montre que la méthode vérifie formellement que le fonctionnement décrit par le cahier de recettes sera toujours respecté par le programme API. Il reste toutefois à vérifier formellement la sûreté de fonctionnement de ces programmes.

2.5.2 Formalisation des états interdits pour la vérification de la sûreté de fonctionnement

Un état interdit représente une configuration dangereuse pour le système, capable d'engendrer des pertes importantes et irréversibles. Pour vérifier formellement qu'un programme API est sûr de fonctionnement (propriété 2), il suffit de prouver que « **quelle que soit la commande envoyée par l'opérateur** » et « **quel que soit l'état des entrées/sorties de l'installation** », celle-ci n'atteindra jamais des états interdits. Comme le montre la figure 30, le cahier de recettes ne permet d'explorer qu'une partie des évolutions possibles du système, il reste donc insuffisant pour vérifier la propriété 2. Le modèle du cahier de recettes (figure 54) ne sera alors pas utilisé pour la suite. En échange, on ajoute sur le modèle global du système, des modèles qui feront évoluer l'installation de telle sorte que durant chaque cycle automate, tout événement (du type apparition de défauts ou commandes opérateur) peut se produire sur le système. En effet, pour prouver la sûreté de fonctionnement des programmes, il est nécessaire de se positionner dans le pire des cas, c'est-à-dire le cas où les événements (défauts) se produisent de manière aléatoire

à tout moment. De cette manière, on couvre tout l'espace d'état du système.

Le modèle de la figure 56, synchronisé avec le modèle du cycle API (figure 35 à travers le message « **reading ?** » (lecture des entrées de l'API), permet d'affecter de manière aléatoire des valeurs **true** ou **false** à un défaut (représenté par l'argument « **b** » du modèle) pendant chaque cycle API. Ce modèle sera instancié pour chacun des défauts observés par le système.

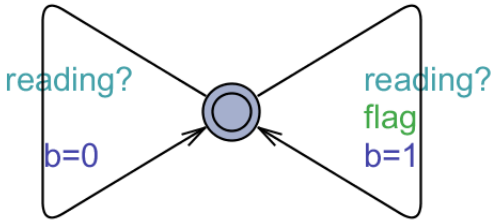


FIGURE 56 – Modèle générateur aléatoire de défaut

Quant au modèle de la figure 57, il permet d'envoyer aléatoirement à l'appareil auquel il est associé, une commande ouverture (co=1 et cf=0), fermeture (co=0 et cf=1), ou aucune des deux (co=0 et cf=0) au début de chaque cycle API. Il est alors synchronisé avec le modèle du cycle API à travers le message « **command ?** » (lecture des commandes). Ce modèle sera instancié autant de fois qu'il y a d'appareils à commander dans l'installation. Il est important de noter que les commandes d'ouverture/fermeture (**co/cf**) du modèle de la figure 57 sont différentes des commandes (**open/close**) reçues par les modèles de la partie opérative (exemple figure 40). Les commandes **co/cf** correspondent aux ordres envoyés par l'opérateur vers le système de contrôle commande. Dans notre modèle, ces commandes aléatoires co/cf sont lues uniquement par le modèle du cycle API. Tandis que les commandes **open/close** correspondent aux sorties calculées par le programme automate, et destinées à la partie opérative.

Pour des systèmes de taille importante avec plusieurs appareils commandés et défauts observés, il faut s'attendre à ce que l'espace d'états augmente avec ces deux derniers modèles. Pour minimiser cet impact, des conditions ont été ajoutées sur les transitions de ces modèles. Par exemple, la condition « **flag** » ajoutée sur le modèle de la figure 56 n'autorise l'activation du défaut concerné que lorsque celui-ci aura un impact sur le programme pendant le cycle API en cours, car dans le cas contraire cela n'aura servi à rien.

À présent nous allons présenter un exemple d'application de la méthodologie de vérification formelle sur un cas concret : les programmes d'asservissement entre disjoncteurs

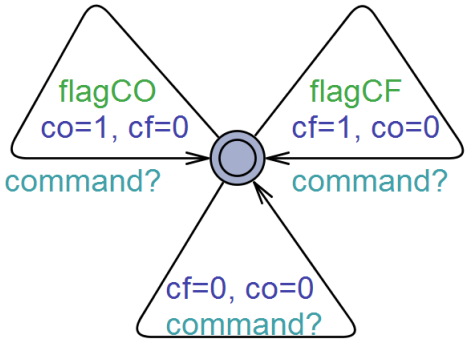


FIGURE 57 – Modèle générateur aléatoire de commandes appareil

de Départ Traction 1500V. Des nouveaux blocs fonctionnels ont été développés par les chargés d'études, et afin d'évaluer l'efficacité de notre approche, l'objectif de la section 3 est de vérifier formellement ces nouveaux programmes API.

3 Étude de cas : Asservissement Fil Pilote (FP) entre disjoncteurs de Départ Traction 1500V

3.1 Présentation du cahier des charges

L'asservissement est un principe qui permet de rendre les disjoncteurs d'un même secteur caténaire solidaires les uns des autres, c'est-à-dire que l'ouverture ou la fermeture du disjoncteur (maître) doit provoquer le même effet chez les autres disjoncteurs (esclaves) du même secteur. On dit alors que les disjoncteurs esclaves sont **asservis** au disjoncteur maître, d'où le nom **asservissement entre disjoncteurs**.

Prenons le secteur de la figure 58 alimenté par 2 sous-stations **SstA** et **SstB** par l'intermédiaire des disjoncteurs **DjA** et **DjB**. Ce dispositif permet par exemple, en cas de défaut détecté sur la caténaire par l'un des deux disjoncteurs du secteur, de faire ouvrir également l'autre disjoncteur de manière à interrompre l'alimentation du défaut, pour éviter la détérioration des équipements ou l'atteinte à la sécurité des personnes.

Techniquement, l'asservissement FP entre disjoncteurs consiste à faire circuler un courant dans une boucle de cuivre (voir figure 58) constituée de 2 ou plusieurs modules d'asservissement connectés en série, chaque module d'asservissement étant relié à un disjoncteur. Un module d'asservissement dispose d'une source d'alimentation (qui peut être shuntée ou non en fonction de l'état du circuit du module) et d'un relais qui a pour

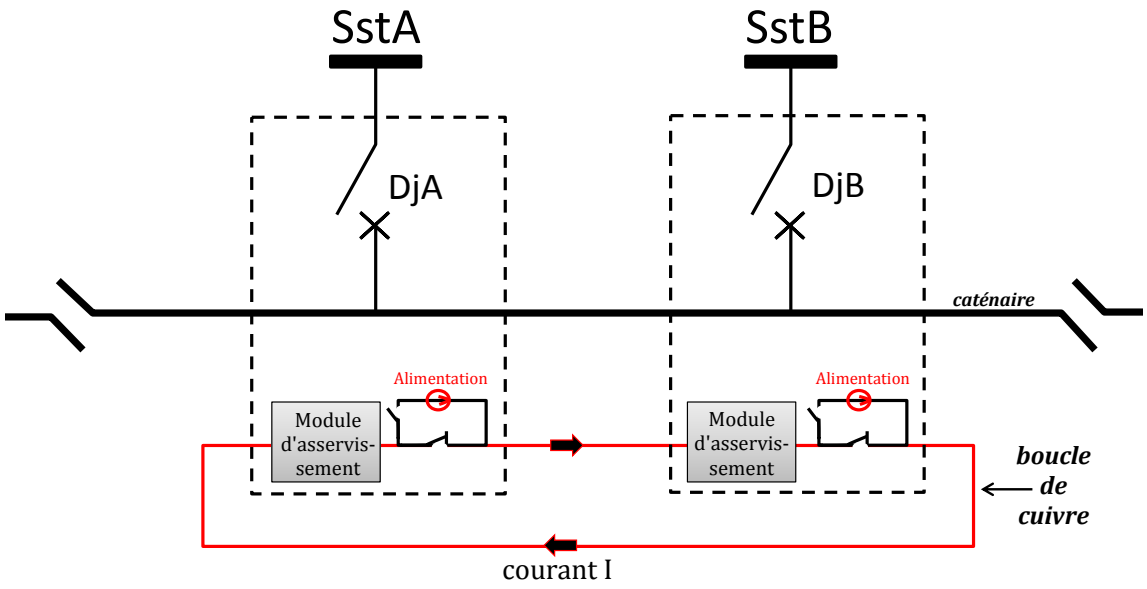


FIGURE 58 – Principe de l'asservissement FP entre deux disjoncteurs

fonction de détecter la présence d'un courant I dans la boucle. Le circuit de shuntage de l'alimentation d'un module d'asservissement ainsi que les entrées/sorties du module sont présentés à la figure 59.

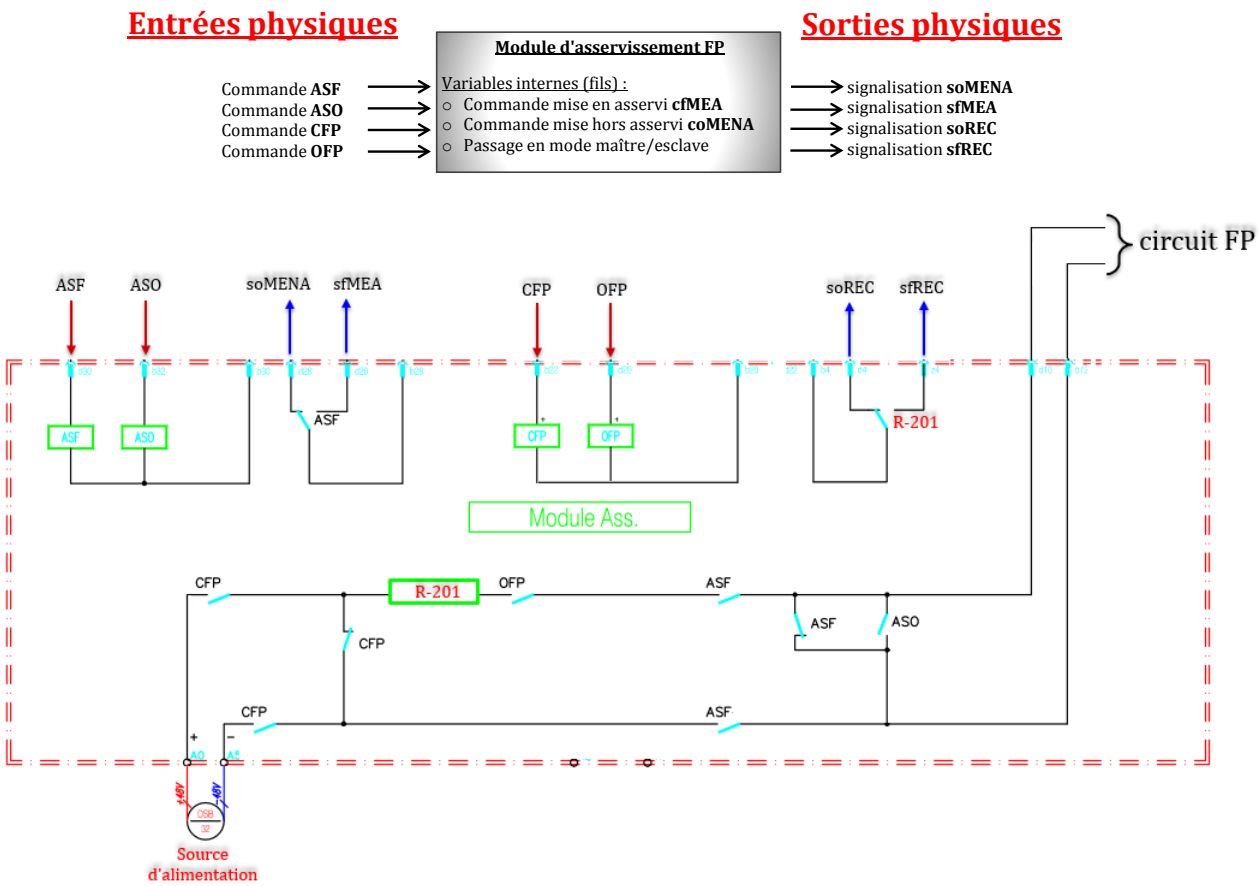


FIGURE 59 – Module d'asservissement

Un module d'asservissement se présente sous la forme d'un schéma à base de relais

(ASF, ASO, CFP, OFP, et R-201) et de contacts monostables ayant chacun une position de repos ouverte ou fermée comme le montre la figure 59. Ces contacts monostables connectés entre eux (l'alimentation y compris) constituent le circuit du module d'asservissement qu'on appellera **circuit FP** (voir figure 59). L'interconnexion des modules d'asservissement FP se fait donc à travers leur circuit FP (figure 58).

La présence du courant dans la boucle de cuivre indique que les disjoncteurs sont fermés et asservis entre eux. En revanche la perte du courant dans la boucle provient de l'ouverture d'un des disjoncteurs du secteur, ce qui entraîne l'ouverture des autres disjoncteurs. La source du courant provient uniquement du module du disjoncteur maître du secteur, les esclaves n'étant que des récepteurs dont les sources d'alimentation sont shuntées. Le statut « maître » ou « esclave » d'un disjoncteur peut changer pendant le fonctionnement à la demande de l'opérateur, et il ne doit jamais y avoir plus d'un « disjoncteur maître » à la fois dans la même boucle, car cela pourrait créer une surcharge de courant et endommager les modules du circuit. De plus, il doit toujours y avoir un disjoncteur maître dans le secteur. Un disjoncteur peut être également en mode **asservi** ou **non asservi** (c'est-à-dire qu'il est solitaire et fonctionne indépendamment des autres disjoncteurs du secteur).

Chaque module d'asservissement renferme 4 entrées et 4 sorties connectées aux E/S de l'automate. Les entrées du module (**ASF**, **ASO**, **CFP**, et **OFP**) correspondent aux entrées servant à exciter les relais respectifs portant le même nom (voir figure 59). Quant aux 4 sorties **soMENA**, **sfMEA**, **soREC**, et **sfREC**, elles représentent respectivement les signalisations « DJ en mode non asservi », « DJ en mode asservi », « absence de courant », et « présence de courant dans la boucle FP ». L'automatisation de la fonction d'asservissement est assurée par 2 blocs fonctionnels SFC (pour chaque disjoncteur) dans le programme automate principal.

Ces nouveaux programmes (créés pour remplacer les anciennes versions) ont été implémentés et testés une fois en usine, mais leur vérification (avec l'exécution de fiches de recettes par les chargés d'études) fut un échec car ces programmes présentaient encore des défauts. Étant contraints par la deadline du projet, les chargés d'études ont décidé d'intégrer les anciennes versions de programmes pour éviter de perdre davantage de temps sur la correction de ces nouveaux programmes en usine.

Pour évaluer l'efficacité de l'approche de vérification formelle, nous l'appliquons à ces nouveaux programmes afin de vérifier s'ils respectent ou non les spécifications fonc-

tionnelles et les propriétés de sécurité. Cela a permis d'établir un rapport de correction détaillant l'ensemble des erreurs détectées, avec proposition de correction. La prochaine étape consiste à modéliser le système (programmes API + PO) et à formaliser les propriétés à vérifier.

3.2 Modélisation du système

D'après le schéma de principe de l'asservissement (figure 58), la partie opérative est uniquement constituée de deux disjoncteurs (appareil type 2, voir figure 41) et de deux modules d'asservissement connectés en série. Un module d'asservissement est modélisé en équations logiques, chacune des 4 sorties est exprimée sous forme d'équations en fonction des 5 relais du circuit FP (voir annexe A). La première étape consiste alors à instancier les bons éléments depuis la base de données des modèles de sous-ensembles de PO.

Quant aux programmes API, les 2 blocs fonctionnels SFC à vérifier (voir annexe A) ont été traduits et instanciés dans le programme principal pour chaque disjoncteur. Deux fiches de tests sont nécessaires pour valider le fonctionnement des programmes d'asservissement, elles sont présentées en annexe A avec leur modélisation sous Uppaal. Enfin, en ce qui concerne la sûreté de fonctionnement, deux états interdits, extraits du cahier des charges, sont présentés ci-dessous :

— Surcharge dans la boucle de courant, dû à la présence de deux disjoncteurs maîtres à la fois ;

— Les deux disjoncteurs fermés et asservis, sans qu'il n'y ait présence de courant dans la boucle FP.

3.3 Vérification des blocs fonctionnels SFC

3.3.1 Vérification des spécifications fonctionnelles

La figure 60 présente le modèle du cahier de recettes nécessaire pour valider le fonctionnement des nouveaux programmes d'asservissement FP. Il permet d'exécuter deux fiches de recettes aux états **f1** et **f3** (l'état **f2** permet d'initialiser la partie opérative pour préparer l'exécution de la fiche associée à **f3**). Les modèles des fiches de tests sont présentés en annexe A.

La vérification de la propriété III.1 dans Uppaal a été faite sur un ordinateur avec 4Gb de RAM Core i5. Le résultat de la première vérification est présenté à la figure 61.

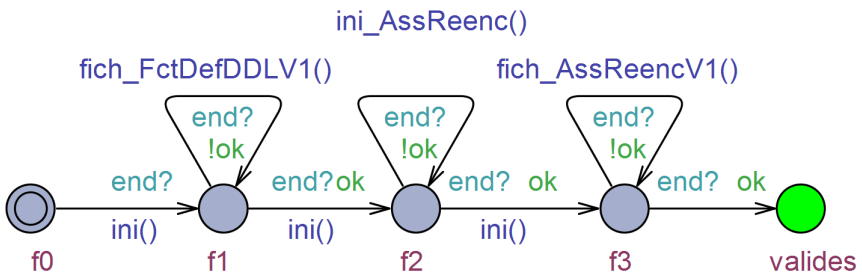


FIGURE 60 – Modèle cahier de recettes pour l'asservissement FP

Comme on peut le constater, la propriété III.1 est satisfaite par les programmes API car l'instance du temporisateur « **CahierDeRecettes** » a bien atteint l'état « **timeout** ». La simulation a duré 0,593s pour 13Mb de mémoire RAM exploitée.

Ce premier résultat confirme donc que les blocs fonctionnels ne respectent pas une partie des spécifications fonctionnelles. En effet, les fiches de tests **f1** et **f2** ont été exécutées avec succès, mais la simulation s'est arrêtée à la fiche **f3** du modèle du cahier de recettes (voir figure 60). La liste des variables d'environnement permet de voir que c'est l'instruction N^o1 qui n'a pas été satisfaite 61. Celle-ci consistait simplement à faire passer le disjoncteur **DjA** en mode asservi, et à observer que le changement de mode a bien eu lieu sans entraîner aucun autre impact sur le système, ce qui n'a pas été le cas. La séquence suivante extraite de la trace de la simulation a permis de diagnostiquer et corriger l'anomalie :

- Condition initiale de la fiche **f3** : les deux disjoncteurs **DjA** et **DjB** sont fermés, mais **DjA** est non asservi.
- Instruction N^o1 : lorsqu'on tente de mettre **DjA** en mode asservi, le programme d'asservissement de ce disjoncteur coupe alors momentanément le circuit de la boucle de cuivre lors de cette manoeuvre, ce qui provoque l'absence de courant dans la boucle FP (figure 58) pendant un court instant.
- Le circuit FP étant coupé, le courant n'est plus détecté par le disjoncteur **DjB**, celui-ci s'ouvre alors instantanément.
- Le disjoncteur **DjA**, étant fermé et maintenant asservi à **DjB**, s'ouvre également.
- Finalement, cette action a entraîné le passage de **DjA** en mode **asservi** et l'ouverture des 2 disjoncteurs (voir figure 61), ce qui ne correspond pas au résultat attendu par le cahier de recettes.

Le diagnostic du défaut consiste à analyser cette trace, celle-ci permet de suivre l'évolution des modèles, des programmes SFC, du cahier de recettes, des variables... Après avoir

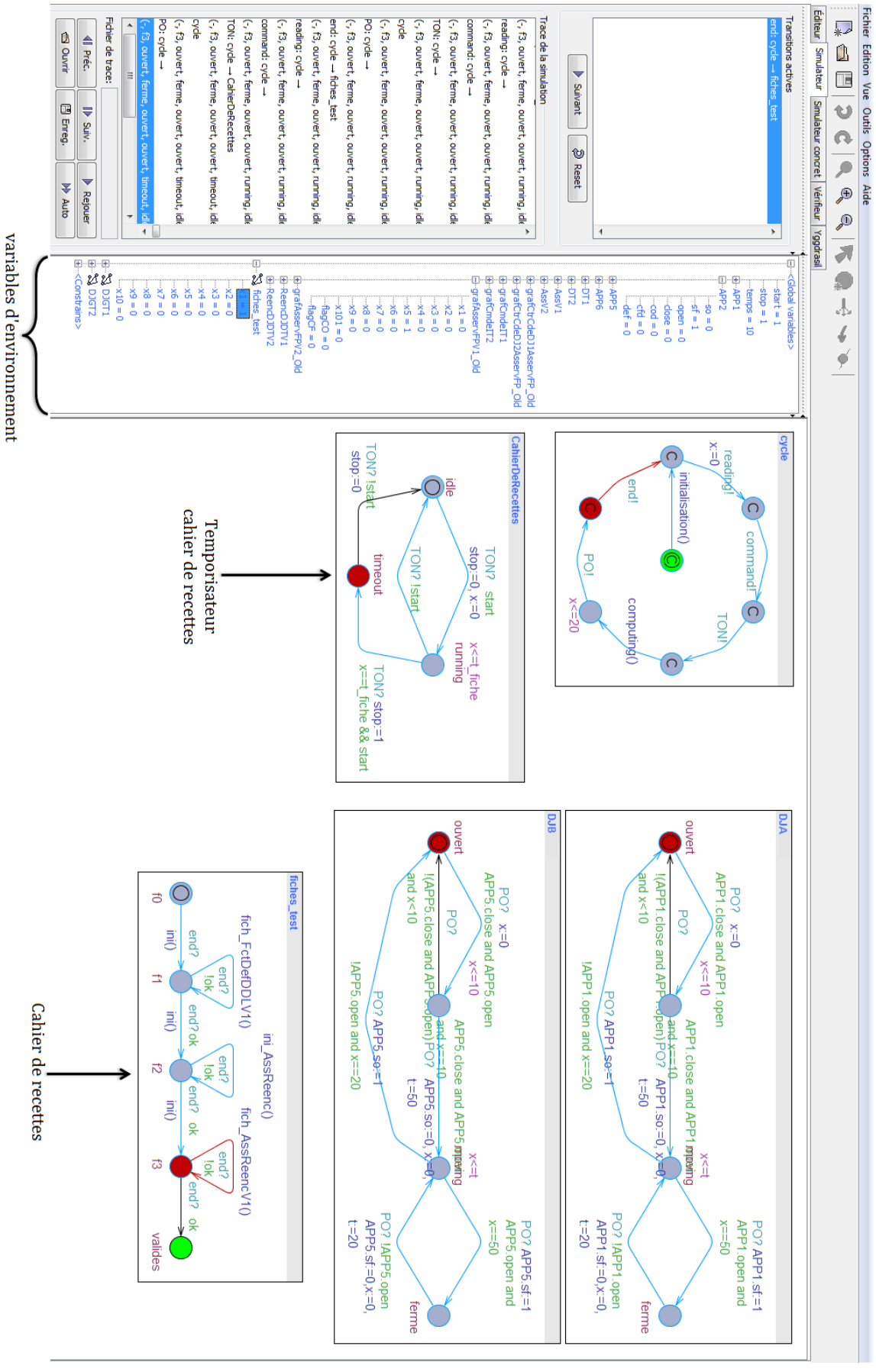


FIGURE 61 – Résultat de la première vérification des programmes

corrigé l'anomalie (qui provenait de la coupure anormale de courant dans la boucle FP lors du passage de **DjA** en mode asservi), la vérification est relancée de manière à confirmer la validité de la correction apportée, et à rechercher d'autres instructions bloquantes si elles existent. Suivant cette approche, un compte rendu de corrections apportées sur ces programmes d'asservissement a été établi. Ces corrections ont permis d'obtenir une nouvelle version de programmes qui respectent les spécifications fonctionnelles. Ce compte rendu est disponible en annexe B.

3.3.2 Vérification de la sûreté de fonctionnement

Les programmes API respectent à présent les spécifications fonctionnelles. En ce qui concerne la sûreté de fonctionnement, le principe consiste à vérifier de manière exhaustive l'atteignabilité des deux propriétés suivantes formalisées en CTL :

Propriété A : « **Existe-t-il une situation de surcharge de la boucle de courant, due à la présence de deux disjoncteurs maîtres à la fois ?** » : *E <> AssV1.OFP* ***and*** *AssV1.ASF* ***and not*** *AssV1.ASO* ***and*** *DT1.preUV1* ***and*** *AssV1.CFP* ***and*** *(AssV2.ASF* ***and*** *AssV2.OFP* ***and not*** *AssV2.ASO* ***and*** *DT2.preUV2* ***and*** *AssV2.CFP)*

Propriété B : « **Existe-t-il un cas où les deux disjoncteurs sont fermés et asservis, sans qu'il n'y ait présence de courant dans la boucle FP fermée ?** » : *E<> AssV1.OFP* ***and*** *AssV1.ASF* ***and*** *not AssV1.ASO* ***and*** *DT1.preUV1* ***and not*** *AssV1.CFP* ***and*** *(AssV2.ASF* ***and*** *AssV2.OFP* ***and not*** *AssV2.ASO* ***and*** *DT2.preUV2* ***and not*** *AssV2.CFP)* ***and*** *APP1.sf* ***and*** *APP5.sf*

La vérification exhaustive a montré que la propriété 1 (ayant duré 199,59s et nécessité 453Mb de RAM) et la propriété 2 (33,4s et 157Mb de RAM) sont satisfaites par les programmes API. On peut noter que les temps de simulation ainsi que la mémoire utilisée dépassent largement les ressources utilisées lors de la vérification des spécifications fonctionnelles, ce qui est normal car l'espace d'états est plus vaste. Les traces retournées par le Model-Checker ont permis d'analyser les scénarios ayant conduit vers ces états interdits (annexe B). Par exemple, la propriété A est satisfaite dans les conditions suivantes : lorsque les deux disjoncteurs sont ouverts et asservis entre eux et qu'ils se ferment au même instant **t** (durant un même cycle API), ils tenteront tous les deux de passer en mode « maître » et d'alimenter la boucle de courant, ce qui provoque la surcharge. Toutefois ces scénarios ne peuvent se produire que très rarement, car dans la pratique l'opérateur

n'a pas la possibilité de fermer simultanément deux appareils au même moment. Mais formellement cela peut représenter un danger, ces types de scénarios ne doivent donc pas être autorisés par le programme API.

Toutefois la résolution de tels problèmes n'est pas évidente, car même si l'approche de vérification des programmes est formelle et capable de détecter des scénarios dangereux, les chargés d'études ne disposent pas de méthodes pour corriger formellement ces programmes automates, de manière à garantir leur sûreté de fonctionnement. Des corrections supplémentaires apportées sur les programmes pourraient les rendre certes sûrs de fonctionnement, mais cela demande beaucoup d'expertise. De plus, ces modifications peuvent radicalement changer le fonctionnement des programmes, de telle sorte qu'ils ne respectent plus les spécifications fonctionnelles. Pour rendre les programmes API sûrs de fonctionnement sans en modifier le contenu, nous proposons d'implémenter un filtre logique de commande par contraintes (ou filtre de sécurité) (Pichard et al., 2018) empêchant l'envoi de commandes susceptibles de ramener le système vers des états interdits (figure 62).

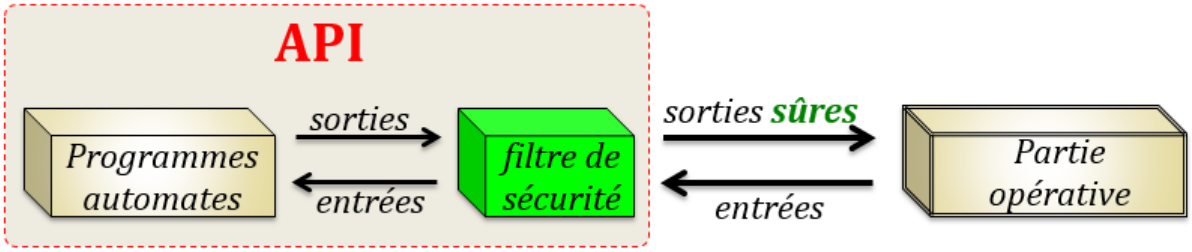


FIGURE 62 – Principe du filtre logique de commandes

Le filtre de commandes est un algorithme qui se présente sous la forme d'un code en langage ST, introduit à la fin du programme automate que l'on souhaite rendre sûr de fonctionnement. Son principe consiste à corriger (ou forcer) les sorties calculées par le programme automate, selon les contraintes de sécurité déduites des états dangereux de la partie opérative, et exprimées sous forme d'équations logiques dans le filtre. Durant chaque cycle automate, le filtre vérifie, avant l'écriture des sorties, si le vecteur de sorties calculées par le programme viole ou non les contraintes de sécurité du filtre. Dans le cas d'une violation des contraintes, il corrige les sorties non prioritaires du programme. La priorité entre les sorties est également définie dans le filtre logique. Les détails sur la structure et le fonctionnement du filtre sont présentés dans (Pichard et al., 2018).

Le filtre de commandes ou filtre de sécurité est donc utilisé en complément du pro-

gramme automate existant, pour garantir en ligne la sûreté de fonctionnement des programmes automates sans en modifier le contenu. De cette manière, le chargé d'études ne sera pas amené à modifier directement ses programmes automates, ce qui irait à l'encontre des principes imposés par la direction ingénierie & Projets de la SNCF. Ce filtre de sécurité peut être également généré automatiquement par un outil, qui reçoit en entrée un fichier texte dans lequel les contraintes de sécurité sont définies en respectant une certaine syntaxe (Pichard et al., 2018).

Toutefois, une des problématiques de l'utilisation du filtre de commandes est liée à la suffisance des contraintes de sécurité sur lesquelles le filtre se base pour corriger éventuellement les sorties du programme. En effet, il est nécessaire de vérifier que l'ensemble des contraintes de sécurité du filtre est suffisant pour garantir formellement la sûreté des programmes. Pour cela, il suffit de procéder une deuxième fois à la vérification de la sûreté de fonctionnement des programmes automates (section 2.5.2), en ajoutant cette fois-ci le filtre de commande à la fin du programme automate.

Sachant que le filtre de commande peut modifier si besoin les valeurs de certaines sorties calculées par le programme automate, cela pourrait également avoir un impact sur le fonctionnement du programme, de telle sorte que celui-ci ne respecte plus les spécifications fonctionnelles (section 2.5.1) une fois qu'il est combiné au filtre de sécurité. Dans ce cas, il est également nécessaire de vérifier, après implémentation du filtre de commandes, que l'ensemble « programmes API + filtre de sécurité » respecte les spécifications fonctionnelles tout en garantissant la sûreté de fonctionnement. Dans le cas contraire, le filtre de sécurité doit être corrigé puis revérifié. Par conséquent, la méthodologie de vérification formelle et de correction des programmes API (initialement présentée à la figure 31) se résume à présent à la figure 63.

Cette nouvelle méthodologie a permis de vérifier formellement et de corriger les programmes d'asservissement dès leur conception, afin d'éviter que les chargés d'études perdent du temps à le faire en usine où les conditions de travail sont largement différentes. Avec cette approche, la recherche d'instructions bloquantes est automatique contrairement à la méthode manuelle utilisée par les chargés d'études, ce qui justifie sa rapidité d'exécution. Les anomalies détectées ainsi que les corrections apportées sur ces programmes ont été confirmées par les chargés d'études en usine, ce qui montre l'avantage de cette technique. Toutefois un dernier verrou concernant la modélisation du système reste à être résolu. En effet, la modélisation du système global (programmes + PO + cahier de re-

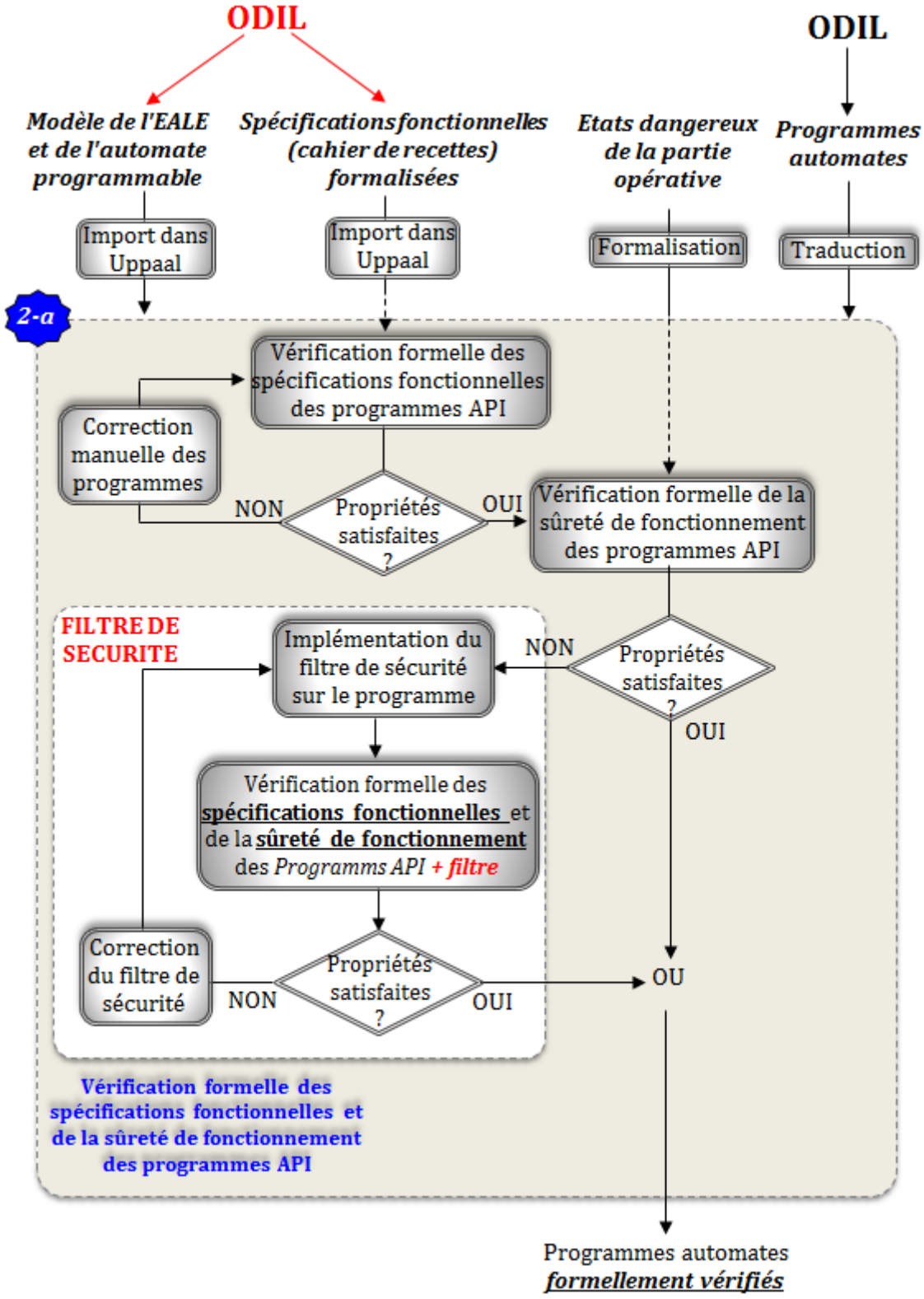


FIGURE 63 – Version 2 de la méthodologie de vérification formelle des programmes API

cettes) se fait manuellement, ce qui n'exclue pas la présence d'erreurs dans les modèles. Ces erreurs de modélisation peuvent compromettre la fiabilité des résultats de la vérification formelle et de la correction des programmes automates.

Pour pallier ces inconvénients, nous optons pour la génération automatique des mo-

dèles de vérification des programmes API, basée sur l'utilisation de l'application ODIL (Coupat, 2014)(Coupat et al., 2018).

4 Génération automatique des modèles de vérification

4.1 Génération automatique des modèles de PO, de l'API et du cahier de recettes

Les modèles Uppaal correspondent à des fichiers XML qui respectent une syntaxe bien définie. Par exemple, la figure 64 décrit le fichier XML du modèle d'un appareil type 1. Étant donné qu'ODIL est capable de générer n'importe quel fichier texte ou XML, il a été choisi pour la génération automatique des modèles de PO et des cahiers de recettes. L'avantage d'avoir choisi cet outil est que les données d'entrées nécessaires pour la génération des modèles de PO et de cahier de recettes sont les mêmes que celles renseignées par le chargé d'études dans ODIL pour générer les livrables métiers (Chapitre 1, section 5.2.1). En effet, le chargé d'études décrit nécessairement dans ODIL la structure de la partie opérative et du système de contrôle commande afin de générer ces livrables. L'idée consiste alors à utiliser ces mêmes données d'entrée pour instancier automatiquement tous les modèles Uppaal nécessaires pour la reconstitution de l'installation dans le Model-Checker. Le cahier de recettes sera également généré de la même manière, car la liste des scénarios de tests utilisés pendant les essais est également paramétrée par le chargé d'études dans l'application.

Comme indiqué dans le chapitre 1, la génération d'un document nécessite la création d'un ***Template* de génération**. Celui-ci a été donc créé et intégré dans le « standard EALE » (structure de données d'ODIL) pour générer, en même temps que les livrables métiers, les modèles suivants (figure 63) :

- Modèle de l'EALE, grâce à l'instanciation automatique des modèles des sous-ensembles de la partie opérative ;
- Modèle du cycle API (figure 35) ;
- Modèle du cahier de recettes (spécifications fonctionnelles formalisées).

```
    <template>
        <name x="5" y="5">App_Type1</name>
        <parameter>bool &amp;so, bool &amp;sf, bool &amp;open, bool &amp;close,
        const int &amp;timeOP, const int &amp;timeCL</parameter>
        <declaration>//commande permanente sans auto-maintien
clock x; int t; bool flag;</declaration>
        <location id="id22" x="127" y="-85">
            <name x="111" y="-117">closed</name> </location>
        <location id="id23" x="-16" y="-88">
            <name x="-40" y="-80">moving</name>
            <label kind="invariant" x="-34" y="-68">x&lt;=t</label> </location>
        <location id="id24" x="-152" y="-88">
            <name x="-184" y="-120">opened</name>   </location>
        <init ref="id24"/>
        <transition>
            <source ref="id23"/>
            <target ref="id24"/>
            <label kind="guard" x="-144" y="-42">!flag and x==timeOP
or flag and (!close or open)</label>
            <label kind="synchronisation" x="-102" y="-68">PO?</label>
            <label kind="assignment" x="-144" y="-59">so:=1</label>
            <nail x="-88" y="-40"/>
        </transition>
        <transition>
            <source ref="id22"/>
            <target ref="id23"/>
            <label kind="guard" x="17" y="-51">open</label>
            <label kind="synchronisation" x="42" y="-68">PO?</label>
            <label kind="assignment" x="17" y="-34">sf:=0, flag=0,
x:=0, t:=timeOP</label>
            <nail x="56" y="-40"/>
        </transition>
        <transition>
            <source ref="id23"/>
            <target ref="id22"/>
            <label kind="guard" x="17" y="-178">flag and close
and x==timeCL</label>
            <label kind="synchronisation" x="42" y="-127">PO?</label>
            <label kind="assignment" x="76" y="-144">sf:=1</label>
            <nail x="56" y="-136"/>
        </transition>
        <transition>
            <source ref="id24"/>
            <target ref="id23"/>
            <label kind="guard" x="-119" y="-187">close</label>
            <label kind="synchronisation" x="-102" y="-127">PO?</label>
            <label kind="assignment" x="-127" y="-170">so:=0, flag=1,
x:=0, t:=timeCL</label>
            <nail x="-85" y="-136"/>
        </transition>
    </template>
    <template>
```

FIGURE 64 – Fichier XML du modèle Uppaal d'un appareil type 1

4.2 Traduction automatique des programmes API en équations booléennes

La méthodologie de traduction des programmes SFC, LD et ST en équations logiques est présentée dans les sections 2.4.2 et 2.4.3. Contrairement aux modèles de vérification générés avec ODIL (section 4.1), les modèles des programmes sont obtenus par traduction manuelle. En effet, après leur génération automatique par ODIL, les programmes API sont souvent modifiés par les chargés d'études. Plutôt que de générer les modèles des

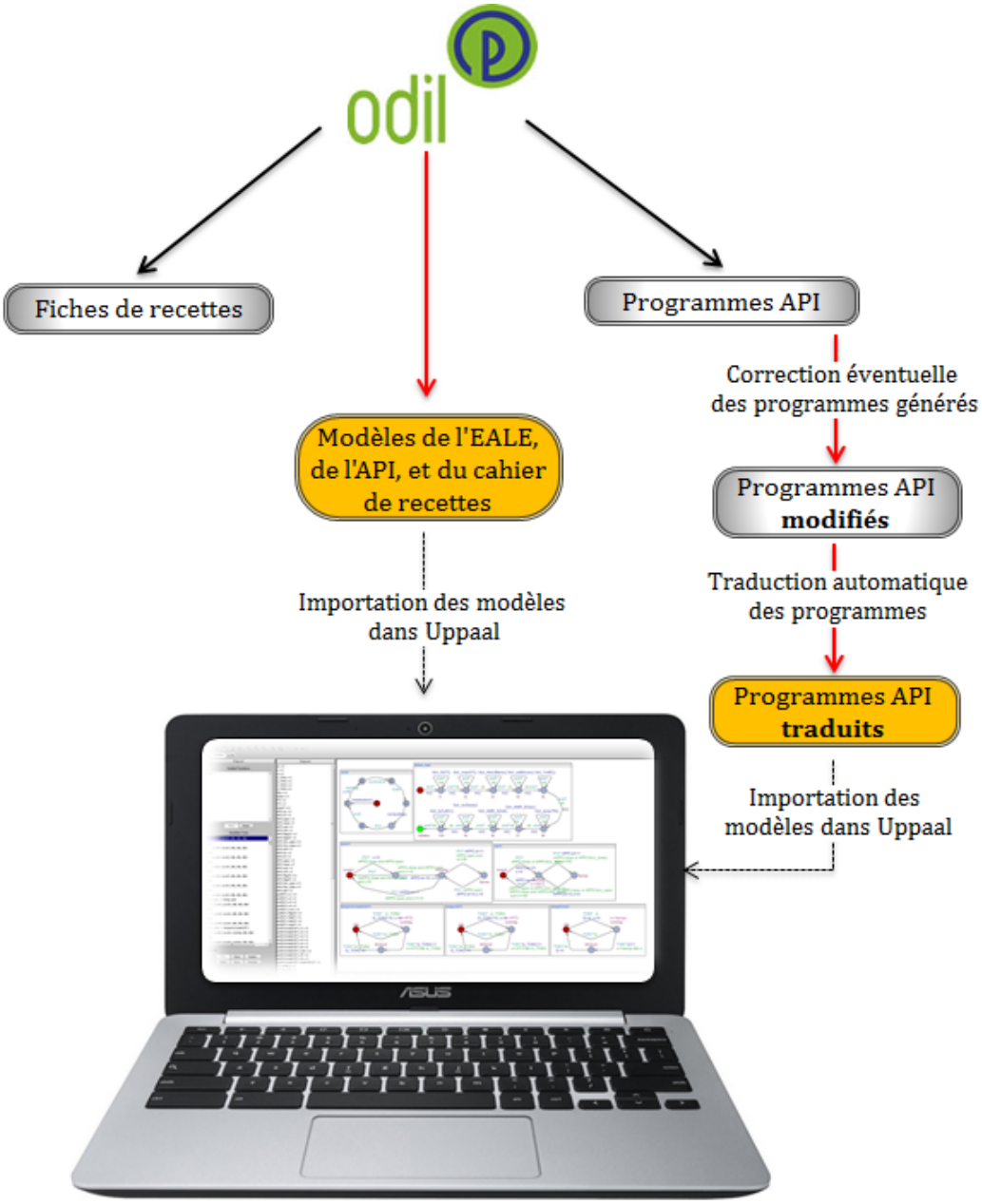


FIGURE 65 – Génération des modèles de vérification

programmes par ODIL (sachant que les originaux pourraient être modifiés), nous propo-

sons de mettre en place un outil qui garantit la traduction automatique des programmes API en langage Uppaal. Cet outil reçoit en entrées les programmes API, puis génère en sortie les programmes traduits, en suivant les règles de traduction décrites dans les sections 2.4.2 et 2.4.3. L'outil de traduction automatique des programmes automates est en cours de développement. Les programmes traduits pourront ensuite être importés dans le Model-Checker Uppaal, en même temps que les autres modèles générés par ODIL (figure 65).

5 Conclusion

La vérification formelle des programmes automates représente la première phase de notre approche méthodologique de V&V des systèmes de contrôle commandes des EALE de la SNCF. Elle est basée sur l'utilisation du Model-Checker Uppaal. Après avoir présenté et décrit le principe de la méthodologie ainsi que les modèles utilisés, nous l'avons appliquée dans ce chapitre sur un exemple concret de programmes automates dédiés au système d'asservissement Fil Pilote (FP) entre disjoncteurs de Départ Traction 1500V. Afin de faciliter l'utilisation de la méthode pour les chargés d'études, les modèles nécessaires pour la vérification des programmes sont automatiquement générés.

Cette approche formelle a montré tout son intérêt à travers de nombreux autres exemples d'applications. Elle a permis de diagnostiquer un certain nombre d'erreurs dans des programmes fournis par les chargés d'études, bien avant que ces derniers n'aient été en mesure de détecter en usine ces mêmes erreurs (et parfois uniquement une partie de ces erreurs, car la vérification formelle est plus exhaustive que leur approche traditionnelle basée sur les tests). Les résultats de simulation s'obtiennent assez rapidement comme l'illustrent l'exemple d'application présenté dans ce chapitre, ce qui est surtout le cas lors de la vérification du respect des spécifications fonctionnelles. En effet, l'évolution du système « PO + programmes » n'est pas aléatoire, car elle est contrôlée par l'exécution séquentielle des scénarios de tests sur le système. Pour assurer la sécurité, l'utilisation du filtre logique permet de garantir formellement la sûreté de fonctionnement des programmes automates.

Cependant, comme pour la plupart des méthodes formelles, cette technique est parfois exposée à l'explosion combinatoire. Ce phénomène peut survenir uniquement lors de la vérification exhaustive de la sûreté de fonctionnement des programmes automates, dû à l'augmentation de l'espace d'états du modèle global. Une des solutions possibles pour

pallier cet inconvénient consiste à combiner le Model-Checker Uppaal avec un supercalculateur lors de la vérification. La puissance de calcul est ainsi multipliée, ce qui permet de parcourir un espace d'états plus vaste. Cette solution a été testée avec l'utilisation du supercalculateur **ROMEO** de l'université de Reims Champagne Ardenne *(www.romeo.univ-reims.fr)*. Un autre inconvénient de la technique de vérification formelle est lié à la formalisation des états interdits. Cette tâche nécessite beaucoup d'expertise, aussi bien sur les langages formels que sur le métier des EALE. Dans le cas contraire, le risque est que la propriété formalisée dans le Model-Checker en logique temporelle CTL peut ne pas correspondre à la propriété réelle que le chargé d'études souhaite vérifier sur le système.

Après la vérification formelle et la correction des programmes automates, la prochaine étape de la méthodologie consiste à valider automatiquement le système de contrôle commande (programmes automates et câblage des armoires de contrôle commande) avec l'utilisation du Virtual Commissioning.

Chapitre IV

Validation automatique des programmes API et du câblage des armoires

1 Introduction

La vérification formelle des programmes automates représente la première phase de notre approche méthodologique de V&V des systèmes de contrôle commande. Cependant, la vérification ainsi que les corrections apportées n'ont pas été directement appliquées sur les programmes automates originaux (i.e. générés à l'issue de la phase 1 du workflow des chargés d'études), mais plutôt sur les « modèles » des programmes automates. Bien que ces modifications apportées sur les modèles de programmes seront manuellement reprises sur les versions originales par les chargés d'études, celles-ci doivent nécessairement être validées en ligne, au même titre que le câblage des armoires du système de contrôle commande.

Ce chapitre a pour but de présenter la solution retenue pour la validation automatique du système de contrôle commande des EALE. Comme énoncé dans le chapitre 2, cette solution, basée sur l'utilisation du Virtual Commissioning, combine les architectures **Sofware-In-the-Loop** et **Hardware-In-the-Loop** pour valider respectivement les programmes automates (après la vérification formelle et la correction) et le câblage des armoires de contrôle commande (en usine). Pour éviter tout changement radical de leur métier actuel et proposer aux chargés d'études la solution la plus adéquate, les spécifications fonctionnelles de la solution proposée (exigences et performances attendues)

ont été définies en fonction des besoins des chargés d'études responsables des tests de validation du système de contrôle commande (programmes automates et câblage des armoires en usine). La section 2 présente d'abord les architectures matérielle et logicielle de la solution proposée, ensuite les spécifications fonctionnelles sont détaillées. Cette solution étant uniquement dédiée à la validation automatique du système de contrôle commande des EALE à la SNCF, les outils et/ou logiciels servant à cet effet n'existent pas encore. Un appel d'offre a alors permis de choisir un intégrateur qui est en charge de la réalisation de cette solution. La réponse technique de l'appel d'offre pour la réalisation de cette solution est présentée en section 3.

2 Présentation de la solution de validation automatique du système de contrôle commande

La validation automatique du système de contrôle commande (programmes automates et câblage des armoires) est principalement constituée de deux étapes (figure 66) :

— Étape 1 : validation automatique des programmes API par Simulation Software-In-the-Loop (SIL) ;

— Étape 2 : validation automatique du câblage des armoires par Simulation Hardware-In-the-Loop (HIL).

Ces deux architectures, SIL et HIL, nécessitent toutes deux l'utilisation d'un logiciel simulateur de partie opérative pour le Virtual Commissioning (chapitre 2, section 2.3, figures 25 et 26). Quant à l'architecture HIL, elle requiert en plus une interface physique servant d'intermédiaire entre le logiciel de Virtual Commissioning et le système de contrôle commande. Cet ensemble constitué par le « logiciel de Virtual Commissioning » et « l'interface physique » est appelée **banc d'essais**.

L'idée du mode SIL est venue du banc d'essais (HIL), et ne faisait pas initialement partie des besoins exprimés par les chargés d'études de la SNCF. En effet, ces derniers avaient besoin d'une solution qui permettrait de valider automatiquement les programmes automates et le câblage des armoires « en même temps ». Cette demande leur permettrait certes d'automatiser, et par conséquent d'accélérer les tests de validation basés sur le cahier de recettes, mais la complexité du diagnostic d'éventuelles erreurs resterait la même (chapitre 2, section 2.6.2), car les problèmes pourraient provenir soit d'un mauvais programme, soit d'une erreur de câblage. C'est dans ce sens que l'architecture SIL a été

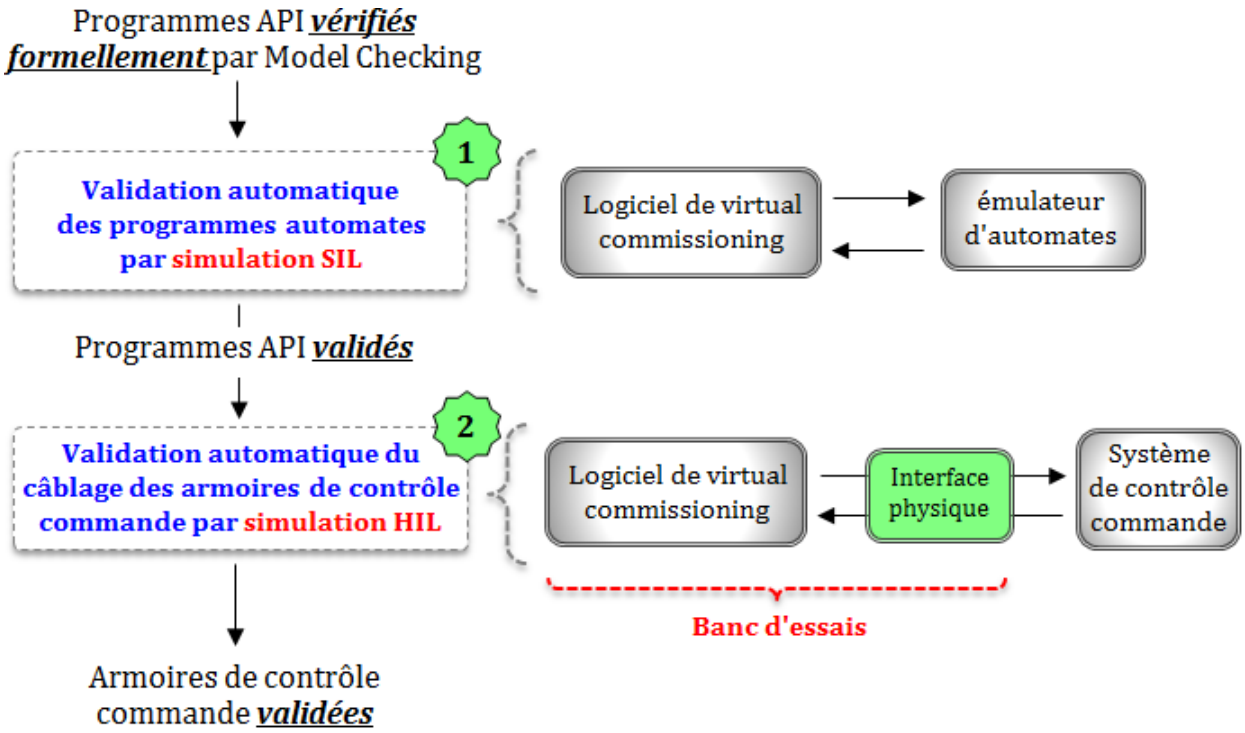


FIGURE 66 – Validation automatique du système de contrôle commande

proposée pour valider en amont les programmes automates, avant la validation du câblage avec le banc d'essais.

Pour faciliter la description des deux architectures HIL et SIL, ainsi que les spécifications fonctionnelles attendues, le banc d'essais (mode HIL, étape 2) est présenté en premier à la section 2.1, puis l'architecture SIL (étape 1), déduite du banc d'essais, est présentée à la section 2.3.

2.1 Architecture du mode HIL (banc d'essais)

L'architecture du banc d'essais prévu pour les tests de validation du système de contrôle commande en usine est présentée à la figure 67. Cette architecture met en évidence les trois composantes essentielles :

— l'armoire de tests, qui représente l'interface physique entre le système de contrôle commande et le PC de simulation,

— un PC de simulation dans lequel est installé le logiciel de Virtual Commissioning (ou le logiciel simulateur de partie opérative),

— le système de contrôle commande à valider, comprenant les différents abonnés automates intégrés dans les armoires de contrôle commande, le CLE, et le RTU (voir chapitre 1).

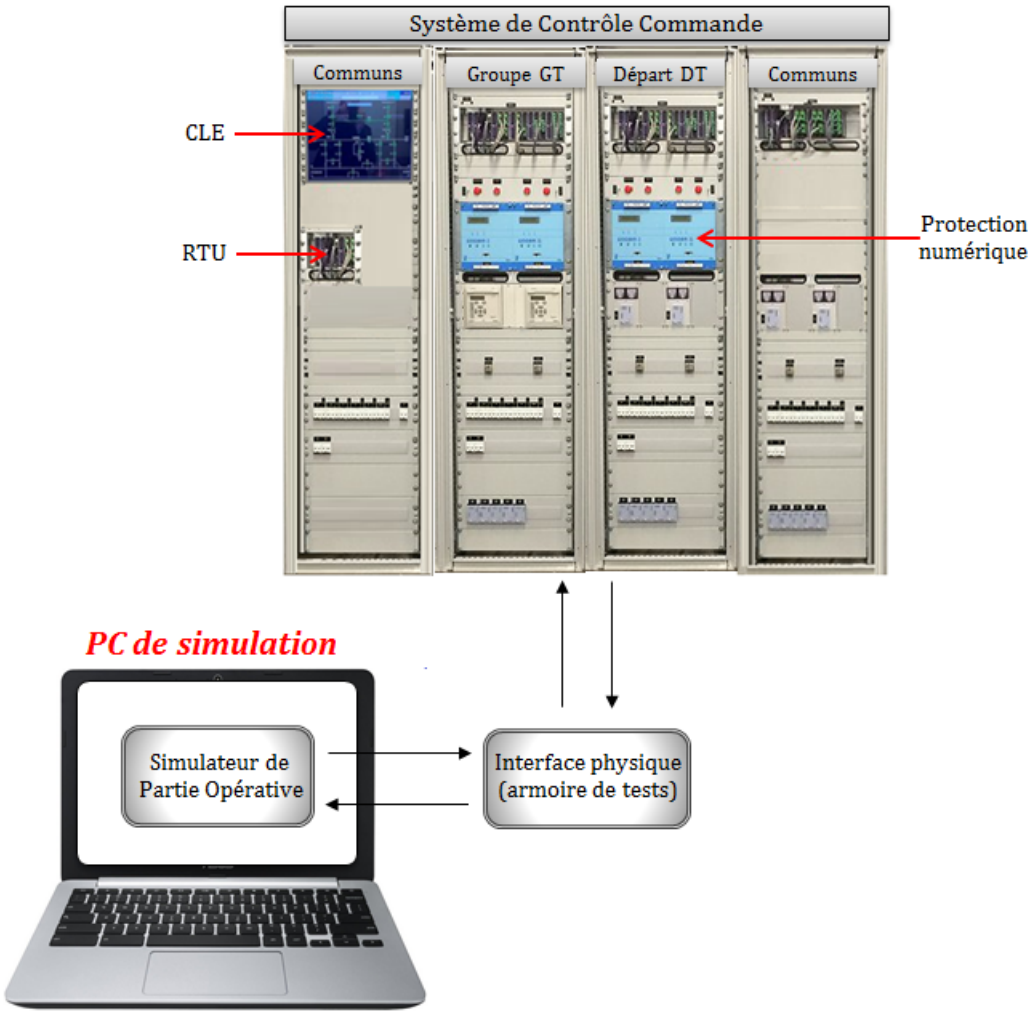


FIGURE 67 – Architecture du banc d'essai

2.1.1 L'interface physique (armoire de tests)

L'armoire de tests permet de connecter les entrées/sorties du simulateur de partie opérative aux entrées/sorties des armoires de contrôle commande (figure 67). Selon la taille de l'EALE étudiée, le nombre d'entrées/sorties du système de contrôle commande est estimé à un maximum de 256 entrées et 768 sorties. Pour bien assurer son rôle d'interface physique, cette armoire de tests doit renfermer un automate programmable qui doit nécessairement répondre aux critères suivants :

— La limite d'entrées/sorties maximale doit être respectée ;

— Les niveaux de tensions admissibles des cartes d'entrées/sorties de cet automate doivent être identiques à ceux des armoires de contrôle commande (24V, 48V, et 120V) ;

— Le raccordement des entrées/sorties du système de contrôle commande vers l'armoire de tests doit se faire par l'intermédiaire d'un dispositif industriel débrochable,

pour faciliter l'interconnexion des équipements ;

— L'automate doit être capable de communiquer avec les automates des armoires de contrôle commande (abonnés automates), le CLE, et l'abonné RTU (AARTU) pour récupèrer toutes les variables du réseau RLI ;

— Le type d'automate de l'armoire de tests doit être programmable avec le logiciel STRATON de Copalp, pour faciliter sa prise en main par les chargés d'études qui maîtrisent déjà ce logiciel ;

Mis à part l'automate programmable, l'armoire de tests devra également renfermer des valises d'injections (figure 68). Celles-ci seront connectées aux équipements de protection numérique du système de contrôle commande pour simuler des défauts électriques tels que les court-circuits, les surcharges, des absences tensions sur les caténaires ... Aujourd'hui pour simuler des défauts lors des essais en usine, le chargé d'études utilise également des valises à injections qu'il paramètre manuellement pour simuler des défauts internes (chapitre 2 section 2.6). L'intégration de ces valises dans l'armoire de tests permet d'automatiser leur paramétrage (par l'automate de l'armoire de tests) afin d'accélérer le processus de validation.

FIGURE 68 – Exemple de valise d'injection paramétrable par ordinateur

La validation du système de contrôle commande consiste également à valider le paramétrage de la connexion avec le Central Sous-Station (figure 11 du chapitre 1). Pour cela, l'armoire de tests doit intégrer un module permettant de simuler le Central Sous-

Station. Ce module devra alors communiquer avec l'abonné RTU du système de contrôle commande via le protocole de communication HNZ. Cet échange pourra être supervisé par le chargé d'études depuis le PC de simulation de partie opérative.

L'armoire de tests doit également intégrer un perturbographe pour surveiller les états des entrées/sorties échangées avec le système de contrôle commande. Un perturbographe est un appareil de surveillance qui enregistre en temps réel la forme des signaux des données d'un système, et les représente sous forme de chronogrammes mettant en évidence leur évolution en fonction du temps.

En ce qui concerne les contraintes physiques de l'armoire de tests, elle devra être la moins encombrante possible (en poids et en volume) avec une bonne mobilité, étant donné qu'elle pourra être réutilisée sur différents sites.

Après avoir été raccordé au système de contrôle commande et au PC de simulation, cette armoire de tests doit être intégralement vérifiée avant de démarrer la recette usine, ceci pour s'assurer qu'il n'y ait pas d'anomalies telles que :

— Une mauvaise configuration de l'armoire de tests et de son automate ;
— Des entrées/sorties mal câblées ;
— Des matériels défectueux ;
— Un réseau de communication non établi ;
— Une mauvaise calibration des valises d'injection ;
— ...

La vérification de l'armoire doit être automatique, une méthodologie de vérification automatique de l'armoire de tests (basée par exemple sur l'exécution d'un script depuis un logiciel externe connecté à l'armoire) devra être mise à disposition du chargé d'études (par le futur concepteur du banc d'essais).

2.1.2 Le logiciel de Virtual Commissioning du PC de simulation

Exploitable depuis un ordinateur, le simulateur de partie opérative (ou simulateur d'EALE) représente en mode virtuel l'installation électrique étudiée, avec un comportement identique. Il permet au chargé d'études de faire la commande et la supervision de l'EALE depuis l'interface graphique. Pour les besoins de la recette usine, ce logiciel doit intégrer deux interfaces principales, dont l'une permet de commander et de visualiser l'état de la partie opérative (interface N^o1, figure 69), et l'autre permet d'accéder au paramétrage du cahier de recettes (interface N^o2, figure 71). On rappelle que ce logiciel de

Virtual Commissioning dédié à la simulation des EALE de la SNCF n'existe pas encore et doit être conçu, les IHM des figures 69 et 71 sont des dessins qui illustrent les besoins attendus dans chacune des interfaces du logiciel de Virtual Commissioning. L'interface graphique N^o1 (figure 69) doit renfermer les éléments suivants :

Le synoptique de l'EALE simulée dans lequel le nom de chaque équipement de l'EALE doit être visible. Tous les équipements présents doivent respecter la règle de représentation des symboles pour schémas électriques décrite dans la figure 70. Les jeux de barres et les caténaires doivent être simplement représentés par des traits pleins, de préférence animés en couleurs lorsqu'ils sont sous tension. L'état de chaque appareil présent doit être graphiquement mis en évidence (appareil ouvert/fermé/perte de position, Transformateur/redresseur en défaut, présence/absence de tension au niveau des jeux de barres ou des caténaires...) via des animations en couleurs et/ou en mouvements.

Le scénario de tests *« en cours d'exécution »* : l'interface N^o1 doit également intégrer une zone dans laquelle s'afficheront les informations de la fiche de recettes en cours d'exécution, telles que :

- le mode d'exécution du scénario de tests en cours (voir section 2.2.2),
- le nom du scénario de tests en cours d'exécution,
- son état (fiche validée / fiche bloquante / fiche non exécutée),
- sa condition initiale : sous forme d'équation booléenne par exemple, dont la valeur booléenne est affichée en temps réel lors de la simulation (TRUE ou FALSE), ce qui permet à l'utilisateur de savoir si la fiche à exécuter est bien dans les conditions initiales de démarrage,
- les actions à réaliser et les résultats attendus, avec un indicateur qui permet de localiser (lors de la simulation) l'action en cours d'exécution,
- une zone qui affiche les commentaires de l'utilisateur rentrés en ligne ou hors ligne (voir dans la case « commentaires utilisateur » de la figure 69),
- un panneau de commande qui permet durant la simulation de démarrer, d'interrompre momentanément, ou de stopper l'exécution d'une fiche de recettes, et de passer au scénario de tests précédent ou suivant du cahier de recettes.

La barre des menus du projet qui doit disposer d'onglets permettant entre autres :

- d'importer, d'exporter, ou de créer un nouveau projet de recette usine,

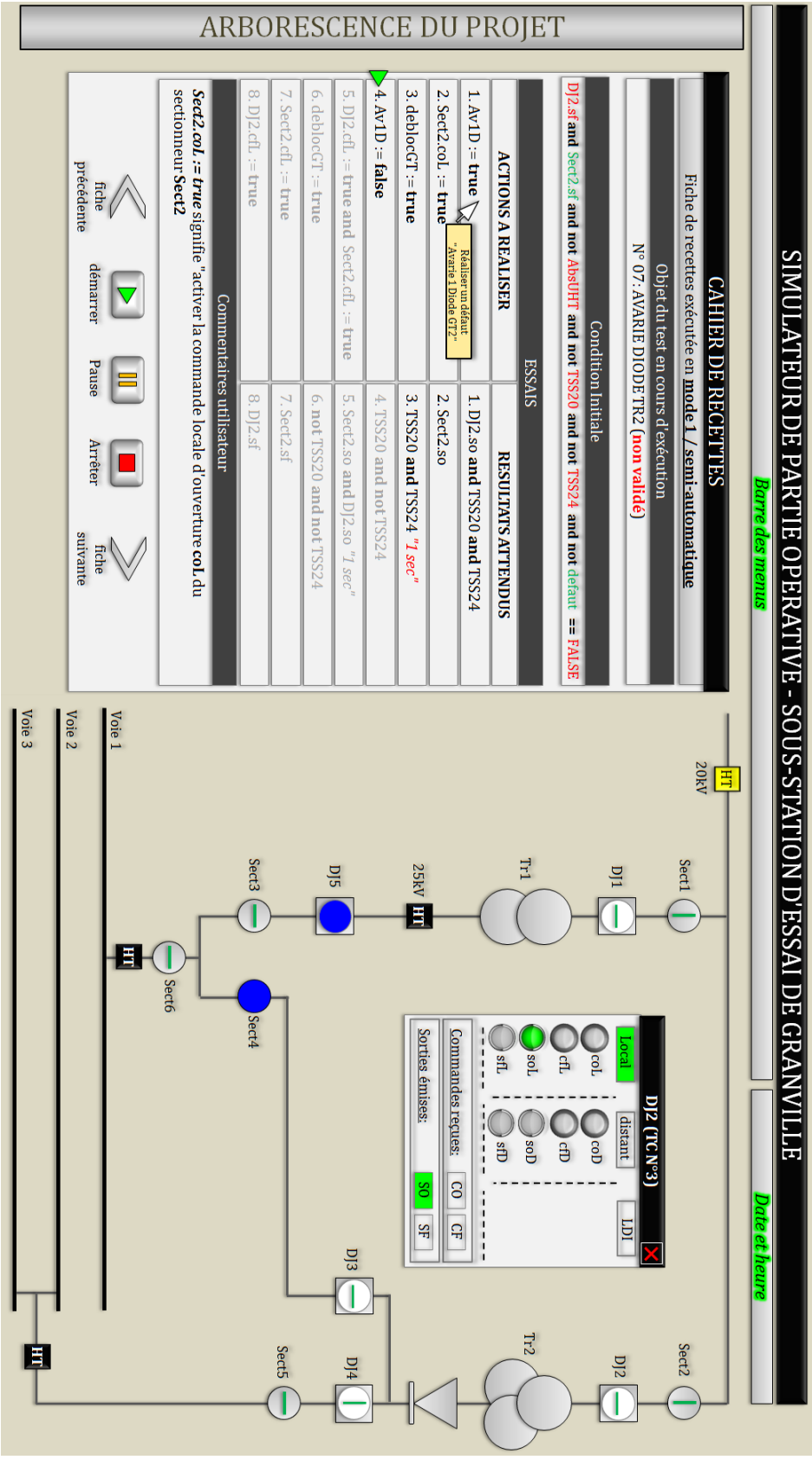


FIGURE 69 – Spécifications de l'interface N^o1 du simulateur d'EALE

— d'accéder au dictionnaire de variables du simulateur d'EALE contenant leurs noms, adresses, libellés, et toutes les informations nécessaires,

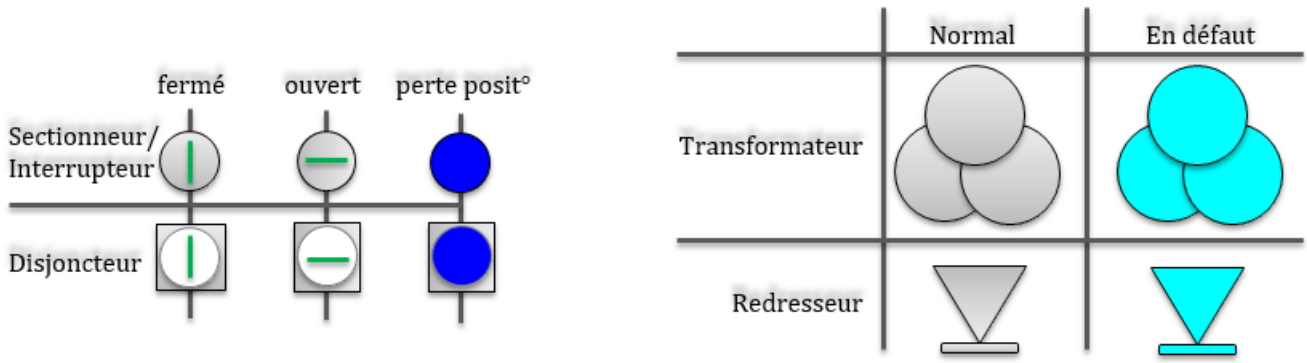


FIGURE 70 – Règle de représentation des équipements électriques de l'EALE

— de lancer la simulation du projet (voir section 2.2),
— d'accéder à la table de données échangées entre le simulateur d'EALE et l'ensemble RTU, CLE, et armoires de contrôle commande,
— d'accéder à l'historique des événements durant la simulation,
— d'accéder au paramétrage des fiches de recettes (interface N^o2 de la figure 71).

La date et l'heure en cours, ainsi que l'arborescence du projet (pour une navigation plus facile dans le logiciel) seront également accessibles depuis l'interface N^o1.

Les fiches de recettes actuellement utilisées par les chargés d'études sont rédigées au format Word. A l'instar de la formalisation des fiches de recettes dans le Model-Checker Uppaal, celles-ci doivent également être converties en un langage compatible avec le futur logiciel de Virtual Commissioning. Toutefois pour ne pas éventuellement égarer le chargé d'études avec ce nouveau langage (lors des essais), il doit pouvoir accéder au libellé ou commentaire de toute « action à réaliser » (ou « résultat attendu ») de la fiche de tests en pointant la souris sur la case correspondante (voir exemple figure 69, sur l'action à exécuter N^o1).

En ce qui concerne l'interface N^o2, celle-ci doit renfermer, en plus de l'arborescence du projet et la barre des menus, une liste structurée de tous les scénarios de tests du cahier de recettes. Cette liste doit mettre en évidence l'état de chaque scénario de tests du cahier de recettes durant la simulation (fiche « validée », « bloquante », ou « non exécutée ») grâce à des animations en couleurs (voir partie gauche de la figure 71). De plus en sélectionnant une fiche de recettes sur la liste de gauche, son contenu doit apparaître à droite de l'interface avec toutes ses informations associées. Une fiche de recettes ne peut être modifiée (en rajoutant, supprimant ou modifiant des lignes) ou supprimée qu'à partir de cette interface et lorsque l'utilisateur n'est pas en mode simulation. L'utilisateur

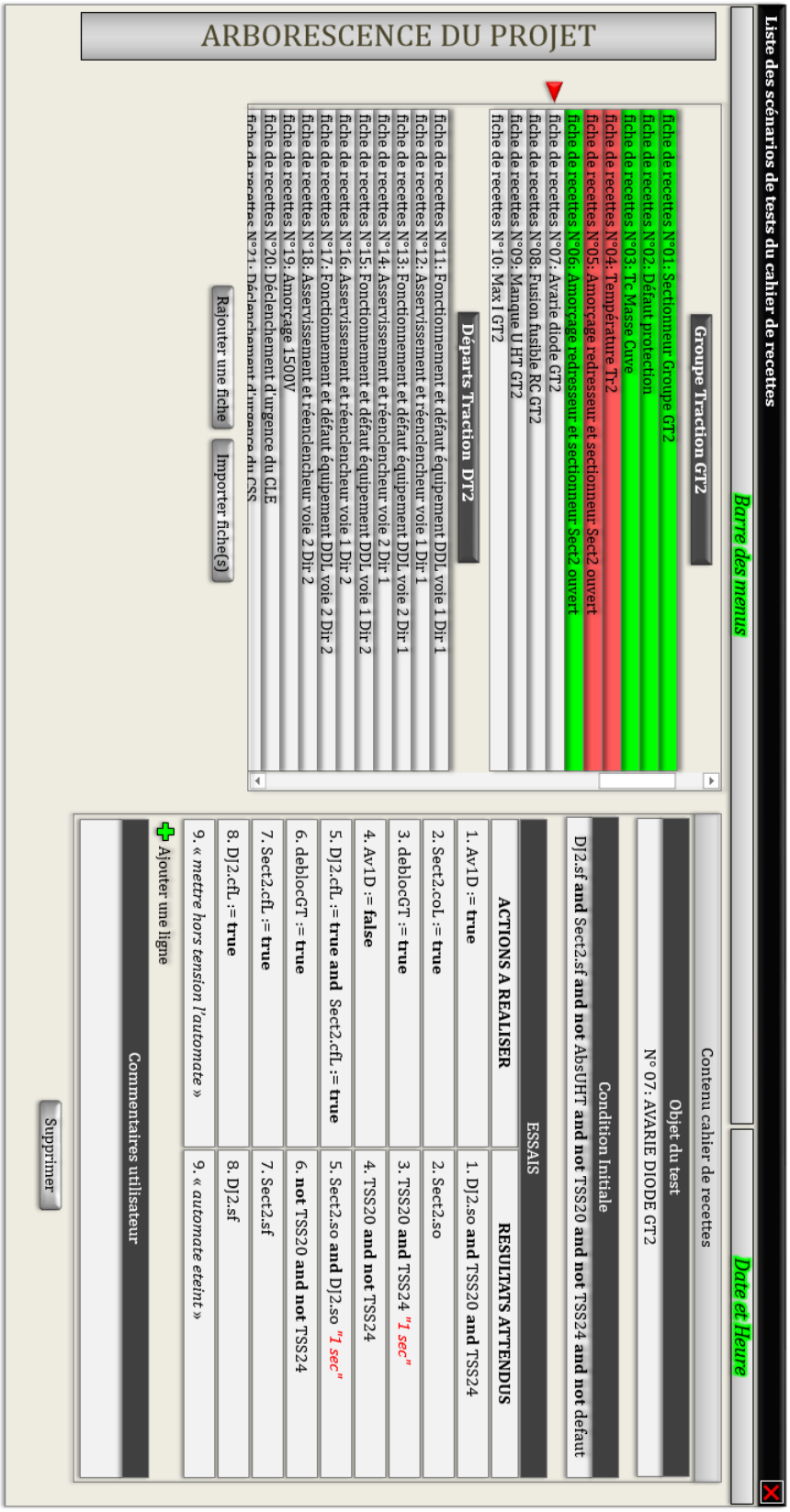


FIGURE 71 – Interface N^o2, paramétrage des fiches de tests

doit également avoir la possibilité de rajouter si besoin une nouvelle fiche de recettes, en prenant soin de préciser le nom de la fiche, sa condition initiale, les actions et les

résultats attendus (en les décrivant avec des mots clés et commentaires, figure 71). La fiche sera alors rajoutée sur la liste, en dessous de celle qui aura été présélectionnée, et par conséquent les numéros des fiches de recettes suivantes seront décalés d'une unité. De la même manière l'utilisateur peut importer une ou des fiches de recettes (format texte ou XML), les rajouter sur le cahier de recettes et les modifier si besoin.

Toutefois l'ajout d'un scénario de tests sur le cahier de recettes ne peut être validé qu'après vérification automatique de la syntaxe des données rentrées sur la fiche, définie comme suit :

— L'objet du test, et les commentaires utilisateur sur la fiche et sur chaque action et résultat attendu : au format « chaîne de caractères » ;
— La ou les actions à réaliser : il peut s'agir d'une ou de plusieurs affectations de valeurs « TRUE » ou « FALSE » à des variables booléennes existantes (voir figure 69). Dans le cas d'une action devant être réalisée manuellement et qui ne peut être automatisée, saisir l'action entre des guillemets (voir exemple figure 71, action $N^o 9$ de la fiche de tests) ;
— Le ou les résultats attendus : sous forme d'expressions booléennes avec des variables existantes, ou bien sous forme de texte entre guillemets dans certains cas ;
— La condition initiale de la fiche de tests : sous forme d'équation booléenne définie avec des variables existantes, ou sous forme de texte entre guillemets.

Pour chacune des cases (actions, résultats, ou condition initiale), l'utilisateur pourra s'il le souhaite rentrer un commentaire ou libellé, qui pourra être affiché pendant la simulation à la demande de l'utilisateur (avec la souris comme décrit précédemment).

Durant l'exécution d'un scénario, tout texte mis entre des guillemets (action à réaliser, résultat attendu, ou condition initiale) ne pourra être interprété par le logiciel, mais ce dernier doit pouvoir automatiquement les reconnaître et par conséquent, attendre que l'utilisateur effectue manuellement l'action (ou vérifie le résultat ou la condition) avant de la (le) valider pour passer à la suite. Dans le cahier de recettes, il existe des scénarios de tests qui ne renferment que des actions manuelles. Dans ce cas toutes les instructions et les résultats pourront être saisis entre guillemets, et par conséquent l'exécution de ces fiches ne pourra être faite qu'en mode pas à pas (voir section 2.2.2).

Les spécifications détaillées ci-dessus permettront aux chargés d'études d'adopter facilement une nouvelle méthodologie de validation automatique des systèmes de contrôle commande, méthodologie dans laquelle ils retrouveront toujours leur métier, leur exper-

tise et leur savoir-faire. Toutefois, cette approche nécessite de renseigner dans le logiciel les données nécessaires pour les essais, à savoir :

— la structure de l'EALE (pour la représenter dans interface N^o1),

— l'ensemble des fiches de tests du cahier de recettes (interface N^o2).

La saisie des données dans le logiciel de Virtual Commissioning représente une tâche supplémentaire et peut prendre du temps, sans compter les éventuelles erreurs lors de la saisie. Pour pallier cela, il est impératif que toutes les données d'entrées du simulateur puissent être automatiquement générées par ODIL et importées depuis le logiciel de Virtual Commissioning. On rappelle qu'au stade de la recette usine, le logiciel ODIL aura déjà été utilisé par le chargé d'études pour générer les programmes API et le cahier de recettes du projet. De ce fait, en ajoutant de nouveaux *templates* de génération dans la structure de données d'ODIL (Standard EALE), les données d'entrées du logiciel de Virtual Commissioning, ainsi que le programme de l'automate de l'armoire de tests, pourront être générés en même temps que les livrables du projet. Il suffira alors d'importer les données générées vers les applications associées (figure 72).

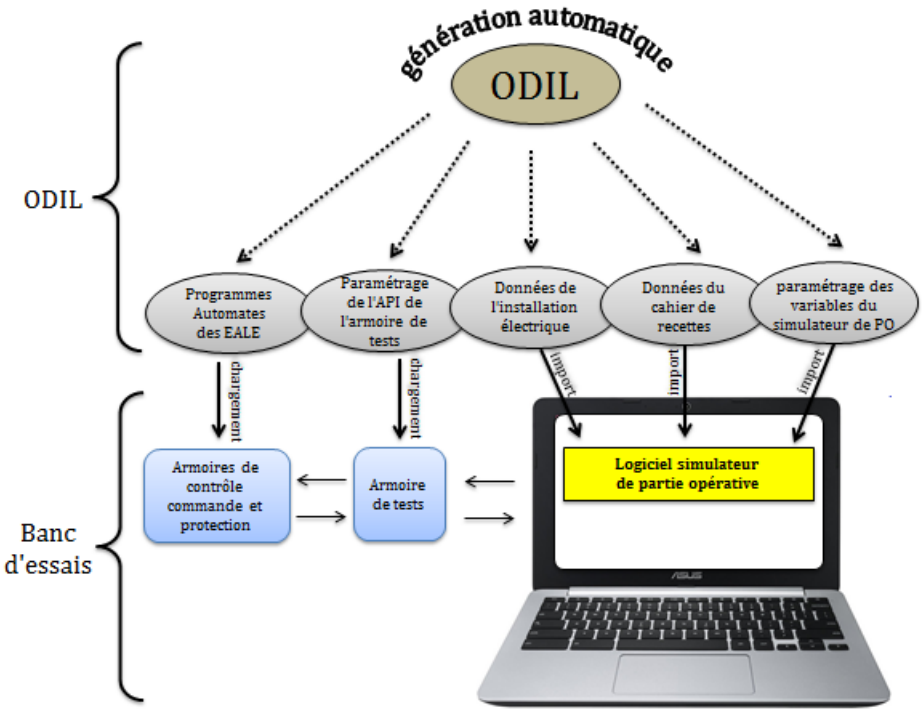


FIGURE 72 – Génération automatique des données du banc d'essais par ODIL

Le chargé d'études doit ensuite pouvoir adapter les données générées si besoin (pour corriger certains aspects dus aux particularités que présente l'EALE étudiée) avant de procéder directement à la simulation. Il doit également pouvoir partir de zéro pour créer son

projet de simulation, ou réutiliser (importer) les données d'un autre projet de simulation qu'il pourra ensuite modifier.

Les données d'entrées générées par ODIL et nécessaires pour le Virtual Commissioning sont de trois types :

— les données de l'EALE, pour sa représentation dans l'interface N^o1,

— les données du cahier de recettes,

— les données de paramétrage des variables du simulateur d'EALE.

En ce qui concerne les données de paramétrage des variables du simulateur, elles permettent de définir et d'adresser toutes les variables de simulation, de telle sorte que chacune d'elles puisse être correctement reliée à la bonne variable du système de contrôle commande. La figure 73 décrit le flux d'échanges de données entre le simulateur d'EALE, l'armoire de tests, et l'ensemble CLE-RTU-Armoires électriques.

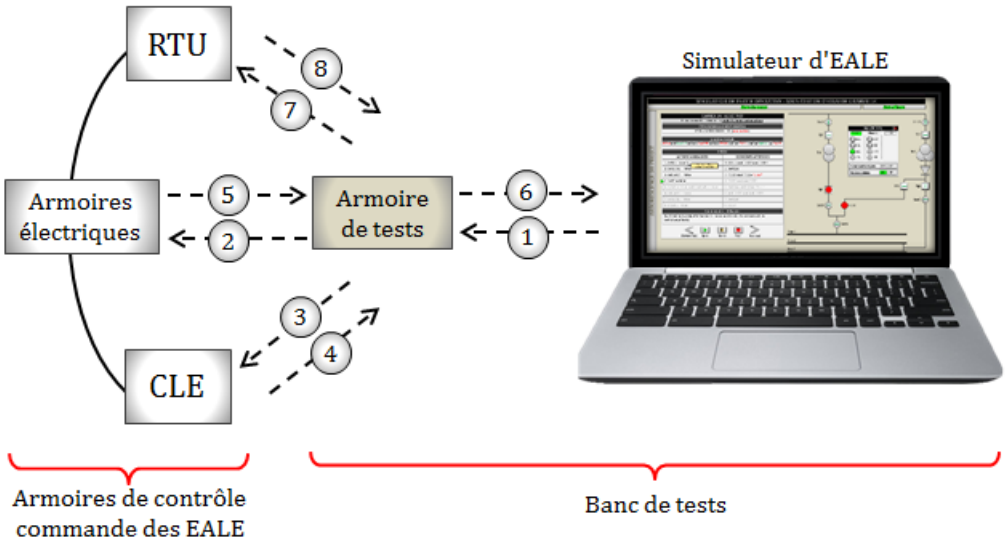


FIGURE 73 – flux d'échange de données entre le banc d'essais et le système de contrôle commande

Ces données échangées sont classées en fonction de leurs types, elles sont numérotées de 1 à 8 comme indiqué sur la figure 73.

Données type 1 *(du simulateur d'EALE vers l'armoire de tests)*

Il s'agit des données envoyées par le simulateur d'EALE vers l'armoire de tests. Depuis l'interface N^o1 du simulateur, le chargé d'études doit pouvoir émettre des commandes locales ou distantes d'ouverture et de fermeture pour chaque appareil, créer des défauts (à shunt ou à injection) sur l'EALE, ou modifier certaines variables réseau du CLE ou du RTU à partir de la table de données échangées. Le simulateur d'EALE transmet également

en temps réel tous les états ouverts/fermés de tous les appareils. Toutes ces informations seront récupérées par l'automate de l'armoire de tests, avant d'être transmises aux armoires de contrôle/commande/protection des EALE (données type 2), au CLE (données type 3), et au RTU (données type 7).

Données type 2 *(de l'armoire de tests vers les armoires électriques)*
Les variables associées aux états des appareils simulés (ouverts/fermés) et récupérées par l'automate de l'armoire de tests sont transmises aux automates des armoires électriques, en passant respectivement par les borniers de l'armoire de tests, les borniers des armoires électriques, puis les cartes d'entrées/sorties des automates. Il en est de même pour les demandes d'activation de défauts à shunt envoyées par le simulateur. En ce qui concerne les défauts à injection, l'ordre d'activation est plutôt envoyé aux valises d'injection, qui injectent alors du courant ou de la tension aux borniers auxquels elles ont été préalablement connectées.

Données type 3 *(de l'armoire de tests vers le CLE)*
Les commandes en mode local envoyées par le simulateur d'EALE et reçues par l'automate de l'armoire de tests sont transmises au CLE. Ce dernier dispose d'une table de variables adressées dont certaines d'entre elles doivent être accessibles en écriture depuis le simulateur d'EALE.

Données type 4 *(du CLE vers l'armoire de tests*)
Elles correspondent aux variables du CLE qui sont lues par l'automate de l'armoire de tests. Elles sont par la suite transmises au simulateur d'EALE. Ces variables renferment comme informations : les ordres de télécommande locale, les variables du réseau FIP provenant des abonnées automates, la datation...

Données type 5 (*des armoires électriques vers l'armoire de tests*)
Une fois que les programmes automates des EALE ont calculé les sorties, ces données sont récupérées par l'armoire de tests puis affectées aux entrées physiques de l'automate de l'armoire de tests. Ces sorties seront reçues par les appareils respectifs du simulateur de partie opérative et feront évoluer leurs états.

Données type 6 (*de l'armoire de tests vers le simulateur d'EALE*)
Il s'agit de l'ensemble des données de type 4, type 5, et type 8.

Données type 7 / type 8 (*entre l'armoire de tests et le RTU*)
Il s'agit des données distantes échangées entre le RTU et le simulateur de CSS intégré dans l'armoire de tests.

2.2 Spécifications fonctionnelles du banc d'essais

Après le paramétrage et la vérification des données du logiciel de Virtual Commissioning, la validation automatique du système de contrôle commande peut débuter. Le fonctionnement du futur logiciel doit respecter les spécifications fonctionnelles décrites ci-dessous (figure 74). On rappelle que ce logiciel sert non seulement pour la validation automatique des programmes automates (architecture Software-In-the-Loop), mais aussi du câblage des armoires de contrôle commande en usine (architecture Hardware-In-the-Loop). Excepté le paramétrage du logiciel (pour l'une ou l'autre des architectures), son fonctionnement reste identique.

La première étape consiste à vérifier automatiquement la bonne configuration matérielle de l'armoire de tests dans le cas du mode HIL (comme spécifié à la fin de la section 2.1). Ensuite, le logiciel doit proposer à l'utilisateur de choisir entre les deux modes de simulation :

- le mode Software-In-the-Loop (pour la validation automatique des programmes API) : le logiciel de Virtual Commissioning communique alors avec des émulateurs d'automates dans lesquels sont transférés les programmes à valider (figure 77).
- le mode Hardware-In-the-Loop en usine, pour la validation automatique du câblage des armoires de contrôle commande (figure 67).

Après le choix de l'architecture, le chargé d'études peut débuter la simulation en mode « **manuel** » ou en mode « **cahier de recettes** ».

2.2.1 La simulation en mode manuel

En mode manuel, l'interface N^o1 du simulateur d'EALE doit être sous la forme définie à la figure 75.

Le chargé d'études doit avoir la liberté d'agir sur l'EALE (en commandant des appareils, en créant des défauts. . .). Pour cela, il doit cliquer sur les objets de l'installation pour

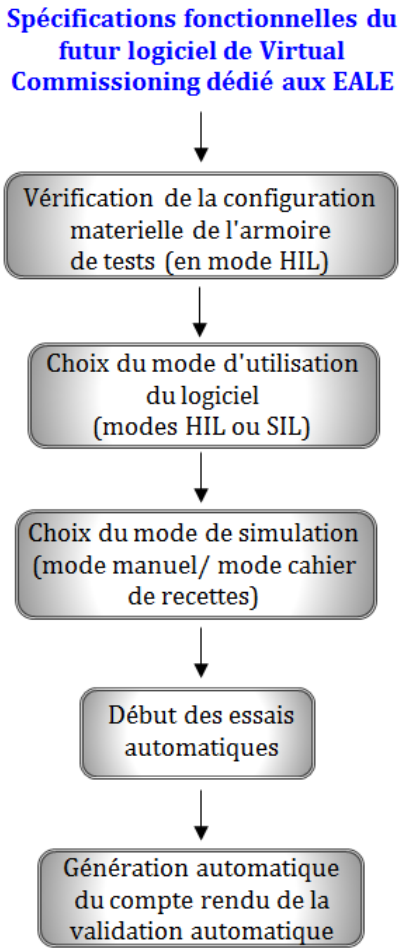


FIGURE 74 – Spécifications fonctionnelles du futur logiciel de Virtual Commissioning dédié aux EALE

faire apparaître les panneaux de commandes sous forme de fenêtres pop-up lui permettant de piloter l'installation (en mode distant ou local). Ces fenêtres peuvent être également ouvertes depuis l'arborescence du projet. Ces panneaux de commande doivent contenir les informations suivantes pour chaque appareil (figure 76) :

- **Pour les disjoncteurs, sectionneurs, et interrupteurs** : le nom de l'appareil, le numéro, les modes de commande disponibles (modes local/distant/local individuel), l'état des commandes d'ouverture/fermeture provenant du système de contrôle commande, et l'état ouvert/fermé de l'appareil.
- **Pour les transformateurs et redresseurs** : le nom, le numéro, et la liste des défauts observés ainsi que leur état.

Les fenêtres pop-up affichées doivent être déplaçables et de dimensions modifiables, afin d'adapter la supervision de l'EALE. A la différence de la figure 69, l'interface N^o1 n'affiche pas, en mode manuel, le scénario de tests en cours d'exécution, ce qui est normal vu que l'évolution de la partie opérative est contrôlée par le chargé d'études à travers ses

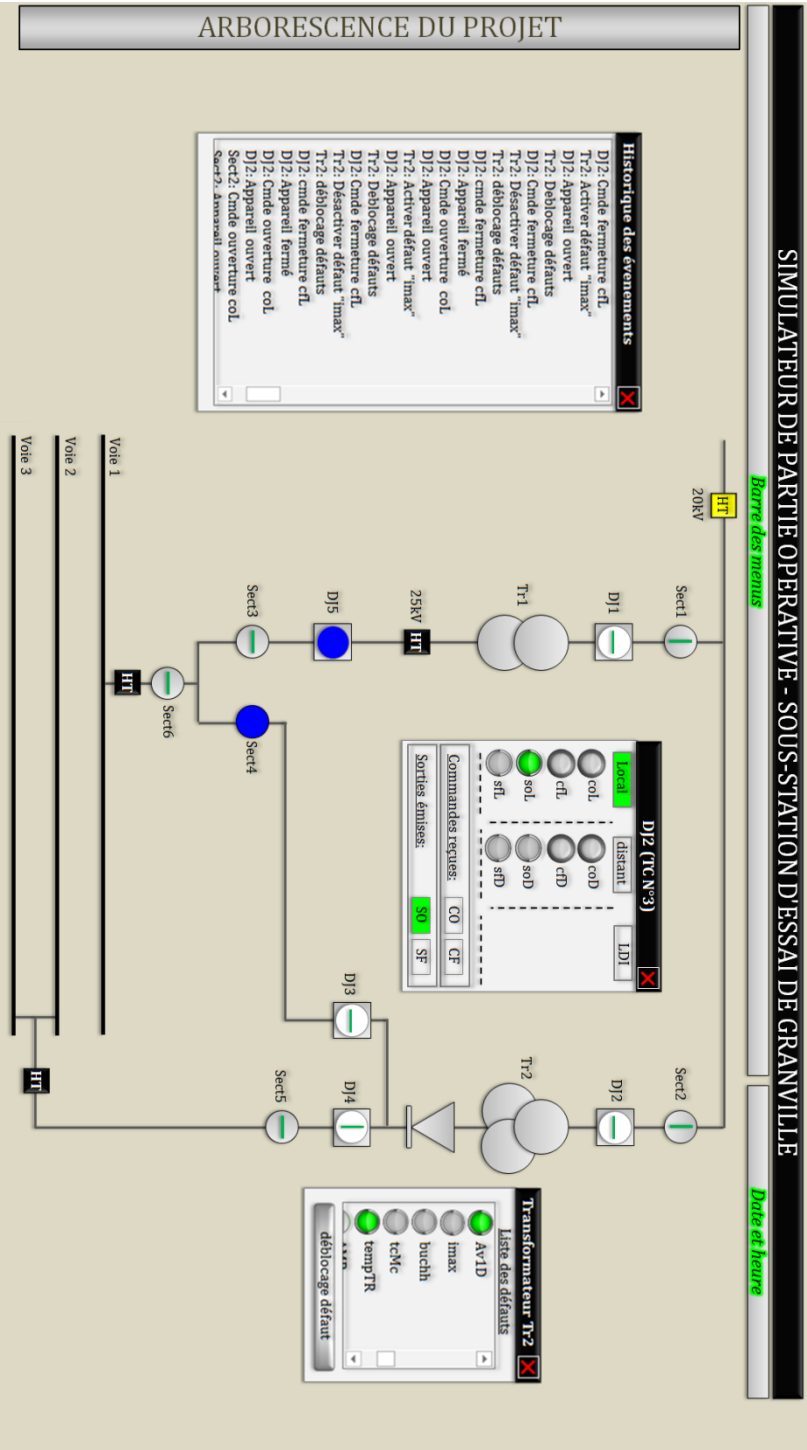


FIGURE 75 – Mode manuel du simulateur d'EALE

manœuvres. Par contre il aura accès à l'historique des événements pour pouvoir mieux suivre l'évolution de son installation. Ce mode permet de rejouer par exemple les scénarios dangereux précédemment retournés par le Model Checker lors de la vérification exhaustive de la sûreté de fonctionnement des programmes.

Durant la simulation, l'utilisateur doit également pouvoir accéder (en lecture et/ou en écriture selon les droits d'accès) aux tableaux de données suivants :

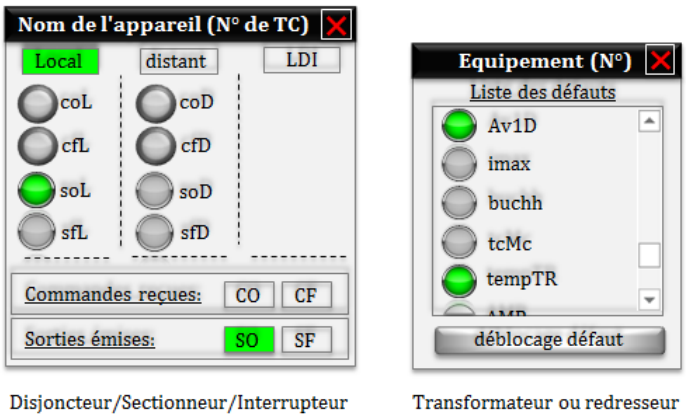


FIGURE 76 – Aperçu des panneaux de commandes des appareils

— tableau des données locales échangées avec le CLE,
— tableau des données distantes échangées avec le CSS via le RTU,
— tableau des données échangées avec les armoires de contrôle/commande/ protection (entrées/sorties câblées).

En mode HIL, lorsque le chargé d'études veut effectuer une commande d'ouverture d'un appareil (en mode local, via son panneau de commandes), l'ordre est transmis à l'automate de l'armoire de tests, puis ce dernier l'envoie à l'abonné automate (du système de contrôle commande) contrôlant cet appareil. La sortie calculée par l'abonné automate est ensuite transmise à l'appareil du simulateur d'EALE pour faire évoluer son état. En ce qui concerne l'activation d'un défaut depuis l'interface N^o1 en mode manuel, l'ordre est soit :

— transmis aux valises d'injection présentes dans l'armoire de tests par l'intermédiaire de l'automate (s'il s'agit d'un défaut déclenchable par injection). Ces valises précâblées au système de contrôle commande, injectent alors les bonnes valeurs de courant/tension (amplitude, phasage, temps...) de manière à activer le défaut concerné,
— transmis directement à l'automate sur lequel le défaut est câblé, s'il s'agit d'un défaut à shunt.

2.2.2 La simulation en mode « cahier de recettes »

Dans ce mode, l'utilisateur a accès à l'interface N^o1 définie à la figure 69 et pourra visualiser en même temps l'état de l'installation électrique et le contenu animé de la fiche de recettes en cours d'exécution. Il doit pouvoir choisir la manière dont le cahier de recettes sera exécuté, parmi les 3 suivantes :

— **Exécution pas à pas** des instructions du cahier de recettes, en partant de la première jusqu'à la dernière. Avec ce mode, l'exécution et la validation des instructions se fera une par une par l'utilisateur, qui devra valider un à un les résultats obtenus pour chaque instruction exécutée, afin de faciliter le suivi du déroulement des essais ;

— **Exécution semi-automatique** : l'exécution des instructions d'un scénario s'enchaîne automatiquement l'une après l'autre tant que les résultats attendus sont validés, mais le passage d'un scénario de tests au suivant doit d'abord être autorisé par l'utilisateur ;

— **Exécution automatique** : l'exécution de toutes les instructions du cahier de recettes s'enchaîne automatiquement, la fin de l'exécution d'une fiche N entraîne le début de l'exécution de la fiche passage N+1 (si sa condition initiale le permet).

Le choix du mode d'exécution peut être changé au cours de la simulation.

Avant de pouvoir exécuter un scénario de tests, le chargé d'études doit d'abord mettre l'installation dans les conditions initiales définies sur la fiche de tests, sans quoi l'exécution ne pourra pas démarrer. Le logiciel de Virtual Commissioning devra alors automatiquement remettre l'EALE dans les conditions initiales de démarrage des essais. La valeur booléenne de l'expression de la condition initiale doit être affichée sur la fiche concernée (TRUE ou FALSE). L'initialisation de la partie opérative peut également être faite manuellement à la demande du chargé d'études.

Prenons en exemple la fiche de tests nommée « **avarie diode TR2** » définie sur l'interface N^o1 à la figure 69. En analysant la valeur booléenne en ligne de la condition initiale, on s'aperçoit que celle-ci n'est pas satisfaite parce que le disjoncteur « DJ2 » n'est pas fermé (variable « **DJ2.sf** » surligné en rouge) et que le transformateur est en défaut (« **défaut** » surligné en vert). Il suffira alors :

— d'ouvrir la liste des défauts du transformateur (via son panneau de commandes) et de désactiver tous les défauts,

— d'ouvrir le panneau de commande du disjoncteur « DJ2 » et d'envoyer une commande de fermeture du disjoncteur (en mode distant, local, ou local individuel).

Cette manœuvre ramènera alors le système en conditions initiales d'exécution de la fiche de recettes. L'utilisateur pourra démarrer l'exécution en appuyant sur la touche «START» (s'il n'est pas en mode automatique).

L'action N^o1 de la fiche est d'abord exécutée, et la suivante ne pourra l'être que

lorsque le résultat attendu de la première instruction est obtenu. Il peut arriver pendant la simulation que l'exécution soit bloquée sur une instruction, dû à un résultat attendu non satisfait. Dans ce cas l'utilisateur devra avoir deux possibilités :

— stopper l'exécution de ce scénario puis en exécuter un autre : la fiche sera alors automatiquement marquée comme étant « bloquante » et l'utilisateur pourra la réexécuter ultérieurement,
— la mettre en pause, corriger le bug (au niveau des programmes automates en mode SIL, ou du câblage des armoires en mode HIL), puis la reprendre.

En supposant qu'une fiche de recettes N est en cours d'exécution, le passage à la fiche suivante ou précédente ne peut être autorisé tant que cette fiche N n'aura pas été stoppée. A la fin de l'exécution du scénario (qu'il soit bloquant ou non), l'utilisateur doit pouvoir ajouter un commentaire pour résumer par exemple le déroulement du scénario de tests, ou noter le(s) problème(s) qu'il a rencontré(s) pendant l'exécution.

Il peut arriver pendant les tests, que l'opérateur soit obligé d'interrompre ses essais et de quitter la simulation. Dans ce cas, le logiciel devra impérativement lui proposer de sauvegarder toutes les données de simulation, afin qu'il puisse reprendre les essais là où il les avait suspendus. Ce qui lui éviterait de tout reprendre à zéro.

A la fin des essais, l'utilisateur doit pouvoir générer automatiquement le compte rendu de recettes usines (inspiré du format de cahier de recettes actuellement utilisé par les chargés d'études de SNCF). En plus des informations générales de l'installation (schéma unifilaire, liste des entrées/sorties...), celui-ci doit contenir toutes les informations relatives à la recette usine pour chaque scénario de tests, telles que :

— le nom du test effectué,
— son numéro,
— son contenu (libellé des actions effectuées et résultats attendus),
— l'état de la fiche (correcte / bloquante / non exécutée),
— le nom de l'exécutant (le chargé d'études),
— ses commentaires sur la fiche,
— la date et l'heure du test,
— ...

2.3 Description du mode Software-In-the-Loop (SIL) pour la validation automatique des programmes API

En dehors de son utilisation en usine pour la validation automatique du câblage des armoires (mode HIL - figure 69), le logiciel simulateur d'EALE doit pouvoir être utilisé pour valider uniquement des programmes automates transférés et exécutés dans un ou plusieurs émulateurs d'automates (mode SIL - figure 77). Ce mode SIL permet au chargé d'études de valider automatiquement ses programmes automates bien avant le début de la recette usine. Pour cela, le logiciel de Virtual Commissioning doit être connecté au(x) émulateur(s) d'automate(s) de STRATON intégré(s) dans le même ordinateur de simulation. Par conséquent, il doit être compatible avec le protocole de communication type « mémoire partagée » auquel les émulateurs d'automates STRATON sont déjà compatibles.

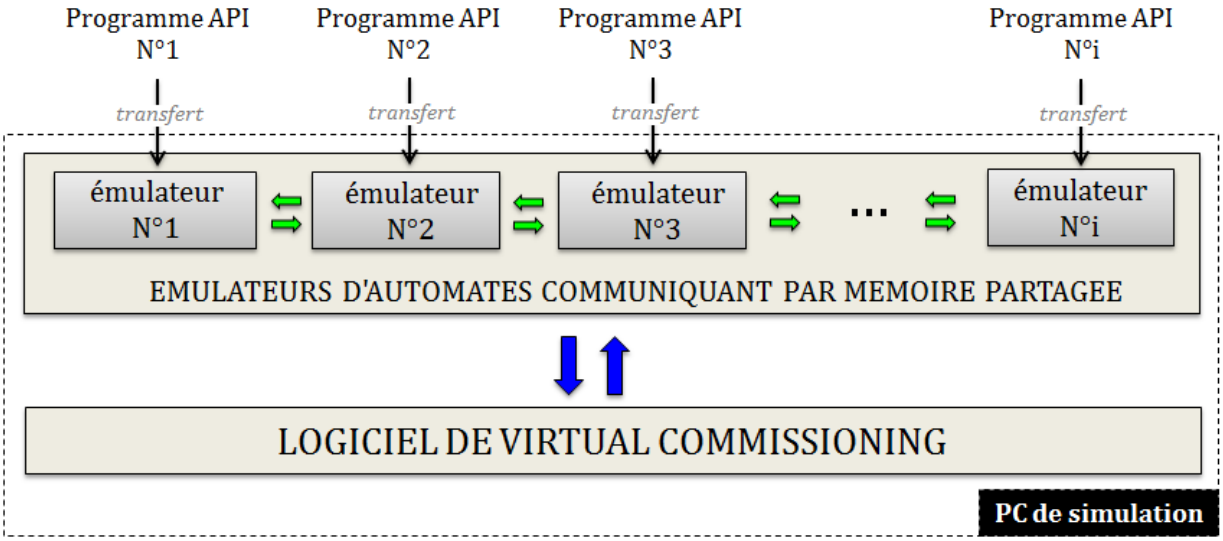


FIGURE 77 – Architecture du mode SIL

En plus de permettre aux chargés d'études de valider les programmes automates en amont, ce mode SIL est particulièrement très bénéfique pour la formation des nouveaux agents SNCF, grâce à son interface graphique simulant l'installation.

Les données générées par ODIL pour la validation automatique du câblage des armoires de contrôle commande (figure 72) sont réutilisées dans ce mode SIL (figure 78) à quelques différences près :

— Les programmes automates des EALE sont transférés et exécutés dans les émulateurs d'automates du PC de simulation, et non dans les vrais automates du système de contrôle commande. ODIL génère alors, à la demande, la configuration

matérielle des programmes automates pour chacun des modes de simulation ;

— En mode SIL, le fichier de paramétrage des variables du logiciel de Virtual Commissioning n'est pas le même que pour le mode HIL, car les échanges entre les émulateurs d'automates et le logiciel simulateur d'EALE se font en interne à travers une table commune (mémoire partagée). Par conséquent, ODIL génère un fichier de paramétrage pour chacun des deux modes ;

— Le programme API généré pour l'armoire de tests n'est pas nécessaire dans ce mode SIL.

Ces particularités sont donc prises en compte par ODIL au moment de la génération, en fonction du choix du mode de simulation HIL ou SIL.

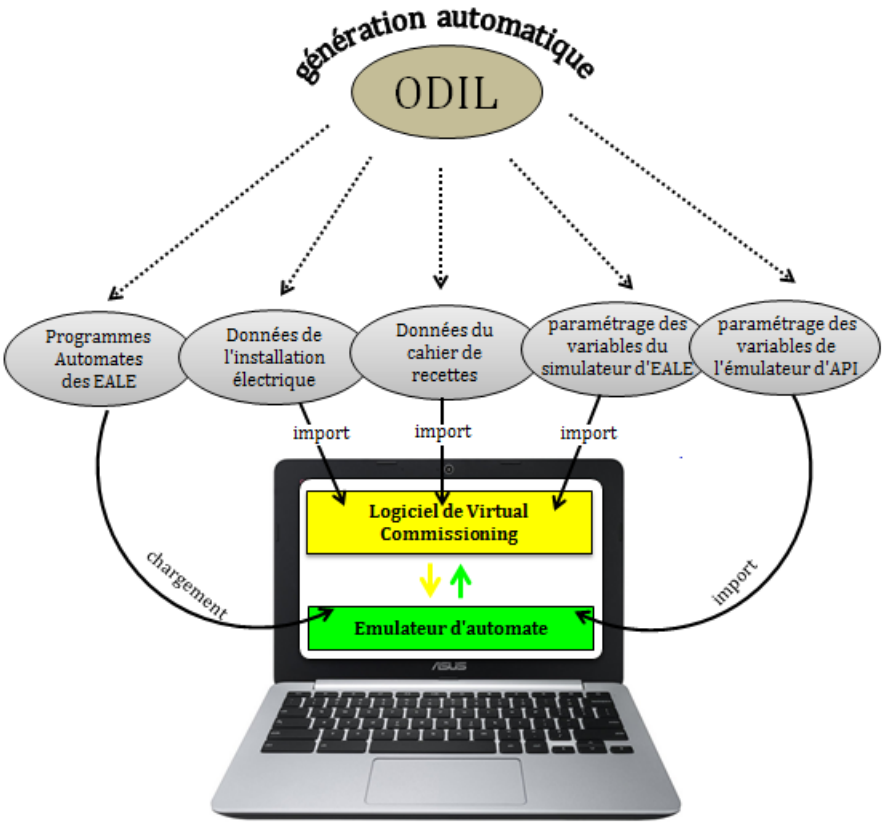


FIGURE 78 – Génération automatique des données d'entrées pour le mode SIL

Après la spécification fonctionnelle du banc d'essais répondant aux attentes des chargés d'études, un appel d'offre a été lancé pour la mise en oeuvre de la solution proposée. Suite à cet appel d'offre et après examen des réponses techniques reçues, la solution technique répondant le mieux au cahier des charges a été retenue.

3 Mise en oeuvre de la solution proposée : réponse technique de l'appel d'offre

La solution technique proposée suite à l'appel d'offre résulte de la collaboration entre les sociétés **Fournié Grospaud Énergie** (*www.fgsynerys.fr*) et **Prosyst** (*www.prosyst.fr*) qui s'occupent respectivement de la conception de l'armoire de tests et du logiciel de Virtual Commissioning.

3.1 Présentation de la société « Fournié Grospaud Énergie »

La société « Fournié Grospaud Énergie » est créée en 1925 par les ingénieurs Raymond Fournié et Henri Grospaud. Elle est spécialisée dans la conception, le développement et la rénovation des systèmes de contrôle commande destinés au pilotage des équipements d'alimentation des lignes électrifiées (EALE). Elle fait partie des intégrateurs sollicités par la SNCF lors de la conception du système de contrôle commande. Leur périmètre d'intervention couvre la conception et la mise au point matérielle et logicielle du système d'automatisme, l'étude et l'intégration des différents équipements dans les armoires, ainsi que les phases de recette usine et de mise en service sur site. Elle est également spécialisée dans la conception de banc d'essais automatisé (maquettes relais, maquettes électroniques programmables, applicatifs d'automatismes spécifiques, logiciels d'émulation de protocoles...). En plus d'avoir une bonne maîtrise des systèmes de contrôle commande de la SNCF ainsi que du métier des chargés d'études (recette usine), leur compétence technique convient parfaitement à la réalisation du banc d'essais. Toutefois en ce qui concerne le logiciel de Virtual Commissioning, la société « Fournié Grospaud Énergie » a choisi de faire appel à un partenaire spécialisé dans les applications de Virtual Commissioning. Il s'agit de la société Prosyst, qui a également déjà collaboré avec la SNCF dans le cadre du développement de l'outil de génération automatique ODIL.

3.2 Présentation de la société « Prosyst »

La société Prosyst est une PME créée en 1986 et spécialisée dans la performance industrielle. Elle développe et commercialise des produits et services dans le domaine des automatismes industriels. Les solutions de PROSYST sont basées sur des concepts innovants, brevetés pour certains d'entre eux :

— modélisation et simulation de parties opératives,
— méthodes de surveillance non intrusives,
— diagnostic de pannes,
— calcul et optimisation de la performance.

L'apport des réalisations de PROSYST est significatif en termes d'augmentation de qualité et de performance des automatismes industriels. Aujourd'hui, les solutions PROSYST issues de son fort engagement dans la recherche et le développement, ont donné lieu à d'étroits partenariats avec des acteurs majeurs du secteur ferroviaire comme BOMBARDIER TRANSPORT et la SNCF (avec ODIL). De ce fait, elle a une maîtrise parfaite des installations électriques, des programmes automates de la SNCF, et du logiciel ODIL.

Dans le cadre du projet du banc d'essais, PROSYST propose son savoir-faire dans le domaine de la simulation de partie opérative autour de son offre **SIMAC** (*http ://www.prosyst.fr/simulation/vf/simac.htm*), qui est un logiciel de simulation de machines, process et infrastructures, destiné à fiabiliser les systèmes de contrôle commande et les IHM. Il permet d'assister l'automaticien en bureau d'étude pendant les phases de tests, de mise au point et de validation, ainsi que pour la formation des opérateurs. L'outil SIMAC sera bien entendu réadapté en fonction des spécifications fonctionnelles décrits en section 2.

3.3 Proposition technique

La réponse technique des sociétés « Fournié Grospaud Énergie » et « Prosyst » intègre deux solutions techniques, identiques sur le plan fonctionnel, mais qui différent selon l'architecture matérielle du banc d'essais :

Solution N^o1 : banc d'essais avec armoire de tests (*conformément à la figure 67*)
Cette solution couvre toutes les demandes exprimées, elle est donc conforme au cahier des charges sur les plans fonctionnels et constitutionnels. L'architecture proposée présente néanmoins quelques inconvénients :

— Les dimensions de l'armoire rendent cette dernière difficile à déplacer d'un site à un autre (colis lourd, encombrant et relativement fragile) ;
— La conception monolithique du banc d'essais impose un positionnement unique de l'armoire de tests, et par conséquent la nécessité d'une longueur importante des cordons de raccordements entre l'armoire de tests et le système de contrôle

commande à tester. Le rangement de l'ensemble de ces cordons sera donc mal aisé et encombrant.

Pour pallier ces inconvénients, une deuxième solution technique est proposée.

Solution N^o2 : armoire de tests remplacée par des valises robustes (*figure 79*)
Cette solution technique reprend les mêmes caractéristiques logicielles et fonctionnelles de l'offre de base. Elle reste donc entièrement conforme aux exigences exprimées dans le cahier des charges. Son principe consiste à remplacer l'armoire de tests par un jeu de valises robustes type **Pelicase** (figure 79) réparties comme suit :

— La valise principale : qui renferme un automate programme compatible avec Straton, un PC portable équipé du logiciel de Virtual Commissioning, et un switch Giga Ethernet 24 ports. Elle est connectée aux autres valises ;

— La Valise passerelle HNZ, pour simuler les échanges avec le Central Sous-Station via le protocole de communication HNZ. Elle est connectée au RTU du système de contrôle commande ;

— La valise à injection, pour simuler des défauts sur l'EALE virtuelle ;

— La valise de rangement (non visible sur la figure 79) qui permet de ranger les modules d'entrées/sorties déportés et les câbles de connexion. Ces modules d'entrées/sorties sont directement connectés aux borniers des armoires électriques. Ils sont reliés en série par une liaison IP, et connectés à la valise principale.

Les avantages de cette solution sont multiples :

— Les valises sont déplaçables à la main par un opérateur, et peuvent être transportables avec un véhicule léger ;

— Les valises sont robustes, étanches et résistantes aux chocs et aux environnements difficiles ;

— Les valises sont de taille modeste et se logent donc aisément autour du châssis à tester, en fonction de la configuration et de la taille de la salle de test. L'ergonomie de travail se trouve ainsi grandement améliorée.

— Les modules d'entrées/sorties indépendants peuvent être installés au plus près des borniers châssis. Il n'est donc plus nécessaire de réaliser des cordons de longueur importante. Ces cordons de faible longueur peuvent donc se ranger dans la même valise que les modules.

— Les modules d'entrées/sorties intègrent la fonction switch. Ils sont donc reliés entre

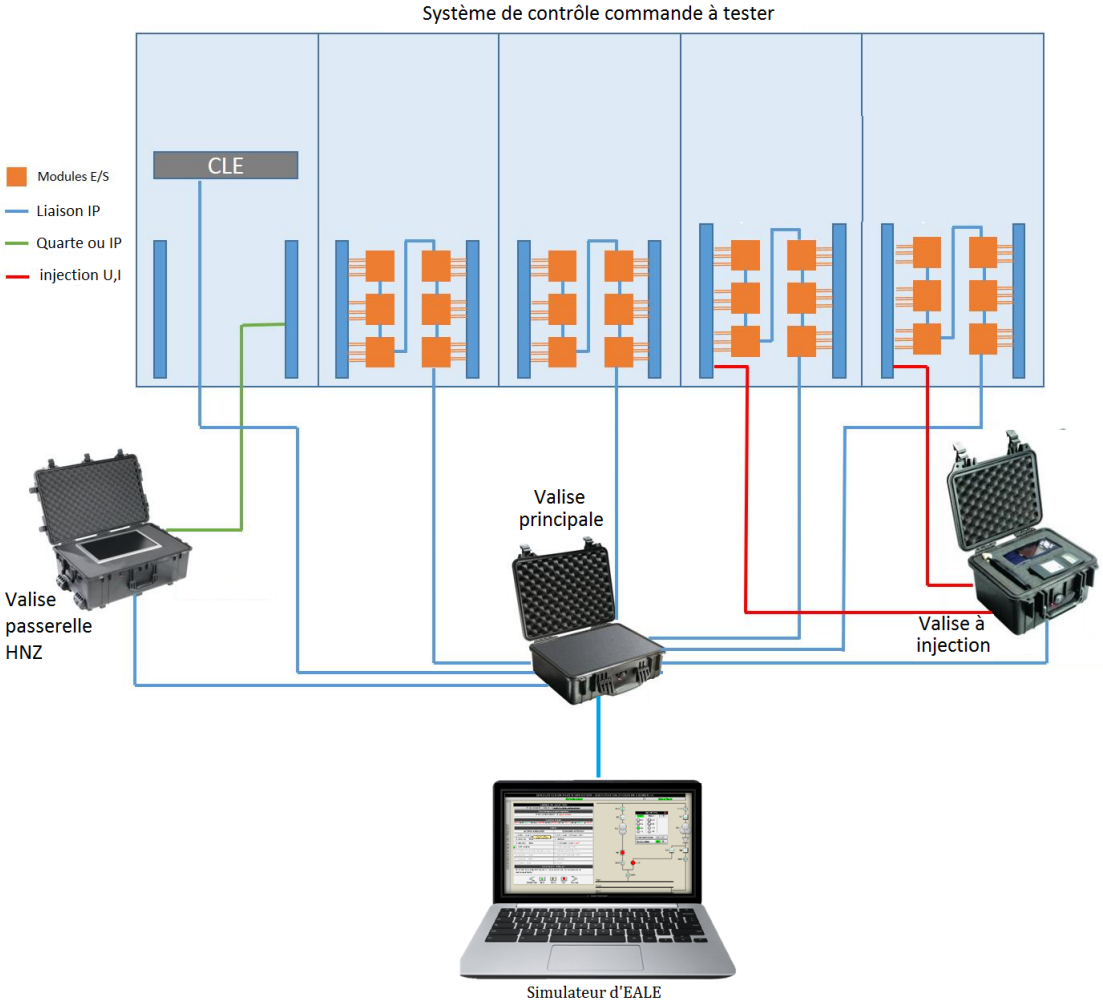


FIGURE 79 – Architecture matérielle de la solution N^o2

eux en série par liaison IP au sein d'une armoire électrique. Ces modules sont ensuite connectés à la valise principale à l'aide d'un seul câble RJ45.

Après examen des deux solutions techniques, la deuxième a été retenue. Malgré que son architecture matérielle est différente de celle présentée dans le cahier des charges (figure 67), les fonctionnalités restent les mêmes et elle reste moins encombrante que la solution N^o1 avec l'armoire de tests.

En ce qui concerne le logiciel de Virtual Commissioning, Prosyst propose son offre SIMAC pour la modélisation et la simulation des EALE. Même si les fonctionnalités actuelles de cet outil ne correspondent pas exactement à celles décrites dans le cahier des charges, le noyau de SIMAC sera réadapté par Prosyst pour répondre aux besoins des chargés d'études (Interfaces Homme Machines, fonctionnalités...). À l'instar de la vérification formelle des programmes automates dans Uppaal, la modélisation des EALE dans le logiciel de Virtual Commissioning consiste dans un premier temps à modéliser l'ensemble

des constituants d'une EALE. Par la suite, l'instanciation automatique (par ODIL) des modèles constituants l'EALE cible permettra de reconstituer l'installation dans le logiciel de simulation de partie opérative. À cette fin, un deuxième cahier des charges a été transmis à Prosyst qui est responsable de la modélisation (et de l'intégration des modèles sous forme de *templates* dans ODIL). Ce cahier des charges décrit de manière structurelle et fonctionnelle l'ensemble des équipements génériques présents dans une EALE. L'annexe C présente le contenu de ce cahier des charges.

Les sociétés **Fournié Grospaud Énergie** et **Prosyst** disposent aujourd'hui des éléments nécessaires pour la conception du banc d'essais. Les tests en usine du premier prototype sont prévus au premier trimestre de l'année 2019.

4 Conclusion

La validation automatique des systèmes de contrôle commande des EALE représente une solution innovante permettant d'améliorer le workflow actuel des chargés d'études du département Ingénierie & Projets de la SNCF. Cette technique basée sur l'utilisation du Virtual Commissioning (en modes SIL et HIL) s'applique en deux temps :

— Validation automatique des programmes API par Simulation Software-In-the-Loop (SIL) ;
— Validation automatique du câblage des armoires par Simulation Hardware-In-the-Loop (HIL).

Cette approche permet de séparer la validation des programmes automates à celle du câblage des armoires de contrôle commande en usine. Après la vérification formelle des programmes automates, le chargé d'études procède (également depuis son bureau) à la validation automatique des programmes API par simulation SIL (figure 77). Quant à la validation automatique du câblage des armoires, cette tâche est vouée à être externalisée. En effet, cette dernière nécessite des déplacements en usine pour le chargé d'études, ce qui le rend indisponible pour réaliser d'autres tâches. Cette mission, qui nécessite principalement une simple supervision du déroulement des tests automatisés grâce à l'autonomie du banc d'essais, sera désormais laissée à la charge de l'intégrateur ayant conçu et câblé les armoires de contrôle commande. Le chargé d'études vérifie et valide ses programmes automates, puis les envoie (en même temps que les données du simulateur d'EALE générées par ODIL) à l'intégrateur, qui sera déjà en possession du banc d'essais pour valider le

câblage. Après cela, le système de contrôle commande sera prêt pour une mise en service sur site.

Conclusion générale et perspectives

Durant les projets d'automatisation des EALE, le workflow des chargés d'études automatisme est globalement constitué de deux phases : la conception et la V&V du système de contrôle commande. La phase de conception consiste à développer les livrables associés au projet, à savoir les schémas électriques, les programmes automates, et le cahier de recettes permettant de valider le système de contrôle commande en usine. Initialement, ces livrables sont manuellement conçus par les chargés d'études. Pour cela, ils réutilisaient des livrables dont l'EALE associée est similaire à celle en cours d'étude, et les adaptaient en fonction du nouveau cahier des charges. Pour les chargés d'études, cette approche requiert moins d'effort que de partir de zéro pour concevoir ces livrables, mais est cependant sujette aux erreurs humaines. La deuxième phase consiste à valider le système de contrôle commande en usine. Après relecture et correction des livrables générés, les programmes automates sont transférés dans les automates du système de contrôle commande, et les procédures de tests du cahier de recettes sont utilisées pour valider les programmes automates et le câblage des armoires en usine.

Ces deux étapes, en plus d'être complexes, demandent beaucoup de temps et de ressources de la part des chargés d'études durant les projets. La fatigue générée par cette surcharge de travail, ainsi que le stress lié aux deadlines imposées par la direction Ingénierie & Projets pour des raisons de productivité, exposent le chargé d'études à une surcharge mentale. Par conséquent, ces mauvaises conditions de travail ne le mettent pas à l'abri d'erreurs humaines durant les phases de conception et de V&V, ce qui peut compromettre la fiabilité des systèmes de contrôle commande.

Pour remédier à ces inconvénients, les travaux de recherche de Coupat (2014) ont permis d'améliorer la phase 1 du workflow des chargés d'études (conception des livrables du système de contrôle commande). Cette étude a abouti au développement d'un outil nommé ODIL, qui permet de générer automatiquement les livrables d'un projet (programmes automates, cahier de recettes, schémas...) à partir de la description du projet

dans l'interface du logiciel. Cette nouvelle méthodologie a été adoptée depuis lors par les chargés d'études, et leur a permis de gagner deux fois plus de temps durant la phase 1. Néanmoins, cette solution n'a eu d'impact que sur la phase 1 du workflow traditionnel. La phase de V&V du système de contrôle commande n'a fait l'objet d'aucune amélioration en dépit de sa complexité et des nombreux inconvénients qu'elle présente (essais manuels sujets à des erreurs humaines, manque d'exhaustivité, longue durée des essais...).

Pour supprimer toutes ces contraintes liées à la phase de V&V, les travaux menés dans ce projet de recherche ont permis de proposer une approche méthodologique dédiée aux chargés d'études automatisme de la SNCF, pour la vérification et la validation des systèmes de contrôle commande des EALE. Cette méthodologie est basée sur l'utilisation des méthodes formelles et du Virtual Commissioning. Elle est constituée des étapes suivantes :

- **Étape 1** : Vérification formelle (hors ligne) des spécifications fonctionnelles et de la sûreté de fonctionnement des programmes API ;
- **Étape 2** : Validation automatique des programmes automates par Simulation *Software-In-the-Loop* (en ligne) ;
- **Étape 3** : Validation automatique du câblage des armoires en usine par simulation *Hardware-In-the-Loop* (en ligne).

En combinant l'utilisation du Model-Checking pour la vérification et du Virtual Commissioning pour la validation, on aboutit à une méthodologie formelle, automatisée, et fiable. Le Model-Checking remédie au manque d'exhaustivité de l'approche traditionnel, car il permet de vérifier formellement que les programmes automates respectent les spécifications fonctionnelles tout en garantissant la sûreté de fonctionnement de l'installation électrique. De plus, dans le cas où les programmes ne sont pas sûrs de fonctionnement, l'implémentation du filtre de commande permet de remédier à ce problème. Toutefois, la fiabilité des modèles de vérification ainsi que l'explosion combinatoire dans certains cas représentent les principaux inconvénients de cette technique. Quant au Virtual Commissioning, il assure l'automatisation de la validation du système de contrôle commande ainsi que la traçabilité des essais grâce à la génération de compte rendu. Sa convivialité facilite le déroulement des essais, et permet également de former de nouveaux chargés d'études afin de les préparer à des situations réelles.

Le département Ingénierie & Projets envisage d'externaliser la mission de validation du câblage des armoires de contrôle commande en usine, en la confiant aux intégrateurs responsables de la réalisation et du câblage des armoires électriques. En effet, cette

mission nécessite aujourd'hui la présence en usine des chargés d'études responsables de la conception des programmes automates. Grâce à cette nouvelle approche, les chargés d'études pourront vérifier (par Model-Checking) et valider automatiquement (par simulation Software-In-the-Loop) les programmes automates dès leur conception. Ensuite, ils pourront transmettre les programmes validés et les données du logiciel de Virtual Commissioning aux intégrateurs. Ces derniers transféreront ces programmes validés vers le système de contrôle commande en usine, puis procéderont à la validation du câblage des armoires électriques en utilisant le Virtual Commissioning en mode Hardware-In-the-Loop. La figure 80 résume la répartition des tâches pour la V&V des systèmes de contrôle commande des EALE.

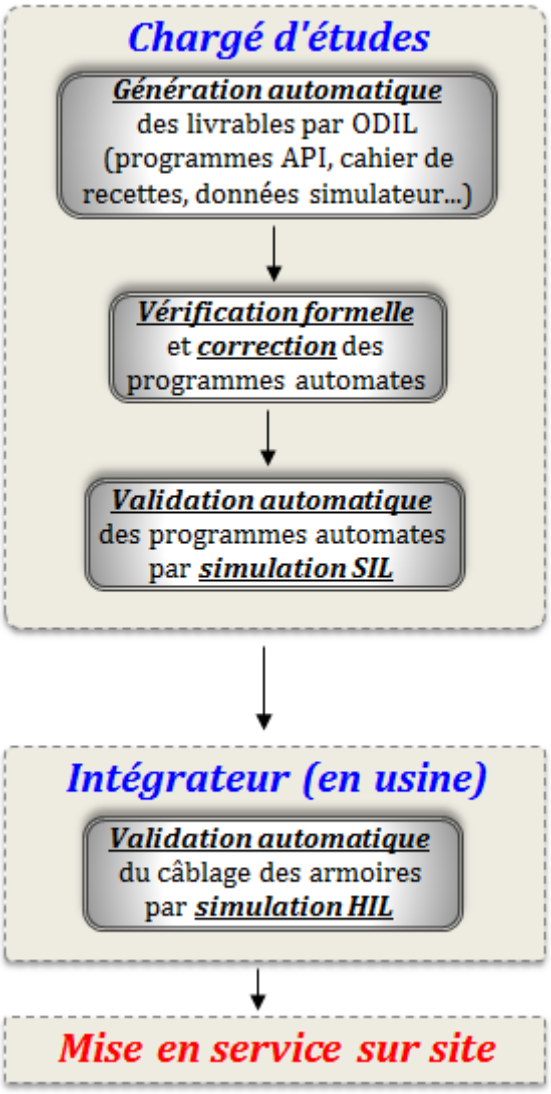


FIGURE 80 – Répartition des tâches du Workflow final

Cette nouvelle méthodologie présente de nombreux avantages pour les chargés d'études, parmi lesquels on peut citer :

— **La validation séparée** des programmes (mode SIL) puis du câblage des armoires (mode HIL), contrairement à la méthodologie traditionnelle des chargés d'études. Cette nouvelle approche permet de réduire considérablement la complexité du diagnostic et de la correction des erreurs de conception du système de contrôle commande en usine (chapitre 2, section 2.6.2) ;
— **Un gain de temps durant la recette usine** grâce à la validation en amont des programmes API, et à l'automatisation de la méthodologie. Les chargés d'études estiment que le banc d'essais permettra de valider le système de contrôle commande en 48 heures au maximum, contre 1 à 2 semaines avec la méthode actuelle (chapitre 2, section 2.6.1) ;
— **La convivialité du banc d'essais** grâce au logiciel de Virtual Commissioning ;
— **La fiabilité** et la **traçabilité** des essais grâce à la génération automatique de compte rendu après la validation ;
— **La suppression des déplacements en usine des chargés d'études** : grâce à l'externalisation de la mission de validation du câblage des armoires, ce qui représente un gain énorme pour le département Ingénierie & Projets de la SNCF ;
— **L'augmentation de la sécurité** (grâce à la vérification formelle), **la diminution des coûts des essais**, et **la formation des nouveaux opérateurs** grâce au Virtual Commissioning constituent les autres avantages de cette méthode.

Les chargés d'études automatisme de la SNCF ne sont pas les seuls à recourir à des approches de V&V des systèmes de contrôle commande à travers des tests manuels (basés sur un cahier de recettes). Il existe en effet beaucoup d'industriels qui ont recours à cette méthode (Bjørner and Havelund, 2014), et qui souffrent par conséquent des mêmes problèmes. La méthodologie de V&V développée dans ce projet de recherche peut par conséquent s'appliquer dans des domaines autres que le ferroviaire, à l'instar des systèmes manufacturiers en industrie.

Deux directions sont envisageables pour la poursuite de ces travaux de recherche. La première est liée à la modélisation des programmes automates dans le Model-Checker Uppaal. En effet, lors de la vérification formelle, ces programmes sont manuellement traduits en équations algébriques dans le Model-Checker. Cette traduction manuelle peut être source d'erreurs, et par conséquent, les programmes formellement vérifiés peuvent ne pas correspondre aux programmes d'origine. Même si les chargés d'études sont aujourd'hui convaincus par l'efficacité de notre approche formelle (à travers ses exemples

d'applications), 90% d'entre eux estiment qu'ils ne peuvent l'utiliser par eux-mêmes que si la traduction des programmes en équations algébriques est automatisée, afin d'éviter toutes erreurs pouvant compromettre la fiabilité de la vérification. Dans ce sens, nous prévoyons de mettre en place un outil qui permettra de traduire automatiquement ces programmes automates en équations algébriques. De plus, après vérification et correction des programmes automates « modélisés » dans Uppaal, les chargés d'études procèdent au reverse engineering afin de reporter ces mêmes corrections sur les programmes automates d'origine (générés par ODIL). Pour éviter également les erreurs humaines lors de cette phase, l'outil prévu pour la traduction automatique des programmes automates en équations algébriques, devra également pouvoir assurer la traduction dans le sens inverse (équations algébriques vers programmes API en langage automate). Cet apport permettra de faciliter l'utilisation de cette technique aux chargés d'études, tout en minimisant le risque d'erreurs.

L'autre direction envisageable pour la poursuite de ces travaux est également liée à la vérification formelle des programmes, et concerne les traces retournées par le Model-Checker après la vérification formelle de la sûreté de fonctionnement des programmes. Ces traces renferment comme information les scénarios dangereux pouvant amener le système à violer les contraintes de sécurité. Elles sont exportables en fichiers texte depuis le Model-Checker (après la vérification). Lors de la validation automatique des programmes API (simulation SIL), il est important pour le chargé d'études de rejouer ces « scénarios dangereux » sur le système pour vérifier « en ligne » que le filtre de sécurité garantit bien la sécurité. Dans ce sens, nous envisageons une solution de génération automatique de nouvelles fiches de tests à partir de ces traces exportées au format texte. Cela permettra d'enrichir la base de données des fiches de tests du cahier de recettes.

Aujourd'hui, le département I&P.TE de la SNCF est satisfait de l'approche méthodologique de V&V proposée dans ce travail de recherche, et n'a pas hésité à investir le budget nécessaire pour la réalisation du banc d'essais. 100% des chargés d'études automatisme de la SNCF estiment que grâce à cette solution, les tests de validation des programmes et du câblage pourront être effectués beaucoup plus efficacement en 48 heures maximum, sans compter les autres avantages de cette approche (cités plus haut). Toutefois, le département I&P.TE n'est pas tout à fait prêt à adopter la solution du filtre de sécurité, car même si son efficacité a été formellement prouvée, le filtre peut dans certains cas modifier, en cas de besoin, le comportement fonctionnel d'un programme automate pour garantir

la sécurité (comme décrit à la section 3.3.2 du chapitre 3).

La contribution majeure de ce travail de recherche est à la fois méthodologique et théorique. Comparée aux méthodes traditionnelles de V&V, l'approche proposée dans ce projet est structurée car elle vérifie et valide séparément les programmes API dès leur conception, puis le câblage des armoires en usine, de manière automatisée. De plus, elle garantit formellement la sûreté de fonctionnement du système de contrôle commande grâce à la vérification formelle et à l'implémentation d'un filtre de sécurité.

Ce travail a également permis de montrer l'intérêt des méthodes formelles dans le contexte industriel, malgré qu'elles y soient très peu connues et utilisées. De plus en combinant ces méthodes formelles à d'autres techniques comme le Virtual Commissioning, on aboutit à une approche méthodologique de V&V formelle, automatisée, fiable et surtout utilisable.

Bibliographie

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

61131-3, I. (2013). Programmable controllers – Part 3 : Programming languages, Reference number CEI/IEC 601131.

Admin, A. (2014). Automated Testing vs Manual Testing : Which Should You Use, and When ?

Alur, R. and Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235.

ANSI, I. S. (1983). IEEE 729-1983 - IEEE Standard Glossary of Software Engineering Terminology.

Baccari, S., Cammeo, G., Dufour, C., Iannelli, L., Mungiguerra, V., Porzio, M., Reale, G., and Vasca, F. (2012). Real-Time Hardware-in-the-Loop in Railway : Simulations for Testing Control Software of Electromechanical Train Components. *Railway Safety, Reliability, and Security : Technologies and Systems Engineering*, pages 221–248.

Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. The MIT Press, Cambridge, Mass. OCLC : ocn171152628.

Bauer, N., Engell, S., Huuck, R., Lohmann, S., Lukoschus, B., Remelhe, M., and Stursberg, O. (2004). Verification of PLC programs given as sequential function charts. In *SoftSpez Final Report*, pages 517–540. Springer.

Bender, D. F., Combemale, B., Crégut, X., Farines, J. M., Berthomieu, B., and Vernadat, F. (2008). Ladder metamodeling and plc program validation through time petri nets. In Schieferdecker, I. and Hartman, A., editors, *Model Driven Architecture – Foundations and Applications*, pages 121–136, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bjørner, D. and Havelund, K. (2014). 40 Years of Formal Methods. Lecture Notes in Computer Science, pages 42–61. Springer, Cham.

Boehm, B. W. (1979). Software Engineering-as It is. ICSE '79, pages 11–21, Piscataway, NJ, USA. IEEE Press.

Buxton, J. N. and Randell, B., editors (1970). *Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee*, Brussels 39, Belgium. NATO, Science Committee. Rome, Italy, 27^{th} to 31^{th} October 1969 ; `http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF` – geprüft : 28. März 2004.

Bérard, B. and Schnoebelen, P. (1999). *Vérification de logiciels : techniques et outils du model-checking*. Vuibert.

Canet, G., Couffin, S., Lesage, J. J., Petit, A., and Schnoebelen, P. (2000). Towards the automatic verification of PLC programs written in Instruction List. In *2000 IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 2449–2454 vol.4.

CENELEC (2012). CENELEC - EN 50128 - Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems | Engineering360.

Chow, T. S. (1978). Testing Software Design Modeled by Finite-State Machines. *IEEE Transactions on Software Engineering*, SE-4(3) :178–187.

Chériaux, F., Picci, L., Provost, J., and Faure, J.-M. (2010). Conformance test of logic controllers of critical systems from industrial specifications.

Clarke, E. M. and Emerson, E. A. (1982). Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, Berlin, Heidelberg. Springer-Verlag.

Clarke, E. M. and Wing, J. M. (1996). Formal Methods : State of the Art and Future Directions. *ACM Computing Surveys*, 28 :626–643.

Clarke, Jr., E. M., Grumberg, O., and Peled, D. A. (1999). *Model Checking*. MIT Press, Cambridge, MA, USA.

Constant, C., Jéron, T., Marchand, H., and Rusu, V. (2007). Integrating formal verification and conformance testing for reactive systems. *IEEE Transactions on Software Engineering*, 33(8) :558–574.

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 Électrifiées.* PhD thesis, Université de Reims - Champagne Ardenne, Reims.

Coupat, R., Philippot, A., Niang, M., Courtois, C., Annebicque, D., and Riera, B. (2018). Methodology for Railway Automation Study and Automatic Generation of PLC Programs. *IEEE Intelligent Transportation Systems Magazine*, pages 1–1.

Di Tommaso, P., Flammini, F., Lazzaro, A., Pellecchia, R., and Sanseviero, A. (2005). (4) The simulation of anomalies in the functional testing of the ERTMS/ETCS trackside system.

Drath, R., Weber, P., and Mauser, N. (2008). An evolutionary approach for the industrial introduction of virtual commissioning. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pages 5–8.

Duffy, D. A. (1991). *Principles of Automated Theorem Proving.* John Wiley & Sons, Inc., New York, NY, USA.

Fantechi, A. (2014). Twenty-five years of formal methods and railways : What next ? In *Revised Selected Papers of the SEFM 2013 Collocated Workshops on Software Engineering and Formal Methods - Volume 8368*, pages 167–183, Berlin, Heidelberg. Springer-Verlag.

Fantechi, A., Flammini, F., and Gnesi, S. (2014). Formal methods for railway control systems. *Int. J. Softw. Tools Technol. Transf.*, 16(6) :643–646.

Fantechi, A., Fokkink, W., and Morzenti, A. (2012). Some Trends in Formal Methods Applications to Railway Signaling. In *Formal Methods for Industrial Critical Systems*, pages 61–84. Wiley-Blackwell.

Fernandez, B., Blanco, E., and Merezhin, A. (2014). Testing & verification of PLC code for process control. page 5.

Ferrari, A., Fantechi, A., Gnesi, S., and Magnani, G. (2013). Model-based development and formal methods in the railway industry. *IEEE Softw.*, 30(3) :28–34.

Ferrari, A., Magnani, G., Grasso, D., and Fantechi, A. (2011). Model checking interlocking control tables. In Schnieder, E. and Tarnai, G., editors, *FORMS/FORMAT 2010*, pages 107–115, Berlin, Heidelberg. Springer Berlin Heidelberg.

Fitting, M. (1996). *First-Order Logic and Automated Theorem Proving*. Texts in Computer Science. Springer-Verlag, New York, 2 edition.

Forsberg, K. and Mooz, H. (1991). The Relationship of System Engineering to the Project Cycle - INCOSE International Symposium.

Frey, G. and Litz, L. (1998). Verification and Validation of Control Algorithms by Coupling of Interpreted Petri Nets. In *IEEE conference on Systems Man and Cybernetics, IEEE SMC'98*, pages 7–12.

Frey, G. and Litz, L. (2000a). Correctness analysis of Petri net based logic controllers. In *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, volume 5, pages 3165–3166 vol.5.

Frey, G. and Litz, L. (2000b). Formal methods in PLC programming. In *2000 IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 2431–2436 vol.4.

Hassapis, G., Kotini, I., and Doulgeri, Z. (1998). Validation of a SFC Software Specification by Using Hybrid Automata. *IFAC Proceedings Volumes*, 31(15) :107–112.

Haxthausen, A. E., Le Bliguet, M., and Kjær, A. A. (2010). Modelling and verification of relay interlocking systems. In Choppy, C. and Sokolsky, O., editors, *Foundations of Computer Software. Future Trends and Techniques for Development*, pages 141–153, Berlin, Heidelberg. Springer Berlin Heidelberg.

Hetzel, W. (1984). The complete guide to software testing de William C Hetzel : QED Information Sciences 9780894351105 Paperback - Ergodebooks.

IEC-60848 (2002). IEC 60848 : Programmable controllers – Part 3 : Programming languages GRAFCET specification language for sequential function charts Reference number CEI/IEC 60848.

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

ISA-62381 (2012). ansi/isa 62381 : factory acceptance test (fat), site acceptance test (sat), and site integration test (sit), automation systems in the process industry.

ISO-9000 (2015). ISO 9000, 2015 - Systèmes de management de la qualité — Principes essentiels et vocabulaire.

Jiang, J. and Holding, D. (1996). The formalisation and analysis of Sequential Function Charts using a Petri net approach.

Kesraoui, S. (2017). *Intégration des techniques de vérification formelle dans une approche de conception des systèmes de contrôle-commande : application aux architectures SCADA*. Lorient.

Kleijn, C. (2014). Introduction to hardware-in-the-loop simulation. *URL : http ://www. hil-simulation. com/images/stories/Documents/Introduction% 20to% 20Hardware-in-the-Loop% 20Simulation. pdf (visited on 01/21/2016)*.

Koo, L.-J., Park, C. M., Lee, C. H., Park, S., and Wang, G.-N. (2011). Simulation framework for the verification of PLC programs in automobile industries. *International Journal of Production Research*, 49(16) :4925–4943.

Kowalewski, S. and Preußig, J. (1996). Verification of sequential controllers with timing functions for chemical processes.

Lampérière-Couffin, S., Rossi, O., Roussel, J. M., and Lesage, J. J. (1999). Formal validation of PLC programs : A survey. In *1999 European Control Conference (ECC)*, pages 2170–2175.

Larsen, K. G., Pettersson, P., and Yi, W. (1997). Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2) :134–152.

Lee, C. G. and Park, S. C. (2014). Survey on the virtual commissioning of manufacturing systems. *Journal of Computational Design and Engineering*, 1(3) :213–222.

Leitner, A., Ciupa, I., Meyer, B., and Howard, M. (2007). Reconciling Manual and Automated Testing : The AutoTest Experience. pages 261a–261a. IEEE.

Loborg, P. and Törne, A. (1996). Towards Error Recovery in Sequential Control Applications. In *In Proc 6 :th int. Symp. on Robotics in Manufacturing (ISRAM96.*

Machado, J. M., Denis, B., Lesage, J.-J., Faure, J.-M., and Ferreira Da Silva, J. C. (2006). LOGIC CONTROLLERS DEPENDABILITY VERIFICATION USING A PLANT MODEL. *IFAC Proceedings Volumes*, 39(17) :37–42.

Marangé, P. (2008). *Synthèse et filtrage robuste de la commande pour des système manufacturiers sûrs de fonctionnement.* phdthesis, Université de Reims - Champagne Ardenne.

McMillan, K. (1999). *The SMV Language.* Cadence Berkeley Labs.

McMillan, K. L. (1993). Symbolic Model Checking. In *Symbolic Model Checking*, pages 25–60. Springer, Boston, MA.

Mokadem, H. B. (2006). *Vérification des propriétés temporisées des automates programmables industriels.* PhD Thesis, École normale supérieure de Cachan-ENS Cachan.

Mokadem, H. B., Berard, B., Gourcuff, V., Smet, O. D., and Roussel, J. M. (2010). Verification of a Timed Multitask System With Uppaal. *IEEE Transactions on Automation Science and Engineering*, 7(4) :921–932.

Moon, I. (1994). Modeling programmable logic controllers for logic verification. *IEEE Control Systems*, 14(2) :53–59.

Moon, I., Powers, G., Burch, J., and Clarke, E. (1991). Automatic Verification of Sequential Control Systems using Temporal Logic. *Computer Science Department.*

Moor, T., Raisch, J., and O'Young, S. (1998). Supervisory control of hybrid systems via l-complete approximations. page 6.

Myers, G. J. (1979). *Art of Software Testing.* John Wiley & Sons, Inc., New York, NY, USA.

NFC-63850 (1988). NF C 63-850. Appareillage industriel à basse tension - Automates programmables.

Niang, M., Philippot, A., gellot, F., Coupat, R., Riera, B., and Lefebvre, S. (2017). Formal verification for validation of pseel's plc program. *The 14th International Conference on Informatics in Control, Automation and Robotics, Madrid, Spain*.

Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Pichard, R., Philippot, A., Saddem, R., and Riera, B. (2018). Safety of Manufacturing Systems Controllers by Logical Constraints With Safety Filter. *IEEE Transactions on Control Systems Technology*, pages 1–9.

Queille, J. P. and Sifakis, J. (1982). Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming*, Lecture Notes in Computer Science, pages 337–351. Springer, Berlin, Heidelberg.

Raskin, J.-F. (2005). An Introduction to Hybrid Automata. In Hristu-Varsakelis, D. and Levine, W. S., editors, *Handbook of Networked and Embedded Control Systems*, pages 491–517. Birkhäuser Boston, Boston, MA.

Reisig, W. (2013). *Understanding Petri Nets : Modeling Techniques, Analysis Methods, Case Studies*. Springer-Verlag, Berlin Heidelberg.

Riera, B., Pichard, R., Philippot, A., Saddem, R., Gellot, F., Annebicque, D., and Emprin, F. (2018). Home i/o et factory i/o : 2 logiciels innovants de simulation de po pour la formation à l'automatique.

Rusu, V., Marchand, H., Tschaen, V., Jéron, T., and Jeannet, B. (2004). From Safety Verification to Safety Testing. In Groz, R. and Hierons, R., editors, *Testing of Communicating Systems (Testcom)*, volume 2978 of *Lecture notes in computer science*, pages 160–176, Oxford, United Kingdom. Springer.

Selby, P. C. and Selby, R. W. (2007). 4.4.2 Measurement-Driven Systems Engineering Using Six Sigma Techniques to Improve Software Defect Detection. *INCOSE International Symposium*, 17(1) :640–651.

Soliman, D. and Frey, G. (2011). Verification and validation of safety applications based on PLCopen safety function blocks. *Control Engineering Practice*, 19(9) :929–946.

Sourisse, C. and Boudillon, L. (1997). *La sécurité des machines automatisées : Techniques et moyens de prévention opératifs, systèmes de commande, utilisation des machines*. Institut Schneider Formation. Google-Books-ID : spBXPgAACAAJ.

Sreenivas, R. S. and Krogh, B. H. (1991). On condition/event systems with discrete state realizations. *Discrete Event Dynamic Systems*, 1(2) :209–236.

Thistle, J. G. and Wonham, W. M. (1986). Control problems in a temporal logic framework. *International Journal of Control*, 44(4) :943–976.

Tretmans, J. (1999). Testing concurrent systems : A formal approach. In Baeten, J. C. M. and Mauw, S., editors, *CONCUR'99 Concurrency Theory*, pages 46–65, Berlin, Heidelberg. Springer Berlin Heidelberg.

Vu, L. H. (2015). *Formal Development and Verification of Railway Control Systems - In the context of ERTMS/ETCS Level 2*. PhD thesis, Technical University of Denmark (DTU).

Völker, N. and Krämer, B. J. (1999). Modular Verification of Function Block Based Industrial Control Systems. *IFAC Proceedings Volumes*, 32(1) :159–164.

Weng, X. and Litz, L. (2000). Verification of logic control design using SIPN and model checking : methods and case study. In *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, volume 6, pages 4072–4076 vol.6.

Winter, K. (2012). Optimising ordering strategies for symbolic model checking of railway interlockings. In Margaria, T. and Steffen, B., editors, *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies*, pages 246–260, Berlin, Heidelberg. Springer Berlin Heidelberg.

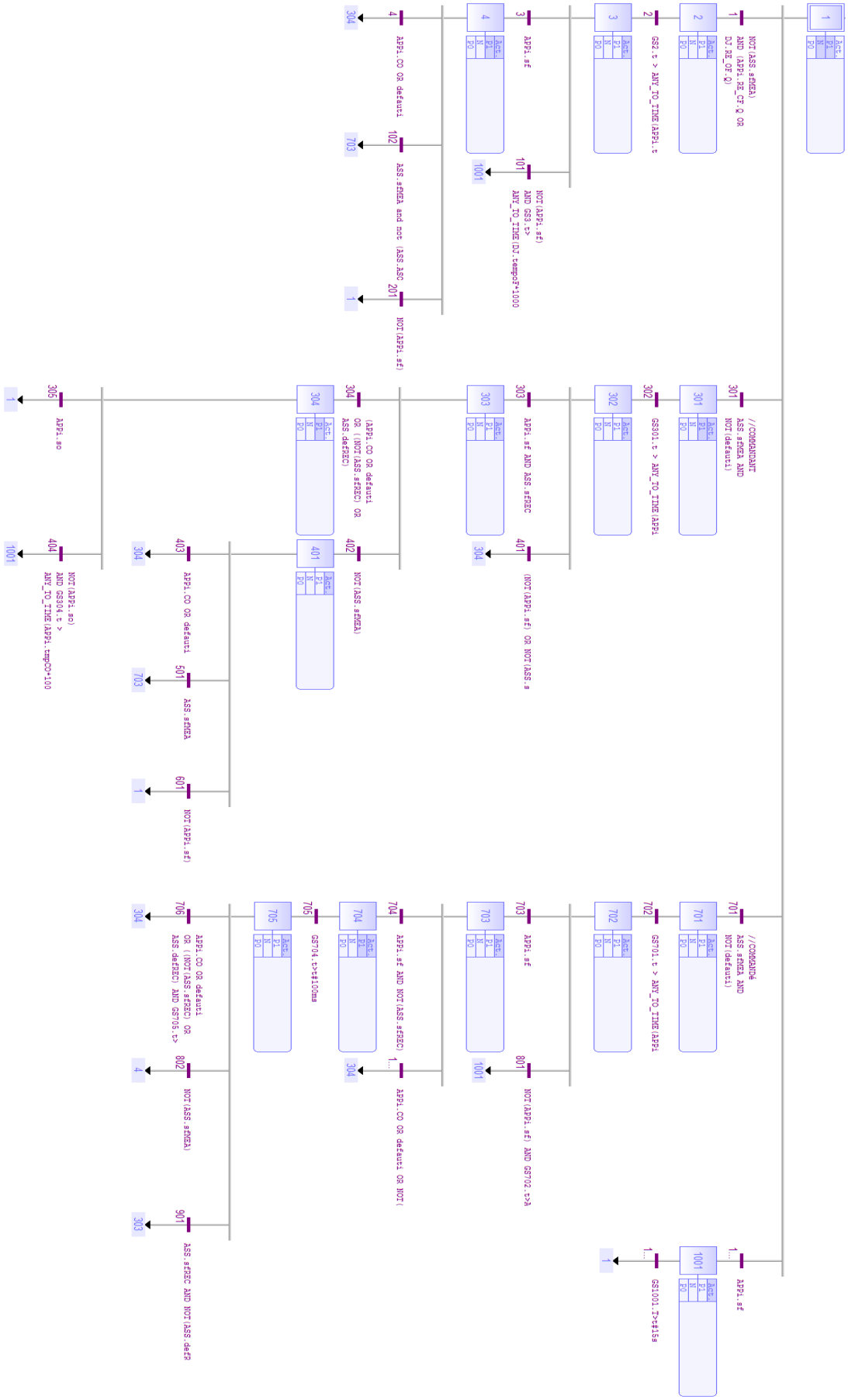
Xin-feng, Z., Jian-dong, W., Xin-feng, Z., Bin, L., Jun-wu, Z., and Jun, W. (2009). Methods to Tackle State Explosion Problem in Model Checking. In *Proceedings of the 3rd International Conference on Intelligent Information Technology Application*, IITA'09, pages 329–331, Piscataway, NJ, USA. IEEE Press.

ZAYTOON, J. (2018). Automatic Generation and Formal Verification of Safe PLC Programs for Railway Control Systems, Plenary session 3 - October 2018, ICCAS, Korea.

Annexe A

Présentation des modèles de vérification des programmes d'asservissement

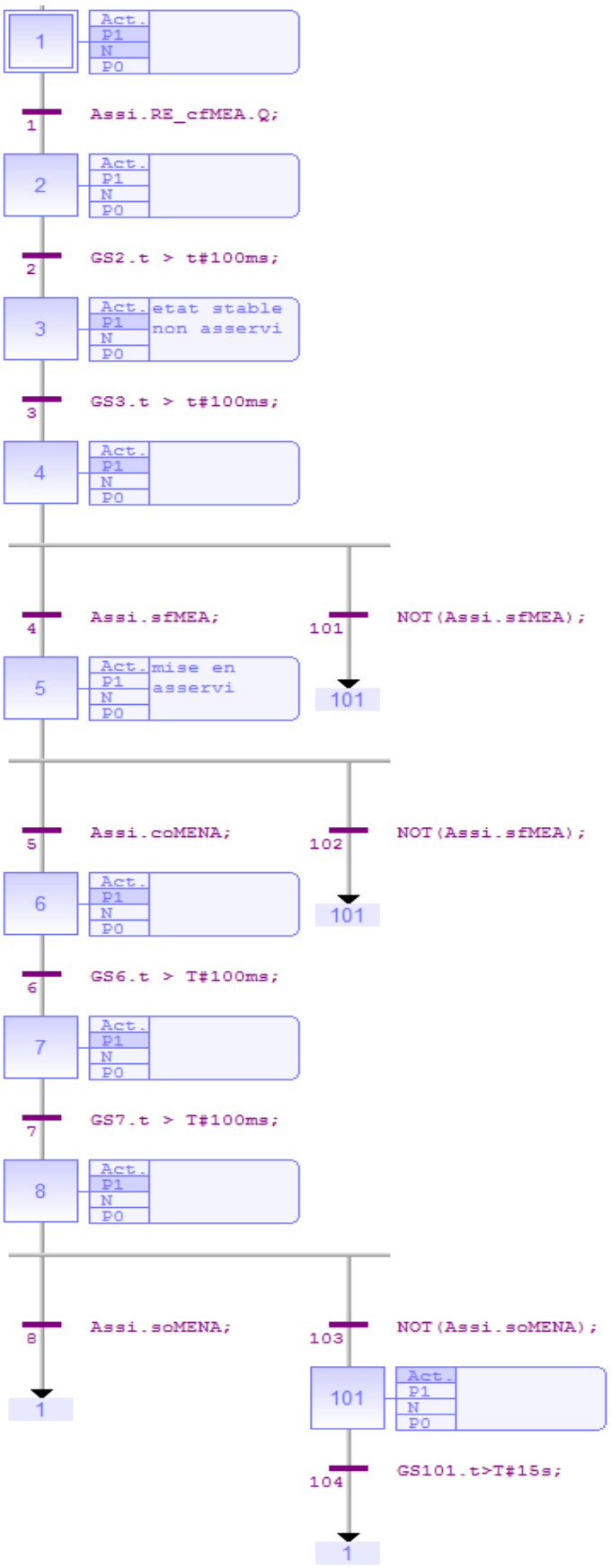
Bloc fonctionnel d’asservissement N°1



Modélisation du bloc fonctionnel d'asservissement N°1

```
void CtrlCmdeDJAservFP_Old (bool &x1, bool &x2, bool &x3, bool &x4, bool &x301, bool &x302,
                            bool &x303, bool &x304, bool &x401, bool &x701, bool &x702, bool &x703,
                            bool &x704, bool &x705, bool &x1001, bool &etat,  bool &sfMEA,
                            bool &sfREC, bool &defREC, bool &COd, bool &CFd, bool &co, bool &cf, bool &so,
                            bool &sf, bool &defauti, bool &CFP, bool &ASO, bool &reenc,
                            bool &def, bool &intmp, bool &Qtmp, int &PT)
{

     bool ft1, ft2, ft3 ,ft101, ft4, ft102, ft201, ft301, ft302, ft303, ft401,
          ft304, ft402, ft305, ft404, ft403, ft501, ft601, ft701, ft702,
          ft703, ft801, ft704, ft1003, ft705, ft706, ft802, ft901, ft1001, ft1002;

     ft1 = x1 and (not sfMEA and (CFd) and not defauti and so);
     ft2 = x2 and (Qtmp);
     ft3 = x3 and (sf);
     ft101 = x3 and (not sf and Qtmp) /* ajout */ and not ft3;
     ft4 = x4 and (COd or defauti);
     ft102 = x4 and (sfMEA /*and not ASO*/)/* ajout */ and not ft4;
     ft201 = x4 and (not sf)/* ajout */ and not ft4 and not ft102;
     ft301 = x1 and (sfMEA and not defauti and not sfREC and (CFd or reenc))/* ajout */ and not ft1;
     ft302 = x301 and (Qtmp);
     ft303 = x302 and (sf and sfREC);
     ft401 = x302 and ((not sf or not sfREC) and Qtmp)/* ajout */ and not ft303;
     ft304 = x303 and (COd or defauti or ((not sfREC or defREC) and Qtmp and sfMEA) or not sf);
     ft402 = x303 and (not sfMEA)/* ajout */ and not ft304;
     ft305 = x304 and (so);
     ft404 = x304 and (not so and Qtmp)/* ajout */ and not ft305;
     ft403 = x401 and (COd or defauti);
     ft501 = x401 and (sfMEA)/* ajout */ and not ft403;
     ft601 = x401 and (not sf)/* ajout */ and not ft403 and not ft501;
     ft701 = x1 and (sfMEA and not defauti and so and sfREC)/* ajout */ and not ft1 and not ft301;
     ft702 = x701 and (Qtmp);
     ft703 = x702 and (sf);
     ft801 = x702 and (not sf and Qtmp)/* ajout */ and not ft703;
     ft704 = x703 and (sf and not sfREC and not defREC);
     ft1003 = x703 and (COd or defauti or not sf)/*and (sfREC) and false*//* ajout */ and not ft704;
     ft705 = x704 and (Qtmp);
     ft706 = x705 and (COd or defauti or ((not sfREC or defREC) and Qtmp) or not sf);
     ft802 = x705 and (not sfMEA)/* ajout */ and not ft706;
     ft901 = x705 and (sfREC and not defREC)/* ajout */ and not ft706 and not ft802;
     ft1001 = x1 and (sf)/* ajout */ and not ft1 and not ft301 and not ft701;
     ft1002 = x1001 and (Qtmp);


     x1 = ft201 or ft601 or ft1002 or ft305 or x1 and not (ft1 or ft301 or ft701 or ft1001);
     x2 = ft1 or x2 and not (ft2);
     x3 = ft2 or x3 and not (ft3 or ft101);
     x4 = ft3 or ft802 or x4 and not (ft4 or ft102 or ft201);
     x301 = ft301 or x301 and not (ft302);
     x302 = ft302 or x302 and not (ft303 or ft401);
     x303 = ft303 or ft901 or x303 and not (ft304 or ft402);
     x304 = ft304 or ft1003 or ft4 or ft401  or ft403 or ft706 or x304 and not (ft305 or ft404);
     x401 = ft402 or x401 and not (ft403 or ft501 or ft601);
     x701 = ft701 or x701 and not (ft702);
     x702 = ft702 or x702 and not (ft703 or ft801);
     x703 = ft703 or ft501 or ft102 or x703 and not (ft704 or ft1003);
     x704 = ft704 or x704 and not (ft705);
     x705 = ft705 or x705 and not (ft706 or ft802 or ft901);
     x1001= ft1001 or ft101  or ft404  or ft801 or x1001 and not (ft1002);

     intmp = (x2 and !ft1) or (x3 and !ft2) or (x301 and !ft301) or (x302 and !ft302) or
             (x303 and !(ft303  or ft1003  or ft901)) or (x304 and !(ft304  or ft4  or ft401
             or ft403  or ft706)) or (x701 and !ft701) or (x702 and !ft702) or (x704 and !ft704)
             or (x705 and !ft705) or (x1001 and !ft1001);
     PT = (x2 or x301 or x701 ? 25 : PT); PT = (x3 or x302 or x702? 100 : PT); PT = (x303 or x704? 20 : PT);
     PT = (x304 ? 100 : PT); PT = (x705 ? 60 : PT); PT = (x1001 ? 200 : PT);

     co = x2 or x3 or x4 or x301 or x302 or x303 or x401 or x701 or x702 or x703 or x704 or x705;
     cf = x2 or x301 or x701;
     def = x1001;
     if (x1) {etat = 0;}
     if (x4) {etat = 1;}
     if (x304 or x401) {CFP = 0;}
     if (x301 or x704) {CFP = 1;}
}
```

Bloc fonctionnel d’asservissement N°2



Modélisation du bloc fonctionnel d'asservissement N°2

```
void AservFP_Old (bool &x1, bool &x2, bool &x3, bool &x4, bool &x5, bool &x6, bool &x7, bool &x8, bool &x9,
                  bool &x101, bool &DJ_x304, bool &so, bool &sf, bool &ASO, bool &ASF, bool &OFP, bool &CFP,
                  bool &coMENA, bool &cfMEA, bool &soMENA, bool &sfMEA, bool &defMEA, bool &soREC,
                  bool &intmp, bool &Qtmp, int &PT)
{
     bool ft1, ft2, ft3, ft4, ft5, ft6, ft7, ft8, ft101, ft103, ft104, ft9, ft10;
     ft1   = x1 and cfMEA and soMENA;
     ft2   = x2 and Qtmp;
     ft3   = x3 and Qtmp;
     ft4   = x4 and sfMEA;
     ft5  = x5 and coMENA;
     ft6  = x6 and Qtmp;
     ft7  = x7 and Qtmp;
     ft8  = x8 and soMENA;
     ft101 = x4 and (not sfMEA);
     ft103 = x8 and !soMENA;
     ft104 = x101 and Qtmp;
     ft9 = x5 and (DJ_x304 /*and sfMEA*/) and not ft5;
     ft10 = x9 and (so and soREC and sfMEA and Qtmp);

     x1   = ft8 or ft104 or x1 and !ft1;
     x2   = ft1 or x2 and !ft2;
     x3   = ft2 or x3 and !ft3;
     x4   = ft3 or x4 and !ft4 and !ft101;
     x5   = ft4 or ft10 or x5 and !ft5 and !ft9;
     x6   = ft5 or x6 and !ft6;
     x7   = ft6 or x7 and !ft7;
     x8   = ft7 or x8 and !ft8 and !ft103;
     x9   = ft9 or x9 and not ft10;
     x101 = ft103 or ft101 or x101 and !ft104;

     PT=(x2 ?20:PT);  PT:=(x3 ?20:PT); PT:=(x6 ?20:PT); PT:=(x7 ?20:PT);
     PT:=(x101 ?150:PT); PT:=(x9 ?150:PT);
     intmp = (x2 and !ft1) or (x3 and !ft2) or (x6 and !ft5) or (x7 and !ft6)
             or (x101 and !(ft103 or ft101)) or (x9 and !ft9);

     ASO = x2 or x3 or x6 or x7;
     OFP = x3 or x4 or x5 or x6;
     if (x1 or x7 or x101) { OFP = false; ASF = false;}
     if (x1 or x101) {ASO = false;}
     if (x3) {ASF=true;}
     if (x9) {CFP=false;}

}
```

Modélisation des modules d'asservissement

```
     //Modules d'asservissement des deux disjoncteurs
     AssV1.soMENA = not (AssV1.ASF);
     AssV1.sfMEA = AssV1.ASF;
     AssV1.sfREC = AssV1.OFP and AssV1.ASF and DT1.preUV1 and (AssV1.CFP and (AssV1.ASO or
                   (AssV2.ASO or not AssV2.ASF) or AssV2.ASF and AssV2.OFP and DT2.preUV2
                    and not AssV2.CFP) or not AssV1.CFP and (AssV2.CFP and AssV2.OFP and
                    AssV2.ASF and DT2.preUV2 and not AssV2.ASO and not AssV1.ASO));
     AssV1.soREC = not AssV1.sfREC;
     AssV2.soMENA = not (AssV2.ASF);
     AssV2.sfMEA = AssV2.ASF;
     AssV2.sfREC = AssV2.OFP and AssV2.ASF and DT2.preUV2 and (AssV2.CFP and (AssV2.ASO or
                   (AssV1.ASO or not AssV1.ASF) or AssV1.ASF and AssV1.OFP and DT1.preUV1
                    and not AssV1.CFP) or not AssV2.CFP and (AssV1.CFP and AssV1.OFP and
                    AssV1.ASF and DT1.preUV1 and not AssV1.ASO and not AssV1.ASO));
     AssV2.soREC = not AssV2.sfREC;
```

Fiches de recettes pour la V&V des programmes d'asservissement FP

Fiche de test Réf.						Objet du test
E	A	01	F	N	U/S	**Fonctionnement et défaut équipement DDL Voie 1 St Clair**

Condition Initiales

- Temporisation maintien fonctionnement DDL = 15s
- Disjoncteur DF2503 fermé

Essais

Actions à réaliser	Résultats Attendus
1. Simuler un fonctionnement DDL DF2503	1. Ouverture Disjoncteur DF2503 - TSS 32 présente - Apparition message CLE : « TSS 32 Fonctionnement DDL » Affichage du message et de la TSS pendant 15s
2. Fermer DF2503. 3. Simuler un défaut protection DDL DF2503	2. Disjoncteur DF2503 fermé. 3. TSS 31 présente et apparition message CLE : « TSS 31 Défaut Equipement DDL »

Commentaires

Usine (Phase 2)	Site (Phase 3)

Décisions

Usine	Correct : ☐	Gênant : ☐	Bloquant :☐	**N° FFT :**	
Site	**Correct : ☐**	**Gênant : ☐**	**Bloquant :☐**		
Test réalisé par	Usine : **Site :**			**Date du test**	Usine : **Site :**

Fiche de test Réf.						Objet du test
E	B	01	F	N	U/S	**Asservissement et Réenclencheur Voie 1 St Clair**

Condition Initiales

- Temporisation Réenclencheur programmée à l'aide de ACSMANAGER = 8s
- Disjoncteur Voie 1 St Clair fermé
- Voie 1 St Clair Non asservi
- Sectionneur IT2503 fermé
- Présence tension caténaire Voie 1 St Clair
- Disjoncteur Voie 1 Les Tabacs à ST CLAIR fermé et asservi

Essais

Actions à réaliser	Résultats Attendus
1. Mettre Voie 1 St Clair en **asservi**.	1. Vérifier sur le module asservissement l'état commandé (Led réception verte et Led émission rouge)
2. A St CLAIR, mettre Voie 1 Les Tabacs en **Non Asservi**	2. Vérifier sur le module asservissement l'état commandant (Led réception verte et Led émission verte) sans déclenchement du DF2503
3. Déclencher DF2503 par le BP rouge en face avant du module de commande (Appuyer 2 fois)	3. Ouverture du DF2503 et Réenclenchement du DJ après la temporisation de 8s.
4. A St CLAIR, mettre Voie 1 Les Tabacs en **Asservi**	4. RAS
5. A St CLAIR, Commande ouverture DJ Voie 1 Les Tabacs	5. Ouverture du DF2503 sans réenclenchement. Ouverture DJ Voie 1 Les Tabacs à St CLAIR → Voie 1 St Clair Hors tension
6. Commande de fermeture du DP1	6. Fermeture DF2503 et DJ Voie 1 Les Tabacs à St CLAIR
7. Commande ouverture DP1	7. Ouverture du DF2503 sans réenclenchement. Ouverture DJ Voie 1 Les Tabacs à St CLAIR → Voie 1 Les Tabacs Hors tension
8. A St CLAIR, Commande de fermeture DJ Voie 1 Les Tabacs	8. Fermeture de DF2503 et DJ Voie 1 Les Tabacs à St CLAIR

Commentaires

Usine (Phase 2)	Site (Phase 3)
	En asservi, Attendre 15s Après la fermeture du disjoncteur pour que le réenclencheur soit armé. Le BP rouge de déclenchement en face avant du module de commande du disjoncteur est un bouton à accrochage ; veiller à le décrocher en appuyant une 2ème fois, sauf si vous souhaitez que le disjoncteur ne se referme pas par réenclencheur

Décisions

Usine	Correct : ☐	Gênant : ☐	Bloquant :☐	**N° FFT :**	
Site	**Correct : ☐**	**Gênant : ☐**	**Bloquant :☐**		
Test réalisé par	Usine : **Site :**			**Date du test**	Usine : **Site :**

Modélisation des fiches de recettes

```
void fich_FctDefDDLV1()
{
      ft0 = ft0 and APP1.so;
      ft1 = x1 and APP1.sf;
      ft2 = x2 and Q;
      ft3 = x3 and APP1.so;
      ft4 = x4 and APP1.sf;
      ft5 = x5 and APP1.sf and Q;
      x1=ft0 or x1 and not ft1;
      x2=ft1 or x2 and not ft2;
      x3=ft2 or x3 and not ft3;
      x4=ft3 or x4 and not ft4;
      x5=ft4 or x5 and not ft5;
      in = x2 and DT1.fctDDLV1 or x5 and DT1.defDDLV1;
      temps =(x2 ?50:temps);  temps =(x5 ?10:temps);
      start = (x1 and not ft0) or (x2 and not ft1) or
              (x3 and not ft2) or (x4 and not ft3) or (x5 and not ft4);
      ft0=0;
      DT1.fctDDLV1 = x2;
      APP1.cfd = x1 or x4;
      DT1.defDDLV1 = x5;
      ok=ft5;
}

void ini_AssReenc()
{
      DT1.preUV1 = APP1.sf or APP5.sf;
      ft1 = x1 and APP1.so;
      ft2 = x2 and APP1.sf and AssV1.soMENA and APP2.sf and DT1.preUV1
            and APP5.sf and AssV2.sfMEA and AssV2.CFP and not AssV2.ASO;

      x1 = ft0 or x1 and not ft1;
      x2 = ft1 or x2 and not ft2;

      APP1.cfd =x2 and APP2.sf ; APP2.cfd =x2; APP5.cfd =x2;
      AssV2.cfMEA = x2 and APP1.sf; APP1.cod = x1;
      ft0=0;
      start = (x1 and not ft0) or (x2 and not ft1);
      ok = ft2;
}

void fich_AssReencV1()
{
      DT1.preUV1 = APP1.sf or APP5.sf;
      ft0=ft0 and APP1.sf and AssV1.soMENA and APP2.sf and DT1.preUV1 and APP5.sf and AssV2.sfMEA;  //condition initiale
      ft1 = x1 and AssV1.sfMEA and AssV1.sfREC and not AssV1.ASO;
      ft2 = x2 and AssV2.soMENA /* ajout */ and AssV1.sfREC and AssV1.CFP and not AssV2.ASO and ReencDJDTV1.x3;
      ft3 = x3 and Q;
      ft4 = x4 and APP1.so;
      ft5 = x5 and APP1.sf ;
      ft6 = x6 and AssV2.sfMEA and AssV2.sfREC and not AssV2.ASO;
      ft7 = x7 and APP5.so and APP1.so and not DT1.preUV1 and Q;
      ft8 = x8 and APP1.sf and APP5.sf;
      ft9 = x9 and APP5.so and APP1.so and not DT1.preUV1 and Q;
      ft10=x10 and APP1.sf and APP5.sf;

      x1 = ft0 or x1 and not ft1; // Mettre Voie 1 en asservi  ---------------------- (==> Voie 1 en asservi)
      x2 = ft1 or x2 and not ft2; // Mettre Voie 2 en non asservi  ------------------ (==> Voie 2 en non asservi,
                                                                                        DJ1 en commandant et reenclenchement DJ1 armé)
      x3 = ft2 or x3 and not ft3; // Appui sur le bouton BPDecl du disjoncteur du DT1 (==> attente...)
      x4 = ft3 or x4 and not ft4; // Attente ouverture disjoncteur 1  --------------- (==> disjoncteur DJ1 ouvert)
      x5 = ft4 or x5 and not ft5; // Attente Réenclenchement du disjoncteur 1 ------- (==> disjoncteur DJ1 fermé)
      x6 = ft5 or x6 and not ft6; // Mettre Voie 2 en asservi ----------------------- (==> Voie 2 en asservi)
      x7 = ft6 or x7 and not ft7; // Ouvrir disjoncteur 2 --------------------------- (==> ouverture DJ2 sans reenc, ouverture
                                                                                        DJ1 par asservissement, absence tension V1)
      x8 = ft7 or x8 and not ft8; // Fermer disjoncteur 1 --------------------------- (==> fermeture DJ1 et DJ2)
      x9 = ft8 or x9 and not ft9; // Ouvrir disjoncteur 1 --------------------------- (==> ouverture DJ1 sans reenc, ouverture
                                                                                        DJ2 par asservissement, absence tension V1)
      x10= ft9 or x10 and not ft10; // Fermer disjoncteur 2 ------------------------- (==> fermeture DJ1 et DJ2)

      in = x3 or x7 or x9; temps =(x3 ?20:temps); temps =(x7 or x9 ?80:temps);
      start = (x1 and not ft0) or (x2 and not ft1) or (x3 and not ft2) or (x4 and not ft3) or (x5 and not ft4) or
              (x6 and not ft5) or (x7 and not ft6) or (x8 and not ft7) or (x9 and not ft8) or (x10 and not ft9);
      ft0=0;
      APP1.cfd = x8;
      APP1.cod = x9;
      APP5.cfd = x10;
      APP5.cod = x7;
      AssV1.cfMEA = x1;
      AssV2.cfMEA = x6;
      AssV2.coMENA = x2;
      DT1.BPdecl = x3;

      ok=ft10;
}
```

Annexe B

Compte rendu de débuggage des programmes d'asservissement FP

COMPTE RENDU DE DEBUGGAGE DES PROGRAMMES

D'ASSERVISSEMENT FP DES DISJONCTEURS DEPARTS TRACTIONS

Ce document liste les différentes erreurs détectées (avec propositions de correction) lors de la vérification formelle des programmes d'asservissement FP.

Soient DJ1 et DJ2, les deux disjoncteurs asservis entre eux. Les scénarios suivants ont révélé des erreurs dans les programmes automates.

- Mon DJ1 est fermé et non asservi (étape x4 du grafcet ***CtrlCmdeDJAservFP***), je fais une demande de mise en asservi (évolution de x4 vers x703). Vu que DJ2 était commandant, DJ1 devrait normalement passer en commandé. Mais vu qu'il attaque l'étape x703 avant même que sa manœuvre de mise en asservi soit achevée (et donc ASO toujours fermé), ce DJ ne reçoit pas de réception de DJ2, et va vouloir automatiquement passer en commandant (x703➔x704➔x705). Ce qui donnerait deux disjoncteurs commandés en même temps.

  Solution : attendre que la manœuvre de mise en asservi soit terminée avant de passer à x703 ➔ remplacer la transition ft102= « *ASS.sfMEA* » par « ASS.sfMEA and not (ASS.ASO_1 and ASS.ASO_2); »

- Mon DJ1 étant asservi et commandant (étape x303 du grafcet ***CtrlCmdeDJAservFP*)**, lorsque j'émets une commande de mise en hors asservi, alors il se met en non asservi tout en ne remettant pas CFP à 0. Par conséquent, en cas de prochain passage en asservi au moment où l'autre DJ était asservi et commandant, on se retrouvera avec deux disjoncteurs commandants.

  Solution : rajouter l'action CFP :=0 à l'étape x401 du grafcet *CtrlCmdeDJAservFP*.

- Lorsque le DJ1 asservi et commandé se ferme, il passe aussitôt en commandant (x703➔ x303) car la transition ft1003=ASS.sfREC sera toujours vrai à ce moment précis, étant donné qu'il reçoit sfREC du DJ2 commandant.

Solution : supprimer la transition ft1003=ASS.sfREC (il sera remplacé par une autre transition, voir ci-dessous);

- Suite à la précédente modification, pour tout disjoncteur asservi commandé et fermé, son grafcet reste à l'étape x703, et la seule évolution possible à ce moment serait qu'il ait une perte de réception et qu'il essaie de passer en commandant. Toutefois, lorsqu'il reçoit l'ordre de s'ouvrir (par commande CO, défaut, ou perte de position sf), il ne réagit pas vu qu'il n'y a aucune transition depuis l'étape x703 qui prendra en compte cette information pour faire ouvrir le disjoncteur ou ouvrir la boucle FP.

Solution : remettre la transition ft1003, et remplacer sa valeur « ASS.sfREC » par « APPi.CO OR defauti OR NOT(APPi.sf); » et cette transition devra mener à l'étape x304 et non x303.

- Lorsque le DJ est fermé, asservi et commandé (étape x703), et qu'il reçoit l'ordre de s'ouvrir (par commande CO, défaut, ou perte de position sf), il devra pendant l'ouverture, ouvrir la

boucle FP en désactivant OFP, afin que l'autre DJ voit la perte de réception pour ensuite s'ouvrir à son tour. Avec les grafcets actuels, lorsque le DJ s'ouvre, la boucle FP reste fermée. Par conséquent, aussitôt après son ouverture, le DJ se referme en commandé (x703➔x304 ➔ x1 ➔ x701 ➔ x703), étant donné qu'après son ouverture, il reçoit sfREC de l'autre DJ qui était resté en commandant.

Solution : voir nouveau grafcet AservFP modifié

- Transition ft706 : problème de parenthèse. Les parenthèses surlignées en jaune ont été omis, car la temporisation de l'étape x705 doit être associée à **not sfREC** et **defREC,** et pas seulement à **defREC.**

APPi.CO OR defauti OR ((NOT(ASS.sfREC) OR ASS.defREC) AND GS705.t> t#1000ms) OR NOT(APPi.sf).

- Transition ft901 : enlever le terme « AND FALSE ».

- Soit le scénario suivant : les deux DJ sont fermés et asservis, DJ1 commandant (x303) et DJ2 commandé (x703).
  1. Je mets DJ1 en non asservi ➔ évolution de x303 vers x401, le module de DJ1 n'alimente plus la boucle FP. Par conséquent, DJ2 passe en commandant (évolution de son grafcet vers x703 ➔ x704 ➔ x705 ➔x303).
  2. Je remets DJ1 en asservi ➔ évolution de son grafcet de x401 vers x705, puis de x705 vers x303 car le DJ1 reçoit de DJ2 qui est commandant, et donc la transition ***ft901=sfREC*** le mène vers l'étape x303.
  3. A ce stade, les deux DJ sont tous à leurs étapes x303 respectives, ce qui est normal pour DJ2 qui est commandant, mais pas pour DJ1 qui est commandé (car il devrait être à x703).
  4. Je fais passer le DJ2 commandant en non asservi, par conséquent il arrête d'émettre. Logiquement on devrait s'attendre à ce que DJ1 qui était commandé, passe en commandant : ce qui ne sera pas possible puisque son grafcet était à l'étape x303 (et non x703), et ne pourra donc pas effectuer la séquence (x703 ➔ x704 ➔ x705 ➔x303) pour passer en commandant. Au lieu de ça, le DJ1 s'ouvre (x303➔x304, car il ne reçoit pas le sfREC).

Solution : Depuis l'étape x401, lorsque le DJ est remis en asservi, il devrait attaquer l'étape x703 et non l'étape 705 pour éviter qu'il ne passe automatiquement en x303. Depuis l'étape x703, il pourra :

- **soit y rester s'il réaliser que l'autre DJ est passé en commandant entre temps,**
- **soit effectuer la manœuvre de passage en asservi (x703 ➔ x704 ➔ x705) s'il ne recevait pas de réception.**

Pour cela, le saut d'étape x401➔x705 devra être remplacé par le saut x401➔x703.

Remarque : il faut noter qu'à la 4ième étape de la séquence définie précédemment, on aurait pu ouvrir le DJ2 au lieu de faire une commande de mise en non asservi. Logiquement, on devrait s'attendre à ce que le DJ1, ayant perdu la réception, essaie d'abord de passer en commandant (x703 ➔ x704 ➔ x705) pour ensuite réaliser (avant de s'ouvrir) que la perte de réception est due à une ouverture de la boucle FP et non une manœuvre de passage en/hors asservissement

du module d'asservissement de DJ2. Le résultat obtenu avec les programmes actuels est pourtant pareil, même si ils renferment quelques erreurs. En effet, quand on ouvre le DJ2 commandant (x303➔x304) et qu'il ouvre la boucle FP, le DJ1 commandé s'ouvre aussi instantanément (x303➔x304), mais sans trouver la raison de la perte de réception comme il devrait le faire. L'instruction de la fiche de recette qui consistait à effectuer cette séquence a pourtant été validée avec le programme, mais n'a pas permis de détecter cette anomalie. Ce qui montre l'intérêt de vérifier également l'état des grafcets, car seule l'analyse des traces du model-checker pendant l'exécution du cahier de recettes nous a permis de repérer cette anomalie.

- Peut-on avoir deux disjoncteurs commandants en même temps ? Oui
  - Lorsque les deux DJ sont ouverts, on les met en asservis, puis on les ferme en même temps (ou quasiment) : les deux n'ayant pas de réception, ils émettent !
  - Lorsque les deux DJ sont ouverts et non asservis, on les ferme séparément, puis on les mets en asservis en même temps: ils passent tous deux à l'étape 703). A ce moment précis, ils n'ont aucune réception, alors ils essaient tous les deux de passer en commandant.

Peut-on avoir deux disjoncteurs tous deux fermés et en commandés ? Non

Annexe C

Spécifications structurelles et fonctionnelles des équipements génériques d'une EALE

SPECIFICATION DES MODELES D'APPAREILS POUR LE SIMULATEUR DE PARTIE OPERATIVE

DOCUMENT ETABLI PAR : MOHAMED NIANG

SAMEDI 22 SEPTEMBRE 2018

SNCF RÉSEAU

SOMMAIRE

1. INTRODUCTION

Le document présente les spécifications fonctionnelles pour la modélisation des équipements présents dans les installations électriques (commandes électriques pour appareils, modules d'asservissement, …). Les catégories de commandes électriques peuvent être classées en 4 types, en fonction de leurs paramètres booléens internes CFimp et COmu. Chacun des équipements sera développé dans une section, avec une description des entrées sorties et du comportement de l'appareil.

2. COMMANDE ELECTRIQUE TYPE 1 (CFIMP=1 ET COMU=1)

2.1. INTRODUCTION

Cette section présente les spécifications fonctionnelles pour la modélisation d'une commande électrique type 1. Nous présentons en section 2.2 les entrées sorties et paramètres associés à cet appareil, puis nous décrivons le fonctionnement en section 2.3.

2.2. DEFINITION DES ENTREES/SORTIES

La figure suivante définit toutes les entrées/sorties et paramètres associés à l'appareil. Pour rappel, les valeurs de ces données devront être accessibles en ligne pendant la simulation.

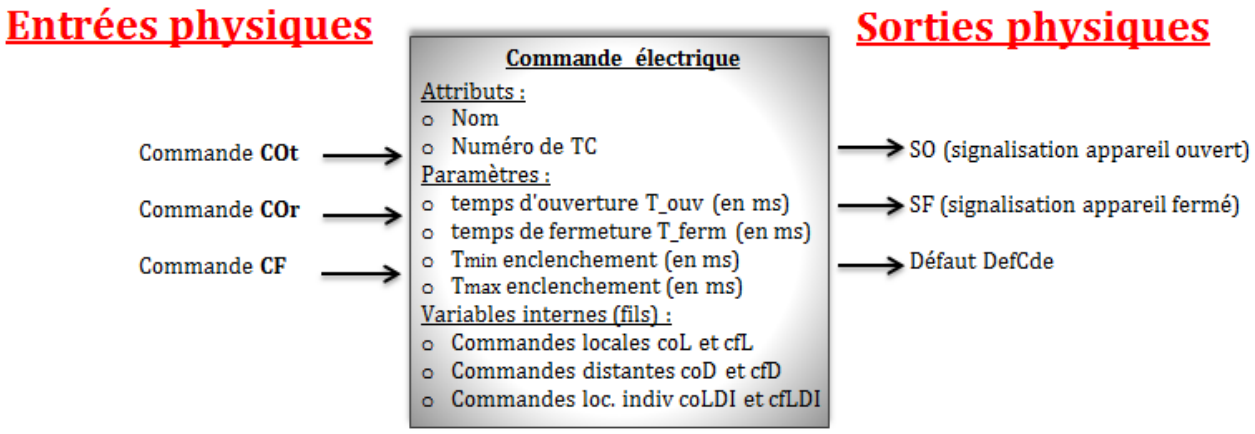


Figure 1 : Entrées/Sorties de l'appareil

Entrées physiques de l'appareil:

Les commandes COt, COr, et CF correspondent aux trois sorties de la carte TC chargée de commander l'appareil. Elles constituent les 3 données d'entrées du modèle.

Attributs disjoncteur :

Un nom et un numéro de TC seront attribués à chaque appareil.

Paramètres appareil :

- Temps d'ouverture T-ouv (en ms) : c'est le temps nécessaire à l'appareil pour effectuer une ouverture complète.
- Temps de fermeture T-ferm (en ms) : c'est le temps nécessaire à l'appareil pour effectuer une fermeture complète.
- T_{min} et T_{max} enclenchement (en ms) : voir section 2.3 dans la description du fonctionnement. Leurs valeurs sont par défaut égales à 0,3s et 2s respectivement.

Les valeurs des temps définis ci-dessus sont initialisées en fonction du type d'appareil, et peuvent être modifiées en cas de nécessité depuis le logiciel simulateur de PO.

Variables internes (fils) :

- Commandes locales coL et cfL : il s'agit des demandes d'ouverture et de fermeture de l'appareil en mode local. Ces variables sont lues par le programme API qui validera ensuite (ou pas) la demande, elles n'ont donc aucun effet direct sur l'appareil.
- Commandes distantes coD et cfD : Même effet que les variables précédentes, mais en mode distant.
- Commandes locales individuelles coLDI et cfLDI : Même effet que les variables précédentes, mais en mode local individuel.

Sorties physiques de l'appareil:

- Signalisations SO et SF : il s'agit des signalisations « appareil ouvert » et « appareil fermé » envoyées par l'appareil vers les 2 entrées de la carte TC de l'appareil.
- Défaut DefCde (défaut de commande électrique) : il s'agit d'un défaut qui peut se produire dans l'appareil, et que l'opérateur peut simuler depuis l'interface du logiciel (soit manuellement, soit par le cahier de recettes). Ce défaut est de type booléen, et simuler un tel défaut peut parfois consister à forcer la variable à 1 (si le défaut est câblé en direct) ou à 0 (s'il est câblé en inverse). Le type de câblage du défaut est renseigné pendant la description du projet dans Odil.

2.3. DESCRIPTION DU FONCTIONNEMENT

Initialement, l'appareil devra être en position ouverte (SO = 1 et SF = 0). Le chronogramme suivant résume le fonctionnement de l'appareil.

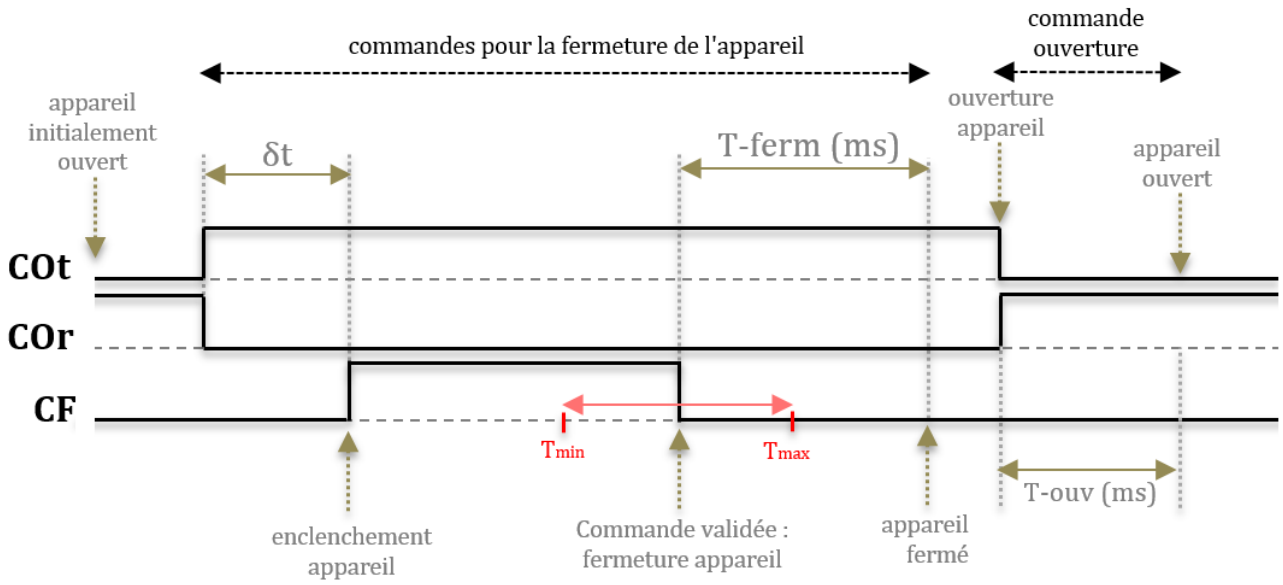


Figure 2 : fonctionnement commande électrique FSK2 pour DJ-IMP-IP-IA

Conditions de fermeture (voir chronogramme) :

L'appareil est initialement ouvert, les entrées physiques CF et COt sont inactives et l'entrée COr est active. Les conditions de fermeture de l'appareil sont définies par la séquence suivante : l'entrée physique COt devra d'abord être activée (en même temps que la désactivation de l'entrée COr). Ensuite, toujours en maintenant COt activée et COr désactivée, l'entrée physique CF devra être activée. L'activation de COt avant celle de CF est nécessaire, sans quoi la commande n'est pas acceptée. La présence du δt (>= au temps de cycle ou temps de scrutation de Simac) sur le chronogramme représente juste le décalage entre les deux activations. A partir de ce moment précis (c'est-à-dire au front montant de CF), la commande de fermeture de l'appareil ne sera valide que si la durée du signal CF (c'est-à-dire la durée mesurée entre le front montant et le front descendant de CF) est comprise entre T_{min} et T_{max}, et que COt et COr maintiennent leurs valeurs, autrement la commande ne sera pas valide et l'appareil restera ouvert. Lorsque la commande est valide, l'appareil entame sa fermeture (SO = 0 et SF = 0) avant de se fermer complétement (SO = 0 et SF = 1) au bout d'un certain temps (paramètre T-ferm en ms). Une commande de fermeture sur un appareil déjà fermé n'a aucun effet sur celui-ci.

Condition d'ouverture :

Quel que soit l'état de l'appareil (fermé, en fermeture,…), la désactivation de l'entrée physique COt **ou** l'activation de COr (**ou** les 2 évènements réunis) provoque automatiquement l'ouverture **complète** du disjoncteur, même pendant la manœuvre de fermeture. Une commande d'ouverture sur un appareil déjà ouvert n'a aucun effet sur celui-ci.

Pour ce type de commande électrique, le défaut « DefCde » n'a aucun impact **direct** sur le comportement de l'appareil simulé.

3. COMMANDES ELECTRIQUES TYPE 2 (CFIMP=1 ET COMU=0)

3.1. INTRODUCTION

Cette section présente les spécifications fonctionnelles pour la modélisation des commandes électriques type 2 (CFimp = 1 et COmu = 0).

3.2. DEFINITION DES ENTREES/SORTIES

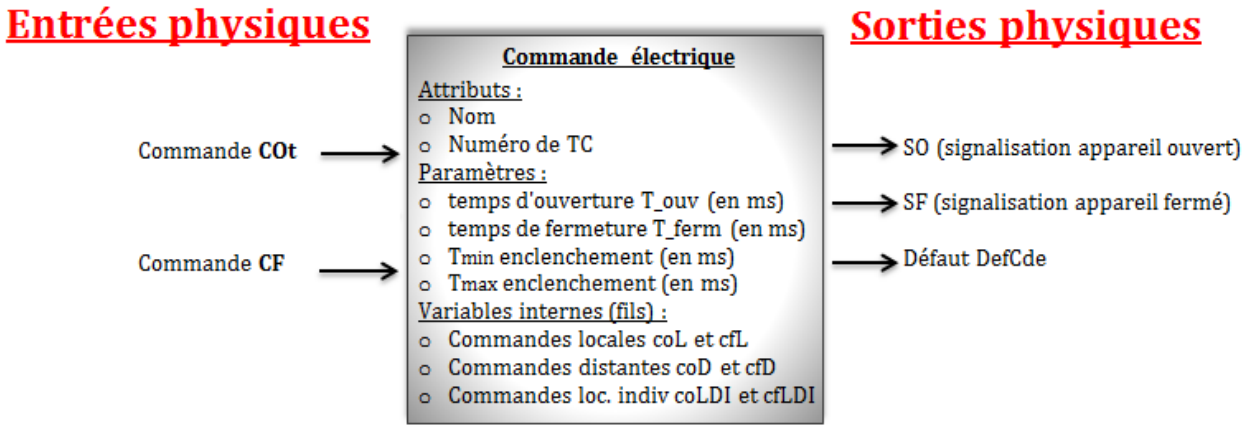


Figure 3 : Entrées/Sorties de l'appareil

Ce type d'appareil reçoit uniquement comme entrée les commandes COt et CF, et renvoie en sortie les signalisations SO/SF en plus du défaut de commande électrique « DefCde ».

3.3. DESCRIPTION DU FONCTIONNEMENT

Initialement, l'appareil peut être en position ouverte (SO = 1 et SF = 0) ou fermée (SO = 0 et SF = 1). La position initiale par défaut est la position ouverte, mais l'utilisateur devra pouvoir la modifier avant le début de la simulation. Les chronogrammes suivants résument le fonctionnement de l'appareil.

Commande de fermeture :

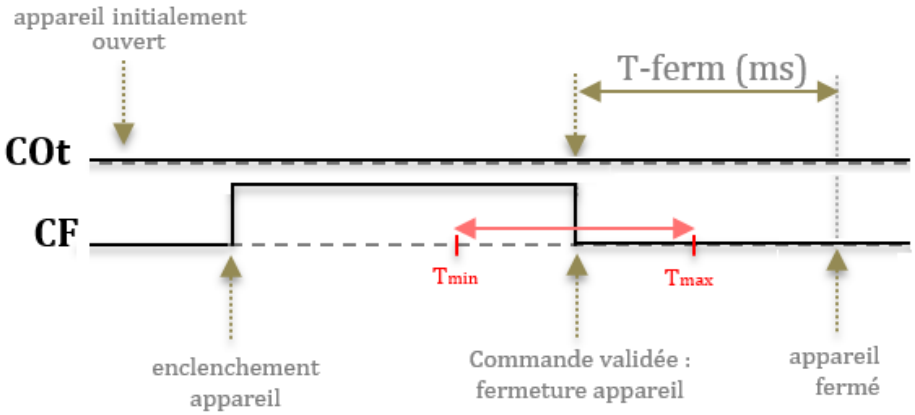


Figure 4 : fermeture commande électrique

Etant initialement ouvert, la fermeture de ce type d'appareil est à peu près identique à celle décrite à la section 2.3, c'est-à-dire : début de fermeture de l'appareil au front descendant de la commande CF, avec une durée du signal CF compris entre T_{min} et T_{max}. La seule différence est que la commande COt doit nécessairement rester inactive tout au long de la manœuvre, sans quoi la commande ne serait pas valide. Lorsque la commande est valide, l'appareil entame sa fermeture (SO = 0 et SF = 0) avant de se fermer complétement (SO=0 et SF = 1) au bout d'un certain temps (paramètre T-ferm en ms). Une commande de fermeture sur un appareil déjà fermé n'a aucun effet sur celui-ci.

Commande d'ouverture :

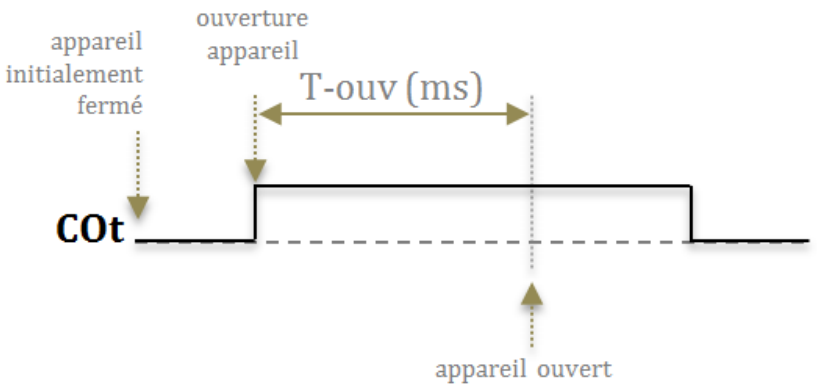


Figure 5 : ouverture commande électrique

Etant initialement fermé, l'appareil entame son ouverture au front montant de la commande COt. L'ouverture complète n'a lieu que si la durée du signal COt est supérieure ou égale au temps d'ouverture T-ouv (ms) de l'appareil. Dans le cas contraire l'appareil reste bloqué entre les deux positions SO et SF, et ne **continue** son ouverture (ou ne se referme) que si elle reçoit à nouveau la commande correspondante. Contrairement à la fermeture, l'ouverture de l'appareil se produit quelle que soit la valeur de la commande CF, et même pendant la fermeture de l'appareil, car l'ouverture est prioritaire. Une commande d'ouverture sur un appareil déjà ouvert n'a aucun effet sur celui-ci.

Pour ce type de commande électrique, le défaut « DefCde » n'a aucun impact sur le comportement de l'appareil simulé. Son état est lu par le programme API en passant par les borniers des châssis de contrôle commande.

4. COMMANDES ELECTRIQUES TYPE 3 (CFIMP=0 ET COMU=0)

4.1. INTRODUCTION

Cette section présente les spécifications fonctionnelles pour la modélisation des commandes électriques type 3 (CFimp = 0 et COmu = 0).

4.2. DEFINITION DES ENTREES/SORTIES

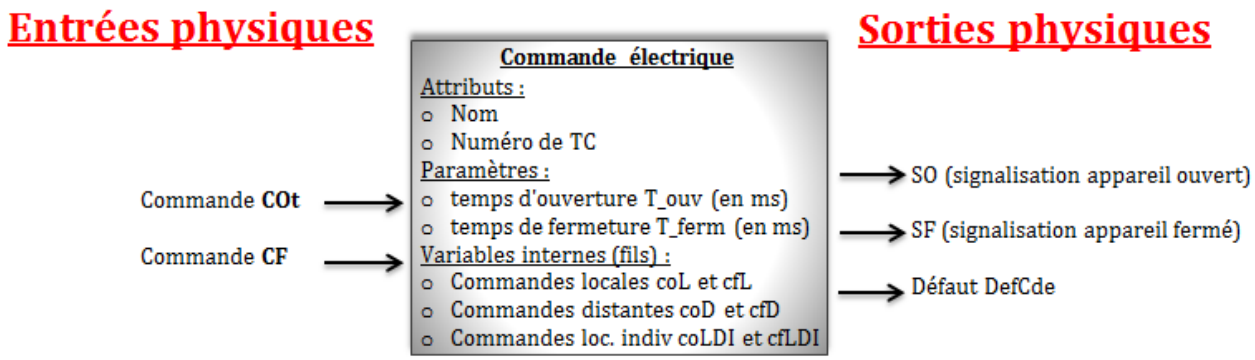


Figure 6 : Entrées/Sorties de l'appareil

Ce type d'appareil reçoit uniquement comme entrée les commandes COt et CF, et renvoie en sortie les signalisations SO/SF en plus du défaut de commande électrique « DefCde ».

4.3. DESCRIPTION DU FONCTIONNEMENT

Initialement, l'appareil peut être en position ouverte (SO = 1 et SF = 0) ou fermée (SO = 0 et SF = 1). La position initiale par défaut est la position ouverte, mais l'utilisateur devra pouvoir la modifier avant le début de la simulation.

Commande de fermeture :

Etant initialement ouvert (ou pas totalement fermé), l'appareil entame sa fermeture au front montant de la commande CF. La fermeture complète ne peut avoir lieu que si la durée du signal CF (c'est-à-dire la durée mesurée entre le front montant et le front descendant) est supérieure ou égale au temps de fermeture de l'appareil (ou au temps de fermeture restant, dans le cas où il n'était pas totalement fermé). Dans le cas contraire l'appareil reste bloqué entre les deux positions SO et SF, et ne **continue** sa fermeture (ou ne s'ouvre) que si elle reçoit à nouveau la commande correspondante. Il est important de noter que cette manœuvre de fermeture n'est possible que si la commande d'ouverture COt reste inactive, car celle-ci est prioritaire sur le CF. C'est-à-dire que lorsque les deux commandes sont activées, seule la commande d'ouverture sera prise en compte. De plus, une commande de fermeture sur un appareil déjà fermé n'a aucun effet sur celui-ci.

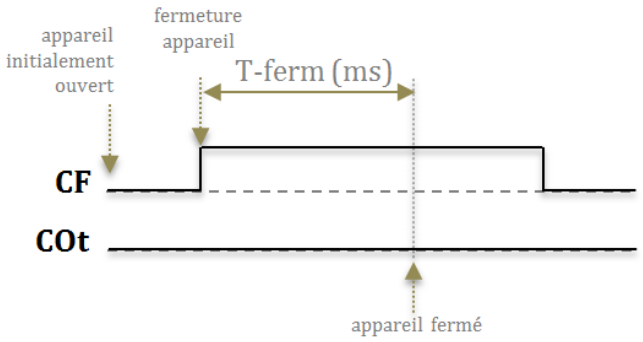


Figure 7 : fermeture commande électrique

Commande d'ouverture :

Etant initialement fermé (ou pas totalement ouvert), l'appareil entame son ouverture au front montant de la commande COt. L'ouverture complète n'a lieu que si la durée du signal COt est supérieure ou égale au temps d'ouverture de l'appareil (ou au temps d'ouverture restant, dans le cas où il n'était pas totalement ouvert). Dans le cas contraire l'appareil reste bloqué entre les deux positions SO et SF, et ne **continue** son ouverture (ou ne se ferme) que si elle reçoit à nouveau la commande correspondante. Contrairement à la fermeture, l'ouverture de l'appareil se produit quelle que soit la valeur de la commande CF. Une commande d'ouverture sur un appareil déjà ouvert n'a aucun effet sur celui-ci.

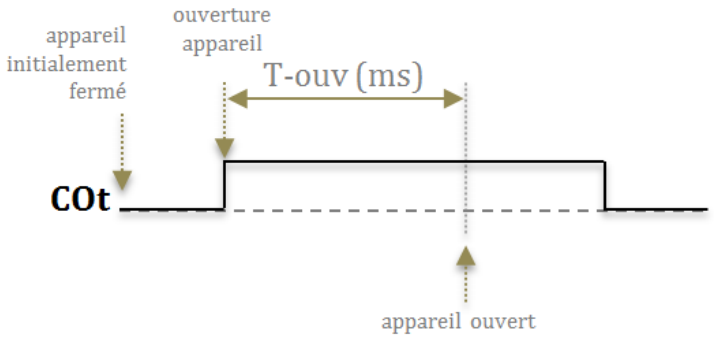


Figure 8: ouverture commande électrique

Le défaut « DefCde » n'a aucun impact sur le comportement de l'appareil simulé. Son état est lu par le programme API en passant par les borniers des châssis de contrôle commande.

En ce qui concerne les commandes électriques type 4, leur fonctionnement est similaire à celui du type 3. La seule différence est que la commande COt est en inverse.

5. MODULE D'ASSERVISSEMENT FILS PILOTE (FP)

5.1. INTRODUCTION

Cette section présente les spécifications fonctionnelles pour la modélisation d'un module d'asservissement type fils pilote.

5.2. DEFINITION DES ENTREES/SORTIES

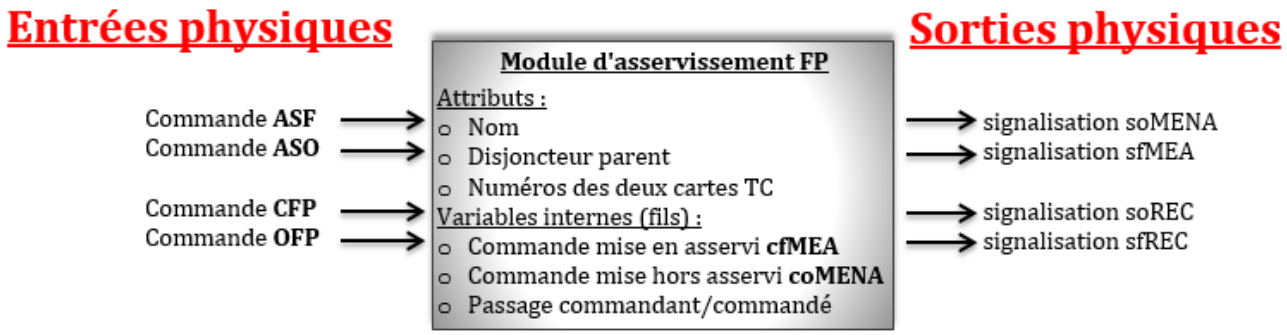


Figure 9 : Entrées/Sorties du module d'asservissement FP

Entrées physiques de l'appareil:

Les commandes ASF, ASO, CFP, et OFP correspondent aux entrées servant à exciter les relais respectifs portant le même nom (voir figure 11).

Attributs:

Un nom sera affecté à chaque module d'asservissement FP instancié. De plus, un module d'asservissement est nécessairement affecté (lors de son instanciation) à un disjoncteur, car on parle d'asservissement ***entre disjoncteurs***. Les modules d'asservissements leur permettent juste de remplir ces fonctions d'asservissement. Enfin, un module d'asservissement est piloté par deux cartes TC de l'automate, les deux numéros de cartes TC devront être renseignés.

Variables internes (fils):

A l'image des commandes distantes et locales associées à toute commande électrique et n'ayant aucun impact direct sur le comportement du modèle (voir section 2.2), trois types de commandes internes sont associées à un module d'asservissement :

- Commande mise en asservi cfMEA : cette commande envoyée à l'automate (depuis l'IHM) correspond à une demande de passage en mode asservi.
- Commande de mise en non asservi coMENA : correspond à une demande de passage en mode non asservi.
- Passage en commandant/commandé : correspond à une demande de passage en mode commandant (i.e. le module d'asservissement alimente la boucle de cuivre, voir figure 10) ou en mode commandé (i.e. le module d'asservissement n'alimente pas la boucle, il devient un récepteur).

Sorties physiques :

- Signalisation soMENA : Disjoncteur parent en mode non asservi
- Signalisation sfMEA : Disjoncteur parent en mode asservi
- Signalisation soREC : Absence de courant dans la boucle FP
- Signalisation sfREC : Présence de courant dans la boucle FP

5.3. PRINCIPE DE L'ASSERVISSEMENT FILS PILOTE (FP)

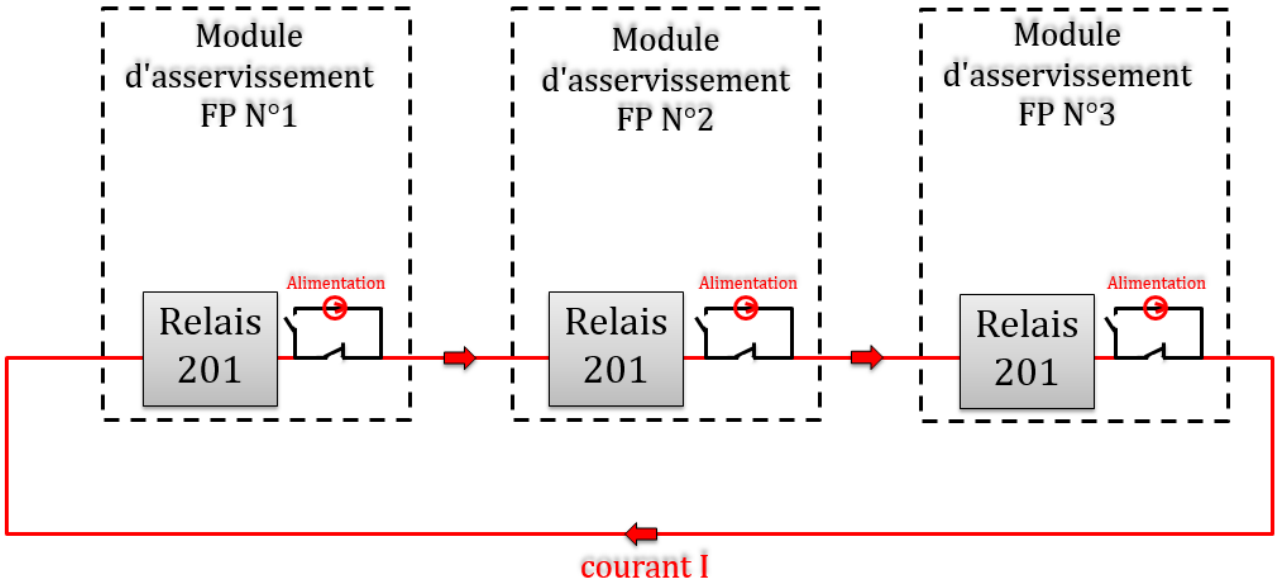


Figure 10 : exemple de connexion entre 3 modules d'asservissements

L'asservissement FP consiste à faire circuler un courant dans une boucle de cuivre (figure 10) constituée de 2 ou plusieurs modules d'asservissement FP connectés en série et semblables à celui de la figure 11. Chaque module d'asservissement dispose d'une source d'alimentation (qui peut être shunté ou non en fonction de l'état du circuit du module) et d'un relais (relais **R-201**) qui a pour fonction de détecter la présence d'un courant I dans la boucle. Cette boucle est alimentée en courant par **au plus un** module d'asservissement à la fois, car dans le cas contraire cela pourrait endommager l'ensemble des modules. Cette interconnexion entre modules d'asservissement implique donc qu'après leur instanciation, ceux-ci devront être électriquement connectés entre eux avant le début de la simulation.

5.4. DESCRIPTION DU FONCTIONNEMENT D'UN MODULE D'ASSERVISSEMENT

Un module d'asservissement se présente sous la forme d'un schéma à base de relais (ASF, ASO, CFP, OFP, et R-201, voir figure 11) et de contacts monostables ayant chacun une position de repos (ouvert ou fermé) comme le montre la figure 11. Ces contacts monostables connectés entre eux (l'alimentation y compris) constituent le circuit du module d'asservissement qu'on appellera ***circuit FP*** (voir figure 11). L'interconnexion des modules d'asservissement FP (figure 10) se fait donc à travers leur circuit FP.

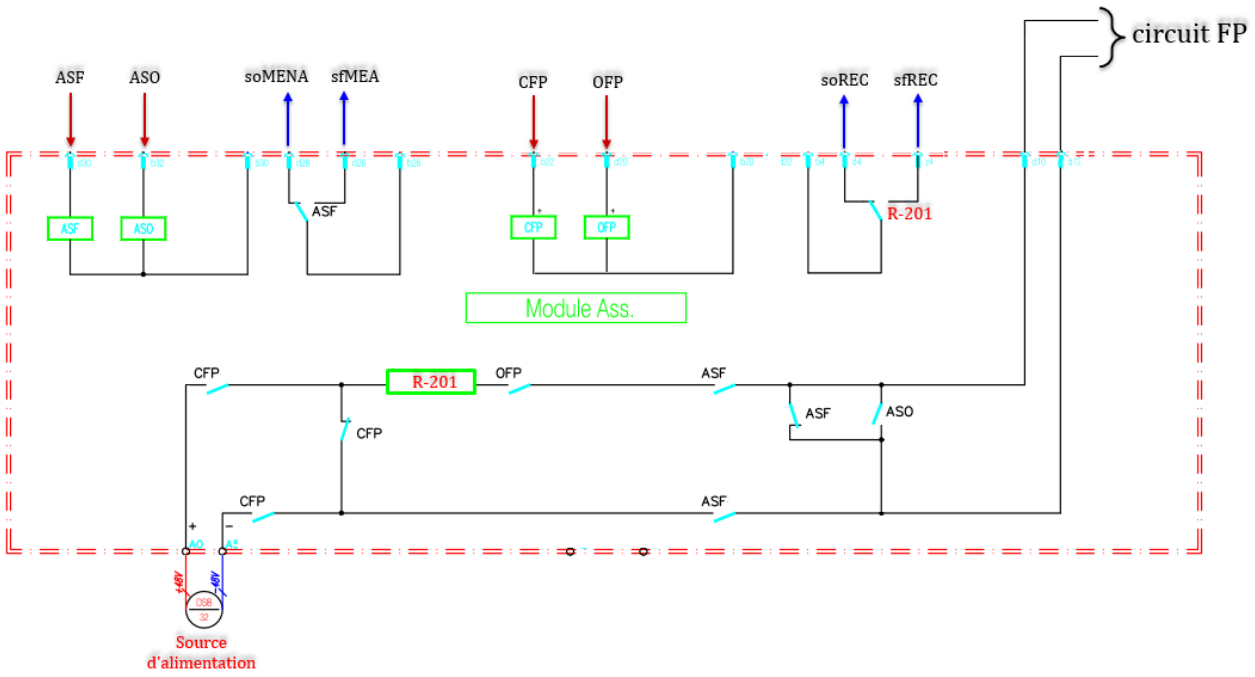


Figure 11 : schéma de principe du module d'asservissement FP

Le circuit FP d'un module d'asservissement se comporte comme un vrai circuit électrique. L'excitation des différents relais ASF, ASO, CFP et OFP provoque l'ouverture ou la fermeture des contacts associés et présents dans le circuit FP, ce qui y'aura pour effet d'exciter ou non le relais R-201 qui permet de détecter la présence de courant dans la boucle. L'état des sorties soMENA et sfMEA dépend de l'état du relais ASF comme l'indique la figure 11, de même que les sorties soREC et sfREC qui dépendent du relais R-201.

Prenons en exemple le cas d'une boucle FP constituée de deux modules d'asservissements :

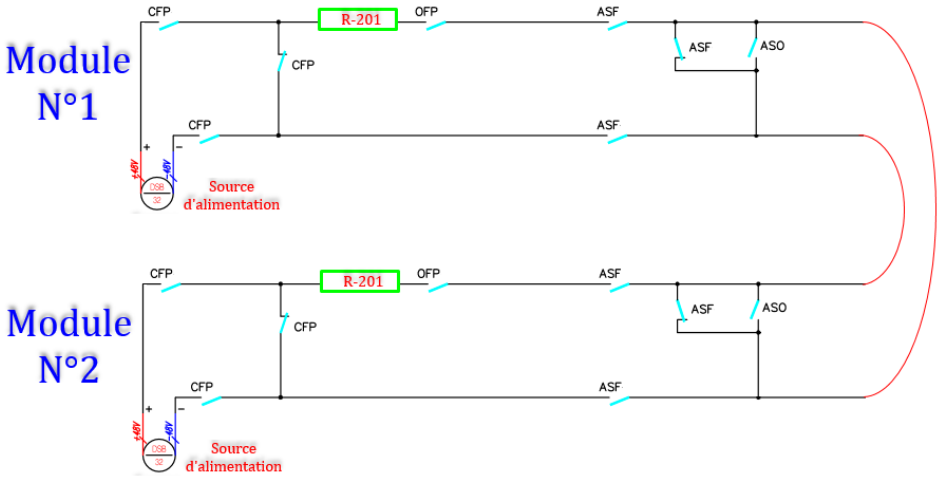
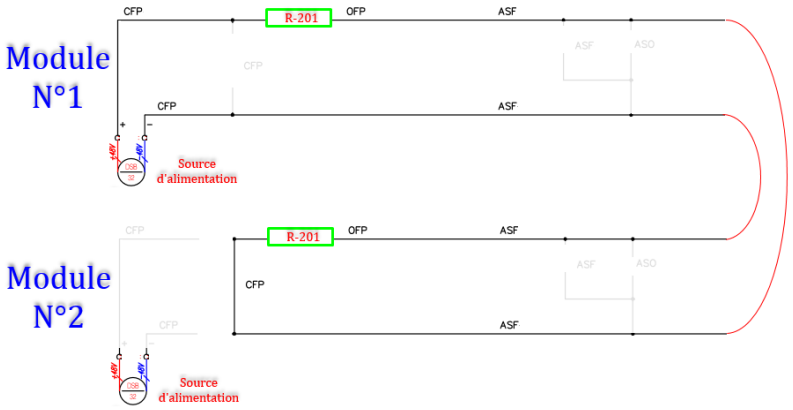


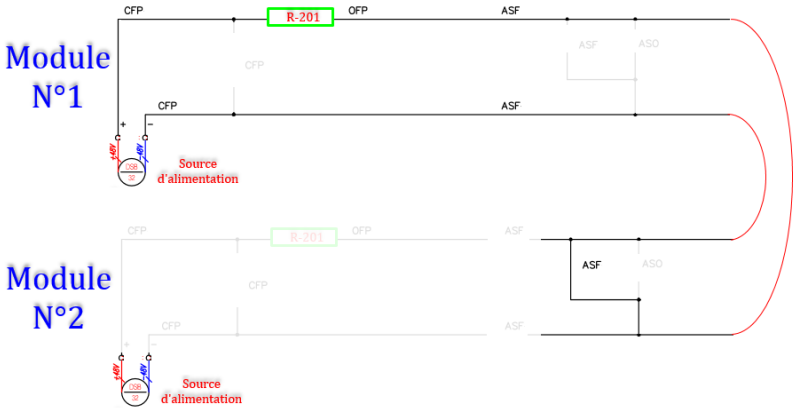
Figure 12 : exemple de boucle FP constituée de deux modules d'asservissement

Cette boucle ne peut être alimentée que par un module d'asservissement à la fois (l'un ou l'autre), dans le cas contraire les relais R-201 de chaque module seront désexcités. Pour mieux comprendre le principe, étudions quelques cas de figures possibles :

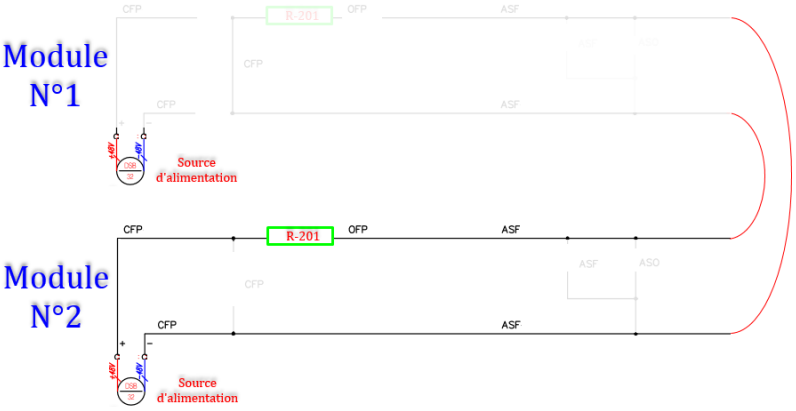
Cas 1 : La boucle est alimentée par le module d'asservissement N°1, le second module est en mode récepteur. Le relais R-201 de chaque module est excité.



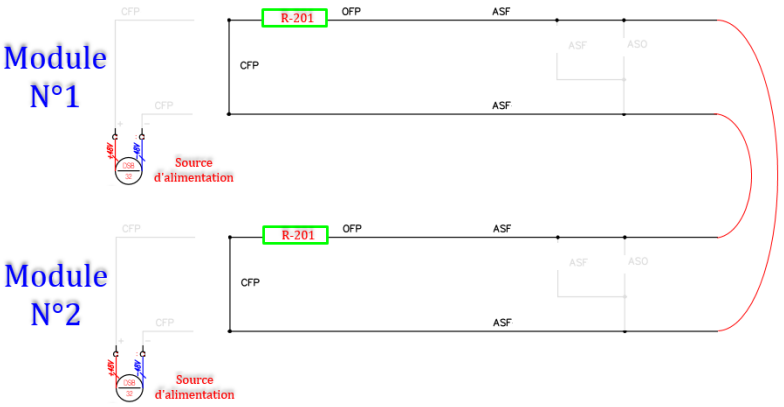
Cas 2 : Le module N°1 alimente toujours la boucle FP, le module N°2 est par contre shunté et ne reçoit plus de courant. La boucle FP étant toujours fermée, le courant I circule mais uniquement aux bornes du module N°1. Par conséquent seul le relais du module N°1 est excité.



Cas 3 : Le module N°2 tente d'alimenter la boucle FP, mais le module N°1 a ouvert son circuit FP. Plus aucun courant ne circule désormais, et ce jusqu'à ce que la boucle soit refermée et alimentée par un module.



Cas 4 : Aucun module n'alimente la boucle FP, par conséquent aucun courant ne circule même si la boucle reste toujours fermée. Aucun relais n'est donc excité.



Ces cas de figures ci-dessus illustrent le principe d'alimentation de la boucle FP. Ce fonctionnement devra être généralisé dans le cas de 3 ou plusieurs modules FP bouclés en série.

Ces modules d'asservissement n'apparaitront pas dans l'IHM 1, et ne sont utilisables qu'en mode 2 pendant la simulation (remarque valable pour tous les types de module d'asservissement).

HISTORIQUE DES MODIFICATIONS

	VERSION	AUTEUR	DESCRIPTION
19/03/2018	0	Mohamed NIANG	Création du document

CONTRIBUTEURS

DESCRIPTION	CONTRIBUTEUR	FONCTION	DATE ET SIGNATURE
Auteur	Mohamed NIANG	Doctorant	
Vérificateur	Raphaël COUPAT	Référent Principes Automatismes	
	Alexandre BOUAZIZ	Ingénieur Automatisme, Protection et Principe BT	
Approbateurs	Raphaël COUPAT	Référent Principes Automatismes	
	Alexandre BOUAZIZ	Ingénieur Automatisme, Protection et Principe BT	

DOCUMENTS JOINTS

DOCUMENT	DESCRIPTION

Résumé

La SNCF cherche à mettre en place des solutions innovantes permettant d'améliorer la sécurité et les conditions de travail des chargés d'études lors des travaux d'automatisation des EALE (Équipements d'Alimentation des Lignes Électrifiées). En partant de l'étude théorique du projet jusqu'à sa validation sur site, en passant par la mise en œuvre des programmes, du câblage des armoires, et de leur vérification sur plateforme et en usine, ces différentes tâches s'avèrent souvent être longues, complexes, et répétitives, sans compter les risques d'erreurs humaines.

En vue d'améliorer les conditions de travail des chargés d'études, ce projet de recherche vise principalement à améliorer leurs méthodologies de vérification et de validation (V&V) des systèmes de contrôle commande (programmes automates et câblage des armoires de contrôle commande). Ce projet intitulé « Vérification formelle et simulation pour la validation des systèmes de contrôle commande des EALE » est basée sur l'utilisation des méthodes formelles et du Virtual Commissioning (mise en service virtuel), il se décompose en deux axes :

- la vérification hors ligne des programmes API : basée sur une approche formelle, la méthode s'appuie sur une modélisation de l'installation électrique, des programmes API et du cahier de recette dans le model-checker Uppaal. Le principe consiste à vérifier automatiquement que les programmes respectent les spécifications fonctionnelles du cahier des charges, tout en étant formellement sûrs de fonctionnement.

- la validation en ligne des programmes et du câblage des armoires de contrôle commande, grâce à l'utilisation du Virtual Commissioning en mode ***Software-In-the-Loop*** (pour la validation des programmes) puis en mode ***Hardware-In-the-Loop*** (pour la validation du câblage des armoires). La validation se fera de manière automatisée.

Mots clé : Contrôle commande, EALE, sûreté de fonctionnement, vérification formelle, validation, simulation de partie opérative, filtre de sécurité

Abstract

In order to keep its leadership in French rail market and to improve working conditions of its systems engineers during automation projects, SNCF (French Railway Company) wants to develop solutions increasing the productivity. One of these improvements focuses on the current methodology used by the systems engineers to verify and validate the control system of Power Supply Equipments of the Electric Lines (PSEEL) during aan automation project. This task remains one of the most important during an automation project because it is supposed to ensure installations safety, but it should be optimized. Through an industrial thesis launched by SNCF engineering management and CReSTIC laboratory (of University of Reims), the aim of this research project is to improve this method and reduce time validation of control system by providing tools which will help systems engineers to formally verify and validate quickly and automatically the control command system during any automation project. It is composed of two axes :

- Offline verification of PLC programs, based on formal methods. This phase allows to verify if the control program satisfies the safety requirements, while guaranteeing the safety of installations.

- Online validation of control program and electrical cabinets' wiring, based on virtual commissioning techniques.

Keywords : Control system, PSEEL, safety, formal verification, validation, Virtual Commissioning, safety filter

Discipline : AUTOMATIQUE, SIGNAL, PRODUCTIQUE, ROBOTIQUE

Spécialité : Automatique et Traitement du Signal

Université de Reims Champagne-Ardenne

CRESTIC - EA 3804

Sciences Exactes et Naturelles, Moulin de la Housse, 51867 Reims, France