# Vérification de propriétés quantitatives des systèmes logiques par model-checking hybride

Zulema Juarez Orozco

*N° ENSC-2008/109*

# THESE DE DOCTORAT
# DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

Présentée par

Zulema JUAREZ OROZCO

**pour obtenir le grade de**

**DOCTEUR DE L'ECOLE NORMALE SUPERIEURE DE CACHAN**

Domaine :
**ELECTRONIQUE–ELECTROTECHNIQUE–AUTOMATIQUE**

**Sujet de la thèse :**

## Vérification de propriétés quantitatives des systèmes logiques par model-checking hybride.

Thèse présentée et soutenue à Cachan le 20 juin 2008 devant le jury composé de :

| | | |
|---|---|---|
| José PINEDA CASTILLO | Professeur – CIATEQ, Querétaro | Président |
| Hervé GUEGUEN | Professeur – SUPELEC Rennes | Rapporteur |
| Ernesto LOPEZ MELLADO | Professeur – CINVESTAV, Guadalajara | Rapporteur |
| Claire VALENTIN | Professeure – UCB Lyon1, Lyon | Examinatrice |
| Jean-Jacques LESAGE | Professeur – ENS de Cachan | Directeur de thèse |
| Bruno DENIS | Maître de conférences – ENS de Cachan | Co-encadrant |

Laboratoire Universitaire de Recherche en Production Automatisée
(ENS CACHAN / Université Paris-Sud 11 / EA 1385)
61, avenue du Président Wilson, 94235 CACHAN CEDEX (France)

*A Elena y Joel*
*mis padres*

# *Remerciements*

Le travail de recherche exposé dans ce mémoire de thèse a été réalisé au sein du Laboratoire Universitaire de Recherche en Production Automatisée (LURPA - EA 1385) de l'Ecole Normale Supérieure de Cachan.

Je tiens tout d'abord à adresser mes sincères remerciements au Professeur Jean-Jacques Lesage pour m'avoir accueilli au sein du laboratoire et pour avoir assumé la direction de mes travaux et pour la qualité de ses remarques.

J'ai également été encadrée et suivi par Bruno Denis. En plus de leurs grandes compétences scientifiques et techniques (qui m'ont beaucoup aidé tout au long de ma thèse), je tiens surtout à le remercier pour son support et son contact humain.

Je remercie Messieurs les Professeurs Hervé Guéguen et Luis Ernesto Lopez-Mellado de me faire l'honneur d'être rapporteurs de cette thèse. Je remercie également Madame le Professeur Claire Valentin et Monsieur le Professeur José Pineda Castillo pour m'accorder le privilège d'être membres de mon jury.

J'adresse également mes sincères remerciements au Conseil National de Sciences et de Technologie du Mexique (CONACYT) pour son soutien économique, ce qui m'a permis de mener à bien mon séjour et mon travail en France.

Je voudrais également remercier le CIATEQ pour la confiance qu'ils m'ont accordée et leur soutien toujours pertinent.

Pour son assistance dans la résolution de problèmes informatiques, l'installation des logiciels et surtout pour ses qualités humaines et son amitié, merci Marc. Mes remerciements vont également à tous mes collègues du Laboratoire pour les conseils techniques et scientifiques et pour les discussions sympathiques. Je tiens aussi à exprimer ma reconnaissance à Françoise et Karina pour leur aide administrative et leur chaleur humain.

Pour son soutien inconditionnel, son aide invaluable et sa charmante manière d'être avec moi, merci Thomas.

J'accorde enfin une attention particulier à toutes les personnes qui comptent tant pour moi : Merci à ma famille pour m'avoir soutenu pendant toutes ces longues années passées en France. Siempre los tengo cerca de mi corazón. Merci Catherine pour ta sollicitude et aide non seulement grammaticale mais aussi pour toutes tes attentions.

Merci à tous mes amis. Lizbet et sa famille pour être toujours là pour moi. Israel pour son amitié très chère et sa bonne humeur. Haydée compañera de aventuras pour tous les moments partagés. Jorge, qui a toujours pensé que « El Labo » c'était un bar, pour la musique. Mes amis du Mexique : Noé, Cuahutémoc et Sergio pour penser à

moi. Paty et Takeshi pour ses encouragements et toutes ces soirées délicieuses. Merci également à tous mes amis au laboratoire : Gaëlle, Sylvain, Vincent, Steve, Silvain, Yann, Pierre, Guillaume et Matthieu pour toute votre aide et compagnie.

# *Table des matières*

# *Introduction*

Le respect des impératifs de sûreté de fonctionnement (SdF) des systèmes industriels, et tout particulièrement des systèmes critiques, impose que soient mises en place des approches globales pour la SdF (de manière à ne pas laisser de maillon faible) qui englobent l'ensemble des activités d'ingénierie, puis, après la mise en service, l'ensemble des activités d'exploitation et de maintien en condition d'exploitation opérationnelle du système (voir à ce sujet la norme IEC 61508 « Sécurité fonctionnelle des systèmes électriques / électroniques / électroniques programmables relatifs à la sécurité » [IEC98]). C'est pourquoi la communauté scientifique s'est employée à procurer des modèles, des méthodes et des outils ayant comme objectifs complémentaires de prévoir les dysfonctionnements (fiabilité prévisionnelle, AMDEC, ...), de tolérer des dysfonctionnements (redondance, modes dégradés ou reconfiguration, ...), ou de démontrer l'obtention d'un niveau requis de SdF (spécifications formelles prouvables, crash tests, ...). Parmi les contributions majeures de la communauté scientifique à ce dernier objectif, les techniques de vérification formelle, qui sont le plus souvent associées au model-checking [BCM+92], sont aujourd'hui largement matures et commencent à être pratiquées dans l'industrie. Ainsi pour les Systèmes à Evénements Discrets (SED), même si d'irréfutables problèmes de robustesse des techniques de model-checking à la taille des modèles sont rencontrés, bon nombre de propriétés de sûreté ou de vivacité peuvent être prouvées sur un système automatisé. Ces preuves peuvent être obtenues en prenant en compte ou non un modèle du système contrôlé (approches "model-based" ou "non model-based"), en prenant en compte ou non le temps (model-checking temporisé ou non), et lorsque les exigences de sûreté de fonctionnement le permettent en prenant en compte le caractère probabiliste des phénomènes modélisés (model-checking probabiliste).

En nous appuyant sur les acquis de la communauté scientifique, et notamment sur ceux du LURPA, en matière de vérification des SED par model-checking, l'objectif de nos travaux est de nous intéresser à la preuve des propriétés relatives à la qualité du service rendu par le système automatisé ; propriétés que nous appellerons par la suite propriétés quantitatives. En effet, par exemple, au lieu de vérifier que plusieurs produits ont effectivement été dosés avant d'enclencher un mélange puis une réaction chimique, il peut être important de prouver que la bonne quantité de ces produits a été dosée. Autre exemple, au lieu de prouver qu'un mobile s'arrête dans une position donnée, il peut être important de prouver que cet arrêt en position s'effectue avec une précision garantie. Ce sont de telles propriétés quantitatives que nous nous sommes attachés à être capable de prouver pour les SED, et plus exactement pour une sous-classe des SED : les systèmes logiques.

Pour ce faire, il convient avant tout de rappeler qu'aucun système contrôlé (également appelé processus) n'est purement discret. C'est l'abstraction qu'est amené à en faire l'automaticien pour satisfaire les exigences du cahier des charges qui confère ou non au système bouclé contrôleur / processus cette qualification de SED. Ainsi, dans le domaine des systèmes mécatroniques par exemple, de nombreux processus possèdent des axes de mobilité servant à la circulation des produits, à leur positionnement,... et de ce fait les grandeurs physiques du processus, contrôlées par le contrôleur, sont des déplacements ou des vitesses. Selon le niveau de qualité qu'il est nécessaire de garantir pour ces grandeurs contrôlées, selon l'hostilité ou la variabilité du milieu extérieur (type et importance des perturbations, intervalle de variation des paramètres de la loi de mouvement,...), et selon le poids des contraintes économiques, l'automaticien est amené à faire le choix d'une commande asservie ou d'une commande discrète pour chacun de ces axes. Ce sont ces choix d'automatisation qui feront des grandeurs physiques observées et commandées par le contrôleur, des variables de commande continues ou discrètes. Dès que la qualité de positionnement requise pour les axes de déplacement le permet, et sous réserve que les perturbations qu'ils subissent soient tolérables, l'automaticien choisit une commande discrète pour des raisons évidentes de limitation des coûts. Une telle commande discrète de déplacements, si elle convient fréquemment aux exigences de qualité du positionnement des mobiles, n'en est pas moins une abstraction, nécessairement réductrice, des grandeurs physiques qui lui sont associées. Ainsi, la commande tout ou rien d'un déplacement linéaire entre deux positions extrêmes, observées grâce à deux détecteurs de fin de course, ne permet naturellement pas de connaître la précision du positionnement du mobile. Pour être à même de vérifier des propriétés relatives à la qualité de service rendue par le système automatisé, il va donc être nécessaire de tenir compte de la véritable nature des grandeurs physiques qui contribuent à élaborer la valeur ajoutée sur les produits traités (déplacement, vitesse, volume de liquide, ...) de ce fait il sera nécessaire de restituer son véritable caractère continu, ou hybride, au système contrôlé. La vérification des propriétés quantitatives des SED nous amène donc inévitablement à un problème de model-checking hybride dans lequel le contrôleur a un comportement discret (temporisé ou non) et le système contrôlé a un comportement hybride.

La vérification par model-checking des Systèmes Dynamiques Hybrides (SDH) est aujourd'hui encore un problème plus difficile que pour les SED. En tout premier lieu, dans le cas général, il s'agit d'un problème non décidable. Par ailleurs, les dynamiques continues ne peuvent dans la pratique être décrites que par des comportements élémentaires (de type équations différentielles linéaires du premier ordre à coefficient constant) et les problèmes d'explosion combinatoire surviennent bien plus rapidement encore que pour les SED. Les progrès récents de la communauté scientifique dans ce domaine ont néanmoins abouti à l'élaboration de model-checker polyédriques, tels que PHAVer [Fre05b], dont nous allons montrer que les fonctionnalités et la puissance nous permettent de traiter des problèmes de taille significative.

En nous intéressant à la vérification des propriétés quantitatives des SED par model-checking hybride, nous visons en fait un double objectif. Le premier, le plus direct, est de contribuer à une meilleure maîtrise de la SdF par la capacité à prouver une nouvelle classe de propriétés qui conditionnent dans certains cas directement la sûreté des systèmes de production : les propriétés relatives à la qualité de service. Le second objectif est de contribuer à l'optimisation des SED dans le sens où la vérification de

propriétés quantitatives par model-checking hybride passe par la détermination des régions atteignables. Nous allons montrer, au travers du traitement d'un exemple de système de production par lots, que l'étude de cette région atteignable permet, en plus de la preuve de propriétés, de déduire des stratégies optimales de contrôle.

Ce mémoire de thèse est découpé en quatre chapitres, qui nous permettent d'exposer successivement la problématique de nos travaux, nos contributions théoriques et méthodologiques ainsi que l'utilisation de ces contributions au travers de deux études de cas.

Plus précisément, le premier chapitre soulève la question de l'utilisation des modèles hybrides pour la vérification de systèmes logiques. Une analyse bibliographique sur la vérification formelle de contrôleurs logiques est présentée. Après avoir rappelé les principes des méthodes de vérification formelle, et en particulier des techniques de model-checking, une synthèse des travaux visant à l'utilisation de ces techniques pour l'analyse des contrôleurs logiques est effectuée. Nous nous intéressons plus spécifiquement à l'évolution de ces travaux en fonction du formalisme de modélisation utilisé. Ce chapitre s'intéresse également aux approches de modélisation modulaire ainsi qu'à la modélisation de l'interaction entre le contrôleur et le système contrôlé.

Le deuxième chapitre nous permet de discuter le besoin d'une approche modulaire pour la modélisation des systèmes hybrides et accorde un intérêt tout particulier aux mécanismes d'interaction inter-modules. Nous proposons ensuite une représentation graphique, inspirée des blocs fonctionnels de la norme CEI 61499 [IEC04], que nous avons retenue pour faciliter une modélisation structurelle modulaire des SDH. Pour leur modélisation comportementale, nous proposons ensuite une classe d'automates hybrides à synchronisations typées qui nous permet de conserver une approche modulaire. L'intérêt de cette classe d'automates est mis en évidence au travers du traitement d'un exemple.

Le troisième chapitre est consacré à la modélisation générique des SDH en vue de leur vérification. Une fois défini le formalisme de modélisation, la mise en place d'une méthodologie permettant de faciliter la modélisation des systèmes automatisés est présentée. L'ensemble des hypothèses que nous avons retenues pour ces travaux est également détaillé.

Dans le quatrième chapitre, nous présentons deux études de cas. La modélisation en est faite conformément à la méthode développée dans le chapitre 3 et aux formalismes définis dans le chapitre 2. La vérification formelle de propriétés quantitatives est réalisée grâce au model-Cheker polyédrique PHAVer, l'ensemble des résultats obtenus est ensuite discuté.

Finalement, après avoir présenté une synthèse des résultats obtenus, nous conclurons et dévoilerons quelques pistes pour des travaux futurs prolongeant ce travail de thèse.

# *Chapitre 1*

# *Pourquoi des modèles hybrides pour vérifier des systèmes logiques ?*

---

*Notre travail de recherche s'intéresse à la vérification formelle de systèmes automatisés à contrôleur logique dont les propriétés à vérifier concernent la qualité du service rendu par le système. Dans ce chapitre seront présentés le contexte de nos travaux et une analyse bibliographique sur la vérification formelle des contrôleurs logiques qui pilotent les systèmes automatisés. Nous allons discuter de l'intérêt de coupler un modèle hybride du processus à un modèle logique de la commande pour atteindre les grandeurs physiques qui portent l'information pertinente vis-à-vis de la vérification. Dans notre travail, nous considérons les interactions entre le modèle de la commande et le modèle du processus. Pour cette raison, une analyse bibliographique sur les approches de modélisation modulaires sera également abordée. Finalement, nous allons conclure ce chapitre par la présentation des apports de cette thèse.*

## 1.1 Contexte et problématique

De manière générale, un système automatisé reçoit un flux de matière ou de produits et génère un flux de produits plus élaborés (moulés, usinés, assemblés, testés, etc.). Il doit également gérer l'alimentation en énergie, ainsi que des flux auxiliaires tels que les consommables ou les déchets, tout en minimisant les stocks initiaux, finaux et intermédiaires. Tout cela, ajouté à des exigences sans cesse accrues de qualité, de sécurité et de flexibilité, impose le développement de systèmes de commande de plus en plus fiables pour lesquels toutes ces exigences doivent être satisfaites.

La structure des système automatisé, montrée dans la figure 1.1, est composée d'une partie opérative (processus) et d'une partie commande (contrôleur). Le contrôleur traite les informations provenant de capteurs installés dans le processus et réagit en transmettant des ordres aux pré-actionneurs qui agissent sur le processus. L'évolution du processus peut impliquer des grandeurs continues et discrètes qui seront acquises par les capteurs afin d'être transmises au contrôleur. Le contrôleur, à son tour, communique avec l'opérateur qui établit les consignes à respecter par le système automatisé.

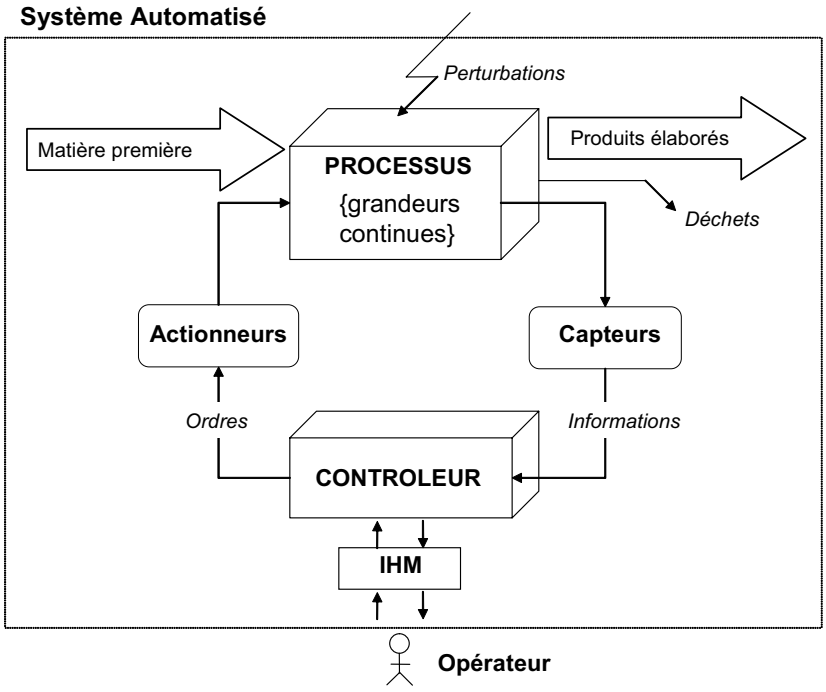



FIG. 1.1 – Structure des systèmes automatisés

Les systèmes automatisés sont susceptibles de subir des erreurs, par exemple, le mauvais fonctionnement de l'un des composants, la non émission d'un ordre de la part du contrôleur, etc. Ces erreurs ont des conséquences beaucoup plus graves lorsqu'elles se produisent dans un système amené à interagir avec un environnement physique mettant en oeuvre des quantités d'énergie importantes ou manipulant des produits dangereux, par exemple. Dans ces applications dites critiques, une mauvaise consigne du système de contrôle-commande peut engendrer des défaillances conduisant à l'arrêt de production dans une usine, à l'endommagement de composants ou, dans des cas extrêmes, à des accidents. La criticité de ces systèmes rend impérative l'utilisation

d'approches rigoureuses et/ou formelles pour leur conception et leur vérification.

Par ailleurs, la nature hybride des processus physiques (interaction de grandeurs continues et discrètes) amène souvent l'automaticien à faire un choix des technologies à utiliser pour leur commande. Cela dépend du cahier des charges, des spécifications à respecter et des contraintes économiques. À titre d'exemple, prenons le cas du remplissage d'un réservoir dont la grandeur continue à commander est le niveau d'eau (fig 1.2). Selon la qualité qu'il est nécessaire de garantir pour cette grandeur et selon l'importance des contraintes économiques, l'automaticien peut choisir, par exemple, soit une commande asservie, soit une commande discrète en boucle ouverte. Cette décision a un impact direct sur le choix des capteurs et des actionneurs. Comme l'illustre la figure 1.2, ce sont ces choix qui feront des grandeurs observées et commandées par le contrôleur des variables continues ou discrètes.

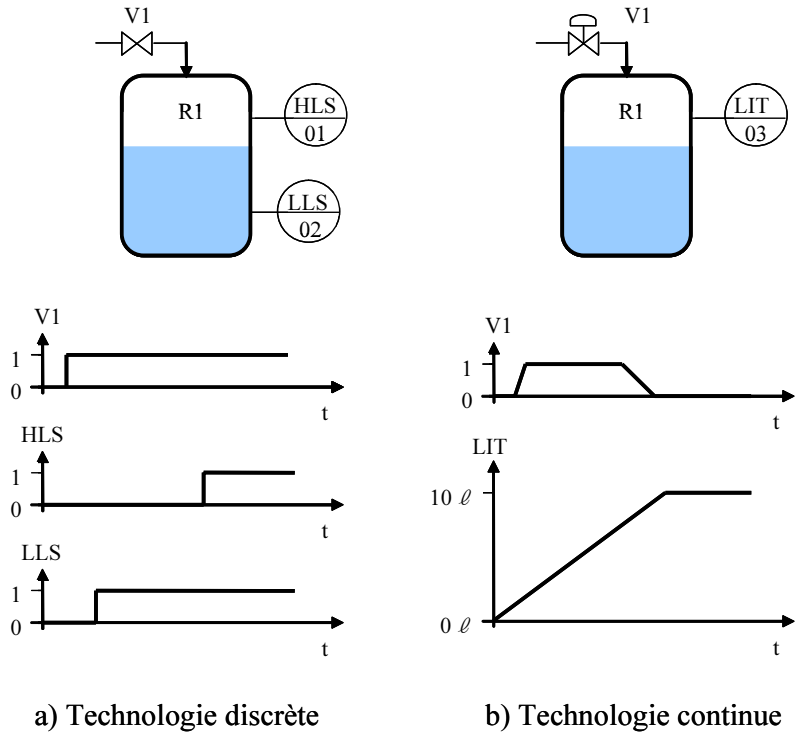



FIG. 1.2 – Deux technologies différentes pour un même exemple : le remplissage d'eau du réservoir R1

Par exemple, une commande Tout ou Rien (TOR) de la vanne V1 peut se satisfaire d'une mesure logique du niveau du réservoir R1 prise par les capteurs HLS (« high level switch ») et LLS (« low level switch »), qui permet de connaître les seuils de haut et bas niveaux (fig 1.2a). Par contre, une mesure continue du niveau de R1 par l'intermédiaire du capteur LIT (« level indicator transmitter ») permettrait aisement d'asservir le niveau de liquide dans le réservoir (fig 1.2b). Lorsqu'une commande TOR satisfait les exigences requises par le cahier des charges, l'automaticien la choisit alors souvent pour des raisons de limitation de coûts. Les capteurs, actionneurs et pré-actionneurs pour l'asservissement sont en effet plus onéreux.

Il est clair qu'une telle commande TOR du niveau nécessite une abstraction logique des grandeurs physiques qui lui sont associées et, même si elle permet de vérifier si le

réservoir est plein ou vide grâce aux détecteurs de seuils, elle ne permet d'exprimer ni le temps de remplissage/vidange du réservoir, ni la qualité de la mesure du niveau. En effet, dès que l'on s'attache à mesurer la qualité du service d'un système, il est légitime de se demander si les abstractions discrètes des grandeurs physiques sont pertinentes vis-à-vis de la propriété à vérifier.

Les systèmes automatisés sont en général élaborés à partir de besoins fonctionnels, c'est-à-dire en fonction de l'action attendue du contrôleur. Prenons l'exemple d'une machine à café : sa spécification fonctionnelle est de faire du café. Cependant, si la machine à café met trop de temps à faire le café, le client ne sera pas satisfait et la prochaine fois, il utilisera une autre machine qui prendra moins de temps. Cette spécification de temps sur le comportement fonctionnel de la machine est une spécification non fonctionnelle communément appelée qualité de service.

C'est pour cela que, lorsque l'on s'intéresse à la vérification de propriétés visant la qualité du service rendu par un système commandé par une commande discrète, il est nécessaire de connaître le comportement continu des grandeurs physiques du processus.

Ainsi, puisque l'on s'intéresse aux processus physiques dont les grandeurs physiques sont continues, commandées par des contrôleurs logiques, il est nécessaire de prendre en compte leurs dynamiques continues et discrètes afin de garantir le bon fonctionnement du système global. On pourra ainsi concevoir des systèmes plus performants.

De ce fait, il existe différents problèmes pratiques pour lesquels des approches soit purement discrètes, soit purement continues, ne répondent pas toujours de manière satisfaisante à l'analyse de systèmes hybrides impliquant des dynamiques continues et discrètes. Parmi ceux-ci, on peut citer :

- La taille des modèles à manipuler, et notamment le nombre de grandeurs qui interviennent et leurs interactions avec le système de commande.
- La nature des spécifications que le système doit garantir. Le besoin de plus en plus exigeant de concevoir des systèmes automatisés performants oblige les automaticiens à s'intéresser non seulement à la vérification de propriétés logiques ou temporelles du contrôleur mais également à la qualité du service rendu par l'ensemble du système.

Pour apporter une réponse à ces problèmes, notre travail s'articule autour des deux axes suivants :

- ***La modélisation des systèmes hybrides***. L'interaction entre les composants ayant des comportements continus et discrets rend le processus de vérification complexe, particulièrement en ce qui concerne la modélisation de comportements hybrides. Cette complexité augmente avec la taille du système modélisé. C'est pour cela que nous proposons une approche de modélisation modulaire qui prend en compte deux aspects importants : le niveau d'abstraction et le niveau de granularité des modules. Nous présentons les paradigmes essentiels de la modélisation modulaire en section 1.3.1.
- ***La vérification formelle des systèmes hybrides***. Nous définissons la vérification formelle de manière générale comme une méthode permettant de garantir l'adéquation des systèmes par rapport à leurs spécifications, en vue de détecter les erreurs tout au long de la conception et développement des systèmes automatisés.

Nous allons maintenant examiner les principaux résultats obtenus par la communauté scientifique dans le domaine de la vérification formelle. Cela nous permettra

ensuite de positionner nos travaux et leurs apports.

## 1.2 Vérification des contrôleurs logiques

L'utilisation de méthodes formelles pour la vérification des systèmes à contrôleurs logiques peut être classée, d'après [FL00], selon les critères suivants :

1. La prise en compte ou non d'un modèle du processus :
   - « model-based » considérant un modèle de processus élaboré au travers d'un formalisme bien défini ([LES05], [Mac06]). Ce modèle peut être plus ou moins raffiné, selon les propriétés que l'on souhaite prouver ;
   - « non model-based » sans considération d'un modèle de processus ([SR02]) ;
2. Le niveau de détail du contrôleur logique :
   - prise en compte du moniteur d'exécution [MW99],
   - prise en compte du temps dans le modèle du programme de contrôle [KEH+98],
   - modélisation des programmes monotâches [LCRRL99] ou multitâches [MBG+05].
3. Le modèle formel adopté : c'est-à-dire, le langage de spécification qui va permettre de décrire le comportement des composants du système à vérifier ainsi que leurs interactions. Nous pouvons citer parmi ceux-ci : les « *Condition/Event Systems* » [SK91], les « *General Transition Systems* », les *Automates* à états finis [Moo56], temporisés [AD94] ou hybrides [Hen96] et différentes classes de *Réseaux de Petri* [DA92],[CGS07]. Ces deux derniers étant parmi les plus utilisés.
4. La technique d'analyse utilisée : dans le cadre des systèmes à événements discrets et de leurs extensions temporisées, un certain nombre de méthodes et d'outils ont été développés pour permettre de vérifier qu'un système satisfait certaines propriétés, de manière automatique et formelle. Parmi les techniques les plus utilisées, on trouve le « *Model-checking* » [McM93] et la *Manipulation algébrique* [RD02].

Sur chacun de ces quatre aspects, plusieurs problématiques ont été successivement abordées par les chercheurs. En effet, sur le premier point par exemple, la décision d'opter pour une approche « model-based » ou « non model-based » est fortement liée à la nature des propriétés à vérifier ([MDL06], [DLJO06]) et implique une modélisation plus ou moins fine du comportement ainsi qu'une définition claire de la frontière de modélisation (prise en compte du modèle du système physique, du temps, du moniteur d'exécution, ...). Nous analyserons plus en détail quelques résultats majeurs de la vérification formelle des systèmes logiques dans la section suivante, mais dans l'immédiat, examinons la nature des propriétés à vérifier.

Comme l'illustre la figure 1.3 on peut classifier les propriétés attendues d'un système réel de la manière suivante :

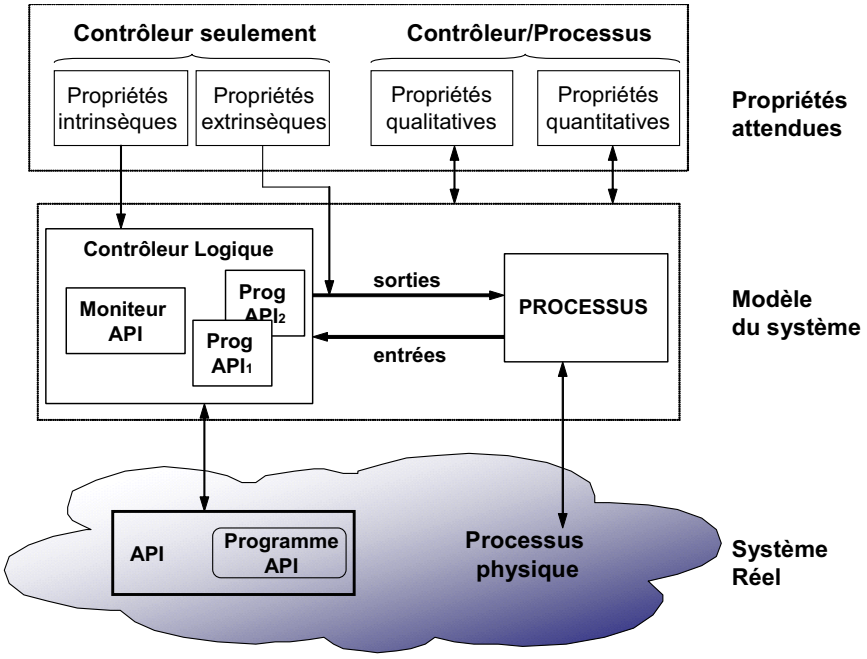



FIG. 1.3 – Classification des propriétés vérifiables dans un système automatisé

**Les propriétés concernant le contrôleur logique seulement.**

- ***Propriétés intrinsèques :*** ces propriétés visent à caractériser un comportement interne de la commande comme proposé par [Huu05]. Ceci comprend notamment les sauts conditionnels invariants, les codes inatteignables, les boucles infinies ou les calculs ou codes inutiles.
- ***Propriétés extrinsèques :*** ces propriétés portent sur les actions de commande sur le processus, et non sur le comportement interne de la commande. [BBF+01] identifie les propriétés suivantes : les propriétés de sûreté, de vivacité, d'équité ou d'atteignabilité.

**Les propriétés concernant le système bouclé Contrôleur/Processus.**

- ***Propriétés qualitatives :*** telles que le comportement logique du programme du contrôleur (absence de blocage, sûreté, vivacité, etc.).
- ***Propriétés quantitatives :*** telles que les performances et la sûreté de fonctionnement du système (temps de réponse, débits, taux de pertes, répétitivité, qualité du résultat exprimé sous forme de précision, etc.).

En ce qui concerne les techniques de vérification formelle, la preuve de théorèmes (« Theorem proving ») nécessite une grande connaissance des systèmes formels (ce qui implique leur utilisation par des personnels hautement qualifiés) et l'on n'a jamais la certitude de pouvoir mener à bien une démonstration. L'avantage principal de cette technique est d'éviter l'explosion combinatoire, problème récurrent en model-checking. Cependant, elle demande une forte expertise et les méthodes automatiques ne s'appliquent qu'à des systèmes simples. Par ailleurs, le « model-checking » construit sa vérification à partir d'un automate à états modélisant le système étudié et d'un ensemble de propriétés à vérifier exprimées dans un formalisme adéquat. Le model-checker vérifie alors automatiquement les propriétés et permet de savoir si la propriété est res-

pectée. Cependant, la taille importante des systèmes industriels à manipuler augmente le problème d'explosion combinatoire, faisant de la vérification une tâche difficile.

Dans ce mémoire, nous nous intéressons à la vérification par « model-checking », en considérant d'une part son caractère automatique et d'autre part le développement croissant d'outils informatiques de plus en plus performants. Le principe du model-checking peut s'exprimer de la manière suivante (voir fig 1.4) : soit un système qui doit vérifier un ensemble de propriétés. La première tâche consiste à formaliser le comportement du système (soit le contrôleur logique, soit l'ensemble {contrôleur logique + processus}) grâce à un langage de spécification choisi, et à formaliser les propriétés à vérifier dans un langage formel. Le model-checker réalise ensuite une analyse exhaustive de l'espace d'états accessibles par le modèle du système à vérifier qui permet de prouver que les propriétés attendues sont vérifiées ou non.

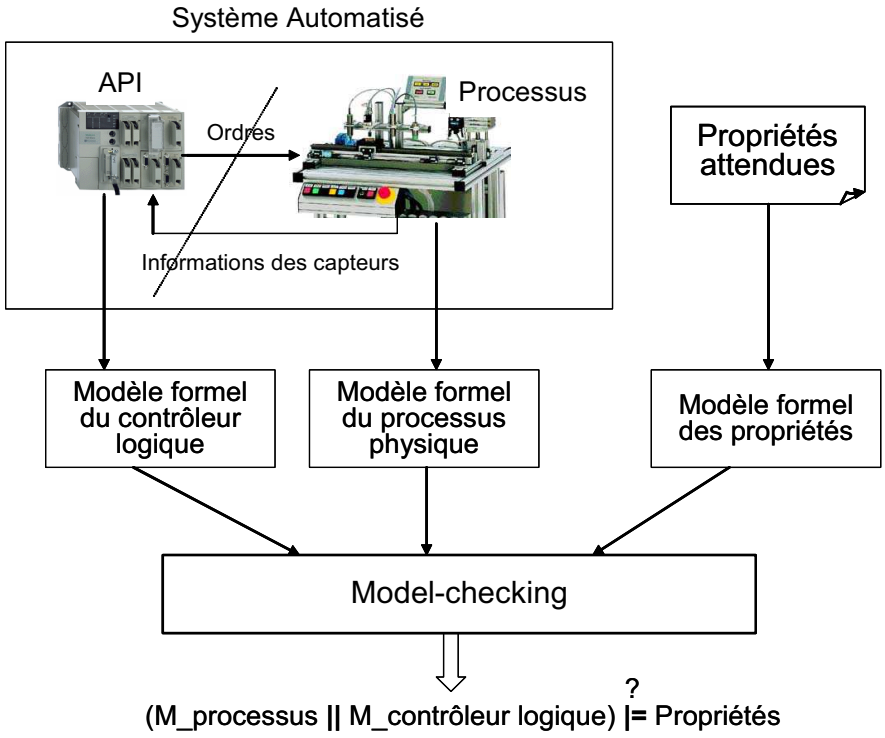



FIG. 1.4 – Démarche de vérification formelle par « model-checking »

De nombreux travaux concernant la vérification de systèmes à contrôleur logique ont été conduits par les chercheurs. La plupart de ces travaux sont inspirés par des approches informatiques qui utilisent les *systèmes à événements discrets (SED)* et leurs extensions temporisées ainsi que les *systèmes dynamiques hybrides (SDH)*. Dans la suite, nous allons décrire l'évolution de ces travaux en fonction des caractéristiques sémantiques du **formalisme de modélisation** utilisé pour vérifier les différents types de propriétés dans les systèmes automatisés.

### 1.2.1 Approches avec modèles non temporisés

La vérification formelle de contrôleurs logiques peut être réalisée par de nombreuses méthodes présentées et classifiées dans [Mad00, FL00]. Les premières approches ont

le plus souvent été limitées au model-checking non temporisé, sans modélisation du processus contrôlé en prenant en compte dans certains cas le cycle du moniteur du contrôleur. On peut citer à ce titre [MPBC92] qui a appliqué le « model-checker » symbolique développé par [BCM+92], où le système à vérifier est modélisé par des automates à états finis et les spécifications du comportement désiré sont représentées par des expressions en logique temporelle. Il s'agit d'une des premières publications qui a abordé le problème de traduction du comportement des contrôleurs industriels composés d'un programme et d'un moniteur d'exécution dans un langage compatible avec les outils de model-checking. Comme présenté dans [LCRRL99], plusieurs travaux postérieurs ont permis de tenir compte du cycle d'exécution du moniteur ainsi que du programme de contrôle dans les outils de model-checking pour vérifier des propriétés de sûreté et de non blocage dans les programmes.

Un peu plus tard, des travaux comme [DSLR01] considèrent le problème de la vérification de programmes spécifiés dans un des langages proposés par la norme IEC-61131-3 [IEC93] et leur pertinence avec le comportement du processus. Plus récemment, des travaux dans ce domaine comme ceux de [Mac06] montrent l'intérêt de la prise en compte du modèle du processus, aussi bien pour améliorer le résultat des preuves que pour permettre la vérification de propriétés non démontrables sans un tel modèle. Les propriétés concernées portent surtout sur le comportement logique du contrôleur : absence de blocage, sûreté, vivacité ou accessibilité.

On peut estimer aujourd'hui que bon nombre de problèmes scientifiques relatifs au « model-checker » non temporisé des systèmes logiques sont résolus, que ce soit par approche « model-based » ou « non model-based ». Subsistent naturellement des problèmes opérationnels de robustesse à la taille des modèles [Gou07] ou d'écriture des propriétés [BS07], par exemple.

### 1.2.2 Approches avec modèles temporisés

Pour certaines applications industrielles à contrôleurs logiques, la synchronisation correcte des sorties logiques dépend de la dynamique continue du processus. Par exemple, lorsque se produit une réaction chimique, le contrôleur doit attendre le temps que dure cette réaction avant d'émettre une sortie, ou lors d'un processus critique qui peut mettre en danger les utilisateurs, le contrôleur doit maintenir stable une sortie pendant un certain temps (un signal d'alarme soit sonore, soit visible). Dans ces cas, l'utilisation de temporisateurs dans la programmation du contrôleur logique est nécessaire.

L'utilisation de temporisations est également un moyen de représenter l'absence d'information sur les procédés physiques, comme celles qui sont fournies par les capteurs. Ainsi, la fin d'une temporisation peut représenter un signal d'entrée du contrôleur. Par exemple, dans le cas d'un processus chimique, lorsqu'une vanne est ouverte pour permettre le remplissage d'un réservoir, le processus continue une fois que le réservoir est rempli. Soit on utilise un capteur pour mesurer le niveau, soit on estime par d'autres moyens hors ligne connaître le temps de remplissage et on le traduit par une temporisation.

La vérification par « model-checking » de cette classe de comportements nécessite alors la prise en compte du temps dans les modèles, ce qui complexifie grandement le

problème de vérification de propriétés.

Dans ce cadre, [KEH+98] propose une formalisation par automates temporisés [AD94], tant pour le processus physique que pour le programme du contrôleur logique, mais sans prendre en compte le cycle du moniteur et en utilisant uniquement la partie du programme SFC (« *Sequential Function Charts* ») qui est pertinente vis-à-vis des propriétés à vérifier. Les propriétés à vérifier expriment des aspects quantitatifs liés au temps et sont vérifiées grâce à des modèles de procédés basés sur des abstractions temporelles.

Dans les travaux de [KGR94, KES99], les modèles du contrôleur et du processus sont construits de façon modulaire par blocs fonctionnels utilisant les systèmes Condition/Event proposés par [SK91]. Les spécifications sont vérifiées par l'analyse d'atteignabilité de la dynamique discrète temporisée. Les blocs représentant le comportement continu sont approximés par modèles discrets temporisés. Cette abstraction est inhérente à la procédure de vérification formelle. Cependant, ce type de formalisme ne permet pas la vérification automatique, c'est pour cela que [KER+99] proposent de transformer les timed condition/event systems dans le formalisme d'automates temporisés développé par [AD94] qui permet la vérification automatique. Les propriétés à vérifier portent essentiellement sur la sûreté de fonctionnement.

Si on s'intéresse aux travaux pour lesquels le « model-checking » a été utilisé pour vérifier non seulement des propriétés logiques (par exemple, l'exactitude du programme de contrôle) mais également pour trouver des séquences de programmes de contrôle optimales, on peut citer [BMF02]. Ce travail prend en compte un modèle temporisé du processus couplé à un modèle discret du programme de contrôle. Le modèle du processus est fait de façon modulaire, mais il continue à être une abstraction temporelle des grandeurs physiques du processus réel. De ce fait, cette abstraction permet l'analyse des études de cas de taille assez importante.

En résumé, le « model-checking » des systèmes temporisés reste aujourd'hui un problème scientifique ouvert. Des travaux récents, comme [LDL07], par exemple proposent des abstractions temporelles pertinentes qui permettent de réduire l'explosion combinatoire inhérente à la prise en compte du temps, mais beaucoup reste à faire dans ce domaine.

### 1.2.3 Approches avec modèles hybrides

Durant la dernière décennie, des recherches visant à modéliser un système continu qui puisse être vérifiable avec des méthodes automatiques ont conduit à proposer le formalisme d'automates hybrides de [Hen96] et l'outil de vérification automatique HyTech [HHWT97]. De nombreux travaux ont été développés depuis dans ce domaine, notamment au travers des projets européens ESPRIT et KONDISK [EFS02]. Ils utilisent des automates hybrides pour modéliser le comportement du processus couplé à un contrôleur logique décrit par des automates temporisés. L'analyse de ces systèmes porte sur la vérification des propriétés qualitatives par model-checking et des propriétés quantitatives par simulation.

Parmi les travaux qui ont étudié la pertinence d'utiliser les automates hybrides pour analyser des systèmes hybrides [ACHH93], on peut citer ceux développés par [SMF97].

Les auteurs ont utilisé le « model-checker » HyTech pour vérifier des propriétés (quantitatives) critiques de sûreté de fonctionnement dans un système où tant le processus que le contrôleur sont hybrides. Les résultats sont intéressants du point de vue de la modélisation car celle-ci est modulaire et permet d'aboutir à la vérification du système global.

Dans les travaux sur la vérification de contrôleurs logiques, on peut remarquer la préférence accordée au langage de programmation Sequential Function Chart (SFC) pour la modélisation du contrôleur logique. Dans le travail de [LES05], une approche pour formaliser le programme SFC en automate temporisé pour contrôler un processus continu et le vérifier a été développée. Quant au processus, il a été modélisé par automates hybrides avec des fonctions mathématiques linéaires. Récemment, [LSE06] a proposé une méthode systématique pour concevoir des contrôleurs logiques à partir de spécifications informelles. Le programme de contrôle est construit comme un SFC en transformant les spécifications via différents formats bien définis (fonctions de procédés et diagrammes de contrôles). Le SFC est ensuite traduit en automate temporisé, procédé systématique implémenté par [BHLE04] et [RLSE04]. Le processus est modélisé soit par des automates temporisés, soit par des automates hybrides. Finalement, l'analyse porte sur la vérification des propriétés de sûreté de fonctionnement.

Plus encore que le « model-checking » des systèmes temporisés, le « model-checking » des systèmes hybrides reste un problème difficile porteur d'enjeux scientifiques importants tels que la recherche d'atteignabilité, la décidabilité des algorithmes et l'explosion combinatoire [NLG06], [LG06].

### 1.2.4 Bilan

Nous nous intéressons dans ce mémoire à la vérification de contrôleurs logiques. D'après les travaux étudiés dans cette section (1.2), les propriétés qui ont été très peu étudiées sont les propriétés quantitatives visant à rendre plus performant le système automatisé, c'est-à-dire qu'au-delà de l'intérêt qu'il attache aux propriétés quantitatives concernant la sûreté de fonctionnement majoritairement lié au temps, notre travail cherche à valider les propriétés quantitatives qui ont un impact sur la qualité du service rendu : qualité de positionnement, robustesse à certaines perturbations, temps de production, taux de perte dans la production, etc. Ces propriétés nous aideront à prendre des décisions pour améliorer ou optimiser les systèmes. Pour cela, il est nécessaire de prendre en compte le processus (approches « model-based ») et d'étudier le système dans son ensemble : contrôleur logique en boucle fermée avec un modèle du processus physique.

D'une manière générale, la principale difficulté de la vérification par model-checking est de trouver un bon compromis entre la finesse de représentation des systèmes et la taille et la complexité du modèle. Plus le modèle est fin, plus les possibilités de vérification augmentent, mais plus le modèle est grand et complexe, plus le temps et la mémoire nécessaires à la vérification sont grands. Avec l'augmentation de la taille des systèmes à modéliser, la tâche de la vérification devient souvent difficile, et notamment parce que la modélisation du contrôleur devient plus complexe, par (voir fig 1.5) :

- les types de signaux qu'il doit gérer (non seulement discrets mais aussi continus),
- le ou les algorithmes de programmation qui assurent le comportement correct

du système par rapport à ces spécifications ainsi que leur ordre de séquence d'exécution, et
- la prise en compte explicite ou implicite du cycle du moniteur.

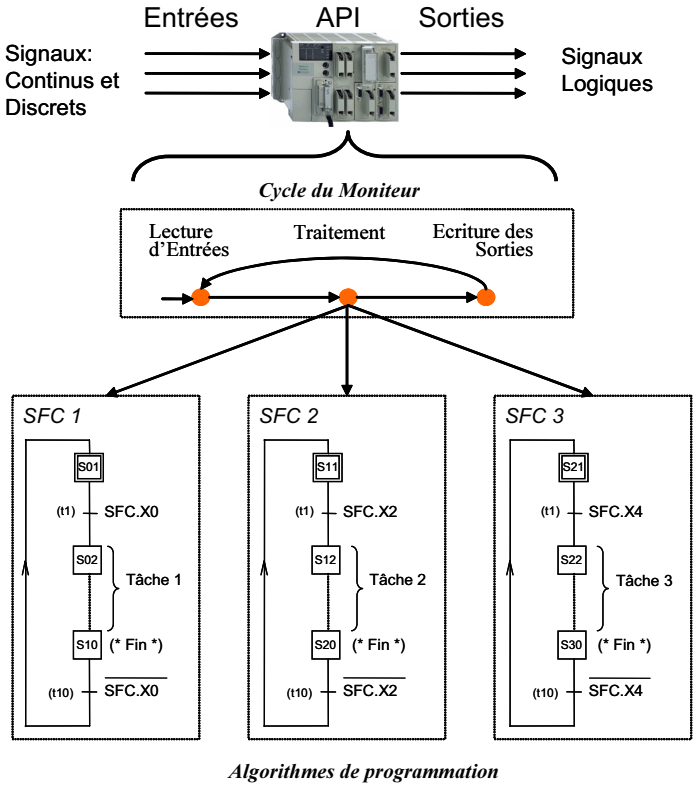



FIG. 1.5 – Complexité du contrôleur logique

La revue de littérature exposée précédemment a permis de mettre en évidence des avancées scientifiques importantes dans le domaine du « model-checking » des systèmes automatisés, réalisées depuis une dizaine d'années. Ces résultats ont été obtenus à la fois grâce à des contributions formelles (propositions de nouvelles classes de modèles, par exemple) et des contributions méthodologiques (techniques de traduction systématique de langages de programmation d'API, par exemple). Cependant, de notre point de vue, bon nombre de ces travaux gagneraient en pertinence, et donc en portée, par une prise en compte plus fidèle des caractéristiques et contraintes technologiques des systèmes modélisés.

Ainsi, par exemple, la plupart des travaux étudiés intègrent le cycle du moniteur dans le modèle du programme du contrôleur logique. Nous pensons que, pour mieux étudier les mécanismes de coordination entre le cycle du moniteur et les programmes ainsi que l'ordre de leurs exécutions, il est nécessaire de séparer le cycle du moniteur du programme de contrôle et de les modéliser séparément.

Par ailleurs, la modélisation des interactions entre le comportement du contrôleur et celui du processus n'est le plus souvent pas considérée. Dans les travaux qui utilisent un modèle du processus, il n'y a pas de réelle prise en compte des aspects technologiques du contrôleur et du processus, notamment de leurs caractéristiques physiques et comportementales. En effet, leurs évolutions dans le temps sont asynchrones mais

avec des instants de resynchronisation, et cela dans des échelles de temps complètement différentes entre un contrôleur très réactif et un processus physique lent par rapport au contrôleur.

Pour cette raison, nous considérons que la modélisation des systèmes à vérifier est très importante et a souvent été négligée. C'est pourquoi nous allons nous y intéresser dans la section suivante.

## 1.3 Modélisation des systèmes hybrides

L'interaction entre des composants ayant des comportements continus (processus) et discrets (contrôleur) rend la modélisation des systèmes hybrides auxquels on s'intéresse très complexe. Cette complexité augmente lorsque l'on considère des systèmes de taille significative. Pour répondre à ces difficultés, nous allons privilégier deux paradigmes de modélisation comportementale : la *modularité*, indispensable pour la modélisation de systèmes complexes, et les *mécanismes d'interaction* entre modules, inhérents à la modularité (synchronisations et échanges de données).

### 1.3.1 Modélisation modulaire

La modélisation modulaire consiste en la division du système en les parties qui le constituent (appelées modules). Un module est un élément du système, comme par exemple un réservoir, un capteur, une vanne, ..., modélisé à l'aide d'un formalisme donné. Le système global est obtenu à partir des modèles des composants en faisant coopérer entre eux ces différents modèles grâce à des mécanismes d'interaction.

Plusieurs approches permettant la modélisation modulaire de systèmes ont été proposées, la plupart étant basées sur la notion de bloc fonctionnel. Un bloc fonctionnel est une structure bien connue et employée couramment par les automaticiens. Il a été présenté pour la première fois dans la norme CEI 61131-3 [IEC93] sur les langages de programmation pour les contrôleurs logiques programmables (« PLC : Programmable Logic Controller »). La norme CEI 61499 [IEC04] a prolongé le concept de bloc fonctionnel de la norme CEI 61131-3 en s'inspirant des approches orientées objets. La norme CEI 61499 est une norme ouverte pour la commande et l'automatisation distribuées. Un bloc fonctionnel (figure 1.6) y est défini comme l'élément de base pour réaliser des applications complètes. Il existe deux types de blocs fonctionnels : les blocs fonctionnels de base et les blocs fonctionnels composés. Un bloc fonctionnel composé contient d'autres blocs fonctionnels composés et/ou d'autres blocs fonctionnels de base. Un bloc fonctionnel de base contient les algorithmes et un diagramme de commande d'exécution (ECC pour *Execution Control Chart*). Chaque bloc fonctionnel possède des entrées et des sorties d'événements aussi bien que des entrées et des sorties de données. Dans un bloc fonctionnel de base, l'exécution d'un algorithme est déclenchée par l'occurrence d'un événement d'entrée. L'algorithme exécuté produit de nouvelles données de sortie à partir des données d'entrée. Quand l'algorithme a terminé son exécution, un événement de sortie est émis. Cet événement de sortie peut alors devenir l'événement d'entrée d'un autre bloc fonctionnel.

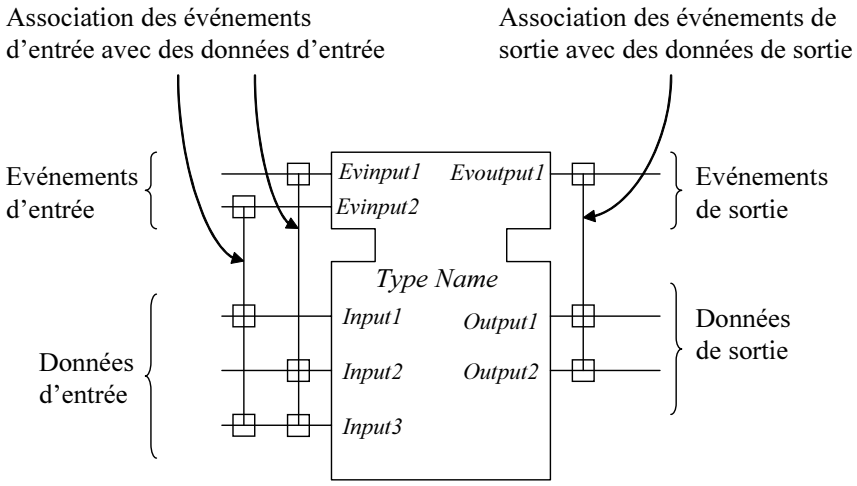



FIG. 1.6 – Un bloc fonctionnel de la norme CEI 61499

La norme CEI 61499 répond à de nombreuses demandes industrielles pour la prise en compte simultanée de signaux et d'événements, mais souffre cependant de lacunes importantes en termes de définition formelle de leur comportement. Pour répondre à ce besoin de formalisation, plusieurs travaux académiques ont proposé des représentations formelles des blocs fonctionnels afin de permettre : leur traduction dans des langages industriels existants [FRV06], la formalisation par automates temporisés [SG04], leur vérification par model-checking [VH99, FLS02], une conception générique de blocs fonctionnels [PF05], le diagnostic de fautes [KJH06], ...

Dans la suite de ce mémoire, nous allons utiliser le graphisme de ces blocs fonctionnels pour représenter les modules d'un Système Dynamique Hybride (SDH). En effet, la séparation entre l'échange de données et le flux d'événements des blocs est très pertinente pour la modélisation et la compréhension du système global. Ainsi, nous présenterons dans le chapitre 2 une approche de modélisation dans laquelle un module est uniquement une interface graphique et ne définit ni le comportement interne de chaque module, ni les relations de communication entre eux. Pour cela, il sera nécessaire de les modéliser à l'aide d'un formalisme permettant d'exprimer le comportement hybride et les interactions des composants du système.

### 1.3.2 Paradigmes de modélisation et mécanismes d'interaction

Comme on vient de le préciser, une démarche de modélisation modulaire conduit à structurer les modèles en plusieurs modules qui interagissent entre eux. Le point de départ consiste donc en l'élaboration du modèle de chaque module. Étant donné que nous nous sommes intéressés à la modélisation des systèmes hybrides et à leurs interactions, deux paradigmes de modélisation nous semblent appropriés : les « HA, Hybrid Automata » introduits par [ACHH93] et les « HIOA, Hybrid Input/Output Automata » développés par [LSV03].

Regardons plus en détail la façon dont la coordination entre automates est réalisée. Pour les automates hybrides (HA), cette coordination est réalisée par synchronisation d'étiquettes de transition. Avec cette approche, les variables sont partagées par tous

les automates et les franchissements de transitions peuvent être synchronisés entre plusieurs automates ayant des étiquettes en commun. Pour les automates hybrides à entrées/sorties (HIOA), la coordination inter-automates est faite par l'échange des variables et l'influence d'un automate sur un autre, par l'observation de ses variables. Du point de vue de la théorie des automates à entrées/sorties [LT87], l'évolution de l'environnement (processus) est indépendante du contrôleur, considérant ainsi comme asynchrone l'interaction des automates. En effet, dans un système réel, le système de commande n'empêche pas l'évolution du processus physique et vice-versa.

Des propositions visant à utiliser un langage de modélisation basé sur les HIOA pour faire de la vérification automatique sont présentées dans [Fre05a]. Dans son travail de thèse, l'auteur présente un nouvel outil de vérification, le model-checker PHAVer [Fre05b] pour lequel une classe d'automates alliant la coordination par échange de variables et par synchronisation d'étiquettes a été proposée. Cette classe d'automate conserve le concept de modularité pour les variables : d'entrée, de sortie ou locale.

Parallèlement, dans la littérature des SED, la synchronisation entre automates a fait l'objet de plusieurs travaux. Parmi ces travaux, le formalisme appelé « NCES : Net Conditions Events Systems » introduit par [RH95] propose des mécanismes très élaborés pour représenter la communication et la synchronisation entre automates : synchronisation bloquante et non bloquante entre un ou plusieurs automates. Ces mécanismes sont inspirés des travaux autour des « CSP : Communicating Sequential Processes » [Hoa85] et des « CCS : Communication and Concurrency Systems » [Mil89]. Il nous semble intéressant d'adapter ces mécanismes à la modélisation des SDH.

En conclusion, bien que la modélisation par automates hybrides permette une description du comportement des procédés physiques, puisque l'on peut exprimer l'évolution des grandeurs continues et des changements d'états de grandeurs discrètes, elle n'est pas suffisante pour décrire l'interaction des modèles dans un réseau.

Les deux principaux mécanismes d'interaction inter-automates sont l'échange des variables et la synchronisation par étiquettes de transition. Dans la pratique (la plupart des applications industrielles), cette interaction est souvent considérée de manière directionnelle, c'est-à-dire que l'on considère qu'un module agit sur d'autres modules qui réagissent seulement aux signaux envoyés par le module *maître*.

Par ailleurs, dans le cadre des systèmes à événements discrets (SED), des mécanismes performants de synchronisation à caractère bloquant ou non bloquant ont été étudiés.

De manière à contribuer à la modélisation modulaire des systèmes dynamiques hybrides (SDH), nous proposons dans le chapitre 2 une nouvelle classe d'automates hybrides à entrées/sorties et synchronisations typées ($HIOA_{TS}$) qui prolonge la sémantique des automates hybrides définis dans [Fre05a] grâce à des mécanismes évolués de synchronisation, inspirés de ceux définis pour les SED dans les NCES [RH95].

## 1.4 Apports de mon travail

Notre étude porte sur la vérification de systèmes qui utilisent les contrôleurs logiques pour commander des processus continus. L'objectif est la validation des propriétés

quantitatives, au-delà du temps, qui visent à l'amélioration de la qualité du service rendu par le système en utilisant la technique du « model-checking » hybride.

Comme le montre la figure 1.7, l'interaction du processus avec le contrôleur caractérisé par le cycle de son moniteur et ses programmes applicatifs nous permet de vérifier ces types de propriétés.

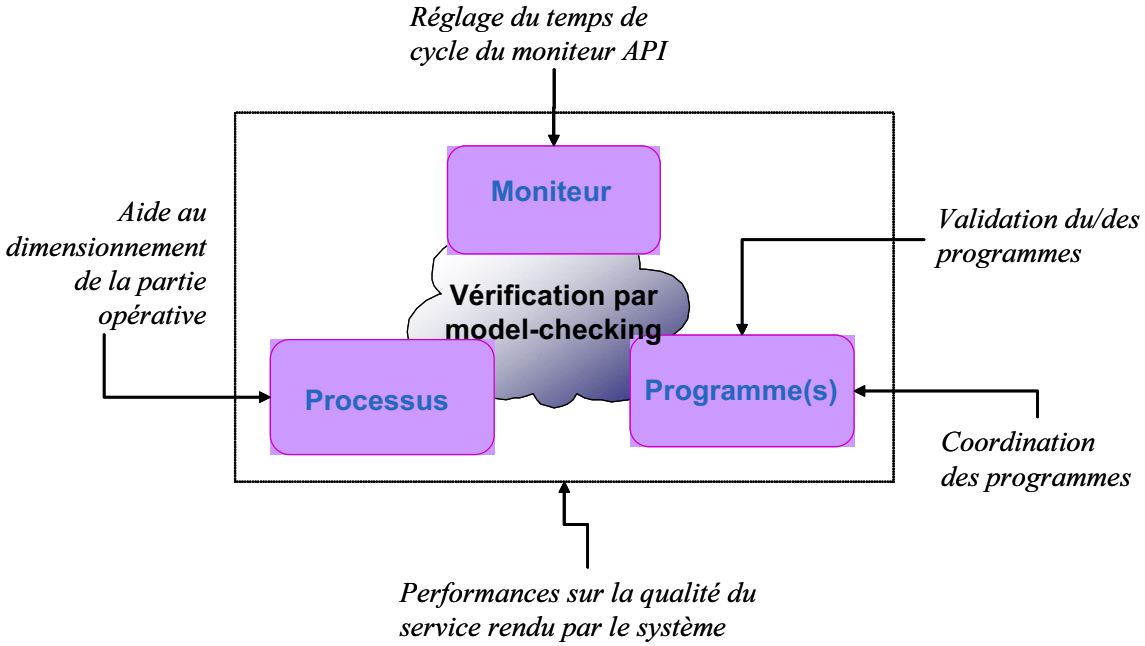



FIG. 1.7 – Vérification de la qualité du service rendu par le système

Pour aboutir à la vérification de ces propriétés, nous avons identifié différentes étapes de modélisation (voir figure 1.8). C'est autour de ces étapes que portent nos apports scientifiques pour constituer une méthodologie pour la vérification de systèmes hybrides.

Les étapes de modélisation sont les suivantes :

1. Définir la frontière de modélisation des systèmes réels.
2. Identifier les besoins de modélisation de ces systèmes.
3. Faire une abstraction des systèmes par modules en prenant en compte leur taille et les objectifs de vérification.
4. Modéliser chaque module avec le formalisme de modélisation Automates Hybrides à Entrées/Sorties que nous proposons et que l'on va appellera par la suite $HIOA_{TS}$.
5. Associer chaque bloc fonctionnel issu de la norme CEI 61499 à un module.
6. Traduire ces modèles dans le langage adapté au model-checker.
7. Chercher les propriétés quantitatives dans la région atteignable par le modèle.

L'apport de ce travail est articulé en trois axes complémentaires :

- ***l'analyse de systèmes de taille réaliste***
- ***la vérification de propriétés quantitatives***
- ***la modélisation modulaire***

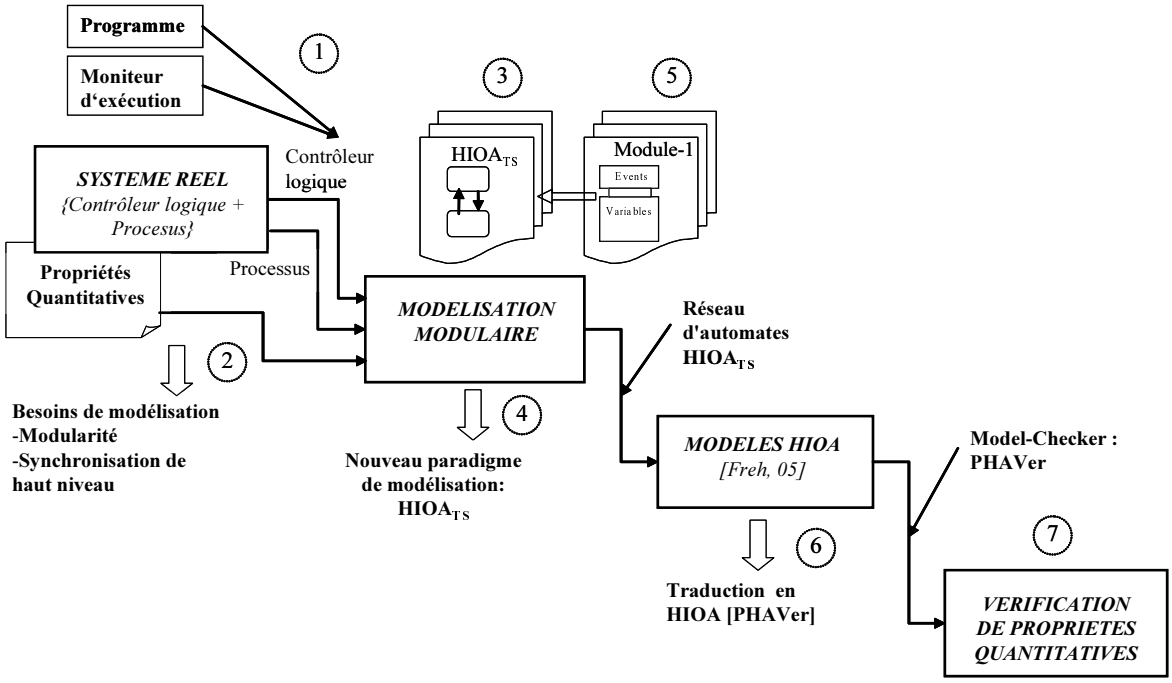



FIG. 1.8 – Identification des étapes de modélisation pour la vérification formelle

### 1.4.1 Analyse de systèmes de taille réaliste

La complexité d'un système réel, d'après [Zay01], peut être de plusieurs natures, à savoir :

- une complexité liée au grand nombre de grandeurs intervenant dans le système,
- une complexité fonctionnelle, due à l'imbrication de fonctions différentes,
- une complexité due à la variabilité extrême des configurations (modes de fonctionnement),
- une complexité due à l'incertitude de certains comportements (cas où l'aspect stochastique est important, systèmes non linéaires).

Nous nous attachons à traiter essentiellement le cas de la complexité fonctionnelle, et en particulier l'imbrication des signaux continus et discrets. Nous considérerons que les données de notre problème de vérification comprendront :

**La description du procédé physique,** des composants et des sous-processus. On déduit alors les équations de comportement, les variables de mesure, les variables de commande et les différents états du système.

**La description de la commande** dans son comportement opérationnel. Quant au contrôleur logique, nous prenons en compte non seulement la partie touchant au langage de programmation mais aussi au cycle du moniteur d'exécution. Une modélisation séparée de ces deux aspects nous permet de faire une différence claire entre les moniteurs d'exécution qui gèrent du temps et ceux qui gèrent des événements.

**La description des objectifs externes** en termes de critères qualitatifs et quantitatifs, c'est-à-dire ce que l'on attend du système.

L'objectif que l'on se fixe influe naturellement sur la conception de la méthode. Il peut s'agir de paramétrer les machines de production, de concevoir un algorithme de commande, ou encore de définir des stratégies de sécurité, ou d'améliorer la qualité de service du système.

### 1.4.2 Proposition d'une méthodologie pour la vérification

Notre méthodologie est la suivante (voir figure 1.9) :

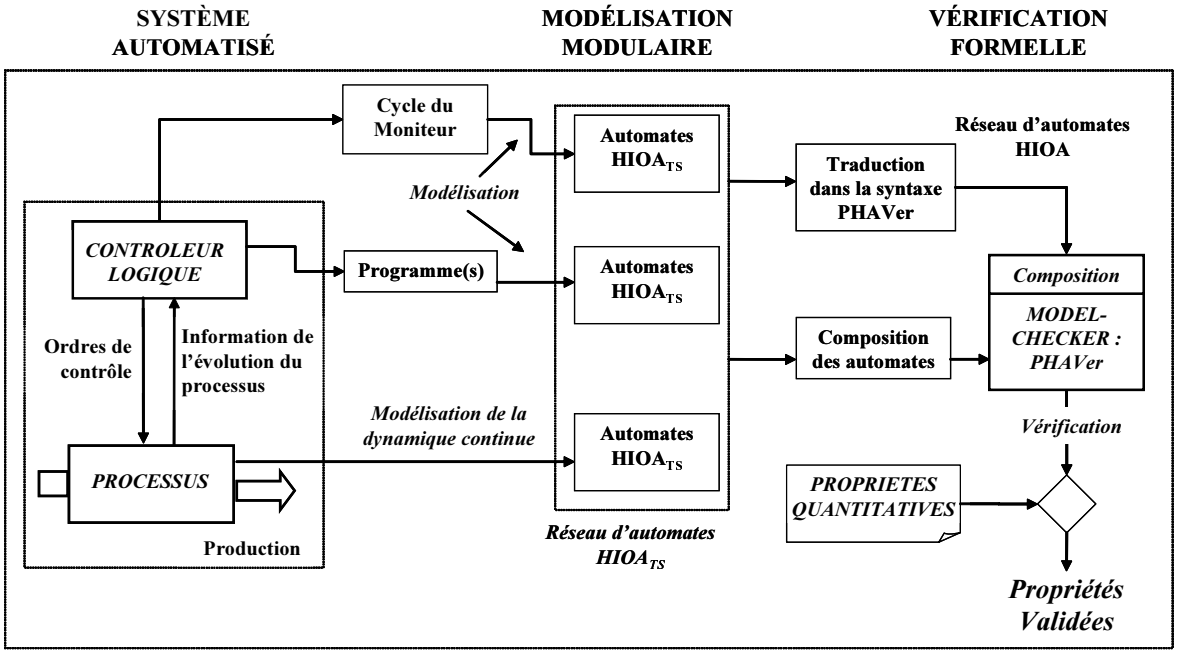



FIG. 1.9 – Méthodologie proposée pour la vérification

1. La première étape est de concevoir le contrôleur à partir d'un ensemble de spécifications fonctionnelles et de la compréhension du système physique. Étant donné le choix des Automates Programmables Industriels pour l'implémentation du contrôleur logique, nous allons prendre en compte pour la modélisation le moniteur d'exécution et le langage de programmation. Parmi les langages de programmation pour les automates programmables décrits dans la norme CEI 61131-3, le langage Sequential Function Charts (SFC) a été choisi[1]. Le comportement séquentiel ainsi que les exécutions alternatives ou parallèles sont quelques-unes de ses caractéristiques. Ce modèle du contrôleur ainsi que celui du moniteur d'exécution sont exprimés dans la syntaxe des automates hybrides à E/S que nous proposons ($HIOA_{TS}$), dont le comportement est représenté par des variables discrètes et des horloges. Nous proposons une procédure de traduction systématique du programme SFC dans ces automates.
2. La modélisation du processus est réalisée par des automates hybrides à E/S $HIOA_{TS}$.

[1]La démarche proposée est facilement transposable à d'autres langages normalisés, ce qui ne pose aucune difficulté scientifique.

3. Le réseau d'automates $HIOA_{TS}$ est construit en utilisant l'approche modulaire inspirée de la norme CEI 61499. Chaque module qui représente un composant du système est associé à un automate $HIOA_{TS}$. Les modules sont connectés par des variables partagées et des événements orientés.
4. Pour la composition des automates, il faut traduire les modules dans un langage adapté au model-checker PHAVer. Pour ce faire, nous proposons une procédure de traduction systématique.
5. Les modules du contrôleur et du processus peuvent également être composés directement par le model-checker hybride PHAVer pour permettre la vérification des propriétés. Si le modèle composé satisfait les performances attendues du système, alors la stratégie de contrôle sera validée.

## 1.5 Conclusion du chapitre 1

Dans ce chapitre, nous avons mis en évidence que l'utilisation des contrôleurs logiques pour la commande de processus continus est très répandue, soit parceque ce choix suffit pour assurer les spécifications requises, soit parce qu'il est moins coûteux. Ce choix implique certaines difficultés lorsque l'on s'intéresse à la vérification de propriétés relatives à la qualité du service rendu par le système automatisé, car il faut restituer l'aspect continu au système commandé. La nécessité d'un paradigme de modélisation permettant d'exprimer le caractère hybride du système ainsi que la description modulaire s'impose très vite, dès que la taille du processus à modéliser est significative. Cependant, dès que la taille du modèle du processus augmente, et en particulier celle de son espace d'états atteignables, la praticabilité des techniques utilisées s'amenuise.

Dans les travaux que nous avons étudiés, un aspect qui est souvent éludé est la nécessité de méthodologies générales qui permettent l'aboutissement de l'analyse. Pour augmenter la confiance que nous pourrons accorder aux résultats de vérification, une technique systématique de construction du modèle de processus est donc nécessaire.

Le chapitre 2 va maintenant être consacré à la définition formelle d'une classe d'automates hybrides que nous avons voulu adapter à la modélisation modulaire : les Automates Hybrides à Entrées/Sorties et Synchronisations Typées $HIOA_{TS}$ (« Hybrid Input Output Automata with Typed Synchronisations »).

# *Chapitre 2*

# *Formalisme pour la modélisation modulaire des systèmes hybrides*

---

*Ce chapitre présente une classe d'automates hybrides à haut niveau d'expressivité pour faciliter la modélisation modulaire des Systèmes Dynamiques Hybrides (SDH) : les automates hybrides à entrées/sorties et synchronisations typées. Elle combine le concept de modularité issu des automates à entrées/sorties [LT87] avec plusieurs mécanismes de synchronisation par étiquettes de transition. Afin d'exploiter un tel modèle avec les outils existants, nous proposons (i) un opérateur de composition pour fournir un automate « mis à plat », et (ii) un algorithme de traduction des mécanismes de synchronisations typées en synchronisations par rendez-vous classiques. L'intérêt de notre classe d'automates hybrides pour la modélisation modulaire est mis en évidence sur un exemple de SDH qui nous permet également d'illustrer l'opérateur de composition et l'algorithme de traduction des mécanismes de synchronisations typées. Nous présentons enfin l'instrumentation logicielle qui nous a servi de support pour valider les algorithmes développés dans ce chapitre ainsi que pour la mise au point des modèles des deux études de cas qui seront développées dans le chapitre 4.*

## 2.1 Introduction

L'interaction de dynamiques continues et discrètes fait de la modélisation des Systèmes Dynamiques Hybrides (SDH) un problème difficile. L'introduction des automates hybrides [ACHH93] a largement contribué à donner un cadre rigoureux à l'expression de cette interaction entre dynamique continue et dynamique discrète. Pour les systèmes complexes, une approche modulaire est par ailleurs indispensable à la modélisation. Dans ce cas, un module décrit le comportement d'un sous-système de taille réduite, ce qui le rend plus propice à la modélisation par un automate hybride. La difficulté de modélisation est alors pour une large part reportée sur la définition des mécanismes de coordination entre les modules.

Les deux principales approches pour représenter ces coordinations sont la synchronisation par les étiquettes de transition [HHWT97] et la coordination par échange de variables [LSV03]. Avec l'approche par synchronisation avec étiquettes de transition, les variables sont partagées par tous les automates (pas d'encapsulation dans les modules) et les franchissements de transitions peuvent être synchronisés entre plusieurs automates (notion d'événement). L'approche par coordination par échange de variables laisse les automates évoluer de manière asynchrone, l'influence d'un automate sur un autre se faisant par observation de ces variables. Cette coordination est caractéristique des automates hybrides à entrées/sorties [LSV03] ($HIOA$ : Hybrid Input Output Automata) et elle renforce le concept de modularité en typant les variables : d'entrée, de sortie ou locale. Il existe une classe d'automates proposée par G. Frehse alliant des synchronisations par les étiquettes et la modularité [Fre05a], mais elle a été développée spécifiquement pour la vérification formelle par « model-checking » et non pas plus globalement pour la modélisation des SDH. Les contraintes propres à l'élaboration d'un « model-checker » ont ainsi conduit l'auteur à limiter le pouvoir d'expression des synchronisations entre automates, pour en faciliter la vérification.

Parallèlement aux travaux sur les SDH, la synchronisation entre automates a fait l'objet de nombreux développements dans le domaine des Systèmes à Événements Discrets (SED). Depuis les travaux fondateurs autour des Communicating Sequential Processes [Hoa85] ou de l'algèbre de processus CCS [Mil89], des mécanismes de synchronisation à caractère bloquant ou non bloquant, entre deux automates ou plus, ont été étudiés. Des formalismes tels que les « Net Condition Event Systems » (NCES) [RH95] proposent ainsi au modélisateur des mécanismes multiples et évolués pour représenter les communications et les synchronisations entres automates.

C'est dans ce contexte que nous proposons une nouvelle classe d'automates hybrides à entrées/sorties et synchronisations typées ($HIOA_{TS}$ : Hybrid Input Ouput Automata with Typed Synchronisations) qui prolonge les capacités sémantiques des automates hybrides linéaires définis dans [Fre05a] grâce à des mécanismes évolués de synchronisation, inspirés de ceux définis pour les SED dans les NCES.

La modélisation d'un SDH n'est par ailleurs pas une fin en soi. L'évaluation de performances, l'analyse formelle, la recherche de régions atteignables ou le diagnostic sont parmi les activités qui motivent généralement l'élaboration du modèle. Pour rendre la modélisation par $HIOA_{TS}$ compatible avec ces différentes utilisations, nous proposons deux voies : (i) la composition parallèle d'automates $HIOA_{TS}$ qui génère un unique automate hybride composé exempt de synchronisations, (ii) la traduction d'un

réseau d'$HIOA_{TS}$ en un réseau d'automates qui se coordonnent par synchronisations classiques (rendez-vous bloquant).

Ce chapitre est organisé de la manière suivante. La section 2.2 est consacrée à la définition des automates hybrides à entrées/sorties à synchronisations typées. Un exemple est utilisé pour illustrer la nécessité de définir différents types de synchronisations entre automates. Le comportement global d'un SDH modélisé par une approche modulaire étant donné par l'interaction d'automates, un opérateur de composition est également défini. La section 2.3 présente un algorithme de traduction pour exprimer le comportement des automates hybrides à entrées/sorties à synchronisations typées avec la sémantique d'un outil d'analyse existant. Enfin, l'exemple présenté en section 2.2 est traduit puis analysé avec le « model-checker » polyédrique PHAVer.

## 2.2 Réseaux d'Automates Hybrides Linéaires à Synchronisations Typées

### 2.2.1 Les automates hybrides linéaires à entrées/sorties

La classe d'automates que nous proposons est une extension des automates hybrides linéaires à entrées/sorties ($HIOA$) retenus par G. Frehse dans ses travaux sur le « model-checker » PHAVer [Fre05a]. Un exemple d'$HIOA$ est donné sur la fig 2.1, nous allons en rappeler la définition formelle.

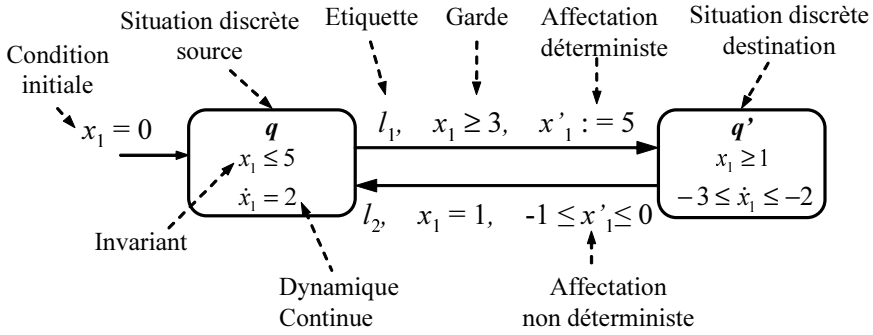



FIG. 2.1 – Syntaxe d'un automate hybride linéaire $HIOA$

Un automate hybride linéaire à entrées/sorties $HIOA = \langle \mathbb{Q}, \mathbb{X}, \mathbb{L}, \mathbb{T}, \mathcal{A}, \mathcal{F}, \mathcal{I}, Init \rangle$ consiste en :

- un ensemble de situations discrètes $\mathbb{Q}$ ;
- un ensemble de variables réelles $\mathbb{X} = \mathbb{X}_I \cup \mathbb{X}_O \cup \mathbb{X}_L$, où $\mathbb{X}_I$ est l'ensemble des variables d'entrée, $\mathbb{X}_O$ l'ensemble des variables de sortie et $\mathbb{X}_L$ l'ensemble des variables locales. De plus, $\mathbb{X}_I \cap \mathbb{X}_O = \mathbb{X}_I \cap \mathbb{X}_L = \mathbb{X}_O \cap \mathbb{X}_L = \varnothing$. Les variables appartenant à $\mathbb{X}_L$ ou à $\mathbb{X}_O$ sont les variables *contrôlées* par l'automate, tandis que les variables appartenant à $\mathbb{X}_I$ sont des variables *non contrôlées* ;
- un ensemble d'étiquettes $\mathbb{L} = \mathbb{L}_I \cup \mathbb{L}_O \cup \mathbb{L}_L$ où $\mathbb{L}_I$ représente l'ensemble des étiquettes d'entrée, $\mathbb{L}_O$ l'ensemble des étiquettes de sortie et $\mathbb{L}_L$ l'ensemble des étiquettes locales ;

- un ensemble de transitions $\mathbb{T} \subseteq \mathbb{Q} \times \mathbb{L} \times \mathbb{G} \times \mathbb{Q}$, où si $(q, l, g, q') \in \mathbb{T}$ alors :
  - $q$ et $q$' sont les situations discrètes source et destination, éléments de $\mathbb{Q}$,
  - $l$ est une étiquette, élément de $\mathbb{L}$ ($l \in \mathbb{L}_L$ si aucune étiquette d'entrée ou de sortie n'est requise),
  - $g$ est une garde, élément de $\mathbb{G}$, c'est-à-dire une condition sur la valeur des variables de $\mathbb{X}$ exprimée sous la forme d'une conjonction de prédicats éléments de $\mathbb{P}$. $\mathbb{P}$ est l'ensemble des prédicats définis sur $\mathbb{X}$. Un prédicat est une égalité ou une inégalité entre expressions affines des variables de $\mathbb{X}$ à coefficients constants et rationnels ;
- une fonction $\mathcal{A}$ qui associe à une transition $(q, l, g, q')$ une liste d'affectations des variables contrôlées $\mathbb{X}_O$ et $\mathbb{X}_L$. Chaque affectation est spécifiée par une conjonction de prédicats de $\mathbb{P}$ ;
- une fonction $\mathcal{F}$ qui décrit la dynamique continue de chaque situation discrète de $\mathbb{Q}$, en associant à chaque couple $(q, x_C) \in \mathbb{Q} \times (\mathbb{X}_O \cup \mathbb{X}_L)$ une conjonction de prédicats de la forme $\dot{x_C} \geq c(\mathbb{X})$, $\dot{x_C} \leq c(\mathbb{X})$ ou $\dot{x_C} = c(\mathbb{X})$, avec $c(\mathbb{X})$ une expression affine des variables de $\mathbb{X}$ à coefficients constants et rationnels ;
- une fonction $\mathcal{I}$ qui associe un invariant à chaque situation discrète de $\mathbb{Q}$. Cet invariant est une région de l'espace d'états des variables de $\mathbb{X}$ défini sous la forme d'une disjonction de conjonctions de $\mathbb{P}$ ;
- un état initial *Init* défini par une situation initiale $q_0 \in \mathbb{Q}$ et un domaine de l'espace d'états des variables de $\mathbb{X}_O \cup \mathbb{X}_L$ défini sous la forme d'une conjonction de prédicats de $\mathbb{P}$.

L'état courant d'un automate hybride linéaire est défini par un couple $(q, v) \in \mathbb{Q} \times \mathbb{R}^{\text{card}(\mathbb{X}_O \cup \mathbb{X}_L)}$ où $v$ est une valuation sur l'ensemble de réels des variables contrôlées. L'état d'un *HIOA* peut donc être modifié de deux manières :

(i) par franchissement d'une transition $(q, l, g, q') \in \mathbb{T}$ qui change la situation discrète courante $q$ en $q'$, et qui fait évoluer les variables contrôlées de $\mathbb{X}$ selon la fonction $\mathcal{A}$. Ce franchissement n'est possible que lorsque la garde l'autorise,

(ii) par l'écoulement du temps dans une situation qui fait évoluer la valeur des variables contrôlées selon la fonction $\mathcal{F}$. Un *HIOA* peut demeurer dans une situation $q$ tant que l'invariant $\mathcal{I}(q)$ est satisfait.

Pour la modélisation des systèmes complexes, une approche modulaire est souvent indispensable. Elle conduit à la construction d'un ensemble d'*HIOA* en interaction appelé *Réseau* d'*HIOA*, défini formellement comme :

$$R = \{i \in [\![1, n]\!] \mid HIOA_i = \langle \mathbb{Q}_i, \mathbb{X}_i, \mathbb{L}_i, \mathbb{T}_i, \mathcal{A}_i, \mathcal{F}_i, \mathcal{I}_i, Init_i \rangle\}$$

$\forall (i, j) \in [\![1, n]\!]^2$ avec $i \neq j$ : les ensembles des situations sont disjoints deux à deux $\mathbb{Q}_i \cap \mathbb{Q}_j = \varnothing$, les variables locales n'appartiennent à aucun autre automate $\mathbb{X}_{L_i} \cap \mathbb{X}_j = \varnothing$, les variables de sortie ne peuvent pas être des variables contrôlées dans un autre automate $\mathbb{X}_{O_i} \cap (\mathbb{X}_{O_j} \cup \mathbb{X}_{L_j}) = \varnothing$ qui se simplifie en $\mathbb{X}_{O_j} \cap \mathbb{X}_{O_i} = \varnothing$ compte tenu des contraintes précédentes. Ainsi, $\mathbb{X}_{O_i} \subset \mathbb{X}_i$ et $\mathbb{X}_i \cap \mathbb{X}_{L_j} = \varnothing$.

**Notation.** Des notations différentes pourront être utilisées dans ce document pour les *HIOA* selon qu'il s'agira d'une figure générée par une toolbox, d'un code d'entrée pour le « model-checker » ou d'une simple illustration graphique. La table 2.1 montre les notations utilisées tout au long de cette thèse pour les *HIOA*.

| contexte | signification | notations | | |
|---|---|---|---|---|
| invariant | x est inférieure ou égale à 3 | $x \leq 3$ | $x <= 3$ | |
| dynamique continue | la dérivée de x est égale à 2 | $\dot{x} = 2$ | $d(x)/dt = 2$ | $x' = 2$ |
| | la dérivée de x est dans l'intervalle fermé [-3,-2] | $-3 \leq x \leq -2$ | $-3 \leq d(x)/dt \leq -2$ | -3<= $x'$ & $x'$ <=-2 |
| garde | x est supérieure à 3 et différente de 5 | $x \geq 3 \wedge \neg(x = 5)$ | $x >= 3 \wedge \neg(x == 5)$ | $x$ >=3 & ! ($x$==5) |
| affectation | 5 est affecté à x | $x' := 5$ | $x' \leftarrow 5$ | $x' == 5$ |
| | la nouvelle valeur de x est plus petite ou égale à sa valeur courante moins 1 | $x' \leq x - 1$ | $x' <= x - 1$ | |

TAB. 2.1 – Notations pour les *HIOA*

### 2.2.2 Approche modulaire pour représenter le réseau

De manière à obtenir une vue d'ensemble d'un réseau d'automates permettant de décrire la connexion de différents *HIOA* et leurs interactions dans un réseau nous avons retenu une représentation graphique, inspirée des « Function Blocs » de la norme CEI 61499 [IEC04]. Selon cette norme, un bloc fonctionnel est une abstraction de la fonction d'un composant (physique ou de contrôle) dont l'interface comporte des événements d'entrée et de sortie et des données d'entrée et de sortie. Ainsi, une application globale est construite par la connexion de plusieurs de ces blocs. En effet, c'est cette caractéristique structurelle de la norme que nous avons prise comme modèle pour appuyer les représentations graphiques de nos automates.

L'idée est de définir un schéma modulaire pour qu'un automate hybride à entrées/sorties puisse être représenté par un bloc (module) et que le modèle global d'une application puisse être construit en composant ces modules de base. Cette représentation graphique nous permet de décrire la structure d'un *HIOA* et de ses interactions avec d'autres *HIOA* via l'échange de données ou/et la synchronisation d'événements. La figure 2.2 montre la représentation graphique résultante d'un *HIOA*.

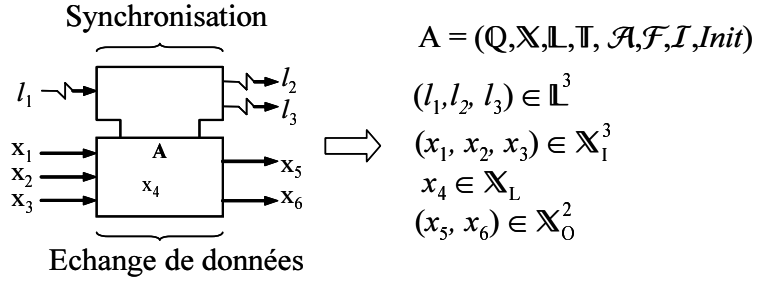



FIG. 2.2 – Représentation graphique modulaire d'un *HIOA*

Les données ainsi que les événements sont porteurs de différents types de signaux

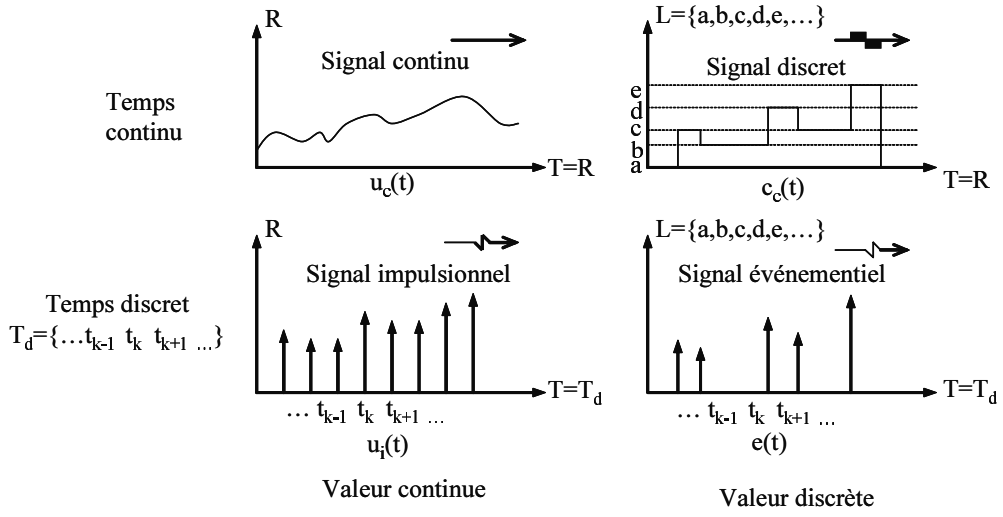


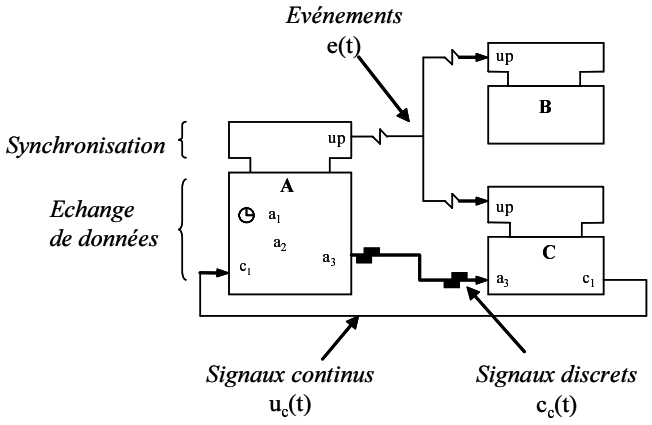

FIG. 2.3 – Classification des types de signaux présentée dans [TF00]

(voir fig 2.3). De manière à décrire les relations externes et internes de chaque module, ces signaux ont été définis par [TF00] comme :

- Signaux à valeurs continues, définis pour tout instant dans $\mathbb{R}$, que l'on appelle signaux continus, ou définis seulement pour certains instants, appelés signaux impulsionnels ;
- Signaux à valeurs discrètes, définis pour un temps variant dans $\mathbb{R}$, que l'on appelle signaux discrets, ou définis seulement pour certains instants, appelés signaux événementiels. Précisons que les signaux discrets prenant leur valeur dans {0,1} sont appelés signaux logiques.

Dans cette classification, les signaux à temps continu correspondent aux échanges de variables dans un réseau d'$HIOA_{TS}$. Nous allons donc en retenir le symbolisme. Par contre, les signaux à temps discret, adaptés aux systèmes échantillonnés, ne correspondent pas aux échanges d'événements dans un réseau d'$HIOA_{TS}$, ceux-ci n'étant porteurs d'aucune valeur d'un signal. Nous allons cependant retenir le symbolisme, qui évoque le caractère événementiel des échanges. Ainsi, en s'inspirant de cette logique des signaux définis dans [TF00], un réseau d'$HIOA$ aura l'allure montrée dans la figure 2.4. Nous avons un réseau composé de trois automates entre lesquels les échanges d'informations sont représentés de la manière suivante :

- $\multimap$ désigne les variables porteuses des signaux discrets et/ou logiques ;
- $\longrightarrow$ désigne les variables porteuses des signaux continus ;
- $\circlearrowleft$ indique que la variable est utilisée comme une horloge de type chronomètre (stop watch) dont l'évolution continue dans le temps est déterminée par une dérivée qui prendra sa valeur uniquement dans $\{0, 1\}$.
- $\rightsquigarrow$ désigne les étiquettes de synchronisation indicatrices des événements de franchissement de transitions (aucune valeur n'est portée sur ces événements).




$A = (\mathbb{Q}, \mathbb{X}, \mathbb{L}, \mathbb{T}, \mathcal{A}, \mathcal{F}, \mathcal{I}, Init)$
$up \in \mathbb{L}$
$c_1 \in \mathbb{X}_I$
$a_3 \in \mathbb{X}_O$
$(a_1, a_2) \in \mathbb{X}_L^2$

$B = (\mathbb{Q}, \mathbb{X}, \mathbb{L}, \mathbb{T}, \mathcal{A}, \mathcal{F}, \mathcal{I}, Init)$
$up \in \mathbb{L}$

$C = (\mathbb{Q}, \mathbb{X}, \mathbb{L}, \mathbb{T}, \mathcal{A}, \mathcal{F}, \mathcal{I}, Init)$
$up \in \mathbb{L}$
$a_3 \in \mathbb{X}_I$
$c_1 \in \mathbb{X}_O$

FIG. 2.4 – Représentation des *HIOA* et de leurs interactions dans un réseau

### 2.2.3 Besoins du modélisateur en synchronisations typées

Lorsqu'il effectue une modélisation d'un système complexe, le concepteur se heurte souvent à des difficultés pour exprimer la coordination entre les automates de modules. Le besoin d'étendre les capacités sémantiques des automates hybrides linéaires pour offrir de multiples capacités d'exprimer différents types de synchronisation entre les automates est bien réel. Pour définir le comportement d'un système modélisé par un réseau d'automates, il est nécessaire de définir comment les étiquettes associées aux transitions permettent la synchronisation des évolutions entre automates. Dans la littérature, on trouve de manière plus générale différents types de synchronisation que l'on peut classer selon :

- deux structures de synchronisation : les synchronisations biparties (1 automate émetteur, 1 automate récepteur) (CCS [Mil89], UPPAAL [KY06]) et les synchronisations multiparties (1 automate émetteur, N automates récepteurs) (CSP [Hoa85], LOTOS [ISO89]) ;
- quatre types de synchronisation :
  (i) l'interaction asynchrone, où les étiquettes ne sont pas utiles et où les interactions entre automates se font sur échanges de variables réelles, comme dans les automates à entrées/sorties [LT87] ;
  (ii) la synchronisation par rendez-vous non bloquant, où un automate émetteur émet une étiquette de synchronisation sans être bloqué par le comportement des récepteurs, alors que les récepteurs sont bloqués en attente de l'émission d'une étiquette de synchronisation. Ce type de synchronisation est par exemple utilisé dans les automates UPPAAL [LPY97] ;
  (iii) la synchronisation par rendez-vous bloquant où tous les automates qui possèdent la même étiquette $l$ doivent s'attendre pour franchir simultanément leur transition étiquetée $l$. Par contre, ils évoluent de manière asynchrone pour les transitions sans étiquette partagée. Ce type de synchronisation très répandu est décrit dans [CL99] et utilisé par exemple dans les automates hybrides de HYTECH [HHWT97] et dans les *HIOA* de PHAVer [Fre05a] ;
  (iv) l'interaction totalement synchrone où tous les automates du réseau n'évo-

luent que par franchissement des transitions possédant une étiquette partagée par tous. Aucune des autres transitions ne pourra être franchie [CL99].

Nous proposons une classe d'automates qui offre trois types de synchronisation (l'interaction asynchrone, la synchronisation par rendez-vous non bloquant et la synchronisation par rendez-vous bloquant) compatibles avec une structure multipartie (1 émetteur, N récepteurs). Par contre, la synchronisation totalement synchrone ne peut pas cohabiter avec les trois autres car elle bannit tout asynchronisme même partiel. Pour illustrer l'intérêt de cette classe, nous allons traiter un exemple volontairement simple mais illustratif des besoins de modélisation en termes de communication inter-automates. Il s'agit de deux véhicules circulant sur deux voies se croisant à un carrefour protégé par deux feux de signalisation (fig 2.5).

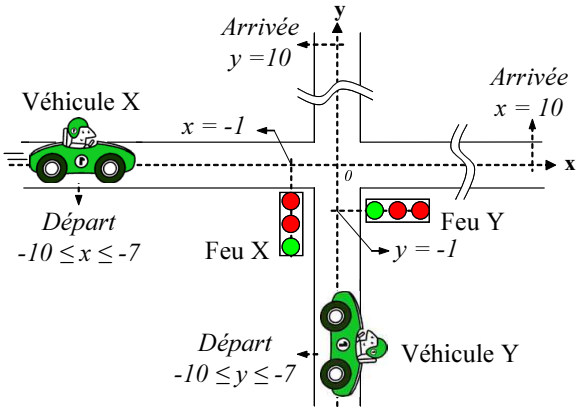



FIG. 2.5 – Exemple : deux véhicules se croisant à un carrefour

Le modèle de ce système peut être obtenu à l'aide de quatre $HIOA$, un par véhicule et un par feu. Une vue globale du modèle ainsi obtenu, ne décrivant que sa structuration en un réseau d'$HIOA$ et les échanges de données et de synchronisations entre $HIOA$, est donnée sur la figure 2.6.

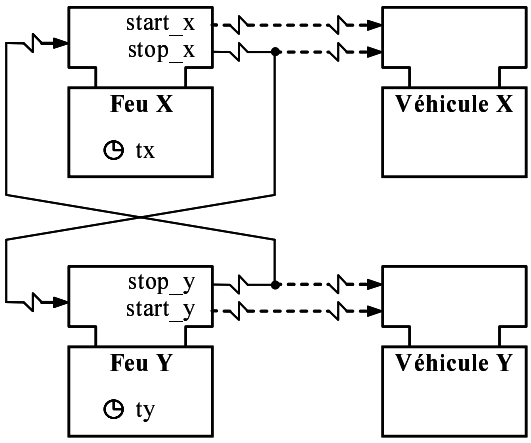



FIG. 2.6 – Représentation d'un réseau d'automates $HIOA$ modélisant le carrefour

Ces quatre automates n'échangent pas de variables continues et ils se coordonnent uniquement par synchronisation. Le Feu X émet le signal de synchronisation start_x à

destination de tout véhicule qui serait à l'arrêt sur la voie X pour signaler son passage au vert. Cette synchronisation doit être non bloquante car le feu ne doit pas attendre de rendez-vous avec un véhicule pour passer au vert. Il est donc nécessaire ici d'utiliser une synchronisation non bloquante. Il en va de même pour le signal de synchronisation stop_x issu du Feu X. Cependant stop_x est par ailleurs un signal de synchronisation d'entrée pour Feu Y où il est légitime d'utiliser cette fois une synchronisation par rendez-vous bloquant pour des raisons évidentes de sécurité. Aucune désynchronisation entre les deux feux ne doit se produire. Cela fait apparaître le besoin d'une *synchronisation mixte* pour stop_x entre trois automates : un émetteur (Feu X), un récepteur non bloquant (Véhicule X) et un récepteur bloquant (Feu Y).

C'est l'identification de ce besoin de modélisation qui nous a conduits à proposer une extension des $HIOA$ : les $HIOA$ à synchronisations typées ($HIOA_{TS}$). Dans un $HIOA_{TS}$, le type de synchronisation souhaité (émetteur / récepteur bloquant / récepteur non bloquant) est associé à chaque étiquette de transition.

### 2.2.4 Les automates hybrides linéaires à entrées/sorties et à synchronisations typées $HIOA_{TS}$

Un automate hybride à entrées/sorties et à synchronisations typées ($HIOA_{TS}$) est de la forme $H_{HIOA_{TS}} = \langle H_{HIOA}, \mathcal{R} \rangle$ avec :

- $H_{HIOA}$ un automate hybride linéaire à entrées/sorties tel que défini en section 2.2.1,
- $\mathcal{R}$ une fonction qui associe un *rôle* aux étiquettes, avec $\mathcal{R} : \mathbb{L} \rightarrow \{!, ?, ¿\}$.

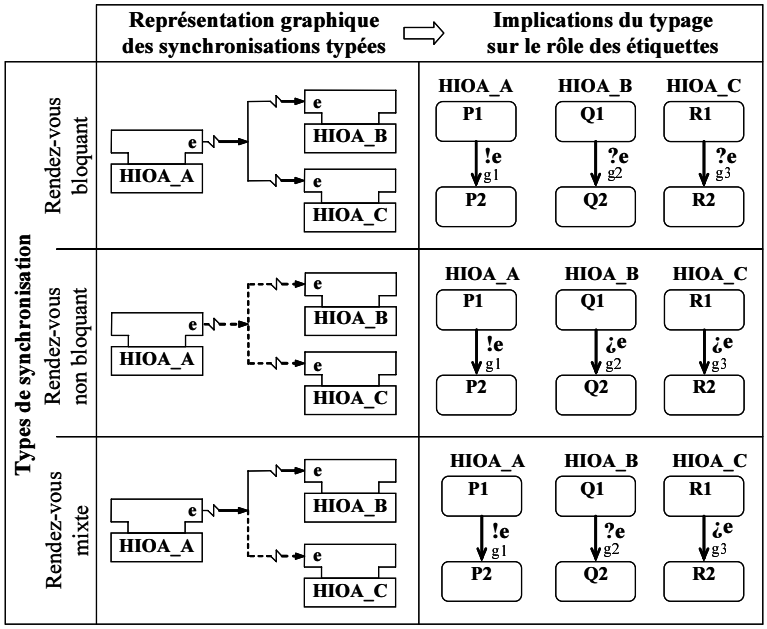
Un réseau d'$HIOA_{TS}$ est donc un ensemble d'$HIOA_{TS}$ qui communiquent entre eux, défini comme :

$$R_{HIOATS} = \{i \in [\![1, n]\!] \mid H_{HIOATS_i} = \langle H_{HIOA_i}, \mathcal{R}_i \rangle\}$$

L'association d'un rôle aux étiquettes impose deux conditions supplémentaires aux $HIOA$ d'un réseau :

- $\forall i \in [\![1, n]\!]\ \mathbb{L}_{I_i} \cap \mathbb{L}_{O_i} = \mathbb{L}_{I_i} \cap \mathbb{L}_{L_i} = \mathbb{L}_{O_i} \cap \mathbb{L}_{L_i} = \varnothing$ ;
- $\forall (i, j) \in [\![1, n]\!]^2$ avec $i \neq j$, $\mathbb{L}_{L_i} \cap \mathbb{L}_j = \varnothing$ et $\mathbb{L}_{O_i} \cap (\mathbb{L}_{O_j} \cup \mathbb{L}_{L_j}) = \varnothing$.

La multiplicité des types de synchronisation entre automates est obtenue en associant un rôle aux étiquettes : le rôle émetteur symbolisé par « ! », le rôle récepteur bloquant symbolisé par « ? » et le rôle récepteur non bloquant symbolisé par « ¿ ». La table 2.2 présente les trois types de synchronisation entre trois automates et leur implication sur le rôle des étiquettes.




TAB. 2.2 – Exemples de synchronisations typées entre trois $HIOA$

La figure 2.7 donne *in extenso* les quatre $HIOA_{TS}$ qui modélisent le problème du carrefour de la figure 2.5 où l'on peut observer le rôle donné à chaque étiquette. Par exemple, l'événement stop_x est utilisé dans trois automates avec à chaque fois un rôle différent :

- Feu X : FXV $\xrightarrow{!stop\_x,...}$ FXR1 (rôle émetteur)
- Feu Y : FYR1 $\xrightarrow{?stop\_x,...}$ FYR2 (rôle récepteur bloquant)
- Véhicule X : VXC $\xrightarrow{¿stop\_x,...}$ VXR (rôle récepteur non bloquant)

Nous avons montré jusqu'à présent qu'un réseau d'$HIOA_{TS}$ s'avère utile pour la modélisation grâce à sa richesse sémantique qui permet une haute expressivité pour la communication inter-automates. Cependant, étant donné que la modélisation n'est pas une fin en soi, c'est cette même caractéristique qui rend les $HIOA_{TS}$ mal adaptés à des utilisations ultérieures telles que la vérification formelle ou la simulation. Pour rendre compatible la modélisation par $HIOA_{TS}$ avec d'autres utilisations, nous avons considéré qu'il est nécessaire de définir la composition des automates hybrides à entrées/sorties. En effet, la définition sémantique d'un réseau d'$HIOA_{TS}$ n'est complète que lorsque l'opérateur de composition est formalisé [CC95]. Le résultat de la composition est un unique automate hybride composé, exempt de toute synchronisation, qui est plus facilement adaptable à des outils informatiques pour des applications telles que la recherche d'atteignabilité ou la simulation.

### 2.2.5 Composition des automates d'un réseau d'$HIOA_{TS}$

Nous allons présenter l'opérateur de composition d'un réseau d'$HIOA_{TS}$ sous la forme d'un algorithme (voir algorithme 1). Soient $H_1$ et $H_2$ deux automates du réseau, l'automate composé $H$ est construit de la manière suivante :

- Lignes [1 : 3] Calcul de l'ensemble des variables réelles $\mathbb{X}$ ;

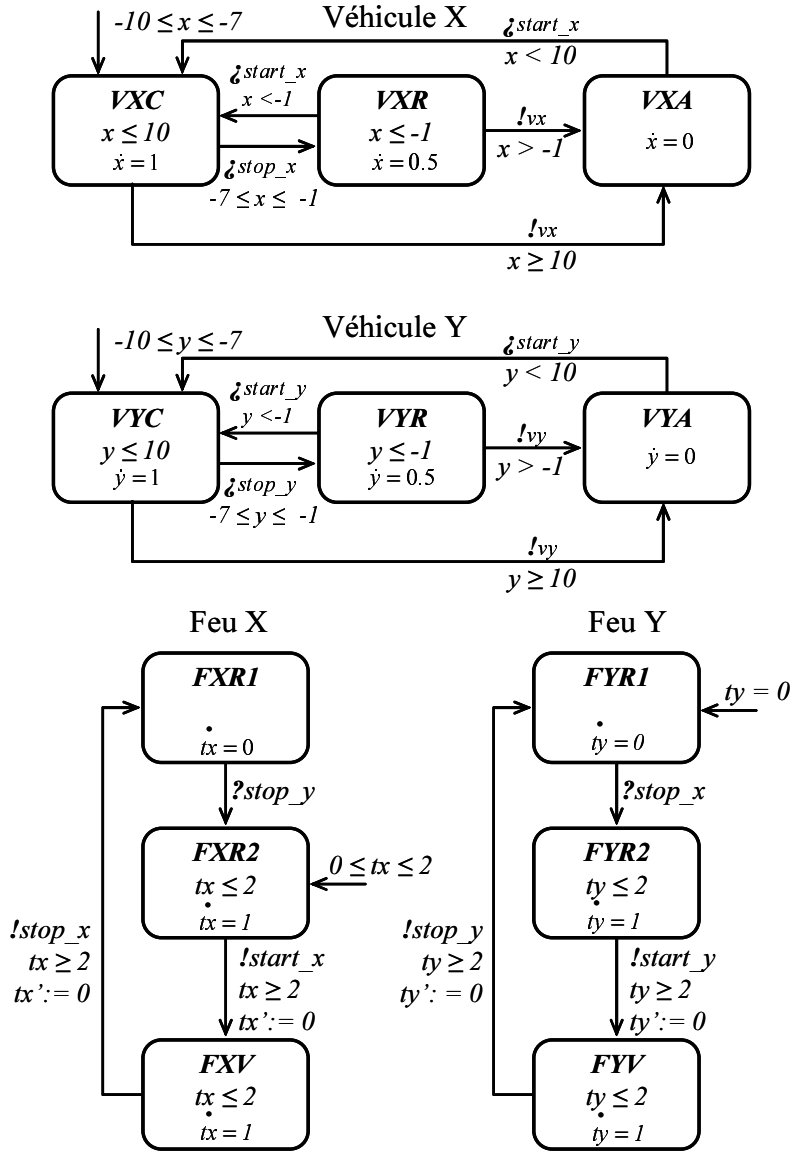



FIG. 2.7 – Modèles $HIOA_{TS}$ de l'exemple de la figure 2.5

- Lignes [4 : 6] Calcul de l'ensemble des étiquettes $\mathbb{L}$ ;
- Lignes [7 : 8] Association d'un "rôle" à l'ensemble d'étiquettes $\mathbb{L}$ ;
- Lignes [9 : 11] Initialisation de l'ensemble des situations discrètes $\mathbb{Q}$. L'algorithme commence avec le couple des situations initiales de $H_1$ et $H_2$ pour former la première situation discrète de l'automate composé, laquelle est ajoutée à l'ensemble $\mathbb{Q}$.
- Lignes [12 : 13] Construction d'une pile FIFO (« First In First Out ») de couples de situations discrètes. Le premier couple est ajouté à la pile ;
- Lignes [14 : 32] Boucle « While » : Identification des différents cas de composition. Tant que la pile n'est pas vide, l'algorithme recherche les transitions issues des couples ajoutés à la pile pour construire l'ensemble de situations $\mathbb{Q}$ et l'ensemble de transitions $\mathbb{T}$ comme suit :
  - Ligne [15] Le premier couple de situations $(q_{1s}, q_{2s})$ sort de la PILE
  - Lignes [16 : 17] Recherche des transitions issues de la situation $q_{1s}$ dans $H_1$ et de la situation $q_{2s}$ dans $H_2$
  - Lignes [18 : 26] En premier, pour toutes les transitions à partir de la situation $q_{1s}$ qui font partie des transitions de $H_1$ :
    - si l'étiquette $l_1$ ne fait pas partie de l'ensemble d'étiquettes de $H_2$, c'est que l'on est dans le **premier cas de la composition** (voir algorithme 2)
    - si, par contre, $l_1$ fait partie des étiquettes de $H_2$, deux situations sont possibles :
      - le **deuxième cas de la composition** apparaît si, pour toutes les transitions de $H_2$, les étiquettes portent le même nom (voir algorithme 2),
      - le **troisième cas de la composition** apparaît si le rôle de l'étiquette $l_1$ est ¿ (voir algorithme 2),
  - Lignes [27 : 32] En deuxième, pour toutes les transitions à partir de $q_{2s}$ qui font partie des transition de $H_2$ :
    - si l'étiquette $l_2$ ne fait pas partie de l'ensemble d'étiquettes de $H_1$, c'est que l'on est dans le **quatrième cas de la composition** (voir algorithme 2),
    - si, par contre, $l_2$ fait partie des étiquettes de $H_1$ et que le rôle de l'étiquette $l_2$ est ¿, on est dans le **cinquième cas de la composition** (voir algorithme 2),
- Ligne [33] Construction de la fonction $\mathcal{F}$ qui décrit la dynamique continue de chaque situation de l'automate composé ;
- Lignes [34 : 35] Construction de la fonction $\mathcal{I}$ qui associe une région de l'espace d'état des variables $\mathbb{X}$ à chaque situation discrète ;
- Lignes [36 : 37] Calcul de l'état initial *Init*.

**Algorithme 1** : Composition d'un réseau d'automates $HIOA_{TS}$

**Data** : $H_1$ et $H_2$, deux automates du réseau d'$HIOA_{TS}$
**Result** : $H = H_1 \| H_2$, un automate composé

$\mathbb{X}_O \leftarrow \mathbb{X}_{O1} \cup \mathbb{X}_{O2}$
$\mathbb{X}_I \leftarrow (\mathbb{X}_{I1} \cup \mathbb{X}_{I2}) - \mathbb{X}_O$
$\mathbb{X}_L \leftarrow (\mathbb{X}_{L1} \cup \mathbb{X}_{L2})$
$\mathbb{L}_O \leftarrow \mathbb{L}_{O1} \cup \mathbb{L}_{O2}$
$\mathbb{L}_I \leftarrow (\mathbb{L}_{I1} \cup \mathbb{L}_{I2}) - \mathbb{L}_O$
$\mathbb{L}_L \leftarrow (\mathbb{L}_{L1} \cup \mathbb{L}_{L2})$
$\mathcal{R} : \mathbb{L} \rightarrow \{!, ?, ¿\}$

$$l \mapsto \begin{cases} "!" & \text{si} \quad l \in \mathbb{L}_O \cup \mathbb{L}_L \\ "?" & \text{si} \quad l \in \mathbb{L}_I \text{ et } (\mathcal{R}_1(l) \neq "¿" \text{ ou } \mathcal{R}_2(l) \neq "¿") \quad \text{et} \\ "¿" & \text{si} \quad l \in \mathbb{L}_I \text{ et } (\mathcal{R}_1(l) = "¿" \text{ ou } \mathcal{R}_2(l) = "¿") \end{cases}$$

$q_{1s} \leftarrow q_{01}$ état initial de $H_1$
$q_{2s} \leftarrow q_{02}$ état initial de $H_2$
$\mathbb{Q} \leftarrow \mathbb{Q} \cup \{(q_{1s}, q_{2s})\}$
PILE $\leftarrow$ CREER PILE()
EMPILER(PILE, $(q_{1s}, q_{2s})$)
**while** *not PILE-VIDE (PILE)* **do**
    $(q_{1s}, q_{2s}) \leftarrow$ DEPILER (PILE)
    $T_1 = \{q \xrightarrow{l,g} q' \in \mathbb{T}_1 \mid q = q_{1s}\}$
    $T_2 = \{q \xrightarrow{l,g} q' \in \mathbb{T}_2 \mid q = q_{2s}\}$
    **forall** $q_{1s} \xrightarrow{l_1,g_1} q_{1d} \in T_1$ **do**
        **if** $l_1 \notin \mathbb{L}_2$ **then**
            Cas 1 de la composition
        **else**
            **forall** $q_{2s} \xrightarrow{l_2,g_2} q_{2d} \in T_2$ **do**
                **if** $l_1 = l_2$ **then**
                    Cas 2 de la composition
            **if** $\mathcal{R}_2(l_1) = "¿"$ **then**
                Cas 3 de la composition
    **forall** $q_{2s} \xrightarrow{l_2,g_2} q_{2d} \in T_2$ **do**
        **if** $l_2 \notin \mathbb{L}_1$ **then**
            Cas 4 de la composition
        **else**
            **if** $\mathcal{R}_1(l_2) = "¿"$ **then**
                Cas 5 de la composition

$\mathcal{F} : \mathbb{Q} \times (\mathbb{X}_O \cup \mathbb{X}_L) \rightarrow$ CP de la forme $\dot{x_C} \geq c(\mathbb{X}), \dot{x_C} \leq c(\mathbb{X}), \dot{x_C} = c(\mathbb{X})$

$$((q_1, q_2), x_C) \mapsto \begin{cases} \mathcal{F}_1(q_1, x_C) & \text{si} \quad x_C \in \mathbb{X}_{O1} \cup \mathbb{X}_{L1} \\ \mathcal{F}_2(q_2, x_C) & \text{si} \quad x_C \in \mathbb{X}_{O2} \cup \mathbb{X}_{L2} \end{cases}$$

$\mathcal{I} : \mathbb{Q} \rightarrow \mathbb{P}(\mathbb{X})$
$(q_1, q_2) \mapsto \mathcal{I}_1(q_1) \wedge \mathcal{I}_2(q_2)$
**if** $Init_1 = (q_{01}, d_{01})$ ***et*** $Init_2 = (q_{02}, d_{02})$ **then**
    $Init = ((q_{01}, q_{02}), d_{01} \wedge d_{02})$

Les différents cas de composition sont développés dans l'algorithme 2. Nous allons détailler cet algorithme comme suit :

- Lignes [1 : 6] **Cas 1**. Le franchissement de la transition $t_1 \in H_1$ n'est pas synchronisé avec $t_2 \in H_2$ (voir fig. 2.8). Seul le franchissement de la transition appartenant à $H_1$ est permis et cette transition est ajoutée à l'ensemble de transitions $\mathbb{T}$. Ensuite, si la nouvelle situation atteinte $(q_{1d}, q_{2s})$ n'appartient pas à l'ensemble $\mathbb{Q}$, on l'ajoute à l'ensemble $\mathbb{Q}$ ainsi qu'à la pile. On calcule ensuite la fonction $\mathcal{A}$ et on ajoute à la liste d'affectation courante les affectations de la transition $t_1$.

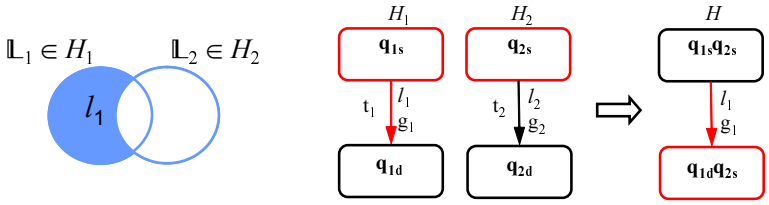



FIG. 2.8 – Premier cas de l'algorithme de composition 1

- Lignes [7 : 12] **Cas 2**. Le franchissement de la transition $t_1 \in H_1$ est synchronisé avec $t_2 \in H_2$ (voir fig. 2.9). Le franchissement des deux transitions est possible et elles sont ajoutées à l'ensemble de transitions $\mathbb{T}$. Ensuite, si la nouvelle situation atteinte $(q_{1d}, q_{2d})$ n'appartient pas à l'ensemble $\mathbb{Q}$, on l'ajoute à l'ensemble $\mathbb{Q}$ ainsi qu'à la pile. On calcule ensuite la fonction $\mathcal{A}$ et on ajoute à la liste d'affectation courante les affectations des transitions $t_1$ et $t_2$.

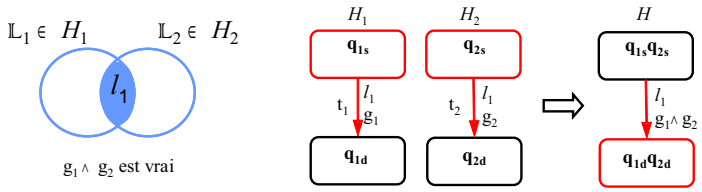



FIG. 2.9 – Deuxième cas de l'algorithme de composition 1

- Lignes [13 : 23] **Cas 3**. Le franchissement de $t_1 \in H_1$ n'est pas bloqué par la transition $t_2 \in H_2$ qui a un rôle non bloquant "¿" (voir fig. 2.10). Dans ce cas, les deux transitions porteuses du même nom ne sont pas obligées de s'attendre. La transition $t_1$ est franchissable dès que $g_1$ est validée à cause du rôle non bloquant de $t_2$. Ensuite, si la nouvelle situation atteinte $(q_{1d}, q_{2s})$ n'appartient pas à l'ensemble $\mathbb{Q}$, on l'ajoute à l'ensemble $\mathbb{Q}$ ainsi qu'à la pile. On calcule ensuite la fonction $\mathcal{A}$ et on ajoute à la liste d'affectation courante les affectations de la transition $t_1$.
- Lignes [24 : 29] **Cas 4**. Le franchissement de la transition $t_2 \in H_2$ n'est pas synchronisé avec $t_1 \in H_1$ (voir fig. 2.11). Seul le franchissement de la transition appartenant à $H_2$ est permis et cette transition est ajoutée à l'ensemble de transitions $\mathbb{T}$. Ensuite, si la nouvelle situation atteinte $(q_{1s}, q_{2d})$ n'appartient pas à l'ensemble $\mathbb{Q}$, on l'ajoute à l'ensemble $\mathbb{Q}$ ainsi qu'à la pile. On calcule ensuite la fonction $\mathcal{A}$ et on ajoute à la liste d'affectation courante les affectations de la transition $t_2$.

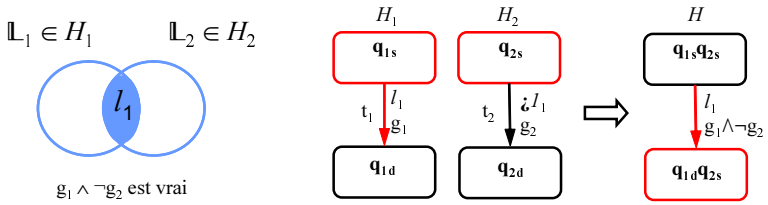



FIG. 2.10 – Troisième cas de l'algorithme de composition 1

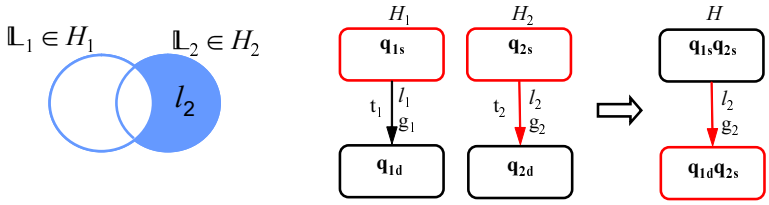



FIG. 2.11 – Quatrième cas de l'algorithme de composition 1

– Lignes [30 : 40] **Cas 5**. Le franchissement de $t_2 \in H_2$ n'est pas bloqué par $t_1 \in H_1$ qui a un rôle non bloquant "¿" (voir fig. 2.12). Dans ce cas, les deux transitions porteuses du même nom ne sont pas obligées de s'attendre. Cette fois, la transition $t_2$ est franchissable dès que $g_2$ est validée à cause du rôle non bloquant de $t_1$. Ensuite, si la nouvelle situation atteinte $(q_{1s}, q_{2d})$ n'appartient pas à l'ensemble $\mathbb{Q}$, on l'ajoute à l'ensemble $\mathbb{Q}$ ainsi qu'à la pile. On calcule ensuite la fonction $\mathcal{A}$ et on ajoute à la liste d'affectation courante les affectations de la transition $t_2$.

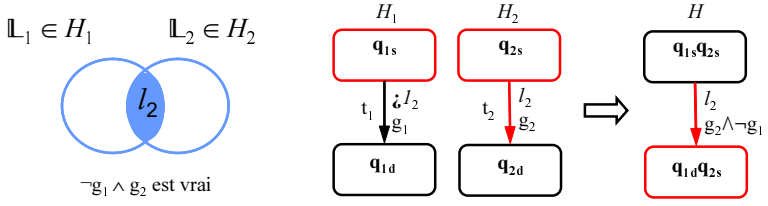



FIG. 2.12 – Cinquième cas de l'algorithme de composition 1

**Algorithme 2** : Cas de composition d'un réseau d'automates $HIOA_{TS}$

**Cas 1 de la composition**

$\mathbb{T} \leftarrow \mathbb{T} \cup \{(q_{1s}, q_{2s}) \xrightarrow{l_1, g_1} (q_{1d}, q_{2s})\}$

**if** $(q_{1d}, q_{2s}) \notin \mathbb{Q}$ **then**

  $\mathbb{Q} \leftarrow \mathbb{Q} \cup \{(q_{1d}, q_{2s})\}$

  EMPILER (PILE,$(q_{1d}, q_{2s})$)

$\mathcal{A}((q_{1s}, q_{2s}) \xrightarrow{l_1, g_1} (q_{1d}, q_{2s})) \leftarrow \mathcal{A}_1(q_{1s} \xrightarrow{l_1, g_1} q_{1d})$

**Cas 2 de la composition**

$\mathbb{T} \leftarrow \mathbb{T} \cup \{(q_{1s}, q_{2s}) \xrightarrow{l_1, g_1 \wedge g_2} (q_{1d}, q_{2d})\}$

**if** $(q_{1d}, q_{2d}) \notin \mathbb{Q}$ **then**

  $\mathbb{Q} \leftarrow \mathbb{Q} \cup \{(q_{1d}, q_{2d})\}$

  EMPILER (PILE,$(q_{1d}, q_{2d})$)

$\mathcal{A}((q_{1s}, q_{2s}) \xrightarrow{l_1, g_1 \wedge g_2} (q_{1d}, q_{2d})) \leftarrow \mathcal{A}_1(q_{1s} \xrightarrow{l_1, g_1} q_{1d}) \cup \mathcal{A}_2(q_{2s} \xrightarrow{l_2, g_2} q_{2d})$

**Cas 3 de la composition**

$g \leftarrow g_1$

**forall** $q_{2s} \xrightarrow{l_2, g_2} q_{2d} \in T_2$ **do**

  **if** $l_1 = l_2$ **then**

    $g \leftarrow g \wedge \neg g_2$

**if** $g \neq$ *"false"* **then**

  $\mathbb{T} \leftarrow \mathbb{T} \cup \{(q_{1s}, q_{2s}) \xrightarrow{l_1, g} (q_{1d}, q_{2s})\}$

  **if** $(q_{1d}, q_{2s}) \notin \mathbb{Q}$ **then**

    $\mathbb{Q} \leftarrow \mathbb{Q} \cup \{(q_{1d}, q_{2s})\}$

    EMPILER (PILE,$(q_{1d}, q_{2s})$)

$\mathcal{A}((q_{1s}, q_{2s}) \xrightarrow{l_1, g} (q_{1d}, q_{2s})) \leftarrow \mathcal{A}_1(q_{1s} \xrightarrow{l_1, g_1} q_{1d})$

**Cas 4 de la composition**

$\mathbb{T} \leftarrow \mathbb{T} \cup \{(q_{1s}, q_{2s}) \xrightarrow{l_2, g_2} (q_{1s}, q_{2d})\}$

**if** $(q_{1s}, q_{2d}) \notin \mathbb{Q}$ **then**

  $\mathbb{Q} \leftarrow \mathbb{Q} \cup \{(q_{1s}, q_{2d})\}$

  EMPILER (PILE,$(q_{1s}, q_{2d})$)

$\mathcal{A}((q_{1s}, q_{2s}) \xrightarrow{l_2, g_2} (q_{1s}, q_{2d})) \leftarrow \mathcal{A}_2(q_{2s} \xrightarrow{l_2, g_2} q_{2d})$

**Cas 5 de la composition**

$g \leftarrow g_2$

**forall** $q_{1s} \xrightarrow{l_1, g_1} q_{1d} \in T_1$ **do**

  **if** $l_2 = l_1$ **then**

    $g \leftarrow g \wedge \neg g_1$

**if** $g \neq$ *"false"* **then**

  $\mathbb{T} \leftarrow \mathbb{T} \cup \{(q_{1s}, q_{2s}) \xrightarrow{l_2, g} (q_{1s}, q_{2d})\}$

  **if** $(q_{1s}, q_{2d}) \notin \mathbb{Q}$ **then**

    $\mathbb{Q} \leftarrow \mathbb{Q} \cup \{(q_{1s}, q_{2d})\}$

    EMPILER (PILE,$(q_{1s}, q_{2d})$)

$\mathcal{A}((q_{1s}, q_{2s}) \xrightarrow{l_2, g} (q_{1s}, q_{2d})) \leftarrow \mathcal{A}_2(q_{2s} \xrightarrow{l_2, g_2} q_{2d})$

Pour illustrer la composition parallèle que l'on vient de définir, la composition des automates Feu X et Véhicule X de l'exemple du carrefour est présentée dans la figure 2.13. L'automate résultant de cette composition par synchronisations typées est constitué de 9 situations et 16 transitions.

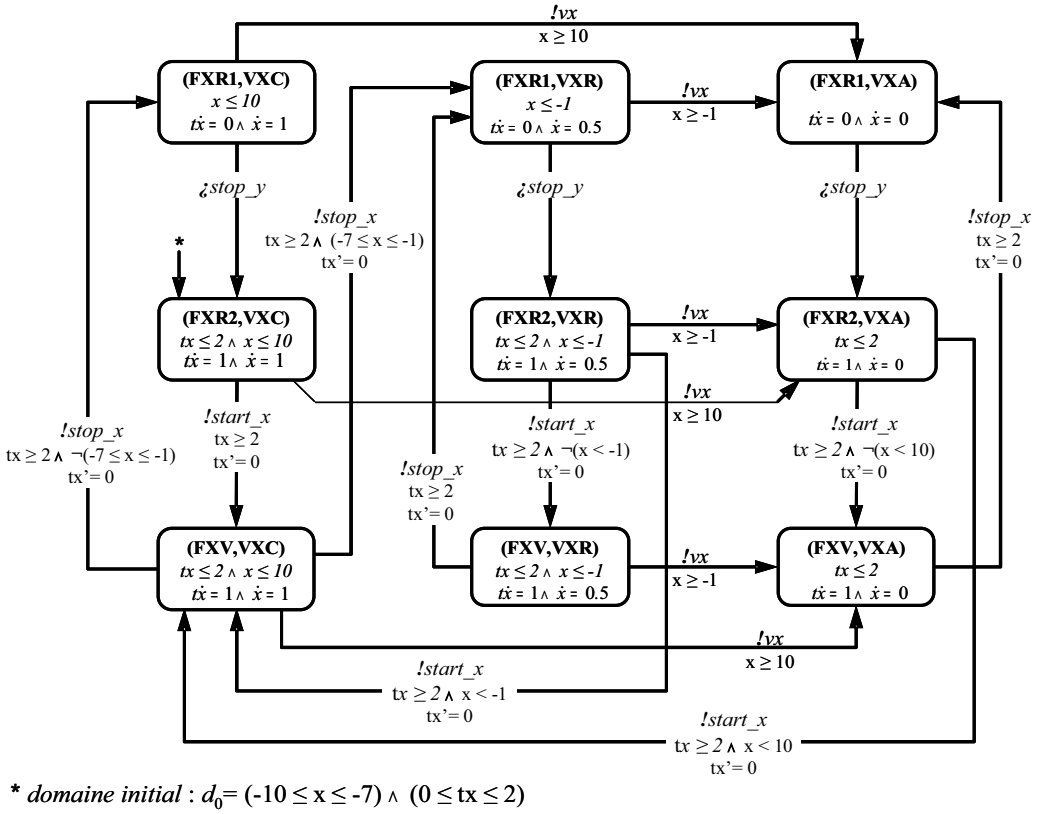



FIG. 2.13 – Composition des automates Véhicule X et Feu X

## 2.3 Traduction d'un réseau d'$HIOA_{TS}$

Toutes les utilisations d'un réseau d'automates $HIOA_{TS}$ ne requièrent pas la construction de l'automate composé. C'est par exemple le cas de la vérification par « model-checking » où le « model-checker » réalise lui-même la composition des automates à vérifier. Dans ce cas, il est plus facile de passer par une traduction du réseau d'automates $HIOA_{TS}$ dans la classe d'entrée de l'outil utilisé. Dans nos travaux, nous avons retenu PHAVer comme outil de vérification. Dans ce cas, les enrichissements que nous avons apportés à la classe d'automates $HIOA$ doivent donc être traduits à l'aide des seules synchronisations multiparties à rendez-vous bloquant dont dispose PHAVer. Pour un automate possédant des transitions avec une étiquette $l$ au rôle non bloquant, sa traduction implique la mise en place, depuis toutes les situations, d'une transition de type « self loop » avec l'étiquette $l$ et sans affectation. Si, avant traduction, une situation était déjà source d'une ou plusieurs transitions $t_i$ avec l'étiquette $l$, alors la garde de cette nouvelle transition doit être le complément logique de la conjonction des gardes des transitions $t_i$ (fig 2.14), sinon la garde est sans restriction.

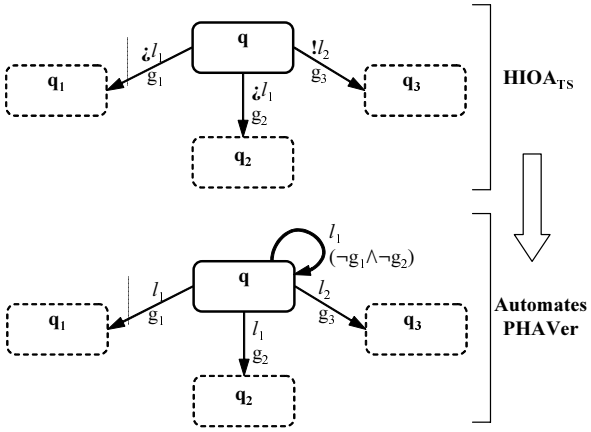



FIG. 2.14 – Traduction des automates $HIOA_{TS}$ dans la syntaxe du « model-checker » PHAVer

D'une manière plus générale, la traduction systématique d'un réseau d'automates $HIOA_{TS}$ en un réseau d'automates synchronisés par rendez-vous bloquant tels qu'utilisés dans PHAVer est décrite par l'algorithme 3. Pour illustrer cette traduction, le résultat de la traduction des modules Feu X et Véhicule X est présenté dans la figure 2.15.

**Algorithme 3** : Traduction d'un réseau d'automate $HIOA_{TS}$

**Data** : $R_{TS}$, un réseau d'automates $HIOA_{TS}$
**Result** : $R_{tr}$, un réseau d'automates PHAVer

$R_{tr} \leftarrow \emptyset$
**forall** $\langle \mathbb{Q}_c, \mathbb{X}_c, \mathbb{L}_c, \mathbb{T}_c, \mathcal{R}_c, \mathcal{A}_c, \mathcal{F}_c, \mathcal{I}_c, Init_c \rangle \in R_{TS}$ **do**
  $\mathbb{T}_{tr} \leftarrow \mathbb{T}_c$
  **forall** $l_c \in \{l \in \mathbb{L}_c \mid \mathcal{R}(l) = "¿"\}$ **do**
    **forall** $q_c \in \mathbb{Q}_c$ **do**
      $g_{self\_loop} \leftarrow \emptyset$
      **forall** $t \in \mathbb{T}_c \mid q = q_c \ et \ l = l_c$ **do**
        **if** $T_c = \emptyset$ **then**
          $\mathbb{T}_{tr} \leftarrow \mathbb{T}_{tr} \cup \{q_c \xrightarrow{l_c, vrai} q_c\}$
        **else**
          $\mathbb{T}_{tr} \leftarrow \mathbb{T}_{tr}$, $g_{self\_loop} \leftarrow vrai$
          **forall** $q_c \xrightarrow{l_c, g_c} q'_c \in T_c$ **do**
            $g_{self\_loop} \leftarrow g_{self\_loop} \wedge (\neg g_c)$
          $\mathbb{T}_{tr} \leftarrow \mathbb{T}_{tr} \cup \{q_c \xrightarrow{l_c, g_{self\_loop}} q_c\}$
  **forall** $t \in \mathbb{T}_{tr}$ **do**
    **if** $t \in \mathbb{T}_c$ **then**
      $\mathcal{A}_{tr}(t) \leftarrow \mathcal{A}_c(t)$
    **else**
      $\mathcal{A}_{tr}(t) \leftarrow \emptyset$
  $R_{tr} = R_{tr} \cup \langle \mathbb{Q}_c, \mathbb{X}_c, \mathbb{L}_c, \mathbb{T}_{tr}, \mathcal{A}_{tr}, \mathcal{F}_c, \mathcal{I}_c, Init_c \rangle$

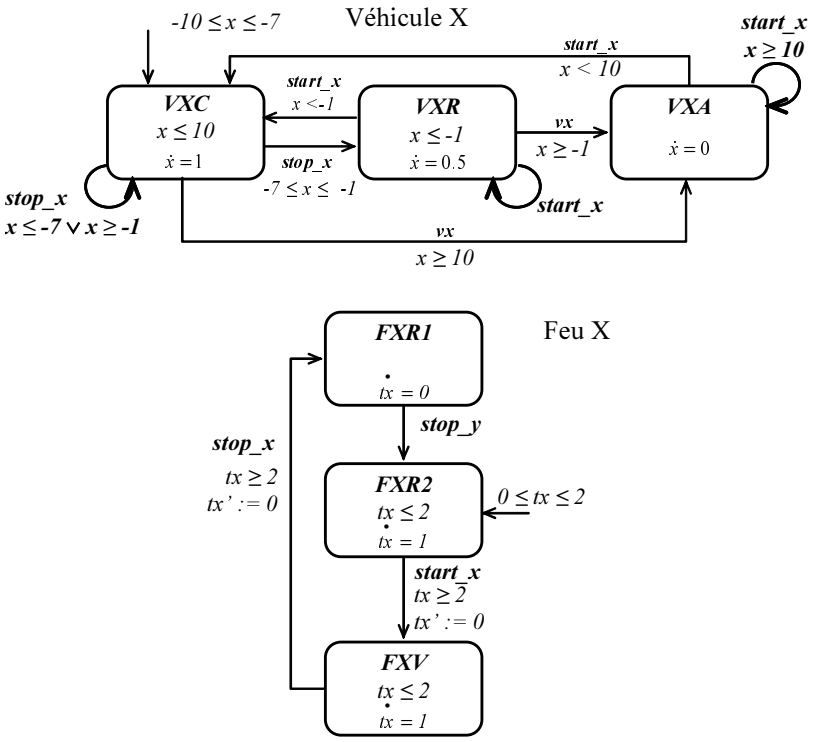



FIG. 2.15 – Traduction des automates Véhicule X et Feu X

Jusqu'à présent, nous avons donné les définitions formelles tant de la composition d'un réseau d'$HIOA_{TS}$ que de la traduction d'un réseau d'$HIOA_{TS}$ en $HIOA$. Ces définitions ne sont potentiellement pas exemptes d'erreurs. C'est pourquoi, même s'il ne s'agit pas d'une preuve formelle, nous allons comparer les résultats donnés par ces deux algorithmes (composition d'$HIOA_{TS}$ et traduction en réseau d'$HIOA$) afin de vérifier que les automates résultats sont porteurs des mêmes comportements.

À cette fin, nous nous proposons de déterminer la région atteignable pour les deux véhicules du carrefour (fig 2.5) à l'aide du « model-checker » PHAVer. Pour ce faire, le réseau d'$HIOA_{TS}$ modélisant le carrefour (fig 2.7) sera injecté en entrée de PHAVer :

- une première fois après composition selon les algorithmes 1 et 2 (puisque le résultat de la composition est un $HIOA_{TS}$ monolitique sans synchronisations donc un $HA$),
- une seconde fois après sa traduction selon l'algorithme 3 (devenant ainsi un $HIOA$)

La comparaison des résultats obtenus dans ces deux expériences devait nous donner la même région atteignable (figures 2.16 et 2.17).

L'objectif des feux de croisement, tel qu'il est décrit par le réseau d'automates $HIOA_{TS}$ de la figure 2.7, est d'éviter la présence simultanée des deux véhicules dans la zone du plan définie par $\{-1 \leq x \leq 1$ et $-1 \leq y \leq 1\}$.

Les figures 2.16 et 2.17 représentent la projection des positions atteignables par les deux véhicules dans le plan X et Y (les variables des automates véhicule X et véhicule Y). La figure 2.16 a été obtenue par calcul de la région atteignable du réseau d'$HIOA_{TS}$ de la figure 2.7, traduit dans la syntaxe d'entrée de PHAVer grâce à l'algorithme 3. La figure 2.17 donne une visualisation du calcul de la région atteignable de l'automate

composé (selon les algorithmes 1 et 2) des $HIOA_{TS}$ de la figure 2.7.

On peut constater que ces deux résultats sont identiques. À défaut de constituer une preuve formelle, notre confiance dans les algorithmes que nous avons développés est désormais renforcée. Par ailleurs, on établit bien la preuve que les deux véhicules ne peuvent jamais se trouver simultanément dans la zone critique du carrefour.

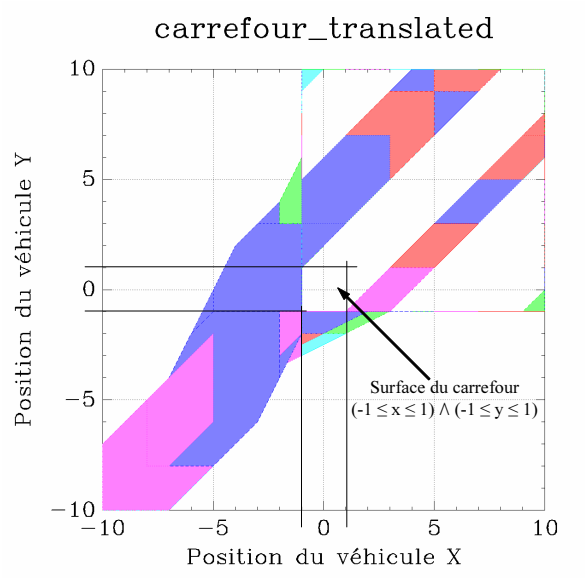



FIG. 2.16 – Positions atteignables par les deux véhicules. Résultat calculé par PHAVer à partir de la traduction du réseau d'$HIOA_{TS}$ par l'algorithme 3.

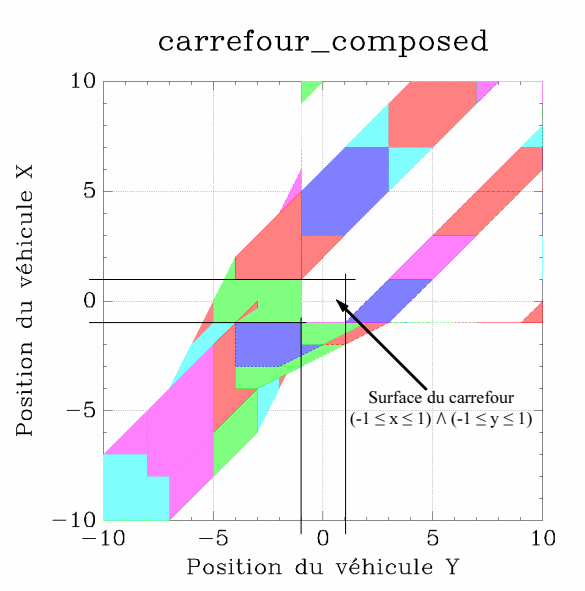



FIG. 2.17 – Positions atteignables par les deux véhicules. Résultat calculé par PHAVer à partir de l'automate composé des $HIOA_{TS}$ selon les algorithmes 1 et 2.

## 2.4 Instrumentation logicielle

L'instrumentation logicielle consiste en la mise en place d'un ensemble des fonctions programmées en langage Python. Nous avons développé une bibliothèque de fonctions, non seulement afin de valider les algorithmes présentés tout au long de ce chapitre, mais également pour fournir une assistance au modélisateur. Les fonctions de cette bibliothèque permettent de produire automatiquement une vue graphique du réseau d'$HIOA_{TS}$, du réseau traduit d'$HIOA$ et de l'$HA$ composé à partir de la saisie des Automates Hybrides à Entrées/Sorties et Synchronisations Typées. Elle permet de mettre au point des modèles de manière simple, de les recompiler selon la syntaxe de PHAVer ainsi que de présenter des résultats d'analyses. La figure 2.18 montre une vue d'ensemble de l'instrumentation logicielle réalisée.

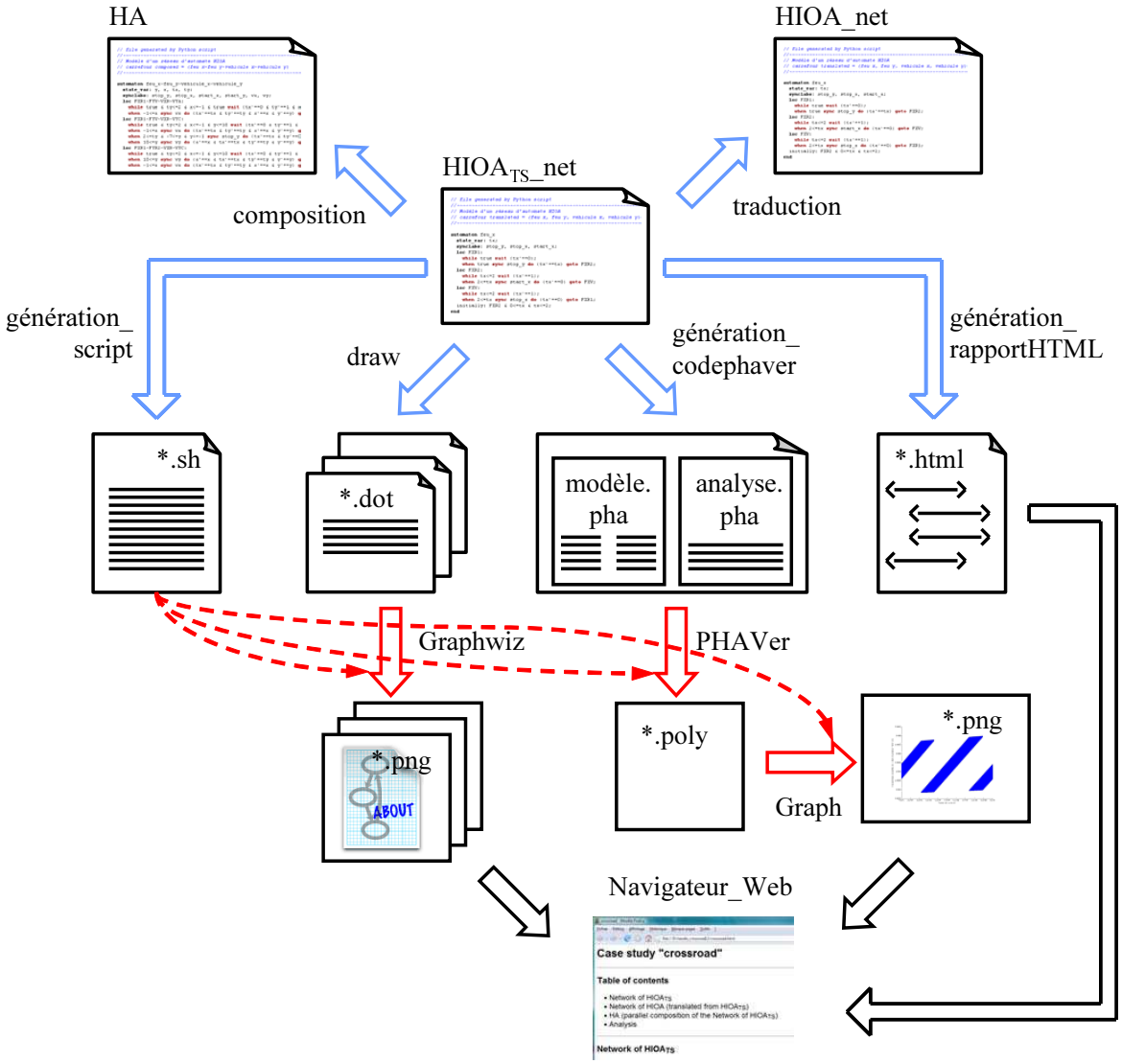



FIG. 2.18 – Vue globale de l'instrumentation logicielle

Le point de départ est une description textuelle des automates $HIOA_{TS}$ dans une classe développée sous Python. La figure 2.19 illustre la saisie de l'$HIOA_{TS}$ qui modélise

le Véhicule X de l'exemple du carrefour de la figure 2.7.

```
# ================================================
# Description du réseaux d'automates HIOATS
# ================================================

# ==== Définition de l'automates vehicule_x ========
vehicule_x = HIOATS("vehicule_x")
vehicule_x.add_local_var("x")
vehicule_x.add_input_label("stop_x","¿")
vehicule_x.add_input_label("start_x","¿")
vehicule_x.add_local_label("vx")
vehicule_x.add_location("VXC",["x'==1"],["x<=10"])
vehicule_x.add_location("VXR",["x'==0.5"],["x<=-1"])
vehicule_x.add_location("VXA",["x'==0"],["true"])
vehicule_x.add_transition("VXC","stop_x","-7<=x & x<=-1","VXR",["x'==x"])
vehicule_x.add_transition("VXC","vx","10<=x","VXA",["x'==x"])
vehicule_x.add_transition("VXR","vx","-1<=x","VXA",["x'==x"])
vehicule_x.add_transition("VXR","start_x","x<-1","VXC",["x'==x"])
vehicule_x.add_transition("VXA","start_x","x<10","VXC",["x'==x"])
vehicule_x.add_init_state("VXC",["-10<=x","x<=-7"])
```

FIG. 2.19 – Entrée de la description des automates $HIOA_{TS}$

Une fois les modèles $HIOA_{TS}$ ainsi décrits, les fonctions du script Python permettent l'instanciation des objets $HIOA_{TS}$ pour construire le réseau d'$HIOA_{TS}$ et le réseau d'$HIOA$. Les images générées automatiquement à partir du script Python sont présentées dans les figures 2.20 et 2.21.

## 2.5 Conclusion du chapitre 2

Dans ce chapitre, nous avons présenté une classe d'automates hybrides linéaires à synchronisations typées pour la modélisation modulaire des SDH complexes : les $HIOA_{TS}$. Elle offre plusieurs mécanismes de synchronisation entre automates : des synchronisations multiparties (1 automate émetteur, $N \geq 1$ automates récepteurs) sur rendez-vous bloquant, non bloquant ou mixte. Bien que construite sur la base des automates hybrides à entrées/sorties, notre proposition de mécanismes de synchronisation peut se transposer à toutes les classes d'automates incluant la notion d'étiquette de transition, puisque c'est le rôle attribué à l'étiquette qui est porteur de la sémantique de synchronisation.

De manière corollaire, en proposant de puissants mécanismes de synchronisation pour satisfaire les besoins des modélisateurs, un écart s'est creusé entre le formalisme $HIOA_{TS}$ et les formalismes d'entrée des principaux outils d'analyse ou de simulation des SDH. Pour combler cet écart, deux propositions complémentaires ont été présentées. La première est la définition d'un opérateur de composition des $HIOA_{TS}$. Il permet l'obtention, de façon systématique, d'un seul automate composé à partir d'un réseau d'$HIOA_{TS}$. Cet automate composé n'est donc plus porteur d'aucune marque de modularité ni de synchronisation, ce qui lui confère une large compatibilité pour

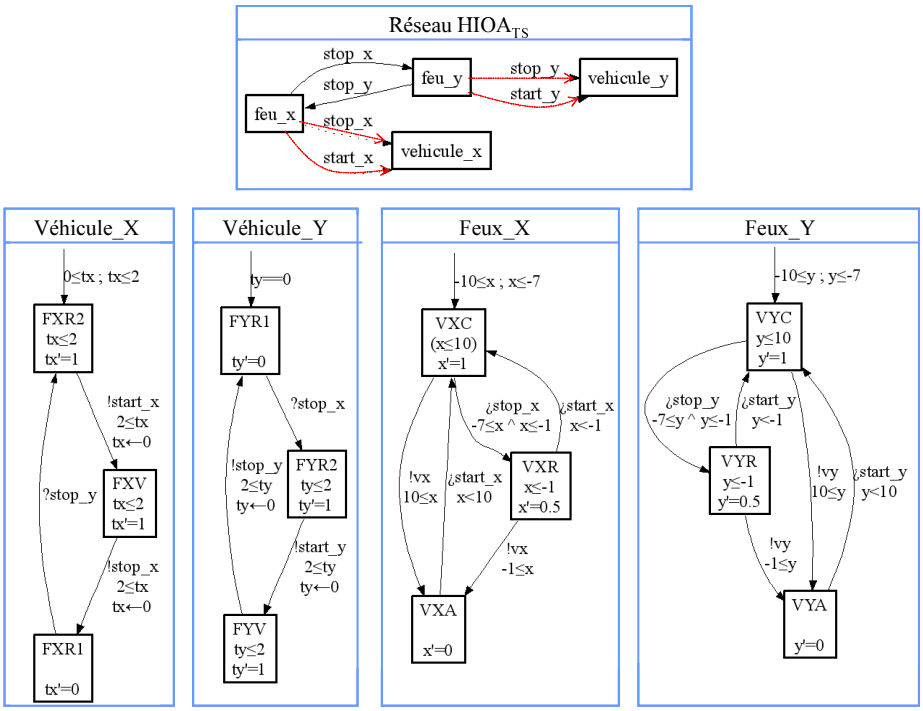



FIG. 2.20 – Images du réseau d'$HIOA_{TS}$ obtenues à partir du code de la figure 2.19 et de la fonction « draw » de la classe Python HIOATS

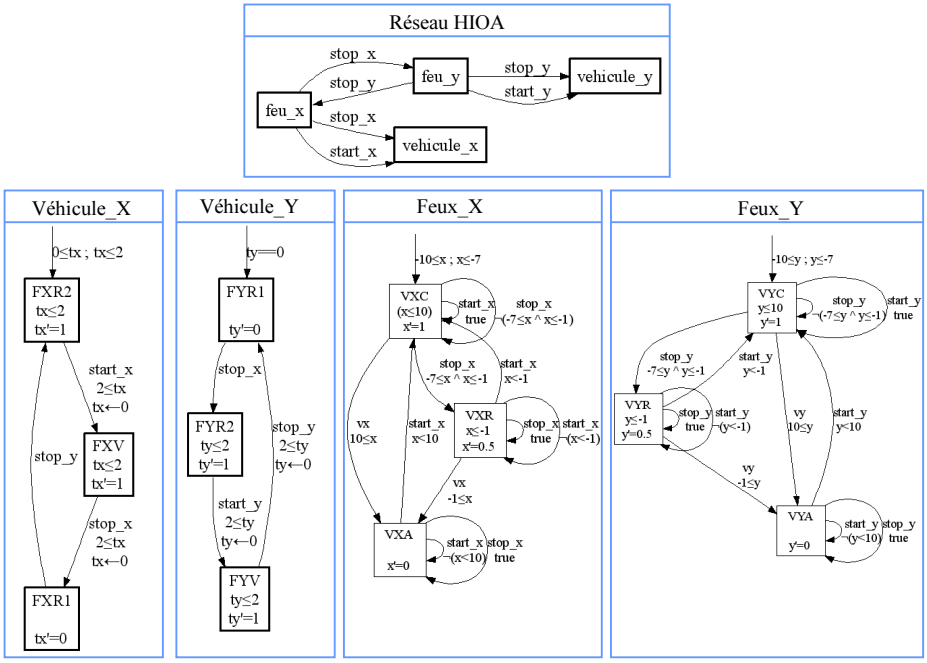



FIG. 2.21 – Images du réseau d'$HIOA$ obtenues après traduction de l'$HIOA_{TS}$ de la figure 2.19

tout usage ultérieur. La deuxième proposition visant à faciliter l'exploitation d'un réseau d'$HIOA_{TS}$ est un algorithme de traduction des mécanismes de synchronisations typées à l'aide du mécanisme le plus répandu de synchronisation : par rendez-vous bloquant.

Puisque notre objectif est celui de la vérification automatique de propriétés quantitatives par model-checking hybride, nous allons utiliser par la suite la traduction d'$HIOA_{TS}$ pour obtenir des automates appropriés au model-checker hybride PHAVer, à savoir les $HIOA$.

# *Chapitre 3*

# *Vérification des propriétés quantitatives : Méthodologie générale*

---

*Ce chapitre présente l'aspect méthodologique de nos travaux. Nous allons décrire les systèmes cibles auxquels nous allons nous intéresser et les hypothèses retenues pour leur modélisation. Ces hypothèses concernent des aspects liés au contrôleur, au processus, à leurs interactions ainsi qu'aux propriétés à vérifier. En se basant sur l'approche de modélisation modulaire retenue et détaillée dans le chapitre précédent, nous allons proposer une démarche pour la génération de modèles génériques à partir de chaque module constituant le système automatisé cible. Ainsi, la modélisation de ces systèmes sera ultérieurement réalisée par instanciation de ces modules génériques.*

## 3.1 Introduction

La méthodologie que nous proposons pour la vérification des systèmes automatisés consiste en trois étapes. D'abord, la modélisation des composants du système à l'aide des automates hybrides à entrées/sorties et synchronisations typées $HIOA_{TS}$ (formalisme proposé dans le chapitre précédent) pour composer un réseau d'automates. Ensuite, la traduction du réseau d'$HIOA_{TS}$ en un réseau d'$HIOA$ qui est le formalisme d'entrée pour le « model-checker » PHAVer. Enfin, la vérification des propriétés quantitatives.

Dans ce chapitre, nous allons détailler les systèmes automatisés auxquels nous nous intéressons et nous allons donner des guides de modélisation pour chaque composant. Deux études de cas seront ensuite analysées en utilisant cette méthodologie.

## 3.2 Description des systèmes automatisés cibles

### 3.2.1 Structure des systèmes automatisés cibles

Les systèmes automatisés sont composés de deux sous-systèmes en interaction : le contrôleur et le processus (voir figure 3.1). Ce dernier peut être un processus physique quelconque, comme un système manufacturier, un système de production par lots, un système ferroviaire ou un système monétique.

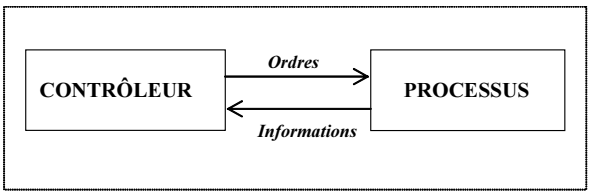



FIG. 3.1 – Système automatisé : interactions entre le contrôleur et le processus

Les interactions entre le contrôleur et le processus ne sont pas symétriques. Les ordres émis par le contrôleur entraînent une modification systématique que subit le processus tandis qu'une information émise par le processus n'implique pas nécessairement un changement d'état du contrôleur.

### 3.2.2 Description du processus

Le processus physique interagit avec le contrôleur via des capteurs et des actionneurs. Les capteurs transforment les caractéristiques d'un phénomène physique (mesurandes) en signaux qui informent le contrôleur. Ces mesurandes sont très variés, position, vitesse, accélération, pression, niveau, débit, masse, pH, ... Nous identifions trois types d'informations portées par les signaux issus des capteurs : les informations logiques ou Tout ou Rien (TOR), les informations numériques et les informations analogiques. Les actionneurs, quant à eux, sont chargés de piloter les composants du processus selon les ordres du contrôleur. Parmi les actionneurs les plus répandus, on

peut citer les vannes, les moteurs, les vérins, etc. Selon leur technologie, ils peuvent réagir soit à des ordres logiques (TOR), numériques ou analogiques.

En résumé, comme le montre la figure 3.2, les capteurs et les actionneurs[1] sont donc des composants du processus qui sont à l'interface avec le contrôleur.

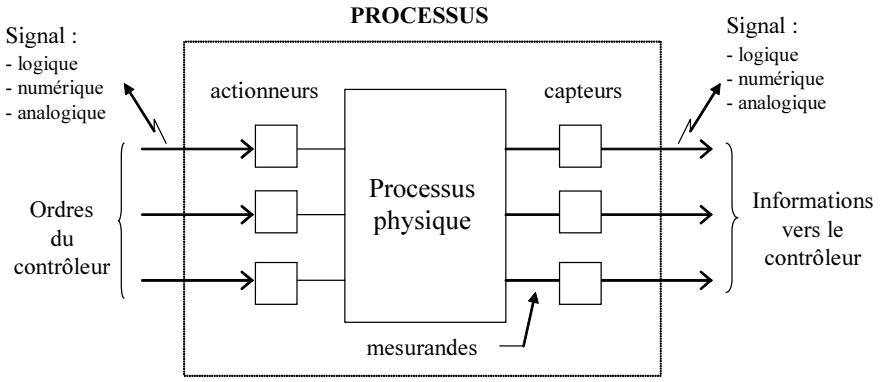



FIG. 3.2 – Fonction des capteurs et actionneurs dans le processus

### 3.2.3 Description du contrôleur

Le contrôleur est un Automate Programmable Industriel (API) ou un calculateur destiné à être utilisé dans un environnement industriel. Il fonctionne de manière numérique et utilise une mémoire programmable pour le stockage interne de variables et de programmes d'application. On qualifie le contrôleur de contrôleur logique lorsque les programmes mettent en œuvre majoritairement des fonctions logiques, de mise en séquence, de temporisation et de comptage. Le contrôleur dispose également de fonctions d'interface avec les capteurs et les actionneurs. Pour les capteurs (resp. actionneurs) avec une interface analogique, le contrôleur dispose d'un convertisseur analogique-numérique (resp. numérique-analogique) de sorte que le fonctionnement interne du contrôleur soit entièrement numérique.

L'exécution des programmes se fait grâce à une fonction du système d'exploitation appelée "moniteur". Le moniteur peut gérer plusieurs tâches qui exécutent toutes séquentiellement un ensemble de programmes et de blocs fonctionnels. Selon la norme CEI 61131-3 [IEC93], il existe deux types de tâches :

- les tâches périodiques, qui exécutent à intervalles périodiques leurs programmes
- les tâches événementielles, qui exécutent leurs programmes sur occurrence d'un événement booléen.

Si un programme n'est pas associé à une tâche, il est exécuté en boucle de manière cyclique. La figure 3.3 montre l'exemple d'un programme composé de 2 blocs fonctionnels implantés dans la même ressource processeur/mémoire du contrôleur et exécuté par une tâche périodique de période Tc.

Ce détail de description d'un contrôleur et de son fonctionnement a son importance, puisque certaines propriétés quantitatives dépendent de la "réactivité" du contrôleur,

[1]Nous considérons comme indissociable le couple {actionneur, pré-actionneur} (voir l'étude de cas 1 du chapitre 4) et on l'appellera par la suite simplement actionneur.

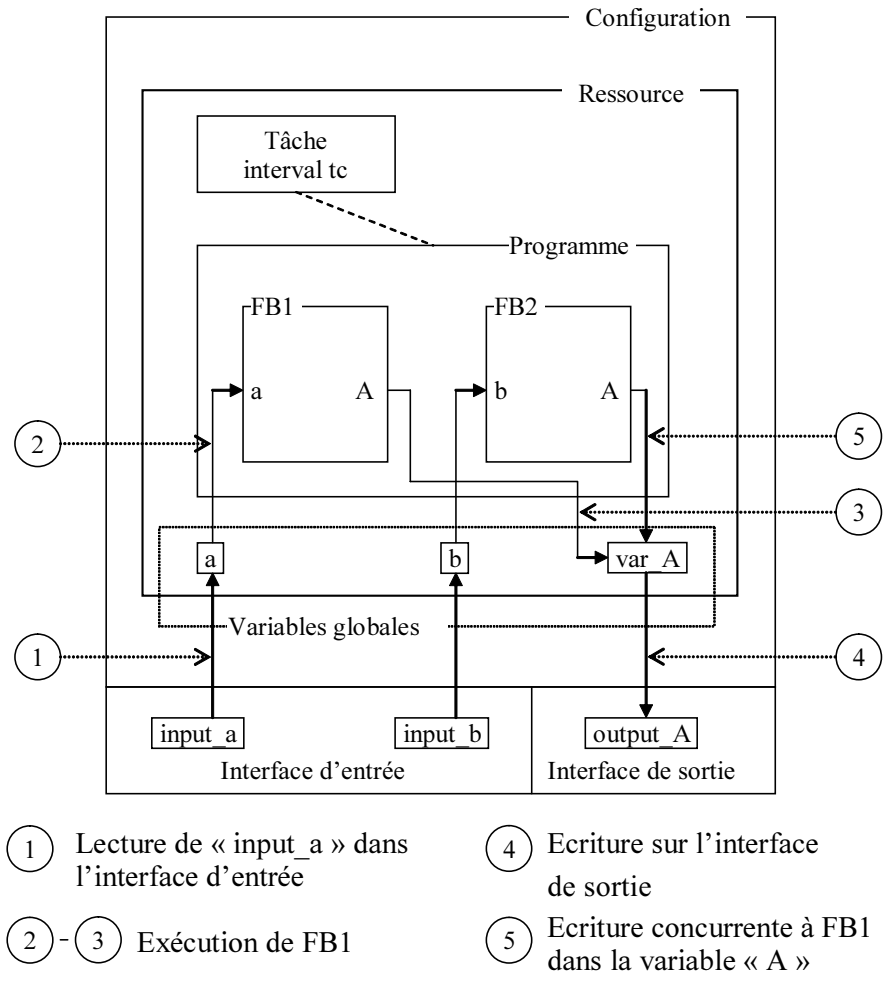



FIG. 3.3 – Exemple de programme d'application

c'est-à-dire de la vitesse d'enchaînement des activités (1), (2), (3), (4), et événementiellement de la valeur de la sortie « output_A », qui dépend elle-même de la dernière activité (3) ou (5) qui s'est produite avant (4).

## 3.3 Hypothèses retenues pour la suite de l'étude

Pour la modélisation des systèmes automatisés, nous avons retenu certaines hypothèses :

- Aucune restriction concernant le processus.
- Aucune restriction concernant les capteurs/actionneurs. Nous allons prendre en compte les trois types de signaux qu'ils peuvent échanger avec le contrôleur, à savoir les signaux logiques, numériques et analogiques.
- Le contrôleur est constitué d'un seul processeur de traitement (une seule ressource au sens de la norme CEI 31131-3 [IEC93]).
- Les capteurs/actionneurs sont directement connectés au contrôleur (pas d'entrées/sorties reportées sur un réseau local industriel).
- Un seul programme d'application est implanté dans le contrôleur, mais il peut

être constitué d'autant de blocs fonctionnels que nécessaire.
- Les blocs fonctionnels sont décrits avec le langage de programmation « Sequential Function Chart » (SFC) [IEC93].
- Le moniteur d'exécution du programme d'application est monotâche.
- La tâche d'exécution est de type périodique.
- Les interactions homme/machine ne sont pas considérées.
- L'étude des propriétés quantitatives ne portera que sur les phases de fonctionnement entièrement automatiques.

## 3.4 Modélisation par $HIOA_{TS}$ d'un système automatisé : principe de modélisation

Nous allons montrer dans cette section la démarche de modélisation d'un système, en mettant en évidence les parties génériques de chaque module.

### 3.4.1 Structure générale du modèle

Dans le chapitre précédent, nous avons présenté un formalisme modulaire pour la modélisation de systèmes complexes. Il permet la construction d'un modèle global du système par l'interconnexion des différents modules. En se basant sur ce principe, le modèle global est constitué d'un réseau d'$HIOA_{TS}$ qui contient des modules concernant le processus et le contrôleur. La figure 3.4 présente le modèle générique d'un système automatisé structuré comme suit :

$$
\begin{array}{l}
R_{System} = R_{Controller} \cup R_{Processus} \\
R_{Controller} = (\langle H_{Monitor1} \rangle \cup R_{SFC} \cup R_{Variables} \cup R_{Output}) \cup \\
(\langle H_{Monitor2} \rangle \cup R_{SFC} \cup R_{Thd} \cup R_{Output}) \\
R_{Processus} = R_{Actuator} \cup R_{Process} \cup R_{Sensor}
\end{array}
$$

où $R_{System}, R_{Controller}, R_{Processus}, R_{SFC}, R_{Variables}, R_{Output}, R_{Thd}, R_{Actuator}, R_{Process}, R_{Sensor}$ sont des réseaux d'$HIOA_{TS}$ et $H_{Monitor1}$ et $H_{Monitor2}$ sont deux $HIOA_{TS}$.

Le modèle du processus contient ainsi les modèles $HIOA_{TS}$ pour les actionneurs ($H_{Actuator_i} \in R_{Actuator}$), les processus physiques ($H_{Process_i} \in R_{Process}$), les capteurs analogiques ($H_{ASensor_i} \in R_{Sensor}$) et les capteurs digitaux ($H_{DSensor_i} \in R_{Sensor}$). D'autre part, le modèle du contrôleur est composé des modèles $HIOA_{TS}$ du programme d'application ($H_{SFC_i} \in R_{SFC}$), du moniteur d'exécution périodique ($H_{Monitor1}$), du moniteur d'exécution a temps de cycle nul ($H_{Monitor2}$), des modèles des détecteurs de seuils « Threshold Detector » ($H_{Thd_i} \in R_{Thd}$) pour le traitement des signaux continus issus du processus physique, des modèles des variables partagées ($H_{Variables_i} \in R_{Variables}$) et des modèles des variables de sortie ($H_{Output_i} \in R_{Output}$).

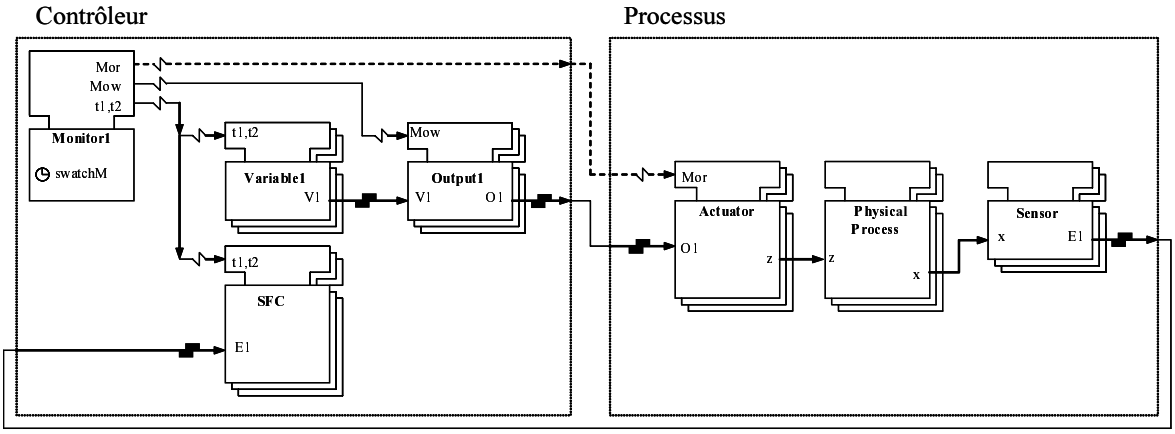



FIG. 3.4 – Modèle générique d'un système automatisé modélisé par un réseau d'$HIOA_{TS}$

### 3.4.2 Modélisation du processus

#### Modules d'actionneurs : Actuator

Les actionneurs sont des composants qui agissent directement sur le fonctionnement du processus en fonction des ordres issus du contrôleur, par exemple, pour arrêter/démarrer un moteur ou ouvrir/fermer une vanne. La communication entre le contrôleur et les actionneurs est faite par les interfaces de sortie du contrôleur. Une sortie du contrôleur apparaît comme une variable de sortie d'un $HIOA_{TS}$ du contrôleur. Cependant, partager des variables logiques ou numériques entre deux $HIOA_{TS}$ ne permet pas d'imposer par l'un une évolution discrète de l'autre. En effet, ces variables partagées logiques ou numériques n'ont pas une évolution continue ($\dot{x} = 0$), ce qui rend impossible la maîtrise de la date de franchissement d'une transition par un couple {invariant de situation, garde de transition}.

Ainsi, le modèle générique d'un actionneur doit avoir une synchronisation non bloquante avec le module du moniteur d'exécution. Le module du moniteur émet l'étiquette de synchronisation *Mor* pour informer que la valeur des sorties a été mise à jour. Le rôle de cette étiquette doit être non bloquant dans l'actionneur de façon à ce que les modèles des actionneurs n'imposent rien au moniteur.

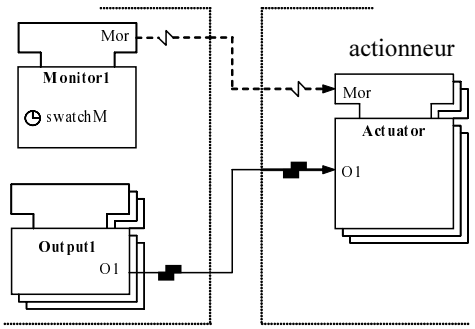



FIG. 3.5 – Aspects génériques d'un modèle d'actionneur

La figure 3.5 présente les aspects génériques d'un modèle d'actionneur. Ils se limitent

à deux entrées, une étiquette et une variable. Le contenu du module est à spécifier pour chaque actionneur. De manière formelle, un module de ce type est défini comme :

$$H_{Actuator} = \langle \mathbb{Q}, \mathbb{X}, \mathbb{L}, \mathbb{T}, \mathcal{A}, \mathcal{F}, \mathcal{I}, Init, \mathcal{R} \rangle \text{ , avec}$$

$Mor \in \mathbb{L}_I$ et $\mathcal{R}(Mor) =$ "¿"
$\exists\ x \in \mathbb{X}_{O_{Output}} \mid x \in \mathbb{X}_{I_{Actuator}}$

#### Modules de capteurs : Sensor

Les capteurs sont des équipements permettant de capturer l'état des grandeurs physiques du processus. Les capteurs peuvent générer des signaux analogiques ou digitaux (logiques/numériques). Lorsque le capteur doit produire un signal analogique à partir d'un phénomène physique déjà modélisé par une variable dans un $HIOA_{TS} \in R_{Processus}$, le modèle du capteur se réduit à une simple transmission d'une variable.

Ainsi, sauf si l'on veut intégrer des phénomènes de saturation ou d'hystérésis au comportement du capteur, on peut s'abstenir d'ajouter un modèle pour le capteur et transmettre directement la variable élaborée par un module de processus physique au contrôleur. Pour les détecteurs, le signal logique généré dépend de la valeur du mesurande par rapport à la zone de détection du détecteur.

La figure 3.6a montre l'évolution type du signal logique "s" généré par le détecteur en fonction du mesurande "x" du processus physique. Le modèle générique correspondant du détecteur est donné sur la figure 3.6b. Seules les valeurs haut et bas de la zone de détection sont à instancier pour obtenir le modèle particulier d'un détecteur.

De manière formelle, un module de ce type est défini comme :

$$H_{Sensor} = \langle \mathbb{Q}, \mathbb{X}, \mathbb{L}, \mathbb{T}, \mathcal{A}, \mathcal{F}, \mathcal{I}, Init, \mathcal{R} \rangle \text{ , avec,}$$

$\exists\ x \in \mathbb{X}_{O_{Process}} \mid x \in \mathbb{X}_{I_{Sensor}}$

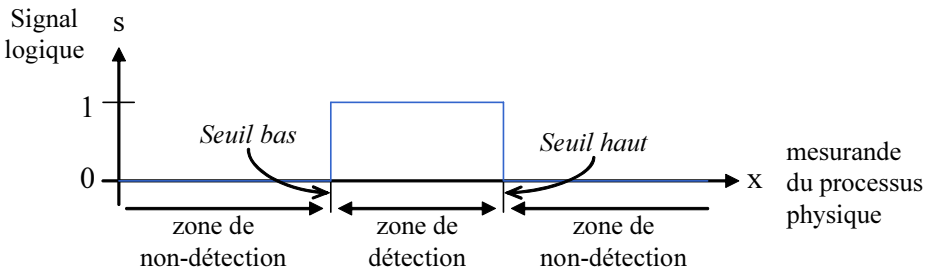
### 3.4.3 Modélisation du contrôleur

#### Module générique d'un programme SFC et des variables globales partagées

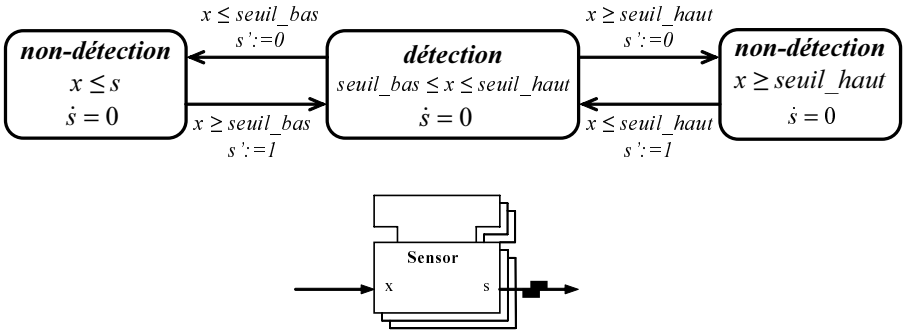
Les programmes de contrôle seront écrits dans un des langages issus de la norme CEI 61131-3 [IEC93], à savoir le « Sequential Function Charts » (SFC). Ce langage de haut niveau permet la programmation aisée de tous les processus séquentiels dans l'automate programmable. Ce langage a fait l'objet de différents travaux en vérification formelle, en particulier pour lui donner une syntaxe et une sémantique formelles [BHLE04], [FB05].

Nous allons nous appuyer sur la syntaxe formelle proposé par [BHLE04] mais en limitant la définition à la sous-classe suivante des SFC, suffisante à nos travaux :

- Pas de notion de parallélisme. On ne considère que des transitions simples ou avec bifurcations alternatives (voir figure 3.7).
- Pas de priorités définies entre les transitions. Pour éviter les parallélismes interprétés, les réceptivités doivent être mutuellement exclusives.




**a)** Signal logique "s" en fonction du mesurande "x"




**b)** Aspects génériques d'un modèle de détecteur

FIG. 3.6 – Modélisation des modules des détecteurs

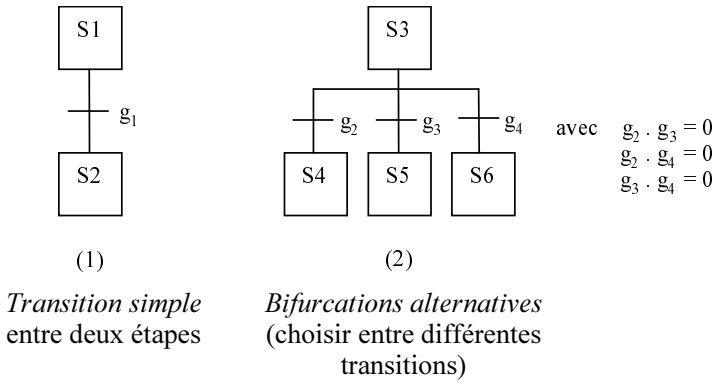



FIG. 3.7 – Types de transitions considérées par la sous-classe des SFC considérée

– Pas de notion de temps. Les variables "step.T" ne sont pas modélisées.

Ainsi, un SFC est un ensemble $SFC = (X, S, A_c, S_o, T_r, block)$, où :

– $X = X_I \cup X_O \cup X_S$ est un ensemble fini de variables, où $X_I$ est l'ensemble des variables d'entrée (uniquement lues par le SFC), $X_O$ est l'ensemble des variables de sortie (modifiées par le SFC) et $X_S$ est l'ensemble des variables d'activité d'étape ($step.X$),
– $S$ est l'ensemble fini des étapes,
– $A_c$ est l'ensemble fini des actions de contrôle,
– $S_o$ est l'étape initiale,
– $T_r \subseteq (S \times G \times S)$ est l'ensemble fini des transitions. C'est ici que la principale restriction de notre sous classe de SFC est définie : une seule étape en amont et une seule étape en aval d'une transition. $G$ est l'ensemble de toutes les réceptivités qui sont des expressions booléennes des variables logiques de $X$ et des prédicats construits à partir des variables de $X$.
– $block$ est une fonction qui assigne un ensemble de blocs d'action à chaque étape appartenant à $S$. B représente l'ensemble de tous les blocs d'action. Un bloc d'action est un couple $(a, quali)$ tel que $a \in A_c$ et $quali \in \{N, S, R, P0, P1\}$ est un qualificatif de l'action (N : non mémorisée, S : mise à 1, R : mise à 0 prioritaire, P1 impulsion front montant, P0 : impulsion front descendant).

Nous proposons une démarche générique pour la construction du modèle $HIOA_{TS}$ qui représente un bloc fonctionnel SFC. Le principe de la traduction d'un SFC en $HIOA_{TS}$ repose sur la séparation entre les évolutions du SFC et ses actions. Un bloc fonctionnel SFC1 agissant sur deux variables de sorties sera ainsi traduit par un module $HIOA_{TS}$ modélisant les étapes, les transitions et les réceptivités du SFC, et par deux modules $HIOA_{TS}$ modélisant les variables de sortie avec les actions qui leur sont associées. La figure 3.8 présente le principe général de cette traduction.

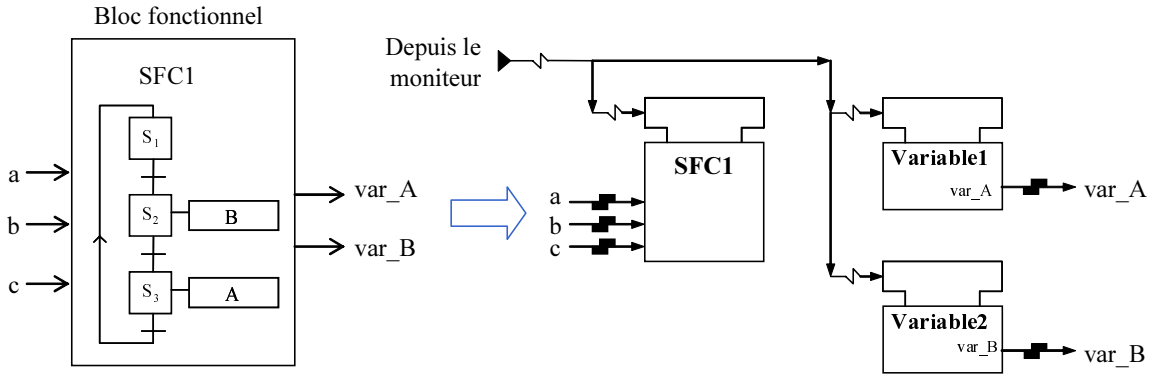



FIG. 3.8 – Principe de traduction d'un $SFC$ en $HIOA_{TS}$

Les évolutions du SFC sont réalisées à l'initiative du moniteur d'exécution qui est le seul à fixer des dates d'exécution des programmes et des blocs fonctionnels. À la demande du module $HIOA_{TS}$ du moniteur, le module $HIOA_{TS}$ du SFC devra accepter le franchissement de l'une des transitions, soit une transition issue du SFC traduisant un changement de situation du SFC, soit une transition en boucle sur une situation du modèle $HIOA_{TS}$ traduisant une exécution du SFC n'ayant pas conduit à un changement d'activité d'étapes de ce dernier.

Ce contrôle du modèle à l'aide de synchronisations est également rendu nécessaire à cause de la différence fondamentale entre le comportement déterministe d'un SFC et le comportement non déterministe d'un automate hybride. En effet, comme l'illustre la figure 3.9, lorsqu'un automate hybride ne manipule que des variables à évolution discrète, il ne peut pas contraindre les dates de franchissement, alors qu'une transition d'un SFC est franchie dès que la transition est validée (étape active) et que sa réceptivité est vraie. Dans des conditions semblables (situation active et garde vraie), un $HIOA_{TS}$ se contente d'autoriser le franchissement sans le rendre obligatoire dans l'instant. Le recours à une synchronisation avec le moniteur d'exécution pour imposer la date de franchissement nous paraît donc indispensable.

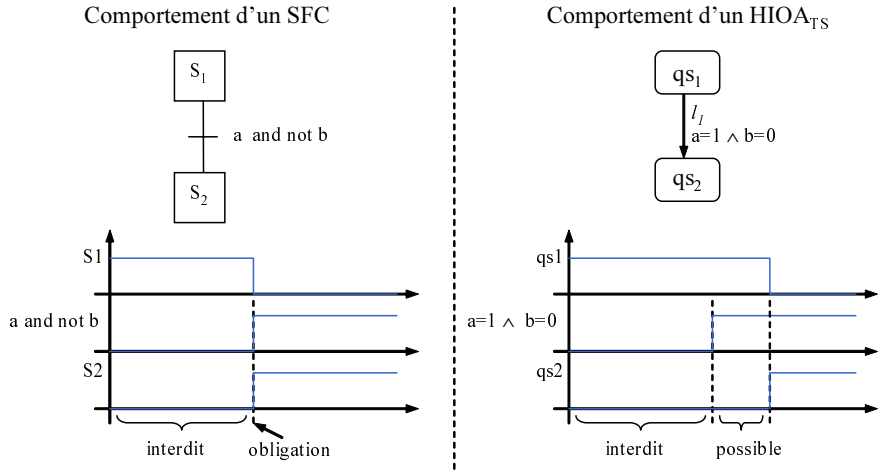



FIG. 3.9 – Différences de comportement entre un SFC et un $HIOA_{TS}$

La figure 3.10 montre un exemple de SFC avec trois entrées et trois sorties. On remarquera que la variable "c", qui sert de compteur, est à la fois une entrée et une sortie du SFC.

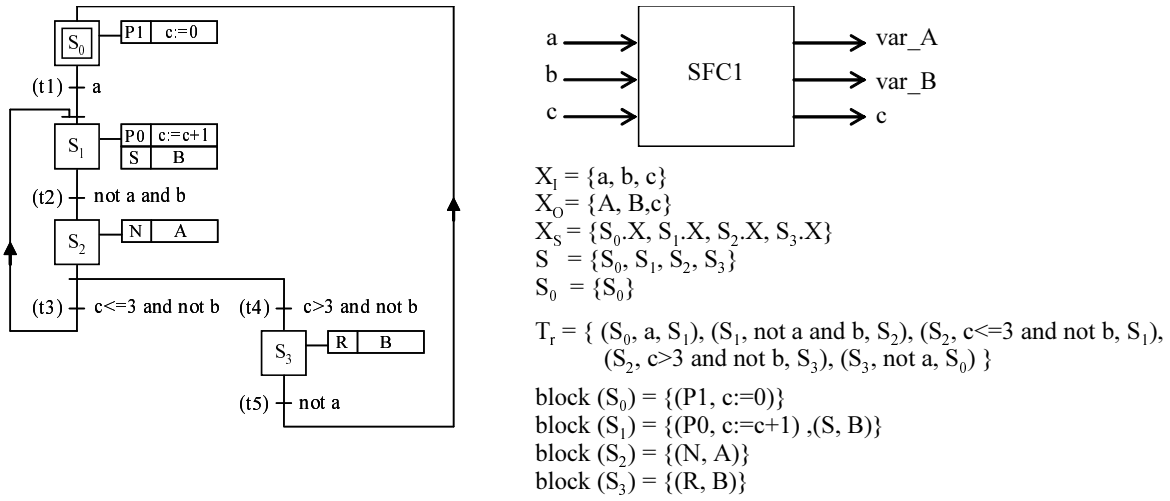



FIG. 3.10 – Exemple de modélisation par $HIOA_{TS}$ du SFC à 3 entrées/3 sorties

La figure 3.11 donne la structure du réseau d'$HIOA_{TS}$ qui traduit la structure du SFC de la figure 3.10. On y retrouve un modèle pour la structure du SFC et les trois

modules correspondant aux variables de sortie. La notation retenue pour les étiquettes de synchronisation est la suivante : $syncS1t3$ désigne l'étiquette de la transition $t3$ du SFC1, et $syncS1$ désigne l'étiquette des transitions qui bouclent sur la même situation dans le SFC1.

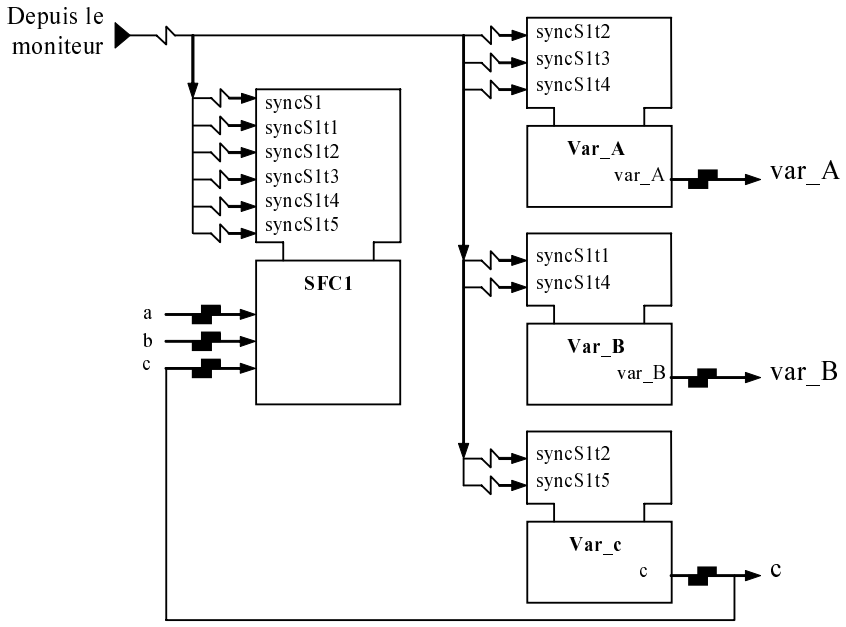



FIG. 3.11 – Réseau d'$HIOA_{TS}$ du SFC1 de la figure 3.10

La figure 3.12 présente le détail des quatre modules du réseau de la figure 3.11. On voit que chaque étape du SFC est traduite par une situation de l'$HIOA_{TS}$ du SFC et que chaque transition du SFC a son équivalent dans le modèle traduit avec comme garde la réceptivité de la transition du SFC. Les transitions en boucle sur chaque situation ont une garde correspondant à la conjonction de la négation des gardes des autres transitions issues de cette situation. De la sorte, lorsque le moniteur demandera "l'exécution du SFC" en émettant l'une des synchronisations $syncS1t_i$ avec $i \in [\![1,5]\!]$ ou $syncS1$ alors

- au moins une synchronisation est possible (le SFC est exécutable à tout moment),
- si une synchronisation $syncS1t_i$ avec $i \in [\![1,5]\!]$ est possible, alors $syncS1$ est impossible (si des transitions du SFC sont franchissables, alors l'une d'elles sera franchie dès la prochaine demande d'exécution de la part du moniteur).

De manière formelle, si $SFC_j = (X, S, A_c, S_o, T_r, block)$ est un bloc fonctionnel SFC, alors le module $HIOA_{TS}$ qui traduit sa structure est défini comme suit :

$$H_{SFC_j} = \langle \mathbb{Q}, \mathbb{X}, \mathbb{L}, \mathbb{T}, \mathcal{A}, \mathcal{F}, \mathcal{I}, Init, \mathcal{R} \rangle \text{ , avec}$$

- $\mathbb{Q} = S$
- $\mathbb{X} = \mathbb{X}_I \cup \mathbb{X}_L \cup \mathbb{X}_O$ avec $\mathbb{X}_I = X_I$, $\mathbb{X}_L = \emptyset$ et $\mathbb{X}_O = \emptyset$
- $\mathbb{L} = \mathbb{L}_I \cup \mathbb{L}_L \cup \mathbb{L}_O$ avec $\mathbb{L}_L = \emptyset$, $\mathbb{L}_O = \emptyset$ et $\mathbb{L}_I = \{syncS_j\} \cup \{syncS_jt_i\}$ avec $i \in [\![1, card(T_r)]\!]$
- $\mathbb{T} = \{s \xrightarrow{syncS_jt_i,g} s' \mid \exists\ (t_i = s \xrightarrow{g} s') \in T_r\} \cup \{s, syncS_jt_i, \bigwedge_{g_i \in GS(s)} \neg g_i, s'\}\ \exists\ s \in S\}$

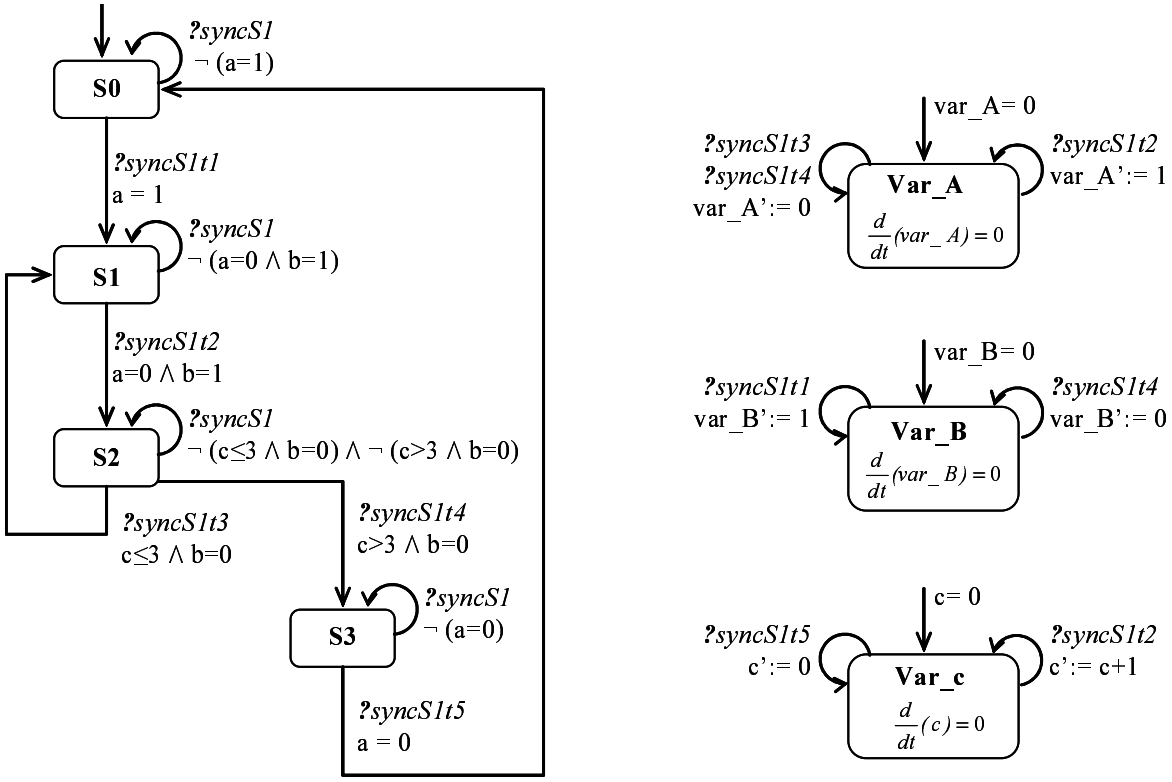



FIG. 3.12 – Modules $HIOA_{TS}$ du réseau de la figure 3.11

où $GS(s)$ est l'ensemble des gardes des transitions de $SFC_j$ dont $s$ est l'étape source. $GS(s) = \{g \in G \mid \exists\ s' \in S \text{ et } \exists\ (s \xrightarrow{g} s') \in T_r\}$
- $\forall\ t \in \mathbb{T} : \mathcal{A}(t) = \emptyset$ ($H_{SFC_j}$ ne contrôle pas de variables $\mathbb{X}_L = \mathbb{X}_O = \emptyset$)
- $\forall\ q \in \mathbb{Q} : \mathcal{F}(q) = \emptyset$ ($H_{SFC_j}$ ne contrôle pas de variables $\mathbb{X}_L = \mathbb{X}_O = \emptyset$)
- $\forall\ q \in \mathbb{Q} : \mathcal{I}(q) = \text{true}$
- $Init = (S_0, \emptyset)$
- $\forall\ l \in \mathbb{L}\ \mathcal{R}(l) =$" ?"

Les variables de sortie du SFC seront modélisées séparément pour qu'elles puissent être modifiées si nécessaire par plusieurs SFC. Dans un contrôleur, elles apparaissent donc comme des variables globales qui peuvent être partagées par plusieurs blocs fonctionnels et programmes. Le modèle $HIOA_{TS}$ d'une variable est formé d'une seule situation et d'autant de transitions sur elle-même qu'il y a de transitions dans les modules $HIOA_{TS}$ des SFC qui impliquent une modification de sa valeur. De manière formelle le modèle $HIOA_{TS}$ d'une variable $V$ est défini comme suit :

$$H_V = \langle \mathbb{Q}, \mathbb{X}, \mathbb{L}, \mathbb{T}, \mathcal{A}, \mathcal{F}, \mathcal{I}, Init, \mathcal{R} \rangle \text{ , avec}$$

- $\mathbb{Q} = \{var\_V\}$
- $\mathbb{X} = \mathbb{X}_I \cup \mathbb{X}_L \cup \mathbb{X}_O$ avec $\mathbb{X}_I = \mathbb{X}_L = \emptyset$ et $\mathbb{X}_O = \{var\_V\}$
- $\mathcal{F}(var\_V) =$" $\frac{d}{dt}(var\_V) = 0$"
- $\mathcal{I}(var\_V) = \text{true}$
- $Init = (var\_V,$"$var\_V = 0$")

L'ensemble $\mathbb{T}$ des transitions et la fonction $\mathcal{A}$ des assignations sont construits avec l'algorithme 4, avec $T_{pr_i}(s)$ l'ensemble des transitions du $SFC_i$ dont l'étape $s$ est la

destination et $T_{post_i}(s)$ l'ensemble des transitions du $SFC_i$ dont l'étape $s$ est la source.

**Algorithme 4** : Construction de l'ensemble $\mathbb{T}$ et de la fonction $\mathcal{A}$

1 $T_{pr_i}(s) = \{(s_1 \xrightarrow{g} s_2) \in T_r \mid s_2 = s\}$
2 $T_{post_i}(s) = \{(s_1 \xrightarrow{g} s_2) \in T_r \mid s_1 = s\}$
3 **forall** $SFC_i = (X_i, S_i, A_{c_i}, S_{0_i}, T_{r_i}, block_i)$ **do**
4  **forall** $s_j \in S_i$ **do**
5  **forall** $(a_k, quali_k) \in block(s_j)$ **do**
6  soit $var_k$ la variable modifiée dans $a_k$
7  soit $H_{var_k}$ l'automate $HIOA_{TS}$ qui modélise le comportement de la variable $var_k$
8  soit $q_{var_k}$ la seule situation de $H_{var_k}$
9  soit $\mathbb{T}_{var_k}$ l'ensemble des transitions de $H_{var_k}$
10  soit $\mathcal{A}_{var_k}$ la fonction d'affectation de $H_{var_k}$
11  **if** $quali_k = S$ *ou* $quali_k = N$ **then**
12  **forall** $t_l \in T_{pr_i}(s_j)$ **do**
13  ajouter $t = q_{var_k} \xrightarrow{syncS_i t_l, \text{true}} q_{var_k}$ à $\mathbb{T}_{var_k}$
14  ajouter "$var_k := 1$" à l'ensemble $\mathcal{A}_{var_k}(t)$
15  **if** $quali_k = R$ *ou* $quali_k = N$ **then**
16  **forall** $t_l \in T_{post_i}(s_j)$ **do**
17  ajouter $t = q_{var_k} \xrightarrow{syncS_i t_l, \text{true}} q_{var_k}$ à $\mathbb{T}_{var_k}$
18  ajouter "$var_k := 1$" à l'ensemble $\mathcal{A}_{var_k}(t)$
19  **if** $quali_k = P1$ **then**
20  **forall** $t_l \in T_{pr_i}(s_j)$ **do**
21  ajouter $t = q_{var_k} \xrightarrow{syncS_i t_l, \text{true}} q_{var_k}$ à $\mathbb{T}_{var_k}$
22  ajouter "$a_k$" à l'ensemble $\mathcal{A}_{var_k}(t)$
23  **if** $quali_k = P0$ **then**
24  **forall** $t_l \in T_{post_i}(s_j)$ **do**
25  ajouter $t = q_{var_k} \xrightarrow{syncS_i t_l, \text{true}} q_{var_k}$ à $\mathbb{T}_{var_k}$
26  ajouter "$a_k$" à l'ensemble $\mathcal{A}_{var_k}(t)$

La figure 3.12 illustre les résultats de cet algorithme sur les variables A, B et c.

### Module générique du moniteur d'exécution et des interfaces de sorties

Le moniteur d'exécution monotâche périodique assure que chaque bloc fonctionnel du programme d'application est ré-exécuté périodiquement.

La figure 3.13 présente la séquence des activités du moniteur. À chaque début de période, il assure la lecture de la valeur des signaux sur les interfaces d'entrée du contrôleur, puis il exécute séquentiellement et dans un ordre pré-établi chaque bloc fonctionnel du programme (les SFC). Pour finir, il attend la fin de la période courante

pour émettre vers les interfaces de sortie du contrôleur la valeur des sorties calculée dans le cycle.

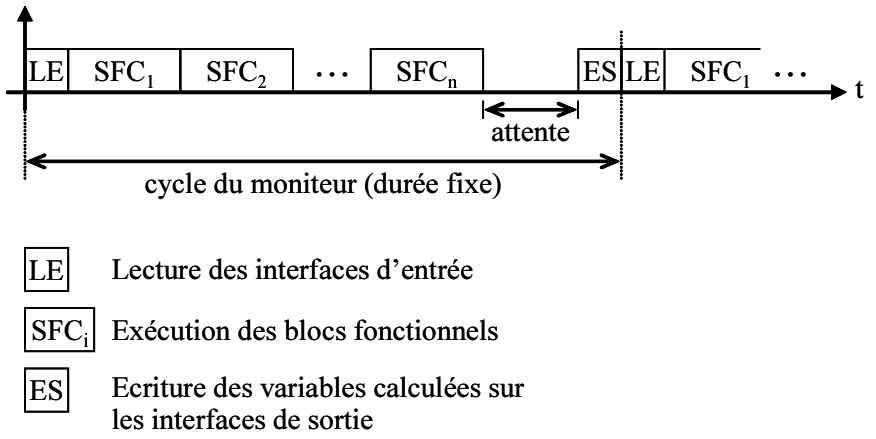



FIG. 3.13 – Cycle du moniteur d'exécution périodique

Le modèle générique que nous proposons pour ce moniteur agit selon le chronogramme de la figure 3.14. En début de cycle, et à temps nul on impose successivement à chaque SFC de franchir une transition ($!syncS_it_j$ ou $!syncS_i$) entraînant par la même occasion l'évolution des variables de sortie, puis une attente de la durée du temps de cycle est modélisée avant d'émettre successivement et à temps nul une demande d'écriture des variables sur les interfaces de sortie ($!syncMow$) et une information en direction du processus que les sorties sont prêtes ($!syncMor$).

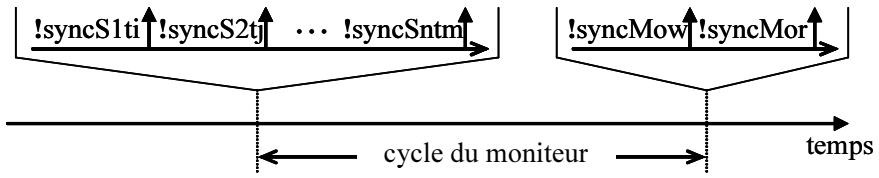



FIG. 3.14 – Chronogramme du modèle générique du moniteur d'exécution périodique

Á chaque cycle du moniteur, les modèles $HIOA_{TS}$ des SFC devront donc franchir une transition. Or, une transition du SFC n'est pas systématiquement franchissable à cause des réceptivités. C'est pour cette raison que, dans le modèle $HIOA_{TS}$ des SFC, nous avons ajouté à chaque situation une transition bouclant sur elle-même pour accepter la synchronisation du moniteur lorsque l'ensemble des réceptivités des transitions en aval de l'étape courante sont fausses (voir section précédente).

La figure 3.15 présente un exemple avec deux blocs fonctionnels $SFC_1$ et $SFC_2$ exécutés par une tâche de période $tc$, et partageant la variable de sortie "A". Le modèle du contrôleur correspondant à cet exemple est donné sur les figures 3.16 et 3.17.

La structure du modèle $HIOA_{TS}$ du moniteur est composée de 2+(nombre de SFC) situations. Toutes les situations sont fugaces sauf « waiting EoC » où, après avoir exécuté tous les SFC à temps nul, l'automate attend que la durée $tc$ du temps de cycle se soit écoulée avant d'émettre successivement $!syncMow$ et $!syncMor$ puis de commencer un nouveau cycle.

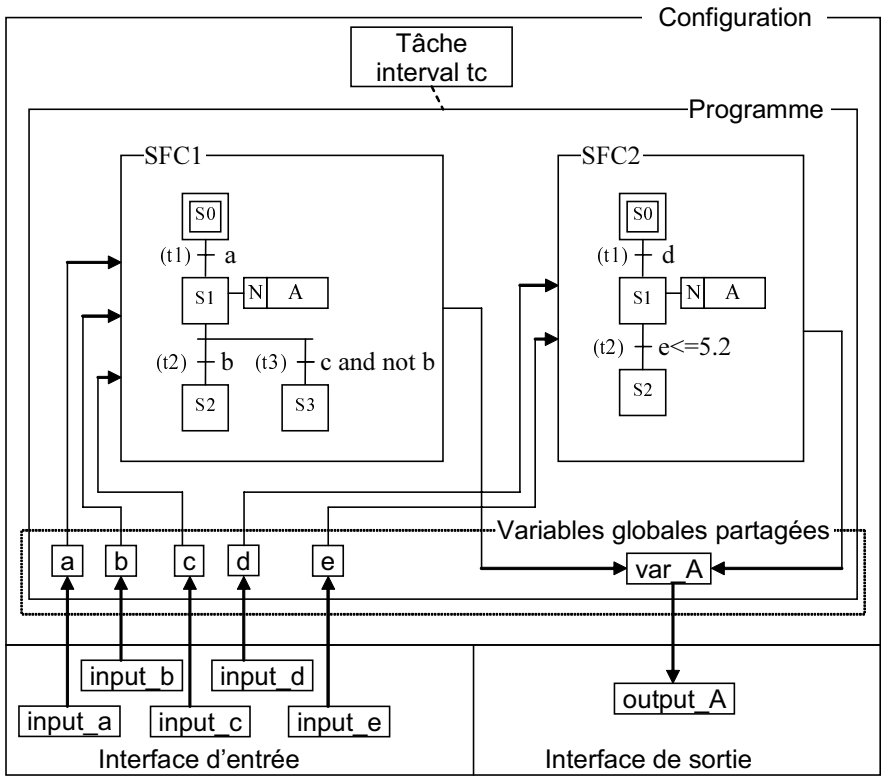



FIG. 3.15 – Exemple d'une configuration d'API avec 2 blocs fonctionnels

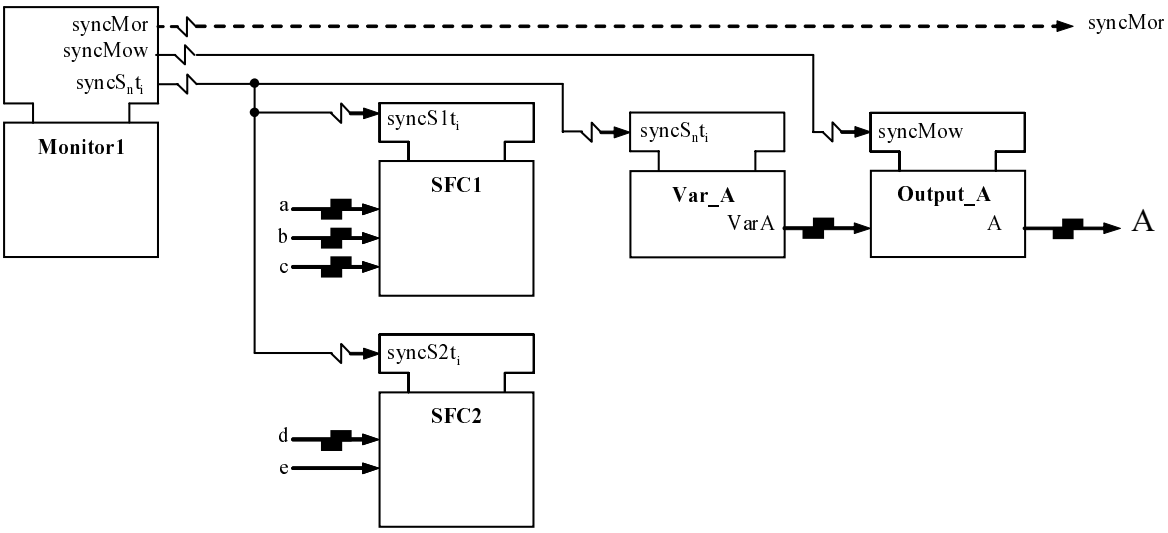



FIG. 3.16 – Réseau d'$HIOA_{TS}$ de l'exemple de la figure 3.15

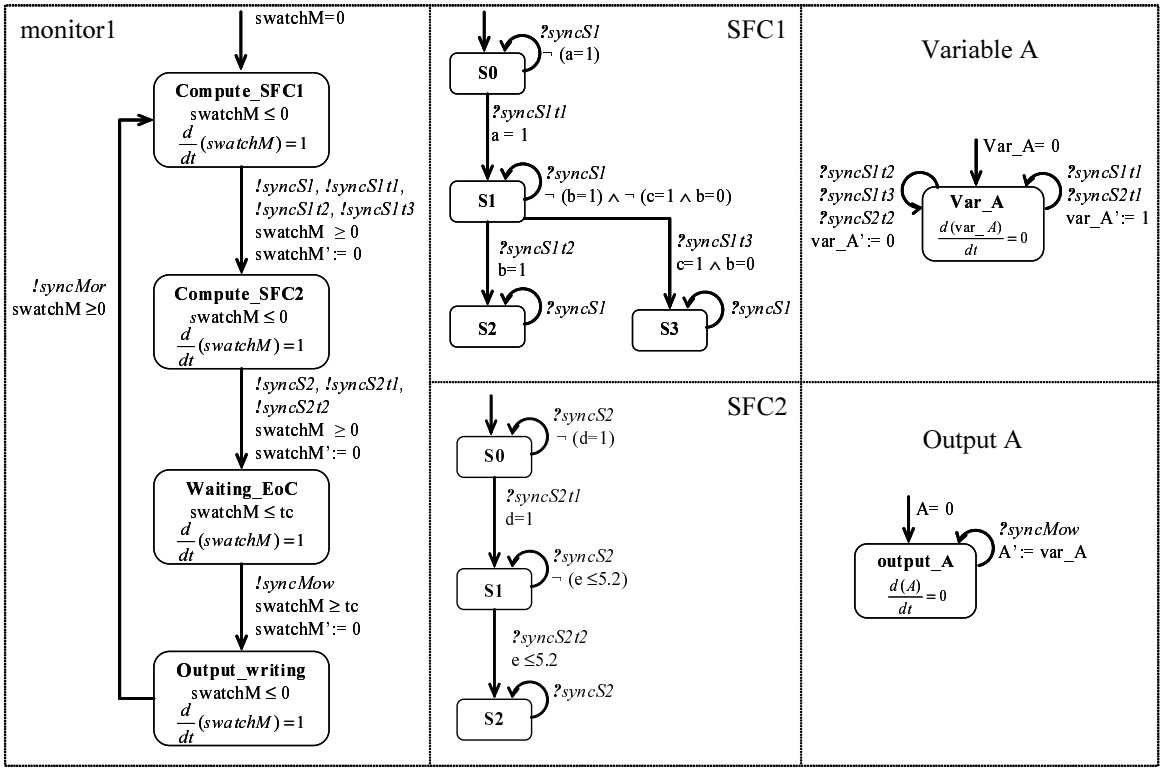



FIG. 3.17 – Modèles $HIOA_{TS}$ pour l'exemple de la figure 3.15

De manière formelle, le modèle $HIOA_{TS}$ d'un moniteur d'exécution périodique (module Monitor1) est défini comme :

$$H_{Monitor1} = \langle \mathbb{Q}, \mathbb{X}, \mathbb{L}, \mathbb{T}, \mathcal{A}, \mathcal{F}, \mathcal{I}, Init, \mathcal{R} \rangle \text{ , avec}$$

si $n$ est le nombre de SFC, $\{H_{SFC_i}\}_{i \in [\![1,n]\!]}$ l'ensemble des $HIOA_{TS}$ modélisant les SFC et $\mathbb{L}_{SFC_i}$ l'ensemble des étiquettes du modèle $H_{SFC_i}$, alors

- $\mathbb{Q} = \{Compute\_SFC_i\}_{i \in [\![1,n]\!]} \cup$ {Waiting_EoC, Output_writing}
- $\mathbb{X} = \mathbb{X}_I \cup \mathbb{X}_L \cup \mathbb{X}_O$ avec $\mathbb{X}_I = \mathbb{X}_O = \emptyset$ et $\mathbb{X}_L = \{swatchM\}$
- $\mathbb{L} = \mathbb{L}_I \cup \mathbb{L}_L \cup \mathbb{L}_O$ avec $\mathbb{L}_I = \mathbb{L}_L = \emptyset$ et $\mathbb{L}_O = (\bigcup_{i=1}^{n} \mathbb{L}_{SFC_i}) \cup \{syncMow, syncMor\}$
- $\mathbb{T} = \bigcup_{\substack{i \in [\![1,n]\!] \\ l \in \mathbb{L}_{SFC_i}}} \{Compute\_SFC_i \xrightarrow{l, swatchM \geq 0} Compute\_SFC_{i+1}\}$
  $\cup \{Waiting\_EoC \xrightarrow{!syncMow, swatchM \geq tc} Output\_writing,$
  $Output\_writing \xrightarrow{!syncMor, swatchM \geq 0} Compute\_SFC1\}$
  (par notation $Compute\_SFC_{n+1} = Waiting\_EoC$)
- $\mathcal{A}$ : $(q \xrightarrow{l,g} q') \rightarrow \begin{cases} \{\text{"swatchM' :=0"}\} & \text{si } q' = \{Output\_writing\} \\ \emptyset & \text{sinon} \end{cases}$
- $\mathcal{F}(q) = \{\frac{d}{dt}(swatchM) = 1\}\ \forall\ q \in \mathbb{Q}$
- $\mathcal{I}(q) = \{swatchM \leq 0\}$ si $q \in \mathbb{Q} = \{Compute\_SFC_i, Output\_writing\}$ et $\mathcal{I}(q) = \{swatchM \leq tc\}$ si $q \in \mathbb{Q} = \{Waiting\_EoC\}$
- $Init = (Compute\_SFC1, \{\text{"swatchM=0"}\})$

- $\mathcal{R}(t) =$" !" $\forall\ t \in \mathbb{T}$

Le modèle des sorties du contrôleur est simple. Son rôle est de recopier la valeur d'une variable, telle qu'elle est à la fin de l'exécution de l'ensemble des SFC, sur l'interface de sortie, et de maintenir cette valeur jusqu'à la fin du cycle suivant. La figure 3.17 montre un exemple de variable de l'interface de sortie "Output_A". La description formelle d'un $HIOA_{TS}$ qui modélise l'interface de sortie de la variable x est la suivante :

$$H_{Output_x} = \langle \mathbb{Q}, \mathbb{X}, \mathbb{L}, \mathbb{T}, \mathcal{A}, \mathcal{F}, \mathcal{I}, Init, \mathcal{R} \rangle \text{ , avec}$$

- $\mathbb{Q} = \{output\_x\}$
- $\mathbb{X} = \mathbb{X}_I \cup \mathbb{X}_L \cup \mathbb{X}_O$ avec $\mathbb{X}_I = \{var\_x\}$, $\mathbb{X}_L = \emptyset$ et $\mathbb{X}_O = \{x\}$
- $\mathbb{L} = \mathbb{L}_I \cup \mathbb{L}_L \cup \mathbb{L}_O$ avec $\mathbb{L}_L = \mathbb{L}_O = \emptyset$ et $\mathbb{L}_I = \{syncMow\}$
- $\mathbb{T} = \{x \xrightarrow{syncMow,true} x\}$
- $\mathcal{A}(x \xrightarrow{syncMow,true} x) =$"$x' := var\_x$"
- $\mathcal{F}(output\_x) = \{\frac{d}{dt}(x) = 0\}$
- $\mathcal{I}(q) =$"$true$" $\forall\ q \in \mathbb{Q}$
- $Init = (output\_x, \{$"x=0"$\})$
- $\mathcal{R}(syncMow) =$" ?"

En résumé, la modélisation du contrôleur inclut le modèle d'un moniteur d'exécution, les modèles des SFC (à l'exception des blocs d'action), les modèles des variables qui sont modifiées par les blocs d'action et le modèle des sorties du contrôleur. Avec cette modélisation du contrôleur, on garantit d'une part les données produites sur les sorties à partir des données d'entrée et d'autre part les dates de production des données : « les bonnes sorties sont produites au bon moment ». Grâce à cette double caractéristique, il sera à la fois possible d'analyser des propriétés quantitatives portant sur la logique de production des données (stratégie de commande) et des propriétés quantitatives portant sur la qualité métrologique des grandeurs contrôlées (performance de réactivité de la commande).

### 3.4.4 Modélisation alternative du contrôleur

Le respect de la logique de production des données de sortie en fonction des données présentes aux entrées est une caractéristique indispensable du modèle du contrôleur. En revanche, il n'est pas systématiquement nécessaire de disposer d'un modèle intégrant le temps de réaction du contrôleur comme dans le module monitor1. Le modèle d'un contrôleur infiniment réactif (c'est-à-dire à temps de cycle nul) est une abstraction suffisante pour toute les analyses n'impliquant que des grandeurs sur lesquelles l'impact du temps de cycle du moniteur est négligeable. Or le calcul de la région atteignable par le modèle nécessite au moins autant d'itérations qu'il y a de cycles du moniteur dans l'horizon temporel de fonctionnement du système analysé. Pour l'analyse de très grands horizons temporels par rapport au temps de cycle du moniteur, le coût de calcul devient pénalisant. Nous proposons donc une modélisation alternative du contrôleur qui n'évolue que sur changement d'état de ses entrées et non plus à chaque cycle du moniteur. En effet, si aucune des entrées du contrôleur n'a évolué, alors la nouvelle exécution du programme conduira à l'émission de la même valeur des sorties. Ce nouveau modèle générique « monitor2 » que nous proposons pour le moniteur agit en respec-

tant l'ordre des activités présentées précédemment figure 3.13 mais sans en respecter les délais de traitement. Le chronogramme de ses activités est présenté figure 3.18.

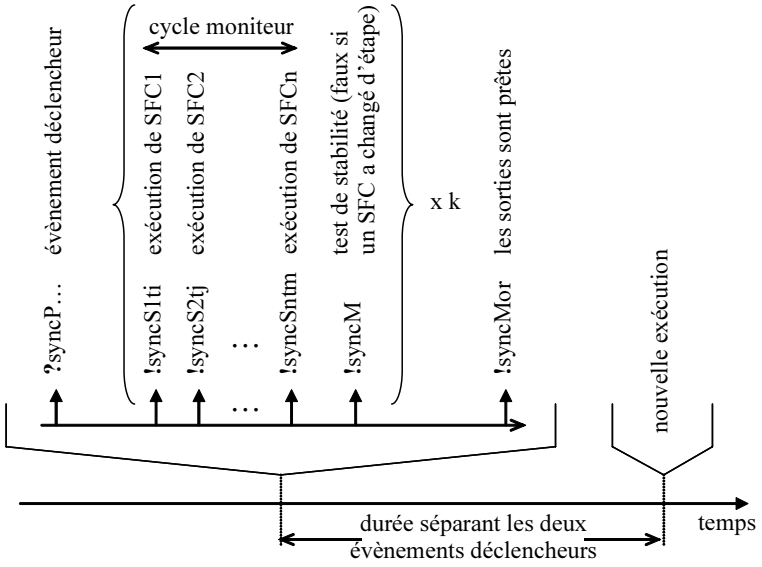



FIG. 3.18 – Chronogramme du modèle générique du moniteur d'exécution périodique « monitor2 »

Une nouvelle exécution du programme de l'API est déclenchée sur occurrence d'un évènement du processus ou sur détection d'un seuil sur une variable d'entrée. Dans les deux cas, il s'agit d'un évènement d'entrée pour le module monitor2. Comme pour monitor1, ce moniteur enchaîne l'exécution des blocs fonctionnels (les SFC). Après exécution d'un bloc fonctionnel ayant modifié la valeur d'une variable partagée, un nouveau cycle d'exécution des blocs fonctionnels doit être entrepris pour propager toutes les conséquences de l'évènement d'entrée. Un test de stabilité qui génère l'évènement interne !*syncM* permet de quitter la boucle d'exécution et d'émettre une information en direction du processus indiquant que les sorties sont prêtes (!*syncMor*). De l'occurrence de l'évènement déclencheur jusqu'à l'émission de !*syncMor*, le temps ne s'est pas écoulé.

À chaque occurrence d'un évènement déclencheur, plusieurs cycles du moniteur sont donc exécutés à temps nul. Le moniteur est ensuite en sommeil jusqu'à l'occurrence d'un nouvel évènement déclencheur. Les modèles $HIOA_{TS}$ des SFC ne sont pas impactés par ce nouveau modèle du moniteur. Nous reprenons l'exemple de la figure 3.15 avec deux blocs fonctionnels $SFC_1$ et $SFC_2$ exécutés par une tâche de période $tc$, et partageant la variable de sortie « A ». Le modèle du contrôleur correspondant à cet exemple, et construit autour d'un moniteur alternatif « monitor2 », est donné sur les figures 3.19 et 3.20.

La structure du modèle $HIOA_{TS}$ du moniteur « monitor2 » est composée d'autant de situations qu'il y a de blocs fonctionnels SFC à exécuter plus 2 situations supplémentaires. Toutes les situations sont fugaces (le temps ne s'y écoule pas) sauf « wait_for_event » où l'automate attend un évènement déclencheur « *SyncP...* ». Trois causes sont à l'origine d'un évènement déclencheur :

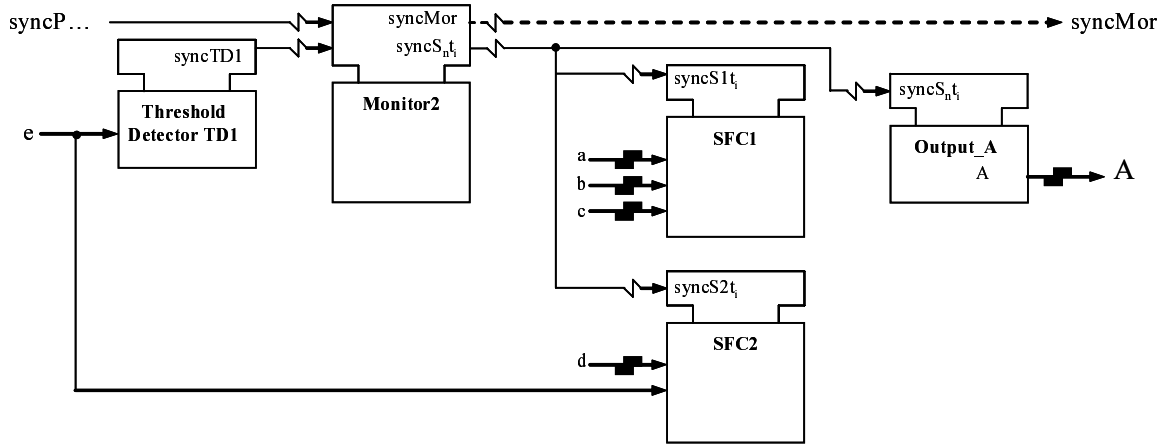



FIG. 3.19 – Réseau d'$HIOA_{TS}$ de l'exemple de la figure 3.15 avec un moniteur « monitor2 »

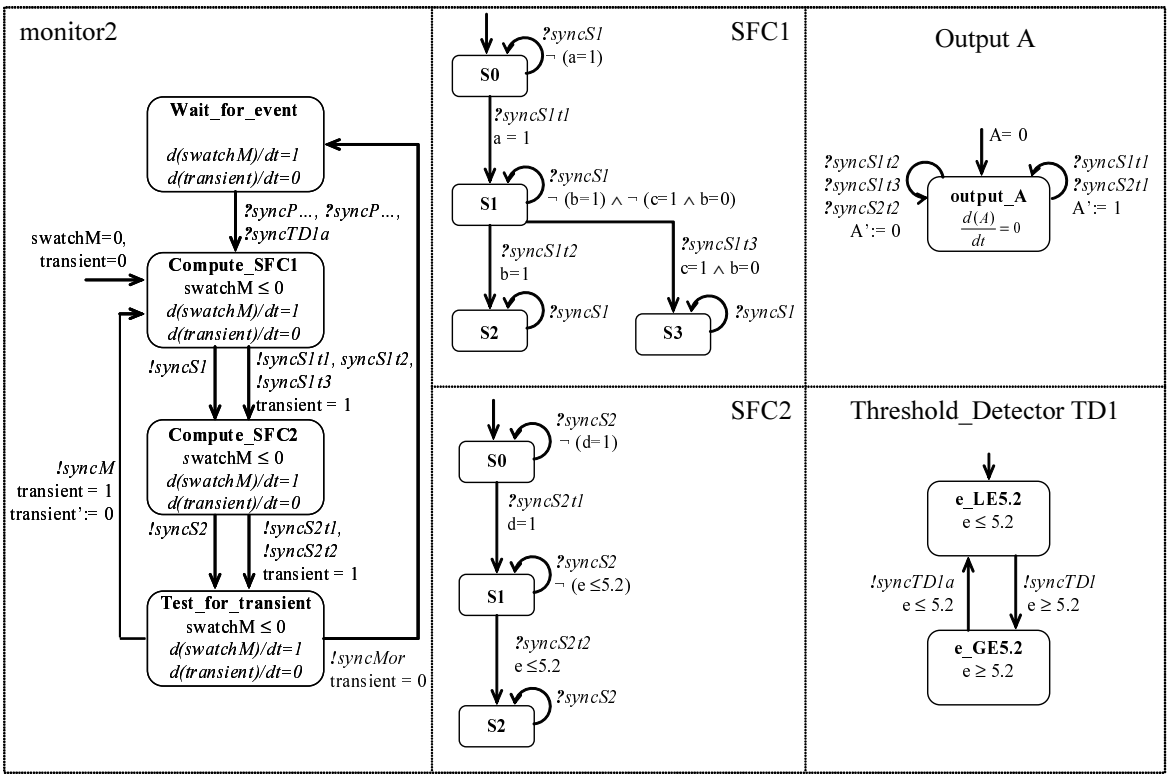



FIG. 3.20 – Modèles $HIOA_{TS}$ pour l'exemple de la figure 3.15 avec un moniteur « monitor2 »

- Le franchissement d'une transition d'un module du processus de type capteur qui modifie de manière discrète la valeur d'une variable d'entrée du contrôleur.
- Le franchissement d'une transition d'un module du processus de type capteur qui se produit à cause de la détection d'un seuil sur une variable à dynamique continue du processus, seuil qui est également utilisé dans une réceptivité d'un SFC.
- Le franchissement d'une transition d'un nouveau type de module du contrôleur, les « threshold_detectors » qui sont chargés de détecter un seuil sur une variable à dynamique continue du processus qui n'est pas détecté par le processus lui-même.

Le module monitor2 dispose d'une variable locale « transient ». C'est une variable logique qui est mise à 1 par le moniteur dès qu'une transition étiquetée *syncSitj* est franchie, c'est-à-dire dès que l'exécution d'un bloc fonctionnel $SFCi$ a conduit à une évolution du SFC. En revanche, le franchissement des transitions étiquetées *syncSi* laisse la variable « transient » à sa valeur courante parce qu'aucune transition n'est franchie dans le bloc fonctionnel $SFCi$ (franchissement d'une transition en boucle sur la situation courante dans le module $SFCi$). Après avoir exécuté tous les blocs fonctionnels $SFCi$ (situations « Compute_SFCi »), le moniteur se trouve dans la situation « Test_for_transient », d'où il détermine si la situation courante du programme est stable ou transitoire. Si $transient = 1$ (transitoire), le moniteur exécute à nouveau les blocs fonctionnels $SFCi$ en franchissant la transition vers « compute_SFC1 », tandis que si $transient = 0$ (stabilité) le moniteur émet le signal $!syncMor$ (sorties prêtes) en franchissant la transition qui le reconduit dans la situation « Wait_for_event ». Dans les deux cas, le moniteur réinitialise la variable « transient » à 0.

L'exécution d'un cycle du moniteur du module « monitor2 » se faisant à temps nul, il n'est plus nécessaire de distinguer temporellement l'écriture d'une variable de sortie par un SFC de l'émission de la sortie du contrôleur en fin de cycle. Ainsi, les modules de type *variable* et de type *output* se fusionnent en un seul et unique module dont le nom est *output* et dont le modèle $HIOA_{TS}$ est celui du module *variable* (voir figure 3.20).

La réceptivité « $e \leq 5.2$ » de la transition (t2) du SFC2 test un seuil sur la variable e. Si cette variable évolue de façon continue dans le modèle du processus et qu'aucun franchissement de transition dans les modules du processus ne matérialise l'évènement $\uparrow (e \leq 5.2)$ pour le moniteur2, alors un nouveau module est ajouté au modèle du contrôleur : « Threshold_Detector_TD1 ». La figure 3.20 montre qu'un module détecteur de seuil est constitué de deux états : « e_LE5.2 » pour lequel $e \in ]-\infty, seuil]$ et « e_GE5.2 » pour lequel $e \in [seuil, +\infty[$. L'évènement $\uparrow (e \leq 5.2)$ est alors matérialisé par le franchissement de la transition étiquetée $!syncTD1a$ qui sert à déclencher l'exécution du moniteur, bien que le module « Threshold_Detector_TD1 » déclenche le moniteur à chaque front montant du prédicat « $e \leq 5.2$ », et qu'à chaque fois que le bloc fonctionnel SFC2 est exécuté (après SFC1), le module SFC1 reste maître du franchissement de sa transition selon qu'elle est ou non validée.

En résumé, la modélisation alternative du contrôleur inclut le modèle d'un moniteur d'exécution (monitor2), les modèles des SFC (à l'exception des blocs d'action), le modèle des sorties du contrôleur qui sont modifiées par les blocs d'action et le modèle des détecteurs de seuil sur variable. L'introduction de modèles de détecteurs de seuil sert à la construction, pour le moniteur monitor2, des évènements associés au franchissement des seuils, le moniteur monitor2 travaillant selon une logique « event driven ».

Avec cette modélisation du contrôleur, on garantit les données produites sur les sorties à partir des données d'entrée mais, cette fois-ci, on ne garantit pas les dates de production des données : « les bonnes sorties sont produites au plus tôt ». Ce contrôleur alternatif a pour ambition de réduire l'effort de calcul des régions atteignables du modèle par réduction du nombre d'itérations de l'algorithme grâce à la réduction du nombre de transitions à franchir pour explorer le modèle.

### 3.4.5 Modélisation des propriétés

Compte tenu du choix d'un outil de model-checking polyèdrique pour automates hybrides, les propriétés s'exprimeront typiquement sous l'une des formes suivantes :

- (région atteignable depuis un état du modèle) $\cap$ (région à tester) $= \emptyset$ ?
- (région à tester) $\subset$ (région atteignable depuis un état du modèle) ?

Lorsque la région à tester est très élaborée, ou qu'elle doit se construire en parallèle de la recherche de la région atteignable, il est d'usage de compléter un modèle par un ou plusieurs automates dits « automates observateurs ». Ainsi, un module $HIOA_{TS}$ observateur aura pour fonction de construire l'information pertinente vis-à-vis des propriétés à vérifier. La seule contrainte à respecter est qu'il soit non invasif (c'est-à-dire qu'il ne doit pas influer sur le comportement des autres modules). La figure 3.21 montre les aspects génériques d'un tel modèle. Sa frontière se limite à des entrées (pour ne rien modifier) : soit des étiquettes de synchronisation de type non bloquant (pour ne pas contraindre), soit des variables discrètes ou continues. Le contenu du module est à spécifier pour chaque observateur et dépend naturellement des propriétés à vérifier.

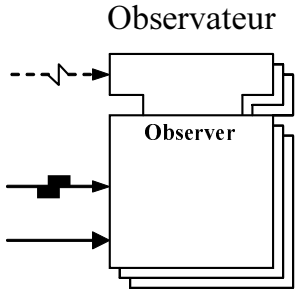



FIG. 3.21 – Aspects génériques d'un modèle d'observateur

De manière formelle, le modèle $HIOA_{TS}$ du module Observateur est défini comme suit :

$$H_{Observateur} = \langle \mathbb{Q}, \mathbb{X}, \mathbb{L}, \mathbb{T}, \mathcal{A}, \mathcal{F}, \mathcal{I}, Init, \mathcal{R} \rangle \text{ , avec}$$

- $\mathbb{X} = \mathbb{X}_I \cup \mathbb{X}_L$ avec $\mathbb{X}_O = \emptyset$
- $\mathbb{L} = \mathbb{L}_I \cup \mathbb{L}_L$ avec $\mathbb{L}_O = \emptyset$ et $\forall\ l \in \mathbb{L}_I,\ \mathcal{R}(l) =$ "¿"

## 3.5 Conclusion du chapitre 3

Dans ce chapitre, nous avons présenté l'aspect méthodologique de nos travaux. Nous avons proposé des modules génériques basés sur les Automates Hybrides à en-

trées/sorties et synchronisations typées afin de construire le réseau d'$HIOA_{TS}$ modélisant le contrôleur logique des systèmes automatisés. Ces modules sont : le programme de contrôle écrit en langage SFC, deux versions du moniteur d'exécution (Monitor1 et Monitor2), deux types différents de sorties discrètes et le détecteur de seuil.

Nous allons maintenant étudier dans le chapitre suivant, au travers de deux études de cas, comment utiliser ce cadre méthodologique pour la modélisation et la vérification de propriétés quantitatives de systèmes réels.

Avec cette bibliothèque de modules, nous avons tenu compte de la technologie des contrôleurs logiques et de leurs interfaces avec le processus. Cela réduira considérablement la difficulté de modélisation de chaque nouveau cas d'étude, en particulier pour le contrôleur, pour lequel tous les aspects ont été couverts de manière générique.

En complément de la bibliothèque, une structure générique du système est proposée sous la forme d'un réseau d'$HIOA_{TS}$. On peut noter que le formalisme $HIOA_{TS}$ défini dans le chapitre 2 s'est révélé très utile, bien sûr à cause de sa modularité, mais principalement pour la souplesse de modélisation qu'il a offerte pour les synchronisations bloquantes et non bloquantes.

# *Chapitre 4*

# *Vérification des propriétés quantitatives : deux cas d'étude*

---

*L'objectif de ce chapitre est la validation de la démarche adoptée pour la vérification formelle de propriétés quantitatives. Ce chapitre est consacré à l'étude de deux exemples avec des objectifs de vérification très différents. La première étude a pour objectif l'analyse de l'impact des caractéristiques temporelles d'un contrôleur logique sur l'exactitude de positionnement d'un axe de déplacement. La deuxième étude a, quant à elle, pour objectif l'analyse de l'impact de la stratégie de commande logique sur les performances d'une production par lots. La modélisation des deux exemples est réalisée par l'instanciation de modules génériques présentés dans le chapitre précédent. Chacune des études comprend une partie dédiée à l'analyse des propriétés quantitatives à vérifier et à la discussion des résultats.*

## 4.1 Présentation des études de cas

Nous allons maintenant illustrer notre approche de vérification des propriétés quantitatives d'une commande discrète au travers du traitement de deux études de cas.

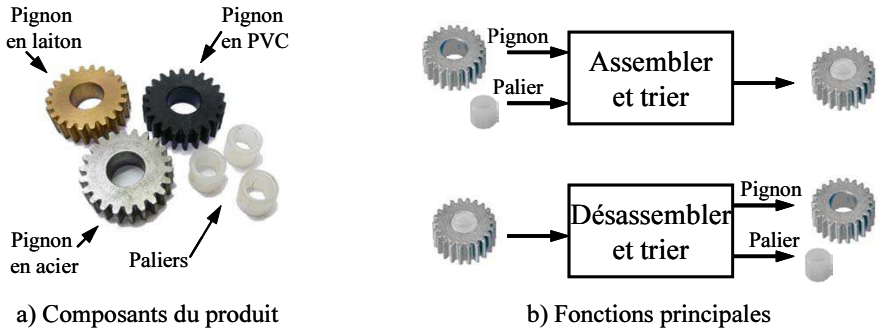
La première étude porte sur l'exactitude d'une grandeur contrôlée par abstraction logique. Elle est menée pour prévoir l'implication du choix d'un contrôleur logique sur l'erreur qui pourra se produire entre la valeur visée de la grandeur contrôlée et la valeur réellement obtenue. Cette première étude s'attache donc à quantifier un aspect qualité métrologique du système étudié [ISO07]. Le support retenu est le contrôle de la position longitudinale d'un convoyeur dans un système d'assemblage/désassemblage du site expérimental du LURPA.

La seconde étude porte sur la performance d'un système de production. Elle est menée pour analyser les impacts du choix d'une stratégie de commande sur la performance du système. Cette seconde étude s'attache donc à quantifier des indicateurs de performance de type coût, délai et valeur (liée à la satisfaction du client) du système de production [Tah03]. Le support retenu est une installation de production de lots d'eau salée du Laboratoire de « Process control » de l'université de Dortmund [KER+99].

## 4.2 Etude de cas 1 : Axe de déplacement

### 4.2.1 Présentation de l'étude de cas

Le système auquel nous allons nous intéresser est un système de tri et d'assemblage/désassemblage de paliers dans des pignons. La figure 4.1 présente les différents composants du produit et illustre les fonctions principales du système. Pour assurer ces fonctions, ce dernier est constitué de quatre stations (figure 4.2a). Les quatre stations ont respectivement pour fonction le déstockage des pignons en entrée de ligne (station 1), l'identification du matériau du pignon et de la présence d'un éventuel palier (station 2), la pose ou la dépose d'un palier dans le pignon (station 3), et enfin leur tri, en fonction du matériau, en vue de leur évacuation (station 4). L'ensemble de cette ligne d'assemblage est contrôlé par un Automate Programmable Industriel (API).




a) Composants du produit b) Fonctions principales

FIG. 4.1 – Composants du produit et principales fonctions du système d'assemblage/désassemblage de paliers dans des pignons

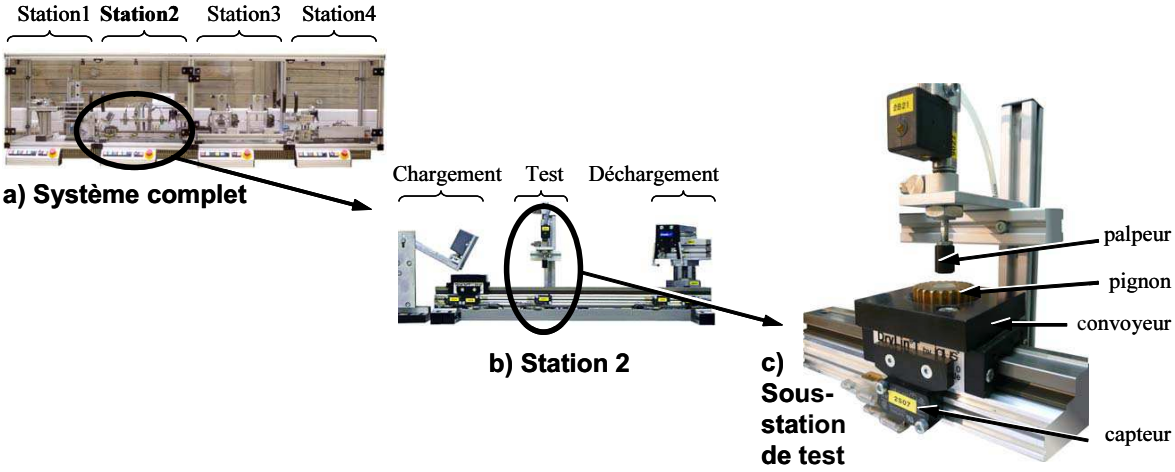



FIG. 4.2 – Ligne d'assemblage/désassemblage de pignons

La commande est purement discrète. En effet, comme c'est le plus souvent le cas pour ce type de système, la recherche de coûts compétitifs conduit à éviter au maximum les commandes asservies au profit de commandes Tout ou Rien. Toutes les grandeurs physiques continues du processus à contrôler (pour l'essentiel des positions) sont donc observées et commandées au travers de leurs abstractions discrètes. Au total, ce sont 82 entrées et 50 sorties logiques qui sont gérées par l'API. Pour la plupart des grandeurs physiques, la qualité requise est directement obtenue par le processus. Par exemple, la position de plusieurs solides en mouvement est obtenue à l'aide de butées mécaniques utilisées en complément des actionneurs de type vérin. Il n'en demeure pas moins que l'exactitude de positionnement de certains solides reste dévolue à la commande, en particulier pour ceux actionnés par des moteurs électriques.

L'objectif de l'étude que nous allons conduire est de vérifier si la qualité des positionnements obtenus par une telle commande discrète est compatible avec les exigences de précision requises par le procédé. Cette notion de précision n'a de sens que relativement au positionnement des objets manipulés par un axe de déplacement ou par un ensemble d'axes combinés. Il n'est donc pas nécessaire de prendre en compte la totalité du modèle de la commande ni du processus pour vérifier la qualité des positionnements, mais uniquement le sous-ensemble qui concerne la commande de l'axe (ou de l'ensemble des axes) impliqué(s) dans le déplacement étudié.

Pour cette étude de cas, nous allons nous concentrer sur un seul mouvement de solide, sachant que l'étude des autres mouvements se ferait de façon similaire. Le mouvement choisi est celui du convoyeur de la station 2 (fig. 4.2b) qui transporte, depuis une position de chargement, un produit (pignon ou pignon avec palier) pour l'arrêter à une sous-station de test (fig. 4.2c) puis, après le test, qui transporte le produit testé vers une sous-station de déchargement. Le mouvement de translation rectiligne est obtenu par un moto-réducteur rotatif et une transmission par courroie crantée. Deux pré-actionneurs de type "contacteur électrique" assurent une distribution de l'énergie électrique au moteur, permettant un mouvement dans les deux directions. Les deux arrêts de "fin de course" (chargement et déchargement) sont obtenus par des "butées électriques" (interrupteurs de position) qui coupent directement, par un câblage électrique, la distribution d'énergie pour la direction considérée du mouvement. Cela

assure une grande qualité du positionnement du convoyeur tout en permettant, par la commande du mouvement dans l'autre direction, de faire repartir le chariot. En revanche, la position "à mi-course" de la sous-station de test ne permet pas le recours à cette technologie : l'arrêt du convoyeur sera ici obtenu par un arrêt commandé par le contrôleur après réaction à un évènement issu d'un capteur.

La figure 4.3 décrit le déroulement d'une séquence de test de présence d'un palier. Ce test est réalisé par un palpeur cylindrique de diamètre $\phi 10mm$ qui plonge dans l'alésage $\phi 12mm$ du pignon. Sans la présence d'un palier, le palpeur atteint la fin de course basse de son mouvement vertical, tandis qu'avec un palier, l'alésage intérieur $\phi 8mm$ de ce dernier limite la course du palpeur qui s'arrête en butée sur la face supérieure du palier. Une erreur trop importante sur la position d'arrêt du convoyeur peut donc conduire à un résultat incorrect du test (en concluant à la présence d'un palier qui n'existe pas) qui, par la suite, peut conduire à des dysfonctionnements, voire à des incidents dommageables au système d'assemblage. La figure 4.4 illustre les positions extrêmes du pignon qui permettent un test correct.

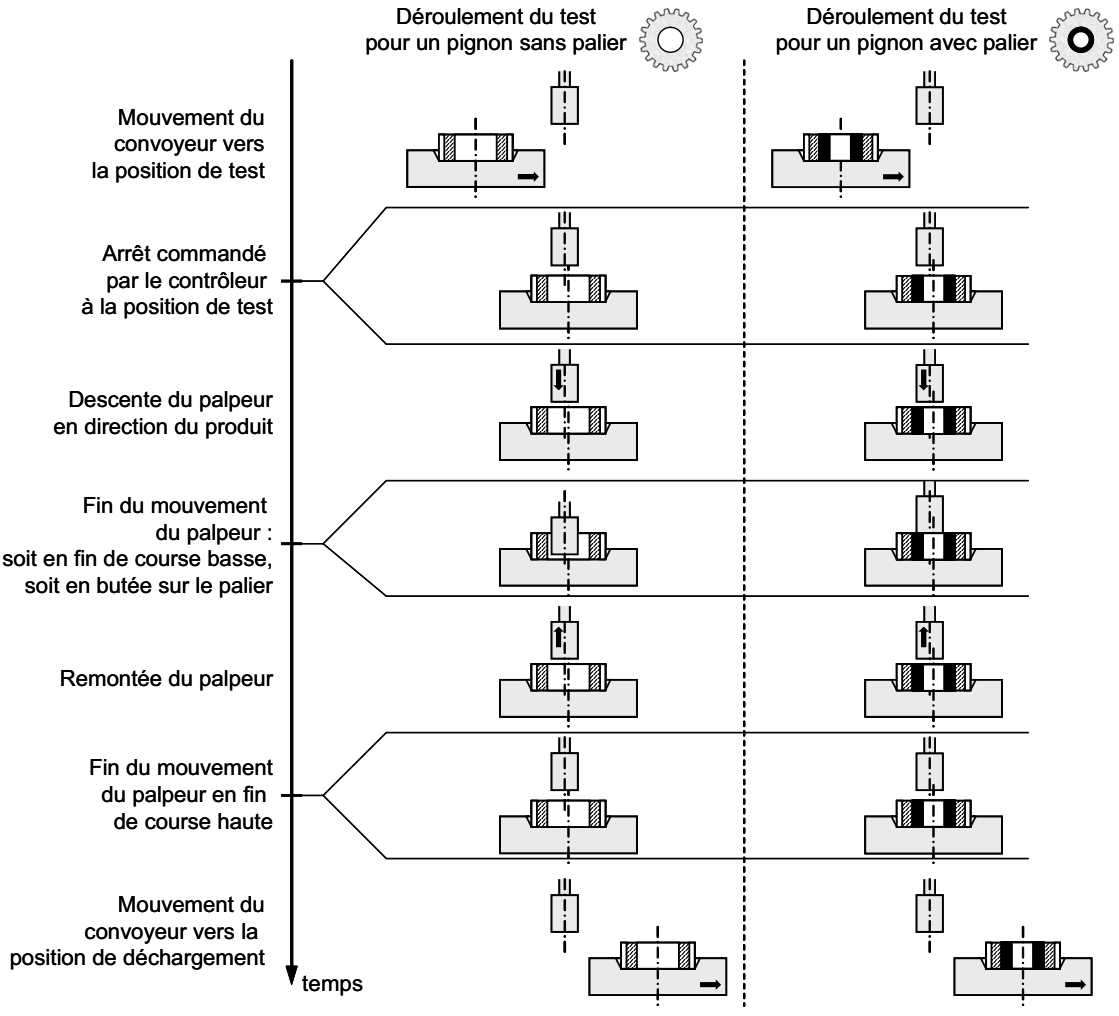



FIG. 4.3 – Séquence de test de présence d'un palier

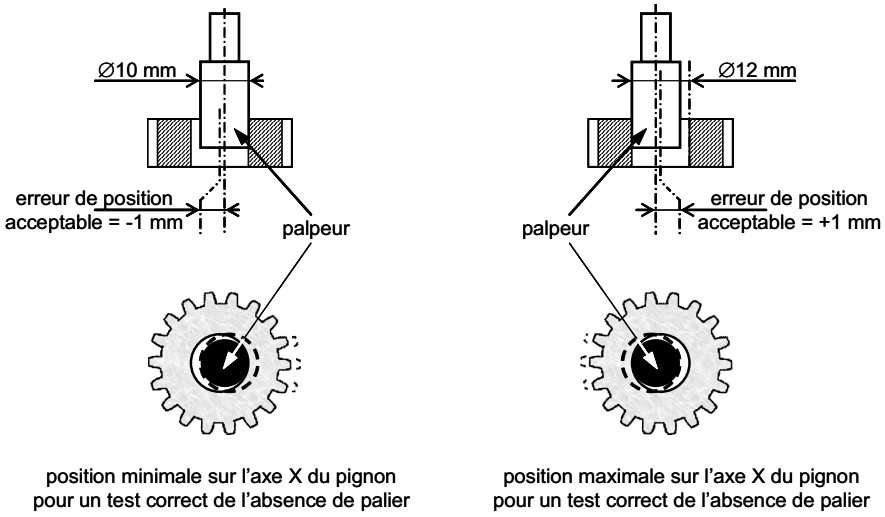



FIG. 4.4 – Positions de test du pignon

### 4.2.2 Présentation de la commande

Dans la suite de cette étude, nous allons uniquement nous intéresser à l'axe de déplacement de la station 2 dont les seules informations pertinentes pour la commande du déplacement sont données sur la figure 4.5.

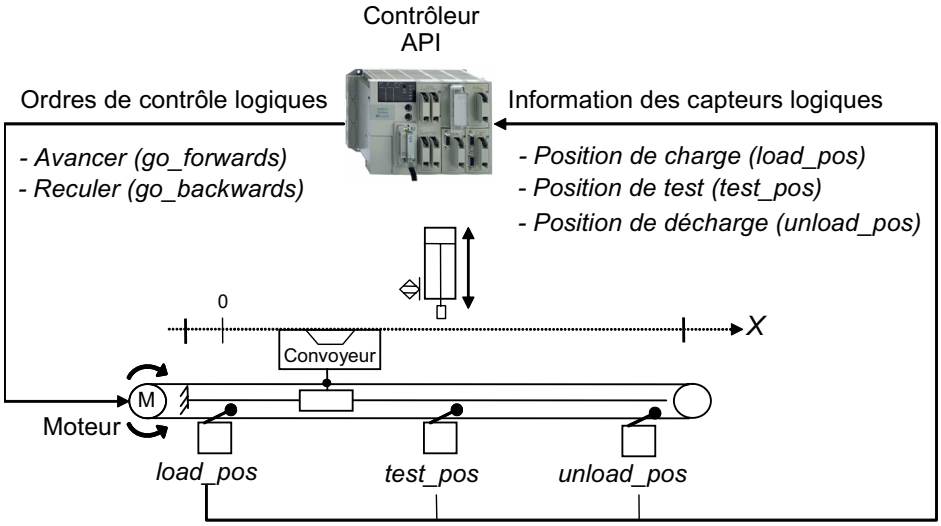



FIG. 4.5 – Axe de déplacement de la station 2

Le pignon se trouve initialement à la sous-station de chargement (load_pos) où il a été déposé sur le convoyeur. Une commande de rotation du moteur dans le sens horaire (go_forward) amène le convoyeur droit vers la sous-station de test (test_pos). Pour que le test se déroule dans de bonnes conditions, l'erreur de position d'arrêt doit être inférieure à 1 mm. La position du convoyeur est notée X. Le test de présence d'un palier une fois terminé, une nouvelle commande de rotation du moteur dans le sens horaire amène le pignon en fin de course droite où il est déchargé (unload_pos). Une rotation du moteur dans le sens anti-horaire (go_backward) permet le retour du convoyeur en position de chargement où un nouveau cycle de test peut démarrer.

Le modèle du programme de contrôle, écrit en syntaxe SFC, qui traduit le comportement du processus décrit précédemment est montré dans la figure 4.6. C'est *SFC1* qui commande les déplacements du convoyeur pour amener les pignons sous les différentes sous-stations de travail (chargement, test et déchargement des pignons) tandis que les opérations de chacun de ces postes sont décrites par SFC2, SFC3 et SFC4, respectivement. Ainsi, le moteur ne démarre pas (étapes $SFC1.S0$, $SFC1.S2$ et $SFC1.S4$) tant que l'opération de chaque poste n'est pas terminée (étapes $SFC2.S10$, $SFC3.S20$ et $SFC4.S30$).

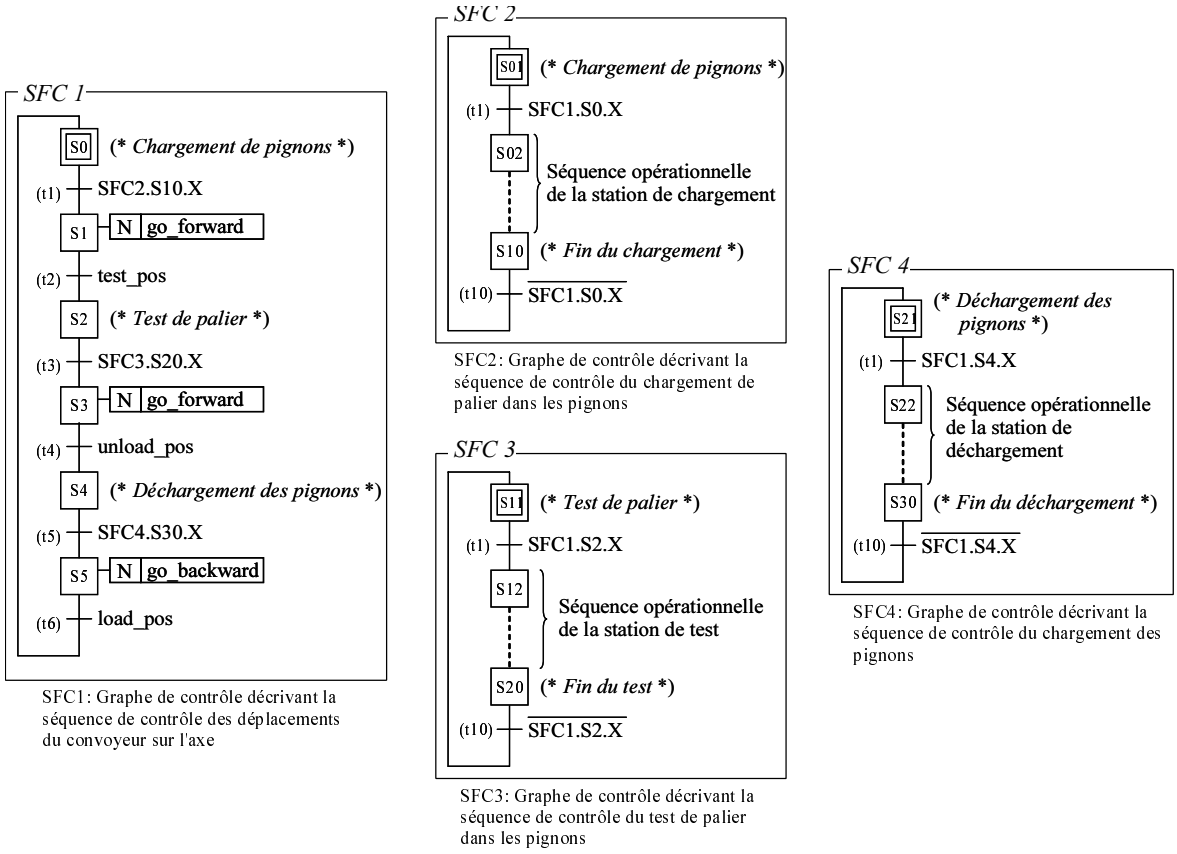



FIG. 4.6 – Commande de l'axe de déplacement de la station 2

Les SFC se coordonnent selon une structure classique d'appel/réponse à l'aide des variables d'étapes, $SFC_i.S_j.X$ désignant la variable logique représentant l'activité de l'étape $S_j$ du graphe $SFC_i$. Ces quatre $SFC$ sont implantés dans un API sous la forme de quatre blocs fonctionnels regroupés dans un programme qui enchaîne leurs exécutions dans l'ordre suivant : $SFC1$, $SFC2$, $SFC3$ puis $SFC4$. Le programme est lui-même lancé par une tâche périodique pour chercher à obtenir une bonne répétabilité du positionnement du convoyeur. Compte tenu de la taille du système et de l'automate choisi, le concepteur estime pouvoir garantir une période d'exécution de la tâche de $12ms$. En dessous de cette valeur, il sera peut-être difficile d'exécuter tout le programme à chaque cycle.

### Présentation du processus

La motorisation de l'axe est assurée par un moto-réducteur qui impose au convoyeur une vitesse moyenne de 200 mm/s. Les contacteurs $KM\_forwards$ et $KM\_backwards$ ne sont pas directement pilotés par les sorties de l'API « go_forwards » et « go_backwards », un câblage électrique les assujettit également aux butées électriques de fin de course « el_load » et « el_unload ». Le schéma électrique de puissance et de commande du moteur est présenté figure 4.7. Ainsi, lorsque le convoyeur se déplace dans le sens des $x$ positifs, il est arrêté lorsque la commande « go_forwards » s'arrête ou bien lorsque la butée électrique « el_unload » est atteinte.

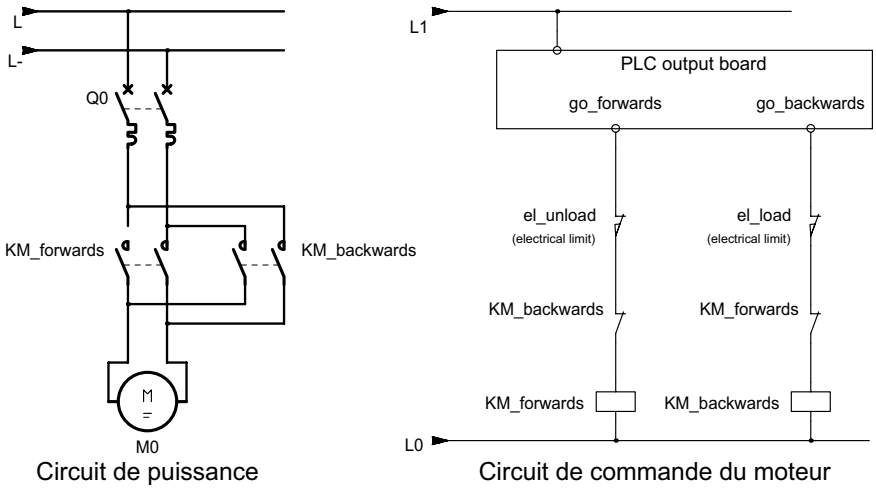



FIG. 4.7 – Schéma électrique de puissance et de commande du moteur de l'axe

La figure 4.8 présente la position des 3 sous-stations (chargement à $x = 0$ mm, test à $x = 120$ mm, déchargement à $x = 320$ mm) et l'implantation des 3 capteurs correspondant.

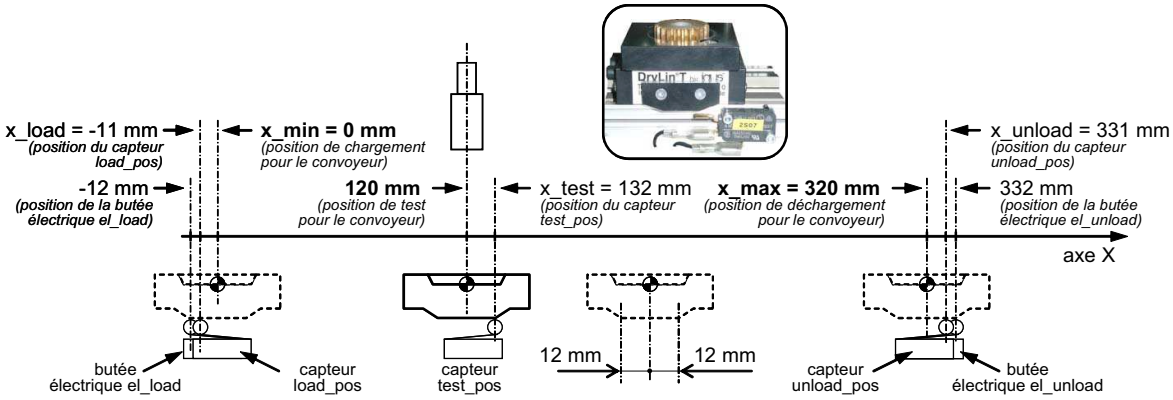



FIG. 4.8 – Positions des trois sous-stations et des trois capteurs

Sur le convoyeur est fixée une came pour que les capteurs puissent être accostés à l'aide d'une rampe (spécification du constructeur des capteurs). Compte tenu de la dimension de la rampe (12 mm de part et d'autre de l'axe de l'empreinte qui accueille les pignons), la position des capteurs est décalée par rapport aux positions visées pour

l'axe des pignons. Le sens de ce décalage dépend de la direction d'accostage. Le capteur « test_pos » est ainsi implanté à la position x_test = 132 mm.

### 4.2.3 Propriétés quantitatives recherchées assurant la faisabilité de la commande logique

Comme nous l'avons déjà mentionné, le choix d'une commande discrète de déplacement, même s'il convient fréquemment aux exigences de qualité de positionnement des mobiles, est une abstraction, nécessairement réductrice, des grandeurs physiques qui lui sont associées. Ainsi, la commande Tout ou Rien d'un déplacement linéaire vers une position observée grâce à un détecteur de position ne permet naturellement pas de connaître l'erreur de positionnement du mobile. En effet, l'ensemble des grandeurs observées et commandées sont des abstractions discrètes de leurs grandeurs physiques.

Nous allons faire une étude de l'exactitude du positionnement du convoyeur à la sous-station de test, sachant que :

- les dimensions du palpeur ($\phi$10 mm) et de l'alésage du pignon ($\phi$12 mm) autorisent une erreur de position relative de leurs axes de $\pm$1 mm,
- la position de l'axe du palpeur est x = 120 mm,
- la position pré-réglée du capteur « test_pos » est x_test = 132 mm,
- la période du cycle d'exécution du programme par l'API est de 12 ms,
- la vitesse de translation du convoyeur est de 200 mm/s.

Le test nécessite pour son déroulement correct que la position d'arrêt du convoyeur, obtenue lorsque l'étape $S2$ de $SFC1$ est active, soit comprise entre $(120 - 1)$mm et $(120 + 1)$mm. Autrement dit que :

$$(SFC1.S2.X = 1) \wedge (\text{Axe dans la situation stop}) \Rightarrow 119 \text{ mm} \leq x_{convoyeur} \leq 121 \text{ mm}$$

On considère qu'aussi bien le contrôleur que le processus subissent certaines perturbations liées à la durée du cycle du moniteur d'exécution pour le premier, et à la vitesse du moteur qui amène le convoyeur à la sous-station de test pour le second. Des mesures effectuées dans le cadre de la thèse de Pascal Meunier [Meu06] ont montré que la période d'exécution d'un programme dans un API admet une gigue typique de $\pm$0,5 %. Concernant la vitesse de déplacement du convoyeur, quelques mesures du temps de parcours entre les capteurs « load_pos » et « test_pos » ont montré que la vitesse reste à $\pm 0,1$ % de la vitesse nominale. Cette très faible amplitude de la variation de vitesse moyenne s'explique par le fait qu'il s'agit de la vitesse moyenne sur le même parcours de 120 mm. Il est probable que la vitesse instantanée puisse être soumise à de plus amples variations.

En résumé, la faisabilité de la commande par un contrôleur logique pour :
- tc = 12 ms $\pm$ 0,5 % (temps de cycle API),
- $\frac{dx}{dt}$ = 200 mm/s $\pm$ 0,1 % (vitesse du convoyeur en mouvement),
- x_test = 132 mm (position de pré-réglage du capteur « test_pos »)

sera assurée si :

$$(SFC1.S2.X = 1 \wedge \text{Axe.stop}) \Rightarrow x \in [119, 121], x \text{ étant exprimé en mm}$$

### 4.2.4 Modélisation du système

La structure du modèle présenté figure 4.9 reprend les principes de modélisation définis dans la section 3.4.1. Le contrôleur est modélisé par quinze modules $HIOA_{TS}$ tandis que quatre modules $HIOA_{TS}$ modélisent le processus.

Pour le contrôleur, on retrouve le moniteur d'exécution (Monitor1), un module pour chacun des quatre SFC (SFC1, SFC2, SFC3, SFC4), un module pour chacune des variables d'étape qui sont partagées par les SFC (Variable1, Variable2, Variable3, Variable4, Variable5, Variable6), un module pour chacune des variables de sortie qui peuvent être partagées par les SFC (Variable7, Variable8) et un module pour chacune des sorties (Output1, Output2).

Pour le processus, l'actionneur (en fait, le moto-réducteur avec ses deux contacteurs) n'a pas été modélisé par un module autonome, mais il a été regroupé avec le convoyeur dans un seul module nommé Axe. Chacun des 3 capteurs a quant à lui été modélisé par son propre module obtenu par instanciation du modèle générique proposé dans le chapitre 2.

L'interaction entre le processus et le contrôleur est réalisée par les variables logiques qui informent sur la position du convoyeur devant les différentes sous-stations (load_pos, test_pos et unload_pos). À son tour, le contrôleur émet ses ordre au processus par les sorties logiques (go_forward et go_backward). L'émission des sorties est mise à jour à chaque fin de cycle de l'API, modélisée dans le module du moniteur Monitor1 par l'évènement *Mor*.

Maintenant, regardons plus en détail les modèles de comportement associés à chaque module et leurs interactions.

#### Modélisation par $HIOA_{TS}$ du contrôleur

La modélisation du contrôleur consiste en la traduction des quatre SFC puis en la génération du moniteur associé (voir section 3.4.3). Chaque SFC est traduit par un module $HIOA_{TS}$ qui décrit le séquencement des situations et par un ensemble de modules pour les variables partagées que le SFC modifie (voir section 3.4.3).

Dans le module SFC1, présenté figure 4.38, on retrouve une situation pour toutes les étapes de SFC1 (fig. 4.6). Les deux sorties go_forward et go_backward sont traitées dans les modules variable7 et variable8. Par exemple, lorsque le module SFC1 franchit la transition allant de S0 à S1 (étiquette *?syncS1t1*), alors le module variable7 franchit la transition étiquetée *?syncS1t1* et la variable v_go_forward est mise à 1 conformément au bloc d'action associé à l'étape S1 de SFC1. L'état d'une étape est considéré comme une variable de sortie lorsque cette étape est utilisée par d'autres SFC, par exemple dans le cadre d'un appel/réponse entre SFC. Ainsi, les modules variable1, variable 2 et variable3 modélisent respectivement l'état des étapes S0, S2 et S4 de SFC1.

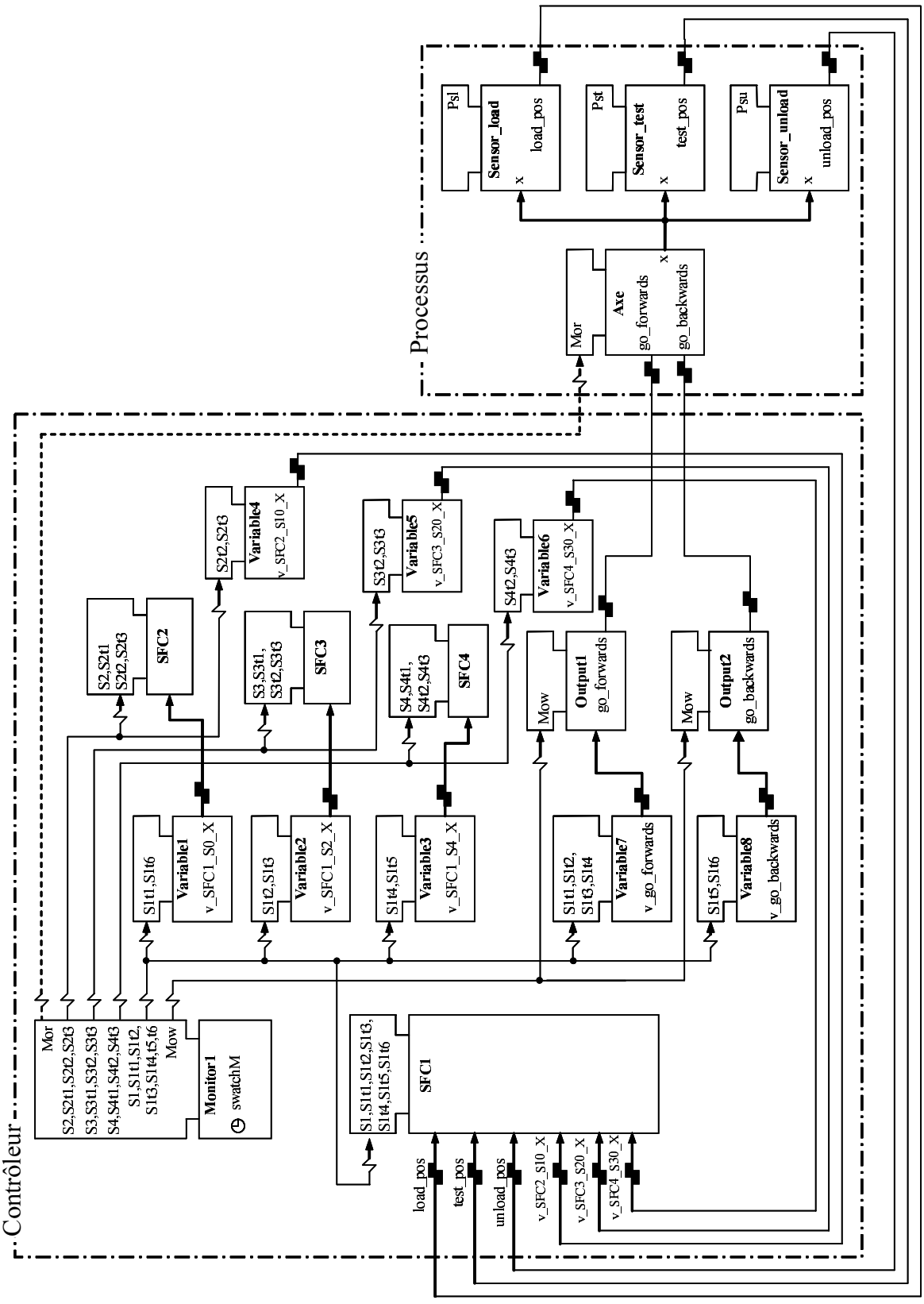
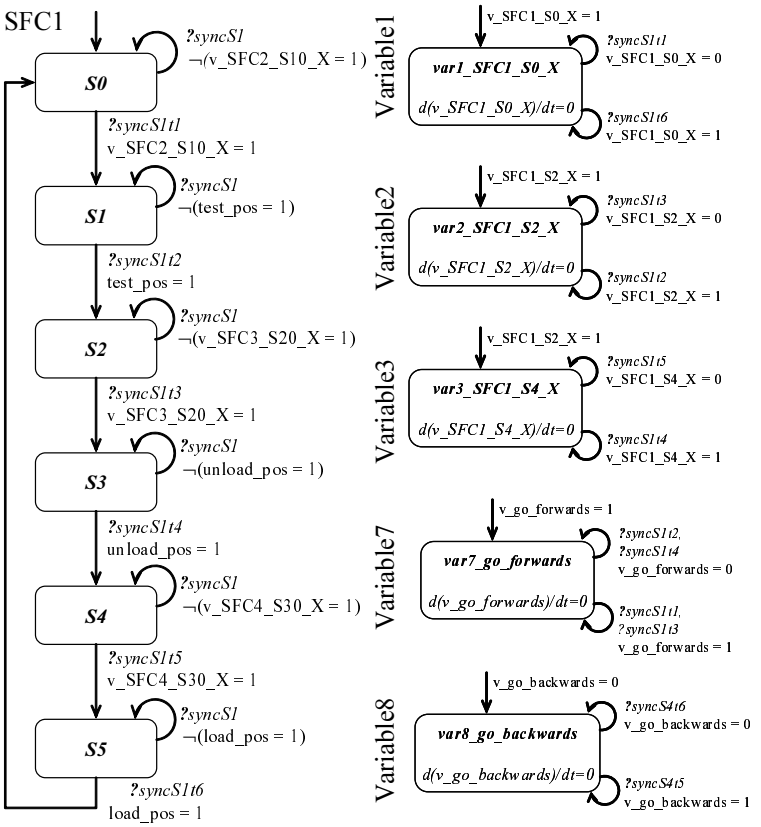



FIG. 4.9 – Modélisation modulaire pour le système de l'axe de déplacement




SFC1

| Variables d'entrée | Etiquettes d'entrée |
|---|---|
| v_SFC2_S10_X<br>v_SFC3_S20_X<br>v_SFC4_S30_X<br>load_pos<br>test_pos<br>unload_pos | *syncS1,*<br>*syncS1t1, syncS1t2,*<br>*syncS1t3, syncS1t4,*<br>*syncS1t5, syncS1t6* |

Variable1

| Variables de sortie | Etiquettes d'entrée |
|---|---|
| v_SFC1_S0_X | *syncS1t1, syncS1t6* |

Variable2

| Variables de sortie | Etiquettes d'entrée |
|---|---|
| v_SFC1_S2_X | *syncS1t2, syncS1t3* |

Variable3

| Variables de sortie | Etiquettes d'entrée |
|---|---|
| v_SFC1_S4_X | *syncS1t4, syncS1t5* |

Variable7

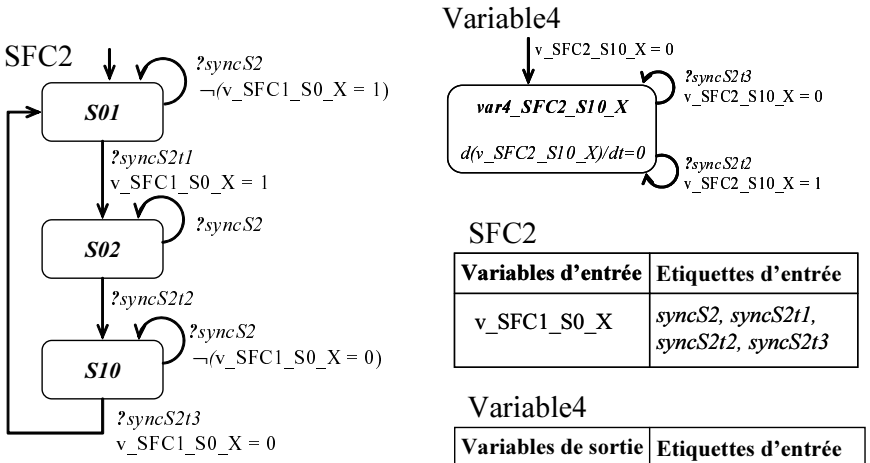
| Variables de sortie | Etiquettes d'entrée |
|---|---|
| v_go_forwards | *syncS1t1, syncS1t2,*<br>*syncS1t3, syncS1t4* |

Variable8

| Variables de sortie | Etiquettes d'entrée |
|---|---|
| v_go_backwards | *syncS1t5, syncS1t6* |

FIG. 4.10 – Modèle $HIOA_{TS}$ du programme SFC1 avec les variables qu'il génère

Dans le module SFC2, présenté figure 4.11, on retrouve une situation pour les étapes S01 et S10 de SFC2 (fig. 4.6). Le reste de la séquence entre ces deux étapes a été modélisé par une unique situation notée S02 ; en effet, le détail des actions nécessaires au chargement d'un produit sur le convoyeur n'a aucune influence sur le positionnement de ce dernier. L'état de l'étape S10 est modélisé par le module variable4 car il est partagé avec SFC1, qui en lit la valeur.




SFC2

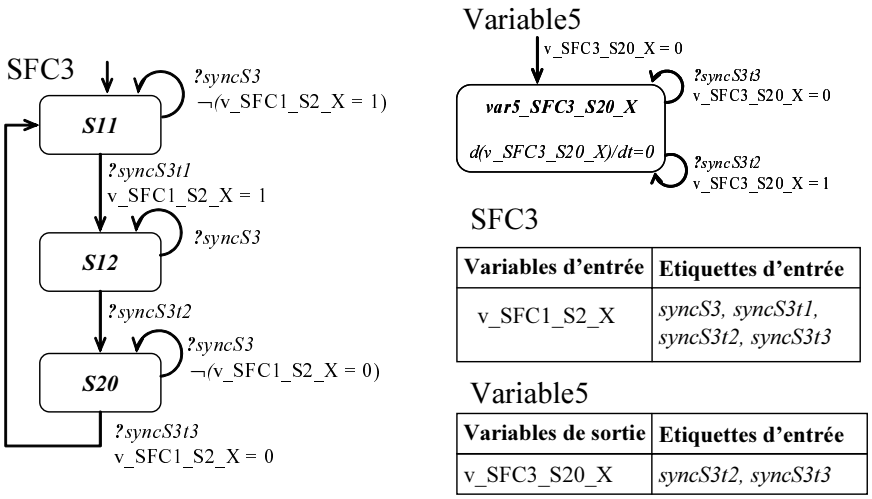
| Variables d'entrée | Etiquettes d'entrée |
|---|---|
| v_SFC1_S0_X | *syncS2, syncS2t1,*<br>*syncS2t2, syncS2t3* |

Variable4

| Variables de sortie | Etiquettes d'entrée |
|---|---|
| v_SFC2_S10_X | *syncS2t2, syncS2t3* |

FIG. 4.11 – Modèle $HIOA_{TS}$ du programme SFC2 avec la variable qu'il génère

Les modules SFC3 et SFC4, présentés figures 4.12 et 4.13, sont construits de la même façon que le module SFC2. On y retrouve les situations S12 et S22 qui regroupent en une seule situation l'ensemble des actions effectuées au niveau de la sous-station correspondante. L'état de l'étape S20 et l'état de l'étape S30 sont modélisés par les modules variable5 et variable6 pour être lus par SFC1.



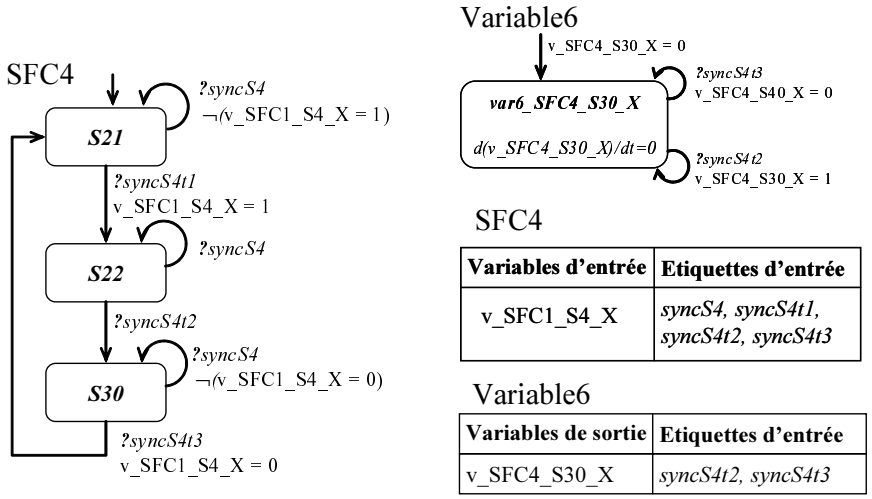

SFC3

| Variables d'entrée | Etiquettes d'entrée |
|---|---|
| v_SFC1_S2_X | *syncS3, syncS3t1, syncS3t2, syncS3t3* |

Variable5

| Variables de sortie | Etiquettes d'entrée |
|---|---|
| v_SFC3_S20_X | *syncS3t2, syncS3t3* |

FIG. 4.12 – Modèle $HIOA_{TS}$ du programme SFC3 avec la variable qu'il génère




SFC4

| Variables d'entrée | Etiquettes d'entrée |
|---|---|
| v_SFC1_S4_X | *syncS4, syncS4t1, syncS4t2, syncS4t3* |

Variable6

| Variables de sortie | Etiquettes d'entrée |
|---|---|
| v_SFC4_S30_X | *syncS4t2, syncS4t3* |

FIG. 4.13 – Modèle $HIOA_{TS}$ du programme SFC4 avec la variable qu'il génère

Le module du moniteur est obtenu par instanciation selon la procédure exposée section 3.4.3, et le résultat de l'instanciation est présenté figure 4.14. Ce module assure le lancement successif de l'exécution de SFC1, SFC2, SFC3 puis SFC4. Ensuite, dans la situation waiting_EoC, il attend la fin du cycle en cours. Par rapport au modèle générique présenté dans la section 3.4.3, une gigue sur la période du moniteur a été ajoutée. Ainsi l'invariant de la situation devient $swatchM \leq tc + (tc * ptcjitter)$ à la place de $swatchM = tc$, et la garde de la transition qui permet de quitter la situation waiting_EoC devient $swatchM \geq tc - (tc * ptcjitter)$ à la place de $swatchM = tc$.

Ainsi, la durée d'un cycle du moniteur sera comprise entre $tc - (tc * ptcjitter)$ et $tc + (tc * ptcjitter)$, où tc est le temps de cycle nominal du moniteur et ptcjitter est l'écart exprimé en pourcentage du temps de cycle nominal. Ainsi le temps d'attente dans la situation waiting_EoC est égal à $tc \pm ptcjitter$ %. Ensuite, conformément au modèle générique du moniteur, le franchissement de la transition de waiting_EoC vers output_writing assure la mise à jour des sorties de l'API (étiquette *!syncMow*), puis le processus en est informé par le franchissement de la transition issue de la situation output_writing qui émet la synchronisation *!syncMor* (sorties prêtes). Dans le module Monitor1, tc et ptcjitter sont des constantes.

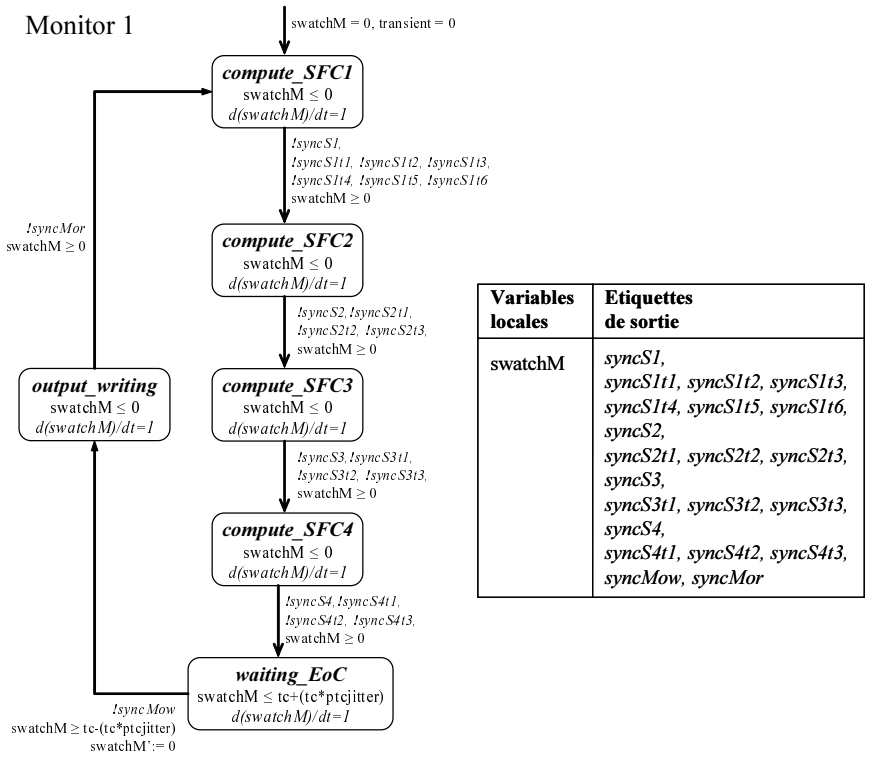
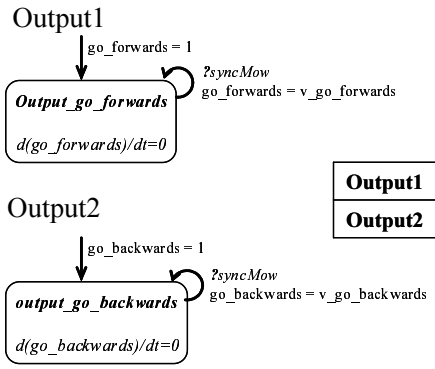



FIG. 4.14 – Modèle $HIOA_{TS}$ du moniteur d'exécution Monitor1

Pour en finir avec la modélisation du contrôleur, les sorties go_forwards et go_backwards sont modélisées par les modules output1 et output2 présentés figure 4.15. On y retrouve l'étiquette *syncMow* sur chaque transition. Cela garantit que l'émission de la nouvelle valeur des sorties est bien forcée par le moniteur à la fin de son cycle après avoir exécuté les différents SFC et attendu la fin du cycle.

### Modélisation par $HIOA_{TS}$ du processus

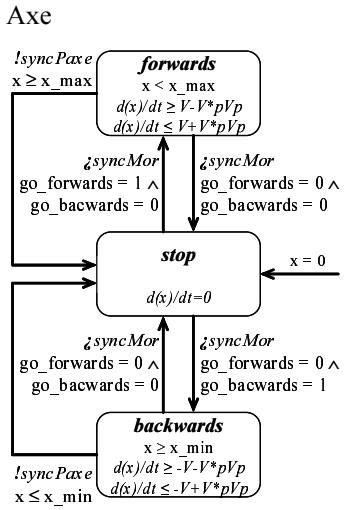
L'automate qui modélise le comportement de l'axe électromécanique de la station 2 (figure 4.16) est composé de trois situations discrètes : une situation stop correspond à la position d'arrêt du convoyeur et deux situations correspondent à la phase de mouvement, soit dans le sens horaire (forwards), soit dans le sens anti-horaire (backwards). La seule variable d'état continue associée à cet automate est la position du convoyeur « x ». À l'initialisation, le convoyeur se trouve arrêté dans la position de chargement, on a donc $x = 0$. À la situation d'arrêt est associée une vitesse de déplacement nulle




| | Variables d'entrée | Variables de sortie | Etiquettes d'entrée |
|---|---|---|---|
| **Output1** | v_go_forwards | go_forwards | *syncMow* |
| **Output2** | v_go_backwards | go_backwards | *syncMow* |

FIG. 4.15 – Modèle $HIOA_{TS}$ des deux sorties du contrôleur

$(\frac{dx}{dt} = 0)$ ; aux deux situations de mouvement est associée une dynamique continue de la forme $\frac{dx}{dt} \geq Vmin$ et $\frac{dx}{dt} \leq Vmax$, où Vmin et Vmax sont des constantes rationnelles telles que $Vmin \leq Vmax$. Cette dynamique continue décrit une affectation arbitraire de la vitesse dans l'intervalle [Vmin, Vmax], ce qui traduit la variabilité de la vitesse de déplacement du convoyeur, due aux perturbations mécaniques subies par l'axe de déplacement (frottement mécanique, couple résistant sur l'axe du moteur, non linéarités de la cinématique). Dans le modèle de la figure 4.16, l'intervalle pour la vitesse a été modélisé sous la forme d'une vitesse nominale $V$ (respectivement $-V$) pour la situation forwards (respectivement backward) et d'un écart exprimé en pourcentage de la vitesse nominale. Ainsi la dynamique continue dans la situation forwards est $\frac{dx}{dt} = V \pm V \cdot pVp\%$, et $\frac{dx}{dt} = -V \pm V \cdot pVp\%$ dans backwards.




| Variables d'entrée | Variables de sortie | Etiquettes d'entrée | Etiquettes locales |
|---|---|---|---|
| go_forwards<br>go_backwards | x | *syncMor* | *syncPaxe* |

FIG. 4.16 – Modèle $HIOA_{TS}$ de l'axe de déplacement

L'étiquette d'entrée *¿syncMor* autorise la synchronisation avec l'$HIOA_{TS}$ qui modélise le comportement du moniteur d'exécution. Elle est associée aux transitions correspondant à la phase de mouvement du convoyeur, et traduit l'émission des sorties en fin de cycle API. Les deux transitions étiquetées *!syncPaxe* modélisent les arrêts sur les deux butées électriques de fin de course. Il s'agit ici d'une étiquette locale car l'arrêt n'est pas dû à un ordre de l'API mais à un comportement provenant du câblage

électrique des contacteurs du moto-réducteur. Dans le module Axe, V, pVp, x_min et x_max sont des constantes.

Il y a un détecteur sur l'axe de déplacement devant chacune des trois sous-stations de travail : le premier à la position de charge Sensor_load, le deuxième à la position de test Sensor_test et le dernier à la position de décharge Sensor_unload. La figure 4.17 montre la modélisation de ces détecteurs. Chacun des modèles correspond à l'instanciation du modèle générique proposé dans la section 3.4.2. La seule modification concerne la modélisation de l'étendue de détection des capteurs. Nous avons ici retenu une formulation de type $position\_centrale \pm e$ au lieu d'un intervalle entre deux valeurs minimale et maximale. L'étendue « e » correspond à la demi-largeur de la came fixée au convoyeur (fig. 4.8) et vaut dans notre cas 12 mm. Dans les modèles, x_load, x_test, x_unload et e sont des constantes.

| Variables d'entrée | Variables de sortie | Etiquettes locales |
|---|---|---|
| x | load_pos | *syncPsl* |

| Variables d'entrée | Variables de sortie | Etiquettes locales |
|---|---|---|
| x | test_pos | *syncPst* |

| Variables d'entrée | Variables de sortie | Etiquettes locales |
|---|---|---|
| x | unload_pos | *syncPsu* |

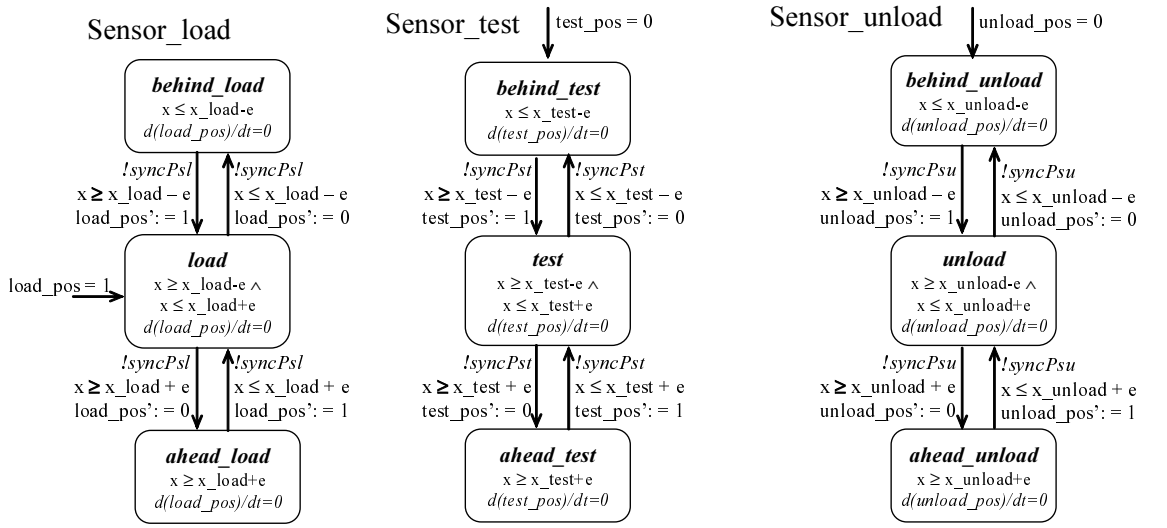



FIG. 4.17 – Modèle $HIOA_{TS}$ des détecteurs de position

### Modèle optimisé pour la vérification

Après traduction en réseau d'*HIOA*, le modèle décrit dans la section précédente n'a pas permis au model-checker PHAVer de réaliser entièrement la composition des automates avec la fonction « compose » dans l'espace mémoire de 3 Go qui lui a été alloué par un OS GNU/Linux travaillant sur une machine avec 4 Go de RAM. La taille de l'automate composé était trop grande pour être stockée en mémoire. La composition s'est interrompue alors que 16384 situations et 58463 transitions avaient déjà été construites. Pour traiter cette étude de cas, il faut donc reprendre de manière experte le réseau d'automates obtenu pour une large part de façon systématique par instanciation de structures génériques.

Nous proposons d'« alléger » le modèle en restreignant la partie modélisée des SFC aux seules étapes pertinentes pour notre étude de cas. Comme nous nous concentrons

sur le mouvement du convoyeur de la sous-station de chargement à la sous-station de test, la figure 4.18 montre à l'aide de traits gras les portions du modèle correspondantes.

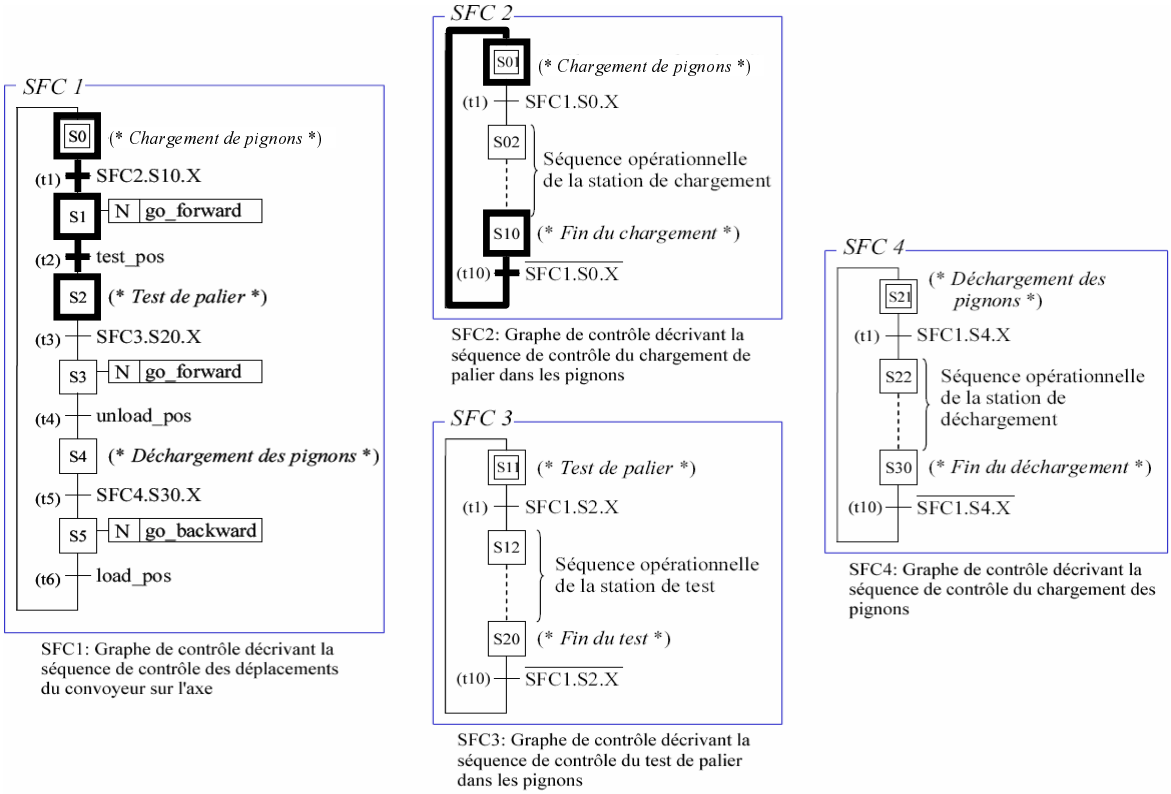



FIG. 4.18 – Commande de l'axe de déplacement de la station 2

Pour les vérifications quantitatives sur la position d'arrêt du convoyeur à la sous-station de test, voici les éléments du programme de commande que nous conservons dans le modèle :

- Pour SFC1, seules les étapes S0, S1 et S2 et les transitions t1 et t2 sont conservées. Cela implique la réduction du module SFC1 et la disparition des modules variable5 (SFC3_S20_X) et variable6 (SFC4_S30_X) qui ne sont plus utilisés par le module SFC1.
- Pour SFC2, seules les étapes S10 et S01 et la transition t10 sont conservées. Cela implique la réduction du module SFC2
- Pour SFC3, rien n'est conservé. Cela implique la disparition du module SFC3, la réduction du module monitor1 qui n'a plus à exécuter SFC3 et la suppression du module variable2 (SFC1_S2_X) qui était une variable utilisée par le module SFC3.
- Pour SFC4, rien est conservé. Cela implique la disparition du module SFC4, la réduction du module monitor1 qui n'a plus à exécuter SFC4 et la suppression du module variable3 (SFC1_S4_X), qui était une variable utilisée par le module SFC4.

Les modules du processus ne sont pas affectés par cette restriction du modèle du programme de commande. La nouvelle structure modulaire du modèle est présentée figure 4.19, tandis que le détail des modules est présenté figures 4.20 et 4.21.

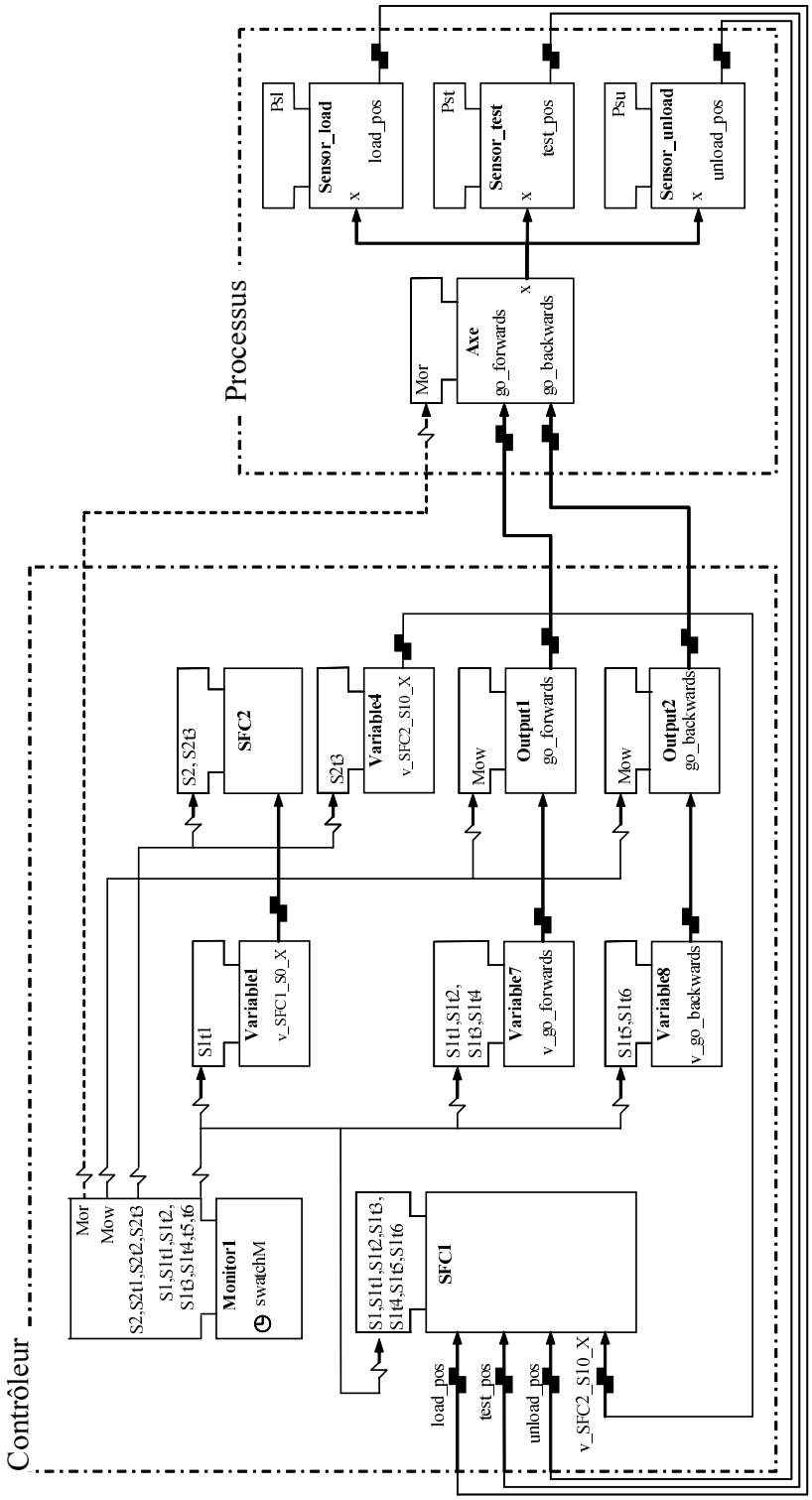



FIG. 4.19 – Modélisation modulaire pour le système avec optimisation pour le model-checker

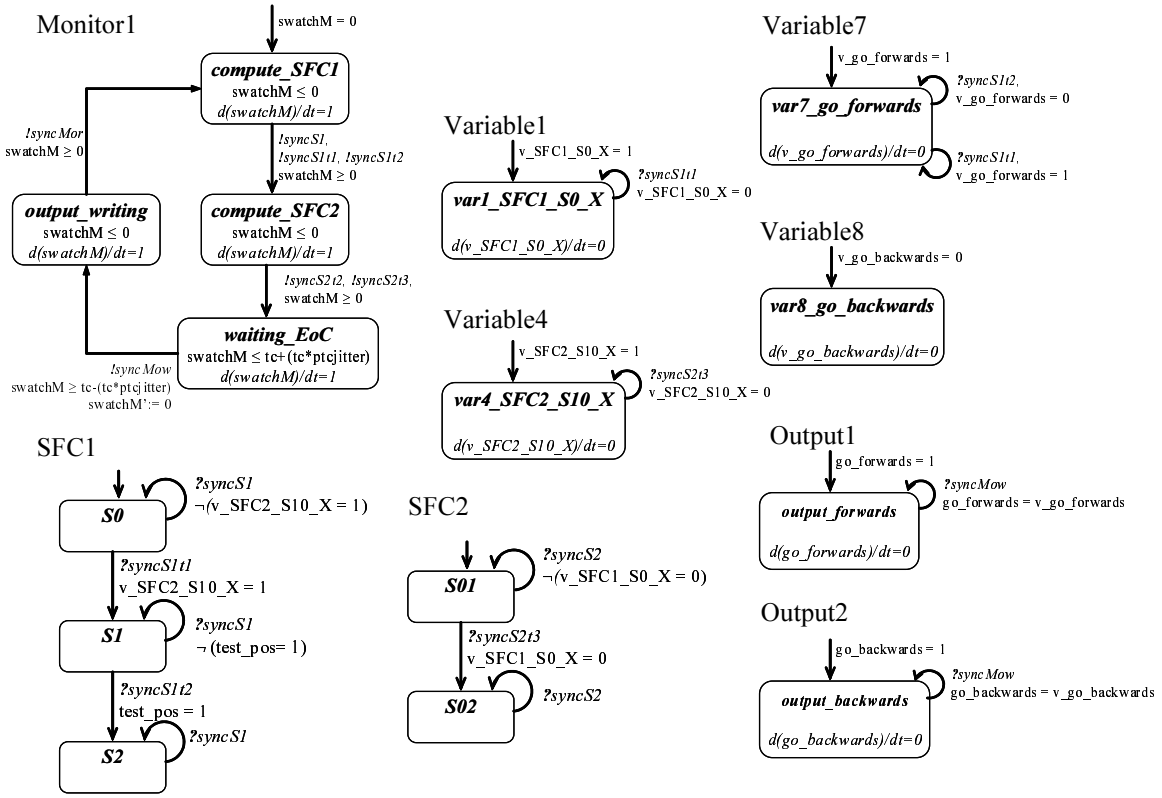



FIG. 4.20 – Modèle complet du contrôleur avec optimisation pour le model-checker

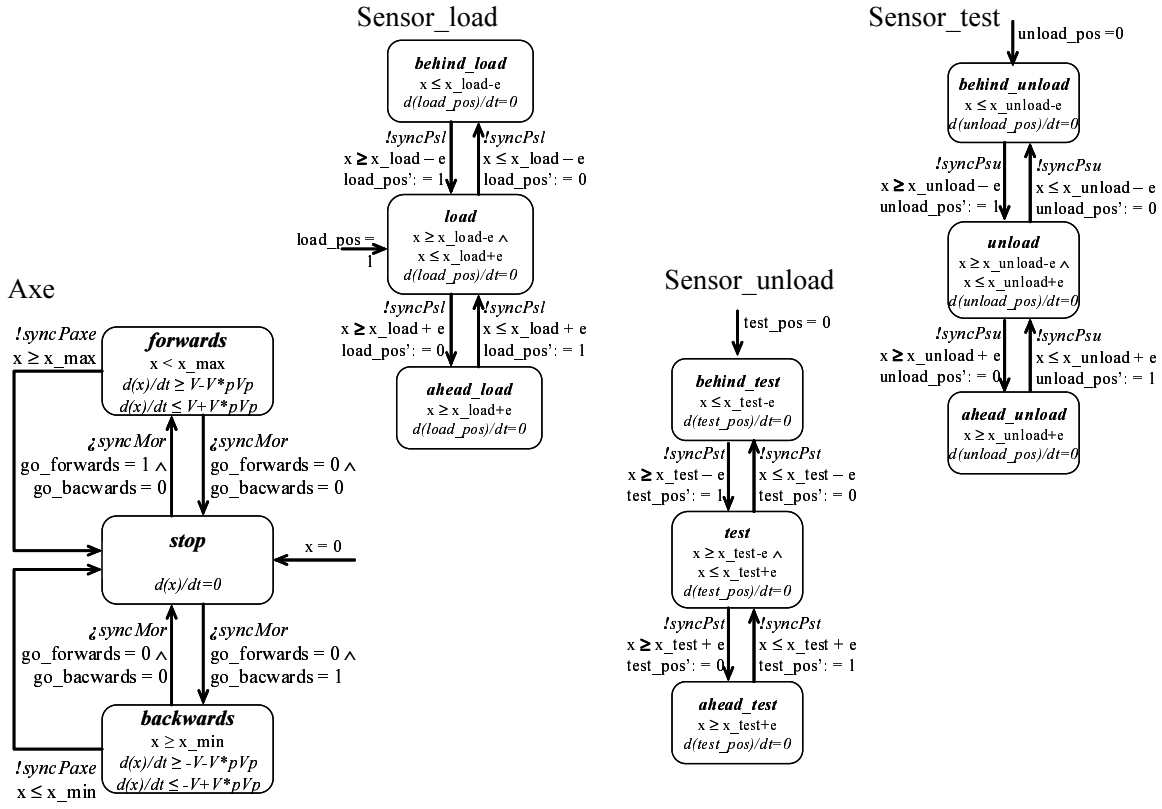



FIG. 4.21 – Modèle complet du processus avec optimisation pour le model-checker

### 4.2.5 Vérification des propriétés quantitatives

Dans toute la suite de l'étude le réseau d'automates $HIOA_{TS}$ des figures 4.19, 4.20 et 4.21 sera utilisé. Pour qu'il puisse être vérifié avec PHAVer, la première étape consiste à traduire le réseau d'automate $HIOA_{TS}$ en un réseau d'automates $HIOA$ à l'aide des outils présentés dans le chapitre 2 (section 2.4). Les figures 4.22 et 4.23 présentent le modèle du module « axe » du processus avant et après traduction. On y voit comment les synchronisations *¿syncMor* avec un rôle non bloquant génèrent des transitions en boucle sur chaque situation avec une garde enrichie.

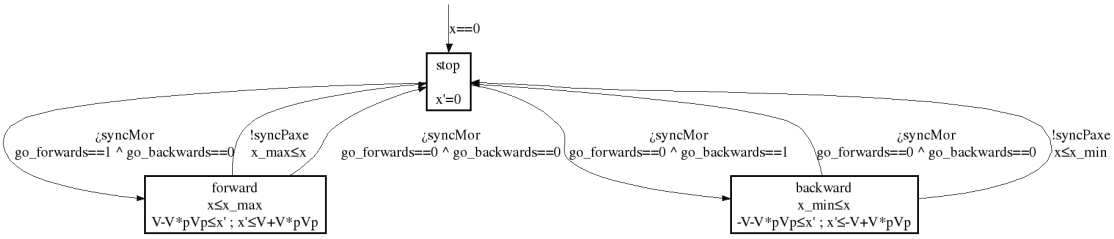



FIG. 4.22 – Modèle $HIOA_{TS}$ du module « axe » du processus avant traduction

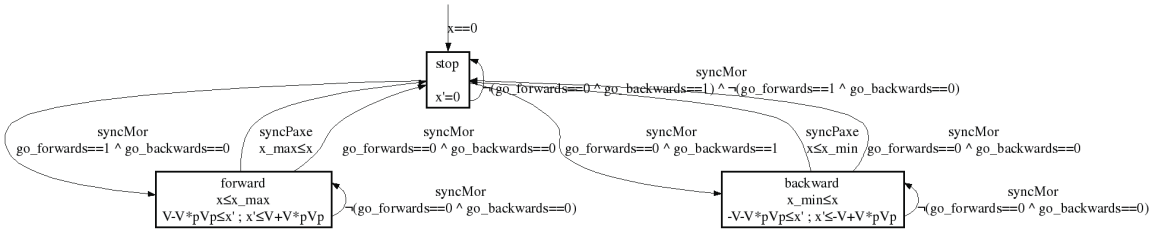



FIG. 4.23 – Modèle $HIOA$ du module « axe » du processus après traduction

Après traduction de ce nouveau réseau d'$HIOA_{TS}$ en réseau d'$HIOA$, la composition de ces automates est, cette fois, correctement réalisée par PHAVer. L'automate composé ainsi obtenu est calculé par PHAVer en $22,31$ s et comporte 504 situation et 1689 transitions (composition réalisée avec un calculateur utilisant un microprocesseur Pentium4 cadencé à $2,4$ GHz). La restriction du programme du contrôleur aux seules parties actives pendant un déplacement du convoyeur vers la sous-station de test a donc parfaitement joué son rôle.

Les valuations des constantes du modèle sont :

- Pour monitor1, tc = 12 ms et ptcjitter = $0,005$, ce qui correspond à un temps de cycle de 12 ms $\pm 0,5$ %
- Pour l'axe, x_min = 0 mm, x_max = 320 mm, V = $0,2$ m/s et pVp = $0,001$, ce qui correspond à une vitesse de $0,2$ m/s $\pm 0,1$ %
- Pour les détecteurs sensor_load, sensor_test et sensor_unload, e = 12 mm, x_load = $-11$ mm, x_test = 132 mm et x_unload = 332 mm

**Première investigation : la région interdite est-elle atteignable ?**

On rappelle que la faisabilité de la commande par un contrôleur logique à la qualité exigée est assurée si $(SFC1.S2.X = 1 \wedge \text{Axe.stop}) \Rightarrow x \in [119, 121]$, $x$ étant exprimé en mm ($SFC1.S2.X$ est l'état de l'étape S2). Soient :

- $Rg_a$ la région atteignable par le modèle à partir de sa situation initiale
- $Rg_1$ la région où $v\_SFC1\_S2\_X = 1$ et Axe.stop est active et $x \geq 121$ mm
- $Rg_2$ la région où $v\_SFC1\_S2\_X = 1$ et Axe.stop est active et $x \leq 119$ mm

La propriété à vérifier s'exprime donc de la façon suivante : « la faisabilité de la commande est assurée si $Rg_a \cap Rg_1 = \emptyset$ et $Rg_a \cap Rg_2 = \emptyset$ et $Rg_a \neq \emptyset$ ». Le résultat du « model-checker » PHAVer est

- la propriété $Rg_a \cap Rg_1 = \emptyset$ et $Rg_a \cap Rg_2 = \emptyset$ est fausse pour tc = 12 ms$\pm 0,5$ %, V = $0,2$ m/s $\pm 0,1$ % et x_test = 132 mm.

La première conclusion est que le contrôleur logique n'est pas assez réactif pour assurer un arrêt dans la région attendue. On se propose donc de vérifier si pour un temps de cycle, un peu plus court le contrôleur est en mesure de garantir la propriété. Les résultats obtenus avec le model-checker PHAVer sont consignés dans le tableau 4.1.

| tc | $Rg_a \cap Rg_1 = \emptyset$ et $Rg_a \cap Rg_1 = \emptyset$ |
|---|---|
| 12 ms | propriété fausse |
| 11 ms | propriété fausse |
| 10 ms | propriété fausse |
| 9 ms | propriété fausse |
| 8 ms | propriété fausse |
| 7 ms | propriété fausse |
| 6 ms | propriété fausse |
| 5 ms | propriété fausse |
| 4 ms | propriété fausse |
| 3 ms | propriété fausse |
| 2 ms | propriété vraie |

TAB. 4.1 – Résultats de vérification

Il est donc *a priori* nécessaire d'utiliser un contrôleur extrêmement réactif (tc = 2 ms) pour que la position d'arrêt du convoyeur soit compatible avec le test. On peut cependant se demander si, lorsque la propriété est fausse, c'est avec une violation « flagrante » ou non de la propriété, ou bien si la position atteignable par le convoyeur reste « proche » de la région acceptable $[119, 121]$, $x$ étant exprimé en mm.

**Deuxième investigation : quantification de la région d'arrêt du convoyeur**

Dans cette deuxième exploration, non seulement la propriété inclut des valeurs quantitatives, mais le résultat de la vérification inclut également une quantité : la position d'arrêt du convoyeur. Soient

- $Rg_a$ la région atteignable par le modèle à partir de sa situation initiale, et
- $Rg_3$ la région où $v\_SFC1\_S2\_X = 1$ et Axe.stop est active.

On cherche à déterminer la région « projection sur l'axe $x$ de $Rg_a \cap Rg_3$ ». Les résultats obtenus avec le model-checker PHAVer sont reportés dans le tableau 4.2. Tous les résultats sont présentés avec seulement quatre chiffres significatifs, mais les valeurs exactes fournies par PHAVer peuvent en contenir beaucoup plus. Par exemple, pour tc = 11 ms, le résultat exact est :

$$[0.12246141600000000003490022 \text{ m}, 0.12393981600000000008200388 \text{ m}]$$

| tc | projection de $Rg_a \cap Rg_3$ sur l'axe $x$ |
|---|---|
| 12 ms | $[0.1224m, 0.1231m] \cup [0.1241m, 0.1248m]$ |
| 11 ms | $[0.1225m, 0.1239m]$ |
| 10 ms | $[0.1220m, 0.1227m] \cup [0.1233m, 0.1240m]$ |
| 9 ms | $[0.1218m, 0.1231m] \cup [0.1234m, 0.1236m]$ |
| 8 ms | $[0.1216m, 0.1223m] \cup [0.1225m, 0.1232m]$ |
| 7 ms | $[0.1214m, 0.1225m] \cup [0.1225m, 0.1228m]$ |
| 6 ms | $[0.1212m, 0.1219m] \cup [0.1217m, 0.1224m]$ |

TAB. 4.2 – Régions obtenues par vérification

Les résultats du tableau 4.2 montrent que les régions d'arrêt sont décalées par rapport à la valeur visée de 120 mm, mais que leur étendue est presque systématiquement compatible avec les 2 mm de la spécification. Cela s'explique par le retard introduit par le contrôleur entre la détection de la position du convoyeur et l'émission de l'ordre d'arrêt du mouvement. On se propose donc d'explorer le lien entre la position d'arrêt et la position du détecteur sensor_test, afin de chercher une position du détecteur qui recentre la région d'arrêt du convoyeur sur l'intervalle visé.

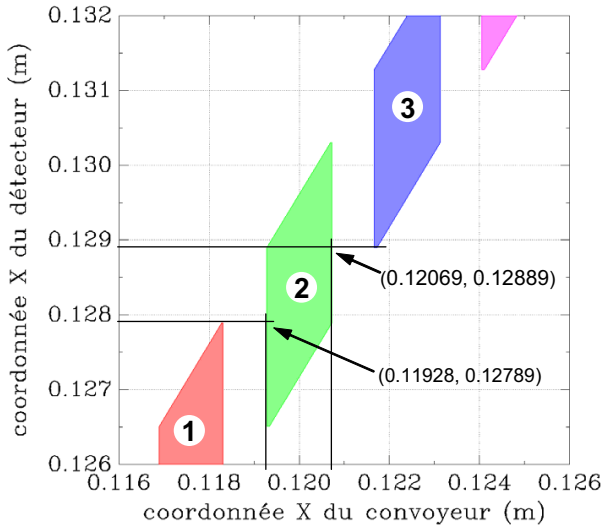
**Troisième investigation : relation entre la région d'arrêt du convoyeur et la position du détecteur sensor_test**

Dans cette troisième exploration, nous nous proposons de pousser plus avant la vérification quantitative en exploitant une fonctionnalité du model-checker PHAVer : l'analyse paramétrique. Il est en effet possible de déclarer à PHAVer qu'une constante doit être considérée comme un paramètre symbolique. Ainsi, non seulement la propriété peut inclure des valeurs quantitatives, mais le résultat de la vérification est exprimé sous la forme d'une fonction d'un ou plusieurs paramètres. Soient :

- $Rg_a$ la région atteignable par le modèle à partir de sa situation initiale, et
- $Rg_3$ la région où $v\_SFC1\_S2\_X = 1$ et Axe.stop est active.

On cherche maintenant à déterminer la région « projection sur l'axe $x$ de $Rg_a \cap Rg_3$ » en fonction du paramètre formel x_test (position où est fixé le détecteur sensor_test) pour tc = 12 ms ± 0,5 % et V = 0,2 m/s ± 0,1 %. Le résultat obtenu avec le model-checker PHAVer est un ensemble de polyèdres dans l'espace à deux dimensions (x, x_test). Ces polyèdres sont tracés figure 4.24.

On y découvre que, pour toutes les positions de réglage du détecteur dans l'intervalle [0.12789 m, 0.12889 m], la région d'arrêt du convoyeur se limite à l'intervalle [0.11928 m, 0.12069 m], qui est dans la limite acceptable pour valider le choix du




Coordonnées des points aux angles du polyèdre 1
(0.11827738800000000011070156, 0.12789177600000000012558644)
(0.11689498800000000005223811, 0.12650937600000000006712298)
(0.11689498800000000005223811, 0.12600000000000000000888178)
(0.11830618800000000006566026, 0.12600000000000000000888178)
(0.11830618800000000006566026, 0.12789177600000000012558644)
(0.11827738800000000011070156, 0.12789177600000000012558644)

Coordonnées des points aux angles du polyèdre 2
(0.12069180000000000001857892, 0.13030618800000000017224167)
(0.11928060000000000000515676, 0.12889498800000000015881952)
(0.11928060000000000000515676, 0.12650937600000000006712298)
(0.11933820000000000005385203, 0.12650937600000000006712298)
(0.12072060000000000011231549, 0.12789177600000000012558644)
(0.12072060000000000011231549, 0.13030618800000000017224167)
(0.12069180000000000001857892, 0.13030618800000000017224167)

Coordonnées des points aux angles du polyèdre 3
(0.12238561200000000004728761, 0.13200000000000000006217249)
(0.12166621200000000000968533, 0.13128060000000000025051605)
(0.12166621200000000000968533, 0.12889498800000000015881952)
(0.12172381200000000000677069, 0.12889498800000000015881952)
(0.12313501200000000002019284, 0.13030618800000000017224167)
(0.12313501200000000002019284, 0.13200000000000000006217249)
(0.12238561200000000004728761, 0.13200000000000000006217249)

FIG. 4.24 – Régions atteintes par le convoyeur selon la position du détecteur (version graphique à gauche, coordonnées exactes des points situés aux angles des polyèdres à droite)

contrôleur logique. La preuve apportée par cette troisième investigation est donc

$$\left.\begin{array}{r} x\_test \in [0.12789\text{ m},\ 0.12889\text{ m}] \\ tc = 12\ ms \pm 0,5\% \\ V = 0,2\ m/s \pm 0,1\% \end{array}\right\} \Rightarrow x \in [0.11928\text{ m}, 0.12069\text{ m}]$$

Le meilleur compromis pour la position de réglage du détecteur sensor_test est le centre de l'intervalle [0.12789 m, 0.12889 m], soit x_test = 128,4 mm.

### Quatrième investigation : influence du temps de cycle de l'API

La figure 4.25 présente la région de la position d'arrêt du convoyeur à la sous-station de test en fonction du temps de cycle du moniteur pour x_test = 128,4 mm, V = 0,2 m/s ± 0,1 % et une gigue sur le temps de cycle de ± 0,5 %. On constate que la region d'arrêt du convoyeur évolue en fonction de tc selon deux phénomènes que nous allons maintenant différencier. Le premier correspond à une erreur systématique (droite de pente positive passant par la valeur moyenne de la région d'arrêt à tc donné), que l'on peut corriger par un réglage adapté de la position du capteur Sensor_test (voir la troisième investigation). Le second phénomène est homogène à une erreur aléatoire (l'étendue de la région d'arrêt pour un tc donné) dont l'amplitude générale croît avec le temps de cycle tc.

Cependant, on remarque que pour certaines valeurs de tc (par exemple tc = 12 ms ou tc = 30 ms), l'étendue de la région d'arrêt (erreur aléatoire) est plus faible qu'elle ne l'est pour les valeurs voisines de tc (par exemple, tc = 12,1 ms ou tc = 29 ms). L'existence de ces zones singulières de plus faible erreur de positionnement, alors même que le temps de cycle de l'API croît, met en évidence un phénomène non trivial de corrélation temporelle entre le cycle API et l'occurrence de l'événement d'arrivée en position de test qui est représenté dans la figure 4.26.

En effet, si malgré la gigue sur le *tc* réglé et la variation de la vitesse $V$, la région temporelle de détection de la position de test (↑ test_pos) est inclue dans un seul cycle de l'API, l'arrêt du convoyeur sera systématiquement commandé lors de la phase d'affectation des sorties du cycle suivant (figure 4.26 scénario1). En revanche, si la région temporelle de détection de la position de test (↑ test_pos) est à cheval sur deux cycles API, l'arrêt se fera lors de la phase d'affectation des sorties de l'un des deux cycles suivants (figure 4.26 scénario2). L'étendue de la zone d'arrêt sera donc plus importante.

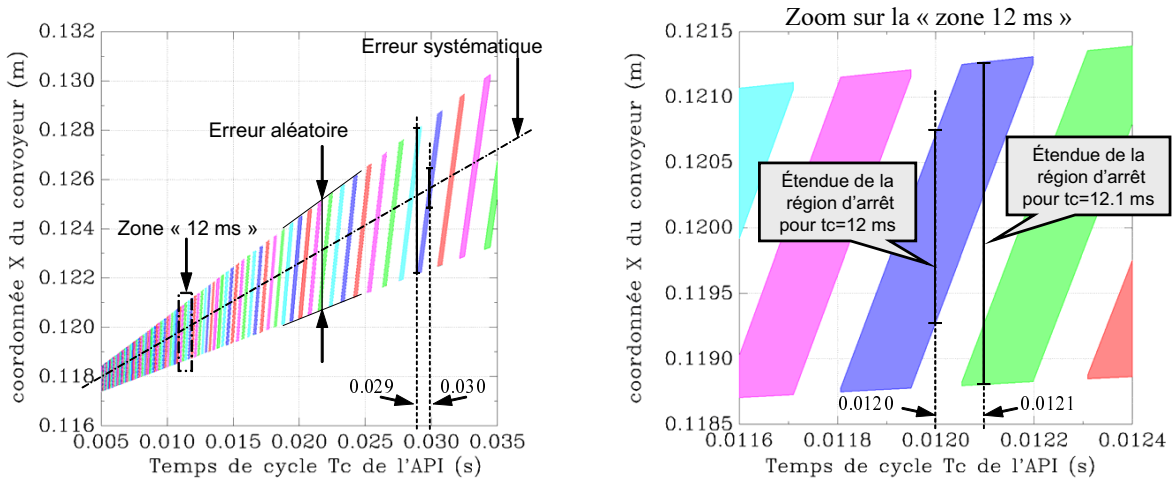



FIG. 4.25 – Région d'arrêt du convoyeur en fonction de tc, avec x_test = 128,4 mm, V = 0,2 m/s ± 0,1 % et une gigue sur le temps de cycle de ± 0,5 %

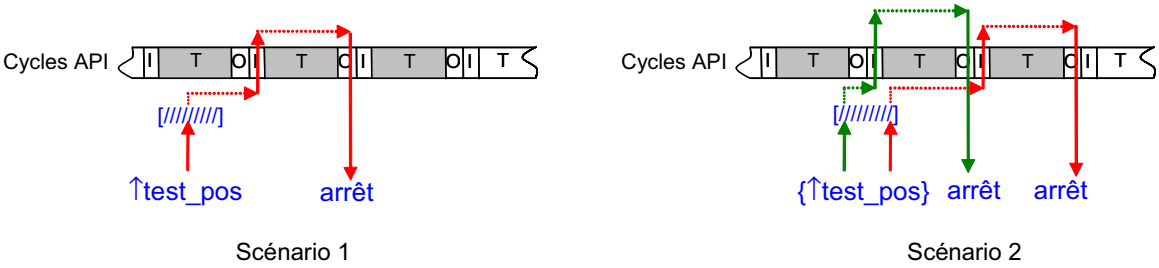



FIG. 4.26 – Corrélation entre tc, les dates d'occurrence de x_test et l'arrêt du convoyeur

Cette dispersion temporelle de l'émission de l'ordre d'arrêt du convoyeur est fortement corrélée à la gigue du temps de cycle de l'API et à la variation de la vitesse $V$. En effet, si le ratio entre la distance à parcourir et tc est tel que, malgré la gigue sur tc et la variation de la vitesse $V$, le nombre N de cycles API qui se sont écoulés entre l'ordre de départ du convoyeur (go_forward) et la détection de la position de test (↑ test_pos) par le capteur est identique à chaque nouveau déplacement du convoyeur de x = 0 mm à x = 120 mm, alors l'ordre d'arrêt du convoyeur ($\neg go\_forward \wedge \neg go\_backward$) est toujours émis à la fin du même cycle API et l'étendue de la région d'arrêt ne dépend que de la gigue, de la variation de la vitesse et du nombre de cycles N (configuration que nous appellerons par la suite « scénario 1 »). Au contraire, si le ratio entre la distance à parcourir et tc est tel que le nombre de cycles nécessaire au déplacement du convoyeur est variable, alors l'ordre d'arrêt pourra être émis à la fin de deux cycles

API différents. L'étendue de la région d'arrêt est alors fonction de tc (configuration que nous appellerons par la suite « scénario 2 »).

La figure 4.27 met bien en évidence ce phénomène en montrant comment la région de l'étendue de la position d'arrêt pour tc donné est construite à partir de la région de la position d'arrêt obtenue par le model-ckecker.

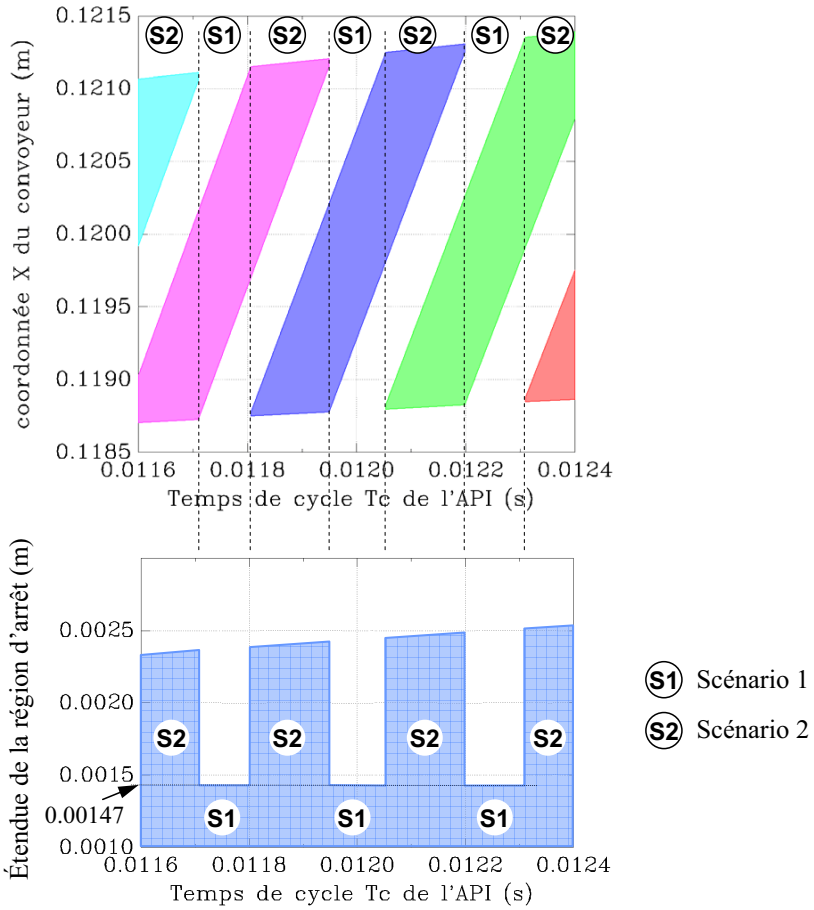



FIG. 4.27 – Construction de l'étendue des positions d'arrêt à partir de la région d'arrêt pour x_test = 128,4 mm, V = 0,2 m/s ± 0,1 % et une gigue sur le temps de cycle de ± 0,5 %

La figure 4.28 donne l'allure générale de la région d'arrêt. Pour les valeurs de *tc* correspondant au scénario 2, l'étendue de la région d'arrêt est proportionnelle à la valeur du temps de cycle. Pour limiter l'erreur de positionnement du convoyeur à ± 1 mm (étendue de 2 mm), il faut dans ce cas imposer un temps de cycle API inférieur à 8,2 ms (après correction de l'erreur systématique par réglage de la position du détecteur Sensor_test à 128, 4 mm). Pour les valeurs de tc correspondant au scénario 1, par contre, l'erreur de positionnement est presque indépendante du temps de cycle et vaut 1,47 mm.

En résumé, pour x_test = 128,4 mm, V = 0,2 m/s ± 0,1 % et une gigue sur le

temps de cycle de ± 0,5 % :

$$\left.\begin{array}{r} \forall \text{tc} \leq 8,9 \text{ ms et } \forall \text{ scénario} \\ ou \\ \forall \text{tc} \in \text{scénario1 (ex : 12 ms, 18 ms)} \end{array}\right\} \Rightarrow \text{erreur de positionnement} \leq 0^{\pm 1\ mm}$$

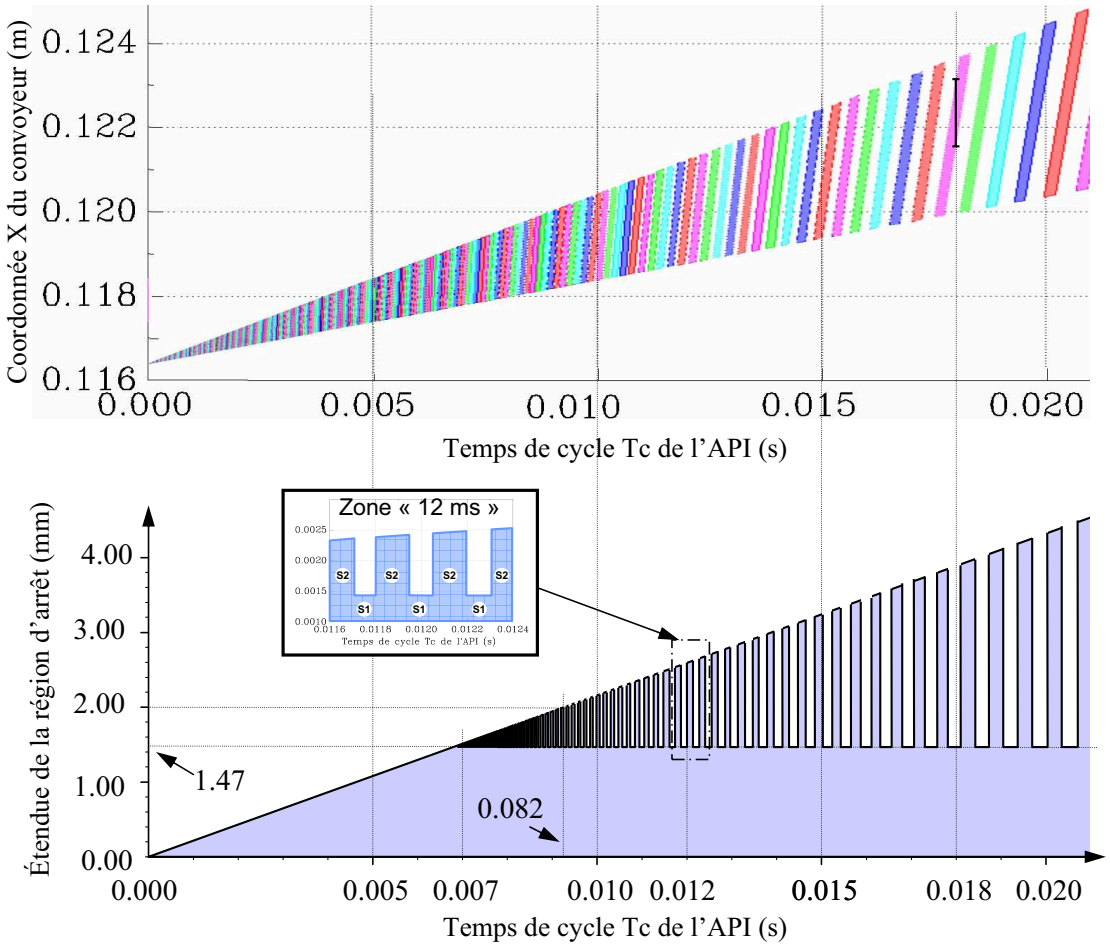



FIG. 4.28 – Allure de l'étendue des positions d'arrêt pour x_test = 128,4 mm, V = 0,2 m/s ± 0,1 % et une gigue sur le temps de cycle de ± 0,5 %

### 4.2.6 Conclusion

Le traitement de cette première étude de cas a montré la pertinence de la démarche de modélisation proposée dans le chapitre 3 pour les systèmes automatisés à contrôleur logique. Seul un automate du modèle du processus, l'axe, a nécessité une conception à partir d'une feuille blanche, tous les autres automates ont été construits par instanciation d'un modèle générique. Après la restriction du modèle au sous-ensemble pertinent pour les propriétés recherchées, la boîte à outils logicielle, développée à cet effet, a permis de traduire le réseau d'$HIOA_{TS}$ en réseau d'$HIOA$ puis de générer le fichier d'entrée pour le « model-checker ». Au travers de quatre investigations successives sur le modèle, nous avons pu découvrir la grande richesse des analyses quantitatives

permises avec un automate hybride. Malgré l'aspect très majoritairement discret du support de cette étude, la modélisation par un modèle hybride ouvre tout un champ d'exploration à l'analyste qui est nouveau et complémentaire à celui qui lui est offert par le « model-checking » des Systèmes à Evénements Discrets (SED).

Enfin, cette première étude de cas a permis de montrer en quoi la qualité d'un service rendu par un système (dans ce cas la précision d'un positionnement) peut directement influer sur sa sûreté de fonctionnement (dans ce cas la capacité de l'opération de test à détecter à coup sûr la présence d'un palier dans un pignon).

## 4.3 Etude de cas 2 : Système de production par lots

### 4.3.1 Présentation de l'étude de cas

Le système que nous allons étudier est un système de production d'eau salée (ou saumure) par lots. Le but est la production de lots d'eau salée à volume, concentration et température donnés. Le processus (figure 4.29) est un système didactique utilisé au laboratoire « Process Control » de l'Université de Dortmund. Etant donné le caractère didactique de ce système, les lots produits sont recyclés en lots de matière première (de l'eau douce et de l'eau salée concentrée). De ce fait, les clients consommateurs sont simulés par des réservoirs de matière première. Le volume, la concentration et la température des lots sont donc fixés par cette contrainte de recyclage pour que le système puisse fonctionner en circuit fermé tant qu'aucun incident ne se produit durant la production.

Nous allons maintenant décrire les principales tâches de la production de lots (une description plus détaillée du système peut être trouvée dans [Kow98]). La production de lots commence par la préparation d'une solution d'eau salée à basse concentration de sel dans le réservoir B3, en mélangeant une solution d'eau salée préalablement contenue dans le réservoir B1 avec de l'eau douce provenant du réservoir B2. La solution résultante est versée de B3 dans le réservoir B4 puis dans le réservoir B5. Dans B5, un processus d'évaporation est réalisé (par chauffage grâce à une résistance électrique H1) pour ajuster les propriétés souhaitées des lots. L'eau évaporée est collectée dans le condenseur K1 puis condensée et récupérée dans le réservoir B6. Dans B6, l'eau est refroidie à température ambiante puis renvoyée dans le réservoir B2 pour être réutilisée. Le produit à haute température qui reste dans B5, une fois l'évaporation arrêtée (chauffage H1 éteint), est versé dans le réservoir B7 où une étape de post-traitement a lieu. Elle consiste à refroidir le lot d'eau salée à une température adéquate et à le renvoyer dans le réservoir B1.

La commande de ce système est réalisé par un Automate Programmable Industriel (API). En effet, même si le système commandé a un caractère hybride dans la mesure où il assemble de nombreuses unités de traitement continu interconnectées, le mode opératoire de ce procédé est de nature essentiellement événementielle, puisqu'il s'agit d'un enchaînement d'étapes pour réaliser une recette. Ainsi, la commande est composée de routines écrites en syntaxe « Sequential Function Chart » (SFC) [IEC93] implantées dans l'API qui assurent la production de lots (décrite précédemment) ainsi que des procédures d'initialisation, d'arrêt et de rinçage du processus.

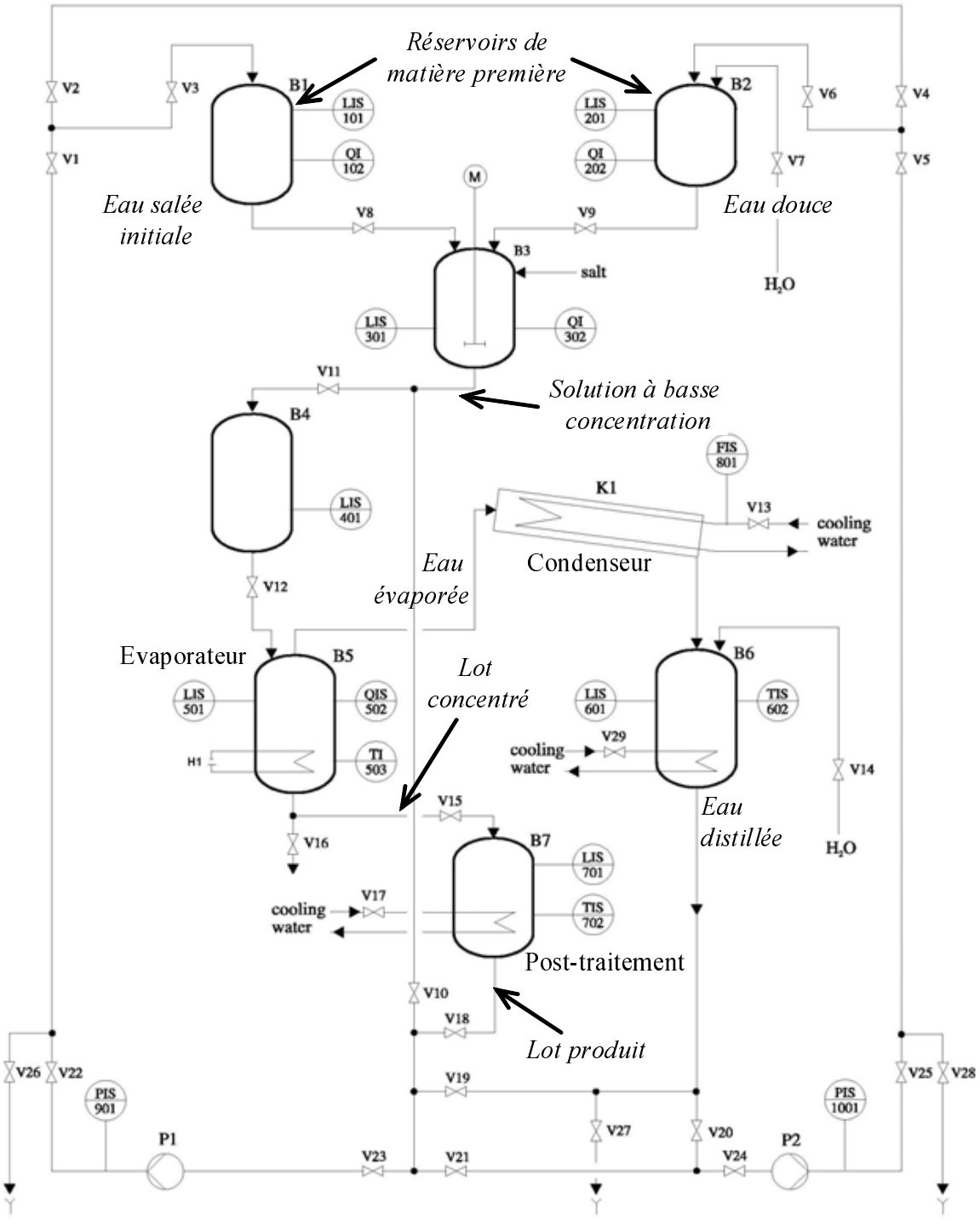


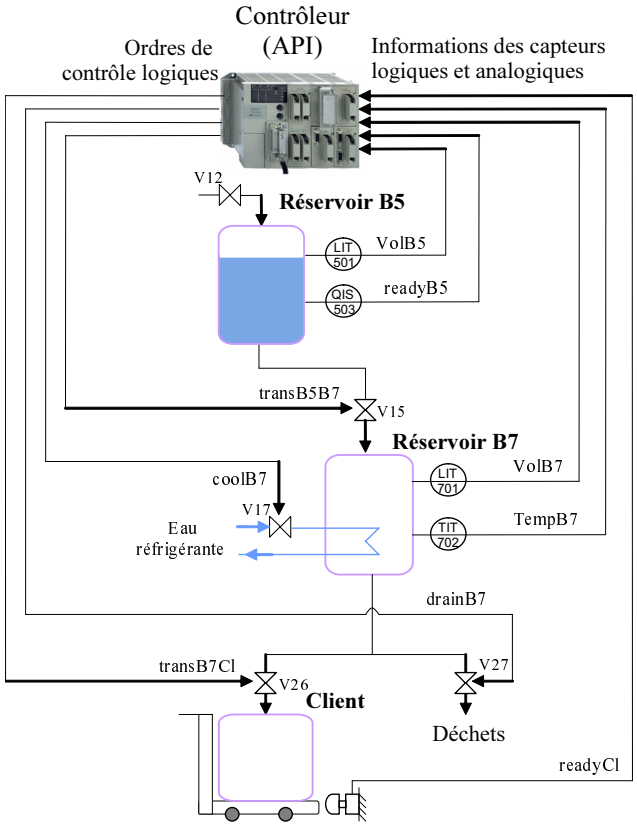

FIG. 4.29 – Diagramme PCF du système de production de saumure par lots

L'objectif de l'étude que nous allons mener est la vérification des propriétés quantitatives permettant d'évaluer les performances du système de production de lots de saumure. L'évaluation de performances est entendue comme la mise en évidence des impacts du choix d'une stratégie de commande sur la production des lots. Une vue externe et non interne du système est nécessaire à cet effet, puisque l'on souhaite qualifier les résultats de la production globale et non ceux d'une étape particulière.

Pour cette étude, nous allons nous concentrer sur la phase finale de la production considérant que le comportement nominal du processus est exempt de fautes. Cette phase comprend les comportements des réservoirs B5 et B7, où les étapes d'ajustement et de post-traitement sont effectuées sur les lots avant qu'ils soient délivrés au client.

### 4.3.2 Présentation de la commande

Dans la suite de cette étude, nous allons uniquement nous intéresser au sous-ensemble des réservoirs B5 et B7 de la production de lots, dont les détails sont montrés sur la figure 4.30. Cette fois, les clients consommateurs ne seront pas simulés par les réservoirs de matière première mais par un réservoir externe, nommé « Client », qui a pour fonction la récupération des lots d'eau salée produits. À cette fin, le circuit en aval de B7 est réduit à une bifurcation gérée par deux vannes V26 et V27.




Entrées du Contrôleur

**VolB5** (Volume du réservoir B5)
**readyB5** (Seuil de concentration du sel atteint)
**VolB7** (Volume du réservoir B7)
**TempB7** (Température du lot dans le réservoir B7)
**readyCl** (Client disponible)

Sorties du Contrôleur

**transB5B7** (Ouverture de la vanne V15 pour le transfert du lot du réservoir B5 vers le réservoir B7)
**coolB7** (Ouverture de la vanne V17 pour le refroidissement du lot dans le réservoir B7)
**transB7Cl** (Ouverture de la vanne V26 pour le transfert du lot du réservoir B7 vers le Client)
**drainB7** (Ouverture de la vanne V27 pour la vidange du lot)

FIG. 4.30 – Réservoirs B5 et B7 de la production de lots de saumure

Le réservoir B5 est rempli avec une solution d'eau salée à basse concentration contenue dans B4. Une fois B5 rempli, un processus d'évaporation permet d'ajuster le volume

et la concentration des lots requis par le cahier des charges. Quand la concentration est atteinte (readyB5), l'eau salée concentrée et à haute température est alors versée de B5 dans le réservoir B7 (transB5B7). Une étape de post-traitement a lieu dans B7 quand le transfert est fini (VolB5). Cette étape consiste à refroidir le lot d'eau salée (coolB7) jusqu'à obtenir la température requise pour la spécification (TempB7). Une fois cette température atteinte, si le client est présent (readyCl), le lot de saumure lui est livré (transB7Cl). Dès que le réservoir B7 est vide (VolB7), ce réservoir est prêt à traiter un autre lot. Cependant, le client peut ne pas être disponible dès qu'un lot est produit. Dans ce cas, la question se pose de savoir que faire du lot produit, et au pire des cas, il est jeté (drainB7).

Le modèle de la stratégie de commande écrit en syntaxe SFC, qui traduit le comportement du processus décrit ci-dessus, est montré dans la figure 4.31. La stratégie de commande de ce SFC est simple. Elle consiste à vérifier la présence du client en fonction de la température requise du lot. Durant la phase de refroidissement (étape S3), si le lot a atteint la température requise de 25°C et que le client est disponible, alors le lot est transféré vers le client (étape S4). Par contre, si le lot atteint la température souhaitée sans que le client ne soit présent, le contrôleur arrête de refroidir et ordonne la vidange du réservoir, cette fois pour rejeter le lot (étape S5). La variable « c » représente un compteur des lots produits et transférés au client (étape S4), « nbatch » étant le nombre de lots à produire. Une fois arrivé à la quantité de lots demandée (par nbatch), la production s'arrête (étape S6).

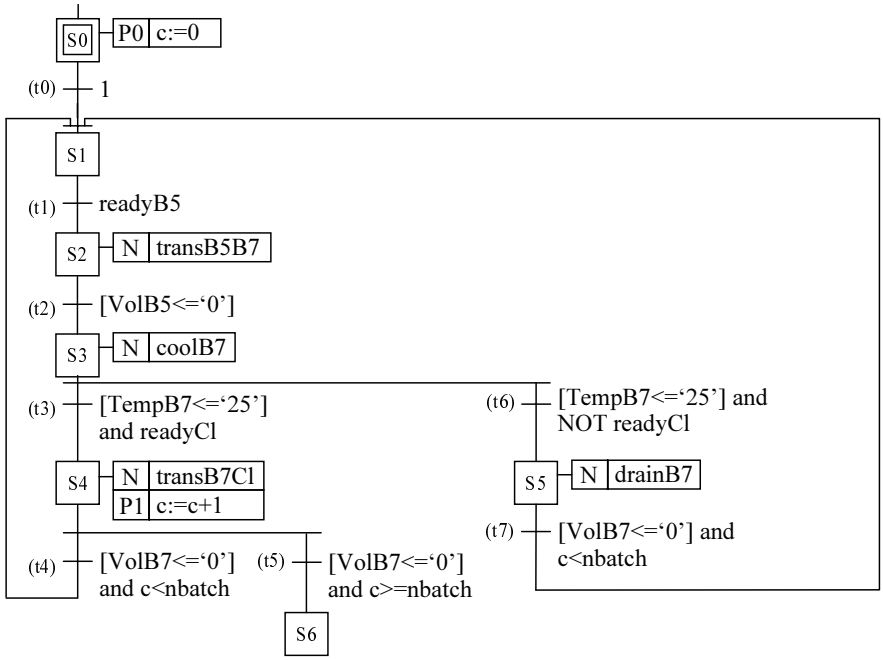



FIG. 4.31 – SFC de la stratégie de commande 1

Cependant, nous nous intéressons au comportement de la commande quand un lot est à la température souhaitée et que le client n'est pas encore disponible. Si le lot continue à refroidir en attendant le client, une cristallisation (solidification du soluté) peut se produire dans le réservoir B7, provoquant l'arrêt de la production et le lancement d'un cycle de rinçage (il y a également le risque d'endommager le réservoir).

Pour éviter cela, le contrôleur est obligé de rejeter le contenu du réservoir (perte du lot et pollution). Cette situation est indésirable puisque la perte de lots signifie que le système doit consommer plus de matière première, et que le temps de production de lots augmente provoquant une augmentation des coûts de production. Plusieurs solutions alternatives existent. Nous en proposons deux : une qui consiste à moduler le refroidissement du mélange dans le réservoir B7 en fonction de la présence ou non du client (appelée stratégie 2) et l'autre qui consiste à ne refroidir le mélange qu'en présence du client (appelée stratégie 3).

La deuxième stratégie de commande, modélisée également en syntaxe SFC, est montrée dans la figure 4.32. Elle consiste à surveiller l'arrivée du client et en fonction de sa disponibilité , àcontrôler le refroidissement du lot. Lorsque B7 est rempli, le refroidissement du lot commence (étape S3) jusqu'à atteindre une limite de température LimitTempB7 (à régler). Ensuite, le contrôleur vérifie la disponibilité du client. Si le client est disponible (branche gauche du SFC), le refroidissement continue (étape S4) jusqu'à atteindre la température de 25°C, puis le lot est délivré au client (étape S5). Par contre, si le client n'est pas encore disponible lorsque la température du lot est LimitTempB7 (branche droite), le contrôleur ordonne l'arrêt du refroidissement pour ralentir la descente de température (étape S7). Dès que le client dévient disponible, et si le lot n'a pas atteint la température désirée, le contrôleur redémarre le refroidissement et retourne à la branche gauche pour terminer la livraison du lot. Dans le cas contraire, si le client n'est toujours pas disponible lorsque la température du lot est arrivée à la température désirée, le contrôleur ordonne la vidange du réservoir pour rejeter le lot (étape S8). Une fois arrivé à la quantité de lots demandée (par nbatch), la production s'arrête (étape S6).

La troisième stratégie de commande est modélisée également en syntaxe SFC (figure 4.33). Elle consiste à n'ordonner le refroidissement du lot qu'en présence du client. Lorsque B7 est rempli (étape S3), le contrôleur vérifie la disponibilité du client. Si le client est disponible (branche gauche du SFC) le refroidissement du lot commence (étape S4) jusqu'à atteindre la température de 25°C (branche droite), puis le lot est délivré au client (étape S5). Par contre, si le client n'est pas encore disponible lorsque la température du lot est 25°C, le contrôleur ordonne la vidange du réservoir pour rejeter le lot (étape S7). Une fois arrivé à la quantité de lots demandée (par nbatch), la production s'arrête (étape S6).

En résumé, nous avons trois stratégies de commande à tester :

1. stratégie 1 (rapide) : refroidissement forcé immédiatement et maintenu en permanence,
2. stratégie 2 (raisonnée) : refroidissement forcé immédiatement puis arrêt à une limite de température à régler si le client n'est pas présent,
3. stratégie 3 (conservative) : refroidissement naturel en absence de client.

Pour pouvoir choisir une stratégie de commande, nous allons chercher à quantifier des indicateurs de performances en termes de vitesse de production de la demande de lots, d'indice de rejet (nombre de litres de produit jeté), de délai de temps d'attente du client, ... de manière à évaluer l'impact du choix d'une stratégie de commande sur les performances de la production globale du système.

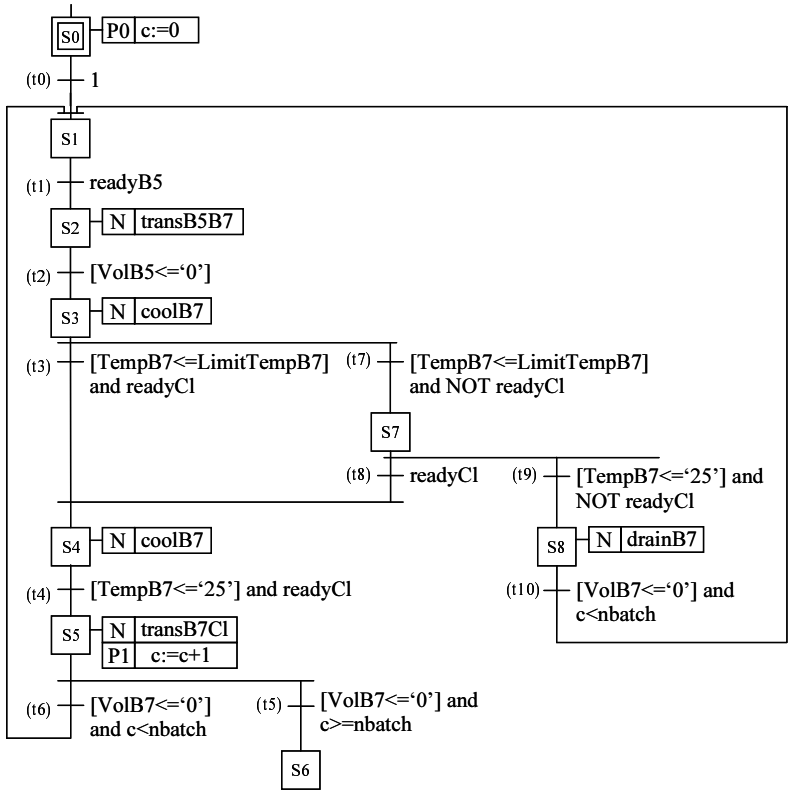



FIG. 4.32 – SFC de la stratégie de commande 2

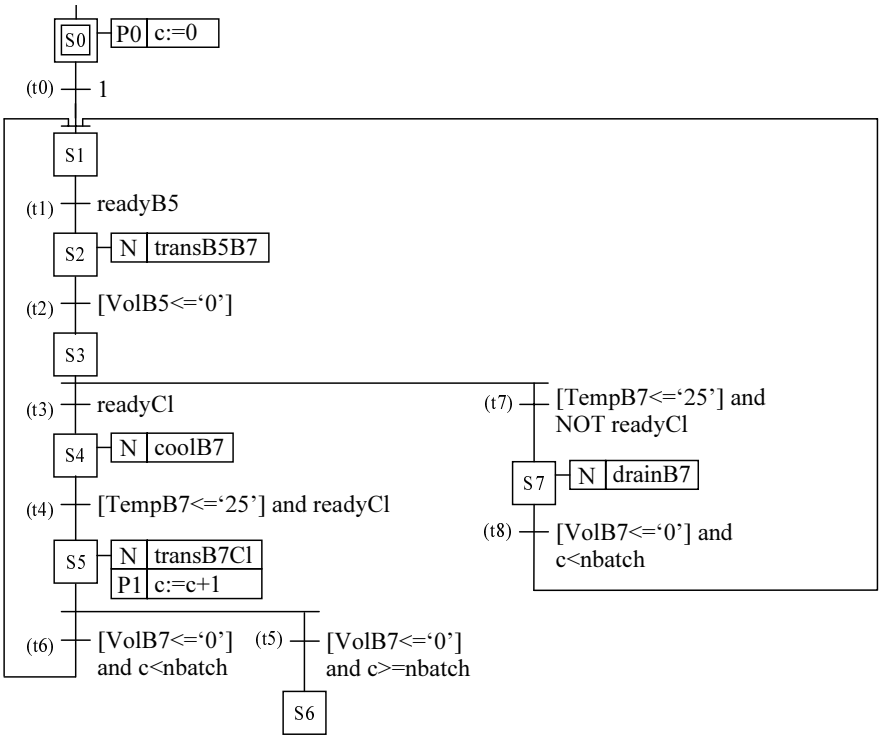



FIG. 4.33 – SFC de la stratégie de commande 3

### Présentation du processus

Un lot produit est composé d'un volume de 4,2 litres d'eau salée à haute concentration (5 grammes de NaCl/litre d'eau) et à une température de 25°C.

Afin de produire des lots avec ces caractéristiques, le réservoir B5 est rempli avec un volume de $7\ell$ d'eau salée (transvasé du réservoir B4 dans B5) à basse concentration ($3g$ de NaCl/$\ell$ d'eau) et à température ambiante. Après une étape d'ajustement du volume et de la concentration dans B5, le lot transvasé de B5 dans B7 contient 4,2 litres d'eau salée à haute concentration, à une température supérieure ou égale à 100°C. Une fois dans le réservoir B7, une étape de post-traitement est donc nécessaire pour refroidir le lot à la température requise pour le Client de 25°C. La figure 4.34 résume ces données.

| Réservoir | Volume | Concentration | |
|---|---|---|---|
| *B5* | Avant évaporation | | Volume non concentré (VolBnoc) |
| | *7 l* | *3 g/l* | |
| | Après évaporation | | Volume concentré (VolBc) |
| | *4,2 l* | *5 g/l* | |

| Réservoir | Volume | Concentration | Température |
|---|---|---|---|
| *B7* | *4,2 l* | *5 g/l* | Avant post-traitement |
| | | | *100°C* |
| | | | Après post-traitement |
| | | | *25°C* |

FIG. 4.34 – Données des volumes, concentrations et températures pour B5 et B7

Les débits des vannes permettant le remplissage et/ou la vidange des réservoirs B5, B7 et Client sont présentés dans la figure 4.35 :

| *Vanne* | *Débits* | *Description* |
|---|---|---|
| *V12* | 1,02 *l*/min | Remplissage de B5 |
| *V15* | 0,72 *l*/min | Transfert de B5 vers B7 |
| *V26* | 1,09 *l*/min | Transfert de B7 vers le Client |
| *V27* | 1,09 *l*/min | Vidange de B7 pour jeter le lot |

FIG. 4.35 – Débits de remplissage et de vidange des réservoirs B5, B7 et Client

Le refroidissement du produit dans le réservoir B7 lorsque l'on injecte de l'eau réfrigérante dans la double enveloppe du réservoir (refroidissement forcé) est représenté par le gradient $-7,5$°C/min (GradTempB7for) tandis que le refroidissement sans injection d'eau réfrigérante (refroidissement naturel) est représenté par le gradient $-0,75$°C/min (GradTempB7nat).

Étant donnée la complexité du processus, certaines hypothèses ont été retenues pour délimiter son comportement. Elles sont décrites comme suit :

- B5 produit en permanence des lots de $4,2\ell$ qui seront traités dans B7,
- Le volume dans B7 reste constant pendant la phase de refroidissement du lot, $coolB7 \Rightarrow \{(\neg transB5B7 \wedge \neg transB7Cl \wedge \neg drainB7) \wedge (VolB7 \geq 4,2\ell)\}$

- La vitesse de refroidissement dépend de la sortie du contrôleur coolB7,
  $coolB7 = 1 \Rightarrow \frac{d}{dt}(TempB7) = GradTempB7for$
  $coolB7 = 0 \Rightarrow \frac{d}{dt}(TempB7) = GradTempB7nat$
- Le client, une fois disponible, attend la réception d'un lot produit et
- Les sorties du contrôleur permettant le transfert des lots d'un réservoir à un autre ne sont jamais activées au même moment,

$$\forall\, t \in \mathbb{R}^* \begin{cases} transB5B7 \wedge transB7Cl = 0 \\ transB5B7 \wedge drainB7 = 0 \\ transB7Cl \wedge drainB7 = 0 \end{cases}$$

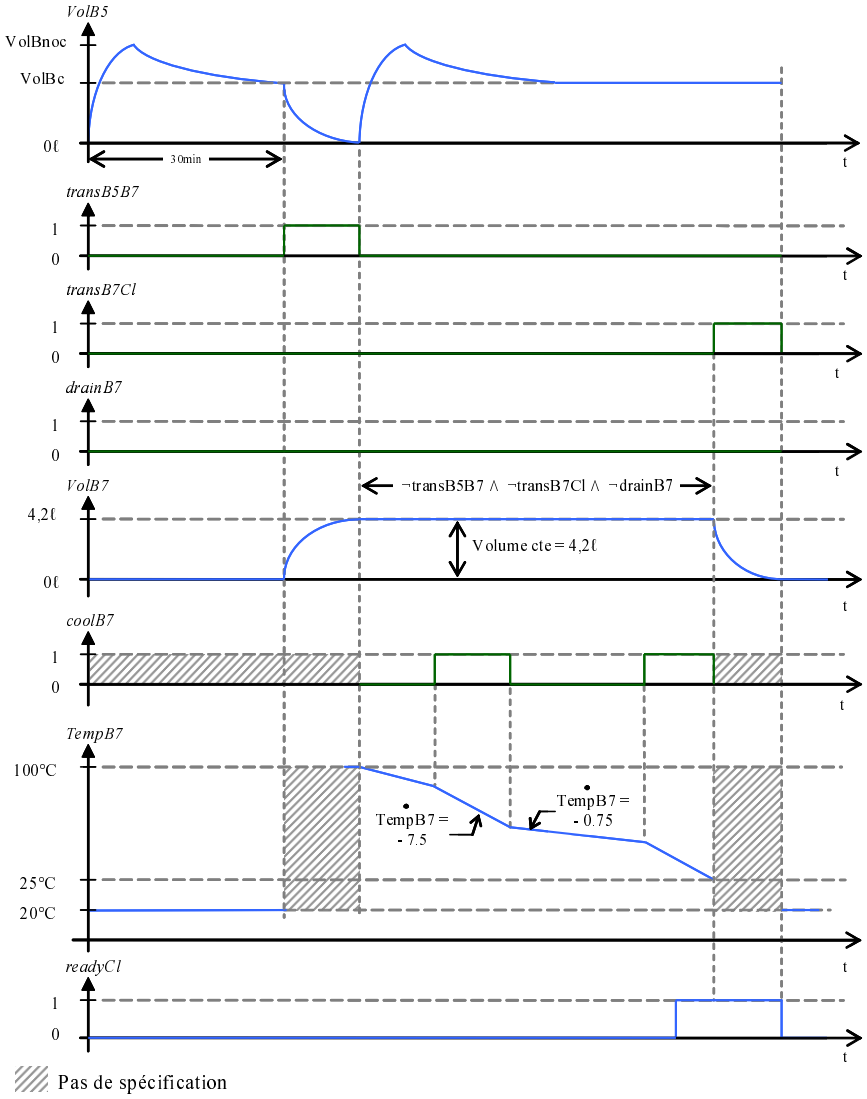



FIG. 4.36 – Hypothèses de fonctionnement du processus

Le client consommateur des lots (Client) est disponible régulièrement pour la réception des lots produits par le réservoir B7 à intervalles déterminés par [delayCmin, delayCmax]. Nous supposons que le client peut être utilisé par d'autres processus qui peuvent éventuellement retarder sa disponibilité pour la réception des lots.

La figure 4.36 montre le comportement du système une fois prises en compte ces hypothèses.

### 4.3.3 Propriétés quantitatives recherchées pour valider la stratégie de commande

Nous nous sommes intéressés à l'évaluation de différents critères quantitatifs sur le système de production afin d'évaluer les performances du système compte tenu du choix de la stratégie de commande, sachant que :

- Nous allons tester les trois stratégies de commande détaillées dans la section 4.3.2,
- La limite de température LimitTempB7 est à régler pour l'utilisateur dans l'intervalle [25°C, 100°C],
- La disponibilité du client est régulière et, une fois disponible, il attend la réception d'un lot produit.

Ainsi, pour une demande de lots déterminée par « nbatch », les critères quantitatifs à évaluer sont les suivants :

- Vitesse de production de la demande de lots, mesurée par le temps de production global,
- Indice de rejet, mesuré par le nombre de lots perdus, et
- Niveau de satisfaction du client, mesuré par le temps d'attente cumulé du client.

Ces propriétés, sous la forme de critères quantitatifs, sont observées à la fin de la production, c'est-à-dire dès que $SFC1.S6.X$ est active.

### 4.3.4 Modélisation du système

La structure du modèle, présentée dans la figure 4.37, reprend les principes de modélisation définis dans la section 3.4.1. Le contrôleur est modélisé par neuf modules $HIOA_{TS}$ tandis que trois modules $HIOA_{TS}$ modélisent le processus. Un module $HIOA_{TS}$ modélise un observateur.

Pour le contrôleur, on retrouve le moniteur d'exécution (Monitor2), un module SFC pour la stratégie choisie (SFC1), un module pour la variable de sortie qui est partagée par le SFC1, un module pour chacune des sorties (Output1, Output2, Output3, Output4), un module pour chacun des détecteurs de seuil de la variable continue de température (ThresholdD1, ThresholdD2) et un module pour le compteur c (Variable1).

Pour le processus, les réservoirs B5, B7 et Client ont été modélisés chacun par un module.

L'interaction entre le processus et le contrôleur est réalisée soit par les variables numériques qui informent sur le niveau des réservoirs B5 et B7 (VolB5, VolB7) et sur la température du réservoir B7 (TempB7), soit par les variables logiques informant que l'on a atteint la concentration requise dans B5 (readyB5) et que le client est présent (readyCl). À son tour, le contrôleur émet ses ordres au processus par les sorties logiques (transB5B7, coolB7, transB7Cl et drainB7). L'émission des sorties est mise à jour selon la cadence du module du moniteur Monitor2 par l'évènement *Mor*.

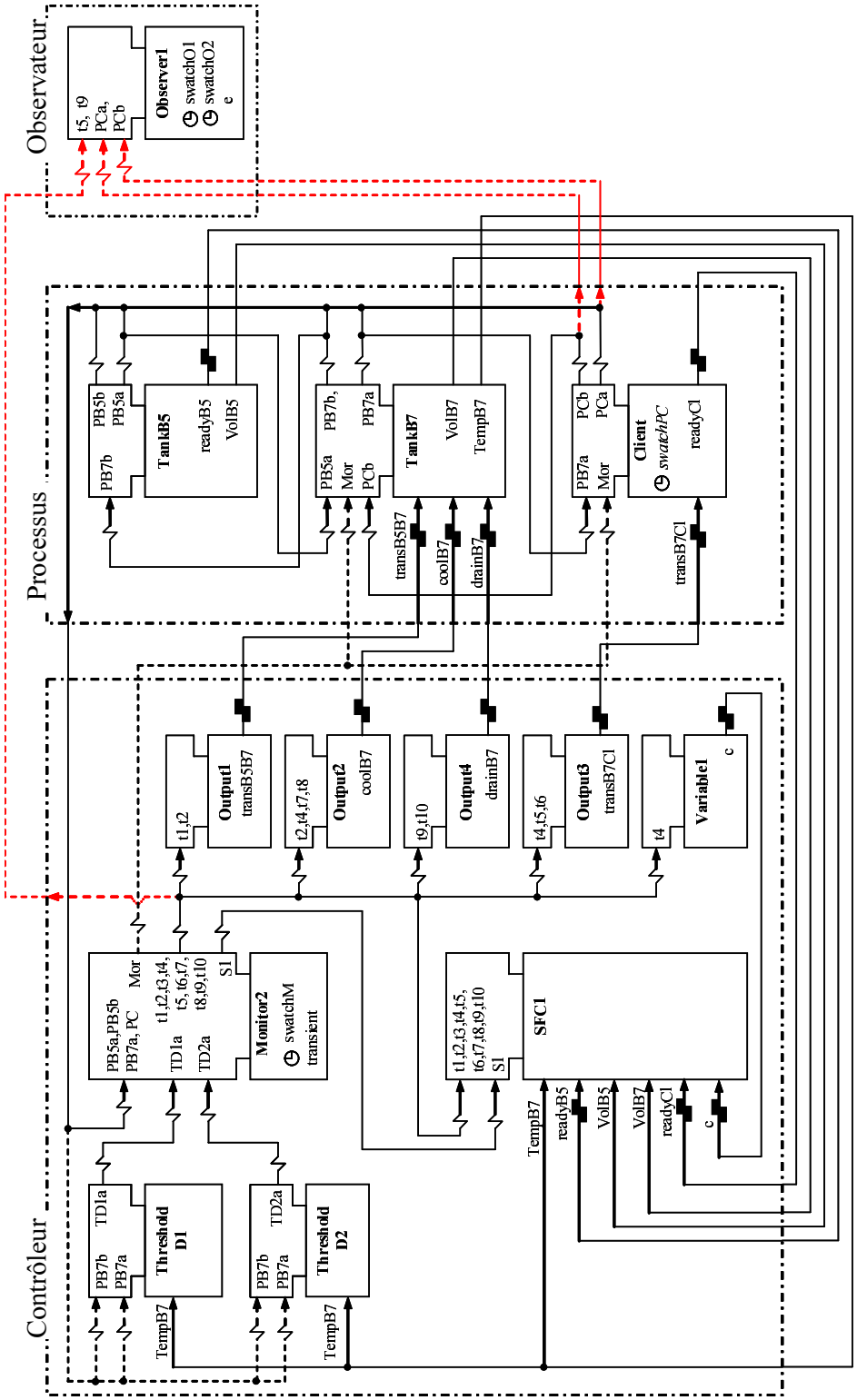


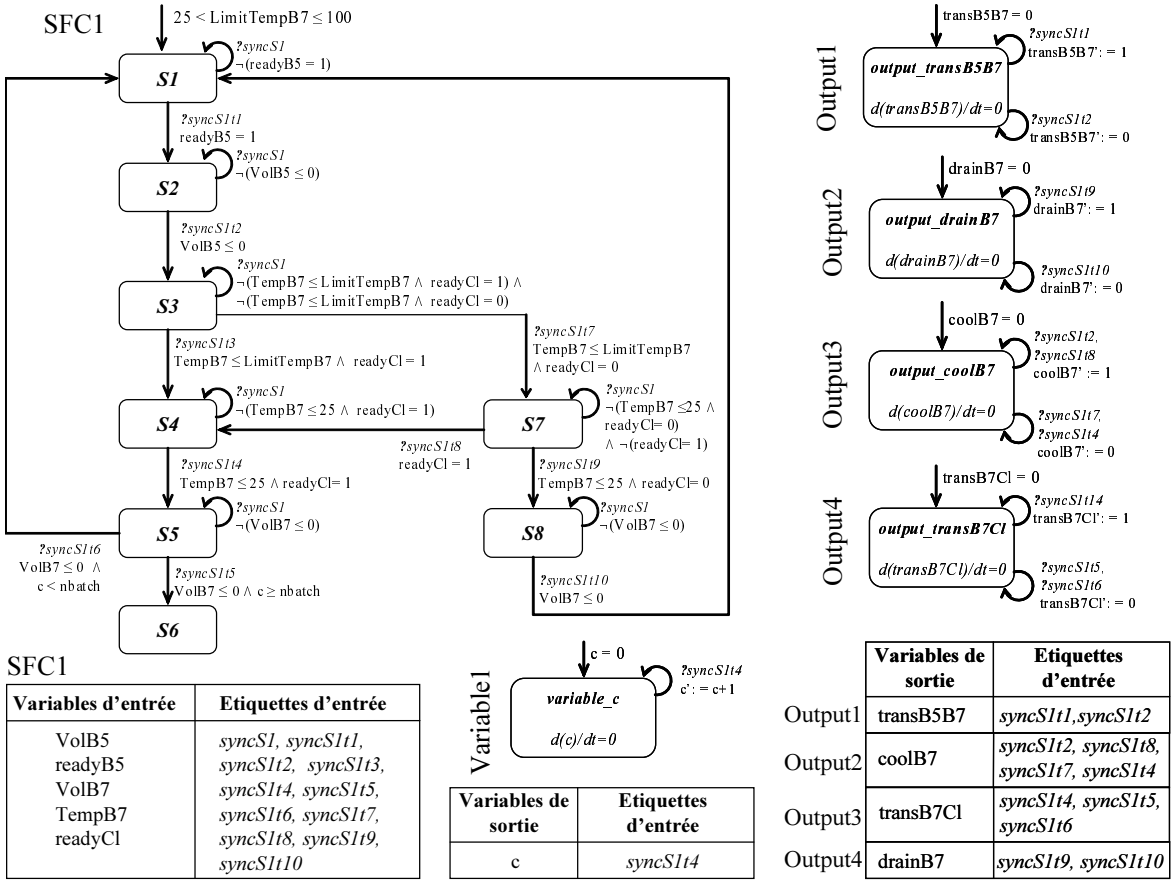

FIG. 4.37 – Structure modulaire du système de refroidissement

Maintenant, regardons plus en détail les modèles de comportement associés à chaque module et leurs interactions.

### Modélisation par $HIOA_{TS}$ du contrôleur logique

La modélisation du contrôleur consiste en la traduction du SFC et la génération du moniteur associé (voir section 3.4.3). Le SFC est traduit par un module $HIOA_{TS}$ qui décrit le séquencement des situations et par un ensemble de modules pour les variables partagées que le SFC modifie (voir section 3.4.3).

Dans le module SFC1, présenté dans la figure 4.38, on retrouve une situation pour chaque étape de SFC1 (fig. 4.32) à partir de S1. L'étape S0 du SFC est traduite par l'état initial du module SFC1. Les quatre sorties transB5B7, drainB7, coolB7 et transB7Cl sont traitées dans les modules Output1, Output2, Output3 et Output4. Par exemple, lorsque le module SFC1 franchit la transition allant de S1 à S2 (étiquette *?syncS1t1* ), alors le module Output1 franchit la transition étiquetée *?syncS1t1* et la variable transB5B7 est mise à 1 conformément au bloc d'action associé à l'étape S2 de SFC1. La variable qui modélise le compteur des lots (variable1) est considérée comme une variable interne au bloc SFC1 étant donné qu'elle n'est pas partagée. LimitTempB7 est une valeur à régler.




SFC1

| Variables d'entrée | Etiquettes d'entrée |
|---|---|
| VolB5<br>readyB5<br>VolB7<br>TempB7<br>readyCl | *syncS1, syncS1t1, syncS1t2, syncS1t3, syncS1t4, syncS1t5, syncS1t6, syncS1t7, syncS1t8, syncS1t9, syncS1t10* |

| Variables de sortie | Etiquettes d'entrée |
|---|---|
| c | *syncS1t4* |

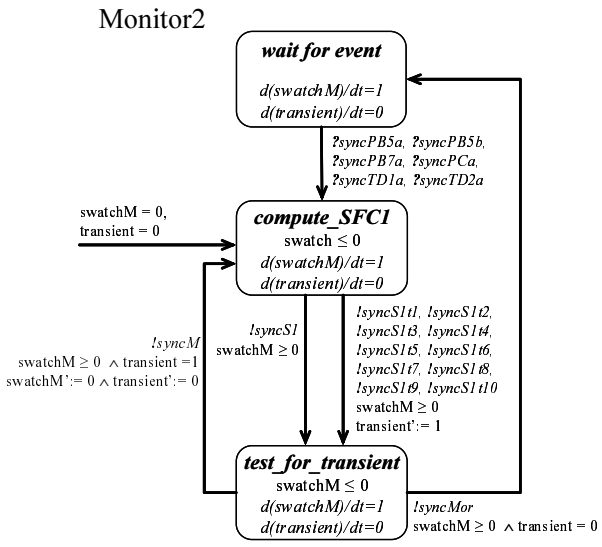
| | Variables de sortie | Etiquettes d'entrée |
|---|---|---|
| Output1 | transB5B7 | *syncS1t1,syncS1t2* |
| Output2 | coolB7 | *syncS1t2, syncS1t8, syncS1t7, syncS1t4* |
| Output3 | transB7Cl | *syncS1t4, syncS1t5, syncS1t6* |
| Output4 | drainB7 | *syncS1t9, syncS1t10* |

FIG. 4.38 – Modèle $HIOA_{TS}$ du programme de contrôle SFC1

Pour cette étude, l'évolution du processus est beaucoup plus lente que la durée d'un cycle d'exécution de l'API. Par exemple, si le contrôleur ordonne d'ouvrir une vanne,

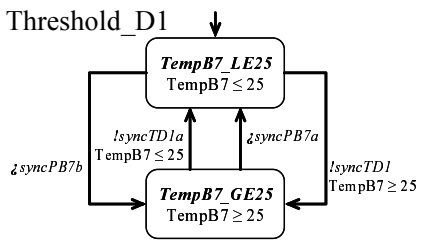
le programme attend simplement que le capteur lui indique que le processus a atteint un nouvel état (réservoir plein) pour réagir. Dans ce cas, l'exécution du programme n'est nécessaire que si le processus évolue et si ce changement satisfait les conditions pour que le système puisse évoluer. C'est pour cette raison que nous allons utiliser le modèle générique de moniteur2.

Le module du moniteur est obtenu par instanciation selon la procédure exposée dans la section 3.4.4, et le résultat de l'instanciation est présenté figure 4.39. Ce module lance l'exécution de SFC1 (compute_SFC1). Il vérifie ensuite toutes les évolutions possibles de SFC1 (test_for_transient). Lorsque aucune évolution du SFC n'est possible (transient=0), il met à jour les valeurs des variables de sortie de l'API par l'étiquette *!syncMor*. Ensuite, il attend l'avertissement des changements du processus (wait_for_event).



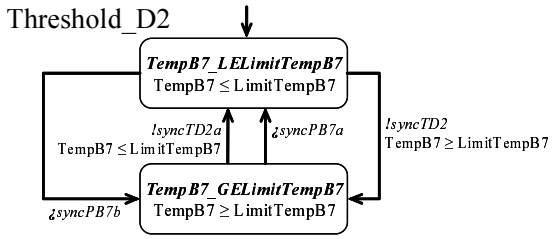

Monitor2

| Variables locales | Etiquettes d'entrée | Etiquettes locales | Etiquettes de sortie |
|---|---|---|---|
| swatchM transient | *syncPB5a, synchPB5b syncPB7a syncPCa syncTD1a syncTD2a* | *syncM* | *syncS1, syncS1t1, syncS1t2, syncS1t3, syncS1t4, syncS1t5, syncS1t6, syncS1t7, syncS1t8, syncS1t9, syncS1t10 syncMor* |




Threshold_D1

| Variables d'entrée | Etiquettes d'entrée | Etiquettes locales | Etiquettes de sortie |
|---|---|---|---|
| TempB7 | *syncPB7b, syncPB7a* | *syncTD1* | *syncTD1a* |




Threshold_D2

| Variables d'entrée | Etiquettes d'entrée | Etiquettes locales | Etiquettes de sortie |
|---|---|---|---|
| TempB7 | *syncPB7b, syncPB7a* | *syncTD2* | *syncTD2a* |

FIG. 4.39 – Modèle $HIOA_{TS}$ du moniteur d'exécution 2

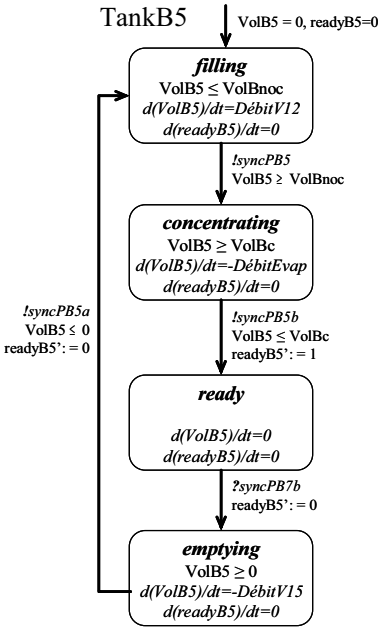
L'information provenant des signaux continus du processus est prise en compte par le moniteur 2 au travers des étiquettes d'entrée non bloquantes générées par les modules des détecteurs de seuil, comme il a été détaillé dans la section 3.4.3. Par rapport au modèle générique, le processus génère les évènements qui préviennent les changements de niveau dans les réservoirs B5 et B7, il n'est donc pas nécessaire de modéliser les détecteurs de seuil pour ces signaux continus. Ainsi, les détecteurs de seuil pour la variable température sont modélisés par les deux modules Threshold_D1 et Threshold_D2 construits selon la section 3.4.4. L'automate D1 génère l'étiquette de synchronisation $syncTD1a$ dès que la température du lot dans le réservoir B7 est

inférieure ou égale à 25°C alors que l'automate D2 génère l'étiquette de synchronisation $syncTD2a$ lorsque la température est inférieure ou égale à la température limite à régler dans le contrôleur.

### Modélisation du processus

Le modèle du processus est composé de trois automates modélisant le comportement des réservoirs B5, B7 et Client.

L'automate qui modélise le comportement du réservoir B5 (fig 4.40) est composé de quatre situations discrètes : la situation *filling*, qui représente le remplissage du réservoir, la situation *concentrating*, qui représente la préparation d'un lot à la concentration requise, la situation *emptying*, qui correspond à la phase de vidange du réservoir, et la situation *ready*, une phase transitoire entre ces deux états qui indique que le lot a été préparé et qu'il peut être transféré vers le réservoir B7. Une fois que B5 est vide, on peut commencer à produire un nouveau lot. Cet automate manipule deux variables de contrôle, l'une continue pour l'évolution du volume dans le réservoir (VolB5) et l'autre logique pour indiquer que le lot a été préparé (readyB5=1). À l'initialisation, le réservoir est supposé vide et, par conséquent, aucun lot n'est préparé, on a donc $VolB5 = 0$ et $readyB5 = 0$. Étant donné que la variable logique n'évolue pas dans le temps, aux quatre situations discrètes du modèle est associée la fonction $\frac{d}{dt}(readyB5) = 0$. Dans trois situations le volume du réservoir subit des changements. Dans chacune de ces situations sont associées les fonctions continues de la forme $d(VolB5)/dt = \alpha$, $\alpha$ étant le débit (DébitV12) du remplissage à volume non concentré (VolBnoc), la vitesse (DébitEvap) pour atteindre le volume à la concentration requise (VolBc), ou le débit (DébitV15) pour vider le réservoir. Dans la situation transitoire le volume reste constant. Pour ce modèle, VolBnoc, VolBc, DébitV12, DébitEvap et DebitV15 sont des constantes.




| Variables de sortie | Etiquettes d'entrée | Etiquettes locales | Etiquettes de sortie |
|---|---|---|---|
| VolB5<br>readyB5 | *syncPB7b* | *syncPB5* | *syncPB5a*<br>*syncPB5b* |

FIG. 4.40 – Modèle $HIOA_{TS}$ du réservoir B5

Cet automate synchronise sa vidange (de la situation *emptying* vers la situation *filling*) avec le remplissage de l'automate modélisant le réservoir B7 (TankB7) par l'étiquette bloquante *?syncPB7b*. L'automate manipule trois étiquettes : deux étiquettes de sortie ( *!syncPB5a* et *!syncPB5b*) pour informer monitor2 des changements dans le volume, et une étiquette locale ( *!syncPB5*) pour réinitier son cycle.

L'automate qui modélise le comportement du réservoir B7 (fig. 4.41) est composé de 7 situations discrètes et initialisé à la situation *empty*. L'automate y demeure jusqu'à ce que le contrôleur ordonne le transfert du lot du réservoir B5 vers le réservoir B7 (transB5B7=1). La situation *empty2filling* est une phase transitoire qui permet la synchronisation de *syncPB7b* avec le réservoir B5 et le début du remplissage de B7 dans la situation discrète *filling*. Dès que le réservoir B5 a transféré complètement son contenu vers B7, une phase de refroidissement commence. Elle est représentée par deux situations discrètes : *forced_ cooling* si le contrôleur ordonne le refroidissement (coolB7=1) et *natural_ cooling* dans le cas contraire. La situation discrète *emptying* correspond à la phase de transfert du lot vers le client et la situation *draining* représente la vidange du réservoir B7 en cas de perte du lot. Les deux variables de contrôle continues sont l'évolution du volume dans le réservoir B7 (VolB7) et l'évolution de la température du lot (TempB7).

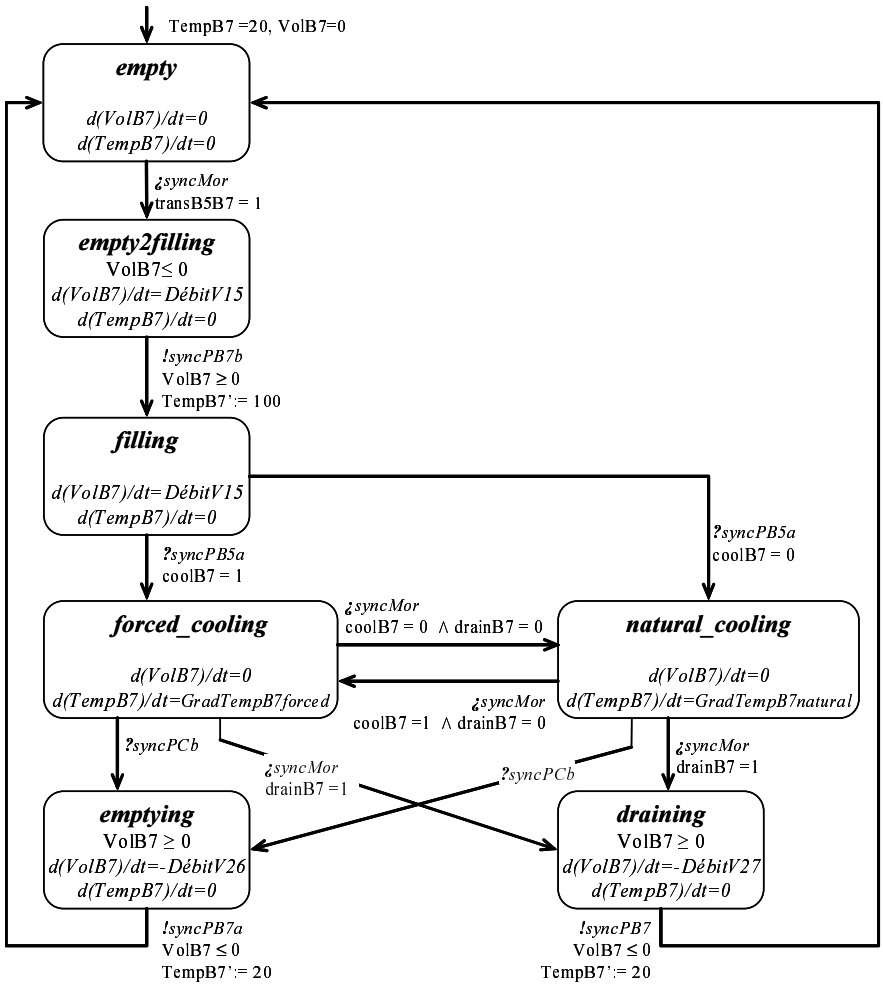


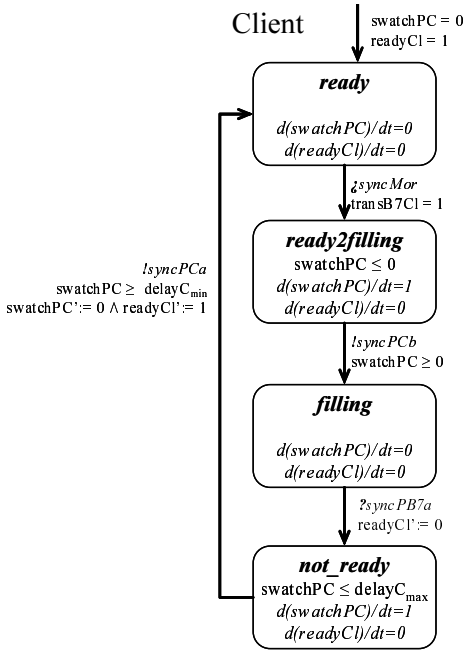

FIG. 4.41 – Modèle $HIOA_{TS}$ du réservoir B7

À l'initialisation, le réservoir est vide et à température ambiante (20°C), on a donc $VolB7 = 0$ et $TempB7 = 20$. Les deux variables continues sont complémentaires, c'est-à-dire que pendant l'évolution du volume, la température ne change pas, et vice-versa, durant l'évolution de la température, le volume reste constant. À la situation initiale sont associées les fonctions $d(VolB7)/dt = 0$ et $d(TempB7)/dt = 0$. Aux situations discrètes où le volume reste constant sont associées les fonctions $d(TempB7)/dt = GradTB7for$ pour le refroidissement forcé et $d(TempB7)/dt = GradTB7nat$ pour le refroidissement naturel. Aux situations où le volume du réservoir change sont associées les fonctions continues de la forme $d(VolB7)/dt = DbitV15$ pour le remplissage de B7, $d(VolB7)/dt = -DbitV26$ pour le transfert vers le client et $d(VolB7)/dt = -DbitV27$ pour le rejet du lot.

Les ordres reçus par le modèle du contrôleur sont les trois variables d'entrée transB5B7, coolB7 et transB7Cl. L'automate est synchronisé avec les automates Monitor2, TankB5 et Client par six étiquettes : trois étiquettes ( *!syncPB7*, *!syncPB7b* et *!syncPB7a*) fabriquées par lui même dont deux ( *!syncPB7b* et *!syncPB7a*) permettent de communiquer avec les automates TankB5 et Client, ainsi que par trois étiquettes d'entrées, provenant des automates TankB5 ( *?syncPB5a*), Client ( *?syncPCb*) et monitor2 (*¿syncMor*). Pour cet automate, GradTempB7for, GradTempB7nat, DébitV15, DébitV26 et DébitV27 sont des constantes.

L'automate qui modélise le comportement du client (fig 4.42) est composé de quatre situations discrètes : la situation *not_ ready*, qui correspond aux périodes où le client n'est pas présent, la situation *ready*, qui représente la phase où le client apparaît, la situation *ready2filling*, qui est une phase transitoire qui indique que le client est présent et que le lot contenu dans B7 peut être délivré au client, et finalement, la situation *filling*, qui correspond au transfert du lot vers le client. Une fois que B7 est vide, le client devient à nouveau indisponible pendant un certain temps. Cet automate manipule deux variables de contrôle, une horloge interne à l'automate (swatchPC) et une variable logique (readyCl=1) pour indiquer au contrôleur que le client est disponible. La période pendant laquelle le client n'est pas disponible est choisie arbitrairement dans un intervalle fermé : [delayCmin, delayCmax]. Ainsi, l'invariant de la situation not_ready devient $swatchPC \leq delayCmax$ et la garde de la transition qui permet de quitter cette situation devient $swatchPC \geq delayCmin$. À l'initialisation, le client est toujours présent et l'horloge est mise à zéro, par conséquent, nous avons $swatchPC = 0$ et $readyCl = 1$. Étant donné que la variable logique n'évolue pas dans le temps, aux quatre situations discrètes du modèle est associée la fonction $d(readyCl)/dt = 0$. Dans la situation *ready*, l'horloge *swatchPC* s'active jusqu'à ce que le lot puisse être délivré. L'horloge de l'automate est activée pour décompter la durée d'indisponibilité du client et pendant la situation transitoire. Durant le remplissage, les horloges sont inactives.

Le seul ordre reçu par le modèle du contrôleur est la variable d'entrée transB7Cl. L'automate est synchronisé avec les automates Monitor2, TankB7 et Observateur par quatre étiquettes, deux étiquettes de sortie appelées *!syncPCa* et *!syncPCb* et deux étiquettes d'entrée, une bloquante ( *?syncPB7a*) et une non bloquante (*¿syncMor*). Dans ce modèle delayCmin et delayCmax sont des constantes.




| | Etiquettes locales | Etiquettes de sortie |
|---|---|---|
| | *syncPB7a* *syncMor* | *syncPCa* *syncPCb* |

| Variables d'entrée | Variables locales | Variables de sortie |
|---|---|---|
| transB7Cl | swatchPC | readyCl |

FIG. 4.42 – Modèle $HIOA_{TS}$ du Client

### Modélisation de l'observateur

L'observateur permet la recherche des propriétés quantitatives (indicateurs de performance) comme expliqué dans le chapitre 3 (section3.4.5). Le module présenté dans la figure 4.43 est composé des trois situations discrètes : *client_ waiting* , *production* et *end_ of_ production*. Trois variables sont contrôlées par cet automate : le compteur « e » pour calculer le nombre de lots rejetés (observé par l'étiquette *¿syncS1t9* de SFC1 dans la fig. 4.38), l'horloge « swatchO1 » qui mesure le temps de production globale de la demande des lots exigée (nbatch) et l'horloge « swatchO2 » qui mesure le temps global d'attente du client pendant toute la production des lots. À l'initialisation, la production de lots démarre, donc nous avons $e = 0$, $swatchO1 = 0$ et $swatchO2 = 0$. Le compteur n'étant pas une variable continue, sa dynamique est de la forme $d/dt(e) = 0$ pour toutes les situations discrètes. L'horloge « swatchO1 », quant à elle, est activée $(d/dt(swatchO1) = 1)$ dans les situations discrètes client_waiting et production, par contre, l'horloge « swatchO2 » n'est activée $(d/dt(swatchO2) = 1)$ que dans la situation discrète client_waiting.

L'automate observateur synchronise avec l'automate Client par les étiquettes *¿syncPCa* et *¿syncPCb* pour permettre l'observation de la production et du temps d'attente du client. L'étiquette *¿syncS1t5* provenant de l'automate SFC1 permet d'arrêter l'observation de l'information.

### Modèle optimisé pour la vérification

Après la traduction en réseau d'$HIOA$ du modèle décrit dans la section précédente (section 4.3.4), celui-ci est transmis au « model-checker » PHAVer, pour d'abord, réaliser la composition des automates et, ensuite, permettre le calcul de la région atteignable. Le modèle composé par le « model-checker » contient 6652 situations discrètes

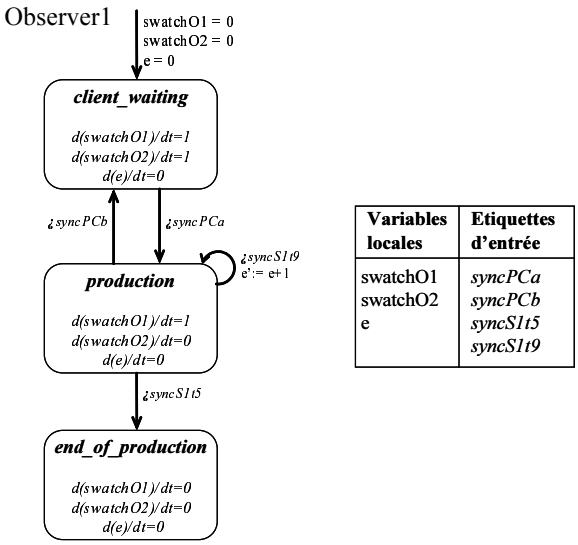



FIG. 4.43 – Modèle $HIOA_{TS}$ de l'observateur1

et 23846 transitions et il est obtenu en approximativement 7 minutes. Cependant, lors de la recherche de la région atteignable ce modèle est limité à une production de 2 lots (nbatch=2), avec un délai de disponibilité du client de 30 minutes ±10 % (delayCmin=27 min, delayCmax=33 min, voir fig. 4.42), avant d'atteindre les limites dans l'espace mémoire de 3 Go de l'ordinateur (même machine que celle utilisée pour l'exemple précédent).

Afin d'aller un peu plus loin dans les tests de vérification, nous proposons d'alléger le modèle en réduisant le nombre de situations discrètes de la modélisation décrite précédemment. Pour cela, il faut donc reprendre de manière experte le réseau d'automates obtenu pour une large part de façon systématique par instanciation de structures génériques. Pour cette étude de cas, nous proposons de réduire la taille du module TankB5. Nous proposons la fusion des situations discrètes *filling* et *concentrating* en une nouvelle situation *filling_and_concentrating* (voir fig 4.44). Cette simplification est possible parce que l'évolution de *filling* vers *concentrating* est propre à l'automate (étiquette locale *!syncPB5*) et qu'aucun autre module n'utilise cette information.

La structure modulaire du système ne change pas (figure 4.37), mais cette modélisation permet une vérification plus efficace dans les limites du « model-checker » PHAVer. Les détails des modules sont présentés sur les figures 4.45 et 4.46.

### 4.3.5 Vérification des propriétés quantitatives

Dans toute la suite de l'étude le réseau d'automates $HIOA_{TS}$ des figures 4.37, 4.45 et 4.46 sera utilisé. Pour qu'il puisse être vérifié avec PHAVer, la première étape consiste à traduire le réseau d'automates $HIOA_{TS}$ en réseau d'automates $HIOA$ à l'aide des outils présentés dans le chapitre 2, section 2.4. À titre d'exemple, les figures 4.47 et 4.48 présentent les modèles du module « TankB7 » du processus avant et après traduction. On y voit comment les synchronisations *¿syncMor* avec un rôle non bloquant génèrent des transitions en boucle sur chaque situation avec une garde élaborée.

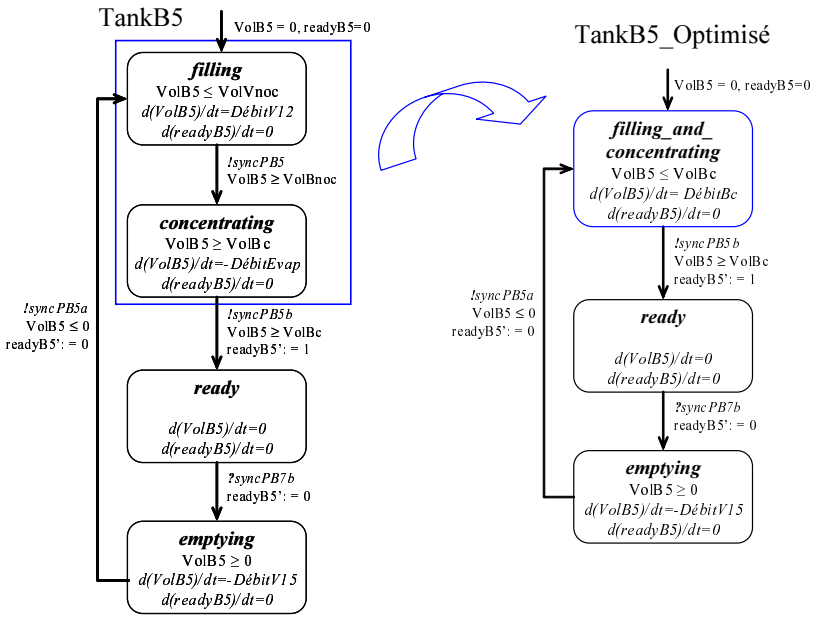



FIG. 4.44 – Modèle $HIOA_{TS}$ du réservoir B5 optimisé

Après traduction de ce nouveau réseau d'$HIOA_{TS}$ en réseau d'$HIOA$, la composition de ces automates par PHAVer nécessite 265, 78 s, toujours avec un calculateur utilisant un microprocesseur Pentium4 cadencé à 2, 4 GHz. L'automate composé contient 3436 situations et 12648 transitions. La fusion des deux situations (filling et concentrating) du module TankB5 du processus en une seule (filling_and_concentrating) a donc permis une réduction significative de la taille de l'automate composé. Rappelons que le modèle composé sans cette simplification (voir section 4.3.4) était constitué de 6652 situations et 23846 transitions.

Les données du modèles sont les suivantes :

***Concernant le produit :***
- Le volume du lot avant concentration, VolBnoc=$7\ell$,
- Le volume du lot après concentration, VolBc=$4,2\ell$,

***Concernant le processus :***
- Le débit de la vanne V12, DébitV12=$1,02\ell/min$,
- Le débit d'évaporation dans B5 lors du chauffage, DébitEvap=$0,93\ell/min$,
- Le débit de la vanne V15, DébitV15=$0,72\ell/min$,
- Le débit de la vanne V18, DébitV18=$1,09\ell/min$,
- Le gradient de température dans B7 lors de l'injection d'eau réfrigérante, $GradTempB7for = -7,5$°C,
- Le gradient de température dans B7 sans injection d'eau réfrigérante, $GradTempB7nat = -0,75$°C,

***Concernant le contrôleur :***
- Les stratégies de commande à tester : 1, 2 et 3 (voir fig. 4.31, 4.32 et 4.33),
- La température de coupure pour la stratégie 2, LimitTempB7 $\in$ $[25, 100]$°C,

***Concernant la production :***
- Le nombre de lots à produire, nbatch,

***Concernant le client :***

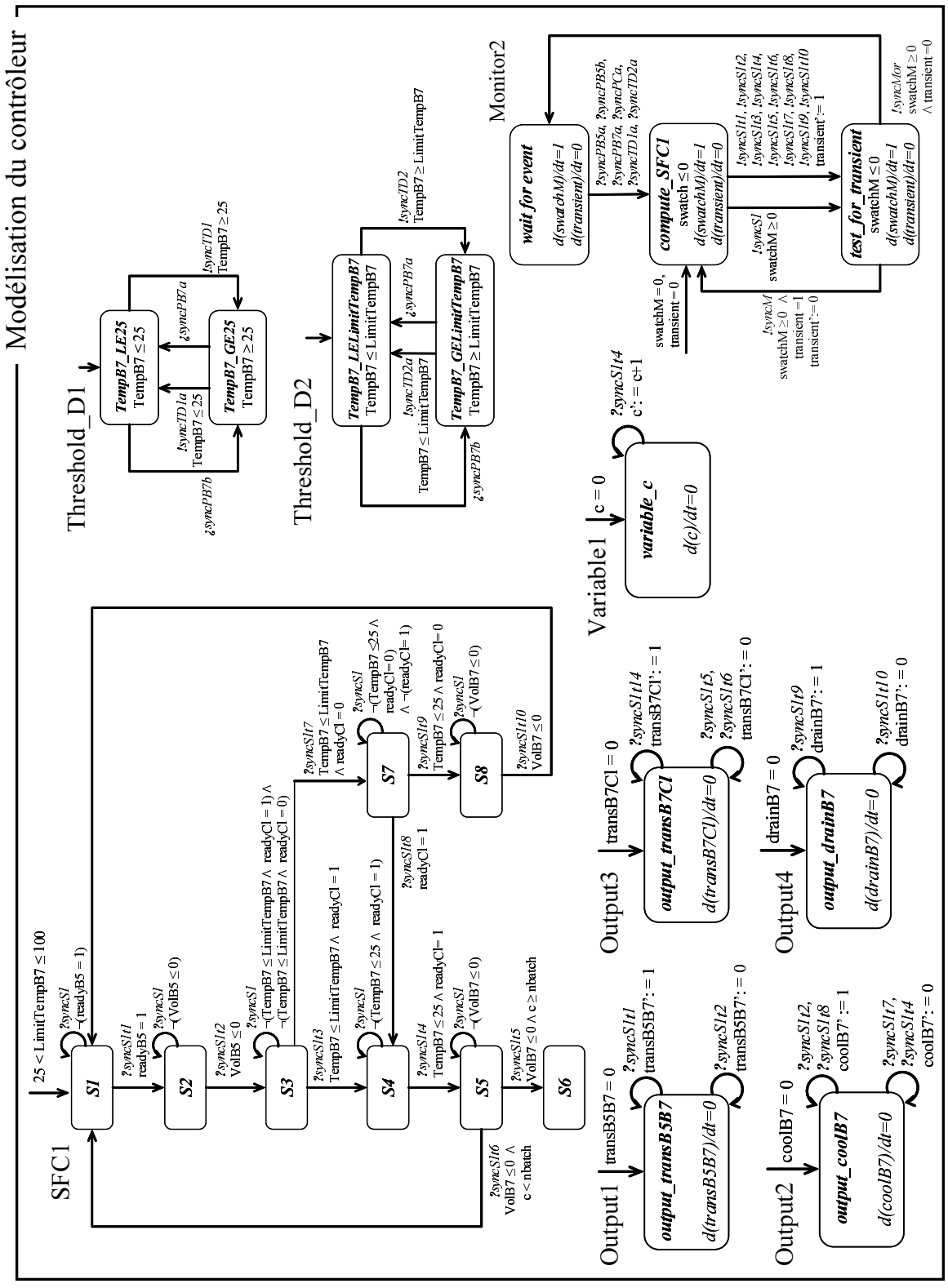



FIG. 4.45 – Réseau d'$HIOA_{TS}$ du contrôleur

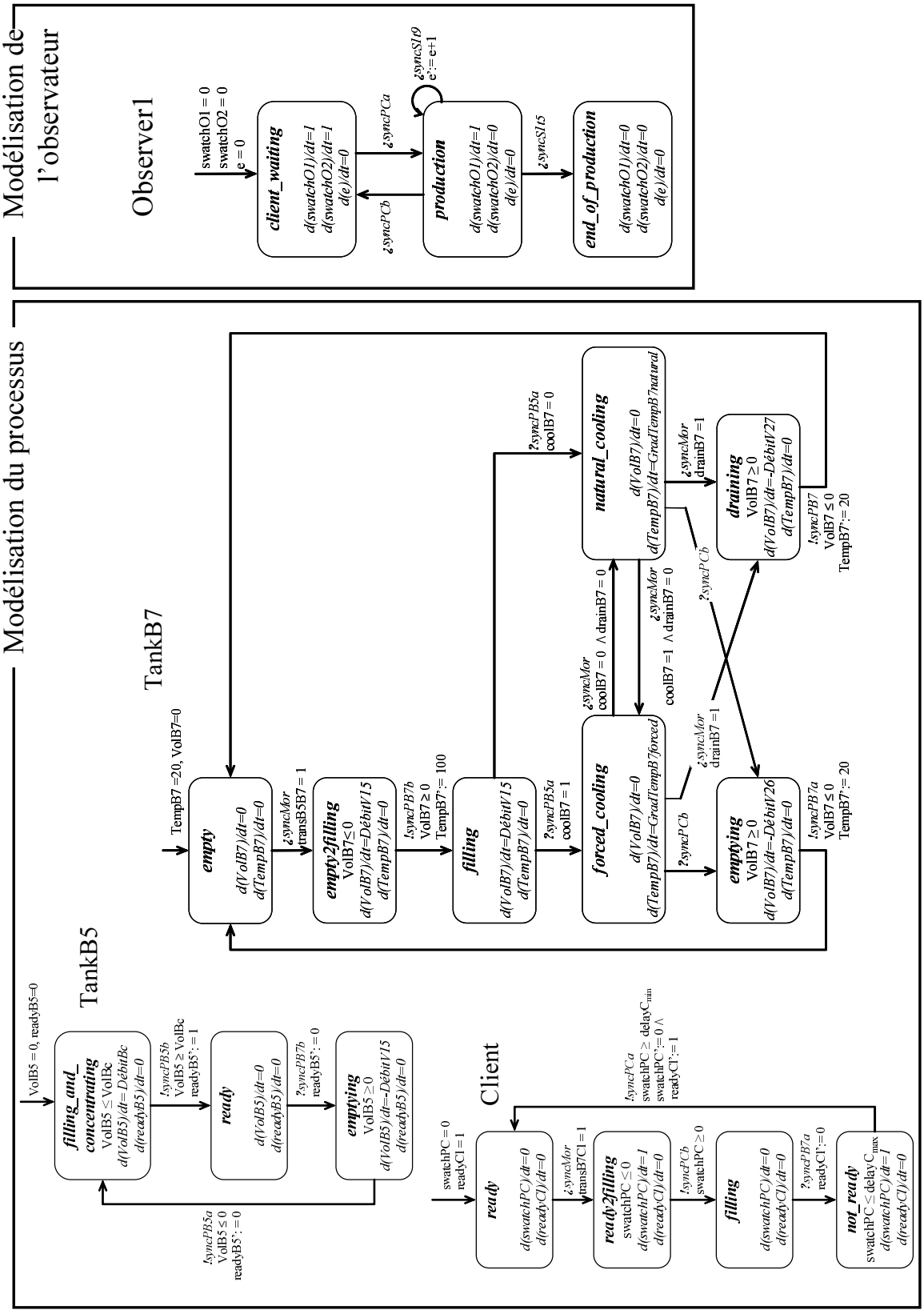



FIG. 4.46 – Réseau d'$HIOA_{TS}$ du processus et module $HIOA_{TS}$ de l'observateur

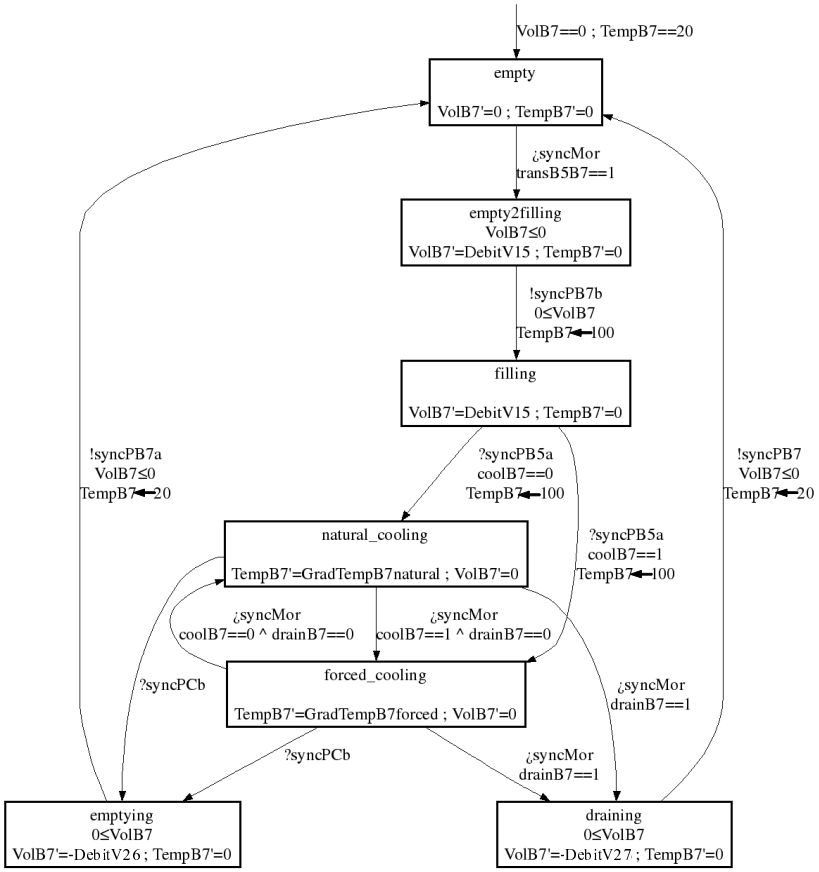



FIG. 4.47 – Modèle $HIOA_{TS}$ du module « TankB7 » du processus avant traduction

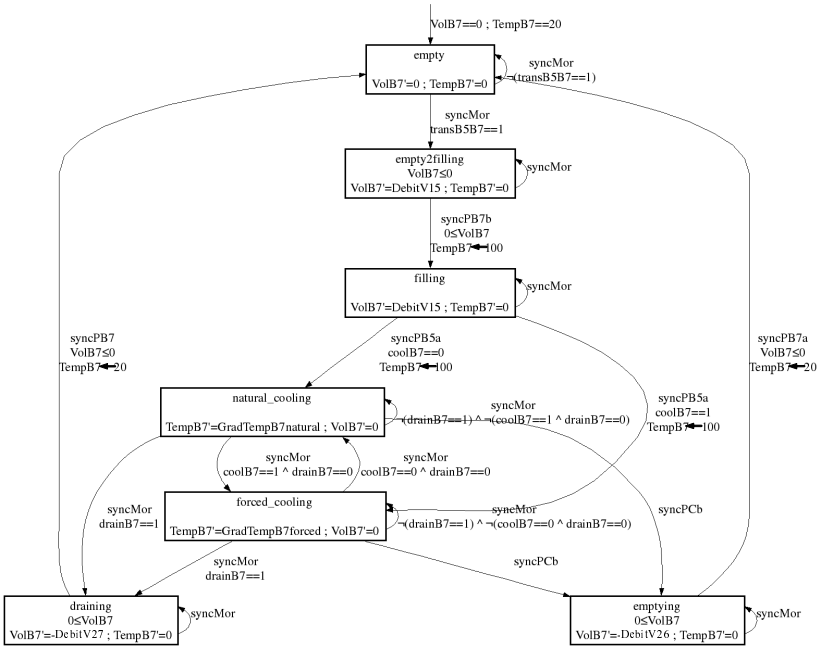



FIG. 4.48 – Modèle $HIOA_{TS}$ du module « TankB7 » du processus après traduction

– Le délai d'attente du prochain Client $delayC$ compris dans l'intervalle $[delayCmin, delayCmax]$.

Nous allons évaluer différents critères quantitatifs afin de discriminer entre différentes stratégies de commande. On rappelle que, pour une production de lots déterminée par « nbatch », ces critères quantitatifs sont les suivants :

– Durée totale de production (vitesse de production de la demande de lots),
– Nombre de lots rejetés (indice de rejet),
– Durée cumulée des attentes clients (niveau de satisfaction du client).

Ces propriétés, sous la forme de critères quantitatifs, seront observées à la fin de la production, c'est-à-dire dans la situation end_of_production de l'automate observer1, par les variables :

– $swatchO1$ pour la durée totale de production,
– $e$ pour le nombre de lots rejetés,
– $swatchO2$ pour la durée cumulée des attentes clients.

Dans nos investigations, nous allons exploiter l'analyse paramétrique proposée par le model-checker PHAVer. Ainsi, non seulement la propriété pourra inclure des valeurs quantitatives, mais le résultat de la vérification sera exprimé sous la forme d'une fonction d'un ou plusieurs paramètres.

### Première investigation : Étude de la vitesse maximale de production des lots

Pour cette première investigation, nous allons analyser la première stratégie de commande (voir fig. 4.31) afin de calculer la vitesse maximale de production en fonction du nombre de lots à produire. Soient

– $Rg_a$ la région atteignable par le modèle à partir de sa situation initiale, et
– $Rg_1$ la région où $Observer1.end\_of\_production$ est active.

On cherche à déterminer la région « projection sur la durée de production globale $swatchO1$ de $Rg_a \cap Rg_1$ » en fonction du paramètre formel $nbatch$ (nombre de lots à produire) pour un délai d'attente du client nul ($delaiC = 0$). Le résultat obtenu avec PHAVer est un ensemble de points situés sur la droite d'équation $y = 37,196 \cdot x + 13,852$ est présenté dans la figure 4.49. On peut observer que la droite ne passe pas par l'origine. Cela s'explique par une durée nécessaire à la mise en production avant d'atteindre le régime établi pour la production des lots. Ce résultat du « model-checker » PHAVer nous permet de donner la vitesse de production maximale que peut atteindre le système en fonction du nombre de lots demandé (nbatch). Ainsi, par exemple, pour nbatch=4, le temps de production calculé est de $162,64$ min.

### Deuxième investigation : Étude comparée des trois stratégies de commande

Connaissant la vitesse maximale de production, on va s'intéresser au comportement des trois stratégies de commande quand le client n'est pas toujours présent lors de la production d'un lot à la température souhaitée. Soient,

– $Rg_a$ la région atteignable par le modèle à partir de sa situation initiale, et
– $Rg_2$ la région où $Observer1.end\_of\_production$ est active et $e = 0$.

On cherche maintenant à déterminer la région « projection sur la durée de production $swatchO1$ et sur le nombre de lots rejetés $e$ de $Rg_a \cap Rg_2$ » en fonction du paramètre

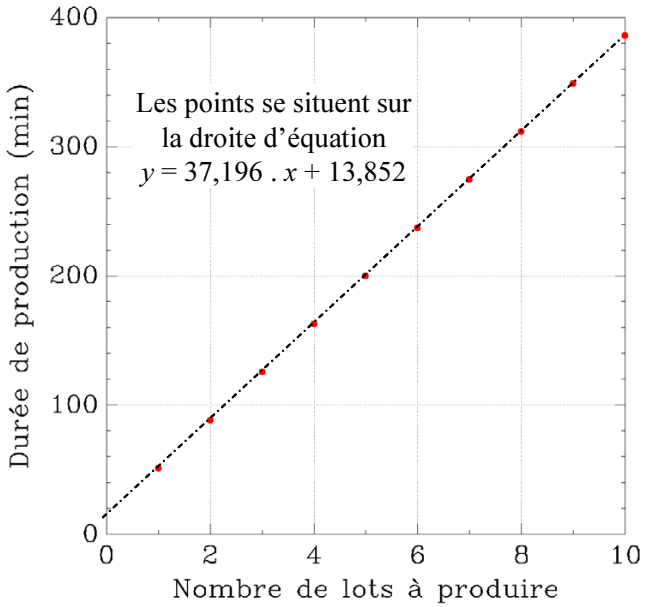



FIG. 4.49 – Région atteignable par la stratégie 1 avec des clients toujours disponibles (délai d'attente nul)

formel $delayC$ (durée d'attente du prochain client) pour une production de 4 lots. Ici, la durée du paramètre $delayC$ est déterministe, c'est-à-dire que $delayC = delayCmin = delayCmax$. Pour chacune des trois stratégies de commande, nous cherchons, dans la région projetée, la durée d'attente minimale du prochain client pour laquelle on n'a pas de produit à rejeter. Le résultat du model-checker PHAVer pour les trois stratégies de commande est présenté dans la figure 4.50.

La figure 4.50a montre que :

- pour la stratégie 1 (refroidissement forcé immédiat et permanent), on observe que, jusqu'à un délai d'attente du prochain client ($delayC$) inférieur ou égal à $33,34$ min cette stratégie ne perd pas de lots et conserve la même durée de production de $162,64$ min pour nbatch=4 lots, ce qui correspond à la vitesse maximal de production déterminée dans l'investigation précédente,
- pour la stratégie 2 (refroidissement forcé immédiat puis arrêt à une limite de température réglée à 60°C), tant que $delayC$ est inférieur ou égal à $28,68$ min, la vitesse maximale de production est conservée avec toujours aucun lot rejeté. Par contre, au-delà de $28,8$ min et jusqu'à $57,83$ min cette stratégie ne perd pas de lot mais la durée de production augmente. En effet, si le client n'est pas présent lorsque le lot atteint la température de coupure LimitTempB7 réglée à 60°C, le refroidissement forcé est arrêté en attendant la présence d'un client, ce qui provoque le ralentissement de la production,
- pour la stratégie 3 (refroidissement naturel en absence de client), $delayC$ doit être inférieur ou égal à $23,34$ min pour conserver la vitesse maximale de production et ne pas perdre de lot. Passée cette valeur, le refroidissement naturel ralentit la production.

La figure 4.50b présente les trois stratégies superposées. On peut en conclure que la stratégie 1 est adéquate lorsque la durée d'attente du prochain client ne dépasse pas $33,34$ min. Si la durée d'attente du client augmente en pouvant dépasser $33,34$ min

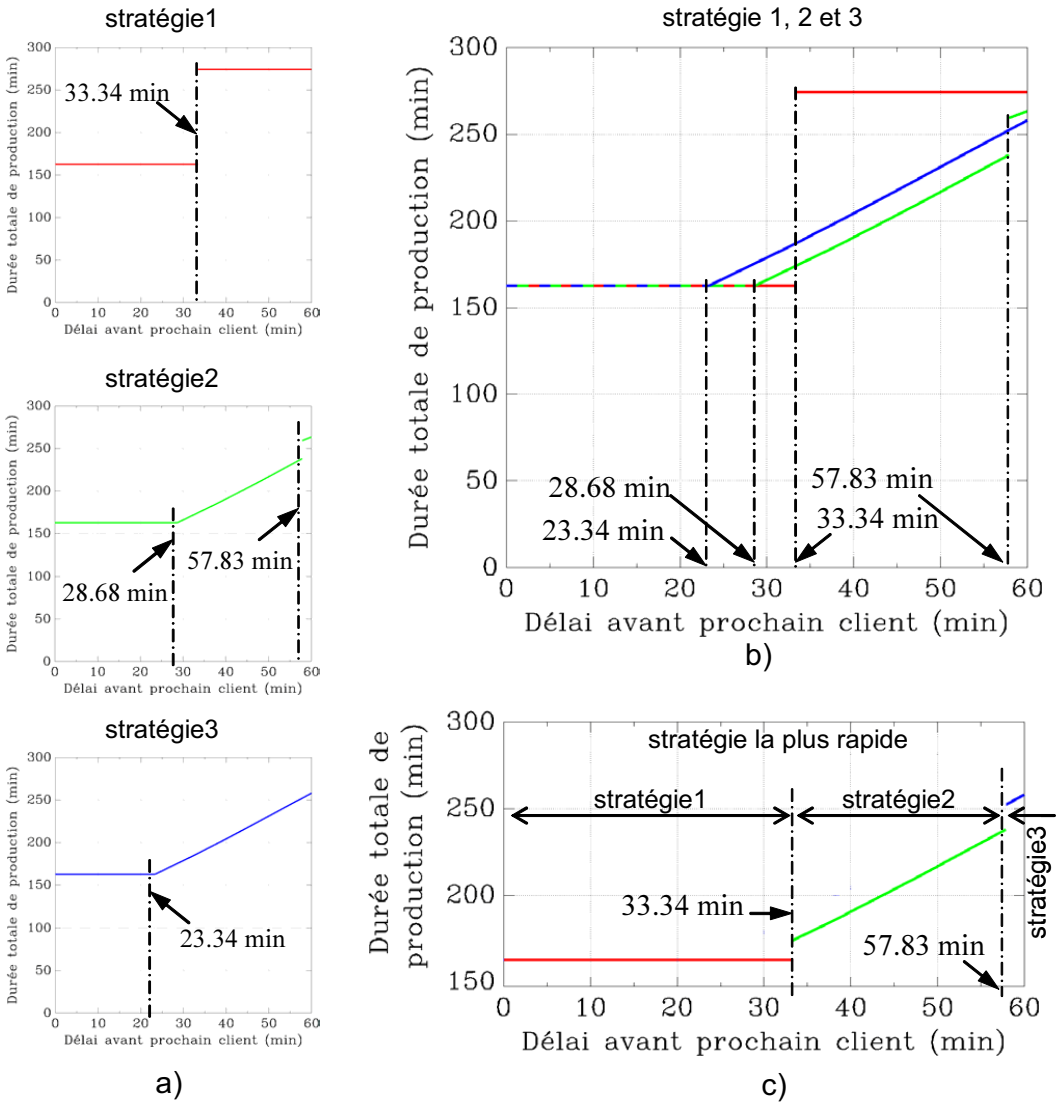



FIG. 4.50 – Régions atteignables par les trois stratégies de commande : durée de production de 4 lots en fonction de la durée d'attente du prochain client.

sans dépasser 57, 83 min, alors la stratégie 2 fournit la plus grande vitesse de production avec aucune perte de lot, cependant, le temps total de production augmente. Pour des durées d'attente du prochain client au-delà de 57, 83 min et jusqu'à 60 min, la stratégie 3 devient plus efficace (voir figure 4.50c).

En résumé, la preuve apportée pour cette deuxième investigation est donc :

$$\left.\begin{array}{l} \text{stratégie1 \& } delayC \in [0, 33.34]min \\ \text{stratégie2 \& } delayC \in [33.34, 57.83]min \& LimitTempB7 = 60^{\circ}C \\ \text{stratégie3 \& } delayC \in [57.83, 60]min \end{array}\right\} \Rightarrow \begin{array}{l} e = 0 \ \& \\ \text{vitesse max} \\ \text{de production} \end{array}$$

**Troisième investigation : Étude de l'impact de la température de coupure dans la stratégie 2**

Nous allons maintenant nous intéresser au comportement de la stratégie 2. On remarquera que, pour $LimitTempB7 = 25^{\circ}C$ et pour $LimitTempB7 = 100^{\circ}C$, la stratégie 2 se comporte respectivement comme la stratégie 1 et la stratégie 3. Soient :

- $Rg_a$ la région atteignable par le modèle à partir de sa situation initiale, et
- $Rg_1$ la région où $Observer1.end\_of\_production$ est active

On calcule la région « projection sur la durée de production *swatchO1* et sur la durée cumulée des attentes du client *swatchO2* de $Rg_a \cap Rg_1$ » en fonction du paramètre formel $LimitTempB7$ (température de coupure) pour une production de 4 lots. La durée du délai $delayC$ est cette fois non déterministe, c'est-à-dire qu'elle subit une importante variation $jitterCl = \pm 20$ %. Ainsi, $delayC$ est défini comme suit :

$$delayCl \in [\underbrace{(delayC - (delayC * jitterCl))}_{delayCmin}, \underbrace{(delayC + (delayC * jitterCl))}_{delayCmax}]$$

Nous allons tester différentes valeurs pour l'attente du prochain client (voir figure 4.51). Nous cherchons un bon compromis entre le cumul des attentes des clients et la durée totale de production en fonction de la limite de coupure de température sans perdre de lots ($e = 0$).

On peut observer que :

- avec $delayC = 25\text{min} \pm 20\%$, on observe qu'à vitesse maximale de production ($LimitTempB7 = 25^{\circ}\text{C}$), nous n'avons aucune perte de lots ($e = 0$) et l'indicateur de cumul des attentes est compris dans $[57min, 87min]$. Lorsque l'on dépasse les 25°C et jusqu'à une température de 50°C, la vitesse de production reste constante avec toujours $e = 0$. Quant à $LimitTempB7 \geq 50^{\circ}\text{C}$, à ce seuil, le cumule des attentes des clients et la durée totale de production augmentent.
- avec $delayC = 30\text{min} \pm 20\%$, lorsque $LimitTempB7 = 25^{\circ}\text{C}$, 3 lots peuvent être rejetés, provoquant une augmentation du cumul de l'attente des clients, et par conséquent du temps total de production. Par contre, lorsque $LimitTempB7 \geq 33^{\circ}\text{C}$, on réduit le nombre de lots rejetés à 0, provoquant une diminution du cumul de l'attente des clients et du temps total de production.
- avec $delayC = 35\text{min} \pm 20\%$, on observe que plus l'attente du prochain client augmente, plus la température de coupure qui garantit l'absence de perte de lots est grande. En effet, il faut régler la température de coupure à $LimitTempB7 \geq 44^{\circ}\text{C}$ pour que la perte de lots soit nulle et que le cumul des attentes des clients ainsi que la durée totale de production diminuent.

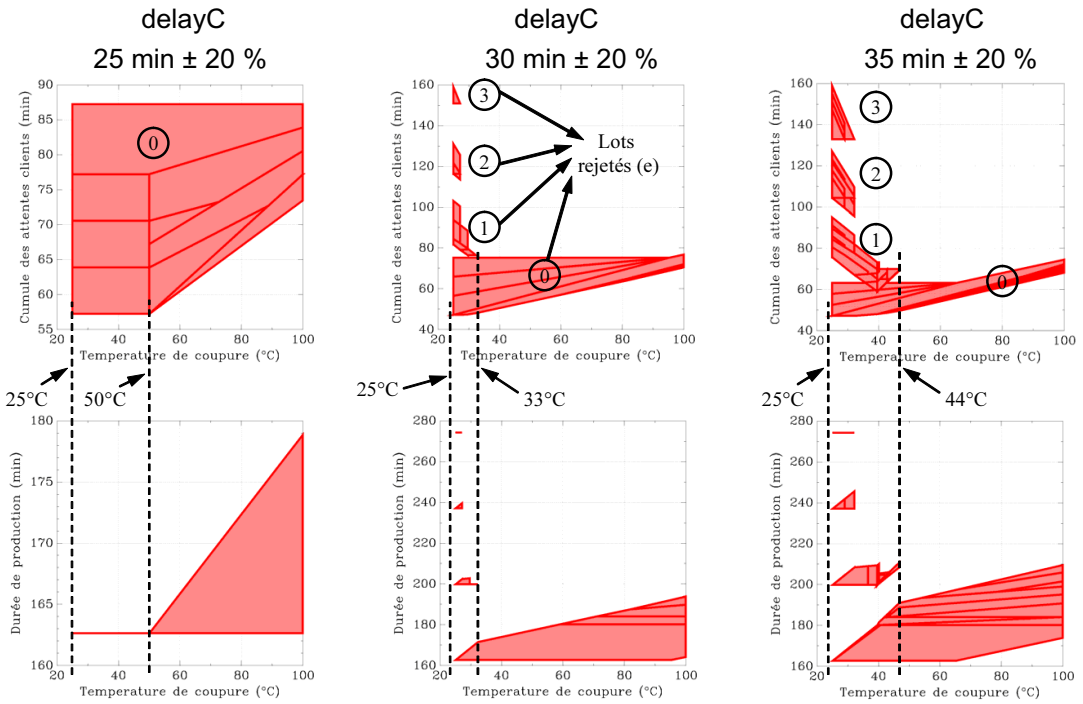



FIG. 4.51 – Région atteignable montrant les indicateurs (satisfaction du client (cumul des attentes du client) et la vitesse de production) pour la stratégie 2 en fonction de la température de coupure (4 lots).

En résumé, pour la production de 4 lots, si l'on impose l'absence de perte de lots comme premier critère, les réglages qui minimisent à la fois le temps total de production et le cumul des attentes des clients sont : $LimitTempB7 \in [25, 50]$ pour $delayC = 25\text{min} \pm 20\%$, $LimitTempB7 = 33^{\circ}C$ pour $delayC = 30\text{min} \pm 20\%$ et $LimitTempB7 = 44^{\circ}C$ pour $delayC = 35\text{min} \pm 20\%$.

### 4.3.6 Conclusion

Le traitement de cette deuxième étude de cas a montré la pertinence de la démarche de modélisation proposée dans le chapitre 3 pour les systèmes automatisés à contrôleur logique. Seuls les automates concernant le processus (TankB5, TankB7 et Client) ont nécessité une conception à partir d'une feuille blanche, tous les autres automates ont été construits par instanciation d'un modèle générique. Après la simplification du modèle TankB5, l'instrumentation logicielle développée à cet effet a permis de traduire le réseau d'$HIOA_{TS}$ en réseau $HIOA$ puis de générer le fichier d'entrée pour le « model-checker ». Au travers de trois investigations successives sur le modèle, nous avons pu découvrir la grande richesse des analyses quantitatives permises avec un automate hybride. La modélisation par un modèle hybride du processus nous a permis de quantifier trois indicateurs de performances afin de comparer différentes stratégies de commande et, de cette manière, d'évaluer leur impact sur les résultats de la production.

## 4.4 Conclusion du chapitre 4

Deux études de cas représentatives des systèmes automatisés à contrôleur logique ont été traitées dans ce chapitre. De très nombreux enseignements sont à en tirer.

Les premiers retours concernent le cadre de modélisation proposé dans le chapitre 3. La structure modulaire générique proposée a été suivie à la lettre dans les deux études, à l'exception des modules dédiés aux actionneurs qui ne se sont pas révélés utiles au modélisateur. En effet, en partant d'une feuille blanche ou presque pour le modèle du processus, il est paru naturel pour l'étude sur l'axe de déplacement d'intégrer toute la chaîne d'énergie dans un seul module (les contacteurs, le moto-réducteur, la transmission par courroie et le convoyeur) et, pour le système de production par lot, d'intégrer les vannes directement dans le modèle des cuves. Les modules d'actionneurs verront leur intérêt augmenté avec le développement d'une bibliothèque d'actionneurs types par secteur applicatif (manufacturier, pétrochimie, etc.) et par technologie (électrique, pneumatique, etc.). Si la structure générique du réseau d'$HIOA_{TS}$ a rempli son office pour la modélisation, une inquiétude subsistait quant à la production d'un modèle peu compact et peut-être impossible à analyser par le model-cheker. Le réseau résultant de l'instanciation contient un module par variable partagée, un module par sortie de l'API, un module par détecteur, et un module par détection de seuil sur entrée d'API. L'expérience a montré que cela n'était pas rédhibitoire. Tout d'abord, les modules des variables et les modules des sorties n'étant composés que d'une seule situation, ils n'ont pas d'impact sur le nombre de situations de l'automate composé par le model-cheker. Ensuite, les modules des capteurs et les modules des détecteurs de seuil sont fortement synchronisés avec le ou les modules du processus qu'ils surveillent. Dans ces conditions, l'impact sur l'espace des situations du modèle global reste limité. La généricité et la structure modulaire ont donc été bénéfiques pour la modélisation, sans nuire à l'explosion combinatoire du réseau d'automates.

Les retours des deux études de cas portent également sur la taille des systèmes appréhendables par modélisation hybride. La première étude sur l'axe a montré qu'avec un processus composé de quatre modules eux-mêmes composés de trois situations et un contrôleur avec quatre SFC totalisant quinze étapes, la composition de l'automate explose (plus de 3 Go nécessaires à PHAVer). La restriction des modules des SFC aux seules étapes utiles a permis l'analyse avec PHAVer, mais la vraie source de l'explosion combinatoire provient de la modélisation conjointe du contrôleur et du processus. Ils fonctionnent de manière très asynchrone lorsque le moniteur « monitor1 » est retenu : les évolutions du moniteur ne sont pas synchronisées avec les évolutions des capteurs mais uniquement avec son cycle interne. L'usage du moniteur « monitor1 », qui est indispensable aux études d'impacts de la performance temporelle du contrôleur sur les qualités de service obtenues du système, doit se limiter à des analyses sur une fonction de commande partielle (un sous-ensemble des étapes des SFC). Afin de limiter l'asynchronisme des modules du contrôleur avec les modules du processus, le moniteur « monitor2 » assure une évolution synchrone entre les détecteurs et le contrôleur. La notion de temps de réaction du contrôleur a été supprimée tout en conservant la logique de l'ordre des exécutions dans le contrôleur. La seconde étude de cas a été traitée avec le moniteur « monitor2 » et, bien que le modèle du processus soit plus volumineux que pour l'axe et que le SFC soit composé de huit étapes, l'automate a été composé en 7 min,

il comprend 6652 situations et 23846 transitions et il occupe un espace mémoire de 965 Mo. Même si une abstraction a été proposée sur un module du processus pour réduire la taille de l'automate composé, sa taille est manipulable par un simple calculateur de bureautique avec 4 Go de mémoire. Compte tenu de notre choix de technique d'analyse, la vérification formelle par model-checking, nous jugeons la taille des deux études de cas très réaliste et encouragante. Pour envisager le traitement de cas de plus grande taille, le recours à un expert pour rendre le modèle plus compact après instanciation nous semble être une première voie. L'utilisation d'un serveur de calcul autorisant un espace mémoire de plus de 3 Go pour un programme est également une autre voie pour l'étude de plus grand cas d'étude puisque le temps de composition de l'automate s'est révélé peu coûteux.

Enfin, les enseignements relatifs à l'analyse de propriétés quantitatives de ces deux études de cas sont les suivants. La première étape de toutes nos analyses est le calcul de la région atteignable par le modèle à partir d'une situation donnée. L'algorithme de recherche de la région atteignable n'a pas posé de problème de convergence, probablement parce que nos deux modèles ont un sens physique. Soit l'algorithme s'est terminé en moins de 10 min, soit le seuil des 3 Go a été atteint par le model-cheker (limite pour un programme sur notre calculateur). C'est la complexité due à l'incertitude du comportement de certains modules (la variabilité de la vitesse, la gigue sur le temps de cycle du moniteur, l'incertitude sur le délai d'attente d'un client) qui s'est révélée être la plus redoutable au moment de l'analyse. L'introduction de non déterminisme dans le modèle augmente le nombre de nouveaux polyèdres atteignables après chaque itération de l'algorithme à la manière d'une réaction en chaîne. On peut cependant remarquer que deux variables subissent simultanément du non déterminisme dans le modèle de la première étude de cas, et qu'un très fort indéterminisme d'une étendue de 40% est appliqué à la variable delayC de la deuxième étude de cas. Malgré ces indéterminismes, le calcul de la région atteignable a pu être mené à son terme. Cela nous a permis de décrire avec encore plus de réalisme les deux études. L'introduction d'indéterminismes permet également, dans un contexte de sûreté de fonctionnement, d'analyser l'impact de variabilités sur la qualité du service rendu : la robustesse du système à des perturbations. Finalement, la richesse des investigations que nous avons pu mener sur les deux études est à elle seule une véritable justification de l'intérêt d'utiliser un modèle hybride pour analyser des contrôleurs logiques en interaction avec un processus.

# *Conclusion*

Ces travaux se sont placés dans le contexte de la sûreté de fonctionnement des systèmes industriels, et tout particulièrement des systèmes critiques. L'objectif SdF doit être présent à tous les stades de la vie du système : de sa conception à son démantèlement, en passant par sa réalisation et son exploitation. Notre apport se situe en phase de conception, lorsque des choix doivent être validés vis-à-vis de la qualité du service rendu par le système. Parmi les choix les plus engageants pour la suite de la vie du système, il y a celui qui consiste à retenir un contrôleur logique pour régir le fonctionnement d'un processus. Bien que les processus soient par nature hybrides, mélangeant des dynamiques continues (mouvement, remplissage, etc.) et discrètes (fin de course, changement de mode, etc.), des raisons économiques poussent les concepteurs à choisir des contrôleurs logiques. Le contrôleur doit alors piloter le processus et fournir une qualité de service qu'il n'est lui-même pas apte à appréhender à cause de l'abstraction logique au travers de laquelle il perçoit le processus et son environnement. La qualité du service, qui ne pourra donc pas être garantie en ligne (durant le fonctionnement du système), sera garantie par une approche hors ligne (en phases de conception et de réalisation). Dans ce contexte, et en se limitant aux qualités de service prenant la forme de propriétés temporelles, de nombreux travaux ont exploré la voie de la vérification formelle par Model-Checking. Notre contribution a consisté à évaluer les apports de la vérification formelle de la qualité de service des systèmes automatisés avec un contrôleur logique. À cette fin, le système est modélisé par un réseau d'automates hybrides et il est vérifié par un model-checker polyédrique.

Pour atteindre cet objectif, nous avons pris le parti d'utiliser un model-checher existant et de focaliser notre attention sur deux problématiques scientifiques : la modélisation par automates hybrides de systèmes complexes et la modélisation et vérification par model-checker polyédrique de la qualité de service.

Les origines de la complexité des systèmes automatisés sont : le nombre de grandeurs échangées entre le contrôleur et le processus auquel s'ajoutent les grandeurs internes au processus, le nombre de fonctions du système, le nombre d'états de la commande logique et le nombre de comportements non déterministes pris en compte. Pour prendre en compte la complexité, une approche modulaire inspirée des blocs fonctionnels de la norme CEI 61499 a été retenue. Elle a nécessité l'introduction d'une extension au formalisme des automates hybrides à entrées/sorties utilisés dans le model-checker PHAVer pour enrichir la sémantique des synchronisations de transitions (chapitre 2). Ce formalisme étendu, que nous avons appelé $HIOA_{TS}$ (Automate Hybride à Entrées/Sorties et à Synchronisations Typées), permet à un automate d'être émetteur, récepteur bloquant ou récepteur non bloquant d'un événement de synchronisation. Une

seconde contribution à la prise en compte de la complexité des systèmes automatisés est le développement d'une structure générique du modèle d'un contrôleur et du processus qu'il pilote. Cette structure s'appuie sur une bibliothèque d'automates génériques dont les mécanismes d'instanciation ont été formalisés dans le chapitre 3. On peut ainsi résumer notre contribution à la difficile modélisation des systèmes complexes en un apport théorique, le formalisme $HIOA_{TS}$, et en un apport méthodologique, la structure générique du modèle et la bibliothèque de modules prêts à être instanciés.

Pour notre contribution à la modélisation et à la vérification de la qualité de service par le biais de propriétés quantitatives, nous avons retenu une démarche basée sur le traitement de cas représentatifs de la complexité des systèmes et de la diversité des qualités de service attendues. L'élément central de l'analyse formelle d'un modèle hybride par un model-checker polyédrique est la région atteignable par ce dernier depuis une situation donnée. Cette région est très riche en informations exploitables pour la vérification des contrôleurs logiques, qui ont donné lieu à plusieurs investigations dans le chapitre 4. Les grandeurs qui ne seront pas accessibles au contrôleur pendant son fonctionnement, à cause de l'abstraction logique, sont rendues accessibles à l'analyste au moment de la vérification grâce à la modélisation de la dynamique continue du processus par les automates hybrides. L'analyste a alors tous loisirs de valider ses choix, en particulier vis-à-vis de la SdF. Les résultats opérationnels obtenus sont encourageants en ce qui concerne le temps de calcul nécessaire au model-checker . Moins de 20 minutes sont nécessaires à l'analyse d'un modèle (composition du réseau d'automates plus calcul de la région atteignable) avec une machine équipée d'un processeur Pentium4 à 2,4 GHz et de 4 Go de mémoire vive. Les seules difficultés opérationnelles concernent la mémoire nécessaire pour manipuler à la fois l'automate composé et la région atteignable.

Les premières perspectives de ce travail ont pour objet de repousser la limitation de la taille des problèmes traitables à cause de la demande croissante en mémoire du model-checker . Une première voie consiste à introduire des règles expertes pour l'optimisation du modèle instancié en vue de sa vérification. Une deuxième voie d'exploration consiste en la construction de la région atteignable par parties. Par exemple, pour la deuxième étude de cas, on peut envisager d'étudier les unes après les autres les régions atteignables avec aucun lot rejeté (e=0), avec un lot rejeté (e=1), etc. Et enfin, une dernière voie d'exploration est l'utilisation de techniques d'approximation sur les polyèdres du model-checker en limitant le nombre de bits pour leur représentation en mémoire. Selon la nature de la propriété quantitative, une sur-approximation ou une sous-approximation sera retenue.

Une perspective plus ambitieuse s'offre à nous pour prolonger les apports théoriques de ces travaux. L'approche modulaire de modélisation que nous avons proposée est une approche « à plat ». L'introduction d'une extension hiérarchique au formalisme $HIOA_{TS}$ est probablement une voie d'avenir pour repousser le niveau de complexité appréhendable par une classe d'automates hybrides, et envisager une analyse formelle descendante.

# *Références Bibliographiques*


[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid Automata : An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer, 1993.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theorical Computer Science*, 126 :183–235, 1994.

[BBF+01] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking : 1020 states and beyond. *Information and Computation, 1992*, 98(2) :142–172, 1992.

[BHLE04] N. Bauer, R. Huuck, B. Lukoschus, and S. Engell. A unifying semantics for sequential function charts. In *Integration of Software Specification Techniques for Applications in Engineering, Berlin, 2004, Series LNCS 3147*, pages 400–418. Springer, 2004.

[BMF02] E. Brinksma, A. Mader, and A. Fehnker. Verification and optimization of a plc control schedule. In *International Journal on Software Tools for Technology Transfer (STTT)*, volume ISSN 1433–2779, pages 21–33, 2002.

[BS07] Israel Barragan-Santiago. *Elaboration de propriétés formelles de contrôleurs logiques à partir d'analyse prévisionnelle par Arbre de défaillances*. PhD thesis, Ecole Normale Supérieure de Cachan, 2007.

[CC95] P. Cousot and R. Cousot. Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game-theoretic form, invited paper. In P. Wolper, editor, *Proceedings of the Seventh International Conference on Computer Aided Verification, CAV '95*, pages 293–308, Liège, Belgium, Lecture Notes in Computer Science 939, 3–5 July 1995. Springer-Verlag, Berlin, Germany.

[CGS07] M.P. Cabasino, A. Giua, and C. Seatzu. Marking estimation of petri nets with arbitrary transition labeling. In *1st IFAC workshop on Dependable Control of Discrete Systems*, pages 109–114, Paris, France, 2007.

[CL99] Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*, chapter 2, pages 83–91. Kluwer Academic Publishers, 1999.

[DA92] R. David and H. Alla. *Petri Nets and Grafcet : Tools for Modelling Discrete Event Systems*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1992.

[DLJO06] B. Denis, J.-J. Lesage, and Z. Juarez-Orozco. Performance verification of discrete event systems using hybrid model-checking. In *2nd IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'06)*, pages 365–370, Alghero (Italy), June 2006.

[DSLR01] O. De Smet, J.J. Lesage, and J.M. Roussel. Formal verification of industrial control systems. In *Proc. 10th IFAC Symp. Information Control Problems in Manufacturing (INCOM'2001)*. CDRom paper, 2001.

[EFS02] S. Engell, G. Frehse, and E. Schnieder. *Modelling, Analysis, and Design of Hybrid Systems*. Springer, 2002.

[FB05] G. Frensel and P.M. Bruijn. From Grafcet to hybrid automata. In *Proceeding of Information Control in Manufacturing (INCOM'05)*, pages 47–52, 2005.

[FL00] G. Frey and L. Litz. Formal methods in PLC programming. In *Proceedings of the IEEE SMC 2000*, pages 2431–2436, Nashville, October 2000.

[FLS02] J.M. Faure, J.-J. Lesage, and C. Schnakenbourg. Towards IEC 61499 function blocks diagrams verification. *IEEE Int. Conference on Systems, Man and Cybernetics (SMC02)*, 2002.

[Fre05a] Goran Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. PhD thesis, Radboud Universiteit Nijmegen, October 10 2005.

[Fre05b] Goran Frehse. PHAVer : Algorithmic Verification of Hybrid Systems Past HyTech. In *HSCC*, pages 258–273, 2005.

[FRV06] L. Ferrarini, M. Romano, and C. Veber. Automatic Generation of AWL Code from IEC 61499 Applications. *IEEE International Conference on Industrial Informatics*, pages 25–30, 2006.

[Gou07] Vincent Gourcuff. *Représentations formelles efficaces pour l'aide à la certification de contrôleurs industriels*. PhD thesis, Ecole Normale Supérieure de Cachan, 2007.

[Hen96] T.A. Henzinger. The theory of hybrid automata. In *Proc. 11th IEEE Symp. Logic in Computer Science (LICS'96), New Brunswick, N.J. USA*, pages 278–292, 1996.

[HHWT97] T.A. Henzinger, P.H. Ho, and H. Wong-Toi. HYTECH : a model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1) :110–122, 1997.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Huu05] R. Huuck. Semantics and analysis of instruction list programs. In *Second Workshop on Semantic Foundations of Engineering Design Languages (SFEDL 2004)*, pages 3–18. Electronic Notes in Theoretical Computer Science 115, Elsevier, January 2005.

[IEC93] IEC (International Electrotechnical Commission). *IEC Standard 61131-3 : Programmable controllers - Part 3*, 1993.

[IEC98] IEC (International Electrotechnical Commission). *IEC Standard 61131-3 : Sécurité fonctionnelle des systèmes électriques / électroniques / électroniques programmables relatifs à la sécurité*, 1998.

[IEC04] IEC (International Electrotechnical Commission). *IEC Standard 61499 : Function blocks for industrial-process measurement and control systems*, 2004.

[ISO89] ISO (International Organization of Standardization). *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*, 1989.

[ISO07] ISO : Organisation Internationale de Normalisation. *ISO/IEC Guide 99 : 2007. Vocabulaire international de métrologie – Concepts fondamentaux et généraux et termes associés (VIM)*, 2007.

[KEH+98] S. Kowalewski, S. Engell, R. Huuck, Y. Lakhnech, B. Lukoschus, and L. Urbina. Using model-checking for timed automata to parameterize logic control programs. In *Proc. 8th European Symposium on Computer Aided Process Engineering*, Brugge, Belgium, May 24–27 1998.

[KER+99] S. Kowalewski, S. Engell, V. Roßmann, R. Huuck, Y. Lakhnech, B. Lukoschus, and L. Urbina. Integrating timed condition/event systems and timed automata to the verification of hybrid systems. *Journal of Parallel and Distributed Computing Practices 1998*, 1999.

[KES99] S. Kowalewski, S. Engell, and O. Stursberg. Verification of logic controllers for continuous plants. In *Advances in Control : Highlights of ECC'99*, pages 345–389. Springer, 1999.

[KGR94] S. Kowalewski, R. Gesthuisen, and V. Roßmann. Model-based verification of batch process control. In *SMC (USA)*, pages 331–336, 1994.

[KJH06] H. Kaghazchi, R. Joyce, and D. Heffernan. Function Blocks for Fieldbus Diagnostics. pages 322–327, 2006.

[Kow98] S. Kowalewski. Description of case study CS1 : Experimental batch plant. July 1998. http ://www-verimag.imag.fr/VHS/main.html.

[KY06] Pavel Krcál and Wang Yi. Communicating Timed Automata : The More Synchronous, the More Difficult to Verify. In *CAV*, pages 249–262, 2006.

[LCRRL99] S. Lampérière-Couffin, O. Rossi, J.-M. Roussel, and J.-J. Lesage. Formal validation of PLC programs : a survey. In *European Control Conference 1999 (ECC'99), Karlsruhe, Germany, Aug.-Sep. 1999*, 1999. proceedings on CD-ROM, communication 741.

[LDL07] S. Limal, B. Denis, and J.-J. Lesage. Formal verification of redundant media extension of Ethernet PowerLink. In *Emerging Technologies and Factory Automation. ETFA. IEEE Conference on*, pages 1045–1052, 2007.

[LES05] S. Lohmann, S. Engell, and O. Stursberg. Verification of embedded supervisory controllers considering hybrid plant dynamics. In *International Journal of Software Engineering and Knowledge Engineering*. World Scientific Publishing Company, 2005.

[LG06] Marie-Anne Lefebvre and Hervé Guéguen. Hybrid abstraction of affine systems. *Nonlinear Analysis*, 65(6) :1150–1167, 2006.

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2) :134–152, 1997.

[LSE06] S. Lohmann, O. Stursberg, and S. Engell. Systematic design of logic controllers for processing plants starting from informal specifications. In *Proc. 16th European Sym. On Computer Aided Process Engineering*, 2006.

[LSV03] Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O automata. *Inf. Comput.*, 185(1) :105–157, 2003.

[LT87] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. of the 6th Annual ACM Symposium on Principles of Distributed Computing, PODC'87*, pages 137–151, Vancouver, BC, Canada, August 1987.

[Mac06] Jose Machado. *Influence de la prise en compte d'un modèle du processus en vérification formelle de Systèmes à Evénements Discrets*. PhD thesis, Ecole Normale Supérieure de Cachan (France) et Université de Minho (Portugal), 2006.

[Mad00] A. Mader. A classification of PLC models and applications. In *WODES 2000 : 5th Workshop on Discrete Event Systems*, pages 239–247, August 21-23 2000.

[MBG+05] H. Bel Mokadem, B. Berard, V. Gourcuff, J.-M. Roussel, and O. De Smet. Verification of a timed multitask system with uppaal. In *10th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'05, Catania, Italy, Septembre*, 2005. proceedings on CD-ROM, communication CF-000606.

[McM93] K. L. McMillan. *Symbolic model checking*. Kluwer Academic Publishers, 1993.

[MDL06] J. Machado, B. Denis, and J.-J. Lesage. A generic approach to build plant models for DES verification purposes. In *8th International Workshop On Discrete Event Systems (WODES'06)*, pages 407–412, Ann Arbor (USA), July 2006.

[Meu06] Pascal Meunier. *Evaluation de performance d'architectures de commande de systèmes automatisés industriels*. PhD thesis, Ecole Normale Supérieure de Cachan, 2006.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[Moo56] E.F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, 34 :129–153, 1956.

[MPBC92] I. Moon, G.J. Powers, J.R. Burch, and E.M. Clarke. Automatic verification of sequential control systems using temporal logic. *American Institute of Chemical Engineers Journal (AIChE'92)*, 38(1) :67–75, 1992.

[MW99] A. Mader and H. Wupper. Timed automaton models for simple programmable logic controllers. In *Proc. of 11th Euromicro Conference on Real-Time Systems (ECRTS'99)*, pages 114–122, York - UK, June 1999.

[NLG06] Othman Nasri, Marie-Anne Lefebvre, and Hervé Guéguen. Reachability computation for uncertain planar affine systems using linear abstractions. In C.G. Cassandras, A. Giua, C. Seatzu, and J. Zaytoon, editors, *ADHS06 : 2nd IFAC Conference on Analysis and Design of Hybrid Systems*, pages 118–123, Alghero (Italy), 2006. Elsevier.

[PF05] S. Panjaitan and G. Frey. Functional Design for IEC 61499 Distributed Control Systems using UML Activity Diagram. pages 64–70, 2005.

[RD02] J.M. Roussel and B. Denis. Safety properties verification of ladder diagram programs. *Journal européen des systèmes automatisés*, 36(7) :905–917, 2002.

[RH95] M. Raush and H.-M. Hanisch. Net condition/event systems with multiple condition outputs. In *Emerging Technologies and Factory Automation, ETFA'95*, volume 1, pages 592–600, Paris, France, October 1995.

[RLSE04] M.P. Remelhe, S. Lohmann, O. Stursberg, and S. Engell. Algorithmic verification of logic controllers given as sequential function charts. In *Proc. IEEE Conf. On Computer-Aided Control Systems Design, (IEEE, 2004)*, pages 53–58, 2004.

[SG04] M. Stanika and H. Guéguen. Using timed automata for the verification of IEC 61499 applications. In *IFAC Conference WODES, Reims, septembre 2004*, 2004.

[SK91] R.S. Sreenivas and B.H. Krogh. On condition/event systems with discret state realizations. *Discret Event Dynamic Systems, 1991*, 1(2) :209–236, 1991.

[SMF97] Thomas Stauner, Olaf Müller, and Max Fuchs. Using hytech to verify an automative control system. In *Hybrid and Real-Time Systems, International Workshop, HART'97*, volume 1201 of *Lecture Notes in Computer Science*, pages 139–153. Springer, 1997.

[SR02] O. De Smet and O. Rossi. Verification of a controller for a flexible manufacturing line written in ladder diagram via model-checking. In *21th American Control Conference, (ACC'02), Anchorage, USA, May*, pages 4147–4152, 2002. proceedings on CD-ROM, communication 734.

[Tah03] Chrisitian Tahon. *Evaluation de performances des systèmes de production*. Hermes Science Publications, 2003.

[TF00] Luc Thevenon and Jean-Marie Flaus. Modular Representation of complex hybrid systems : Application to the simulation of batch processes. *Accepted for publication in the Journal of Simulation Practice*, 2000.

[VH99] V. Vyatkin and H.M. Hanisch. A modeling approach for verification of IEC 611499 function blocks using net condition/event systems. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 1, 1999.

[Zay01] Janan Zaytoon. *Systèmes Dynamiques Hybrides*. Hermes Science Publications, 2001.