Synthèse de commande sûre de fonctionnement à base de contraintes logiques pour les systèmes manufacturiers

Bernard Riera, David Annebicque, Bruno Vigario, François Gellot, Philippot Alexandre

9e Conférence Internationale de Modélisation, Optimisation et SIMulation - MOSIM'12
06 au 08 Juin 2012 - Bordeaux - France
« Performance, interopérabilité et sécurité pour le développement durable »

SYNTHESE DE COMMANDE SURE DE FONCTIONNEMENT A BASE DE CONTRAINTES LOGIQUES POUR LES SYSTEMES MANUFACTURIERS

B. RIERA, F. GELLOT, A. PHILIPPOT
CReSTIC (EA3804)
UFR des Sciences Exactes et Naturelles
Université de Reims Champagne-Ardenne
Moulin de la Housse
BP 1039, 51687 REIMS Cedex 2, France
bernard.riera@univ-reims.fr

B. VIGARIO
Rua Elisio de Melo, 39 - 3
REAL GAMES
4000-196 PORTO, Portugal
bruno.vigario@realgames.pt

D. ANNEBICQUE
CReSTIC (EA3804)
IUT de Troyes, 9 rue de Québec
BP 396
10026 TROYES, Cedex, France

RESUME : *Cet article présente une approche originale de synthèse de la commande sûre de fonctionnement des systèmes manufacturiers. Il s'agit d'une extension des travaux que le CReSTIC (Centre de Recherche en STIC de l'Université de Reims Champagne-Ardenne) conduit depuis plusieurs années sur la définition et la conception d'un filtre logique (placé dans l'Automate Programmable Industriel (API)) robuste aux erreurs de commande, et vérifié formellement au moyen d'un model checker. L'approche proposée, séparant la partie fonctionnelle de la partie sécurité de la commande, est simple à mettre en œuvre et garantit que la commande obtenue est sûre de fonctionnement. Le principe repose sur une utilisation des contraintes de sécurité du filtre pour obtenir la commande sûre la plus permissive. Cette dernière est ensuite contrainte au moyen de contraintes fonctionnelles. L'approche est illustrée au moyen d'un exemple pédagogique et d'un exemple simulé de tri de caisses de la collection ITS PLC de la société Real Games (www.realgames.pt) avec laquelle le CReSTIC entretient un partenariat scientifique et technique depuis 2008. L'algorithme de commande est présenté et permet d'aboutir à des commandes sûres plus simples à implémenter et à maintenir qu'une approche conventionnelle basée sur une spécification en GRAFCET (IEC 60848) ne distinguant pas l'aspect fonctionnel de l'aspect sécuritaire. Cette approche présente des perspectives intéressantes dans la prise en compte entre autres des différents modes de marche et d'arrêt.*

MOTS-CLES : *Systèmes à événements discrets, Commande, Sécurité, Automate Programmable Industriel, Synthèse algébrique, SCT.*

1 INTRODUCTION

Parallèlement à l'accroissement de la complexité des Systèmes Automatisés de Production (SAP) en terme de quantité, de besoins en communication, de diversité des composants, *etc.*, les exigences des automaticiens concernant la sûreté de fonctionnement (Villemeur, 1988) et l'aide à la conception de programmes se sont également accrues. En effet, même si les Automates Programmables Industriels (API) qui constituent une des principales architectures d'implantation de la commande des SAP, se programment avec des langages normalisés (IEC 61131-3), le concepteur ne dispose jusqu'à présent d'aucune aide véritable pour la conception de programmes. En fait, seule sa compétence et son expérience lui permettent d'élaborer des programmes dont il garantit lui-même qu'ils permettront d'obtenir un système sûr de fonctionnement respectant le cahier des charges. Assurer de manière formelle la sécurité des systèmes automatisés est donc un véritable défi scientifique porteur d'enjeux industriels importants.

Les travaux présentés dans ce papier relèvent de cette problématique. Ils portent uniquement sur les systèmes à événements discrets qui sont pilotés par des API et dont les Entrées/Sorties (E/S) ne sont que des variables logiques (Cassandra et Lafortune, 1999). Une méthodologie pour la conception d'une commande sûre de fonctionnement d'un SAP de type manufacturier est proposée. L'idée est de séparer la partie sécuritaire de la partie fonctionnelle en générant la commande à partir d'un ensemble de contraintes sécuritaires et fonctionnelles. Les contraintes de sécurité sont définies à partir d'une analyse structurelle et dysfonctionnelle du SAP et reposent sur des travaux de recherche (Benlorhfar *et al.*, 2011 ; Marangé *et al.*, 2010, 2009 ; Riera *et al.*, 2009a) réalisés au CReSTIC de l'Université de Reims. Elles permettent d'assurer la sécurité du SAP et de ses produits en cas d'erreurs de commande. La méthodologie est basée sur une vérification hors-ligne des contraintes de sécurité au moyen du *model-checker* UPPAAL (Behrmann *et al.*, 2002), qui permet de valider la suffisance des contraintes et de vérifier la robustesse du filtre

(Barragan *et al.*, 2006 ; Canet *et al.,* 2000). Les contraintes de sécurité sont exprimées sous la forme d'un monôme (produit de variables logiques, forme π) qui est une fonction logique des Entrées/Sorties de l'API et d'éventuels observateurs (fonction des Entrées) pour pallier au manque d'observabilité du système. Le filtre robuste s'implémente à la suite du programme de commande dans l'API afin d'assurer la sécurité quelle que soit la commande implémentée dans le contrôleur industriel (API). L'approche par filtre n'est pas nouvelle en soi, mais la démarche de vérification formelle est originale. Elle est basée sur une modélisation modulaire (Chandra *et al.,* 2002) représentant d'une part le comportement cyclique et séquentiel de la partie commande (lecture des Entrées, programme API et écriture des Sorties) et d'autre part, le comportement des sous-ensembles de partie opérative (PO) en interaction. La modélisation, vue de l'API, est effectuée par le biais d'automates communicants et a été étendue à la gestion des flux de produits (Benlorhfar *et al.*, 2011).

La méthode a été testée avec succès sur plusieurs systèmes manufacturiers réels et simulés, et elle a montré son intérêt dans des applications allant de la télémaintenance à la formation des automaticiens (Magalhães *et al.*, 2010). La méthodologie initiale concernait uniquement la sécurité des équipements. Nous montrons dans cet article que la méthodologie peut être complétée pour faire de la commande sûre de fonctionnement en séparant la partie fonctionnelle de la partie sécuritaire. Le principe est d'autoriser le fonctionnement des actionneurs tout le temps sauf quand le filtre de sécurité l'interdit et ensuite de contraindre le fonctionnement à partir des spécifications fonctionnelles (cahier des charges, contraintes énergétiques, …). Par conséquent, même si le fonctionnel est erroné, le système reste sûr de fonctionnement. La première partie de l'article présente l'approche globale en rappelant brièvement la méthodologie de conception du filtre robuste aux erreurs de commande. La deuxième partie, commence par une bibliographie sur les approches de synthèse de la commande qui ont orienté notre travail de recherche. La méthodologie de synthèse de la commande sûre basée sur des contraintes logiques est ensuite présentée. Enfin, dans la dernière partie de l'article, nous montrons sur deux exemples, l'un pédagogique et l'autre plus réaliste (système virtuel de tri de caisses de la collection ITS PLC (www.realgames.pt)), l'intérêt de cette approche originale de synthèse de la commande. Ce travail offre des perspectives intéressantes qui sont détaillées dans la conclusion. Les notations utilisées dans cet article sont les suivantes :

- t : instant courant (vu de l'API), *t-1* cycle précédent.
- $Sk=Sk(t)$: variable logique qui correspond à la valeur de la $k^{ème}$ Sortie booléenne (actionneur) de l'API à l'instant *t.*
- $Sk^*=Sk(t-1)$: variable logique qui correspond à la valeur de la $k^{ème}$ Sortie booléenne (actionneur) de l'API à l'instant *t-1*.
- $Ej=Ej(t)$: variable logique qui correspond à la valeur de la $j^{ème}$ Entrée (capteur) de l'API à l'instant *t.*
- $Ej(t-1)$: variable logique qui correspond à la valeur de la $j^{ème}$ Entrée (capteur) de l'API à l'instant *t-1*.
- S : ensemble des variables Sk aux instants t, $t-1$, $t-2$, …
- S^* : ensemble des variables Sk aux instants $t-1$, $t-2$, …
- E : ensemble des variables Ej aux instants t, $t-1$, $t-2$, …
- O : ensemble des observateurs aux instants t, $t-1$, $t-2$, …
- « . », « + », « ¯ » correspondent respectivement aux opérateurs logiques ET, OU et NON (complémentation)

2 METHODOLOGIE DE CONCEPTION D'UNE COMMANDE SURE DE FONCTIONNEMENT BASEE SUR UN FILTRE ROBUSTE AUX ERREURS DE COMMANDE

La méthodologie de conception de la commande proposée, repose sur l'utilisation des contraintes de sécurité (exprimées sous la forme de monômes logiques) du filtre. Ce dernier est obtenu à partir d'une approche modulaire et itérative de modélisation et de vérification, afin d'éviter la complexité, les explosions combinatoires et assurer son applicabilité dans le monde industriel. Les étapes nécessaires à l'obtention de la commande sont décrites rapidement ci-après (Benlorhfar *et al.*, 2011 ; Marangé *et al.*, 2010).

- L'expert mène une analyse dysfonctionnelle (au moyen d'une AMDEC par exemple) pour définir l'ensemble des états dangereux à ne pas atteindre pour les éléments de Partie Opérative et les produits. À chaque état dangereux va correspondre un ensemble de propriétés à vérifier formellement.

- L'expert identifie pour chaque état dangereux le sous-ensemble de la PO (les composants et la séquence du flux des produits) à modéliser pour la vérification par *model-checking*.

- L'expert définit pour chaque sous-système l'ensemble des contraintes sécuritaires qui constitue le filtre, et qui devront être vérifiées à chaque cycle de l'API, ainsi que les observateurs nécessaires à la reconstruction binaire des informations manquantes (non-observabilité).

- L'expert procède à la validation de l'ensemble des contraintes pour tous les sous-systèmes en vérifiant les propriétés.

Les 4 premières étapes correspondent à l'approche initiale proposée (Benlorhfar *et al.*, 2011 ; Marangé *et al.*, 2010) pour la conception d'un filtre robuste aux erreurs de commande et reposent sur une modélisation de la PO et de la Partie Commande (PC) à base d'automates à état communicants. Quelques remarques peuvent être faites sur la conception et la réalisation du filtre robuste aux erreurs de commande.

- Les contraintes sont toujours modélisées avec le point de vue de la PC et on suppose que le temps de cycle de l'API est suffisant pour détecter tous les changements du vecteur d'entrées (fonctionnement synchrone, possibilité de changements simultanés des Entrées de l'API). De plus la PO est considérée en fonctionnement normal sans présence de défaillance.

- Chaque contrainte de sécurité s'exprime sous la forme d'une fonction logique (monôme) devant être égale tout le temps (c'est-à-dire : à chaque cycle API) à 0, et on considère l'état de repos à 0 pour les actionneurs (*Sk*) comme sûr.

- Initialement, les contraintes sont définies pour pouvoir garantir la commande la plus permissive et on suppose qu'avec le filtre, le système reste commandable. En d'autres termes, il est possible de réaliser une commande permettant de respecter le cahier des charges. Par exemple, un filtre qui met toutes les sorties à 0 est sûr mais n'assure pas la commandabilité.

- Certaines contraintes font intervenir une seule sortie (contraintes simples *CSs*), d'autres plusieurs (contraintes combinées *CSc*).

- Les contraintes nécessitent la connaissance des Entrées/Sorties (E/S) à l'instant courant, et éventuellement aux instants précédents (présence d'un front montant ou descendant par exemple). Le filtre a donc une fonction mémoire.

- Il peut être nécessaire de définir des observateurs compte tenu du manque d'observabilité du système, cela est particulièrement vrai lorsqu'il y a des flux de produits (*cf.* l'exemple du tri de caisses en fin d'article).

- Les observateurs correspondent à une fonction séquentielle des Entrées de l'API.

- Certaines contraintes peuvent exister compte tenu de l'impossibilité de reconstruire l'observateur uniquement à partir des entrées.

- Les observateurs permettent de se ramener à une contrainte logique combinatoire.

La figure 1 représente la structure interne du filtre de sécurité. La représentation vectorielle représente la valeur des différentes variables (Entrées : E, Sorties : Sorties : S, Observateurs : Obs, aux différents instants t, t-1, …).

Dans notre approche, on suppose que les contraintes de sécurité peuvent toujours être représentées sous la forme d'un monôme et dépendent des entrées (instant courant (*t*) et instants précédents (*t-1*, *t-2*, …)), des sorties (instant courant (*t*) et instants précédents (*t-1*, *t-2*, …)) et d'observateurs (uniquement fonction des entrées à l'instant courant (*t*) et aux instants précédents (*t-1*, *t-2*,…)). Deux types de contrainte sont définis : les Contraintes de Sécurité simples (*CSs*, équation (1)) et les Contraintes de Sécurité combinées (*CSc*, équation (2)).

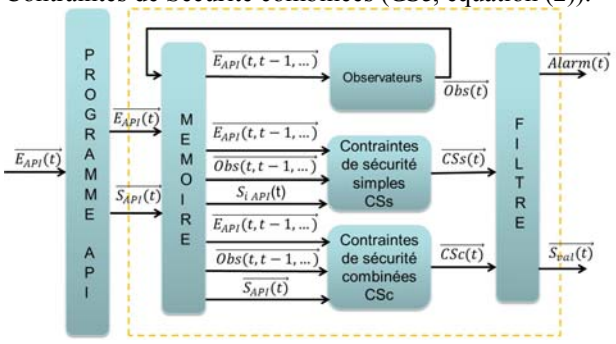

Figure 1 : Structure interne du filtre robuste

$$CSs_i(Sk, Y) = h_i(Sk, E, O, S^*) \quad (1)$$

$$\mathrm{CSc_i(Sk, Sl, Y) = h_i(Sk, Sl, E, O, S^*)} \quad (2)$$

Dans la suite de l'article, les hypothèses suivantes sont faites :

- H1 : A l'état initial toutes les sorties sont à 0 et que l'état est sûr de fonctionnement.

- H2 : Les contraintes combinées font intervenir au plus 2 sorties à l'instant courant *t* (équation (2)).

- H3 : Dans le cas des *CSc*, on considère l'état où les sorties sont à 0 comme sûr.

Les contraintes de sécurité simples (*CSs*), parce qu'elles s'expriment sous la forme d'un monôme et qu'elles font intervenir une seule sortie à l'instant courant *t* peuvent s'écrire de 2 façons différentes (équation (3) et (4)).

$$CSs_i(Sk, Y) = Sk.f_i(E, O, S^*) \quad (3)$$

$f_i(E, O, S^*)$ correspond au cofacteur positif de la fonction logique $CSs_i(Sk, Y)$ par rapport à *Sk* (*i.e.* $CSs_i(1, Y)$).

$$CSs_i(Sk, Y) = \overline{Sk}.f_i(E, O, S^*) \quad (4)$$

$f_i(E, O, S^*)$ correspond alors au cofacteur négatif de la fonction logique $CSs_i(Sk, Y)$ par rapport à *Sk* (*i.e.* $CSs_i(0, Y)$).

Ces contraintes simples expriment le fait que si $f_i(E, O, S^*)$ est égale à 1, *Sk* doit être nécessairement égale à 0 ($Sk.f_i(E, O, S^*)$) ou égale à 1 ($\overline{Sk}.f_i(E, O, S^*)$).

Les contraintes de sécurité combinées (*CSc*) s'expriment également sous la forme d'un monôme logique et font intervenir 2 sorties. Par conséquent, elles peuvent s'écrire de 4 façons différentes (équation (5), (6), (7) et (8)).

$$CSc_i(Sk, Sl, Y) = Sk.Sl.f_i(E, O, S^*) \quad (5)$$

$$CSc_i(Sk, Sl, Y) = \overline{Sk}.Sl.f_i(E, O, S^*) \quad (6)$$
$$CSc_i(Sk, Sl, Y) = \overline{Sl}.Sk.f_i(E, O, S^*) \quad (7)$$
$$CSc_i(Sk, Sl, Y) = \overline{Sk}.\overline{Sl}.f_i(E, O, S^*) \quad (8)$$

L'équation (5) signifie que si *f(E, O, S*)* est égale à 1, les 2 sorties *Sk* et *Sl* ne peuvent pas être toutes les 2 égales à 1. Il est donc nécessaire de donner la priorité à *Sk* ou à *Sl* (en fonction du flux de produit par exemple).

L'équation (6) signifie que si *f(E, O, S*)* est égale à 1, on ne peut pas avoir simultanément *Sl=1* et *Sk=0*. Étant donné l'hypothèse H3, on en déduit que si *Sk=0*, alors *Sl=0*. L'équation (7) est similaire à l'équation (6), si *Sl*=0 alors *Sk=0*.

L'équation (8) signifie que si *f(E, O, S*)* est égale à 1, les 2 sorties *Sk* et *Sl* ne peuvent pas être toutes les 2 à 0. Ce type de contraintes est dans notre cas sans objet compte tenu de l'hypothèse H3.

La méthodologie est maintenant complétée (cf. figure 2) pour réaliser une commande sûre de fonctionnement au moyen des étapes décrites ci-après.

- L'expert définit des contraintes fonctionnelles simples (*CFs*, qui seront définies dans la suite de l'article) pour répondre aux spécifications du cahier des charges : contraintes énergétiques, mode de pilotage, ordonnancement de la production....

- La synthèse de la commande peut ensuite être réalisée en utilisant l'algorithme proposé dans la 4ème partie de l'article.

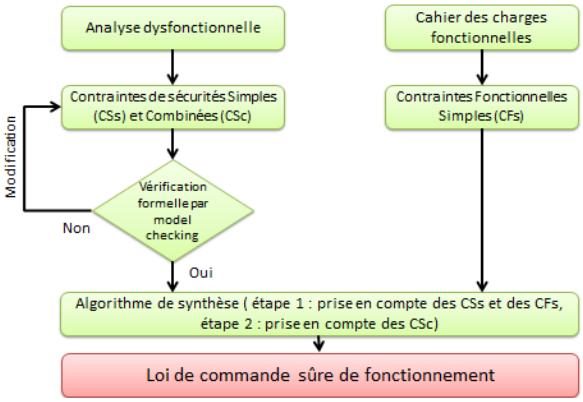

Figure 2 : Méthodologie pour une synthèse de la commande sûre de fonctionnement

3 DIFFERENTES APPROCHES DE SYNTHESE DE LA COMMANDE

Avant de détailler l'algorithme de synthèse proposé, deux approches de synthèse que l'on trouve dans la bibliographie vont être rapidement présentées car elles sont à l'origine de l'approche proposée.

3.1 La « Supervisory Control Theory (SCT) »

La SCT (Ramadge et Wonham, 1989) repose sur la distinction entre les modèles du procédé à contrôler, des propriétés à respecter et de la structure de contrôle (Faraut, 2010 ; Faraut *et al.* 2010). La démarche courante d'utilisation de cette théorie, s'articule autour de 3 étapes, dont la première consiste à construire 2 modèles : le modèle du procédé (représentant les comportements possibles du système), et le modèle des spécifications (propriétés attendues). A partir de ces 2 modèles, l'étape suivante consiste à obtenir mathématiquement le modèle du procédé sous contrôle dont le comportement représente ce qui est possible de faire dans le procédé tout en respectant les contraintes imposées par les spécifications. La troisième et dernière étape consiste à synthétiser le superviseur qui va restreindre les comportements possibles du procédé. Le comportement du procédé et du superviseur est alors le même que celui du procédé sous contrôle. L'avantage de l'approche de la SCT est, en plus du cadre formel proposé, la séparation entre les modèles qui permet de ne pas mélanger plusieurs concepts dans un seul modèle. L'inconvénient de la SCT est la difficulté de l'appliquer dans la pratique car des problèmes de modélisation et d'explosion combinatoire compte tenu des outils mathématiques utilisés (théorie des langages) se posent. En effet, le modèle du procédé sous contrôle qui représente l'ensemble des comportements possibles est difficile à obtenir (explosion combinatoire) pour un système industriel d'une dizaine d'Entrées/Sorties.

3.2 La synthèse algébrique de la commande

Le LURPA de l'ENS Cachan travaille sur des méthodes algébriques (Hietter 2011 ; Hietter *et al.* 2008*a*, 2008*b*) de synthèse de la commande. Le point de départ est un ensemble de spécifications fonctionnelles et de sécurité exprimé au moyen de relations algébriques et/ou de machines à états. L'idée est de synthétiser la commande en résolvant une équation booléenne qui représente toutes les spécifications. Les auteurs obtiennent une formulation paramétrique de toutes les solutions possibles. Le choix d'une solution particulière est laissé alors au soin de l'expert. Cette approche est très intéressante d'un point de vue théorique et la loi de synthèse obtenue facilement implémentable dans un API. Toutefois, le résultat obtenu est difficilement maintenable (car difficile à interpréter), la sûreté du programme n'est pas garantie, et la prise en compte des différents modes de marche difficile. En effet une modification des spécifications entraine nécessairement une nouvelle formulation des besoins et donc une synthèse de la commande (hors ligne) différente.

L'approche que nous proposons reprend certains principes de la SCT et de la synthèse algébrique. On retrouve le principe de la séparation des modèles et de l'utilisation d'un superviseur qui autorise ou interdit les événements liés à la commande. Toutefois, les outils de modélisation et l'approche sont différents car nous ne déterminons pas

de modèle du procédé sous contrôle ce qui évite une explosion combinatoire. En effet, dans notre cas, les contraintes du filtre de sécurité vérifiées formellement hors ligne vont servir à générer la loi de commande sûre la plus permissive. L'algorithme de synthèse que nous proposons est une solution algébrique au problème de la commande et une partie des résultats théoriques obtenus par le LURPA vont être utilisés. Toutefois, il existe plusieurs différences fondamentales avec la méthode algébrique proposée par Hietter (2011). Dans l'approche que nous proposons, les besoins (*i.e.* les spécifications) liés à la sécurité ne sont pas exprimés par un expert mais obtenus à partir du filtre robuste qui a été vérifié formellement au moyen d'un model checker. Si le filtre a été conçu en vue de permettre la commande la plus permissive, celui-ci est valable quelle que soit la loi de commande et donc les spécifications fonctionnelles. La séparation entre le fonctionnel et le sécuritaire permet dans notre cas une gestion des modes de marche grandement facilité et une lisibilité du code API. Enfin, le rôle de l'expert n'est pas de choisir la solution parmi l'ensemble des solutions possibles mais consiste uniquement à indiquer dans le cas des contraintes de sécurité combinées (*CSc*) quelle est la sortie prioritaire.

4 SYNTHESE SURE DE LA COMMANDE A PARTIR DE CONTRAINTES LOGIQUES

L'algorithme de commande proposé sépare la sécurité du fonctionnel. Nous considérons qu'un ensemble de contraintes de sécurité est nécessaire et suffisant si toutes les contraintes sont indispensables pour garantir la sécurité. Toutes les autres contraintes qui peuvent être ajoutées sont considérées comme des contraintes fonctionnelles.

L'algorithme de synthèse consiste à autoriser par défaut tout ce qui n'est pas interdit et à le contraindre par les besoins fonctionnels. Il se compose de 2 étapes séquentielles et s'implémente facilement dans un API.

Etape 1 : définir la commande sûre la plus permissive pour chaque actionneur (*Sk*) à partir des *CSs* en intégrant les contraintes fonctionnelles. Dans le cadre de cet article uniquement les contraintes fonctionnelles simples (*CFs*) faisant intervenir qu'une seule variable de sortie à l'instant courant *t* seront traitées. Cette première étape permet de générer pour chaque sortie une commande intermédiaire (*S'k*).

Etape 2 : Prendre en compte les contraintes de sécurité combinées (*CSc*) pour calculer la commande finale de chaque sortie *Sk*.

4.1 Étape 1 : calcul des *S'k*

Cette étape consiste à définir la commande sûre la plus permissive pour chaque actionneur (*Sk*) à partir des *CSs* (exprimées sous forme de monômes) en tenant compte des contraintes fonctionnelles simples (*CFs*). Pour cela, à partir des *CSs*, et pour chaque sortie, nous pouvons écrire la relation suivante correspondant à un OU logique entre toutes les contraintes simples (équation (9)).

$$\sum_j CSsj = \sum_k f_k(E, O, Sk) = 0 \quad (9)$$

$f_k(E, O, Sk)$ est une fonction logique indépendante des autres sorties car uniquement les *CSs* sont considérées. Compte tenu de ce qui a été énoncé au paragraphe 2, il est possible d'écrire l'équation (10).

$$f_k(E, O, Sk) = Sk.f_{0k} + \overline{Sk}.f_{1k} \quad (10)$$

f_{0k} et f_{1k} sont des polynômes (forme $\sum\pi$) fonction des entrées *E* et des observateurs *O* aux instants *t*, *t-1*, …

On considère dans cet article uniquement des contraintes fonctionnelles simple(*CFs*). Celles-ci ne font donc intervenir qu'une seule sortie *Sk* à l'instant courant *t*. De la même façon que pour les *CSs*, on peut les écrire sous la forme d'un monôme logique en indiquant quand la sortie *Sk* doit être égale à 0 (g_{0k}) ou bien quand la sortie *Sk* doit être égale à 1 (g_{1k}) (équation (11))..

$$g_k(E, O, Sk) = Sk.g_{0k}\ ou$$
$$g_k(E, O, Sk) = \overline{Sk}.g_{1k} \quad (11)$$

On a pour habitude lors de la spécification d'indiquer plutôt quand la sortie doit être activée et donc les g_{1k}. Ces derniers peuvent bien entendu correspondre à des variables d'étapes de GRAFCET (IEC 60848) ou de SFC (IEC 61131-3). En reprenant les résultats de Hietter (2008), nous pouvons écrire que la solution paramétrique de l'équation $fk(E, O, Sk)$ existe si et seulement si l'équation (12) est vérifiée.

$$f_{0k}.f_{1k} = 0 \quad (12)$$

Cette propriété est toujours vraie. En effet, si ce n'est pas le cas, cela signifie que 2 *CSs* sont contradictoires et au moins l'une des 2 est violée et la sécurité n'est donc pas assurée. Cela n'est pas possible s'il n'y a pas de défaillances de la PO, car le filtre a été vérifié formellement. La solution paramétrique de l'équation $fk(E, O, Sk)$ s'écrit selon l'équation (13).

$$S'k = \overline{f_{ok}} + f_{1k}.p \quad (13)$$

Où *p* est un paramètre booléen que nous choisissons égal à 1 pour garantir la sécurité. En intégrant les *CFs*, la solution devient l'équation (14).

$$S'k = \overline{f_{ok}}.g_{1k} + f_{1k}\ ou$$
$$S'k = \overline{f_{ok}}.\overline{g_{ok}} + f_{1k} \quad (14)$$

La commande obtenue est sûre de fonctionnement car la sécurité est assurée quelles que soient les *CFs*. En effet, si les *CFs* venaient à imposer une sortie à 0, contraire à la sécurité, le terme f_{1k} continuerait d'assurer la sécurité. Par conséquent la partie fonctionnelle peut être conçue

sans prendre en compte la sécurité, ce qui facile grandement le travail de l'automaticien.

4.2 Etape 2 : calcul des *Sk*, prise en compte des *CSc*

Les commandes *S'k* ne tiennent pas compte des contraintes de sécurité combinées. Dans le cas de l'équation (5), il est nécessaire de donner (ou pas) la priorité à une des 2 sorties. L'équation (8) est sans objet comme cela a été vu précédemment.

L'algorithme de calcul des *Sk* est donnée figure 3. Le principe de l'algorithme est de vérifier que chaque *CSc* n'est pas violée. Dans le cas contraire, la sortie *Sl* (non prioritaire sur *Sk*) est mise à zéro. On réitère le calcul jusqu'à qu'aucune *CSc* ne soit violée. Il est à noter que la solution existe nécessairement car l'ensemble des contraintes (*CSs* et *CSc*) a été vérifié formellement. Il est possible d'obtenir une solution algébrique pour chaque sortie en appliquant l'algorithme « à la main ». Le résultat sera alors une fonction logique combinatoire pour chaque sortie *Sk*. Comme cela a déjà été souligné, il est très facile d'implémenter l'algorithme ou la solution algébrique dans un API.

Répéter
  Pour i=1 **à** nbre_de_*CSc*
    Pour chaque *CSci*,
      Si *Sk* prioritaire devant *Sl* **alors**

$$Sk = S'k$$

$$Sl = S'l.\overline{CSci(S'k,\ S'l,\ Y)}$$

      Fin Si
    Fin pour
  Fin pour

  Flag = false

  Pour i=1 **à** nbre_de_*CSc*
    Flag= Flag + *CSci(Sk, Sl, Y)*
  Fin pour

  Pour i=1 **à** nbre_de_S
    S'i=Si
  Fin pour
Jusqu'à non(Flag)

Figure 3 : Algorithme de calcul des *Sk*

L'approche proposée présente plusieurs avantages. L'aspect fonctionnel est déconnecté de l'aspect sécuritaire. Il est de plus facile de distinguer les capteurs spécifiques à la sécurité, au fonctionnel et aux deux. La prise en compte des modes de marche est facilitée, car seules les $g_{0k} et\ g_{1k}$ ont besoin d'être modifiées. On aboutit à des commandes sûres beaucoup plus concises et lisibles que l'approche de spécification habituelle (décrire exactement ce que le système doit faire en mélangeant la sécurité et le fonctionnel). Il est possible d'implémenter l'algorithme dans l'API ou d'obtenir une solution algébrique (*cf.* l'exemple du tri de caisses). Enfin, la commande obtenue est sûre même si elle ne respecte pas forcément le cahier des charges si les contraintes fonctionnelles ont été mal définies. L'ensemble de contraintes de sécurité a été conçu (et vérifié formellement) initialement pour servir comme filtre robuste tout en permettant la commande la plus permissive. Dans l'approche de synthèse, il devient possible de définir des contraintes de sécurité par rapport au fonctionnel désiré La dernière partie de l'article est consacrée à deux exemples permettant d'illustrer l'intérêt de l'approche.

5 APPLICATION SUR UN SYSTEME PEDAGOGIQUE

Le système est illustré figure 4. Il se compose de 4 vérins monostables possédant chacun deux capteurs de fin de course (position rentrée, position sortie). Le détail des entrées et des sorties est donné dans le tableau 1. Quatre capteurs de présence d'objets (2 caisses et 1 ballon) devant les vérins sont également installés (*I13*, *I14*, *I15*, *I16*).

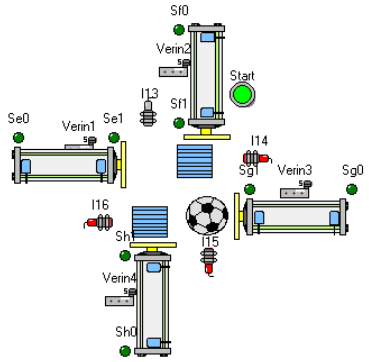

Figure 4 : Système pédagogique des 4 vérins

L'objectif de cet exemple est de faire tourner les 3 objets dans le sens des aiguilles d'une montre, en évitant toutes collisions entre les vérins et les objets.

ELEMENTS	CAPTEURS	ACTIONNEURS
Vérin 1	Se0 (rentrée) Se1 (sortie)	S0 = $Q9^+$
Vérin 2	Sf0, Sf1	S1 = $Q10^+$
Vérin 3	Sg0, Sg1	S2 = $Q11^+$
Vérin 4	Sh0, Sh1	S3 = $Q12^+$

Tableau 1 : Détail des éléments avec capteurs et actionneurs.

5.1 Application de la démarche : Commande par contraintes

Les contraintes de sécurité simples et combinées ont été validées formellement en utilisant la démarche présentée dans la première partie de l'article de conception d'un filtre de sécurité robuste aux erreurs de commande. Dans cet exemple nous obtenons les douze contraintes simples (*CSs*) données dans les équations (15) à (26) et quatre contraintes combinées (*CSc*) données dans les équations (27) à (30).

$$CSs_1 = Q9^+ . \overline{Sf0} \qquad (15)$$

$$CSs_2 = \overline{Q9(t-1)}.Q9^+.I14 \quad (16)$$
$$CSs_3 = Q9(t-1).\overline{Q9^+}.\overline{Se1} \quad (17)$$
$$CSs_4 = Q10^+.\overline{Sg0} \quad (18)$$
$$CSs_5 = \overline{Q10(t-1)}.Q10^+.I15 \quad (19)$$
$$CSs_6 = Q10(t-1).\overline{Q10^+}.\overline{Sf1} \quad (20)$$
$$CSs_7 = Q11^+.\overline{Sh0} \quad (21)$$
$$CSs_8 = \overline{Q11(t-1)}.Q11^+.I16 \quad (22)$$
$$CSs_9 = Q11(t-1).\overline{Q11^+}.\overline{Sg1} \quad (23)$$
$$CSs_{10} = Q12^+.\overline{Se0} \quad (24)$$
$$CSs_{11} = \overline{Q12(t-1)}.Q12^+.I13 \quad (25)$$
$$CSs_{12} = Q12(t-1).\overline{Q12^+}.\overline{Sh1} \quad (26)$$

$$CSc_1 = Q9^+.Q10^+ \quad (27)$$
$$CSc_2 = Q10^+.Q11^+ \quad (28)$$
$$CSc_3 = Q11^+.Q12^+ \quad (29)$$
$$CSc_4 = Q12^+.Q9^+ \quad (30)$$

Les contraintes de sécurité simples (*CSs*) empêchent chaque vérin de sortir si une pièce est présente devant le vérin suivant ou si celui-ci n'est pas rentré. Enfin, une fois l'ordre de sortie du vérin activé, sa désactivation est interdite tant que le vérin n'est pas complétement sorti. Enfin, les contraintes combinées empêchent d'activer simultanément les sorties de 2 vérins adjacents. Les 3 hypothèses (H1, H2 et H3) sont respectées.

L'aspect fonctionnel de cet exemple est réalisé de manière classique en utilisant une spécification au moyen du langage GRAFCET, un par vérin. La figure 5 montre les 4 GRAFCET obtenus. Cette commande ne se soucie pas des aspects sécuritaires et des possibles interactions entre les différents vérins, ce sont les contraintes de sécurité qui s'en chargeront ensuite. Ainsi le vérin 1 ($Q9^+$) est commandé pour sortir (étape X1 du GRAFCET) dès qu'un objet est présent devant lui (capteur I13). De manière identique, le vérin 2 ($Q10^+$) est activé par l'étape X11 d'un deuxième GRAFCET, le vérin 3 ($Q11^+$) avec l'étape X21 et le vérin 4 ($Q12^+$) avec l'étape X31.

Ce qui donne les équations des contraintes fonctionnelles (*CF*) données dans les équations (31) à (34).

$$CF_{s1} = X1.\overline{Q9^+} \quad (31)$$
$$CF_{s2} = X11.\overline{Q10^+} \quad (32)$$
$$CF_{s3} = X21.\overline{Q11^+} \quad (33)$$
$$CF_{s4} = X31.\overline{Q12^+} \quad (34)$$

On peut maintenant écrire les f_{0k}, f_{1k} et g_{1k} pour chaque sortie à partir des *CSs* et des CFs (équations (35) à (38)).

$$Q9^+ \begin{cases} f00 = \overline{Sf0} + \overline{Q9^{+*}}.I14 \\ f10 = \overline{Se1}.Q9^{+*} \qquad g10 = X1 \end{cases} \quad (35)$$

$$Q10^+ \begin{cases} f01 = \overline{Sg0} + \overline{Q10^{+*}}.I15 \\ f11 = \overline{Sf1}.Q10^{+*} \qquad g11 = X11 \end{cases} \quad (36)$$

$$Q11^+ \begin{cases} f02 = \overline{Sh0} + \overline{Q11^{+*}}.I16 \\ f12 = \overline{Sg1}.Q11^{+*} \qquad g12 = X21 \end{cases} \quad (37)$$

$$Q12^+ \begin{cases} f03 = \overline{Se0} + \overline{Q12^{+*}}.I13 \\ f13 = \overline{Sh1}.Q12^{+*} \qquad g13 = X22 \end{cases} \quad (38)$$

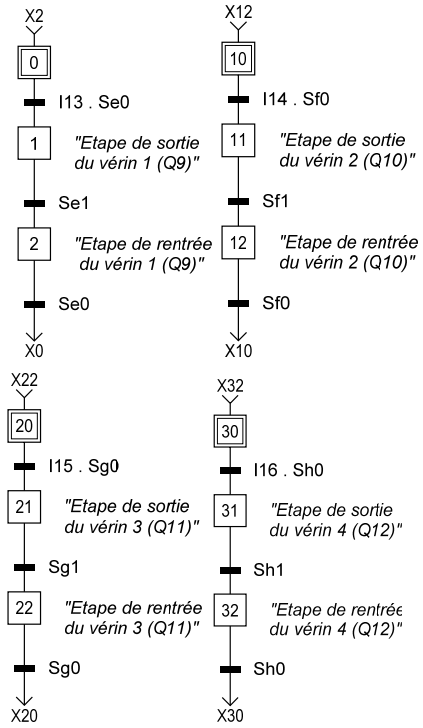

Figure 5 : GRAFCET fonctionnel (commande par contraintes).

L'algorithme peut ensuite être appliqué. Dans la première étape les *S'k* sont calculées à partir de l'équation (39).

$$S'k = \overline{f_{ok}}.g_{1k} + f_{1k} \quad (39)$$

La seconde étape permet de calculer les *Sk* à partir des *CSc*. L'ensemble de l'algorithme a été implanté avec succès sous le programme Unity de Schneider Electric en utilisant les langages de la norme IEC 61131-3. Le langage ST (*Structured Text*) a été utilisé pour l'algorithme de commande, et le SFC pour la partie fonctionnelle avec des réceptivités écrites en LD.

5.2 Spécification GRAFCET de la commande

La démarche utilisée pour établir le GRAFCET permettant de respecter le cahier des charges est la méthode d'analyse structurée par tâches (Taillard, 2010). L'analyse permet d'aboutir aux 4 GRAFCET synchronisés de la figure 6 intégrant l'aspect sécurité et fonctionnel et dont les réceptivités sont délicates à définir sans une approche méthodologique rigoureuse. Il est ensuite simple d'implémenter cette spécification dans un API au moyen des langages SFC, ST ou LD de la norme IEC1131-3.

5.3 Discussions

L'algorithme de commande par contraintes est simple à mettre en place dans un API, notamment en utilisant le langage ST par exemple. La partie fonctionnelle est très facilement implémentable dans un API en SFC. Les contraintes fonctionnelles correspondent alors à des étapes du SFC. Les deux commandes obtenues respectent le même cahier des charges. Cependant la méthode de commande par contraintes présente l'avantage de séparer explicitement l'aspect sécuritaire de l'aspect fonctionnel. Il n'est donc pas nécessaire de se soucier de l'environnement de chaque actionneur et la prise en compte des conditions initiales est par conséquent facilitée.

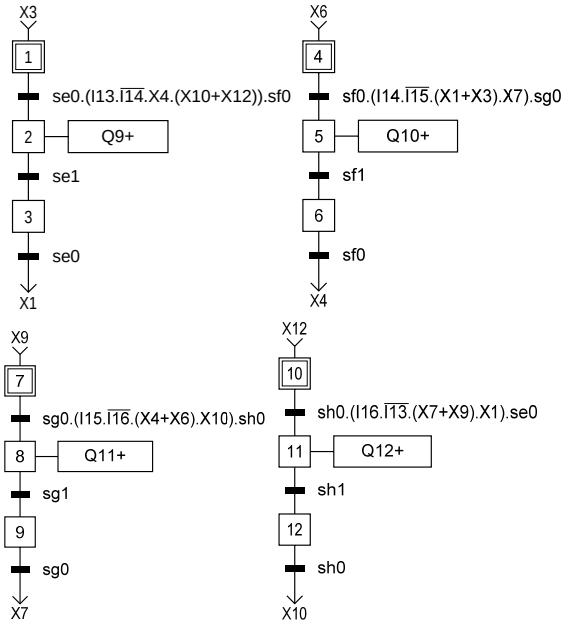

Figure 6 : GRAFCET de commande pour les 4 vérins (approche classique par une analyse structurée par tâches).

6 APPLICATION SUR LE SYSTEME TRI DE CAISSES DE LA COLLECTION ITS PLC

L'ensemble de la méthodologie proposée va être maintenant illustrée au moyen d'un système virtuel de la collection ITS PLC proposée par la société portugaise Real Games. Des versions de démonstration ainsi que des descriptions techniques des 5 systèmes industriels virtuels très réalistes proposés sont téléchargeables gratuitement à l'adresse www.realgames.pt. La collection ITS PLC est un ensemble de logiciels de simulation de PO pour la formation à l'automation (Riera *et al.*, 2009*a*, 2009*b*, 2011). Dans le cadre du travail présenté dans cet article, nous avons utilisé le système « tri de caisses ». L'objectif du système est d'amener des caisses du tapis d'alimentation aux monte-charges en les triant selon leur arrivée ou leur hauteur par exemple (*cf.* figure 7).

Le système est instrumenté au moyen de 11 capteurs (tableau 2) permettant de déterminer la taille des caisses (petite ou grande) et l'entrée ou la sortie d'une caisse des différents convoyeurs (alimentation, intermédiaire, évacuation) ou du plateau tournant. Les 7 sorties de l'API (tableau 3) permettent de mettre en marche les différents convoyeurs et le plateau tournant.

Figure 7 : Système de tri de caisses issu de la collection IST PCLC

Capteur	Information	Position
C0	Fin de tapis	Tapis alimentation
C1	Petite caisse	Tapis intermédiaire
C2	Grande caisse	
C3	Fin de tapis	
C4	Position de chargement	Plateau tournant
C5	Position de déchargement	
C6	Présence de caisse	
C7	Entrée du tapis	Tapis d'évacuation à gauche
C9	Fin de tapis	
C8	Entrée du tapis	Tapis d'évacuation à droite
C10	Fin de tapis	

Tableau 2 : Détail des entrées du système de tri de caisses

Actionneur	Action
S0	Activation du tapis d'alimentation
S1	Activation du tapis intermédiaire
S2	Activation des rouleaux (chargement et évacuation à gauche)
S3	Activation des rouleaux (évacuation à droite)
S4	Rotation du plateau
S5	Activation du tapis gauche
S6	Activation du tapis droit

Tableau 3 : Détail des sorties du système de tri de caisses

Le cahier des charges retenu est le suivant : après un appui sur le bouton « start », les caisses sont acheminées à tour de rôle vers le monte-charge de gauche et le monte-charge de droite (1 caisse sur 2). Après un appui sur le bouton « stop », les caisses en cours d'acheminement sont évacuées. Pour ne pas surcharger l'article, on considère que les convoyeurs de sortie sont toujours en marche ($S5 = S6 = 1$) et la gestion des boutons « start » et « stop » n'est pas présentée. La conception du filtre a permis d'aboutir aux 14 *CSs* (équations (40) à (52)) et 2 *CSc* (équations (53) et (54)) (représentées sous la forme de monômes logiques) ci-après, vérifiées formellement au moyen du model checker UPPAAL. Celles-ci garantissent la commandabilité (il existe au moins une commande permettant d'amener des caisses vers l'ascenseur de gauche et l'ascenseur de droite), et la sécurité quelle que soit la commande. Il est

à noter que ces contraintes sont les plus permissives (large espace de commande autorisé) mais nécessitent 3 observateurs (*2P*, *P36* et *P67*).

$$\mathrm{CSs1} = \mathrm{2P}.\mathrm{S0} \quad (40)$$
$$\mathrm{CSs2} = \mathrm{C3}.\overline{\mathrm{C4}}.\mathrm{S1} \quad (41)$$
$$\mathrm{CSs3} = \mathrm{C3}.\mathrm{C4}.\mathrm{C6}.\mathrm{S1} \quad (42)$$
$$\mathrm{CSs4} = \mathrm{C3}.\mathrm{P36}.\mathrm{S1} \quad (43)$$
$$\mathrm{CSs5} = \overline{\mathrm{C5}}.\mathrm{S3} \quad (44)$$
$$\mathrm{CSs6} = \mathrm{C4}.\mathrm{C6}.\mathrm{S2} \quad (45)$$
$$\mathrm{CSs7} = \overline{\mathrm{C4}}.\overline{\mathrm{C5}}.\mathrm{S2} \quad (46)$$
$$\mathrm{CSs8} = \mathrm{C5}.\mathrm{C6}.\overline{\mathrm{S4}} \quad (47)$$
$$\mathrm{CSs9} = \mathrm{C8}.\overline{\mathrm{S4}} \quad (48)$$
$$\mathrm{CSs10} = \mathrm{C7}.\overline{\mathrm{S4}} \quad (49)$$
$$\mathrm{CSs11} = \mathrm{P67}.\overline{\mathrm{S4}} \quad (50)$$
$$\mathrm{CSs12} = \mathrm{C5}.\mathrm{C7}.\mathrm{S2} \quad (51)$$
$$\mathrm{CSs13} = \mathrm{C4}.\overline{\mathrm{C6}}.\mathrm{S4} \quad (52)$$

$$\mathrm{CSc1} = \mathrm{C0}.\mathrm{S0}.\overline{\mathrm{S1}} \quad (53)$$
$$\mathrm{CSc2} = \mathrm{C3}.\mathrm{C4}.\mathrm{S1}.\overline{\mathrm{S2}} \quad (54)$$

P36 et *P67* sont des observateurs permettant de savoir qu'une caisse est présente respectivement entre les capteurs *C3* et *C6* exclus et *C6* et *C7* exclus. Par exemple, *P36* est mis à 1 sur un front descendant du capteur *C3* et remis à 0 sur un front montant du capteur *C6*.

La distance entre les capteurs *C0* et *C1* est plus petite que la taille d'une caisse. L'observateur *2P* permet de savoir que si *C0*=*C1*=1, 2 caisses sont présentes et non pas une seule (*cf.* figure 8)

2P = 0

2P = 1

Figure 8 : Observateur 2P

Les 3 hypothèses (H1, H2 et H3) sont respectées.

Concernant les *CFs*, l'expert a proposé les 7 contraintes données dans les équations (55) à (61).

$$\mathrm{CFs1} = \mathrm{C4}.\overline{\mathrm{S2}} \quad (55)$$
$$\mathrm{CFs2} = \mathrm{C5}.\mathrm{PC}.\overline{\mathrm{S2}} \quad (56)$$
$$\mathrm{CFs3} = .\overline{\mathrm{PC}}.\overline{\mathrm{S3}} \quad (57)$$
$$\mathrm{CFs4} = \mathrm{C6}.\overline{\mathrm{S4}} \quad (58)$$
$$\mathrm{CFs5} = \mathrm{C8}.\overline{\mathrm{S4}} \quad (59)$$
$$\mathrm{CFs6} = \overline{\mathrm{S0}} \quad (60)$$
$$\mathrm{CFs7} = \overline{\mathrm{S1}} \quad (61)$$

PC est un observateur dont la valeur est complémentée sur un front descendant des capteurs *C7* ou *C8*, et permet de diriger les caisses vers l'ascenseur de gauche ou l'ascenseur de droite. Il est intéressant de constater que la *CFs5* est inutile car incluse dans les *CSs*, mais cela n'a aucune conséquence pour le calcul de la commande. On peut écrire les f_{0k}, f_{1k} et g_{1k} pour chaque sortie à partir des *CSs* (équations (62) à (66)).

$$S0\begin{cases} f00 = 2p \\ f10 = 0 \quad g10 = 1 \end{cases} \quad (62)$$
$$S1\begin{cases} f01 = C3.\overline{C4} + C3.C4.C6 + C3.P36 \\ f11 = 0 \qquad\qquad g11 = 1 \end{cases} \quad (63)$$
$$S2\begin{cases} f02 = C4.C6 + \overline{C4}.\overline{C5} + C5.C7 \\ f12c = 0 \qquad\qquad g12c = C4 + C5.PC \end{cases} \quad (64)$$
$$S3\begin{cases} f03 = \overline{C5} \\ f13 = 0 \quad g13 = \overline{PC} \end{cases} \quad (65)$$
$$S4\begin{cases} f04 = C4.\overline{C6} \\ f14 = C5.C6 + C8 + C7 + P67 \quad g14 = C6 + C8 \end{cases} \quad (66)$$

L'algorithme peut ensuite être appliqué. Dans la première étape les *S'k* sont calculées à partir de l'équation (67).

$$S'k = \overline{f_{ok}}.g_{1k} + f_{1k} \quad (67)$$

La seconde étape permet de calculer les *Sk* à partir des *CSc*. L'ensemble de l'algorithme a été implanté avec succès dans un API avec la version ITS PLC PE mais aussi en IronPython avec une version de ITS PLC en cours de développement qui autorise l'utilisation de scripts. La solution algébrique des *Sk* est obtenue en développant les calculs de l'algorithme. La solution algébrique simplifiée alors obtenue est donnée par les équations (68) à (74).

$$S0 = \overline{C0} + (\overline{2P}).(\overline{C3} + C4.\overline{C6}.\overline{P36}) \quad (68)$$
$$S1 = \overline{C3} + C4.\overline{C6}.\overline{P36} \quad (69)$$
$$S2 = C5.\overline{C7}.PC + C4.\overline{C6}.C3 + C4.P36 \quad (70)$$
$$S3 = C5.GC \quad (71)$$
$$S4 = C6 + C8 + C7 + P67 \quad (72)$$
$$S5 = 1 \quad (73)$$
$$S6 = 1 \quad (74)$$

La commande obtenue (très simple) est sûre de fonctionnement et respecte le cahier des charges mais elle n'est pas optimisée d'un point de vue énergétique. En effet, *S1*, *S2*, *S4* et *S5* sont actionnées même si cela n'est pas nécessaire. Il est simple d'intégrer l'aspect énergétique dans les contraintes fonctionnelles. Ce travail sera présenté dans un prochain article. Une spécification classique ne séparant pas les aspects sécuritaires et fonctionnels aboutie à un GRAFCET (IEC60848) qui nécessitent de nombreuses synchronisations entre tâches et dont la sécurité n'est pas garantie et la lisibilité difficile.

7 CONCLUSIONS

Cet article a proposé une méthode de synthèse de la commande basée sur l'utilisation d'un filtre robuste aux erreurs de commande (représenté sous la forme d'un ensemble de contraintes logiques pouvant être simples ou combinées) qui a été formellement vérifié au moyen d'un *model-checker*. La commande obtenue est sûre même si les contraintes fonctionnelles sont erronées car seule une commande respectant les contraintes de sécurité est autorisée. La méthode de synthèse proposée utilise les concepts de l'approche SCT (autorisation et interdiction d'événements commandables) et certains résultats de la synthèse algébrique. L'implémentation est très simple dans un API et la séparation des parties « sécuri-

té » et « fonctionnelles » permet une gestion aisée des modes de marche et une grande flexibilité. Les perspectives de ce travail sont importantes car les résultats obtenus modifient la façon « classique » de concevoir la commande d'un système manufacturier piloté au moyen d'un API.

REMERCIEMENTS

Ce travail est issu des travaux de recherche menés dans le cadre du projet CPER 2007-2013 EDUCASCOL, et financés par la Région Champagne-Ardenne.

REFERENCES

Barragan Santiago I., Roth M., Faure J.M., 2006. Obtaining temporal and timed properties of logic controllers from fault tree analysis, *12th IFAC Symposium on Information Control Problems in Manufacturing*, Saint-Etienne, France, pp. 243-248

Benlorhfar R., Annebicque D., Gellot F., Riera B., 2011. Robust filtering of PLC program for automated systems of production, *18th World Congress of the International Federation of Automatic Control* , Milano, Italy, august.

Behrmann G., Bengtsson J., David A., Larsen K.G., Pettersson P., Yi W., 2002. Uppaal implementation secrets. *7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*.

Canet G., Couffin S., Lesage J.-J., Petit A., Schnoebelen P., 2000. Toward the automatic verification of PLC programs written in instruction list, *IEEE International Conference on Systems Man and Cybernetics*, Nashville, USA, pp. 2449-2454.

Cassandra C. G., Lafortune S., 1999. *Introduction ti discrete event systems*. Boston, MA: Kluwer Academic Publishers.

Chandra V., Kumar R., 2002. An event occurrence rules based compact modeling formalism for a class of discrete event systems, *Mathematical and computer modeling of dynamical Systems*, Vol. 8(1), pp 49-73.

Faraut G., 2010. *Commutations sûres de mode pour les systèmes à événements discrets*, thèse de doctorat, INSA de Lyon.

Faraut G., Piétrac L., Niel E., 2010. Control law synthesis and reconfiguration using SCT. In: *Conference on Control and Fault-Tolerant Systems, SysTol* Nice, France

Hietter Y., 2011. *Synthèse algébrique de la loi de commande, d'un système à évènements discrets logique*, thèse de doctorat, ENS Cachan.

Hietter Y., Roussel J.-M., Lesage J.-J. 2008a. Algebraic synthesis of dependable logic controllers *17th IFAC World Congress, Seoul (Korea)*, pp. 4132-4137.

Hietter Y., Roussel J.-M., Lesage J.-J. 2008b. Algebraic Synthesis of Transition Conditions of a State Model. *9th International Workshop On Discrete Event Systems, WODES'08*, pp. 187-192, Göteborg (Sweden).

Magalhães A. P., Riera, B., Vigario, B., 2010. Synthetic target systems in control education: lessons teachers are learning from students, *11th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems*, Valenciennes, France.

Marangé P., Benlorhfar R., Gellot F., Riera B., 2010. Prevention of human control errors by robust filter for manufacturing system, *11th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems*, Valenciennes, France.

Marangé P., Gellot F., Riera B., 2009. Industrial risk prevention by robust filter for manufacturing control system", *7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS'09)*, Barcelona, Spain.

Ramadge G., Wonham W. M., 1989. The control of discrete event systems, Proc. IEEE, *Special issue on DEDSs*, 77, pp.81-98.

Riera B., Vigario B., Chemla J-P., Correia L., Gellot F., 2009a. *10 ans de Maquettes Virtuelles pour l'enseignement des automatismes : de WINSIM en 1998 à ITS PLC Professional Edition en 2008*, J3eA 8 Hors-Série 1, 1004, DOI : 10.1051/j3ea: 2008045.

Riera B., Marangé P., Gellot F., Nocent O., Magalhaes A., Vigario B., 2009b. Complementary usage of real and virtual manufacturing systems for safe PLC training, *8th IFAC Symposium on Advances in Control Education (ACE'09)*. Kumamoto, Japan.

Riera B., Benlorhfar R., Annebicque D., Gellot F. , Vigario B., 2011. Robust control filter for manufacturing systems : application to PLC training, *18th World Congress of the International Federation of Automatic Control* , Milano, Italy.

Taillard P., 2010. L'analyse structurée par tâches. *Technologie.* N°170, Novembre-décembre. Pp. 38-43.

Villemeur A., 1988. *Sûreté de fonctionnement des systèmes industriels*, Eyrolles, « Collection de la direction des études et recherches d'Électricité de France », Paris (ISSN 0399-4198).