Etude du comportement global d'un SED en vue de la validation de sa commande spécifiée par Grafcet

Pascale Marangé, Abdelouahed Tajer, François Gellot, Véronique Carré-Ménétrier

Etude du comportement global d'un SED en vue de la validation de sa commande spécifiée par Grafcet

Pascale Marangé[1], Abdelouahed Tajer[2]
François Gellot[1], Véronique Carré-Ménétrier[1]

[1]Centre de Recherche en STIC – UFR Sciences Exactes et Naturelles de Reims - Moulin de la Housse
BP 1039
F-51687 Reims Cedex 2
[2] Polydisciplinary Faculty - Ibn Zohr University
P.B. 638
45000 Ouarzazate, Morocco

{pascale.marange, abdelouahed.tajer, francois.gellot, veronique.carre}@univ-reims.fr

RÉSUMÉ. *Suite à nos travaux sur la synthèse de la commande basée sur la théorie de Ramadge et Wonham, nous proposons une approche de validation de la commande hors ligne. Cette approche nécessite la modélisation de la partie opérative et des contraintes de sécurité et de vivacité, appliquées au système. Nous proposons de modéliser la partie opérative par une méthode à base de règles d'occurrence et de précédence et d'utiliser des équations logiques pour les contraintes pour pallier les problèmes de méthodologie de conception de modèles et d'explosion combinatoire. La démarche de validation de la commande permet de détecter les éventuels blocages ou les contraintes non respectées et d'aider le concepteur à apporter les modifications nécessaires. Pour illustrer ces concepts, nous proposons de les éprouver sur un exemple de tri de caisses.*

ABSTRACT. *Following our work on the control synthesis based on the Ramadge and Wonham theory, we propose an off line control validation approach. This approach requires modeling the plant, and some safety and liveness constraints. We propose to model the plant by a method based of occurrence and precedence rules to model the plant, and to use logical equations to model the constraints. The control validation approach makes it possible to detect possible deadlocking or the not respected constraints and to help the designer to make the modifications necessary. To illustrate these new concepts, we propose to test them on an example of case sorting.*

MOTS-CLÉS : *Systèmes à événements discrets, Grafcet, Synthèse de commande, Théorie de supervision, Validation.*

KEYWORDS: *Discrete Event Systems, Grafcet, Control synthesis, Supervisory control theory, Validation.*

1. Introduction

La complexité des Systèmes Automatisés de Production, accroît les exigences des utilisateurs en terme de sûreté de fonctionnement et d'aide à la conception de programmes. Pour aider le concepteur sur ces aspects, deux approches sont possibles (Faure et al., 2001) : la première par validation et la seconde par synthèse. La première approche consiste à laisser le concepteur du système de commande développer des lois de commande basées sur le cahier des charges et à analyser ensuite automatiquement une représentation formelle de ces lois de commande. Une telle analyse se fonde par exemple sur l'une des techniques d'analyse développées pour les automates à états (model-cheking (Bérard et al., 1999, Machado, 2006)) ou des techniques de calcul symbolique (theorem-proving (Roussel et al., 2002a et 2002b)). Il existe également des méthodes de validation à base de langages synchrones qui apportent de bons résultats sans les inconvénients des méthodes à base d'automates à états (Gaffé, 2003, Delaval et al., 2007). La deuxième approche dite de synthèse (Ramadge et al., 1989, Hellgren et al., 2002, Ghaffari et al., 2002, Gouyon et al., 2004), est prévue pour déduire directement des lois de commande des spécifications et du procédé sans réelle participation du concepteur au processus d'élaboration de la commande. Cette approche nécessite une modélisation formelle du procédé et des spécifications pour manipuler les modèles formels représentatifs.

Dans nos travaux précédents (Carré-Ménétrier et al., 2002), nous nous sommes intéressés au développement de techniques permettant la synthèse automatique d'une implantation correcte de la commande qui soit optimale vis-à-vis d'un modèle Grafcet (IEC, 1988). Ce modèle a été choisi en raison de sa convivialité et de sa large utilisation dans la spécification de la commande des automatismes séquentiels. Nous avons choisi d'asseoir notre démarche sur des outils/méthodes ayant de solides bases formelles, tels que les automates et la théorie de supervision des systèmes à événements discrets (SED) (Ramadge et al., 1989). Le travail a ainsi abouti au développement d'une approche globale de synthèse et d'implantation de la commande réactive la plus permissive vis-à-vis d'un grafcet, d'un modèle du procédé à commander et des contraintes à respecter sur les évolutions du procédé. Cependant, cette approche possède deux inconvénients majeurs liés à la difficulté de modéliser le procédé et les spécifications et à l'explosion combinatoire inhérente à tous SED. Elle est de plus très sensible aux erreurs de modélisation. Cette démarche conduit à une construction automatique sans retour d'information vers le concepteur et, si elle permet la génération d'une commande sûre, déterministe, réactive et sans blocage, elle ne permet pas de s'assurer que le résultat final est en adéquation avec les souhaits initiaux du concepteur.

Ces différentes réflexions ont orienté les travaux que nous présentons dans cet article, vers l'étude du comportement global du système en vue de valider la commande. L'idée est de conserver le principe de la démarche de synthèse et ses

outils/méthodes en l'adaptant pour effectuer de la validation. Ces travaux prennent en compte le besoin de réduire l'explosion combinatoire pour rendre l'approche applicable sur des systèmes réels et proposent :

– Pour la partie opérative, une méthodologie structurante à base d'automates booléens à travers une adaptation de l'approche de V. Chandra (Chandra et al., 2001).

– Pour les contraintes, un principe de modélisation par des équations logiques entre les entrées/sorties du contrôleur.

– Pour le concepteur de la commande, des informations précisant si des évolutions conduisent à un blocage, si des contraintes ne sont pas respectées au niveau de la commande, etc. Ces informations permettront d'agir en modifiant la commande proprement dite ou en relâchant des contraintes peut-être trop restrictives.

L'article est organisé en 6 sections. La section 2 présente les modèles et les principes de modélisation retenus pour la partie opérative et les contraintes. L'ensemble de ces concepts est alors appliqué à un exemple de tri de caisses détaillé dans la section 3. Dans la section 4, une démarche en cinq étapes est proposée pour renseigner le concepteur sur les éventuels problèmes engendrés par les évolutions de la commande qu'il a développée. Un des algorithmes présentés dans cette démarche s'inscrit, comme dans nos travaux précédents (Carré-Ménétrier et al., 2002), dans le cadre formel de la théorie de supervision (Ramadge et al., 1989). Il tient compte de la sémantique booléenne des modèles et la discussion proposée au lecteur démontre que cet algorithme respecte bien les conditions de commandabilité énoncées dans la théorie. Dans la section 5, nous présentons à travers l'exemple les informations présentées au concepteur pour l'aider à élaborer sa commande. Enfin, la partie 6 conclut l'article et présente quelques perspectives.

2. Principe de la modélisation de la partie opérative et des contraintes

La construction d'une commande correcte par synthèse automatique suppose l'élaboration de modèles pour la partie opérative et les spécifications. Dans nos travaux, les spécifications liées à la commande sont scindées en deux catégories : celles concernant la commande proprement dite et celles représentant les contraintes de sécurité et de vivacité. Les contraintes de sécurité correspondent à ce qu'il ne faut pas faire et les contraintes de vivacité correspondent au séquencement que doit respecter le système complet. L'utilisation de la théorie de supervision dans nos travaux antérieurs nous a conduits à modéliser les éléments de la partie opérative et les spécifications, par des modèles à base d'automates à états rudimentaires et événementiels qui manquent de convivialité et d'expressivité. Malgré différents travaux (Philippot et al., 2003), l'élaboration d'un modèle avec ces automates reste très intuitive et génératrice d'erreurs de modélisation et d'explosion combinatoire. De plus, les modèles élaborés conduisent à se poser plusieurs questions : Comment

vérifier la complétude du modèle, comment vérifier que les modèles de partie opérative sont indépendants des spécifications de commande ou des contraintes de fonctionnement imposées au système, comment intégrer de nouveaux éléments dans ces modèles sans tout recomposer ? Pour contribuer à résoudre une partie de cette problématique, nous proposons d'appuyer nos travaux sur une construction modulaire et structurée des modèles, les modèles de commande, de partie opérative et des contraintes étant explicites et indépendants entre eux. Cette section se compose de deux parties : la première est consacrée à la partie opérative et la seconde aux contraintes. La commande est modélisée quant à elle en Grafcet, son élaboration n'est pas traitée dans cet article.

2.1. *Modélisation de la partie opérative*

La modélisation explicite de la partie opérative est une tâche complexe qui se heurte non seulement à des problèmes méthodologiques de structuration et de composition de ses éléments mais aussi au choix de la granularité du comportement modélisé. Par conséquent, nous proposons dans la suite une méthode structurée à base de règles en s'appuyant sur des travaux de V. Chandra (Chandra et al., 2001, Chandra et al., 2002), pour aider le concepteur dans sa tâche de conception des modèles de partie opérative. Nous nous intéressons ici à une approche fonctionnelle sans prendre en compte les défauts et les aléas de fonctionnement.

2.1.1. *Règles d'occurrence et relations de précédence*

Pour conserver le bénéfice du cadre formel proposé par la théorie de supervision (Ramadge et al., 1989), la modélisation de la partie opérative s'effectue sous forme d'automates décrivant les évolutions physiquement possibles en fonctionnement normal du procédé par des événements simples. Pour refléter les interactions entre la commande spécifiée par Grafcet et la partie opérative, nous avons choisi l'interprétation de S. Balemi (Balemi et al., 1993) où les événements commandables Σ_c représentent les entrées du procédé et les événements non commandables Σ_u ses sorties. Cette interprétation est spécialisée de manière à concilier la nature continue des actions et des variables d'entrées du grafcet avec le caractère événementiel du modèle de la théorie. Ainsi, nous avons retenu la correspondance suivante : un événement commandable correspond soit à l'activation $\uparrow z$ soit à la désactivation $\downarrow z$ d'un ordre du grafcet, tandis qu'un événement non commandable est associé soit au front montant $\uparrow e$ soit au front descendant $\downarrow e$ d'une variable d'entrée du grafcet. Les ensembles Σ_c et Σ_u s'écrivent alors $\Sigma_c = +Z \cup -Z$ et $\Sigma_u = +E \cup -E$. Les ensembles d'événements +Z et -Z, correspondent aux activations et désactivations des ordres, tandis que les ensembles d'événements +E et -E reflètent les ensembles des fronts montant et descendant des entrées. Dans ce papier, nous avons retenu une modélisation d'un point de vue commande, c'est-à-dire que les entrées représentent les capteurs et les sorties les actionneurs.

Pour structurer la modélisation de la partie opérative, nous avons retenu une modélisation à base de règles définissant les interactions entre les événements commandables et les événements non commandables, et des relations entre ces événements non commandables (Philippot et al., 2004a). Il s'agit dans le premier cas de règles dites d'occurrence correspondant donc à l'influence des événements commandables sur les événements non commandables, et dans le second cas de relations de précédence représentant la chronologie entre les événements non commandables conséquents au même événement commandable. Ces règles définies par l'utilisateur sont ensuite traduites en automates à états. Nous précisons que cette analyse à base de règles d'occurrence et de relations de précédence, est appliquée à un modèle fonctionnel.

Avant de déterminer les règles d'occurrence, il faut commencer par fixer le contexte, c'est-à-dire les conditions initiales du système. Il s'agit ensuite de recenser tous les événements liés à l'élément de partie opérative que l'on cherche à modéliser puis de définir sous forme de règles, l'influence de l'activation et de la désactivation des événements commandables $\sum_c$ sur les événements non commandables $\sum_u$. Par conséquent, chaque règle va s'exprimer selon le principe simple d'une « cause/conséquence », la cause est liée à l'événement commandable $\downarrow z$ ou $\uparrow z$ en fonction de l'état du système et la conséquence porte sur l'événement non commandable $\uparrow e$ ou $\downarrow e$. Pour chaque règle d'occurrence ayant la même cause, il faut ensuite établir sous forme de relations de précédence, la chronologie liant les événements non commandables conséquents.

Formellement, le principe est le suivant :

– Il faut définir les conditions initiales des entrées et des sorties dans la phase de modélisation de la partie opérative. Ces états initiaux correspondent soit aux états actifs, soit aux états inactifs des entrées/sorties. Autrement dit, ce sont les valeurs initiales des événements commandables $\sigma \in \sum_c$ et non commandables $\alpha \in \sum_u$.

– L'interaction entre les sorties et les entrées permet de déterminer des règles d'occurrence de type cause/conséquence. Il s'agit en fait de déterminer s'il existe des événements non commandables α du système qui sont générés suite à l'occurrence d'un événement commandable σ en tenant compte de l'état du système β (état des entrées et des sorties). Formellement, l'écriture de cette proposition est la suivante : $\forall \sigma \in \beta \sum_c$ avec $\sum_c = +Z \cup -Z$, $\beta \in E \cup Z$ et si $\exists \alpha \in \sum_u$ avec $\sum_u = +E \cup -E$ tel que α est une conséquence de σ alors on définit la règle d'occurrence :

$$\sigma \Rightarrow \alpha \text{ avec } \begin{cases} \sigma = \beta \uparrow z \ \ ou \ \ \beta \downarrow z \\ \alpha = \uparrow e \quad ou \downarrow e \end{cases}$$

– Un événement commandable peut conduire à générer un ou plusieurs événements non commandables. Pour séparer ces événements non commandables en terme de précédence, nous définissons pour chaque règle d'occurrence ayant la même cause, une relation appelée « relation de précédence », permettant de préciser la chronologie liant les événements non commandables conséquents, telle que :

$\forall \sigma \in \sum_c$ si $\exists\ \alpha_1, \alpha_2 \in \sum_u \times \sum_u$ et $\sigma \Rightarrow \alpha_1$, $\sigma \Rightarrow \alpha_2$ alors on définit une relation de précédence entre α_1 et α_2 par :

$$\alpha_1 \rightarrow \alpha_2 \text{ ou } \alpha_2 \rightarrow \alpha_1.$$

A partir des règles d'occurrence et des relations de précédence, nous proposons de générer automatiquement un modèle automate à états à travers une méthode de construction dite booléenne.

2.1.2. *Construction de l'automate de partie opérative*

Les premiers travaux de Ramadge et Wonham (Ramadge et al., 1987) sur la théorie de supervision préconisaient l'emploi d'automates événementiels et rudimentaires mais pas très explicites car les états portent peu d'informations sur le système. De nombreux travaux ont proposé d'autres modèles pour représenter le système tels que les *State-Based Discrete-Event* introduit par (Wong, 1990), les *Vector Discrete-Event System* de (Li, 1991), les *Boolean Discrete Event System* qui sont un cas particulier des *Vector Discrete-Event System* avec une structure d'état à vecteur booléen (Wang, 2000).

Pour prendre en compte les informations sur l'état de la partie opérative et sur l'état de la commande, nous avons donc choisi la dernière structure et utilisant au niveau de chaque état de l'automate, une structure à base de vecteurs d'entrées et de sorties. Par conséquent, la définition de l'automate à états booléens retenus pour nos travaux est la suivante :

Un automate à états booléens est défini par un 4-uplet G = (P, $\sum$, δ, p_0), où P est l'ensemble des états de G, $\sum = \sum_u \cup \sum_c$, δ la fonction de transition définie par δ : $P \times \sum \rightarrow P$, l'état initial p_0 de l'automate G. P est un ensemble de vecteurs d'états booléens à n dimensions. Chaque composante du vecteur d'état peut prendre la valeur 0 ou 1. Le nombre d'états maximum du modèle automate est donné par le nombre de variables d'états, c'est-à-dire 2^n états au maximum pour n variables. A chaque état p de l'automate G est associé le vecteur d'entrées $E = [e_1,..., e_m]$, le vecteur de sorties $Z = [z_1,..., z_s]$ où m est le nombre d'entrées de la commande et s le nombre de sorties de la commande. Dans l'algèbre de Boole, l'ensemble IB est défini par ($\{0,1\}$, $\wedge$, $\vee$, $\neg$). L'ensemble P des états de G est donc défini par :

$P = \{[[e_1,\ldots, e_m] , [z_1,\ldots, z_s]] / e_i , z_j \in \{0,1\}^*\{0,1\}, i=1,\ldots,m, j=1,...,s \text{ et } m+s=n\} \subseteq IB^n$.

La construction de l'automate de la partie opérative est basée sur les règles et les relations et s'effectue selon les 4 étapes suivantes :

1. Construction de la table avec les 2^n états (n est le nombre d'entrées/sorties du système) décrivant toutes les combinaisons possibles entre les entrées/sorties du système. Cet ensemble correspond à l'ensemble des états P.

2. Suppression des incohérences logiques entre les entrées. Ici, nous ne nous intéressons pas aux incohérences logiques liées aux événements commandables mais à celles liées aux conséquences (événements non commandables) car seules les entrées de la partie opérative sont considérées. Après la suppression de ces incohérences logiques correspondant aux états p_{inc}, l'ensemble des états P est donc défini par : $P = P - \{p_{inc}\}$

3. Construction de l'automate équivalent à partir du diagramme des évolutions

commandables. Ce diagramme est déterminé à partir de la table de vérité représentant les états restant après la suppression des incohérences logiques. Pour chaque combinaison, l'occurrence d'un événement commandable $\uparrow z$ et/ou $\downarrow z$ de $\sum_c$ permet de changer la variable d'état de sortie ce qui conduit à un nouvel état. Le principe consiste à construire à partir d'un état p_k de P ($p_k=(E_k, Z_k)$ avec $Z_k = [z_{1k},\ldots, z_{sk}]$ et $E_k= [e_{1k},\ldots, e_{mk}]$) l'état atteignable p_{k+1} suite à l'occurrence d'un événement commandable $\uparrow z_{jk}$ ou $\downarrow z_{jk}$ avec $j=1,\ldots,s$ et $k=1,\ldots,2^n$.

4. Complément de l'automate avec les évolutions non commandables en utilisant les règles d'occurrence, les relations de précédence et les conditions initiales. Suite à l'occurrence d'un événement commandable, les règles d'occurrence permettent de déterminer l'état atteignable dans P. Les relations de précédence permettent de définir la séquence d'états atteignables par des événements non commandables, suite à l'occurrence d'un même événement commandable.

Cette méthode de modélisation est appliquée à chacun des éléments du système. Le modèle complet de la partie opérative est obtenu alors par la composition asynchrone de tous les modèles élémentaires. Nous obtenons un nombre fini d'états dans le modèle de PO qui est égal au maximum à 2^{nc+na} où nc et na représentent respectivement le nombre de capteurs et d'actionneurs. Malheureusement, les modèles obtenus ont leurs limites au niveau de la représentation du système, au niveau des notions temporelles,… En effet, les modèles étant seulement fonction de l'état des capteurs actionneurs, ils ne permettent pas de différentier toutes les situations. L'écart entre le modèle de PO proposé et le comportement réel du système existe et est lié essentiellement au fait que l'aspect temporel n'est pas pris en compte. Actuellement, nous nous orientons vers une méthodologie de modélisation bas niveau de la PO intégrant directement la technologie des pré-actionneurs, des actionneurs et des capteurs (Rohee et al., 2006).

2.2. *Modélisation des contraintes*

Dans notre approche de synthèse, les contraintes à modéliser sont des contraintes de sécurité et/ou des contraintes de vivacité. Intégrer des contraintes, consiste à inhiber des actions et/ou à agencer et séquencer l'exécution des commandes envoyées au procédé. Dans les SED, les contraintes s'expriment classiquement par des automates à états qui ne sont pas sans poser des problèmes de conception et d'interprétation. Pour faciliter l'écriture des contraintes sans utiliser des automates à états, nous préconisons une alternative basée sur l'utilisation d'équations logiques qui sont fonction de l'ensemble des entrées et de l'ensemble des sorties du Grafcet. L'utilisation des équations logiques permet de réduire l'explosion combinatoire lors de la génération de la commande. L'utilisation d'équations logiques ne nous permet pas de représenter une séquence d'événements bien précise.

Il existe quelques travaux utilisant les équations logiques dans l'algèbre de Boole. Parmi ces travaux, nous pouvons citer ceux de Roussel (Roussel et al., 2003), qui utilisent une approche algébrique pour développer une méthode de

synthèse formelle. Cette approche permet d'obtenir, pour un système logique, une commande conforme aux spécifications. Elle repose sur la formalisation des spécifications sous forme d'assertions dans une algèbre de Boole adaptée aux signaux binaires appelé Algèbre II (Roussel, 1994). L'algèbre II permet de décrire des comportements intégrant des aspects temporels, en manipulant cette fois des signaux binaires fonction du temps et des variables booléennes. Les travaux présentés ici sur la modélisation des contraintes s'effectuent dans l'algèbre de Boole classique sans aspect temporel.

Le cadre formel de la modélisation des contraintes est basé sur l'algèbre de Boole des propositions pour laquelle les éléments manipulés sont des valeurs binaires 0 ou 1 qui représentent les valeurs des entrées E et des sorties Z du grafcet de spécification. Ces éléments sont représentés par des fonctions définies dans IB. Nous ne développons pas cette partie dans cet article, le lecteur peut retrouver la définition de ce cadre formel dans (Tajer, 2005).

L'emploi des équations logiques booléennes a pour avantage d'exprimer les contraintes de manière explicite et flexible, dans un langage bien connu du concepteur. L'écriture de ces contraintes se détermine simplement en utilisant le connecteur d'implication « Si …Alors… » entre les événements. Cette implication va permettre d'exprimer les contraintes de sécurité et de vivacité uniquement sous forme statique. La logique utilisée nous limite à des contraintes statiques dépendant seulement du vecteur d'entrées sorties. Cela ne nous permet donc pas d'exprimer des contraintes temporelles ou des séquences d'événements. Par exemple, dans notre démarche, la dynamique des vérins (comme par exemple la vitesse d'avancement) n'est pas considérée. Nos travaux actuels (Marangé et al., 2007) visent à améliorer la modélisation des contraintes en proposant une découpage en fonction de leur type (sécurité ou vivacité) et de leurs conditions intrinsèques (statique, dynamique).

3. Illustration sur un exemple

Pour illustrer ces travaux, nous avons choisi une application dont l'objectif est de trier automatiquement des caisses de deux tailles différentes (figure 1). Le processus industriel se compose du convoyeur 1 qui apporte les caisses les unes après les autres, d'un axe composé de deux vérins double effet V1 et V2, de deux vérins simple effet V3 et V4, et de deux convoyeurs (convoyeur 2 et 3) permettant l'évacuation des caisses. Selon la taille de la caisse, l'axe composé de V1 et de V2, place les caisses devant le vérin V3 ou devant le vérin V4.

La notation est la suivante : v_{i0} correspond à la position rentrée du vérin V_i, v_{i1} à la position sortie du vérin V_i, cp_{11} correspond au capteur de petite caisse et cp_{12} au capteur de grande caisse. Les sorties du système sont définies par : *TAPIS* correspond à l'ordre d'activer les trois convoyeurs, *AVANCER1* (*AVANCER2*) à l'ordre de sortie du vérin V1 (resp.V2), *RECULER1* (*RECULER2*) à l'ordre de rentrée du vérin V1 (resp. V2), *AVANCER3* (*AVANCER4*) à l'ordre de sortie du

vérin simple effet V3 (resp. V4). Du point de vue de la commande, les vecteurs d'entrées et de sorties correspondant au système de tri de caisses, sont définis de la façon suivante : E = {v_{10}, v_{11}, v_{20}, v_{21}, v_{30}, v_{31}, v_{40}, v_{41}, cp_{11}, cp_{12}}, Z = {*TAPIS*, *AVANCER1*, *AVANCER2*, *RECULER1*, *RECULER2*, *AVANCER3*, *AVANCER4*}.

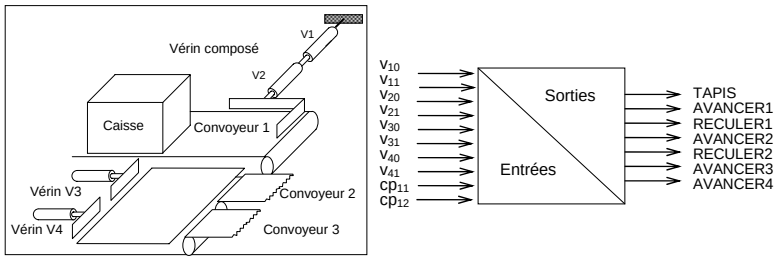


Figure 1. *Système du tri de caisses*

3.1. *Modélisation de la partie opérative*

Le modèle de la partie opérative se compose de 4 automates décrivant le mouvement de chacun des 4 vérins avec une prise en compte de la technologie du vérin. Nous ne détaillons dans cette partie que le mouvement associé au vérin V1.

3.1.1. *Etablissement des règles d'occurrence et de précédence*

Le mouvement du vérin double effet V1 peut être modélisé selon les quatre événements commandables $\downarrow$*AVANCER1*, $\uparrow$*AVANCER1*, $\downarrow$*RECULER1*, et $\uparrow$*RECULER1* et les quatre événements non commandables $\uparrow v_{10}$, $\downarrow v_{10}$, $\uparrow v_{11}$ et $\downarrow v_{11}$. L'ensemble des paramètres décrivant le mouvement du vérin V1 est présenté dans le tableau 1.

Conditions initiales	Règles d'occurrence	Relations de précédence
$\overline{AVANCER1}$	$v_{10} \wedge \uparrow AVANCER1 \Rightarrow \downarrow v_{10}$ $\neg v_{10} \wedge \neg v_{11} \wedge \uparrow AVANCER1 \Rightarrow \uparrow v_{11}$	$\downarrow v_{10}$ *précède* $\uparrow v_{11}$
$\overline{RECULER1}$	$v_{11} \wedge \uparrow RECULER1 \Rightarrow \downarrow v_{11}$ $\neg v_{10} \wedge \neg v_{11} \wedge \uparrow RECULER1 \Rightarrow \uparrow v_{10}$	$\downarrow v_{11}$ *précède* $\uparrow v_{10}$
$\overline{v_{11}}$	$\neg v_{10} \wedge \downarrow AVANCER1 \Rightarrow \uparrow v_{11}$	*Aucune*
v_{10}	$\neg v_{11} \wedge \downarrow RECULER1 \Rightarrow \uparrow v_{10}$	*Aucune*

Tableau 1. *Paramètre du mouvement du vérin V1*

– Nous fixons comme état initial que le vérin est en position rentrée et qu'aucun ordre n'est envoyé à la partie opérative

– Pour définir les règles d'occurrence (colonne 2 du tableau 1), il est nécessaire d'observer les conséquences des ordres envoyés pour un état donné du système. L'ordre *AVANCER1* rend possible le déplacement du vérin et lui permet de quitter sa position rentrée v_{10}. Par conséquent, la désactivation de v_{10} est une conséquence de

l'activation de *AVANCER1*. De même, l'activation de v_{11} est une conséquence de l'activation de *AVANCER1* lorsque le vérin est entre v_{10} et v_{11}.

– Pour établir les règles de précédence (colonne 3 du tableau 1), il faut définir la chronologie des événements non commandables. Par exemple, dans le cadre de *AVANCER1*, la désactivation de v_{10} précède l'activation de v_{11}

– La règle d'occurrence liée à la désactivation de l'ordre *AVANCER1* (ligne 3 du tableau 1) traduit le fait qu'à partir du moment où, sous l'influence de l'ordre, le vérin quitte sa position de départ formulé par un front descendant sur v_{10}, la désactivation de l'ordre devient inopérante et le vérin continue sa course jusqu'à ce que v_{11} soit vrai. Le raisonnement est identique pour la ligne suivante du tableau 1 avec l'ordre *RECULER1*.

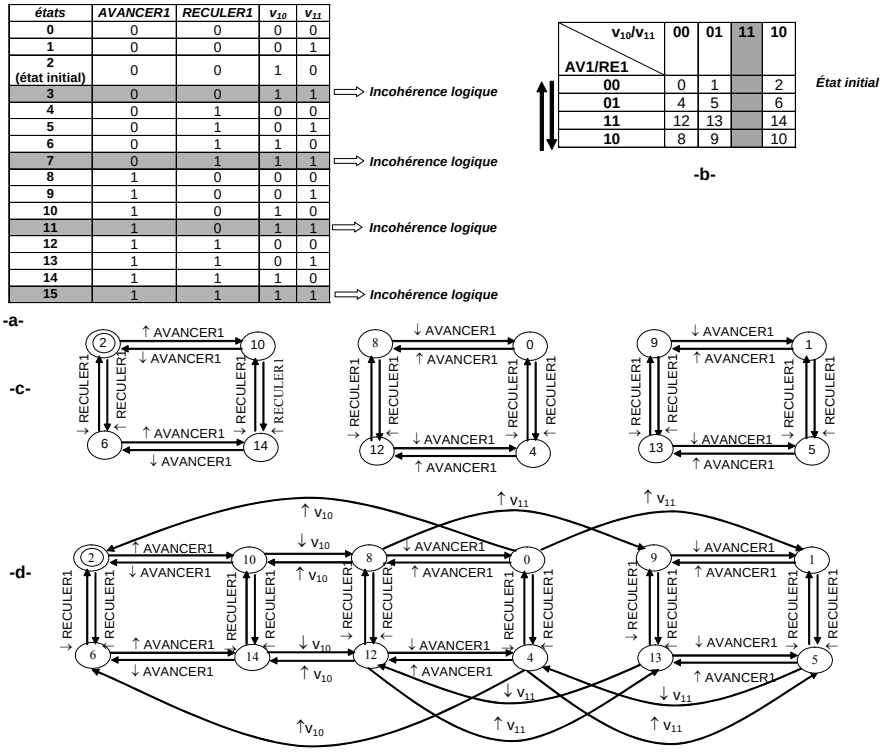
3.1.2. *Construction de l'automate de partie opérative*

Pour le mouvement du vérin V1, la première étape consiste à établir les 2^4 combinaisons entre *AVANCER1*, *RECULER1*, v_{11} et v_{10} (figure 2.a). La deuxième étape consiste à supprimer les 4 incohérences logiques d'événements liées au fait que v_{10} et v_{11} ne peuvent pas être vrais en même temps, nous en déduisons alors le diagramme des évolutions commandables (figure 2.b). L'automate des évolutions commandables est ensuite déterminé à partir du diagramme des évolutions exprimant de ce fait toutes les évolutions possibles entre les états. Ces évolutions sont définies uniquement par les événements commandables : $\uparrow$*AVANCER1*, $\downarrow$*AVANCER1*, $\uparrow$*RECULER1*, $\downarrow$*RECULER1*. Sachant que *AVANCER1* et *RECULER1* peuvent soit prendre la valeur 1 pour l'activation, soit 0 pour la désactivation (figure 2.c).

L'application des règles d'occurrence et des relations de précédence engendre ensuite l'automate final présenté figure 2.d et permet de faire la jonction entre les trois blocs de la figure 2.c.

A partir de l'état initial (état 2 figure 2.d), les événements commandables possibles sortants sont $\uparrow$*AVANCER1* et $\uparrow$*RECULER1*. L'occurrence de l'événement commandable $\uparrow$*AVANCER1* conduit alors l'automate à l'état 10. La relation d'occurrence correspondant à cet événement est définie par l'événement non commandable $\downarrow v_{10}$ permettant ainsi de faire la jonction avec l'état 8. A partir de cet état, deux cas peuvent se produire : occurrence de $\downarrow v_{10}$ qui entraîne l'apparition de l'événement non commandable $\uparrow v_{11}$ (cf tableau 1) conduisant l'évolution vers l'état 9 ou occurrence de $\downarrow$*AVANCER1* conduisant l'évolution vers l'état 0. A partir de l'état 9, l'occurrence de $\downarrow$*AVANCER1* conduit vers l'état 1 et à partir de l'état 0, l'occurrence de $\uparrow v_{11}$ conduit vers l'état 1 également. Donc, quel que soit l'ordre d'arrivée des événements commandables ou non, l'état d'arrivée est le même.

Les évolutions non commandables vont venir s'ajouter de la même manière entre les trois automates des évolutions commandables (figure 2.b). Il est à noter que ces évolutions ne sont possibles que lorsqu'une seule valeur des entrées v_{11} ou v_{10} change à la fois.



états	AVANCER1	RECULER1	v_{10}	v_{11}	
0	0	0	0	0	
1	0	0	0	1	
2 (état initial)	0	0	1	0	
3	0	0	1	1	⇨ Incohérence logique
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	⇨ Incohérence logique
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	⇨ Incohérence logique
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	⇨ Incohérence logique

AV1/RE1 \ v_{10}/v_{11}	00	01	11	10
00	0	1		2
01	4	5		6
11	12	13		14
10	8	9		10

Figure 2. *Etapes de la construction de l'automate du mouvement horizontal*

Pour obtenir l'automate complet de la partie opérative, nous appliquons la même méthode pour chacun des éléments du système et nous effectuons le produit asynchrone entre les modèles élémentaires. Pour cet exemple du tri de caisses, nous obtenons au final un modèle à 15552 états et de 163295 transitions.

3.2. *Modélisation des contraintes*

Dans cet exemple, nous nous intéressons à des contraintes au niveau des vérins eux-mêmes et à leur interaction avec les autres vérins ou avec les caisses manipulées par le système. Le vérin V1 ne peut pas rentrer et sortir en même temps, c'est-à-dire que l'envoi simultané des deux ordres *AVANCER1* et *RECULER1* doit être interdit. Cette spécification s'exprime par un modèle automates à trois états (figure 3.a) qui est à comparer à l'équation logique de la figure 3.b obtenue à travers l'implication suivante : « Si *AVANCER1=1* Alors *RECULER1 = 0* » ou « Si *RECULER1=1* Alors *AVANCER1=0* » qui n'est autre que le complément du ET logique entre *AVANCER1* et *RECULER1*.

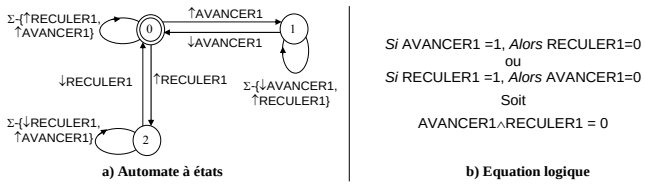


Figure 3. Contrainte **«** *Ne pas AVANCER1 et RECULER1 en même temps »*

La liste ci-dessous présente les cinq contraintes à respecter par le système : la première contrainte comme la deuxième s'applique au vérin V1 et au vérin V2 pour lesquels il est interdit d'effectuer des mouvements en sortie et en entrée simultanément. La troisième et quatrième contrainte concernent exclusivement le vérin V3 pour lequel il est interdit de sortir si (i) le vérin V1 est sorti et/ou si (ii) le vérin V2 est sorti. La dernière contrainte interdit la sortie du vérin V2 lorsqu'une petite caisse se présente. L'implantation des capteurs cp11 et cp12 conditionne fortement la véracité de cette contrainte. Par la suite (§ 5.2.3), nous verrons cette problématique apparaître.

$AVANCER1 \wedge RECULER1 = 0$ *(1)*
$AVANCER2 \wedge RECULER2 = 0$ *(2)*
$AVANCER3 \wedge \neg v_{10} = 0$ *(3)*
$AVANCER3 \wedge \neg v_{20} = 0$ *(4)*
$AVANCER2 \wedge cp_{11}=0$ *(5)*

Il n'y a pas de contraintes particulières sur *AVANCER4*, car nous considérons qu'il n'y a pas de problèmes techniques à ce niveau.

4. De la synthèse automatique de la commande vers une approche de validation de la commande

4.1. *Contexte de la démarche de synthèse automatique*

La démarche de synthèse automatique hors ligne que nous avons développée dans nos précédents travaux, a pour objectif la construction d'une commande sûre, sans blocage, déterministe et réactive. Elle comprend six étapes présentées figure 4.

1- L'étape de modélisation (étape 1) est décomposée en trois phases : (i) l'élaboration du Grafcet correspondant à la spécification de la commande, (ii) la modélisation de la partie opérative et (iii) la spécification des contraintes de sécurité et de vivacité.

2- L'opération de synthèse (étape 2) consiste à partir des contraintes et du procédé modélisés précédemment, à générer l'automate du superviseur *SUP* selon l'algorithme de synthèse proposé par (Kumar, 1991) Cet automate correspond au comportement commandable maximal admissible du procédé par rapport aux contraintes.

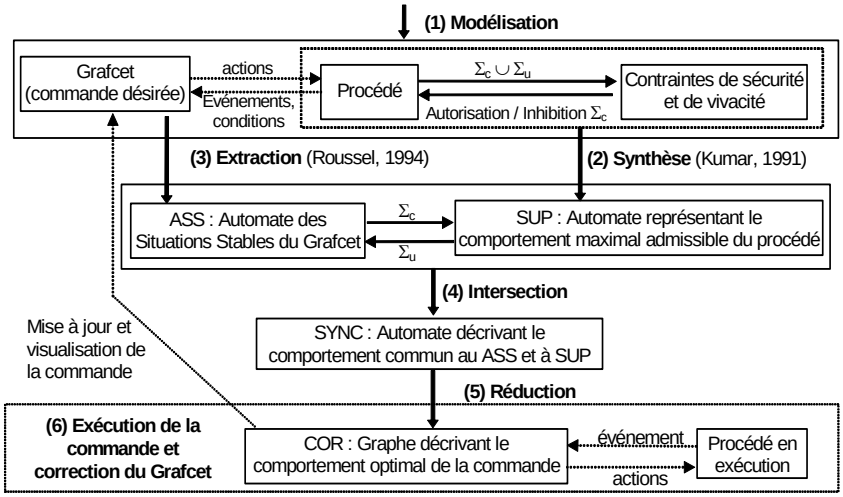


Figure 4. Les étapes de l'approche de synthèse hors ligne

3- L'extraction (étape 3) consiste à appliquer l'algorithme de (Roussel, 1994) afin de traduire le modèle Grafcet en automate équivalent noté *ASS* (Automate des Situations Stables). L'extraction est définie jusqu'à présent en respectant la norme Grafcet (IEC, 1988). Les concepts introduits par la révision technique générale de la norme (IEC, 2002) (événement d'entrée, événement interne, assignation, affectation, forçage, macro-étape et encapsulation) ne sont pas encore pris en compte et leur modélisation reste à étudier.

4- L'objectif de l'étape 4 est de générer l'automate nommé *SYNC* qui décrit le comportement commun aux automates *SUP* et *ASS* de manière à ne retenir, dans le modèle de commande, que les évolutions autorisées par le superviseur. L'algorithme d'intersection parcourt les automates *SUP* et *ASS* et construit *SYNC* à partir des événements admissibles par les deux automates. Le principe d'intersection est particulier car chacun des automates *ASS* et *SUP* repose sur une sémantique propre (Carré-Ménétrier et al., 2002). Pour *ASS*, un état est muni de l'ensemble des actions actives lors de la situation correspondante du grafcet, et les transitions sont décrites par une fonction composée des entrées du grafcet et de fronts sur ces entrées. Quant à *SUP*, il correspond à un automate rudimentaire, où les transitions sont uniquement décrites par des événements.

L'automate *SYNC* se construit en tant que modèle asynchrone dont les transitions correspondent à des événements simples. Les ordres qui doivent être émis en parallèle dans une situation du grafcet, sont représentés dans *SYNC* par une structure arborescente composée de transitions décrivant les activations et les désactivations successives autorisées par le superviseur. Les états reliés par ces transitions constituent une région correspondante à la dite situation du grafcet. Ainsi, l'automate *SYNC* est constitué d'un ensemble de régions (figure 5). Les transitions intra-région correspondent aux événements commandables, et les transitions inter-

régions aux événements non commandables.

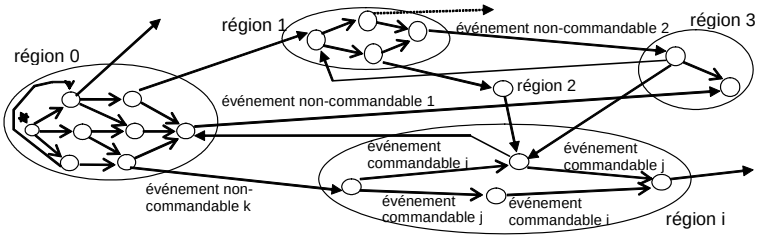


Figure 5. Modèle sémantique de l'automate SYNC

5- L'étape 5 est une étape de réduction qui s'effectue en trois phases : dans la première phase, les états bloquants de SYNC sont rendus non atteignables par suppression de leurs transitions amont, la deuxième phase dite d'optimisation consiste à retirer de l'automate *SYNC* modifié lors de la première phase, les transitions qui sont non atteignables durant l'exécution réelle et enfin, la troisième phase procède à l'agrégation des situations instables, de manière à aboutir à une commande réactive qui évolue, suite à l'occurrence d'un événement non commandable, d'une situation stable à une autre. L'automate résultant de l'étape de réduction (*COR*) est ensuite implanté sur Automate Programmable Industriel (A.P.I.) et correspond à la restriction minimale du comportement du grafcet qui garantit la conformité par rapport aux contraintes spécifiées et le non-blocage du système.

6- La dernière phase (étape 6) concerne l'exécution de la commande optimale implantée (*COR*) et la visualisation des corrections apportées au grafcet originel.

4.2. *Démarche de validation*

Pour obtenir une réelle adéquation des desiderata du concepteur avec la commande obtenue et donc développer un réel concept de validation et non plus de synthèse automatique de la commande, certaines fonctionnalités de ces étapes doivent être modifiées et nous proposons dans l'étape 5 de procéder à l'analyse de l'intersection entre la commande et le comportement maximal admissible du procédé (figure 6). En effet, les informations contenues dans cette intersection sont extrêmement riches et peuvent renseigner le concepteur du modèle de commande sur les évolutions que prennent la commande comme par exemple une évolution vers une situation de blocage ou une évolution ne respectant pas certaines contraintes. L'analyse de cette intersection permet de plus au concepteur du modèle du système qui n'est pas obligatoirement le concepteur de la commande, de déterminer si certaines contraintes ne sont pas trop restrictives et permet également de relever d'éventuelles erreurs dans les modèles de partie opérative.

A l'issue de la phase de validation, nous pouvons proposer au concepteur comme dans la démarche de synthèse présentée au §4.1 d'implanter l'automate d'intersection obtenu à la fin de la phase de validation. Si le concepteur a

correctement intégré toutes les informations de la phase de validation, il ne devrait plus y avoir de blocage et d'évolutions ne respectant pas les contraintes. L'automate *SYNC* obtenu à travers cette phase de validation est alors agrégé comme indiqué dans l'étape de réduction de la démarche automatique de synthèse. L'automate obtenu nommé COR, est ensuite traduit dans un langage de programmation d'automate programmable industriels, tels que le digramme LADDER, le texte structuré ou la liste d'instructions (IEC 2003).

La commande implantée résulte donc de la phase de validation et est, par conséquent, différente du grafcet initial fourni par le concepteur en tout début de la phase de validation. Cette méthode d'implémentation nous permet de vérifier le principe de WYPIWYE[1] (Berry, 1989) car le modèle qui est validé est le même qui nous permet de générer le code pour l'automate.

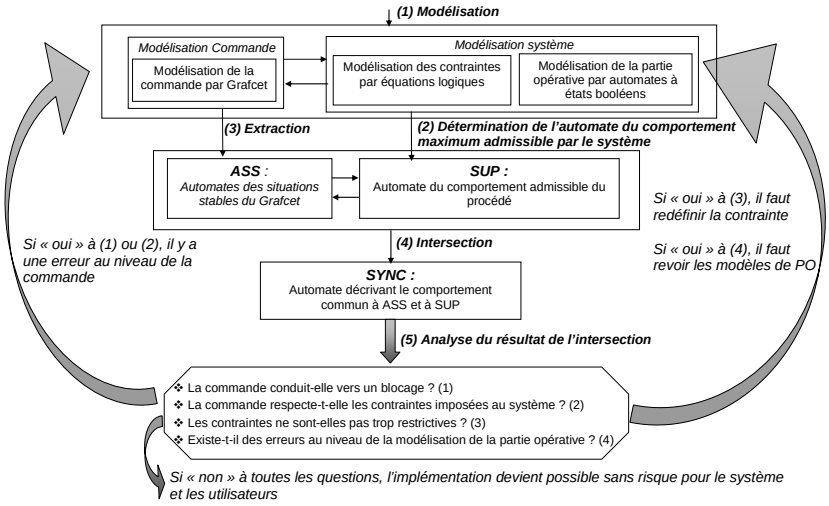


Figure 6. Informations déduites à partir des différents modèles

Dans la suite de l'article, nous proposons de formaliser l'algorithme permettant de générer le modèle du superviseur (*SUP*) que nous illustrons sur l'exemple du tri de caisses.

4.3. *Détermination du comportement maximal admissible du procédé vis-à-vis de contraintes pour des SED booléens*

4.3.1. *Algorithme*

Pour formaliser l'algorithme support de notre démarche (étape 2, figure 6), nous

[1] WYPIWYE : What You Prove Is What You Execute

reprenons le principe de commandabilité définie par Ramadge et Wonham en l'adaptant aux SED booléens. Cette adaptation s'exprime de la manière suivante : pour un procédé G défini par le 4-tuplet (P, Σ, δ, p_0), il faut que la proposition suivante (Wang, 2000) soit vérifiée :

$\forall p \in P$, tel que p vérifie K

Si $\exists \alpha \in \Sigma u$ tel que $\delta(p,\sigma)$ vérifie K

Alors K est commandable, K est l'ensemble des spécifications à respecter par le système.

En d'autres termes, la spécification K est commandable si et seulement si l'occurrence d'un événement non commandable α à partir d'un état vérifiant K ne conduit pas le procédé hors de la spécification K, c'est-à-dire vers un état qui ne vérifie pas K. Pour faciliter la mise en pratique de cette proposition, nous avons choisi, pour l'algorithme de synthèse (Tajer, 2005), d'utiliser la notion d'états défendus et faiblement défendus où les états défendus représentent des états ne respectant pas la spécification K lors de l'occurrence d'un événement non commandable σ et les états faiblement défendus représentent des états conduisant vers un état défendu sous l'occurrence d'un événement non commandable α.

L'algorithme consiste, à partir du modèle global du procédé et des contraintes modélisées selon les principes précédents, à générer l'automate à états booléens du superviseur *SUP*. L'automate *SUP* est décrit par 6-uplet (Σ, Q, Δ, E, Z, q_0), avec Σ l'ensemble des événements, Q l'ensemble des états, q_0 l'état initial et Δ une fonction de transition partielle $\Delta: Q \times \Sigma \rightarrow Q$, un vecteur d'entrées E, et un vecteur de sorties Z. Une transition de l'automate *SUP* est définie par un triplet (q, α, q'), q étant l'état de départ, α l'événement correspondant à la transition et q' l'état d'arrivée. La fonction de transition peut être étendue aux séquences d'événements de la manière suivante : Soit Σ* l'ensemble des séquences d'événements de longueur finie, ε la séquence vide et w une séquence finie d'événements :

$\Delta : Q \times \Sigma^* \rightarrow Q \qquad \Delta(q, \varepsilon) = q$ et $\Delta(q, w\alpha) = \Delta(\Delta(q,w), \alpha)$

Cet algorithme reçoit en entrée un automate à états booléens G représentant le modèle global du procédé, un ensemble K de spécifications logiques représentant les contraintes de sûreté et de vivacité ainsi que la structure initiale de l'automate résultant donnée par (Σ, $\{q_0\}$, δ, q_0). Un état q de *SUP* est caractérisé par (état de G, état du vecteur E, état du vecteur Z) enfin l'état initial s'exprime par $q_0 = (p_0, E_0, Z_0)$. Cet algorithme est basé sur trois étapes :

La première étape (figure 7.a) traite les événements commandables liés à l'activation ou à la désactivation d'une sortie de la commande (↓z ou ↑z). Ces événements décrivent les ordres envoyés de la commande vers la partie opérative. Les contraintes sont structurées en fonction des événements commandables Σ_c. Ainsi, pour chaque événement commandable $\sigma \in \Sigma_c$ est associé un ensemble de contraintes K_σ, $K_\sigma \subset K$. Par conséquent, une itération consiste, pour une évolution commandable ((q, σ, q'), ($\sigma \in \Sigma_c$)), à vérifier l'ensemble des contraintes K_σ. Cette vérification est obtenue par la valeur du vecteur d'entrées E_q, du vecteur de sorties Z_q et de l'événement *s* (l'occurrence de l'événement commandable *s* permet de changer la valeur de l'élément correspondant dans le vecteur Z pour obtenir $Z_{q'}$).

– Si dans l'ensemble K_σ, les équations logiques associées à σ sont vraies alors l'évolution correspondante n'est pas défendue et la transition correspondante est ajoutée à l'ensemble des transitions Δ. En effet, cet ordre est prévu par la commande et autorisé par le superviseur *SUP*. L'état aval q', atteint depuis q suite à l'occurrence de σ, se distingue de l'état amont q par le vecteur de sorties $Z_{q'}$. L'état q' est ensuite ajouté à l'ensemble des états Q s'il n'en fait pas déjà partie.

– Si l'ensemble des contraintes K_σ n'est pas vérifié (les équations ne sont pas vraies), alors l'évolution (q, σ, q') est défendue.

a- Première étape :

$\forall q \in Q$ faire :

Si $\exists\ \sigma \in \sum_c$ tel que $\Delta(q,\sigma) = q'$ **Alors**
Si K_s est vrai **Alors**
- $\Delta = \Delta \cup \{(q, \sigma, q')\}$;
$Q = Q \cup \{q'\}$ Si $q' \notin Q$.

Sinon
(q, σ, q') est une évolution défendue

b- Deuxième étape :

1er pas : **Si** $\exists\ \Delta(q, \alpha)=q'$ ($q' \in Q$, $\alpha \in \sum_u$) tel que q' ne vérifie pas K
Alors q est défendu

2ème pas : **Si** $\nexists\ q \in Q$ tel que q un état défendu
Alors Fin.
Sinon $\forall q \in Q$
Si
- q n'est pas défendu,
- $\exists\ w \in \Sigma^*$ tel que ($\forall \alpha \in w$, $\alpha \in \sum_u$) et ($q_w = \Delta(q, w)$ et q_w est défendu)

Alors q est faiblement défendu.

c- Troisième étape :

$\forall q \in Q$ tel que q est un état défendu ou faiblement défendu.
- $\forall\ (q, \alpha, q') \in \Delta$, $\Delta = \Delta - \{(q', \alpha, q)\}$.
- $\forall\ (q', \alpha, q) \in \Delta$, $\Delta = \Delta - \{(q, \alpha, q')\}$.
- $Q = Q-\{q\}$.
- $\forall\ q' \in Q$ tel que $\nexists (w \in \Sigma^*$ et $q' = \Delta(q_0,w))$ faire $Q = Q-\{q'\}$

Figure 7. *Algorithme de synthèse*

La deuxième étape (figure 7.b) consiste à construire s'il n'existe pas déjà, à partir d'un état courant q, les évolutions de *SUP* caractérisées par des événements non commandables. Ces événements non commandables correspondent aux réactions de la partie opérative par rapport aux ordres de la commande. Dans cette étape, il y a deux pas :

1. Le premier pas concerne l'identification des états défendus conduisant vers des états à interdire où les contraintes K ne sont pas vraies. Pour cet état q, il existe un événement non commandable α possible à partir de l'état correspondant du procédé q' tel que l'état atteignable ne vérifie

pas K. D'après la définition de la commandabilité, si le comportement du système ne contient aucun état défendu alors il est commandable et peut être considéré en tant que superviseur.

2. Le deuxième pas sert à identifier les états faiblement défendus. Ce sont tous les états à partir desquels il existe une séquence d'événements *w* non commandables permettant d'atteindre un état défendu.

La troisième étape (figure 7.c) consiste à retirer dans un premier temps tous les états défendus et faiblement défendus ainsi que toutes les transitions amont et aval qui leurs sont associées. Dans un second temps, les états non atteignables à partir de l'état initial sont retirés de l'automate obtenu.

4.3.2. *Discussion*

D'après la théorie de supervision, le superviseur obtenu par synthèse doit être commandable. Les travaux de Wang (Wang, 2000) vérifient la commandabilité par les deux propriétés suivantes :

– ($\forall q \in Q, \alpha \in \Sigma_u$) si q vérifie l'ensemble des contraintes K et s'il existe une fonction de transition $\Delta(q, \alpha)$ alors $\Delta(q, \alpha)$ doit vérifier l'ensemble K. Cela signifie que l'occurrence d'un événement non commandable ne doit pas conduire dans un état ne vérifiant pas les contraintes.

– ($\forall q \in Q$) si q vérifie l'ensemble des contraintes K alors ($\exists q_1,\ldots, q_n \in Q$, et $\alpha_1,\ldots, \alpha_n \in \Sigma$) tels que $q_n=q$ et $\Delta(q_{i-1}, \alpha_i) = q_i$ doit vérifier l'ensemble K pour i variant de 1 à n. En d'autres termes, à partir de n'importe quel état vérifiant l'ensemble des contraintes, tous les états par lesquels l'état q_n est atteint, doivent vérifier l'ensemble des contraintes K.

Nous allons faire de même en montrant pour chacune d'elles, comment notre algorithme vérifie ces conditions :

– La première condition de commandabilité est vérifiée de manière triviale par la construction définie dans l'algorithme. Par l'intermédiaire du pas n°2 (figure 7.b), si un état est accessible par un événement non commandable et qu'il ne vérifie pas l'ensemble des contraintes K, alors l'état qui conduit vers cet état est défini comme défendu ou faiblement défendu, et éliminé ainsi que les transitions associées à chacun de ces états.

– La seconde condition se détermine de la même manière, la construction de l'automate du superviseur par l'algorithme proposé permet d'éliminer les états ne respectant pas les contraintes et ainsi obtenir un ensemble d'états atteignables qui respectent l'ensemble des contraintes. Donc, une séquence d'événements commandables ou non commandables va toujours conduire vers un état vérifiant l'ensemble des contraintes.

L'algorithme proposé répond par conséquent, au critère de commandabilité de la théorie de supervision. En utilisant les notions d'états défendus et faiblement défendus, l'algorithme proposé peut être vu comme une adaptation aux automates

booléens de l'algorithme de Kumar (Kumar, 1991). Dans le cas de cet algorithme, la complexité en nombre d'états s'exprime dans le pire cas, par $M^k 2^{\Sigma c+\Sigma u}$ où M est le nombre d'états de l'automate des contraintes et k le nombre de contraintes. En conservant cette notation, l'algorithme proposé dans cet article génère $2^{\Sigma c+\Sigma u}$ états. Cette importante diminution vient du fait qu'aucun produit synchrone n'est effectué entre le modèle du procédé et celui des contraintes.

4.3.3. *Application à l'exemple*

Pour obtenir le comportement maximal admissible du procédé par rapport aux contraintes, il faut utiliser l'algorithme décrit précédemment. Les conditions initiales sont définies par les vecteurs d'entrées et de sorties : E_0=[1,0,1,0,1,0,1,0,0,0] et Z_0=[1,0,0,0,0,0,0,0].

Depuis l'état p_1 de l'automate de la partie opérative (figure 8.a), une des transitions de sorties est définie par l'événement $\uparrow RECULER1$. Cette évolution est défendue parce que l'occurrence de cet événement ne vérifie pas la contrainte (1). L'évolution (q_1, $\uparrow RECULER1$, q_9) sera ainsi enlevée de l'automate booléen du superviseur.

L'occurrence de l'événement non commandable $\uparrow v_{11}$ de l'état p_{51} peut conduire à l'état p_{60}. Cependant, dans cet état la contrainte (3) n'est pas respectée. En conséquence, l'état p_{51} est un état défendu dans l'automate du superviseur.

De l'état p_{13} (E_{13} = [1,0,1,0,1,0,1,0,0,0], Z_{13} = [1,1,0,0,0,1,0]), l'événement non commandable $\downarrow v_{10}$ conduit vers l'état défendu p_{51}, en conséquence, l'état q_{13} sera défini comme un état faiblement défendu.

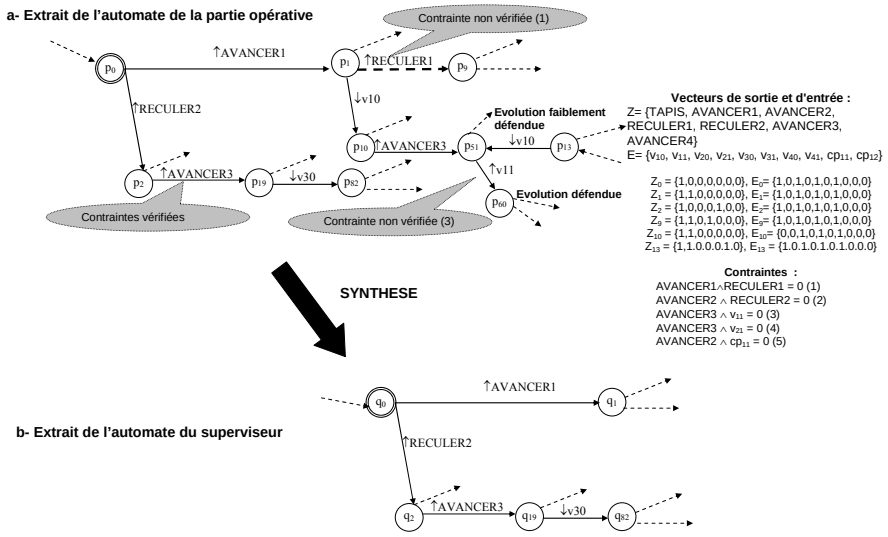


Figure 8. *Application de la synthèse sur l'exemple du tri de caisses*

Après suppression des états défendus, faiblement défendus et non accessibles, nous obtenons un automate booléen représentant le comportement maximal admissible du procédé complet composé de 1920 états et de 26015 transitions ; un extrait de cet automate est donné dans la figure 8.b. Si la modélisation des contraintes avait été réalisée par des automates comme ceux préconisés par la théorie de supervision, l'automate du superviseur serait alors composé de 16524 états et 153432 transitions. Le gain en terme d'états et de transitions apparaît ainsi clairement sur cet exemple simple.

5. Aide à l'élaboration de la commande

Les informations sur les blocages en exécution réelle et les corrections éventuelles qui pourraient porter sur des inhibitions d'ordre au niveau de la commande par la prise en compte des contraintes sont donc visibles au niveau de l'opération d'intersection entre *ASS* et *SUP*. Ainsi, à l'issue de cette opération, deux cas peuvent alors se produire :

– Aucun blocage, aucune correction ne sont visibles au niveau de l'intersection ce qui signifie que la commande respecte les spécifications de sécurité et de vivacité imposées au système. Le concepteur peut implanter la commande synthétisée dans un A.P.I. en sachant qu'il n'y aura aucun risque pour le matériel opératif, les opérateurs ou les produits.

– Des corrections ou des blocages sont détectés, la commande ne respecte donc pas les spécifications imposées au système :

- S'il s'agit d'une situation de blocage, le concepteur analyse alors la situation bloquante grâce aux données élaborées à partir des informations contenues dans les régions de l'automate d'intersection et agit en conséquence.

- S'il s'agit de corrections proposées sous forme d'inhibitions d'ordres ceci signifie que les contraintes imposées au système ne sont pas respectées. Le concepteur visualise alors une liste des contraintes non respectées au niveau de sa commande originelle et en déduit d'éventuelles modifications. Selon l'interprétation, il peut s'agir de contraintes trop restrictives ou d'un modèle de commande erroné fourni par le concepteur.

Ces travaux ont entraîné la conception et le développement d'un outil informatique dédié *OPTIGRAF7*. Développé sur PC, cet outil comprend un environnement d'édition et de calcul supportant l'ensemble des algorithmes évoqués dans cet article et intégrant une partie « visualisation des évolutions problématiques » que nous allons développer à la suite. Pour l'implantation, nous avons développé un traducteur de l'automate *COR* en langage normalisé (IEC, 2003). Il existe cependant un inconvénient majeur à cette implantation dans un A.P.I. qui est lié à son temps de cycle qui, dépendant directement du nombre d'instructions, peut nuire à la réactivité de la commande. Il est possible de s'en affranchir en développant une architecture générique à base de tables et de tâches sous interruption mais la non-normalisation de cette pratique nuit à son utilisation.

Les différents tests se sont effectués sur des A.P.I., TSX 37-21 (Tajer, 2005).

5.1. *Présentation synthétique de l'information proposée au concepteur*

5.1.1. *Automate Agrégé*

Pour aider le concepteur dans la compréhension des erreurs et dans les éventuelles modifications, les informations sont présentées à travers un automate agrégé. Cet automate regroupe les informations sur la situation de la commande, l'ensemble des étapes actives, les ordres autorisés et ceux interdits. Ses états correspondent aux régions de *SYNC*. Les évolutions intermédiaires représentant les états instables d'une région, sont remplacées par un état décrivant la situation stable de la commande. Cet état est ainsi muni de trois ensembles : l'ensemble des étapes de la situation correspondante du Grafcet, l'ensemble des ordres autorisés par le superviseur et l'ensemble des ordres interdits par le superviseur. Les ordres à autoriser et à interdire sont déduits des transitions commandables de la région de l'automate *SYNC*. L'automate agrégé (figure 9) ainsi généré est donné par des états qui sont reliés par des transitions non commandables reliant les régions de *SYNC*. L'automate agrégé obtenu à partir de *SYNC* est défini par 5-uplet (Σ_u, ET, ϕ, $etat_0$, $etat_b$), avec:

– Σ_u l'ensemble des événements non commandables

– ET l'ensemble des états correspondants, chacun à une région de *SYNC*. A chaque état et$\in$ET est associé le 5-uplet (ORD_{et}, $PROH_{et}$, Sit_{et}, E_{et}, Z_{et}) où :

- ORD_{et} : représente l'ensemble des ordres autorisés dans cet état,
- $PROH_{et}$: représente l'ensemble des ordres interdits dans cet état,
- Sit_{et} : représente l'ensemble des étapes Grafcet correspondant à cet état.
- E_{et} : représente le vecteur d'entrée de la commande correspondant à cet état.
- Z_{et} : représente le vecteur de sortie de la commande correspondant à cet état.

– $\phi : \Sigma_u \times ET \rightarrow ET$ est une fonction partielle représentant les transitions de l'automate agrégé.

– $etat_0$ est l'état initial.

– $etat_b$ est l'état de blocage.

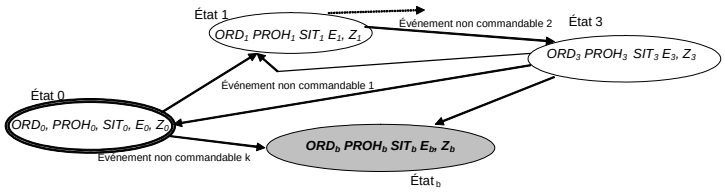


Figure 9. *Modèle de l'automate agrégé*

5.1.2. *Démarche d'analyse*

Pour structurer la démarche d'analyse, nous proposons au concepteur de procéder étape par étape en commençant par la spécification de la commande Grafcet jusqu'au modèle de bas niveau de la partie opérative. Les différentes étapes proposées sont les suivantes :

– La 1ère étape consiste à analyser la spécification Grafcet proposée en connaissant la situation complète du système grâce aux informations Sit_{et}, E_{et} et Z_{et} de l'automate agrégé. Cette analyse permet de comprendre les différentes évolutions possibles au niveau de la commande, c'est-à-dire, les différentes transitions franchissables.

– Si la 1ère étape ne permet pas au concepteur de détecter l'origine du blocage, il peut utiliser les paramètres ORD_{et} et $PROH_{et}$ de l'automate agrégé qui déterminent si les contraintes sont respectées ou non dans l'état concerné. Il peut être nécessaire dans cette étape d'afficher le superviseur *SUP*.

– La 3ème information permettant de renforcer l'analyse du concepteur concerne les sous-modules de partie opérative. Les éléments (capteurs et actionneurs) mis en cause dans un début d'analyse peuvent être visualisés dans le ou les sous-modules auxquels ils appartiennent.

5.2. *Etude de cas*

Dans cette partie, nous analysons trois types d'erreurs : oubli d'envoi d'un ordre à la partie opérative, erreur au niveau de la modélisation de la partie opérative et erreur liée à une contrainte trop restrictive. Pour chacune de ces erreurs, nous montrons l'intérêt de la démarche d'analyse proposée pour aider le concepteur à déterminer ces erreurs.

5.2.1. *Oubli d'un ordre envoyé à la partie opérative*

La commande proposée par le concepteur, est celle de la figure 10.

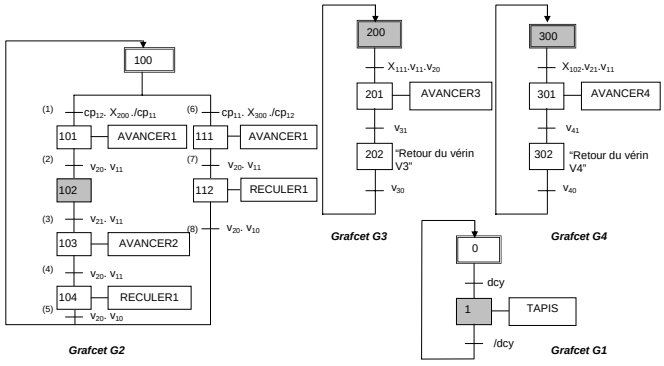


Figure 10. *Grafcet de commande de l'exemple de tri de caisses*

La figure 11 montre que la commande conduit à un blocage. La trace conduisant au blocage est proposée au concepteur pour lui permettre de visualiser et d'analyser l'erreur commise.

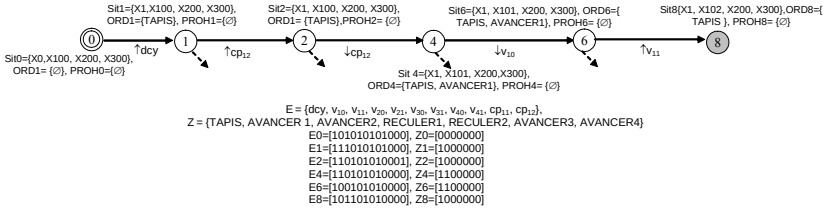


Figure 11. *Visualisation de la séquence de blocage*

La figure 11 illustre la situation de blocage car depuis l'état 8, aucune évolution n'est possible. Dans cet état, la commande (figure 10) se trouve dans la situation {X1, X102, X200, X300} du grafcet. Quant à la partie opérative, les deux vecteurs de sorties et d'entrées E_8= [1,0,1,1,0,1,0,1,0,0,0,0] et Z_8= [1,0,0,0,0,0,0] indiquent que le système est en marche (*dcy = 1*) le vérin V1 est en position sortie, et les vérins V2, V3, et V4 sont en position rentrée. Au niveau des sorties, seul l'ordre *TAPIS* est envoyé.

1ère étape : Analyse du grafcet de commande

Par l'analyse du grafcet, le concepteur peut en déduire que les évolutions possibles au niveau de la commande sont soit le passage à 0 de *dcy*, soit la validation et le franchissement de la transition notée (3) dans le grafcet G2 dont la réceptivité est $v_{21}.v_{11}$. Pour que la réceptivité $v_{21}.v_{11}$ soit vraie, le système se trouvant dans la configuration où le vérin V2 est rentré, il faut que l'ordre *AVANCER2* soit envoyé pour faire sortir le vérin V2 et donc, par conséquent, activer la position v_{21}. A ce stade, soit le concepteur détermine directement l'erreur (il manque l'ordre *AVANCER2* dans l'étape 102) soit, il a besoin d'information supplémentaire lui permettant de vérifier que cet ordre n'est pas interdit par les contraintes.

2ème étape : Analyse du superviseur

Si le concepteur le désire, les informations provenant du superviseur, peuvent lui être fournies. L'état du superviseur correspondant à l'état 8 de l'automate agrégé, est l'état q_{16} où les ordres autorisés sont : *AVANCER2*, …

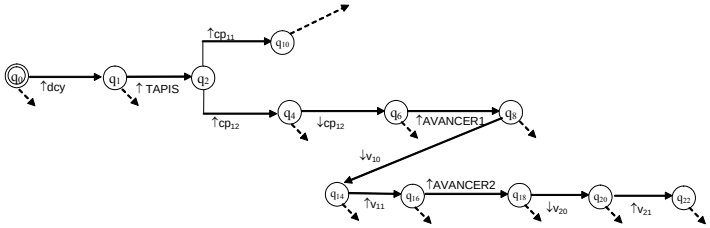


Figure 12. *Extrait de l'automate superviseur*

L'ordre *AVANCER2* n'est pas interdit au niveau de l'automate de la partie opérative contrainte (figure 12). Donc, soit le concepteur détermine que l'erreur se situe au niveau de la commande soit, il a besoin d'une information plus précise sur l'élément de la partie opérative associée à l'événement non commandable v_{21}.

3ème étape : Analyse de la partie opérative.

Pour être certain de son raisonnement, le concepteur peut demander à visualiser l'élément de partie opérative qui l'intéresse. A partir de cette dernière information (figure 13), le concepteur a confirmation que pour avoir l'occurrence de v_{21}, il faut l'envoi de *AVANCER2*. Le concepteur peut donc en conclure que pour que la réceptivité $v_{21}.v_{11}$ soit vraie, il faut que l'ordre *AVANCER2* soit envoyé vers la partie opérative, qu'aucune contrainte ne contredit à ce stade l'envoi de l'ordre et que seul un oubli de l'envoi de l'ordre au niveau de la commande génère le blocage.

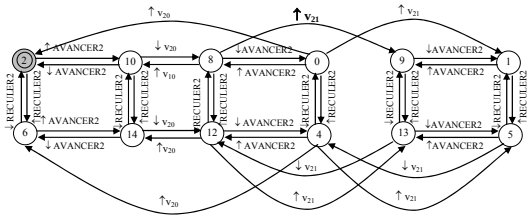


Figure 13. *Modèle du mouvement du Vérin V2*

5.2.2. *Erreur au niveau de la partie opérative.*

La commande proposée par le concepteur, est celle de la figure 14 qui conduit vers un blocage.

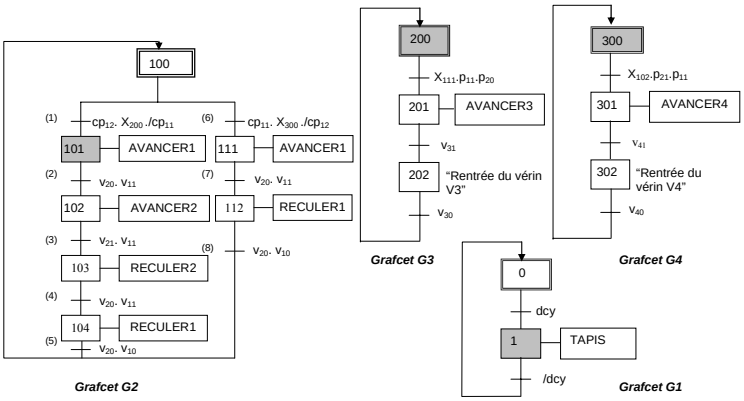


Figure 14. *Grafcet de commande du tri de caisses sans oubli d'ordre*

La trace conduisant au blocage est présentée au concepteur en figure 15. Depuis l'état 6, aucune évolution n'est possible. Dans cet état, les Grafcets (figure 14) se

trouvent dans la situation {X1, X101, X200, X300}. Les deux vecteurs de sorties et d'entrées E_6= [1,0,1,1,0,1,0,1,0,0,0,0] et Z_6= [1,1,0,0,0,0,0] indiquent que le système est en marche (*dcy* = 1), le vérin V1 est en position sortie, et les vérins V2, V3, et V4 sont en position rentrée. Au niveau des sorties, les ordres *TAPIS* et *AVANCER1* sont envoyés vers la partie opérative.

1ère étape : Analyse du grafcet de commande

Par l'analyse du grafcet et de la situation courante donnée par l'automate agrégé, le concepteur peut déduire que les évolutions possibles au niveau de la commande sont soit le passage à 0 de *dcy*, soit la validation et le franchissement de la transition (2) dont la réceptivité est $v_{20}.v_{11}$. Par rapport à l'état du système, pour que la réceptivité de la transition soit vraie, il faut que l'ordre *AVANCER1* soit envoyé, ce qui est le cas au niveau de l'étape 101 de la commande. On peut en déduire que l'erreur ne se trouve pas au niveau de la commande.

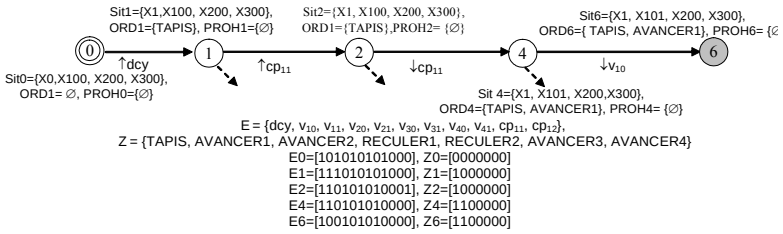


Figure 15. *Visualisation de la séquence de blocage*

2ème étape : Analyse du superviseur

Pour déterminer l'origine du blocage, le concepteur peut demander d'analyser le superviseur (figure 16). Il peut constater que l'ordre *AVANCER1* est bien autorisé par le superviseur. A ce niveau, aucune contrainte n'est donc violée et même si la suite d'événements $\downarrow v_{10}$ puis $\uparrow v_{21}$ suite à l'occurrence de *AVANCER1* ne paraît pas logique, il faut analyser les éléments de partie opérative pour pouvoir conclure.

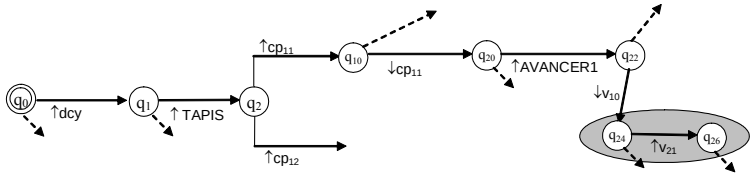


Figure 16. *Extrait de l'automate superviseur*

3ème étape : Analyse de la partie opérative.

Le concepteur peut demander à visualiser les éléments de partie opérative du vérin V1 (figure 17) et du vérin V2 (figure 13).

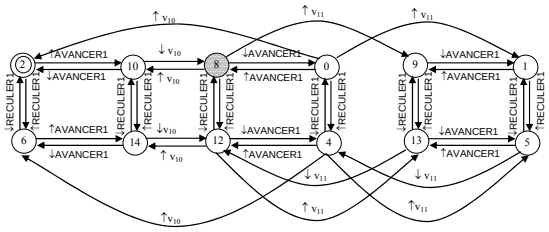


Figure 17. *Modèle du mouvement du Vérin V1*

L'analyse des figures 17 et 13, montre que v_{21} appartient au modèle du mouvement V1 et au modèle du mouvement V2, ce qui est logiquement impossible dans la configuration du système de tri de caisses. La figure 17 décrit la suite d'événements lorsque $\uparrow AVANCER1$ est autorisé : apparition de $\downarrow v_{10}$ puis apparition de $\uparrow v_{21}$. Avec ces informations, le concepteur peut se rendre compte qu'il y a une erreur de modélisation de la partie opérative du vérin V1. Logiquement, l'événement attendu après l'état 8 de la figure 17, est $\uparrow v_{11}$ et non $\uparrow v_{21}$. En prenant en compte cette erreur de modélisation, la figure va évoluer pour retrouver la suite logique $\downarrow v_{10}$ puis $\uparrow v_{11}$ suite à l'occurrence de *AVANCER1*.

5.2.3. *Intégration d'une contrainte restrictive par rapport au cahier des charges*

La commande proposée par le concepteur, est celle de la figure 18 qui conduit vers un blocage.

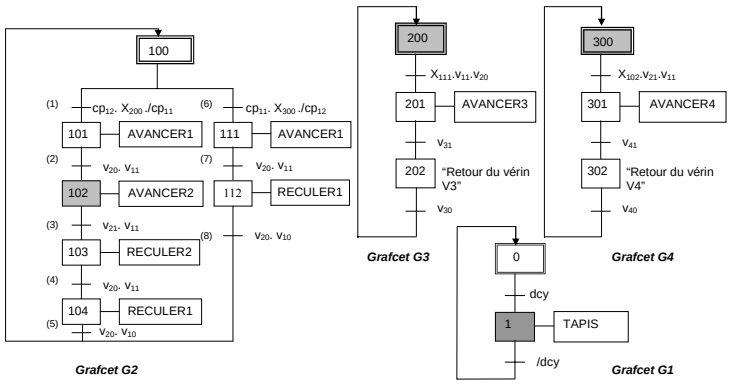


Figure 18. *Grafcet de commande du tri de caisses*

La figure 19 illustre la situation de blocage. En effet, depuis l'état 6, aucune évolution n'est possible. Dans cet état, la commande (figure 18) se trouve dans la situation {X1, X102, X200, X300}. Les deux vecteurs de sorties et d'entrées E_6= [1,0,1,1,0,1,0,1,0,0,0,0] et Z_6= [1,0,1,0,0,0,0] indiquent que le système est en marche (*dcy* = 1), le vérin V1 est en position sortie, et les vérins V2, V3, et V4 sont en position rentrée. Au niveau des sorties, l'ordre *TAPIS* est autorisé, l'ordre

AVANCER2 est quant à lui interdit.

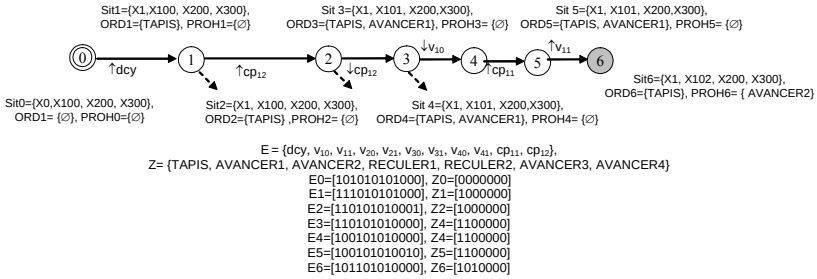


Figure 19. *Visualisation de la séquence de blocage*

1ère étape : Analyse du grafcet de commande

Par analyse de la commande, le concepteur peut en déduire que les évolutions possibles au niveau de la commande sont soit le passage à 0 de *dcy*, soit la validation et le franchissement de la transition $v_{21}.v_{11}$. Pour que le système puisse évoluer, il faut que la réceptivité de la transition $v_{21}.v_{11}$ soit vraie. Le vérin V1 étant déjà sorti, il faut que l'ordre *AVANCER2* soit envoyé pour faire sortir le vérin V2 et donc activer la position v_{21}. La commande répond à ce propos, l'erreur n'est donc pas dans la commande.

2ème étape : Analyse du superviseur

Pour déterminer l'origine du blocage, le concepteur demande à analyser le superviseur (figure 20). Il peut constater que l'ordre *AVANCER1* est bien autorisé par le superviseur. A ce niveau, la suite d'événements qui découle de l'occurrence de *AVANCER1* est $\uparrow cp_{11}$, $\downarrow v_{10}$ puis $\uparrow v_{11}$. Visiblement, l'ordre *AVANCER2* n'apparaît pas, le concepteur peut supposer qu'il est interdit.

Le concepteur peut alors demander à visualiser les contraintes associées à cet événement. Dans ce cas, deux contraintes ont été définies (voir paragraphe 4.2) :

– AVANCER2 ∧ RECULER2=0 (2)
– AVANCER2 ∧ cp11=0 (5)

C'est la contrainte (5) qui n'est pas respectée car pendant la sortie du vérin V1, avant qu'il n'ait atteint sa position de sortie v_{11}, une petite caisse est arrivée. C'est un cas très particulier mais qui ne peut être négligé dans une approche sécuritaire. Le concepteur a alors deux possibilités pour résoudre le blocage. Soit, il choisit de relâcher la contrainte (5) car il la juge trop restrictive et pour lui ce cas de figure ne se produira jamais, soit, il modifie sa commande pour empêcher l'arrivée d'autres caisses pendant le traitement du positionnement d'une caisse. L'utilisation des éléments de partie opérative n'apporte pas dans ce cas, plus d'information au concepteur.

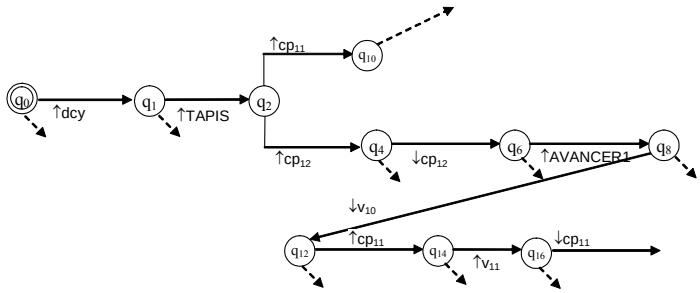


Figure 20. *Extrait de l'automate superviseur*

6. Conclusion

La mise en oeuvre des concepts à base de théorie de supervision sur des exemples diversifiés, nous a contraints à essayer d'apporter une réponse au manque d'expressivité et de convivialité des modèles à états préconisés dans la théorie, aux difficultés de modélisation et aux risques d'explosion combinatoire inhérents à leur utilisation. Nous avons proposé ici une méthodologie de modélisation de la PO et des contraintes et nos récents travaux (Rohee et al., 2006), (Marangé et al., 2007) visent à améliorer la généricité et la pertinence de la modélisation dans un objectif de validation de commande mais aussi de diagnostic des SED.

Globalement la démarche que nous proposons permet de détecter des erreurs qui peuvent conduire à des blocages en exécution réelle comme c'est le cas dans le dernier exemple du §5.2.3, des erreurs au niveau de la commande, des erreurs au niveau de la modélisation de la partie opérative et même de révéler si des contraintes sont trop restrictives. Avec la prise en compte des nouveaux modèles et algorithmes, la démarche peut être utilisée aussi bien pour effectuer une synthèse automatique de la commande que pour effectuer une validation de la commande avant implantation. Un aspect très intéressant de ce travail est son utilisation dans le cadre de l'enseignement des automatismes industriels où les étudiants sont confrontés à l'utilisation de parties opératives réelles leur permettant de tester des lois de commande tout en utilisant du matériel industriel. L'enseignant est le concepteur des modèles de partie opérative et des contraintes et l'étudiant le concepteur de la commande à valider. Ainsi, si des erreurs de commande sont commises, elles ne pourront pas entraîner des risques pour le matériel (Marangé et al., 2006). En fait, la démarche va beaucoup plus loin qu'une simple assistance à l'élaboration de la commande mais permet d'effectuer une validation du comportement global d'un SED.

La possibilité d'utiliser cette approche en ligne a été étudiée dans les travaux de (Philippot et al., 2004b). Mais l'approche se heurte à la taille de fenêtre d'observation du système. Lorsqu'elle est trop grande, la construction de la commande s'éloigne du temps réel et si elle est trop petite, la détection du blocage n'est pas systématique. La complexité de la détermination de la taille de cette

fenêtre nous a par conséquent, conduit à proposer une autre vision de la validation en ligne. Dans cette nouvelle approche, nous proposons de travailler avec une notion filtre entre la commande et la partie opérative. Le filtre contient les différentes contraintes et permet de générer automatiquement des explications au non-respect des contraintes. Nous travaillons à l'automatisation de l'analyse et des explications des erreurs ainsi qu'à une proposition de méthodologie pour le concepteur du modèle lui permettant de déterminer l'ensemble des contraintes sans risque d'en oublier. C'est également avec cette approche que nous travaillons pour prendre en compte le caractère distribué des commandes des systèmes, ce qui n'avait pas été envisagé dans les travaux proposés dans cet article.

7. Bibliographie

Balemi S., Hoffmann G.J., Gyugyi P., Wong-Toi H., Franklin G.F., "Supervisory control of a rapid thermal multiprocessor", *IEEE Transactions on Automatic Control*, vol. 38, n°7, 1993, p1040-1059.

Bérard, B., et al., "*Systems and software verification: Model-checking techniques and tools*", Heidelberg, Springer-Verlag Edition, 1999.

Berry G. "Real time programming: special purpose or general purpose languages", rapport de recherche INRIA, n°1065, Août 1989.

Carré-Ménétrier V., Zaytoon J., "Grafcet: behavioural issues and control synthesis", *European Journal of Control (EJC)*, volume 8 n°4, 2002, p375-401.

Chandra V. and Kumar R., "A Event Occurrence Rules based Compact Modeling Formalism for a Class of Discrete Event Systems". *Mathematical and Computer Modeling of Dynamical Systems*, 2002, volume 8, n° 1, pp 49-73.

Chandra V., Kumar R., "A Discrete Event Systems Modeling Formalism Based one event Occurrences Rules and Precedences", *IEEE Transactions one Robotics and Automation*, Volume 17, n°6, 2001.

Delaval G., Rutten E., "A Domain-Specific Language for Multitask Systems, Applying Discrete Controller Synthesis," *EURASIP Journal on Embedded Systems*, vol. 2007, Article ID 84192, 17 pages, 2007.

Faure J-M., Lesage J-J. "Methods for safe control systems design and implementations", *10th IFAC Symposium on Information Control Problems in Manufacturing, INCOM'2001*, September 2001, Vienna (Austria), CDRom paper, 6 pages

Gaffé D. "Modélisation et vérification d'un système mécatronique par SyncCharts", *MSR'03 Modélisation des systèmes réactifs*, pp 95-107, Metz, octobre 2003

Ghaffari A., Rezg N., Xie X., "Algebraic and geometric characterization of Petri net controllers using the theory of region", *Proceeding of 6th International Workshop on discrete Event Systems, WODES'02*, Saragoza, Espagne, 2-4 octobre 2002

Gouyon, D., et al., "Pragmatic approach for modular control synthesis and implementation", *International Journal of Production Research*, 42(14), 2004, p2839–2858.

Hellgren A., Lennartson B., Fabian M., "Modelling and PLC-based implementation of modular supervisory control", *Proceeding of 6th International Workshop on discrete Event Systems, WODES'02*, Saragoza, Espagne, 2-4 octobre 2002

IEC "Automates programmables – Partie 3 : Langages de programmation - Programmable controllers – Part 3:Programming languages", Norme Internationale / International Standard, CEI/IEC 61131-3, 2003.

IEC "Langage de spécification GRAFCET pour diagrammes fonctionnels en séquence – GRAFCET specification language for sequential function charts", Norme Internationale / International Standard, CEI/IEC 60848, février 2002, 94 pages.

IEC "Etablissement des diagrammes fonctionnels pour systèmes de commande - Preparation of function charts for control systems", Norme Internationale / International Standard, CEI/IEC 60848, 1988, 99 pages

Kumar R., "Supervisory Synthesis Techniques for Discrete Event Dynamical Systems", Thesis for PhD. Degree, University of Texas, 1991

Li Y., "Control of vector discrete-events systems", PhD thesis, Department of electrical and computer engineering, University of Toronto, 1991

Machado, "Influence de la prise en compte d'un modèle de processus en vérification formelle des systèmes à événements discrets", Thèse de doctorat de l'école normale supérieure de Cachan et de l'université de Minho (Portugal), juin 2006

Marangé P., Gellot F., Riera B., "Safety validation of automation systems: Application for teaching of Discrete Event System control", IFAC ICINCO'2007, Angers, Mai 2007

Marangé P., Gellot F., Chemla J.P., Riera B., "Requierement and Use for remote teaching of Discrete Events Systems", *Proceeding of 7th IFAC symposium on Advances in control Education, ACE'06*, Paper WeP02.1sur le CD-ROM, Madrid, Spain, June 21-23, 2006

Philippot A., Tajer A., Gellot F., Carré-Ménétrier V., "Méthodologie de modélisation dans le cadre de la synthèse formelle des SED", Douz, Tunisie, papier n°234, 6 pages, 2004

Philippot A., Tajer A., Gellot F., Carré-Ménétrier V., "On-line synthesis approach based on a structured plant modelling", *In 7th International Workshop On Discrete Event Systems WODES'04*, Reims, France, pp.397-402, September 2004.

Philippot A., Tajer A., Gellot F., Carré-Ménétrier V., "Synthèse de la commande spécifiée en Grafcet: application à un préhenseur pneumatique*"*, *Colloque Francophone sur la modélisation des systèmes réactifs, MSR'03,* Metz, 61-75, octobre 2003.

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

Rohee B., Riera B., Carré-Ménétrier V., Roussel J-M., "A methodology to design and check a plant model", 3rd IFAC Workshop on Discrete-Event System Design DESDes 2006, Rydzyna Castle (Poland), 25-28 September 06

Roussel J.M., A Medina., J.M. Faure., "Synthèse d'un programme de commande d'un système logique à partir de l'expression algébrique de ses spécifications*", Colloque*

Francophone sur la modélisation des systèmes réactifs, MSR'03, Metz, 77-93, 2003.

Roussel, J.M., & Denis, B., "*Safety properties verification of ladder diagram programs",* Journal Européen des Systèmes Automatisés, Hermès Editions, 36(7), 905–917, 2002.

Roussel, J.M., & Faure, J.M. "An algebraic approach for PLC programs verification". *In Proceedings of 6th international workshop on discrete event systems (WODES'02)*, Zaragoza, Spain, 2–4 October 2002, pp. 303–308.

Roussel J.M., "Analyse de Grafcet par génération logique de l'automate équivalent", thèse de l'Ecole Normale Supérieure de Cachan, France, 1994.

Tajer A., "Contribution aux approches formelles de synthèse de commande spécifiée par Grafcet", Thèse de doctorat de l'université de Reims Champagne Ardenne, Spécialité génie informatique et traitement du signal, novembre 2005

Wang Y, "Supervisory control Boolean discrete-events systems", Master's thesis of University of Toronto, june 2000

Wong R.K., "State-based discrete-event modelling and control of concurrent systems", Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, 1990