Simulateur 3D interactif de Parties Opératives et Synthèse sûre de la commande des systèmes manufacturiers

B. RIERA*, B. VIGARIO** , A. PHILIPPOT, D*. ANNEBICQUE*, F. GELLOT*

*CReSTIC, UFR 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

** REAL GAMES LDA, Rua Elísio de Melo nº 39, Piso 3
4000-196 Porto – Portugal (bruno.vigario@realgames.pt)
bruno.vigario@realgames.pt

***Résumé* –Dans le cadre de la collaboration scientifique et technique démarrée en 2008 entre le CReSTIC (Centre de Recherche en STIC) de l'Université de Reims Champagne-Ardenne et la société Real Games au Portugal, la collection ITS PLC a été enrichie au moyen de nouvelles versions. Nous présentons dans cet article une version appelée ITS PLC « bêta » qui permet au moyen de scripts écrits en IronPython de réaliser ses propres contrôleurs mais également d'étendre les possibilités de communication d'ITS PLC en utilisant la puissance de développement de « .NET ». ITS PLC Bêta est utilisé au CReSTIC pour des travaux de recherche dans le domaine de la synthèse sûre de la commande des systèmes manufacturiers. Un algorithme de synthèse original, basé sur la définition de contraintes de sécurité exprimées sous la forme d'un monôme logique et vérifiées formellement hors ligne au moyen d'un model-checker, a été proposé. La séparation entre les aspects sécuritaires et fonctionnels permet d'aboutir à une commande plus simple que celle obtenue à partir d'une spécification complète en GRAFCET. L'intérêt et le bien-fondé de l'approche sont montrés dans l'article sur le système « palettiseur ». Ces contrôleurs « robustes »sont également utilisés pour vérifier la qualité des modèles de Parties Opératives.**

Mots clés – SED, Commande, API, Parties Opératives, rendu 3D, jeux vidéo

1. Introduction

Dans le cadre d'une collaboration scientifique et technique démarrée en 2008 entre le CReSTIC (Centre de Recherche en STIC) de l'Université de Reims Champagne-Ardenne et la société Real Games au Portugal, des simulateurs de systèmes manufacturiers, reposant sur l'utilisation des technologies des jeux vidéo ont été réalisés. La collection ITS PLC [1] [2] est un ensemble de logiciels qui permet de former les apprenants à la programmation des API au moyen de Parties Opératives (PO) simulées. Les environnements virtuels proposés n'ont jamais été aussi réalistes grâce d'une part à une totale interactivité, et d'autre part à la qualité des animations graphiques 3D en temps réel, des dynamiques et des sons. La collection ITS PLC a été enrichie ces dernières années au moyen de nouvelles versions (ITS PLC ATG Edition, ITS PLC SIMU3D, …). Nous présentons dans cet article une version appelée ITS PLC « bêta » qui permet au moyen de scripts écrits en IronPython de réaliser ses propres contrôleurs mais également d'étendre les possibilités de communication d'ITS PLC en utilisant la puissance de développement de « .NET ». En effet, IronPython est une implémentation open source du langage de programmation très populaire Python, et qui est étroitement intégré avec le framework « .NET ». La première partie de l'article est consacrée à la présentation du logiciel ITS PLC « bêta ».

Nous utilisons au CReSTIC cette version Bêta pour des travaux de recherche dans le domaine de la synthèse sûre de la commande des systèmes manufacturiers. Un algorithme de synthèse original, basé sur la définition de contraintes de sécurité exprimées sous la forme de monômes logiques et vérifiées formellement hors ligne au moyen d'un model-checker, a été proposé. La séparation entre les aspects sécuritaires et fonctionnels permet dans certains cas d'aboutir à une commande plus simple que celle obtenue à partir d'une spécification complète en GRAFCET. L'intérêt et le bien-fondé de l'approche ont été testés sur plusieurs systèmes simulés de la collection ITS PLC dont le « palettiseur » qui est présenté dans cet article. Ces contrôleurs sont également utilisés pour vérifier la qualité des modèles de Parties Opératives. En effet, ces derniers reposent sur des moteurs physiques dont les solveurs peuvent parfois présenter quelques faiblesses. La deuxième partie de l'article est consacrée à la présentation de l'algorithme de synthèse de la commande et l'exemple du « palettiseur ».

2. Le logiciel « ITS PLC bêta »

La collection ITS PLC (www.realgames.pt) propose 5 simulations 3D de Parties Opératives (PO) de systèmes industriels (tri de caisses, mélangeur, palettiseur, pick & place, et magasin automatisé) permettant de se connecter soit à un API réel au moyen d'un boîtier USB d'E/S TOR (ITS PLC Professional Edition), ou encore à un client TCP (ITS PLC ATG Edition). Lors des précédentes journées « démonstrateurs » à Angers en 2010 [3], nous avions présenté le logiciel ITS PLC Professional Edition. Toutes ces versions se caractérisent par le fait que la communication avec la Partie Commande (PC) est figée (API, client TCP, …) et non modifiable. De plus, il n'est pas possible de développer ses propres contrôleurs directement dans ITS PLC. En d'autres termes, un logiciel supplémentaire pour la réalisation de la PC est nécessaire pour pouvoir utiliser la collection ITS PLC. La version ITS PLC « bêta » vise à proposer une solution simple et performante à ces limitations. ITS PLC « bêta » permet de réaliser ses propres scripts en IronPython permettant ainsi de concevoir ses propres contrôleurs ou encore de se connecter à différents types d'appareils.

2.1 Principe de fonctionnement

L'idée de base de ITS PLC « beta » est de pouvoir écrire des scripts en IronPython (http://ironpython.net) qui peuvent être utilisés comme des interfaces entre les simulations de PO d'ITS PLC et des technologies ou « devices » externes qui peuvent être matériels ou logiciels. La version ITS PLC « bêta » est fournie de base avec des scripts permettant de se connecter au boîtier d'E/S Advantech USB-4750 DAQ, ou encore à un serveur OPC. De plus, une implémentation en IronPython des blocs fonctions les plus classiques définis dans la norme IEC 61131-3 est également proposée (standard_lib). Cela permet d'écrire ses propres contrôleurs dans un langage proche du ST (Structured Text).

L'utilisation de ITS PLC « bêta » est simple. Une nouvelle option (cf. figure 1) est proposée dans le "menu des options" appelée « Script ». Celle-ci permet de choisir le script que l'on désire utiliser. Si l'on choisit « None », aucun script ne sera lancé, mais il est possible de « jouer» de façon interactive avec l'interpréteur IronPython.

Figure 1 Menu des options

En effet, ITS PLC « bêta » inclut un interpréteur IronPython interactif qui permet d'interagir avec chaque système simulé en développant et testant ses propres scripts. Pour cela, il suffit d'appuyer sur la touche F1 pour afficher ou masquer l'interpréteur (cf. figure 2).


Figure 2 Interpréteur IronPython

On accède très facilement aux entrées/sorties des systèmes simulés au moyen de l'objet IO. IO.Actuators and IO.Sensors renvoient respectivement l'état des actionneurs et l'état des capteurs. Un script peut consommer 3 événements générés par l'objet « Scene » : OnUpdate, OnLoad et OnUnload. L'événement OnUpdate est déclenché toutes les 16 ms, ce qui correspond au temps de cycle de ITS PLC. En vue d'illustrer la manière d'utiliser ces 3 événements, un exemple simple de contrôleur est maintenant proposé. Celui-ci utilise les blocs fonctions R_TRIG (rising edge trigger) et F_TRIG (falling edge trigger) du module « standard_lib ». Le code de la figure 3 permet d'activer la sortie 0 au moyen d'un appui sur le bouton « Start » (entrée 12, normalement ouvert) et de l'arrêter lorsque l'on appuie sur le bouton « Stop » (entrée 13, normalement fermée). Ce script consomme l'événement « Scene.OnUnload ». Quand on quitte la scène courante ou bien quand on recharge un script à partir de la console, l'événement « unload » est déclenché et la fonction unload est donc automatiquement appelée. La fonction unload se décharge elle-même de l'événement « Scene.OnUnload » et décharge également la fonction mycontroller de l'événement « Scene.OnUpdate ». La mise à 1 de IO.connected permet d'indiquer à l'utilisateur, au moyen d'une roue qui tourne sur le panneau de commande de ITS PLC « bêta », que le script a bien été chargé. En cas d'erreur dans le script, le moteur d'exécution s'arrête et informe l'utilisateur sur le type d'erreur.

2.2 Exemples de scripts

Nous avons développé entre autres un script permettant une communication avec un serveur MODBUS TCP. Ce script permet ainsi de se connecter au simulateur d'API de Unity Pro, l'atelier de développement des applications pour les automates de la marque SCHNEIDER ELECTRIC (M340, TSX Premium). Il devient ainsi possible de tester la commande sans avoir d'API réel en utilisant le simulateur d'automates fourni avec Unity Pro. La société Real Games

a développé un script permettant à ITS PLC de s'interfacer avec une Kinect. Il devient alors possible de se déplacer dans la scène 3D uniquement au moyen de mouvements des mains. Toutes ces applications seront présentées lors des journées des 12 et 13 juin 2013 à Angers. Nous utilisons au CReSTIC (Centre de Recherche en STIC, EA 3804) de l'Université de Reims Champagne-Ardenne cette version Bêta pour des travaux de recherche dans le domaine de la synthèse sûre de la commande des systèmes manufacturiers. Un algorithme de synthèse original, basé sur la définition de contraintes de sécurité exprimées sous la forme d'un monôme logique et vérifiées formellement hors ligne au moyen d'un model-checker, a été proposé. La séparation entre les aspects sécuritaires et fonctionnels permet dans certains cas d'aboutir à des commandes simples et performantes. La deuxième partie de l'article présente ces travaux de recherche.

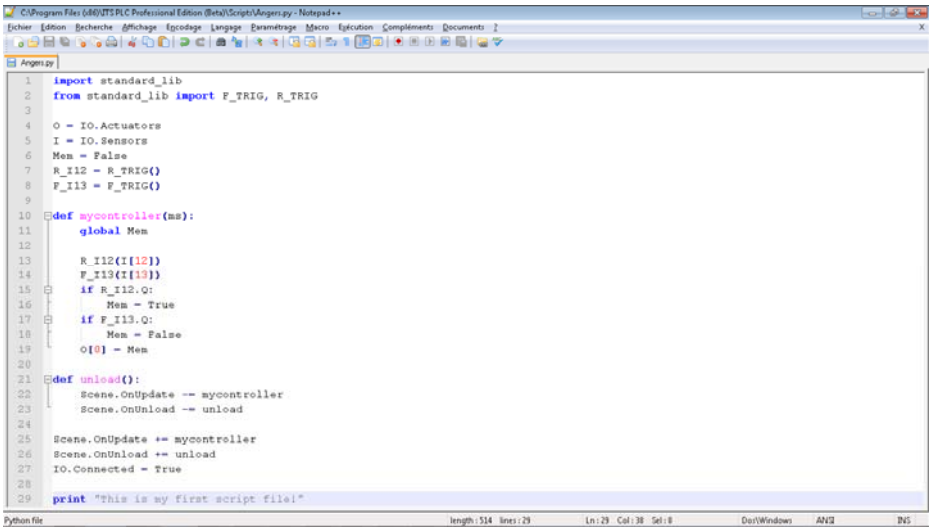


Figure 3 Exemple de script

3. Synthèse de la commande à bases de contraintes logiques

Assurer de manière formelle la sécurité des Systèmes Automatisés de Production (SAP) est un défi scientifique porteur d'enjeux industriels importants. Les travaux présentés dans cet article relèvent de cette problématique. Ils portent uniquement sur les systèmes à événements discrets pilotés par des API et dont les Entrées/Sorties (E/S) ne sont que des variables logiques. Une méthodologie pour la conception d'une commande sûre de fonctionnement d'un SAP de type manufacturier a été 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 [4] [5] [6] réalisés au CReSTIC de l'Université de Reims. Elles permettent d'assurer la sécurité du SAP 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 [7], qui permet de valider la suffisance des contraintes et de vérifier la robustesse du filtre. 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 [9] 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 [10]. Elle est basée sur une modélisation modulaire 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 [6].

La méthode a été testée avec succès sur plusieurs systèmes manufacturiers réels et simulés, et a montré son intérêt dans des applications allant de la télémaintenance à la formation des automaticiens [2] [8]. La méthodologie initiale concernait uniquement la sécurité des équipements. 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, …) [11]. Par conséquent, même si le fonctionnel est erroné, le système reste sûr de fonctionnement.
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). Une variable logique peut prendre les valeurs 0 (faux) ou 1 (vraie).
- Σ et Π sont respectivement la somme logique (OU) et le produit logique (ET) de variables logiques.
- ↑*x* est le front montant de la variable logique *x* (dans l'API, $\uparrow x = \overline{x^*}.x$).
- ↓*x* est le front descendant de la variable logique *x* (dans l'API, $\downarrow x = x^*.\bar{x}$).

Dans cette approche, on suppose que les contraintes de sécurité peuvent toujours être représentées sous la forme d'un monôme logique qui doit être faux (=0) tout le temps, c'est-à-dire : à chaque cycle API. Si une contrainte logique est vraie, cela signifie qu'elle est violée. Les contraintes de sécurité 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 (idéalement 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)). Il est supposé que :

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

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

$$CSs_i(Sk) = \pi_i(Sk, E, O, S^*) \qquad (1)$$

$$CSc_i(Sk, Sl) = \pi_i(Sk, Sl, E, O, S^*) \qquad (2)$$

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 seulement de 2 façons différentes (équations (3) et (4)) où $f_i(E, O, S^*)$ est un monôme logique.

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

ou

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

Pour chaque sortie S_k, il est possible d'écrire la somme des contraintes de sécurité simples (équation (5)) où f_{0k} et f_{1k} sont des polynômes (fonctions ∑Π). L'équation (5) doit toujours être fausse car toutes les contraintes simples de sécurité doivent être fausses à chaque cycle de l'API.

$$\sum_j CSs_j(Sk) = Sk.f_{0k}(E, O, S^*) + \overline{Sk}.f_{1k}(E, O, S^*) = 0 \qquad (5)$$

On notera que l'équation 6 doit être nécessairement respectée. En effet, dans le cas contraire, cela signifie que 2 *CSs* sont en contradiction et qu'au moins une des contraintes n'est pas respectée. Par conséquent, l'ensemble de contraintes n'est pas cohérent. Cela ne peut pas se produire si l'ensemble des contraintes a été validé au moyen de l'approche par model checking évoquée précédemment.

$$f_{0k}(E, O, S^*).f_{1k}(E, O, S^*) = 0 \qquad (6)$$

On notera que si f_{0k}=0 ou si f_{1k}=0, la propriété est nécessairement vérifiée. En complément, si pour une sortie *Sk*, toutes les $CSs_j(Sk)$ sont seulement écrites au moyen de front montant ou de front descendant de *Sk*, alors la propriété est toujours vérifiée.

Les contraintes de sécurité combinées *CSc* s'expriment également sous la forme d'un monôme logique et font intervenir 2 sorties à l'instant *t*. Par conséquent, elles peuvent s'écrire de 4 façons différentes (équations (7), (8), (9) et (10)).

$$CSc_i(Sk, Sl,) = Sk.Sl.f_i(E, O, S^*) \qquad (7)$$
$$CSc_i(Sk, Sl) = \overline{Sk}.Sl.f_i(E, O, S^*) \qquad (8)$$
$$CSc_i(Sk, Sl) = \overline{Sl}.Sk.f_i(E, O, S^*) \qquad (9)$$
$$CSc_i(Sk, Sl) = \overline{Sk}.\overline{Sl}.f_i(E, O, S^*) \qquad (10)$$

L'équation (7) signifie que si $f_i(E, O, S^*)$ est vraie, 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 (8) signifie que si $f_i(E, O, S^*)$ est vraie, on ne peut pas avoir simultanément *Sl=1* et *Sk=0*. L'équation (9) est similaire à l'équation (8). Si $f_i(E, O, S^*)$ est vraie, on ne peut pas avoir simultanément *Sl=0* et *Sk=1*. Dans les 2 cas, il faut donner la priorité à *Sk* ou à *Sl*. Enfin, l'équation (10) signifie que si $f_i(E, O, S^*)$ est vraie, les 2 sorties *Sk* et *Sl* ne peuvent pas être toutes les 2 égales à 0.

A partir des *CSc*, comme pour les *CSs*, il est possible d'écrire l'équation (11) pour chaque paire de sorties S_k et S_l.

$$\sum_i \mathrm{CSc}_i(\mathrm{Sk}, \mathrm{Sl}) = \mathrm{Sk}.\mathrm{Sl}.\mathrm{f}_{00\mathrm{kl}}(\mathrm{E}, \mathrm{O}, \mathrm{S}^*) + \overline{\mathrm{Sk}}.\mathrm{Sl}.\mathrm{f}_{10\mathrm{kl}}(\mathrm{E}, \mathrm{O}, \mathrm{S}^*) + \mathrm{Sk}.\overline{\mathrm{Sl}}.\mathrm{f}_{01\mathrm{kl}}(\mathrm{E}, \mathrm{O}, \mathrm{S}^*) + \overline{\mathrm{Sk}}.\overline{\mathrm{Sl}}.\mathrm{f}_{11\mathrm{kl}}(\mathrm{E}, \mathrm{O}, \mathrm{S}^*) = 0 \qquad (11)$$

Dans le cadre de cet article, nous ne développerons pas les conditions nécessaires et suffisantes pour garantir la cohérence des *CSc*. Pour information, les 4 équations (12,13, 14 et 15) doivent être respectées pour ne pas avoir de contradictions entre les *CSs* et *CSc*.

$$f_{1k}(E,O,S^*).f_{1l}(E,O,S^*).f_{00kl}(E,O,S^*)=0 \quad (12)$$
$$f_{0k}(E,O,S^*).f_{1l}(E,O,S^*).f_{10kl}(E,O,S^*)=0 \quad (13)$$
$$f_{1k}(E,O,S^*).f_{0l}(E,O,S^*).f_{01kl}(E,O,S^*)=0 \quad (14)$$
$$f_{0k}(E,O,S^*).f_{0l}(E,O,S^*).f_{11kl}(E,O,S^*)=0 \quad (15)$$

L'algorithme de synthèse de la commande proposé est donné par la figure 4. Il 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 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.

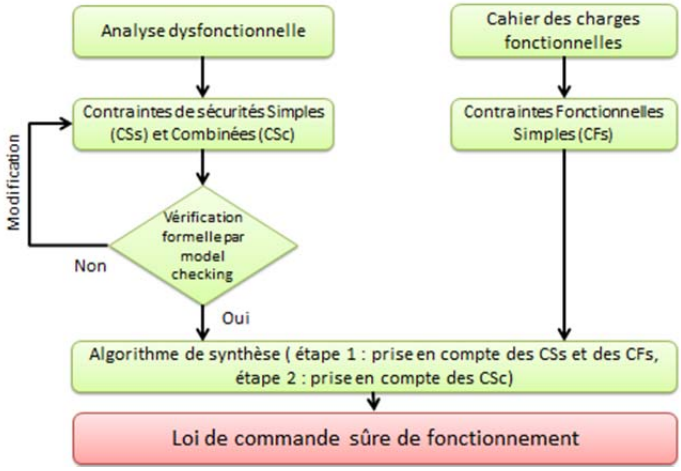


Figure 4 Algorithme de synthèse de la commande

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. 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 (16)).

$$g_k(E,O,Sk)=Sk.g_{0k}\ ou$$
$$g_k(E,O,Sk)=\overline{Sk}.g_{1k} \quad (16)$$

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). Cette première étape de l'algorithme permet de générer pour chaque sortie une commande intermédiaire (*S'k*).

For k=1 **to** Nb_of_S # Nb_of_S correspond au nombre d'actionneurs

$$S'k=\overline{f_{ok}}.g_{1k}+f_{1k}$$

End for

Etape 2 : Prendre en compte les contraintes de sécurité combinées *CSc* pour calculer la commande finale de chaque sortie *Sk*. En effet, les commandes *S'k* ne tiennent pas compte des contraintes de sécurité combinées. Dans un premier temps nous avons considéré que pour

chaque paire d'actionneurs, les *CSc* ne font intervenir qu'un seul type de contraintes. En d'autres termes, l'équation (11) s'écrit :

$$\sum_i CSc_i(Sk, Sl) = Sk.Sl.f_{00kl}(E, O, S^*) = 0$$
$$ou \sum_i CSc_i(Sk, Sl) = \overline{Sk}.Sl.f_{10kl}(E, O, S^*) = 0$$
$$ou \sum_i CSc_i(Sk, Sl) = Sk.\overline{Sl}.f_{01kl}(E, O, S^*) = 0$$
$$ou \sum_i CSc_i(Sk, Sl) = \overline{Sk}.\overline{Sl}.f_{11kl}(E, O, S^*) = 0 \qquad (17)$$

Cette hypothèse n'est pas restrictive car elle est respectée dans la plupart des cas pratiques que nous avons testés. Elle permet de simplifier le calcul de *Sk* et de *Sl* en garantissant que la solution retenue respectera l'équation 11. L'algorithme de calcul des *Sk* à partir des *CSc* est donné ci-après.

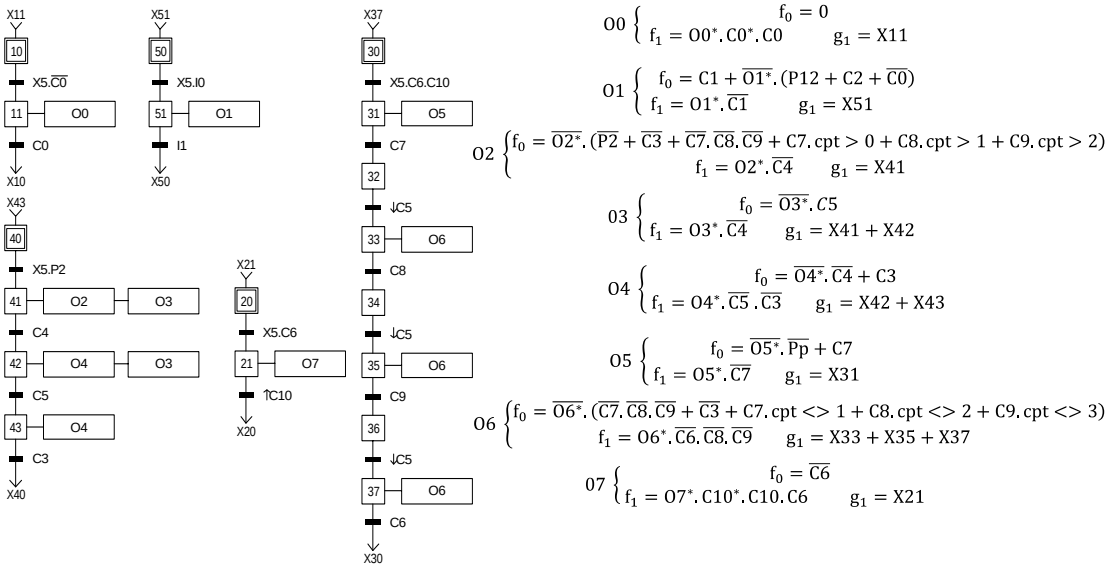
Repeat
  For i=1 **to** Nb_of_CSc # Nb_of_CSc correspond au nombre de CSc
    If CSc_i(*S'k*,*S'l*)=1 **then**
      If *S'k* has priority over *S'l* **then**
        $S'l = \overline{S'l}$
      Else
        $S'k = \overline{S'k}$
      End if
    End if
  End for
  Flag = false
  For i=1 **to** Nb_of_CSc
    Flag= Flag + CSc_i(*S'k*, *S'l*)
  End for
Until not Flag
For i =1 **to** NB_of_S
  Si=*S'i*
End for

Le principe 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 si l'ensemble des contraintes (*CSs* et *CSc*) a été vérifié formellement. En cas d'erreur, le calcul ne sortira pas de la boucle "Repeat Until » et le chien de garde de l'API se déclenchera. Il est également 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*. Il est très facile d'implémenter l'algorithme ou la solution algébrique dans un API. 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. 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 l'exemple du palettiseur de ITS PLC « bêta » et permet d'illustrer l'intérêt de l'approche.

4. Application au Palettiseur de ITS PLC

L'objectif du système « palettiseur » est de créer des palettes de caisses sur trois niveaux. Le palettiseur est composé d'un monte-charge pour les caisses, un corps central et d'une zone d'évacuation. Le monte-charge alimente un convoyeur à bande au moyen d'un éjecteur. Les caisses sont accumulées à la fin du convoyeur au moyen d'une barre d'arrêt. Les caisses sont ensuite prêtes à être chargées sur la palette qui est ensuite évacuée. Le système est composé de 11 capteurs (C) et de 8 actionneurs (O) dont la description est disponible en téléchargeant le manuel utilisateur de ITS PLC PE sur le site www.realgames.pt. La commande obtenue (sans le Grafcet de gestion des modes de marche) en appliquant l'algorithme de synthèse est présentée figure 5. La commande nécessite 4 observateurs : cpt, Pp, P12 et P2 qui indique respectivement le nombre de couches de caisses sur la palette (entier, cpt=0, 1, 2 ou 3), si une palette est présente sur le monte-charge (booléen, Pp=0 ou 1), si une caisse est en cours de déplacement sur le convoyeur d'alimentation (booléen, P12=0 ou 1) et si 2 caisses sont présentes en fin du convoyeur d'alimentation (booléen, P2=0 ou 1). Les contraintes de sécurité, par manque de place, ne sont pas expliquées dans cet article et sont seulement données à titre indicatif.



$$O0\begin{cases} f_0 = 0 \\ f_1 = O0^*.C0^*.C0 \quad g_1 = X11 \end{cases}$$

$$O1\begin{cases} f_0 = C1 + \overline{O1^*}.(P12 + C2 + \overline{C0}) \\ f_1 = O1^*.\overline{C1} \quad g_1 = X51 \end{cases}$$

$$O2\begin{cases} f_0 = \overline{O2^*}.(\overline{P2} + \overline{C3} + \overline{C7}.\overline{C8}.\overline{C9} + C7.cpt > 0 + C8.cpt > 1 + C9.cpt > 2) \\ f_1 = O2^*.\overline{C4} \quad g_1 = X41 \end{cases}$$

$$O3\begin{cases} f_0 = \overline{O3^*}.C5 \\ f_1 = O3^*.\overline{C4} \quad g_1 = X41 + X42 \end{cases}$$

$$O4\begin{cases} f_0 = \overline{O4^*}.\overline{C4} + C3 \\ f_1 = O4^*.\overline{C5}.\overline{C3} \quad g_1 = X42 + X43 \end{cases}$$

$$O5\begin{cases} f_0 = \overline{O5^*}.\overline{Pp} + C7 \\ f_1 = O5^*.\overline{C7} \quad g_1 = X31 \end{cases}$$

$$O6\begin{cases} f_0 = \overline{O6^*}.(\overline{C7}.\overline{C8}.\overline{C9} + \overline{C3} + C7.cpt <> 1 + C8.cpt <> 2 + C9.cpt <> 3) \\ f_1 = O6^*.\overline{C6}.\overline{C8}.\overline{C9} \quad g_1 = X33 + X35 + X37 \end{cases}$$

$$O7\begin{cases} f_0 = \overline{C6} \\ f_1 = O7^*.C10^*.C10.C6 \quad g_1 = X21 \end{cases}$$

$CSc_1 = \uparrow O1.O2$ $(priorité\ à\ O2)$ $\quad CSc_2 = O7.O5$ $(priorité\ à\ O5)$ $\quad CSc_3 = O7.O6$ $(priorité\ à\ O6)$
$CSc_4 = \uparrow O2.\overline{O3}$ $(priorité\ à\ O3)$ $\quad CSc_5 = \uparrow O3.\overline{O2}$ $(priorité\ à\ O2)$

Figure 5 Synthèse de la commande par contraintes de sécurité pour le palettiseur

On notera toutefois que les CSc_4 et CSc_5 ne respectent pas l'équation 17. Ces 2 contraintes combinées imposent que les sorties O2 et O3 soient activées simultanément. Les priorités retenues n'entrainent pas d'incohérence avec l'ensemble des contraintes. La coordination des tâches est donc dans ce cas gérée par les contraintes de sécurité. La solution obtenue est simple et une modification des spécifications fonctionnelles est simple à prendre en compte. Enfin, la solution obtenue est moins complexe qu'une approche « classique » d'obtention du GRAFCET au moyen d'une analyse structurée par tâches.

5. Conclusion

Dans le cadre d'une collaboration scientifique et technique entre le CReSTIC (Centre de Recherche en STIC, EA3804) de l'Université de Reims Champagne-Ardenne et la société Real

Games au Portugal, des simulateurs de systèmes manufacturiers (la collection ITS PLC), reposant sur l'utilisation des technologies des jeux vidéo (moteur 3D, moteur physique, rendu sonore) ont été réalisés. Cet article a dans un premier temps présenté la version ITS PLC « bêta » permettant de réaliser des scripts en IronPython. La seconde partie de l'article s'est intéressée à une méthode originale et sûre de fonctionnement de synthèse de la commande. L'intérêt et le bien-fondé de l'approche ont été testés sur plusieurs systèmes simulés de la collection ITS PLC dont le «palettiseur» qui a été présenté dans cet article. Notons enfin que ces contrôleurs sont également utilisés pour vérifier la qualité des modèles de Parties Opératives. En effet, ces derniers reposent sur des moteurs physiques dont les solveurs peuvent parfois présenter quelques faiblesses.

6. Bibliographie

[1] Riera B., Marangé P., Gellot F., Nocent O., Magalhães A. P., Vigário B. (2009) *"Complementary usage of real and virtual manufacturing systems for safe PLC training"* 8th IFAC symposium on Advances in control Education, ACE'09, Kumamoto, Japan, October 2009.

[2] 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, August 2010.

[3] Riera B., Vigario B., Benlorhfar R., Correia L., Gellot F., Louppe C. (2010) *« Simulateur 3D interactif de Parties Opératives pour la formation et la recherche en SED »*, 3èmes Journées Démonstrateurs 2010 du club EEA, Angers, France.

[4] 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, August 2010.

[5] 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, June 2009.

[6] 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 , Milan, Italie, août 2011.

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

[8] Marangé P., Gellot F., Riera B., (2007) *"Remote control of automation systems for D.E.S. course"*, IEEE Transaction on Industrial Electronics Special Section, pp 3103-3111, 2007

[9] 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

[10] 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.

[11] Riera B., Philippot A., Annebicque D., Gellot F., (2012). *"Safe control synthesis based on Boolean constraints for manufacturing systems"*. 8th Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS 2012), Mexico City, Mexico, august 2012.