HAL
open science

Formalisation et vérification de contraintes fonctionnelles dans le cadre d'un contrôle par le produit

Pascale Marangé, David Gouyon, Jean-François Pétin, François Gellot

To cite this version:

Pascale Marangé, David Gouyon, Jean-François Pétin, François Gellot. Formalisation et vérification de contraintes fonctionnelles dans le cadre d'un contrôle par le produit. Sixième Conférence Internationale Francophone d'Automatique, CIFA 2010, Jun 2010, Nancy, France. pp.CDROM. hal-00492194

HAL Id: hal-00492194

https://hal.science/hal-00492194v1

Submitted on 15 Jun 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Formalisation et vérification de contraintes fonctionnelles dans le cadre d'un contrôle par le produit

Pascale MARANGE[1a], David GOUYON[2], Jean-François PETIN[2], François GELLOT[1b]

[1]URCA - Centre de Recherche en STIC
[a] IUT de Troyes, 9 rue du Québec - BP396, 10026 Troyes cedex, France
[b] UFR Sciences Exactes et Naturelles de Reims, BP 1039, 51687 Reims, France
{pascale.marange, francois.gellot}@univ-reims.fr

[2]Centre de Recherche en Automatique de Nancy,
UMR 7039 – Nancy-Université, CNRS
Faculté des Sciences et Techniques, BP 70239, Vandœuvre-lès-Nancy, France.
{david.gouyon, jean-francois.petin}@cran.uhp-nancy.fr

Résumé— **La validation par filtrage robuste de la commande consiste à déployer, en ligne, un ensemble de règles de sécurité et/ou fonctionnelles interdisant l'exécution de commandes illicites. Cette communication porte sur la formalisation des contraintes fonctionnelles en s'attachant à vérifier à l'aide de techniques de model-checking que, quelle que soit la commande, ces contraintes sont suffisantes pour prévenir toute action non-conforme au regard des spécifications tout en garantissant l'existence d'au moins une trace d'exécution conduisant à l'état final du produit. Cette approche de filtrage fonctionnel s'avère particulièrement pertinente pour la commande de systèmes fabriquant des produits à très forte variabilité qui requiert de nombreuses reconfigurations des lois de commande, sources importantes d'erreurs et de non conformité. Dans ce contexte, l'implantation du filtre de validation fonctionnelle sur chaque produit lui confère un rôle actif de contrôle et de surveillance des opérations qu'il subit. Un cas d'étude proposé par la société TRANE illustre la démarche.**

Mots-clés— **Filtrage de la commande, contraintes fonctionnelles, vérification formelle, contrôle par le produit.**

I INTRODUCTION

Le développement de systèmes de commande sûrs propose habituellement deux approches [9], l'une a priori par synthèse des lois de commande [17] et l'autre, a posteriori, par vérification formelle de propriétés à l'aide de model-checking [4] ou de calcul symbolique [18]. Ces deux approches permettent de s'assurer, respectivement par construction ou par vérification, que les lois de commande respectent les exigences initiales. Partant du constat qu'il est difficile d'envisager toutes les scénarios possibles, à ces deux approches doit s'ajouter une voie complémentaire [2] qui consisterait à ne plus faire d'hypothèses sur les lois de commande implantées mais à garantir, lors de l'exécution du programme, les propriétés fonctionnelles et/ou de sécurité [11] au travers de la mise en œuvre d'un filtre interdisant l'exécution de commandes illicites. Cette communication porte sur la définition des contraintes à embarquer dans un filtre de validation fonctionnelle en s'attachant à démontrer que, quelle que soit la commande, ces contraintes sont suffisantes pour prévenir toute action non-conforme au regard des spécifications du produit, tout en étant suffisamment permissives pour garantir l'existence de trajectoires de commande permettant d'atteindre l'état final du produit. La section 2 rappelle l'intérêt et les principes généraux d'une approche par filtrage de la commande ainsi que le cadre du contrôle par le produit, ce dernier apparaissant comme pertinent pour répondre à la forte variabilité des produits induite par une production personnalisée en fonction des besoins des clients [14]. La section 3 détaille une démarche générique de définition et vérification des contraintes fonctionnelles exprimées sous la forme de pré et post conditions. Cette démarche est appliquée, dans la section 4, sur un cas d'étude proposé par la société TRANE. Enfin, la dernière section présente les conclusions et perspectives.

II CONTEXTE ET PROBLEMATIQUE

A *Validation par filtrage en ligne de la commande*

Les approches couramment utilisées pour assurer la sûreté de fonctionnement des installations, s'appuient sur un modèle de la commande, et mettent en œuvre des techniques formelles de vérification et de synthèse de la commande. Cependant :

- la vérification porte sur le modèle de la commande et non sur le code implanté, et ne prend donc pas en compte divers paramètres tels que la génération du code ou encore le système d'exploitation de la machine cible,
- les programmes de commande peuvent évoluer dans le temps, notamment à la suite d'opérations de maintenance ou pour tenir compte d'une demande spécifique à un produit, ce qui peut là aussi engendrer des comportements non prévus lors de la conception,
- le système de commande est conçu pour un contexte d'utilisation donné duquel peut sortir l'opérateur.

Ce dernier cas est particulièrement sensible pour la commande et le pilotage de systèmes fabriquant des produits à très forte variabilité. En effet, cette situation induit une fréquence importante de reconfiguration des lois de commande dont la sélection et l'exécution restent dans la plupart des cas sous le contrôle des opérateurs, et donc sujette à des erreurs. Ce problème se traduit, dans le cas de l'étude proposé par la société TRANE, par un nombre important de

rebuts dû à des choix erronés lors de l'exécution d'opérations de fabrication d'unités de climatisation personnalisées.

Pour détecter et empêcher les évolutions de la commande dangereuses ou non conformes aux attentes de la gamme, une approche de filtrage est possible. Ce filtre, classiquement placé entre la partie commande (PC) et la partie opérative (PO) d'un système automatisé, n'autorise que les seules actions vérifiant un ensemble de contraintes permettant de garantir un fonctionnement sûr du système sous contrôle. L'utilisation d'un filtre pour effectuer de la surveillance de système n'est pas une idée nouvelle [7][10], mais elle semble rester pertinente dans le cadre du respect de spécifications produit. La difficulté de ces approches réside alors dans la définition des spécifications de ces filtres et leurs implantations.

En se basant sur le principe de filtrage, [11] propose que le filtrage soit réalisé par différents éléments placés entre la PO et la PC :

– filtre de sécurité : ce filtre assure la sécurité au niveau du système (capteurs/actionneurs) ; les spécifications qui y sont définies, sont en fonction des entrées/sorties du système,
– filtre de validation fonctionnelle : ce filtre permet d'assurer le bon séquencement des fonctions, en fonction de la gamme logique,
– exécuteur de fonction : ce bloc permet d'exécuter des parties de commande lorsque l'opérateur en fait la demande.

Pour mettre en place cette approche de validation en ligne par filtrage, l'expert doit au préalable définir un ensemble de contraintes. Seule la définition des contraintes fonctionnelles sera décrite dans ce papier (§ IIIB), le lecteur pourra trouver plus d'informations sur la définition des contraintes de sécurité dans [12][11]. Deux propriétés essentielles doivent être vérifiées sur les contraintes fonctionnelles si l'on veut pouvoir garantir que le filtre préviendra toute exécution de commandes non-conformes vis-à-vis des spécifications du produit. Il faut vérifier dans un premier temps que les contraintes sont suffisantes pour ne pas atteindre les états interdits, c'est-à-dire représentant la fabrication d'un produit non-conforme aux exigences. Une fois que cette complétude des contraintes est vérifiée, l'expert doit s'assurer dans un second temps que l'ensemble des contraintes n'est pas trop restrictif pour permettre l'exécution d'une gamme logique, c'est-à-dire, vérifier que l'état final du produit est atteignable. Il est à noter que, dans l'esprit du filtre de validation fonctionnelle, ces contraintes ne déclenchent pas l'exécution des opérations mais se contentent de ne « laisser passer » que celles correspondant aux spécifications du produit.

B Système contrôlé par le produit

Dans un contexte de forte variabilité des produits, les contraintes fonctionnelles ainsi définies et vérifiées peuvent être spécifiques à chacune des occurrences de produit. Dans ce contexte, le produit lui-même apparaît comme le support le plus pertinent pour l'implantation du filtre de validation fonctionnelle.

Cette proposition rejoint le concept de « Système Contrôlé par le Produit » (SCP) ([14] [13] [5]) qui vise à rendre actif le produit dans l'organisation et le pilotage de sa production. Ce concept tire partie des avancées technologiques dans les domaines de l'identification automatique, de l'électronique ou encore des micro-technologies qui permettent d'embarquer des capacités informationnelles, décisionnelles voire opérationnelles dans le produit. En particulier, le produit peut être considéré comme un support pour l'implantation de lois de commande assurant son routage sur différentes ressources de production et déclenchant l'exécution de traitements spécifiques sur celles-ci [16]. Dans notre cas, l'implantation d'un filtre de validation fonctionnelle dans le produit ne lui confère pas un tel rôle de « commande » mais le limite à un rôle de « contrôle » assurant la cohérence des opérations exécutées par les ressources au regard des spécifications du produit.

Ceci conduit à répartir les filtres de sécurité [11] et de validation fonctionnelle respectivement dans les Automates Programmables Industriels (API) assurant la commande des ressources et dans le support électronique embarqué sur produit (étiquettes électroniques, puces RFID…) (figure 1).

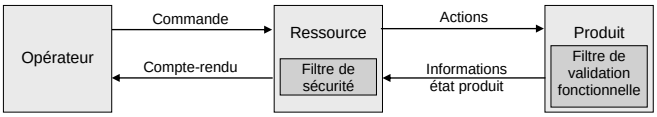


Fig. 1. Principe de répartition des blocs entre produits et ressources

Le filtre de validation fonctionnelle embarqué dans le produit permet de superviser et/ou de désactiver les fonctions non prévues par sa gamme logique. La vérification du respect des contraintes se fait alors en ligne, selon le scénario suivant (figure 2) :

– la ressource détecte la présence du produit (événement noté pp_j),
– la ressource rapatrie les informations contenues dans le produit : contraintes fonctionnelles et état du produit,
– la ressource teste la fonction proposée par l'opérateur :
  – si les contraintes sont respectées, la fonction est exécutée et le produit est mis à jour,
  – sinon, la fonction est filtrée, c'est-à-dire qu'elle n'est pas exécutée, et une erreur est indiquée à l'opérateur.

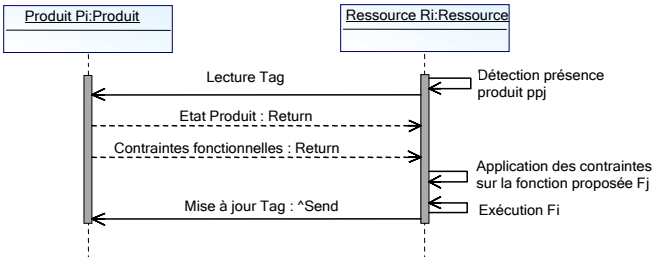


Fig. 2. Scénario d'interaction en ligne produit/ressource

III OBTENTION DES CONTRAINTES DANS LE PRODUIT

L'objectif de l'approche est d'implanter dans le produit des contraintes permettant de détecter et d'interdire des séquences d'opérations de production non conformes à une gamme logique associée à un produit. Elle nécessite pour cela une phase préalable de définition des contraintes, composée d'une étape de formalisation et d'une étape de vérification.

A Principe de définition des contraintes

Afin de garantir la bonne exécution de la gamme logique sur un produit, l'approche proposée dans ce papier se compose de trois étapes :

1. formalisation des contraintes fonctionnelles ;
2. vérification de la complétude de ces contraintes : cela consiste à vérifier que l'ensemble de contraintes est suffisant pour garantir que les séquences illicites seront filtrées ;
3. vérification de l'atteignabilité de l'état final du produit : c'est-à-dire que l'ensemble de contraintes n'est pas trop restrictif et permet l'exécution de gammes acceptables.

Les hypothèses de travail suivantes sont considérées :

– on dispose d'un modèle d'exécution des fonctions à réaliser dont tous les états sont observables,

– les contraintes logiques conditionnant l'exécution des fonctions ne contiennent que des contraintes de précédence ; elles ne contiennent pas de contraintes temporelles ni de contraintes de performances (temps d'exécution d'une fonction, temps d'attente, …) permettant l'optimisation des trajectoires de production. L'expression de ces contraintes pourrait être réalisée en CTL [6] cependant les équations logiques présentent l'avantage d'être utilisables en ligne et facilement implantables.

B Formalisation des contraintes

1 Structuration des fonctions à exécuter

La représentation des fonctions exécutables sur le système de production est structurée sous la forme d'un arbre ET/OU [15]. L'opérateur ET indique que la réalisation d'une fonction en implique l'exécution de plusieurs sous-fonctions. L'opérateur OU indique que la réalisation d'une fonction nécessite l'exécution d'au plus une des sous-fonctions. En d'autres termes, l'opérateur OU permet d'introduire une flexibilité dans les fonctions à exécuter. En utilisant de manière récursive ce principe pour toutes les fonctions nécessaires à la fabrication d'un produit, il est possible de modéliser la hiérarchie des fonctions à exécuter.

2 Modèle générique de l'exécution d'une fonction

Lors de l'exécution d'une loi de commande, une fonction F_i peut se trouver dans trois états : *non exécutée* Fte_i, *en cours d'exécution* Fe_i, ou *exécutée* Fex_i. L'automate de la figure 3 représente l'évolution de l'état de la fonction.

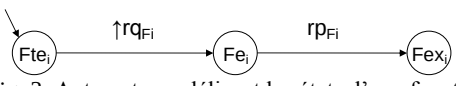


Fig. 3. Automate modélisant les états d'une fonction

Dans le cas d'une fonction ne présentant pas de « sous-fonctions » ET/OU, le passage de l'état *non exécutée* à *en cours d'exécution* se fait lorsque le système reçoit une requête, notée rq_{Fi}. De même, le passage de l'état *en cours d'exécution* à *exécutée* se fait lors de l'émission par le système du rapport d'exécution, noté rp_{Fi}.

Dans le cas d'une fonction F_i pouvant être exécutée de plusieurs manières (OU entre différentes fonctions F_{ik}), le passage de l'état *non exécutée* à *en cours d'exécution* s'effectue lorsque l'une des fonctions F_{ik} sera *en cours d'exécution* ($Fe_i = \sum_{k=0}^{n} Fe_{ik}$), et le passage de l'état *en cours d'exécution* à *exécutée* lorsque l'une des fonctions F_{ik} aura été *exécutée* ($Fex_i = \sum_{k=0}^{n} Fex_{ik}$).

Dans le cas d'une fonction F_i nécessitant l'exécution de plusieurs fonctions (ET entre différentes fonctions F_{ik}), le passage de l'état *non exécutée* à *en cours d'exécution* s'effectue lorsque l'une au moins des fonctions F_{ik} sera *en cours d'exécution* ($Fe_i = \sum_{k=0}^{n} Fe_{ik}$), et le passage de l'état *en cours d'exécution* à *exécutée* lorsque toutes les fonctions F_{ik} auront été *exécutées* ($Fex_i = \prod_{k=0}^{n} Fex_{ik}$).

3 Contraintes sur l'exécution des fonctions

Pour assurer qu'une fonction Fi s'exécutera correctement par rapport à la gamme logique (liée aux spécificités des produits), et aux possibilités des ressources, la requête d'exécution rq_{Fi} ne doit être prise en compte que si cette fonction appartient à la gamme (condition C_{specFi} vraie lorsque Fi appartient à la gamme), et lorsque certaines conditions fonctionnelles de précédence sont respectées (condition C_{pFi}). Ainsi, il est nécessaire d'enrichir le modèle de la figure 3 en ajoutant une garde G_{Fi} sur la prise en compte de la requête rq_{Fi} avec $G_{FI} = C_{specF_i} \wedge C_{pF_i}$ (figure 4).

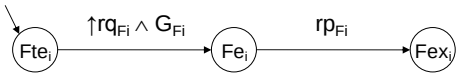


Fig. 4. Automate prenant en compte la garde G_{Fi}

La condition sur les contraintes de précédence C_{pFi} de la fonction Fi dépend de l'état des autres fonctions. En d'autres termes, l'exécution d'une fonction F_i peut nécessiter que certaines fonctions F_k aient été précédemment exécutées, (représentées par l'ensemble, noté Σ_{ex}, des fonctions F_k dans l'état Fex_k); ou bien soient en cours d'exécution (représentées par l'ensemble, noté Σ_e, des fonctions dans l'état Fe_k), ou n'aient pas encore été exécutées (représentées par l'ensemble, noté Σ_{te}, des fonctions dans l'état Fte_k). L'algorithme suivant permet d'obtenir la contrainte C_{pFi} pour une fonction F_i donnée.

Algorithme - Calcul de la contrainte fonctionnelle C_{pFi}

$CpFi \longleftarrow true$ *(initialisation de la contrainte)*
Pour k =1 **à** n **faire** *(n est le nombre de fonctions appartenant à l'arbre ET/OU)*
  Cas de
    Cas 1 : $Fk \in (\Sigma_{te} \cap \Sigma_e \cap \Sigma_{ex})$ **faire** $CpFi \longleftarrow CpFi$;
    Cas 2 : $Fk \in (\Sigma_{te} \cap \Sigma_e \cap \overline{\Sigma_{ex}})$ **faire** $CpFi \longleftarrow CpFi \wedge \overline{Fex_k}$;
    Cas 3 : $Fk \in (\Sigma_{te} \cap \overline{\Sigma_e} \cap \Sigma_{ex})$ **faire** $CpFi \longleftarrow CpFi \wedge \overline{Fe_k}$;
    Cas 4 : $Fk \in (\overline{\Sigma_{te}} \cap \Sigma_e \cap \Sigma_{ex})$ **faire** $CpFi \longleftarrow CpFi \wedge \overline{Fte_k}$;
    Cas 5 : $Fk \in (\Sigma_{te} \cap \overline{\Sigma_{te}} \cap \overline{\Sigma_{te}})$ **faire** $CpFi \longleftarrow CpFi \wedge Fte_k$;
    Cas 6 : $Fk \in (\overline{\Sigma_{te}} \cap \Sigma_e \cap \overline{\Sigma_{ex}})$ **faire** $CpFi \longleftarrow CpFi \wedge Fe_k$;
    Cas 7 : $Fk \in (\overline{\Sigma_{te}} \cap \overline{\Sigma_e} \cap \Sigma_{ex})$ **faire** $CpFi \longleftarrow CpFi \wedge Fex_k$;
  Fin Cas de
Fin pour
Retourne $CpFi$

A priori, les contraintes de précédence peuvent s'évaluer sur la totalité des fonctions de l'arbre, même si dans ce cas, certaines contraintes seront redondantes. Dans la pratique, seules les contraintes sur les fonctions opérationnelles, correspondant aux feuilles de l'arbre seront évaluées même si cela peut entrainer quelques incomplétudes. Afin de lever les ambigüités relatives à ces deux points (complétude) et de s'assurer que les contraintes ne seront pas trop restrictives

pour permettre l'exécution d'au moins une gamme (atteignabilité), il faut donc procéder à une étape de vérification. Nous avons choisi pour la modélisation des automates temporisés (afin de prendre en compte d'éventuelles temporisations dans l'exécution des fonctions) et communicants (afin d'éviter le produit synchrone générateur de problèmes d'explosion combinatoire). La technique de vérification retenue est le model-checking [20] avec l'outil UPPAAL [3].

C *Vérification des contraintes*

1 *Modèles génériques de l'environnement*

Considérant que le système est piloté via un API, et que les produits sont tous équipés d'une étiquette électronique (« tag »), la modélisation repose sur des modèles : de l'environnement d'exécution, de lecture et de mise à jour des étiquettes, de l'évolution de la commande, de l'évolution des fonctions.

Le modèle d'environnement (figure 5) tient compte du fonctionnement cyclique de l'API et de la possibilité d'avoir des évolutions simultanées, aussi bien au niveau de la commande que de la partie opérative. Pour effectuer la vérification des contraintes fonctionnelles, le point de vue adopté est celui du produit, d'où la définition du cycle API suivante : lecture des entrées, lecture du tag, exécution des fonctions, évolution de la commande, application des contraintes, mise à jour des sorties et du tag. Avant l'exécution d'un nouveau cycle de l'API, une étape de vérification est ajoutée pour permettre la vérification des propriétés de complétude et d'atteignabilité.

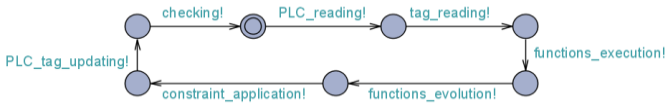


Fig. 5. Modèle générique de l'environnement d'exécution

Quand un produit est présent sur une ressource, les informations contenues dans l'étiquette sont lues, si les fonctions respectent les contraintes, la commande est alors exécutée, puis l'étiquette est mise à jour relativement à l'état des fonctions. Les étapes de lecture et de mise à jour sont effectuées à chaque cycle API pour assurer la cohérence entre l'état du produit et la commande.

Aucune hypothèse sur la commande implantée ne peut être faite, car dépendante de l'opérateur, nous savons seulement que les fonctions peuvent être activées ou désactivées entre les réceptions des messages *function_evolution* et *constraint_application* (figure 6).

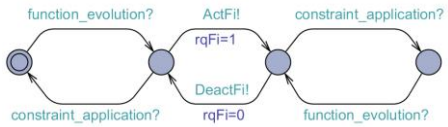


Fig. 6. Modèle générique d'activation/désactivation des fonctions

Avant l'exécution d'une fonction, une étape d'application des contraintes teste si l'évolution des fonctions proposée par la commande respecte l'ensemble des contraintes. Pour cela, lors de la réception du message *constraint_application* la garde G_{Fi} (définie en section III.B) représentant la contrainte doit être fausse, sinon la variable $error_i$ est mise à vrai et interdira l'exécution de la fonction.

Chaque fonction (figure 7) est modélisée par un automate à 3 états : {*Non exécutée*, *En cours d'exécution*, *Exécutée*}. L'automate passe de l'état *Non exécutée* à l'état *En cours d'exécution* quand la demande d'activation de la fonction est vraie (rq_{Fi}=1), si un produit est présent (pp_i=1) et si toutes les contraintes sont vérifiées ($error_i$=0). La fonction est considérée comme exécutée quand le temps *function_time* s'est écoulé, et les états Fe_i et Fex_i sont mis à jour.

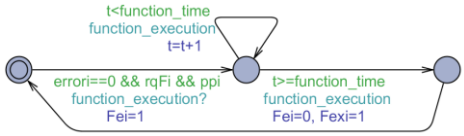


Fig. 7. Modèle générique de l'exécution d'une fonction (F_i)

L'utilisation des automates temporisés permet de simplifier les modèles grâce aux synchronisations qui évitent le produit synchrone et donc l'explosion combinatoire.

2 *Vérification de la complétude*

Pour un ensemble de contraintes fonctionnelles, la complétude doit être vérifiée, pour assurer que toutes les évolutions ne conduisant pas à la fabrication correcte d'un produit seront filtrées, en d'autres termes que les contraintes sont suffisantes. Pour cela, un observateur contenant un état « *forbidden* » représentant le non respect de la spécification, est défini de matière intuitive par l'expert pour chaque contrainte. Si l'ensemble de contraintes est suffisant, alors la propriété « Quelle que soit la commande implantée, l'état « *forbidden* » n'est pas accessible » est toujours vraie.

S'il est simple d'exprimer la propriété de complétude, les principales difficultés résident dans la définition de l'observateur (mauvaise définition, oubli d'un élément, …) ou dans le risque d'explosion combinatoire au niveau de la définition du modèle et des calculs de vérification. Ces deux facteurs justifient l'utilisation d'observateurs modulaires, c'est-à-dire qu'un observateur sera défini pour chaque contrainte.

3 *Vérification de l'atteignabilité*

La définition des contraintes étant faite de manière intuitive par un expert, il est possible qu'utilisées ensembles, ces contraintes soient trop restrictives, c'est-à-dire qu'elles pourraient empêcher l'exécution complète de la gamme logique du produit. Une fois la complétude des contraintes vérifiée, il faut donc vérifier l'atteignabilité de l'état final du produit. Cette vérification peut se faire par un raisonnement par l'absurde, en utilisant un automate à deux états (figure 8) contenant notamment un état *finished_product* accessible uniquement lorsque toutes les fonctions devant être exécutées sur un produit sont terminées. Pour s'assurer de l'atteignabilité de l'état final du produit, autrement dit de la possibilité d'aller jusqu'à la fin de la fabrication du produit, la propriété « il n'y a aucun chemin permettant d'atteindre l'état *finished_product* », est testée. Si le model checker ne vérifie pas cette propriété et renvoie un contre exemple, cela signifie

que les contraintes ne sont pas trop restrictives, le contre exemple fournissant un chemin possible. Si par contre la propriété est vérifiée, cela signifie que les contraintes sont trop restrictives et qu'il est impossible de fabriquer complètement le produit.

Fig. 8. Modèle générique de l'observateur d'atteignabilité

Dans ce dernier cas, il est nécessaire de modifier l'ensemble de contraintes, d'en vérifier à nouveau la complétude, puis l'atteignabilité.

IV APPLICATION

Cette approche s'applique dans le cadre de la fabrication de climatiseurs industriels par la société TRANE. Ces climatiseurs de grande taille sont personnalisés pour chaque client, et peuvent induire des erreurs d'exécution des gammes de la part des opérateurs. L'approche a ici pour but d'assurer que les tâches effectuées par l'opérateur sont correctes par rapport aux spécifications définies dans la gamme logique.

Nous nous intéressons dans ce papier à l'assemblage semi-automatique de ventilateurs sur un support. Pour des raisons de sécurité, l'opérateur positionne les éléments nécessaires à l'entrée du système, puis déclenche la fonction d'assemblage via un pupitre, selon un séquencement défini (figure 9). Le positionnement des ventilateurs est ensuite réalisé de manière automatique.

N° Tâche	Description de la tâche	Critères de qualité & sécurité	Nb d'opérateurs requis
10	Mettre en place la palette de moteur sur la table élévatrice avec le bandeau batterie	Vérifier numéros des moteurs suivant la liste des matières	1
20	Positionner les ventilateurs à 1 vitesse sur le bandeau batterie	Voir bon de commande position et si option variateur de vitesse	1
30	Positionner les ventilateurs à 2 vitesses sur le bandeau batterie		1
40	Mettre les panneaux de toiture gauche et droite		1

Fig. 9. Séquencement des opérations sur le cas d'étude TRANE

En accord avec la commande du client, l'opérateur sélectionne les ventilateurs selon deux vitesses dans deux magasins différents (m1 et m2), et enclenche une des sept fonctions disponibles sur le pupitre (cases grisées dans le tableau de la figure 10).

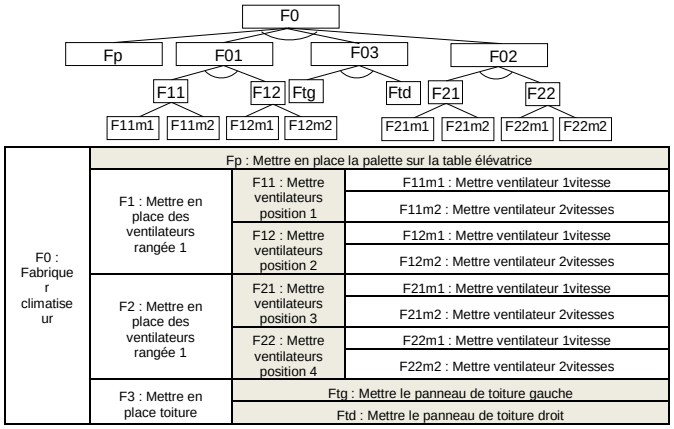


Fig. 10. Tâches pouvant être exécutées par l'opérateur

Pour éviter les erreurs de lancement dans les opérations de montage, certes routinières d'une commande à l'autre mais pouvant néanmoins comporter des différences relatives à la personnalisation des produits, l'approche de filtrage fonctionnelle a été mise en œuvre. Dans ce papier, les modèles des contraintes relatives à l'exécution de la fonction *F11m1* et les observateurs de complétude et d'atteignabilité correspondants sont présentés.

A *Formalisation des contraintes*

Afin de formaliser les contraintes, des conditions d'exécution sont données pour chacune des fonctions exécutables (figure 11), seules les conditions de la fonction *F11m1* sont détaillées :

– il faut que la fonction *Fp* soit exécutée,
– il ne faut pas que la fonction *F11* soit déjà en exécution ou exécutée,
– il ne faut pas que la fonction *F12m2* ait été exécutée ou que F12 soit en cours d'exécution,
– il ne faut pas que la mise en place des ventilateurs ait commencée sur la rangée 2 (*F02*).

Fonction	Fte	Fe	Fex
Fp	Ø	Ø	Ø
F11m1	F11, F12m2, F12, F02	Ø	Fp, F02, F12
F11m2	F11, F12m1, F12, F02	Ø	Fp, F02, F12
F12m1	F12, F11m2, F11, F02	Ø	Fp, F02, F11
F12m2	F12, F11m1, F11, F02	Ø	Fp, F02, F11
F21m1	F21, F22m2,F22, F01	Ø	Fp, F01, F22
F21m2	F21, F22m1, F22, F01	Ø	Fp, F01, F22
F22m1	F22, F21m2, F21, F01	Ø	Fp, F01, F21
F22m2	F22, F21m1, F21, F01	Ø	Fp, F01, F21
Ftg	Ftd	Ø	Fp, F01, F02, Ftd
Ftd	Ftg	Ø	Fp, F01, F02, Ftg

Fig. 11. Conditions sur l'exécution de chaque fonction

A partir de ce tableau de conditions, et en utilisant l'approche de formalisation des contraintes (§III.B.3) proposée, la contrainte suivante est formalisée pour la fonction *F11m1* :

$$\uparrow rq_{F11m1} \wedge C_{p_{F11m1}} \wedge C_{spec_{F11m1}} = 1$$

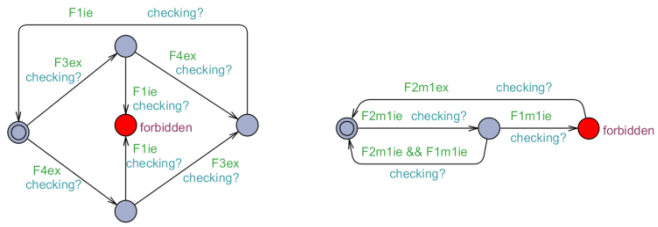
Où $C_{p_{F11m1}} = Fex_p \wedge \overline{Fe_{02}} \wedge \overline{Fe_{11}} \wedge Fte_{11m1} \wedge Fte_{12m2}$

Pour assurer le bon séquencement de la gamme logique, 10 contraintes logiques ont ainsi été définies.

B *Vérification de la complétude*

Une fois l'ensemble de contraintes défini, il faut s'assurer que celui-ci est suffisant pour garantir que les tâches commandées par l'opérateur ne pourront pas conduire à la fabrication d'un mauvais produit. Pour cela, 26 modèles définissent le comportement système. Dans l'exemple de la fonction *F11m1*, il faut définir sept observateurs pour vérifier:

– que l'exécution de *F11m1* a lieu uniquement après *Fp*,
– que *F11m1* ne peut pas s'exécuter si une séquence composée de *F21* et *F22* est en cours d'exécution (figure 12.a),
– que *F11m1* ne peut s'exécuter en même temps que *F12* (figure 12.b),
– que *F11m1* ne peut s'exécuter en même temps que *F21*,
– que *F1m1* ne peut s'exécuter en même temps que *F22*,
– que *F11m1* ne peut plus être exécuté après *F12m2*,
– que *F11m1* ne peut pas être ré-exécuté.



a) Impossibilité d'exécuter *F11m1* pendant F02

b) Impossibilité d'exécuter *F11m1* et *F12* en même temps

Fig. 12. Observateur pour la vérification de la complétude

La simulation sous UPPAAL a montré que les 10 contraintes étaient suffisantes pour que le système d'observateur n'atteigne pas d'état «*forbidden*», ce qui signifie que l'ensemble de contraintes permettra l'exécution correcte du séquencement de tâches.

C *Vérification de l'atteignabilité*

Une fois que la complétude a été vérifiée, il faut s'assurer que les contraintes ne sont pas trop contraignantes. Pour qu'un produit « ventilateur » soit fini, il faut que les sept fonctions soient exécutées (figure 13).

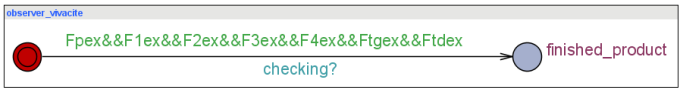


Fig. 13 . Observateur pour la vérification de l'atteignabilité

La simulation sous UPPAAL a montré que les 37 contraintes assuraient l'atteignabilité de l'état final du produit car elles permettent d'atteindre l'état « *finished_product* ».

V CONCLUSION

Ce papier propose une approche de filtrage robuste de la commande à des fins de validation fonctionnelle complémentaires au travail réalisé par [11] sur la validation des contraintes de sécurité. Les principales contributions portent sur la formalisation des contraintes fonctionnelles ainsi que sur la vérification, par model-checking, de leur complétude afin de ne laisser passer aucun déclenchement de fonction non conforme tout en garantissant l'obtention d'au moins une trajectoire d'exécution. Une application originale de l'approche a été proposée en déportant l'implémentation du filtre fonctionnel sur le produit. Ce choix trouve tout son intérêt pour la fabrication de produit à très forte variabilité, comme le cas d'étude proposé par la société TRANE, afin de réduire les risques d'erreurs engendrés par la multiplicité des gammes auxquelles doivent faire face les opérateurs.

En termes de perspectives, trois axes de recherche sont en en cours de développement. Il s'agit, d'une part, de prendre en compte le temps dans la formalisation des contraintes fonctionnelles, et d'autre part, de prendre en compte la définition et la vérification des contraintes sous observation partielle du modèle d'exécution des fonctions (par exemple, seul un compte-rendu de fin d'exécution est observable). Le dernier axe s'intéresse à la mise à l'échelle de cette approche dans le cadre industriel en proposant une approche structurée de définition des contraintes. À plus long terme, le filtre fonctionnel implanté sur le produit, aujourd'hui limité à un rôle de contrôle et de validation des fonctions exécutées, pourrait être étendu pour jouer un rôle plus actif dans une optimisation locale des trajectoires de production. Ceci implique les capacités embarqués dans le produit lui permettent d'avoir une vision globale sur l'état des ressources et des critères d'optimisation.

REMERCIEMENTS

Nous tenons à remercier vivement la société TRANE et plus particulièrement Melle Hind El Haouzi, Manufacturing System Engineer, pour leur collaboration autour du cas d'étude proposé.

REFERENCES

[1] Allen J.F., Maintaining knowledge about temporal intervals, Commun. ACM, 26(11): pp 832–843, novembre 1983.

[2] Arlat J., Fabre J.-C., Rodríguez M., Salles F., Dependability of COTS Microkernel-based Systems. *IEEE Trans. on Computers*, 51 (2), pp.138-163, 2002.

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

[4] Berard B., Bidoit M., Finkel A., Laroussinie F., Petit A., Petrucci L., Schnoebelen P., Systems and software verification: Model-checking techniques and tools, Heidelberg, Springer-Verlag Edition, 1999.

[5] Cardin O., Castagna P., Using online simulation in Holonic Manufacturing Systems, *Engineering Applications of Artificial Intelligence*, 2009.

[6] Clarke E.M., Emerson E.A., Sistla A.P, Automatc verification of finite-state concurrent systems using temporal logic specifications, *ACM Transactions on Programming Languages and systems*, 8(2) 244-263, 1986.

[7] Combacau M., Commande et surveillance des systèmes à événements discrets complexes : application aux ateliers flexibles, thèse de doctorat de l'Université Paul Sabatier de Toulouse, 1991.

[8] Da Silveira G., Borenstein D., Fogliatto F.S., Mass customization: literature review and research directions, *Int. Journal of Production Economics*, 72, pp 1-13, 2001.

[9] 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, Vienna (Austria), CD-Rom paper, 6 pages, 2001.

[10] Lhoste P., Contribution au génie automatique : concepts, modèles, méthodes et outils, Habilitation à diriger des recherches de l'Université de Nancy, février 1994.

[11] Marangé P., Synthèse et filtrage robuste de la commande pour des systèmes manufacturiers sûrs de fonctionnement, PhD dissertation, Université de Reims Champagne Ardenne, 2008.

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

[13] Mc Farlane D., Sarma S., Chirn J-L., Wong C.Y., Ashton K., Auto ID systems and intelligent manufacturing control, *Engineering Application of Artificial Intelligence*, 16, pp. 365-376, 2003.

[14] Morel G., Valckenaers P., Faure J-M., Pereira C., Diedrich C., Survey paper on manufacturing plant control challenges and issues , Proceedings of the 16th IFAC world congress in Prague, ISBN 008045108X, 2005.

[15] Nilsson Nils J., *Principles of artificial intelligence*, Morgan Kauffman Publishers, ISBN : 978-0934613101, 1982.

[16] Pétin J-F., Gouyon, D. Morel G., Supervisory synthesis for product-driven automation and its application to a flexible assembly cell, *Control Engineering Practice*, 15, pp. 595-614, 2007.

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

[18] Roussel J-M., Denis B., Safety properties verification of ladder diagram programs, *Journal Européen des Systèmes Automatisés JESA*, Hermès Editions, 36(7), pp905–917, ISSN 1269-6935, 2002.

[19] Roussel J-M., Lesage J.J, Une Algèbre De Boole Pour l'Approche Événementielle Des Systèmes Logiques, APII-AFCET/CNRS, Ed Hermès, Vol. 27-N°5, pp. 541-560, Décembre 1993.

[20] Schnoebelen P., Bérard B., Bidoit M., Laroussinie F., Petit A., Vérification de logiciels: Techniques et outils du model-checking, Vuibert, ISBN 2-7117-8646-3, 1999.