Pilotage sûr et optimal d'une flotte de véhicules autoguidés

Yoann Arnaud[1], José E.R. Cury[2], Jean-jacques Loiseau[1], Claude Martinez[1]

[1]IRCCyN - Institut de Recherche en Communication et Cybernétique de Nantes
1, Rue de la Noë - BP 92 101 - 44321 Nantes CEDEX 03, France

[2]DAS - CTC - Federal University of Santa Catarina
Florianopolis SC 88040-900, Brazil

yoann.arnaud@irccyn.ec-nantes.fr
cury@das.ufsc.br
jean-jacques.loiseau@irccyn.ec-nantes.fr
claude.martinez@irccyn.ec-nantes.fr
http://www.irccyn.ec-nantes.fr

Résumé— **Ce papier traite de la conception et de la réalisation d'un système de pilotage en ligne de véhicules autoguidés (dits AGV). Ce système assure à la fois la sûreté du processus et l'optimisation de critères de performance. La sécurité de fonctionnement, en l'occurence l'évitement de collisions et de blocages entre AGV, est réalisée à l'aide de la théorie de la supervision de Ramadge et Wonham. On superpose à ce premier niveau de commande, un module de routage des AGV. Les tronçons de circuit devant être empruntés sont choisis de façon à optimiser un certain critère de coût. L'architecture proposée est mise en oeuvre sur l'exemple d'une flotte d'AGV. Le système d'AGV est émulé avec Arena et interfacé avec les modules du système de pilotage. Cette mise en oeuvre valide les concepts proposés.**

Mots-clés— **Systèmes à évènements discrets, Théorie de la supervision, Automates à états, Pilotage, Véhicules autoguidés, AGV, Commande optimale, Arena.**

I. Introduction

Nous nous intéressons dans ce papier au pilotage sûr et optimisé des systèmes de production, et plus particulièrement, aux systèmes de transport dans les ateliers de production flexibles.

Le transport de produits dans les ateliers de production est couramment réalisé à l'aide de véhicules autoguidés (AGV, en anglais). Ces véhicules empruntent un circuit, généralement tracé au sol, afin de joindre les différentes zones de l'atelier (postes de fabrication, zone de stockage).

Lors de la conception d'un système de véhicules autoguidés, il est nécessaire de faire l'étude de plusieurs éléments. Reveliotis [14] établit un classement des points énoncés par Ganesharajah et al. [4] selon deux grandes familles. La première famille concerne la conception du circuit et la détermination de la taille de la flotte. La seconde concerne la planification et la répartition des tâches pour chaque AGV, et l'établissement d'un routage sans conflits. Cet article traite d'éléments de cette seconde famille, plus particulièrement de pilotage en temps réel de systèmes d'AGV tout en assurant un routage sans conflits des véhicules.

Il existe plusieurs sortes d'AGV et pour ceux qui sont de type mono-directionnels (qui ne peuvent pas faire demi-tour sur place) et que nous étudions, il est possible de se retrouver dans des situations de blocage où aucun AGV ne peut avancer (cf. fig. 1). Ces situations sont bien évidemment critiques et elles constituent les conflits important à éviter.

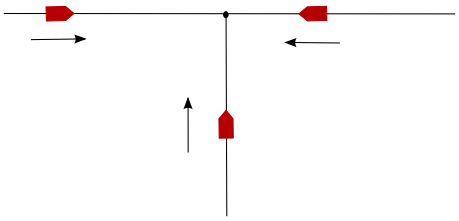

Fig. 1. Situation de blocage entre AGV monodirectionnels.

Nous souhaitons rajouter aux points cités par Reveliotis [14] celui concernant la performance des systèmes d'AGV. En ce qui concerne la résolution de problèmes de routage, la performance et la gestion des conflits sont généralement traités de pair dans la littérature. Les conflits peuvent être gérés de deux manières différentes, soit en calculant un ordonnancement prédictif du système, soit par la détection et la résolution en ligne des situations de blocage à venir. La première méthode, réalisée généralement à l'aide d'heuristiques, a l'avantage de fournir le plus souvent des solutions fonctionnelles et assez proches de l'optimal. Par contre l'occurence d'évènements non désirés (tels que des pannes ou plus couramment une dérive par rapport aux ordonnacements initialement prévus) met rapidement en lumière la faible robustesse de ces solutions. La seconde méthode permet, quand à elle, de s'adapter aux évènements aléatoires mais permet difficilement de planifier à l'avance les tâches à accomplir.

S. Maza ([8], [9], [10]) a proposé un système de pilotage mêlant les deux approches précédentes. Un ordonnance-

ment prédictif ést réalisé hors ligne pour déterminer l'ordre de passage des AGV à chaque carrefour. Puis, lorsque des pannes ou des retards se produisent, des algorithmes éxécutés en ligne permettent de déterminer de nouvelles routes en garantissant l'absence de blocages.

D'autres travaux concernant la commande en-ligne et optimisée de systèmes à évènements discrets ont été publiés ([3], [6], [7]) mais ils ne nous ont pas parus applicable aux systèmes d'AGV.

Nous proposons une nouvelle approche pour la conception d'un système de pilotage d'AGV, qui consiste à s'appuyer sur le théorie de la supervision de Ramadge et Wonham [2]. Cette théorie a pour objet la synthèse de logiques de commande pour des systèmes à évènements discrets de nature logique. Une spécification (également de nature logique) étant donnée, une commande s'obtient par un calcul sur les automates représentant le processus et la spécification. Dans le cas d'un système d'AGV, l'évitement de conflits et d'interblocages rentre dans le cadre applicatif de cette théorie. La commande calculée est maximalement permissive, ce qui laisse un maximum de degré de liberté au processus. Cela signife que le routage des AGV n'est pas imposé avec cette commande. Le superviseur de "Ramadge et Wonham" interdit les séquences de fonctionnement conflictuelles ou bloquantes, mais autorise toutes les autres. Notre proposition consiste à ajouter un module de routage et de calcul du chemin optimal suivi par les AGV.

La structure de l'article est la suivante. Tout d'abord, nous rapellerons en partie II des concepts de base concernant la commande des systèmes à évènements discrets, puis nous introduirons l'architecture de commande proposée. Dans la troisième partie, nous détaillerons sur un exemple typique la modélisation et la conception de la commande, puis sa mise en oeuvre. Enfin, nous conclurons cet article en rapellant les résultats obtenus puis en exposant les directions à étudier pour la suite de nos travaux.

II. Concepts de base

A. Supervision des systèmes à évènements discrets

A.1 Les automates à états

Les systèmes à évènements discrets sont des systèmes dont l'état change de façon discrète dans le temps. Ils peuvent être représentés par un automate $\mathbb{A}$, qui est un quintuplet $(Q, \Sigma, \delta, Q_0, Q_m)$ où :

- Q est l'ensemble des états
- Σ est l'alphabet
- δ est la relation de transition
- $q_0 \in Q$ est un état initial
- $Q_m \subset Q$ est un ensemble d'états finaux (marqués)

Chaque $\sigma \in \Sigma$ est une étiquette associée à un évènement de l'automate. L'occurence de ces évènements fait évoluer le système, en le faisant passer d'un état à un autre. Σ peut être séparé en deux sous-ensembles distincts : Σ_c l'ensemble des évènements commmandables et Σ_u l'ensemble des évènements non commmandables.

Un mot $\sigma_1\sigma_2\cdots\sigma_n$ construit avec l'alphabet Σ est appelé une trace. L'ensemble des traces que peut générer l'automate $\mathbb{A}$ forme un langage que l'on note $L(\mathbb{A})$.

δ est une fonction de $Q \times Q \to \Sigma$. Pour un couple d'état $(q_1, q_2) \in Q \times Q$, la fonction retourne :

$$\begin{cases} \sigma \text{ s'il fait passer l'automate de } q_1 \text{ à } q_2 \\ \emptyset \text{ sinon.} \end{cases}$$

A.2 La composition d'automates à états

Un processus peut être vu comme un regroupement de plusieurs sous-sytèmes. Chacun de ces sous-systèmes se modélise par un automate. Il est possible d'obtenir une modélisation du processus global en composant tous ces sous-systèmes. Soient deux automates $\mathbb{A} = (Q_a,\Sigma_a,\delta_a,q_{0a},Q_{ma})$ et $\mathbb{B} = (Q_b,\Sigma_b,\delta_b,q_{0b},Q_{mb})$. L'automate composition $\mathbb{C}$ est défini par :

$$\begin{aligned} \mathbb{C} &= \mathbb{A} \parallel \mathbb{B} \\ &= (Q_a \times Q_b, \Sigma_a \cup \Sigma_b, \psi, (q_{0a}, q_{0b}), Q_{ma} \times Q_{mb}) \end{aligned}$$

où ψ est la fonction de transition résultant de l'opération de composition.

Pour la suite de cet article, nous devons définir une fonction γ_c. Supposons que $\Sigma_a \cap \Sigma_b = \emptyset$, $Q_a = Q_b = Q$ et que δ_a et δ_b sont bijectives[1]. On définit d'abord les fonctions α_a et α_b, associées respectivement aux automates $\mathbb{A}$ et $\mathbb{B}$ telles que :

$$\alpha_x(\sigma) = \begin{cases} q_2 \text{ si } \delta_x(q_1, q_2) = \sigma \\ \emptyset \text{ sinon} \end{cases} , x \in \{a, b\}$$

Et on pose donc :

$$\gamma_c(\sigma) = \alpha_a(\sigma) \cup \alpha_b(\sigma)$$

Plus simplement, γ_c retourne le nom de l'état qui est à l'extrémité de l'arc portant l'étiquette σ.

A.3 La théorie de la supervision

La théorie de la supervision de Ramadge et Wonham ([11],[12],[13]) a pour objet la synthèse d'un superviseur. Ce superviseur agira sur un système (modélisé par un automate $\mathbb{M}$) en interdisant l'occurence de certains évènements. Il est à noter que seule l'occurence d'évènements commandables peut être interdite.

La supervision peut être caractérisée par ces trois grands principes :

- respect des spécifications
- compatibilité avec l'ensemble d'évènements non commandables Σ_u
- garantie du non blocage

Les spécifications définissent quel est le comportement attendu de notre système. Elles peuvent être définies sous la forme d'un automate modélisant ce comportement ou comme une liste d'états que l'on ne souhaite jamais atteindre. La compatibilité avec les évènements non commandables de Σ_u signifie que l'occurence d'évènements non commandables ne fera évoluer le système que vers des états vérifiant les spécifications. C'est ce que l'on appelle la commandabilité. Enfin, le non blocage assure au système supervisé de pouvoir atteindre à tout moment un état marqué. Dans notre cas, l'état initial est l'état marqué, donc cela

[1]Ces trois conditions seront remplies pour l'exemple d'application présenté en section III

revient à assurer la coatteignabilité des états par rapport à l'état initial.

Le comportement d'un système supervisé est qualifié de *maximalement permissif* car ce système peut évoluer avec un maximum de degré de liberté. Le superviseur ne fait qu'interdire l'accès à des états dangereux ou qui peuvent y mener par l'intermédiaire d'évènements non commandables. Le langage de l'automate modélisant ce système supervisé est qualifié de *plus grand langage commandable.*

Dans le cas où la spécification est exprimée en terme d'états interdits et que tous les évènements sont commandables, le problème de synthèse revient à supprimer les états interdits dans $\mathbb{M}$ et à résoudre un problème d'atteignabilité et de coatteignabilité par rapport à l'atat initial. L'automate modélisant le superviseur calculé est un sous automate de $\mathbb{M}$.

B. *Proposition d'une architecture de pilotage d'un système d'AGV*

Nous proposons de développer une architecture de pilotage hiérarchisée (cf. fig 2) à base de blocs superposés et réalisant chacun une action particulière.

Le bloc inférieur représente le système d'AGV que nous souhaitons piloter. Le bloc situé juste au dessus assurera la sécurité de fonctionnement du système en évitant les collisions entre AGV et les situations de blocage. Cette action de sécurité se doit d'être réalisée au plus près du système pour être la plus performante possible. Un troisième bloc assurera l'envoi d'ordres de mouvements aux véhicules afin qu'ils puissent réaliser leur missions. Les véhicules seront routés en respectant l'optimalité d'un critère.

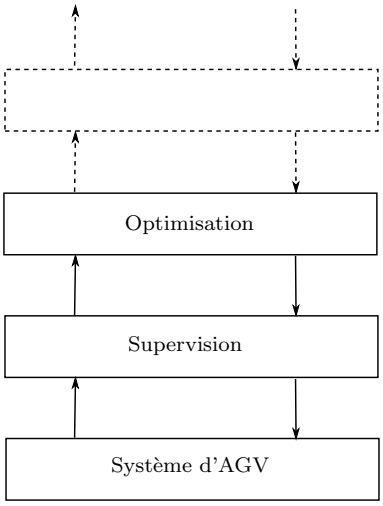

Fig. 2. Concept de l'architecture proposée.

On pourrait dans la suite de ce raisonnement ajouter d'autres blocs, par exemple, un bloc réalisant la génération de missions pour les AGV afin de respecter les gammes de fabrication des pièces transportées.

III. Un exemple d'application

On se propose de réaliser la commande supervisée et optimisée d'une flotte d'AGV circulant sur un circuit représenté fig 3. Les AGV recevront des missions à réaliser (atteindre un poste de travail, par exemple) et le système de pilotage devra router de façon optimale les AGV sur le circuit, tout en assurant l'absence de collisions et de blocages entre AGV.

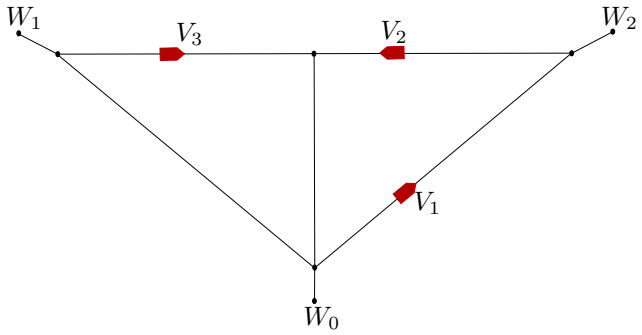

Fig. 3. Schéma du circuit d'AGV choisi.

Ce circuit se compose d'un garage W_0 qui dispose d'autant d'emplacements qu'il y a d'AGVs et de deux stations de travail (W_1 et W_2) accessibles par seulement un AGV à un instant donné. Les AGV V_i ($i \in \{1;2;3\}$) doivent se déplacer sur ce circuit afin d'assurer le déplacement de pièces d'une station de travail à une autre.

Ce circuit muni de trois AGV est un exemple intéressant vis-à-vis de nos objectifs fixés, car tout d'abord il offre des possibilités de blocage que nous chercherons plus tard à interdire, et ensuite il est relativement simple (bien que tout de même réaliste), ce qui limite l'explosion combinatoire du problème posé.

A. *Modélisation*

Le problème étudié peut être défini comme un système à évènements discrets (SED), dont l'état est défini par la position de chaque AGV sur un tronçon. Chaque changement d'état intervient quand un des AGV change de tronçon.

Étant donné que chaque AGV se déplace sur le même circuit, on modélisera dans un premier temps l'automate d'un AGV puis on composera cet automate avec lui même autant de fois qu'il y a d'AGV sur le circuit moins une fois, afin d'obtenir l'automate modélisant le système global.

On nomme les intersections du circuit par des lettres majuscules (c.f. fig 4). Si un AGV est en W_1 ou sur le tronçon qui mène à W_1, alors l'état correspondant dans l'automate sera "W1". Mêmes règle pour tous les W_i. Si un AGV est sur un tronçon entre les intersections A et B dans le sens A vers B, alors le nom que prendra l'état de l'automate sera "AB". Mêmes règles pour tous les tronçons situés entre deux intersections.

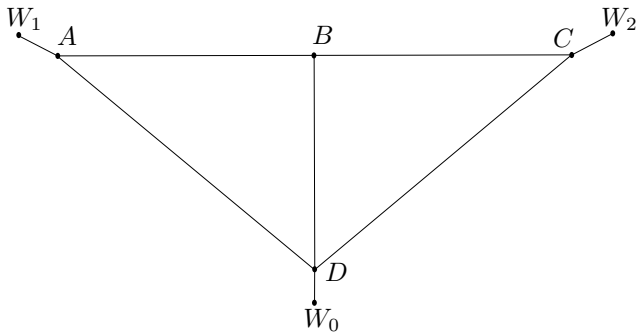

Fig. 4. Modélisation du circuit.

On obtient donc, pour l'AGV V_1, l'automate de la figure 5. Cet automate contient 13 états, nommés comme expliqué précédemment, et 30 transitions numérotées de 1 à 30.

Les transitions de l'automate du premier AGV se nomment "$a1$", "$a2$", ..."$a30$" (c.f. Fig 5). Les transitions des automates des autres AGV n'ayant pas de lien avec celles de l'automate de V_1, il faut les nommer différemment : nous nommons "$b1$", "$b2$",..."$b30$" les transitions de l'automate de V_2 et ainsi de suite en prenant les lettres de l'alphabet dans l'ordre et en minuscule. Tous les évènements associés à ces transitions sont commandables.

Pour maintenant obtenir l'automate $\mathbb{M}$ modélisant l'ensemble du système, on calcule la composition des automates $\mathbb{G}_i,\ i \in \{1;2;3\}$.

$$\mathbb{M} = \mathbb{G}_1 \parallel \mathbb{G}_2 \parallel \mathbb{G}_3$$

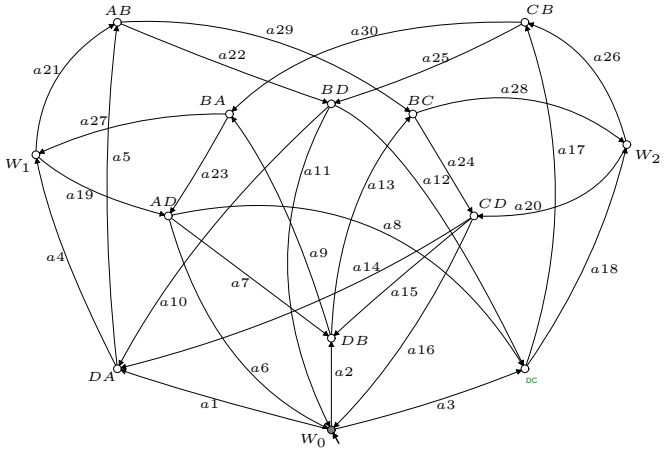

Fig. 5. $\mathbb{G}_1$: Automate de l'AGV V_1.

L'algorithme réalisant le calcul de $\mathbb{M}$ a été développé en *VB.net* (Langage Visual Basic de haut niveau, interprété et orienté objet) et vérifié à l'aide du logiciel Supremica [1]. Il prend en entrée un fichier automate.wmod au format xml construit avec Supremica (représenté à la fig 5) et calcule $\mathbb{M}$ qui sera stocké dans la mémoire vive de la machine. Un export des données de l'automate $\mathbb{M}$ dans un format donné n'est pas réalisable compte tenu du nombre d'information à exporter (ou à récupérer) et du temps que cela requiert. Cela signifie donc que le calcul de $\mathbb{M}$ est réalisé à chaque lancement du programme de pilotage.

L'automate $\mathbb{M}$ a 2197 (soit 13^3) états et 15210 (soit $3 \times 13^2 \times 30$) transitions. Un état de cet automate a comme nom $AB.DB.CB$, AB étant l'état de V_1, DB l'état de V_2 et CB l'état de V_3.

On notera que le calcul de $\mathbb{M}$ pour quatre AGV n'est pas réalisable. En effet, cet automate est censé contenir 28561 (soit 13^4) états et 263640 (soit $4 \times 13^3 \times 30$) transitions ce qui remplit les 4 Go de mémoire vive de la machine utilisée.

B. Supervision

Afin de pouvoir réaliser la synthèse d'un superviseur, il faut exprimer des spécifications de fonctionnement à satisfaire. En ce qui nous concerne, on souhaite que les AGV soient seul à occuper un tronçon ou un poste de travail (Le garage reste accessible par tous les AGV en même temps). Il faut donc supprimer les états $X.Y.Z$ de l'automate composé (et les transitions associées), tels que :

$$X = Y \neq W_0 \text{ ou } X = Z \neq W_0 \text{ ou } Y = Z \neq W_0$$

Nous sommes donc dans le cas d'une spécification exprimée sous forme d'états interdits. La synthèse du superviseur nous permet d'obtenir un automate $\mathbb{H}$ qui est un sous-automate de $\mathbb{G}$ ($\mathbb{H} \subset \mathbb{G}$). $\mathbb{H}$ a la propriété d'être non-bloquant car l'opération de synthèse a supprimé les états non coaccessibles par rapport à l'état initial qui sont responsables des blocages.

L'algorithme (en VB.net) réalisant la synthèse [2] exploite l'automate $\mathbb{M}$ (III-A) afin de calculer l'automate supervisé $\mathbb{H}$. Pour les mêmes raisons que précédemment, il est conservé dans la mémoire vive de la machine. Les résultats de cet algorithme ont aussi été vérifiés avec Supremica.

Pour l'exemple traité $\mathbb{H}$ contient 1387 états et 6858 transitions. 810 états ont été supprimés au total, dont 774 par la spécification exprimée ci-dessus et 36 par l'opération de synthèse.

C. Routage optimal de véhicules

L'automate $\mathbb{H}$ calculé en III-B est un modèle du système supervisé. Un ensemble de destination étant connu pour le système d'AGV, nous calculons le routage optimal des véhicules en cherchant dans l'automate $\mathbb{H}$ la (ou les) trace(s) assurant l'optimalité.

Nous appellerons par la suite *objectifs* les destinations à atteindre. Ces objectifs sont stockés dans un vecteur $Obj()$ tel que $Obj(i)$ est l'objectif de l'AGV V_i.

Nous choisissons de minimiser la distance totale parcourue par tous les AGV. Nous associons tout d'abord un coût à chaque évènement de $\mathbb{H}$. Nous définissons donc une fonction de coût $cost : \Sigma \rightarrow \mathbb{R}$ telle que $cost(\sigma)$ soit égal à la distance à parcourir sur le tronçon $\gamma_h(\sigma)$.

La résolution mise en place consiste à faire la liste des états par lesquels doit passer une trace de l'automate $\mathbb{H}$ pour satisfaire tous les objectifs, puis d'en déduire l'ensemble des traces qui passent par ces états. Par exemple, à un instant T donné, une trace t satisfaisant le vecteur Obj aura la forme suivante :

$$t = q_T \dots\ X.Obj(2).Z \dots\ X'.Y'.Obj(3) \dots\ Obj(1).Y''.Z'',$$

où q_T est l'état dans lequel se trouve le système à l'instant T et X, X', Y', Y'', Z et Z'' des états quelconques pour les AGV $V_i, i \in \{1;2;3\}$.

L'ensemble des traces t étant maintenant listé, nous cherchons lesquelles permettent de minimiser notre critère. Nous définissons donc la fonction $cost\_trace : L(\mathbb{G}) \rightarrow \mathbb{R}$ qui définit le coût d'une trace d'un automate. Ce coût correspond à la somme des coûts des évènements rencontrés sur cette trace et donc à la distance totale parcourue par tous les AGV (qui est notre critère à minimiser). On a alors :

$$cost\_trace(t = \sigma_1\sigma_2 \dots \sigma_n) = \sum_{i=1}^{n} cost(\sigma_i)$$

D'une manière générale, on peut décomposer t de la façon suivante,

$$t = t_1 + t_2 + t_3$$

où + représente la concaténation de deux chaînes de caractères. Avec la trace t de l'exemple ci-dessus, on obtient :

$$t_1 = q_T \dots\ X.Obj(2).Z$$
$$t_2 = X.Obj(2).Z \dots\ X'.Y'.Obj(3)$$
$$t_3 = X'.Y'.Obj(3) \dots\ Obj(1).Y''.Z''$$

On a donc :

$$cost\_trace(t) = \sum_j cost\_trace(t_j)$$

Enfin, on calcule les $cost\_trace(t_j)$ en appliquant l'algorithme de Moore-Dijkstra [5] de recherche de plus court chemin dans un graphe orienté et on en déduit la liste des traces de $\mathbb{H}$ permettant de minimiser la distance totale parcourue par les AGV.

D. Mise en oeuvre et simulation

D.1 Interfaçage et mise en oeuvre du pilotage

Le système de pilotage proposé en partie II-B a été mis en oeuvre selon l'architecture de la figure 6.

Le système d'AGV est émulé avec Arena. Des librairies génériques ont d'abord été développées pour représenter des parties de circuit (en l'occurence : intersection, garage et poste de travail). Puis un modèle basé sur ces librairies a été construit afin de reproduire le circuit de la figure 4.

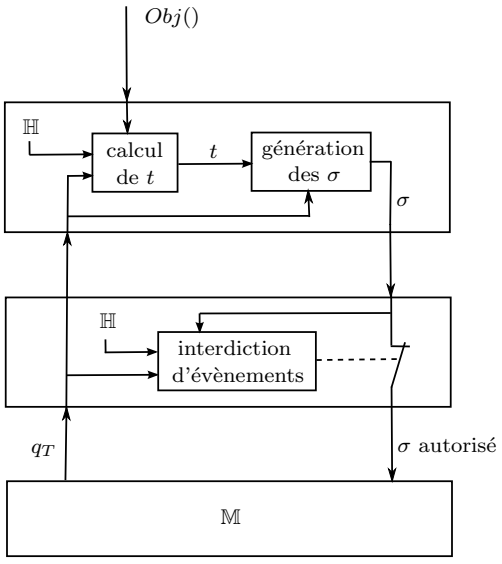

Fig. 6. Architecture de commande développée.

Ce système d'AGV communique avec le bloc de supervision (celui du milieu sur la figure 6) à l'aide de sockets. Il l'informe d'abord de l'état q_T dans lequel il se trouve. Puis le bloc de supervision déduit, grâce à la connaissance de $\mathbb{H}$, quels sont les évènements qui peuvent être éxécutés. Enfin, lorsqu'un évènement σ tente de transiter depuis le bloc supérieur vers le système $\mathbb{M}$, le bloc de supervision vérifie si cet évènement a le droit d'être éxécuté. Si ce n'est pas le cas, l'évènement est bloqué afin de préserver la sureté de fonctionnement.

À l'étage supérieur de la commande, l'optimisation interroge la supervision pour demander l'état q_T du système d'AGV. Une trace $t = \sigma_1\sigma_2 \dots \sigma_n$ optimale est ensuite choisie dans $\mathbb{H}$ et la suite d'ordres $\sigma_i, i \in \{1..n\}$ est envoyée au système d'AGV au fur et à mesure que le système commandé évolue. Dès que le vecteur $Obj()$ est modifié, une nouvelle trace optimale est automatiquement calculée et remplace l'ancienne qui est devenue obsolète compte tenu de l'évolution des objectifs.

Ce système de pilotage, d'un point de vue global, prend comme seul paramètre d'entrée une liste de destinations à atteindre pour les AGV. Nous avons donc créé un système autonome qui réagit en ligne à l'ajout de nouvelles destinations en routant de manière optimale et sûre l'ensemble des AGV.

D.2 Tests du système de pilotage

Ce système a été testé avec succès, ce qui démontre la faisabilité de ce type de structure de commande. Différents essais ont été réalisés afin de tenter de mettre en défaut la partie supervision, aucun blocage ni collision n'a été constaté. En ce qui concerne l'optimisation, les AGV ont bien été routés de façon optimale sur les cas testés.

On remarque que le temps de calcul pour une optimisation avec 3 AGV dépasse une minute, du fait du grand nombre de traces t dont on doit calculer le coût. Ceci ne permet pas d'être réactif pour des systèmes, comme celui étudié, où l'ordre de grandeur du temps écoulé entre deux changements d'état est de l'ordre de quelques secondes. Par contre, l'optimisation pour deux AGV s'exécute en moins de deux secondes, ce qui est satisfaisant pour l'observation du système d'AGV piloté.

IV. Conclusion

Nous avons proposé dans cet article une architecture de commande de systèmes à évènements discrets. Le concept de cette architecture est basé sur la superposition de modules réalisant chacun une action bien définie. Un module réalisant la sécurité de fonctionnement est placé au plus près du système à piloter puis un autre vient assurer un comportement optimal au système. Cette architecture a été développée pour un système de trois AGV se déplaçant sur un circuit. La théorie de Ramadge et Wonham a été implémenté dans le module de sécurité et un algorithme a été proposé pour minimiser la distance totale parcourue par les AGV pour atteindre leurs destinations. Le tout a été mis en oeuvre pour piloter en ligne un système d'AGV émulé avec Arena. Les simulations de pilotage montrent que les fonctions de sécurité et d'optimalité sont bien réalisées. Cependant, bien que le système d'AGV à piloter soit relativement simple, les calculs développés souffrent de l'explosion combinatoire dûe à la complexité exponentielle du problème, aussi bien pour calculer le superviseur que pour calculer le routage optimal. De plus, nous souhaitons à terme minimiser le temps nécessaire à l'accomplissement des missions et non plus la distance parcourue, ce qui nécessite l'emploi d'outils gérant les systèmes à évènements discrets temporisés. Nos récents travaux avec le logiciel UPPAL semblent encourageant en vue de résoudre ce problème avec un temps de calcul faible et en vue d'une mise en oeuvre en ligne dans un système de pilotage d'AGV semblable à celui présenté ici.

Références

[1] K. Åkesson, M. Fabian, H. Flordal, A. Vahidi. « Supremica - A Tool for Verification and Synthesis of Discrete Event Supervisors », *Proceedings of the 11th Mediterranean Conference on Control and Automation*, Rhodos, Greece, 2003.

[2] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, 1999.

[3] S.L. Chung, S. Lafortune, F. Lin. « Limited lookahead policies in supervisory control of discrete event systems », *IEEE Transactions on Automatic Control*, vol. 37(12), 1992.

[4] T. Ganesharajah, N.G. Hall and C. Sriskandarajah. « Design and operational issues in AGV-served manufacturing systems », *Annals of Operations Research*, vol. 76, 1998.

[5] M. Gondran and M. Minoux. *Graphes et algorithmes*, Édition Eyrolles, 1985.

[6] L. Grigorov and K. Rudie. « Near-optimal online control of dynamic discrete-event systems, » *Discrete Event Dynamic Systems*, vol. 16(4), pp. 419–449, 2006.

[7] R. Kumar and V. Garg. « Optimal supervisory control of discrete event dynamical systems, » *SIAM Journal on Control and Optimization*, vol. 33(2), pp. 419–439, 1995.

[8] S. Maza and P. Castagna. « A performance-based structural policy for conflict-free routing of bi-directional automated guided vehicles, » *Computers in Industry*, vol. 56(7), pp. 719–733, 2005.

[9] S. Maza and P. Castagna. « Conflict-free AGV routing in bidirectional network », *IEEE International Conference on Emerging Technologies and Factory Automation*, vol. 2, pp. 761–764, 2001.

[10] S. Maza and P. Castagna. « Robust conflict-free routing of bidirectional automated guided vehicles » , *IEEE International Conference on Systems, Man and Cybernetics*, vol. 7, 2002.

[11] P.J. Ramadge and W.M. Wonham. « Modular feedback logic for discrete event systems. », *SIAM Journal of Control and Optimization*, vol. 25(5), pp. 1202-1218, 1987.

[12] P.J. Ramadge and W.M. Wonham. « Supervisory control of a class of discrete event systems », *SIAM Journal of Control and Optimization*, vol. 25(1), pp. 206-230, 1987.

[13] P.J. Ramadge and W.M. Wonham. « The control of discrete event systems », *Proceedings of the IEEE*, vol. 77(1), pp. 81-98, 1989.

[14] S.A.Reveliotis. « Conflict resolution in AGV systems », *IIE Transactions*, vol. 32(7), 2000.