PROTOTYPAGE RAPIDE D'APPLICATIONS TEMPS-RÉEL POUR VÉHICULE SEMI-AUTONOME

Rémy KOCIK

INRIA Rocquencourt
Domaine de Voluceau BP 105
78153 Le Chesnay CEDEX - France
Tel : 33 1 39 63 51 75 - Fax : 33 1 39 63 54 91 - E-mail : remy.kocik@inria.fr

RÉSUMÉ - Nous avons conçu un nouveau véhicule électrique semi-autonome (CyCab), contrôlé par un système distribué architecturé autour d'un bus CAN. Il possède des modes de fonctionnement spécifiques tels que la conduite automatique ou sécurisée. Cet article montre comment la méthodologie A^3 nous a permis de réduire considérablement les coûts (matériel et développement) de ce prototype.

ABSTRACT - *We have designed a new Semi-Autonomous Electrical Vehicle (Cycab) , controlled by a distributed embedded computer based on a CAN bus. The Cycab owns new features such as secured or automatic driving. This paper shows how the A^3 methodology allowed us to reduce development and material costs.*

1 INTRODUCTION

Les progrès réalisés dans le domaine de la microélectronique permettent de construire des microprocesseurs de plus en plus puissants. Les architectures temps-réel, qui utilisent largement ces composants, bénéficient de cette montée en puissance.

De tels progrès incitent et autorisent la réalisation d'applications temps-réel performantes capables d'accomplir des opérations de complexité croissante.

Une application temps-réel est composée d'un *système temps-réel* (un *logiciel* implanté sur une *architecture matérielle*) et d'un *environnement physique* contrôlé par le système temps-réel. Les changements survenant dans l'environnement sont des événements (stimuli) perçus par le système temps-réel grâce aux *capteurs.* Ces stimuli déclenchent les calculs qui produisent des réactions agissant sur l'environnement grâce aux *actionneurs.*

Pour être performant un système temps-réel doit avoir une bonne perception de l'environnement. C'est pourquoi les systèmes actuels intègrent des capteurs de plus en plus sophistiqués tels que des caméras, des radars, des capteurs ultrasonores…En plus de ces capteurs extéroceptifs, l'architecture matérielle doit posséder des capteurs proprioceptifs (codeurs optiques, accéléromètres, jauges de contraintes, butées électriques…) afin de fournir au logiciel les informations sur l'état du système et d'assurer le feedback dans les asservissements de commande des actionneurs. Pour exploiter efficacement toutes ces informations, le système informatique doit effectuer des calculs bas niveau afin d'extraire les informations utiles des capteurs et réaliser le contrôle des actionneurs. Tout ceci peut impliquer beaucoup de calculs, (c'est le cas par exemple du traitement des images provenant d'une caméra), parfois à des cadences élevées (déterminer une vitesse à partir de positions fournies par un codeur, asservissement du courant dans un moteur…). Les lois de contrôle-commande du système peuvent être implantées à l'aide de techniques de fusion de données, de machines à états finis, de traitement du signal. Pour assurer un comportement autonome du système, il peut être nécessaire d'intégrer aussi de la prise de décision et de la planification, à l'aide de réseaux de neurones, de systèmes experts et de bases de données. Finalement, l'utilisateur doit communiquer avec le système temps-réel à travers une interface le plus souvent graphique. De plus, d'une part, le système doit être en perpétuelle intéraction avec son environnement, et d'autre part, les réactions qu'il produit doivent être calculées en

un temps borné définissant une caractéristique importante, le temps de réponse (*latence*)[HP85].

Afin de satisfaire toutes ces contraintes (volume de calcul, temps de réponse faible et cadences élevées) aussi bien que pour prendre en compte la nature distribuée des ressources (capteurs, actionneurs, mémoire) inhérente à ces systèmes, l'utilisation de calculateurs distribués est souvent nécessaire.

Des DSPs, des microcontrôleurs et des composants spécifiques tels que des ASICs sont utilisés pour réaliser le traitement des signaux et des images et le contrôle bas niveau, alors que des processeurs RISC et CISC réalisent les opérations haut niveau.

On le voit, malgré l'utilisation de composants de plus en plus puissants, les applications temps-réel restent souvent complexes tant au niveau matériel que logiciel et leur implantation génère des programmes difficiles à tester et débogger. Ceci explique le succès des nouveaux outils dédiés à la programmation de ces systèmes. Le rôle de ces outils est d'aider le concepteur à spécifier et vérifier rapidement, et générer automatiquement des exécutifs conformes à la spécification[GVNG94][SECK92].

Nous nous intéressons ici à la conception d'applications temps-réel de type "grand public" tels que les applications automobiles. En plus des contraintes décrites précédemment, ces systèmes sont soumis à des contraintes sévères de coûts (coût matériel, encombrement, consommation ...).

2 PROTOTYPAGE A L'AIDE DE LA METHODOLOGIE A^3

De nos jours, les automobiles intègrent des systèmes temps-réel capables d'assurer le confort (suspensions actives, conduite assistée, boîte automatique), la sécurité (freinage abs, anti-collision) et d'améliorer les performances du véhicule (réduction de la pollution et de la consommation d'essence). Ainsi, l'utilisation de nombreux capteurs "éparpillés" dans le véhicule rend les architectures complexes, alors que les contraintes de coût sont plus que jamais d'actualité[IND].

La méthodologie A^3 (Adéquation Algorithme Architecture) a été conçue pour répondre à ces problèmes de conception. Elle permet une implantation optimisée des algorithmes spécifiés et vérifiés à l'aide des langages Synchrones[Sor94]. Elle s'appuie sur un formalisme de graphes. L'algorithme est décrit sous la forme d'un graphe flot de données (sémantique du langage Synchrone SIGNAL)[LLGL91], itération infinie d'un motif qui est un graphe de dépendance de données entre opérations. L'architecture matérielle est décrite sous la forme d'un hyper-graphe non orienté dont les sommets sont des processeurs et les hyper-arcs des liaisons de communications bidirectionnelles inter-processeurs. La méthodologie permet une distribution et un ordonnancement des opérations de l'algorithme sur une architecture distribuée en respectant les contraintes temps-réel tout en minimisant les ressources matérielles. La distribution et l'ordonnancement consistent à transformer le graphe de l'algorithme en fonction du graphe de l'architecture [LS93]. Cette opération revient à réduire le parallélisme potentiel (intrinsèque) de l'algorithme au parallélisme disponible de l'architecture matérielle. Parmi toutes les distributions et ordonnancements possibles de l'algorithme sur l'architecture une heuristique cherche une solution optimale (probablement sous-optimale car le problème est NP complet) selon un critère de *latence* ou de *cadence* .

Cette méthodologie est supportée par le logiciel d'aide à l'implantation SynDEx. L'utilisateur décrit le graphe de l'algorithme grâce à une interface graphique (ou l'obtient comme résultat de la compilation d'un programme écrit en langage Synchrone), puis il décrit celui de l'architecture matérielle. Il peut alors demander à SynDEx de réaliser l'Adéquation. Une prédiction du comportement montrant la distribution et l'ordonnancement des opérations de l'algorithme sur l'architecture matérielle peut être alors affichée à l'écran. L'utilisateur peut aussi vérifier si l'application satisfait les contraintes temps-réel.

Si c'est le cas, il va chercher à optimiser les ressources matérielles en minimisant le nombre de composants ou en utilisant des composants moins puissants et moins onéreux.

Si au contraire, l'application ne satisfait pas les contraintes, l'algorithme est remis en cause. Bien souvent, le choix d'une granularité plus fine dans le découpage de celui-ci en opérations permet une meilleur distribution. Si ce n'est pas suffisant, il peut être nécessaire d'ajouter d'autres composants à l'architecture, ou d'utiliser des composants plus performants.

Lorsque l'utilisateur est satisfait, il peut demander à SynDEx de générer l'exécutif sans interblocage supportant l'exécution temps-réel de l'application sur l'architecture distribuée. Il est généré sous forme de macro-code traduit au moyen d'un jeu de macros extensible et facilement portable constituant "l'exécutif générique".

Les bénéfices que l'on peut espérer d'une telle approche sont de plusieurs ordres. D'une part, elle autorise le prototypage rapide des applications. En effet, la spécification à l'aide d'un langage Synchrone permet de décrire l'algorithme de l'application

indépendamment de l'architecture matérielle. Ainsi les évolutions de l'architecture n'impliquent pas le redéveloppement du logiciel ; ces modifications peuvent être très rapidement prises en compte. De plus, la génération automatique d'un exécutif distribué sans interblocage évite les longues phases de deboggage des communications. D'autre part, l'optimisation des architectures et des exécutifs générés permet de construire des systèmes "minimaux" dont les coûts sont réduits. Enfin, la conservation des propriétés montrées avec les langages Synchrones et la génération automatique des exécutifs garantissent une sûreté de conception.

3 APPLICATION AU PROTOTYPAGE D'UN VÉHICULE SEMI-AUTONOME

Le projet Praxitèle[PBFH96] étudie le concept d'un nouveau système de transport en commun : une flotte de véhicules électriques en libre service. Le but de ce projet est de permettre à l'usager d'emprunter l'un de ces véhicules dans une station prévue à cet effet. Il peut ensuite se déplacer librement, mais doit restituer le véhicule dans l'une de ces stations.

Le Cycab (véhicule électrique semi-autonome) a été conçu dans le même esprit afin de transporter deux personnes dans des sites protégés tels que les plateaux piétonniers, les grands complexes industriels, les aéroports et les parcs d'attractions, à une vitesse maximale de 30km/h (Fig. 1)

FIG. 1: Véhicule électrique semi-autonome

Il offre de nouvelles fonctionnalités, telles que la conduite assistée et la conduite automatique. La conduite assistée repose sur l'utilisation d'un joystick et d'un écran tactile. Le joystick est connecté au système informatique embarqué qui contrôle le véhicule et procure ainsi une conduite facile et sécurisée : la vitesse peut être limitée dans les courbes et dans certaines zones ; tout le monde peut aisément conduire ce véhicule qui ne nécessite pas d'aptitude particulière de la part du conducteur. L'écran tactile permet à l'usager d'obtenir des informations de localisation, d'autonomie ...

Pour résoudre le problème de la répartition des véhicules dans les stations afin de constamment satisfaire la demande, il est possible de former des "trains de véhicules" vides avec seulement un conducteur (dans le véhicule de tête) [DP96]. D'autres modes de déplacements automatiques sont aussi étudiés tels que la télécommande radio et le déplacement guidé par balises.

Ce véhicule a été conçu avec les contraintes d'utilisation publique et de production en série : coût réduit, faibles dimensions, robustesse, maintenance facile. Toute la conception a été orientée dans ce sens depuis la mécanique jusqu'au système informatique.

3.1 ARCHITECTURE MATERIELLE

La mécanique est dérivée d'un chassis de voiture de golf électrique, déjà produit en petites séries. L'utilisation de quatre blocs moteurs de roues identiques permet de réduire les coûts et le volume (quatre petits moteurs avec des petits contrôleurs de puissance sont plus facilement intégrables qu'un seul avec un contrôleur de forte puissance). Par conséquent, l'architecture est modulaire et le véhicule est facile à manœuvrer (les quatre roues sont motrices et directrices). La direction est assurée par un vérin électrique relié mécaniquement aux roues.

Chaque bloc moteur dispose donc de son propre amplificateur de puissance, piloté par un microcontrôleur (Fig 2). Ce "nœud" intelligent est constitué de trois couches couplées entre elles. La plus basse fournit la puissance aux moteurs. La deuxième réalise les communications avec les autres nœuds et l'acquisition des capteurs. La dernière est architecturée autour d'un microcontrôleur MC68332 (microcontrôleur de la famille 68000 de Motorola, intégrant une unité *Time Processor Unit* capable de générer, par exemple, des signaux PWM et de décoder des signaux quadratures).

Chaque nœud de roue contrôle un moteur de locomotion et un moteur de frein avec tous les capteurs qui leur sont associés (codeur incrémental, capteur de température, jauge de contraintes ...). Un cinquième nœud gère le vérin de direction et le joystick.

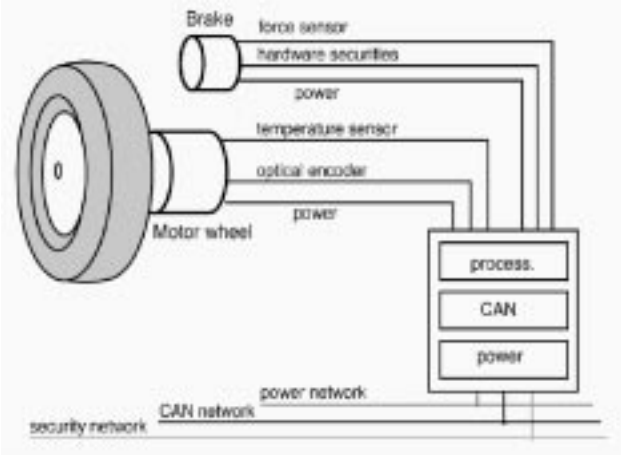

FIG. 2: architecture d'un "nœud"

Les communications entre les nœuds sont faites par un bus série CAN[Bab94]. Celui-ci a été conçu spécialement pour les applications automobiles et assure des communications fiables en environnemment perturbé à une vitesse pouvant atteindre 1Mbits/s. Les messages qu'il véhicule (trames) sont constitués d'au maximum 8 octets de données, et d'environ 8 octets de contrôle d'erreur et d'arbitration du bus.

Le réseau du véhicule est donc constitué de cinq nœuds et d'une carte PC486 qui gère l'écran et le disque dur. (Fig. 3)

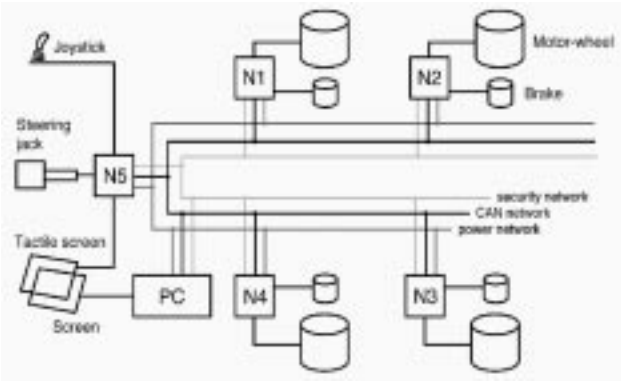

FIG. 3: Architecture du réseau

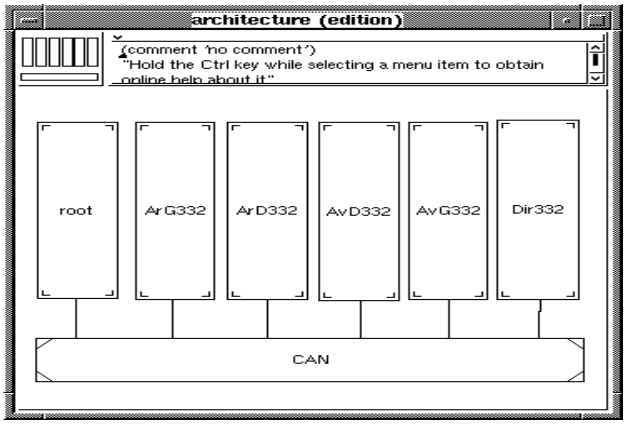

FIG. 4: Graphe Matériel

données entre les opérations. Ce graphe est composé de 3 types de sommets : les sommets d'entrée qui produisent des données, les sommets de calcul qui produisent et consomment des données et les sommets de sortie qui consomment des données.

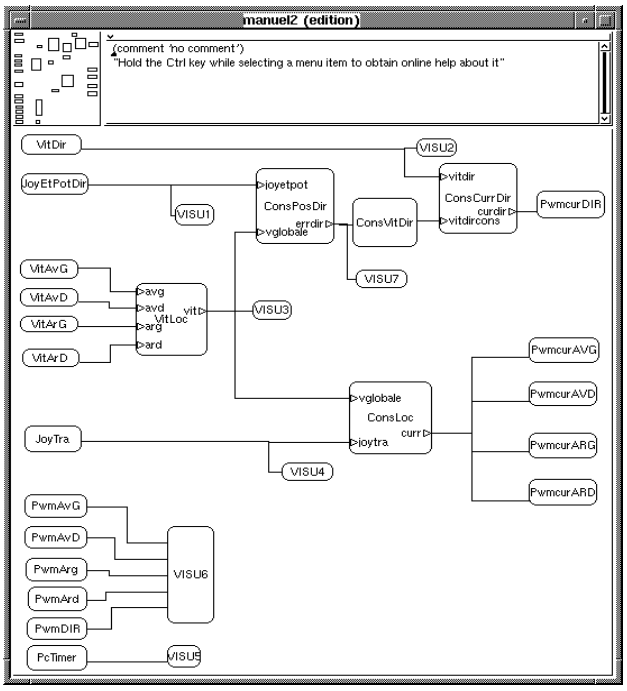

FIG. 5: Graphe logiciel

3.2 IMPLANTATION AVEC SynDEx

La figure 4 montre une copie d'écran du graphe matériel de notre application implantée à l'aide de SynDEx. Cette architecture matérielle est composée des cinq MC68332 (`AvG332`, `AvD332`, `ArG332`, `ArD332`, `Dir332`) et du 486DXII66 (`root`) tous reliés par le bus CAN.

La figure 5 représente le graphe flot de données de l'algorithme de l'application "conduite manuelle sécurisée". Les sommets sont les opérations à exécuter sur les données. Les arcs sont les dépendances de

Typiquement les sommets d'entrée sont les accès capteurs et ceux de sortie sont les accès actionneurs. Le graphe est exécuté itérativement ; les données le traversent de la gauche vers la droite. Le temps d'exécution d'une itération du graphe correspond à la période temps-réel.

La figure 6 montre l'asservissement utilisé pour la commande de direction. L'erreur de position *errdir* est calculée à partir de la position du vérin *potdir* et de la consigne du conducteur *joydir*. On remarquera que *joydir* est au préalable multiplié par un facteur

variant avec la vitesse *v* du véhicule afin de limiter l'accélération latérale. Le seuillage sur *joydir* permet d'obtenir un "zéro franc" sur la position centrale du joystick. L'erreur de position *errdir* est transformée en consigne de vitesse *vitdircons* par multiplication par un facteur K2. A partir de cette consigne *vitdircons* et de la vitesse de déplacement du vérin *vitdir* on calcule une erreur de vitesse *errvit.* Celle-ci est multipliée par un facteur K3. On obtient ainsi la consigne en courant *curdir* dans le moteur du vérin. Un asservissement en courant réalise alors la commande PWM de la tension dans le vérin. Les saturations jouent le rôle de "butées logicielles" et permettent ainsi de limiter le courant dans le vérin électrique ainsi que sa course.

Le sommet `ConsPosDir` du graphe de l'algorithme réalise le calcul de l'erreur de position *errdir*. `ConsVitDir` réalise celui de la consigne de vitesse *vitdircons* et `ConsCurDir` celui de l'erreur en courant et de la consigne en courant. `VitLoc` calcule la vitesse *v* du véhicule en moyennant la vitesse des quatre roues. En réalité, les sommets `JoyEtPotDir`, `Vitdir`, `PwmCurDir`, `VitAvG`, `VitAvD`, `VitArG`, `VitArD` ne contrôlent pas directement les capteurs et actionneurs ; ils envoient et reçoivent des données à et en provenance d'une routine d'interruption bas niveau qui effectue l'acquisition des données, le décodage quadrature des signaux en provenance des codeurs de position, le calcul de la vitesse de la roue, l'asservissement en courant des moteurs.

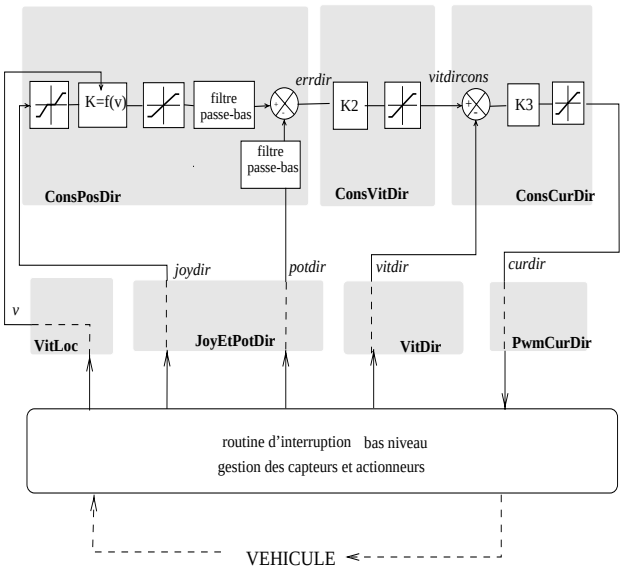

FIG. 6: Asservissement de controle latéral

Ceci permet d'exécuter les opérations de bas niveau à une période fixe (pour le calcul de vitesse des roues) plus petite que la période temps-réel (pour, par exemple, éviter les pics de courant dans les moteurs). Dans notre application, la période temps-réel est de 10ms alors que les contrôles bas niveau sont exécutés à une période de 1ms.

Pour régler certains paramètres des lois de commande, certains sommets appelés `VISU` permettent "d'espionner" les données calculées en les affichant ou en les sauvegardant sur disque. Cette facilité est très utile pendant la période de test et de deboggage.

La figure 7 montre les temps d'exécution prédits par SynDEx une fois l'Adéquation réalisée. Chaque colonne représente la séquence des opérations assignée à chaque processeur, le temps se déroulant du haut vers le bas. Les arcs représentent les communications inter-processeurs. Plus précisemment, l'origine d'un arc est la date à laquelle la donnée est prête à être envoyée, la fin de l'arc est la date à laquelle elle est disponible sur le processeur destinataire.

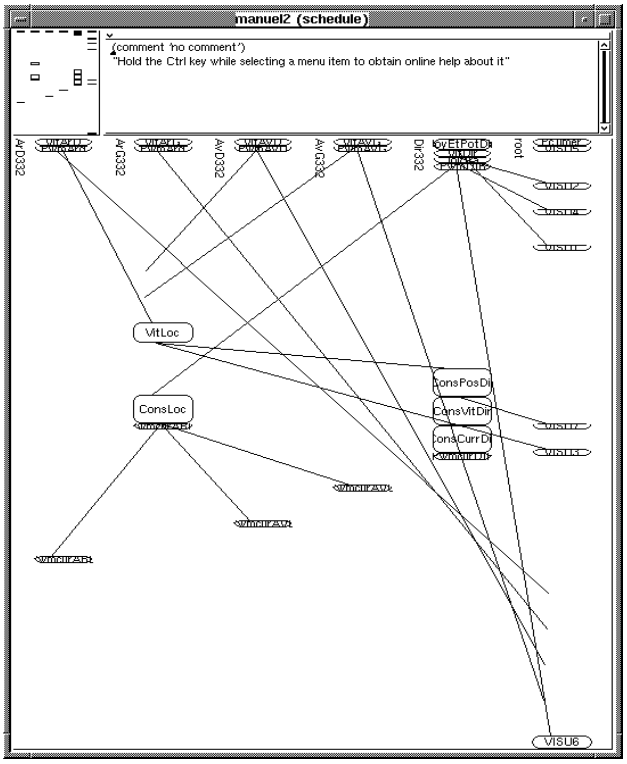

FIG. 7: Schedule

Sur chacun des processeurs le macro-code généré par SynDEx est expansé par le macro-processeur M4 (Gnu) pour produire du code assembleur. Un makefile est aussi généré. Il lance la compilation et le link, appelant Gnu GCC, et charge les exécutables sur les processeurs à travers le bus CAN. Pour nos applications, les macros du "noyau générique" ainsi que les accès capteurs et actionneurs ont été écrits en assembleur alors que les fonctions de calcul faisant intervenir des algorithmes plus complexes ont été écrites en C.

4 CONCLUSION

La méthodologie A^3 est bien adaptée au prototypage des systèmes "grand public" pour l'automobile. La conception et la validation de notre prototype de S.A.E.V avec SynDEx l'a montré.

Elle nous a permis de minimiser les coûts en réduisant les temps de développement. En effet, d'une part, la spécification de l'algorithme indépendamment de l'architecture nous a permis de prendre en compte très rapidement les modifications réalisées sur le matériel (nous avons, par exemple, regénéré notre application sur une version "dégradée" de notre véhicule composée d'un PC et seulement trois "nœuds"). D'autre part la génération automatique des exécutifs distribués, notamment la gestion des communications inter-processeurs, nous a économisé de longues phases de test et de debogage ; nous avons ainsi pu améliorer rapidement notre application en ajoutant de nouveaux modes de fonctionnement : *télécommandé* et *suivi de véhicule.*

La distribution et l'ordonnancement automatique des opérations de l'algorithme sur l'architecture et la génération automatique des exécutifs optimisés ont permi de minimiser les coûts matériels ; l'architecture est modulaire et utilise des composants courants peu onéreux. La production en série de notre prototype en sera ainsi grandement facilitée.

Enfin, la génération automatique d'exécutifs sans interblocage conformes à la spécification accroit la sureté de fonctionnement de notre véhicule.

Références

[Bab94] Michael Babb. New sensors have intelligence, will communicate. *Control Engineering*, pages 84–85, February 1994.

[DP96] Pascal Daviet and Michel Parent. Platooning technique for empty vehicles distribution in the praxitele project. In *Proceedings of the 4th IEEE Mediterranean Symposium on New Directions in Control and Automation*, Maleme, Krete, GREECE, June 1996.

[GVNG94] D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems*. Prentice-Hall, 1994.

[HP85] D. Harel and A. Pnueli. On the development of reactive systems. In Springer-Verlag, editor, *Logics and Models of Concurrent Systems*, volume 13 of *NATO ASI*, pages 477–498. New York, k. r. apt edition, 1985.

[IND] L'automobile prépare sa mutation électronique. Industries et Techniques N74.

[LLGL91] P. Leguernic, M. Leborgne, T. Gautier, and C. Lemaire. Programming real-time applications with signal. Research report, INRIA, June 1991. Research Report.

[LS93] C. Lavarenne and Y. Sorel. Performance optimization of multiprocessor real-time applications by graphs transformations. In *Proc. of the PARCO93 conference*, France, 1993.

[PBFH96] M. Parent, E. Benejam-François, and N. Hafez. Praxitèle : a new public transport with self-service electric cars. In *ISATA Congress*, Florence, Italy, June 1996.

[SECK92] Daniel Simon, Bernard Espiau, Eduardo Castillo, and Konstantinos Kapellos. Computer-aided design of a generic robot controller handling reactivity and real-time control issues. Research Report 1801, INRIA, November 1992.

[Sor94] Yves Sorel. Massively parallel computing systems with real time constraints, the "algorithm architecture adequation". In *Methodology Proc. of Massively Parallel Computing Systems Conference*, Italy, 1994.