Propositions d'architecture de controleur ouvert pour la robotique

Jean-Jacques Borrelly, Daniel Simon

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Rapports de Recherche

N° 1304

Programme 6
Robotique, Image et Vision

PROPOSITIONS D'ARCHITECTURE DE CONTRÔLEUR OUVERT POUR LA ROBOTIQUE

Jean-Jacques BORRELLY
Daniel SIMON

Octobre 1990

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

RR-1304

INRIA
INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET AUTOMATIQUE

Propositions d'Architecture de Contrôleur Ouvert pour la Robotique

Design of an Open Robot Controller Architecture

Programme 6
Robotique, Image et Vision

Jean-Jacques BORRELLY
Daniel SIMON

Table des matières

1 Introduction 4

2 Approche Fonctions de Tâches 4

3 Contrôleurs existants 7

4 Spécifications d'un Contrôleur Ouvert pour la Robotique (COR) 10

4.1 Quelques Définitions 10
4.2 Architecture générale 11
4.3 Niveau Application 13
4.3.1 Langage de Programmation des Applications 13
4.3.2 Système d'Exploitation des Applications 14
4.4 Niveau Commande 15
4.4.1 Programmation des Tâches-Module 15
4.4.2 Programmation des Tâches-Robots 16
4.5 Niveau Système 18
4.5.1 Constitution des Modules 18
4.5.2 Interfaçage Robotique 19
4.5.3 Communication Intermodule 20
4.5.4 Exécutif Temps Réel 22

5 Conclusion et Perspectives 23

Résumé

Les contrôleurs de robots actuellement commercialisés sont généralement des machines fermées, ne permettant pas à l'utilisateur d'accéder au niveau des algorithmes de commande pour réaliser de nouvelles applications, telles que des commandes de robots référencées capteurs.
Les propositions faites dans ce rapport concernent aussi bien l'architecture matérielle que logicielle: les architectures matérielles envisagées sont des architectures distribuées munies de primitives de communication temps-réel respectant les contraintes temporelles induites par la commande en boucle fermée de systèmes multirobots et multicapteurs. L'architecture logicielle et les moyens de programmation correspondants permettent à l'utilisateur d'accéder aux différents niveaux du système suivant ses besoins et sa compétence, soit les niveaux de l'application, des commandes et du système.
Le travail présenté dans ce rapport est en partie soutenu par le projet Esprit II ARMS 2637.

Abstract

Existing robot controlers are closed devices, with no way for the end user to build the control laws needed for new applications such as sensor-based control of robots.
This report deals with proposals concerning both the hardware and the software of modern robot controlers: the hardware uses a distributed architecture and real time communication primitives. The software architecture is decomposed into three layers providing an access to different programming levels: the end user level, the automatic control level and the system level.
This work is partially supported by the Esprit II Project ARMS 2637.

1 Introduction

Un contrôleur de robot doit être capable de traiter une grande diversité d'applications en robotique.
Une application est définie par 3 composantes :

- Le système robotisé
- Les actions effectuées
- Le séquencement des actions

La diversité des applications est liée non seulement au matériel mis en oeuvre (robots , capteurs ...) mais également à la diversité des actions possibles. Le contrôleur doit donc pouvoir s'adapter à différentes configurations matérielles (Interfaçage), permettre l'éxécution des actions (Commande) et leur séquencement(Langage de programmation).

Dans une première section nous rappelons l'approche de commande de robots rigides par fonctions de tâches et nous présentons les implications de cette approche sur les architectures de commande utilisées. Dans la section suivante, nous mettons en évidence les faiblesses des contrôleurs de robots actuels et nous développons dans la dernière section un certain nombre de propositions concernant l'architecture de contrôleurs aptes à la mise en oeuvre de théories modernes de commande de robots.

2 Approche Fonctions de Tâches

Cette approche concerne essentiellement les actions effectuées par les robots, exprimées en terme de régulation. Les problèmes liés à la commande des robots peuvent être classés en deux niveaux:[37]

- Le niveau "haut" consiste à spécifier la tâche à accomplir et à choisir le vecteur des signaux à réguler pour satisfaire les objectifs de l'utilisateur (choix de la fonction de tâche)
- Le niveau "bas", consiste à calculer la commande (couples moteurs) permettant de réguler le vecteur de tâche en fonction du matériel disponible.

Dans cette approche , on définit l'espace de la tâche comme étant l'ensemble des grandeurs à réguler ; cet espace peut être l'espace des coordonnées articulaires comme dans l'approche classique mais également tout autre espace représentatif de l'action que l'on désire faire effectuer au(x) robot(s). Citons par exemple l'espace des coordonnées opérationnelles, ou celui des efforts mesurés au niveau de l'effecteur.
L'élaboration d'une commande fait intervenir les processus suivants:

- Mesures physiques sur le processus (positions, vitesses, efforts ...)
- Calcul du vecteur d'erreur à réguler dans l'espace de la tâche
- Calcul du vecteur de commande (couples appliqués aux moteurs). Cette commande prend en compte les signaux capteurs par l'intermédiaire du vecteur d'erreur. La qualité de la commande obtenue dépend en partie du modèle choisi pour le processus (par exemple, du choix fait pour le calcul de la matrice d'inertie d'un robot): on peut ainsi obtenir plusieurs commandes possibles pour réaliser une fonction de tâche donnée. Le volume de calcul peut

être parfois important (modèle dynamique, Jacobiens, estimateurs ...); les calculs doivent s'effectuer en temps réel avec de fortes contraintes temporelles (la partie "feedback" de la commande est la plus contraignante du point de vue temporel)

Parallèlement au calcul de la commande, on trouvera un générateur de grandeurs de consignes (trajectoire) et un certain nombre d'observateurs générant des évènements destinés à un processus de supervision opérant le séquencement des actions.

Un langage de programmation des applications doit permettre de spécifier le séquencement des différentes actions et des paramètres associés (paramétrage des générateurs de trajectoires, évènements retournés par les observateurs).

L'organisation fonctionnelle de la structure de contrôle permettant la mise en oeuvre de ce système de commande serait la suivante :(figure 1)

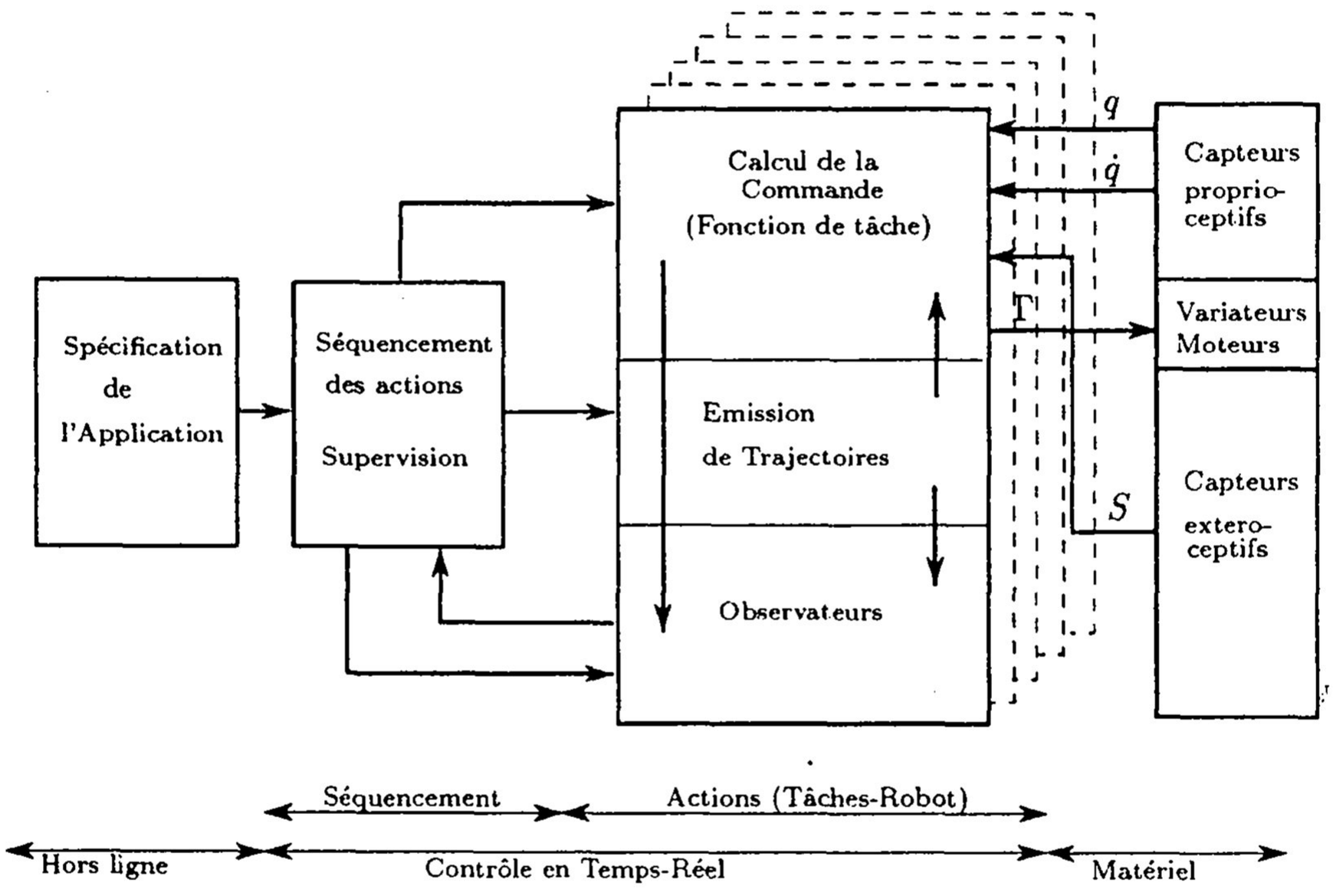

Figure 1 : Contrôleur Ouvert de Robots

La forme générale des lois de commande obtenues est la suivante [36]:

$$\Gamma = -k\hat{M}J_q^{-1}G(\mu D\hat{e} + J_q\dot{\hat{q}} + j_t) + \hat{l}$$

qui peut se réecrire plus simplement en posant $G_v = k\hat{M}J_q^{-1}G$:

$$\Gamma = -G_v(\mu D\hat{e} + J_q\dot{\hat{q}} + j_t) + \hat{l}$$

Γ est le vecteur des couples de commande imposés au robot, $\hat{M}$ est une approximation de la matrice d'inertie du manipulateur, J_q est une approximation de la matrice Jacobienne de la tâche $\frac{\partial e}{\partial q}$, j_t est une approximation de la dérivée partielle $\frac{\partial e}{\partial t}$ de la fonction de tâche par rapport au temps , $\hat{e}$ est une estimée du vecteur d'erreur, $\dot{\hat{q}}$ l'estimée du vecteur des vitesses articulaires, k, μ, G et D sont des gains et $\hat{l}$ un vecteur de précompensation permettant par exemple de compenser l'influence des forces de Coriolis ou des frottements ...

La fonction de tâche dépend des coordonnées articulaires q du manipulateur mais peut également dépendre du temps (poursuite d'un objet mobile par exemple).
Le terme $(-\mu G_v D\hat{e})$ s'interprète comme un retour en position et le terme $-G_v(J_q\hat{\dot{q}} + j_t)$ s'interprète comme un retour en vitesse.
Ces deux termes constituent donc la partie retour d'état de la commande, que l'on peut rapprocher d'une structure de commande linéaire de type régulateur Proportionnel-Dérivé. La stabilité et la robustesse de la commande dépendent fortement de ces deux termes notamment en ce qui concerne leur période d'échantillonnage (liée à la puissance de calcul mise en oeuvre) et la valeur du délai séparant la prise d'une mesure sur le processus commandé de l'instant d'émission de la commande correspondante (liée à l'organisation des calculs, par exemple en pipe-line) [8].
Les termes $\hat{M}$ et $\hat{l}$ permettent de compenser les effets de couplages dynamiques, de la gravité ou des frottements; souvent il ne sera pas nécessaire de les estimer avec une grande précision ni de les calculer à une fréquence aussi élevée que d'autres termes de la commande.

Cette approche permet de retrouver les commandes de robots classiques: suivant les choix faits par l'utilisateur pour le calcul des différents termes de la loi de commande (certains peuvent être très simplifiés ou même annulés) on pourra retrouver les notions classiques de commande cinématique, commande dynamique, commande adaptative, modes glissants ...(par ex., le choix de $\hat{M} = I$, $\hat{l} = 0$, $e = q - q_c$ et de gains constants permet d'obtenir un classique régulateur Proportionnel-Dérivé décentralisé pour la poursuite de trajectoires dans l'espace des coordonnées articulaires).
Un choix judicieux de la fonction de tâche permet également de traiter le cas des tâches redondantes (nombre de degrés de liberté du système supérieur au rang du Jacobien de la tâche) et de construire des commandes référencées capteurs [38].
Enfin, cette approche permet par une analyse préalable de certaines propriétés de la fonction de tâche (conditionnement de la tâche) d'avoir dès le départ une idée de la difficulté que l'on aura pour la réaliser et du degré de robustesse que devra présenter la commande. On peut remarquer que les notions de redondance ou de singularités sont ici liées à la tâche et non pas uniquement au(x) manipulateur(s) utilisé(s).

L'examen de la forme générale de commande appelle un certain nombre de commentaires:

- certains termes $(\hat{M}, \hat{l})$ dépendent principalement du manipulateur utilisé alors que d'autres (J_q) dépendent également de la tâche en cours, donc de l'application . Une nouvelle application, utilisant une fonction de tâche nouvelle entraînera donc l'écriture et l'implantation des programmes de commande correspondants.

- les signaux utilisés pour la construction du vecteur d'erreur $\hat{e}$ peuvent provenir des capteurs proprioceptifs des manipulateurs, mais également de capteurs extéroceptifs (forces, proximétrie, images) et avoir été mesurés et prétraités sur une ressource de calcul distante de celle où est calculée la commande .

- à une même fonction de tâche pourront correspondre plusieurs commandes différentes suivant par exemple la façon dont on calculera $\hat{M}$ ou $\hat{l}$; le choix d'une commande s'effectue suivant des critères divers (précision ou rapidité d'éxécution des trajectoires, puissance de calcul nécessaire ...) et l'ensemble des commandes possibles est à priori difficilement dénombrable.

- il s'agit de commander en boucle fermée des systèmes dynamiques; il est donc essentiel de pouvoir prédire et garantir les périodes d'échantillonnage ainsi que les délais de propagation séparant l'instant des mesures de l'instant d'émission de la commande .

Un contrôleur de robots permettant la mise en oeuvre de ces algorithmes de commande doit donc présenter les caractéristiques suivantes:

- il doit être OUVERT pour permettre aisément l'introduction de nouveaux types de commandes et l'utilisation de capteurs divers. Chaque nouvelle application peut nécessiter l'élaboration de nouvelles commandes et on ne peut donc se satisfaire d'un ensemble figé d'asservissements disponibles.

- il doit être MODULAIRE pour permettre l'évolution du système en fonction de l'application (ajustement de la puissance de calcul, du nombre de manipulateurs, ajout de nouveaux capteurs ...)

- il doit être muni d'un système de COMMUNICATION tenant compte de la nature fondamentalement REPARTIE et TEMPS REEL d'un système robotisé.

- la complexité des applications possibles rend nécessaire l'existence, à tous les niveaux où une intervention est possible, d' une Interface Homme Machine évoluée, comportant des outils d'aide à la conception et à la mise au point, en particulier:

  - au niveau de la programmation des lois de commande de moyens de vérification du respect du cahier des charges par l'implémentation choisie.
  - au niveau de l'application de moyens d'expression du flot d'actions désiré, mélange de séquentialité et de parallélisme.

3 Contrôleurs existants

La plupart des contrôleurs de robots actuels ne permettent que l'asservissement des robots dans l'espace des coordonnées articulaires, les boucles d'asservissement étant en général de type PID à gain fixe axe par axe.[14]
Ils sont généralement structurés en plusieurs niveaux:(figure 2)

- des boucles de commande d'axes dans l'espace articulaire, figées et inaccessibles

- un générateur de trajectoires dans l'espace opérationnel ou articulaire

- un langage de programmation robotique permettant de décrire les actions d'un robot (essentiellement de nature géométrique) avec parfois la possibilité de gérer la synchronisation avec d'autres composants de la cellule robotisée.

- un langage du type GRAFCET peut éventuellement être utilisé au dessus du langage robotique pour décrire le séquencement des actions de la cellule robotisée.

Ces contrôleurs souffrent d'un certain nombre de défauts majeurs en vue de la commande de systèmes complexes Multirobots et Multicapteurs:

- Effets de la Dynamique:
  De par la répartition axe par axe des boucles de commande , il n'est pas possible de prendre en compte les couplages dynamiques entre les différentes articulations du robot. Ces systèmes étant fermés (commandes parfois réalisées par des composants matériels, programmes sources inaccessibles ...) l'utilisateur n'a pas la possibilité d'intervenir sur les boucles de commande de bas niveau; ceci est particulièrement dramatique dans le cas des robots modernes de type à entraînement direct où l'effet des couplages dynamiques n'est plus masqué par les réducteurs.

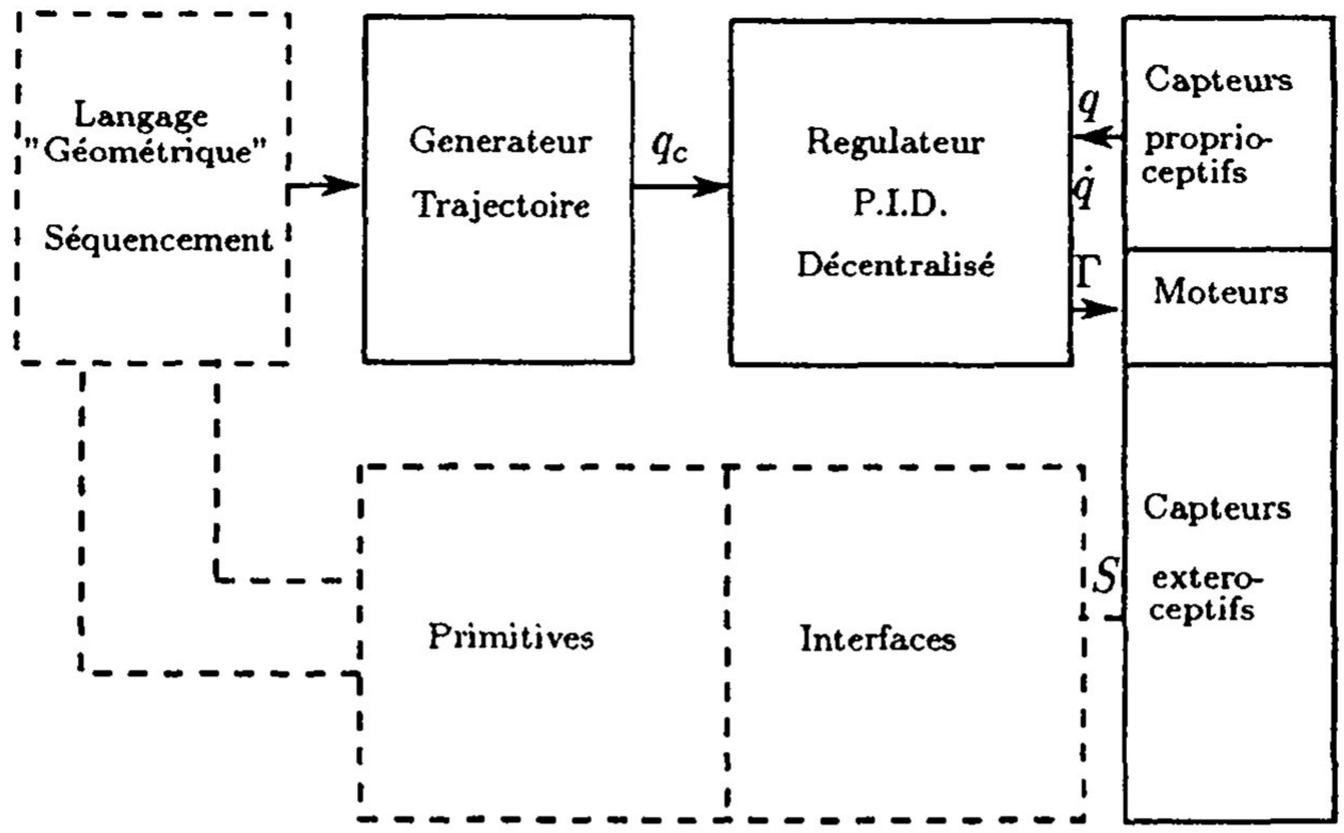

Figure 2 : Contrôleurs actuels

- Coopération Multirobots:
  Certains contrôleurs permettent la prise en compte de contraintes géométriques (anti-collision, points de synchronisation) entre deux bras, mais pas la reconstruction de la commande nécessitée par une coopération fine entre plusieurs bras , par exemple lors d'opérations d'assemblage incluant de la redondance et/ou du retour d'effort.

- Langages:
  Les langages classiquement utilisés (quand ce n'est pas tout simplement une boite à boutons) sont essentiellement des langages de manipulation de repères géométriques. Ils sont très limités dans leur possibilité d'expression du parallélisme et de manipulation du temps. Ils soit d'autre part fermés , avec un vocabulaire limité (impossibilité de prendre en compte un capteur ou une commande non prévus) et ne permettent pas à un utilisateur averti.de descendre au niveau de la commande elle même.

- Commande référencée capteurs:
  L'utilisation de capteurs, lorsqu'elle est permise, reste très élémentaire (souvent tout ou rien) et n'est possible qu'au plus haut niveau de la machine ce qui ne permet pas de séparer clairement l'aspect séquencement des actions du ou des robots (niveau application) de l'aspect cadence d'échantillonnage des commandes permettant de réaliser ces actions. En effet, l'éxécution d'une ligne de commande telle que:

```
Move Gripper with velocity V = -k*Force;
```

  permet de générer une commande en boucle fermée référencée capteur par l'intermédiaire du programme (compilé ou interprété) du niveau application, sans aucune maîtrise de la cadence d'échantillonnage de la commande obtenue donc sans possibilité de calculer la taille des gains avec des effets imprévisibles et éventuellement destructeurs (instabilité, erreur hors spécification ...).

- Communication:
  La réalisation de tâches complexes telles que des tâches d'assemblage peut nécessiter une coopération étroite entre un ou plusieurs robots et de nombreux capteurs (position, vision, proximétrie, efforts ...); il est nécessaire pour construire des commandes de bonne qualité

de pouvoir faire communiquer les différents composants du système en respectant des contraintes de type "temps réel" particulièrement sévères dans ce contexte ; ceci n'est pas possible avec les contrôleurs actuels où les possibilités de communication sont le plus souvent réduites à une ou des liaisons série peu performantes.

Il est donc parfaitement clair que ces contrôleurs ne peuvent convenir pour réaliser les commandes performantes permises par l'application des théories modernes de commande des robots.
Les limitations des contrôleurs de robots actuels ont suscité de nombreux travaux destinés à acroître leurs capacités, aussi bien dans la voie de l'amélioration de leurs performances (puissance de calcul, capacité à gérer le temps réel) que dans celle de la souplesse de programmation avec notament la fourniture de bibliothèques de fonctions spécifiques à la robotique et l'introduction d'architectures permettant l'accès au niveau de la commande de bas niveau des manipulateurs.
L'amélioration des performances des contrôleurs de robots repose d'une part sur l'utilisation des avancées de la technologie et de la densité d'intégration des circuits numériques et d'autre part sur l'utilisation d'architectures de commande généralement basées sur la répartition de la puissance de calcul [17].
Condor [31] du MIT est utilisé comme support de recherches sur la commande de systèmes polyarticulés, en particulier d'une main artificielle: le système est implémenté sur une armoire multiprocesseurs utilisant des cartes de calcul et un bus parallèle industriels, la communication intertâches peut se faire soit par mémoire double accès soit par passage de messages et l'efficacité temporelle du système est obtenue au prix d'une simplification de sa conception, en particulier l'absence de fonctionnalités multitâches.
Le système décrit dans [24] utilise une architecture multiprocesseurs de type maître -esclave pour commander un manipulateur associé à une main; la répartition des processus sur l'architecture est laissée à l'initiative du programmeur, une réduction de l'overhead peut être obtenue en utilisant la notion de "processus léger".
Les problèmes de communication rencontrés dans ces architectures temps-réel distribuées sont mis en évidence dans [16]. L'inefficacité des solutions industrielles et/ou normalisées à répondre à ces problèmes dans le domaine spécifique de la robotique est soulignée.
SIERA [22] utilise une architecture mixte avec des processeurs d'usage général fortement couplés sur un bus parallèle et d'un réseau de processeurs spécialisés faiblement couplés par des liaisons point à point. L'environnement de programmation associé donne accès à trois niveaux d'intervention possibles en fonction des centres d'interêt et des compétences du programmeur.
RIPS [10] vise essentiellement la réduction des temps de calcul par l'utilisation de processeurs spécialisés: DSP pour les calculs d'asservissements et cartes spécialisées à base de circuits CORDIC et de processeurs en tranche pour les calculs spécifiquement robotiques (modèles géométriques, cinématiques...).
Le noyau d'éxécutif temps-réel décrit dans [25] est censé garantir le temps de réponse d'un système distribué implanté sur un réseau de microVAXs. Ce noyau utilise en particulier la notion de communication par ports. Les contraintes temporelles d'éxécution des activités du système sont exprimées à l'aide de la notion d'image temporelle des tâches et l'accent est particulièrement mis sur l'aspect déterministe du système.
CHIMERA II de CMU [42] est un environnement de programmation et d'éxécution temps réel multiprocesseur développé dans le but de faciliter l'évaluation de commandes de robots référencées capteurs; on peut y remarquer en particulier la possibilité de choisir entre diverses politiques d'ordonnancement de tâches temps réel, notament des politiques d'ordonnancement dynamiques utilisant la notion d'échéances associées aux tâches. RNet [13] est un autre exemple de système temps réel distribué mettant en oeuvre des protocoles de communication par ports sur un réseau de processeurs.
Le système GEM, assurant le contrôle du robot hexapode de l'Ohio State University [39], admet la perte occasionnelle de données lors de communications de type asynchrone et prend en compte

le vieillissement des données échangées entre les tâches de commande. Là encore l'utilisation de ports de communication permet d'assembler aisément les processus élémentaires dans le schéma de commande.
Dans [40] sont analysés les différents types possibles de couplage (plus ou moins serrés) entre tâches d'un système intégré multirobots, dont la nature fondamentalement répartie est reconnue. La sémantique de quelques primitives de communication et de synchronisations correspondant aux différents cas est également donnée.
RCCL [27] est une bibliothèque de fonctions robotiques, essentiellement à caractère géométrique, permettant la programmation au niveau effecteur de systèmes multirobots en permettant de programmer des actions indépendantes, synchronisées ou coordonnées. Son évolution Kali [3] implémentée sur une machine multiprocesseurs sur bus parallèle est hiérarchisée en 5 couches logicielles, dont une couche asservissement où l'utilisateur peut choisir entre plusieurs programmes de commande.
RCTS [2] fournit un environnement de programmation modulaire destiné à faciliter le développement, la mise au point et l'intégration de capteurs dans un contrôleur: les aspects traitement de capteurs et commande sont séparés, chacun des deux sous-systèmes est subdivisé en 3 modules (haut, moyen et bas) correspondant à des niveaux de plus en plus proches de l'implémentation. Ces différents modules, implantés sur un ensemble de cartes connectées par des bus de terrains, communiquent grâce à des protocoles de communication compactés.
SPARTA [19] offre un environnement de programmation et d'éxécution sur un réseau de grappes de processeurs de traitement de signal: une réduction des délais d'utilisation des données est obtenue grâce à une soigneuse organisation des calculs et à l'utilisation de tâches de type "processus légers". L'environnement de programmation du niveau utilisateur permet d'accéder au niveau de la loi de commande du manipulateur, un changement au vol de la loi de commande est permis par la possibilité de chargement dynamique de programmes et facilité par la nature monohorloge du programme de commande. La gestion des tâches de calcul s'éxécutant sur les DSP's est centralisée au niveau d'un calculateur de type PC et seul un embryon de système d'exploitation existe au niveau des cartes de calcul.
Nous pouvons souligner la nature beaucoup plus souvent universitaire qu'industrielle des efforts entrepris pour l'amélioration des contrôleurs de robots, efforts souvent motivés par la rigidité et l'abscence d'ouverture des systèmes actuels. On peut noter dans ces différents projets le recours général aux architectures distribuées en vue d'augmenter la puissance de calcul disponible ainsi que le souci d'utiliser des moyens de communication efficaces, le développement des programmes étant fait sur une station de travail associée. La possibilité laissée à l'utilisateur d'intervenir librement au niveau de la commande est beaucoup moins générale et l'on trouve encore souvent la notion classique et limitative de "carte de commande d'axe".

4 Spécifications d'un Contrôleur Ouvert pour la Robotique (COR)

4.1 Quelques Définitions

- Système robotisé : Ensemble de ressources (robots, capteurs ...) coopérants pour la réalisation d'une Application. (exemple : 2 robots, une main gauche, un capteur de forces, une caméra, un convoyeur)
- Application: Séquence d'opérations éxécutées par un système robotisé en vue de réaliser un objectif donné (exemple : "assembler une portière de voiture en 18 s").
- Contrôleur: **Ensemble des ressources informatiques (matérielles et logicielles) permettant le contrôle en ligne du système robotisé.**

Cette définition est donc plus générale que celle habituellement acceptée pour les contrôleurs de robot (armoire de commande livrée avec un robot).

- Module : Ensemble de cartes de calcul et d'interfaces communicant par un Bus. Les différents modules sont connectés par un système de communication. (exemple: bac VME contenant des cartes processeurs, des cartes de conversion A/N, un coupleur réseau local et du logiciel système)

- Tâche-Module (T-M) : Tâche informatique (au sens classique de l'informatique temps réel) résidente dans un Module; le code de cette tâche peut être parallélisé sur plusieurs Processeurs. Ce sont les briques de base permettant de construire les commandes. (exemple: calcul d'une matrice d'inertie, d'une matrice Jacobienne, contrôle du séquencement des actions ...)

- Tâche-Robot (T-R): Correspond à l'implémentation d'une loi de commande associée à une fonction de tâche; elle est obtenue par la connexion de plusieurs Tâches-Modules et peut donc être répartie sur plusieurs Modules. Elle est généralement constituée de tâches de commande, de génération de trajectoire et d'observation.

- Génération de Trajectoires: Dans ce document, la notion de génération de trajectoire recouvre uniquement **l'émission en ligne** de consignes vers la commande, par exemple par lecture d'un fichier de points généré hors ligne; la génération des consignes peut être implicite, par exemple dans le cas d'une commande purement référencée capteurs.

- Automate d'Application (AA): Tâche-Module particulière gérant le séquencement dans le temps des Tâches-Robots pour réaliser l'Application.

4.2 Architecture générale

L'architecture proposée est organisée en 3 niveaux, tous accessibles par des utilisateurs de compétence différente (figure 3).

- Niveau application:
  Ce niveau concerne l'utilisateur du système robotisé . Il contient les outils permettant de :
  - spécifier l'application, c'est à dire de décrire les différentes étapes de l'application ainsi que leur entrelacement temporel
  - exploiter les résultats de la spécification pour produire et charger les programmes d'applications
  - superviser en ligne l'éxécution des programmes d'applications

- Niveau commande: Ce niveau est accessible à l'automaticien chargé de générer les lois de commande et de réaliser les Tâches-Robots. Les outils accessibles permettent de :
  - décrire l'ensemble des Tâches-Modules nécessaires à l'élaboration de la commande
  - décrire l'interconnexion de ces T-M
  - définir les contraintes temporelles (périodes, délais) permettant d'obtenir une qualité de commande spécifiée
  - vérifier la cohérence et l'efficacité de la commande obtenue sur l'architecture matérielle

- Niveau système : Ce niveau est accessible au constructeur des différents éléments constituant le système et doit permettre de :

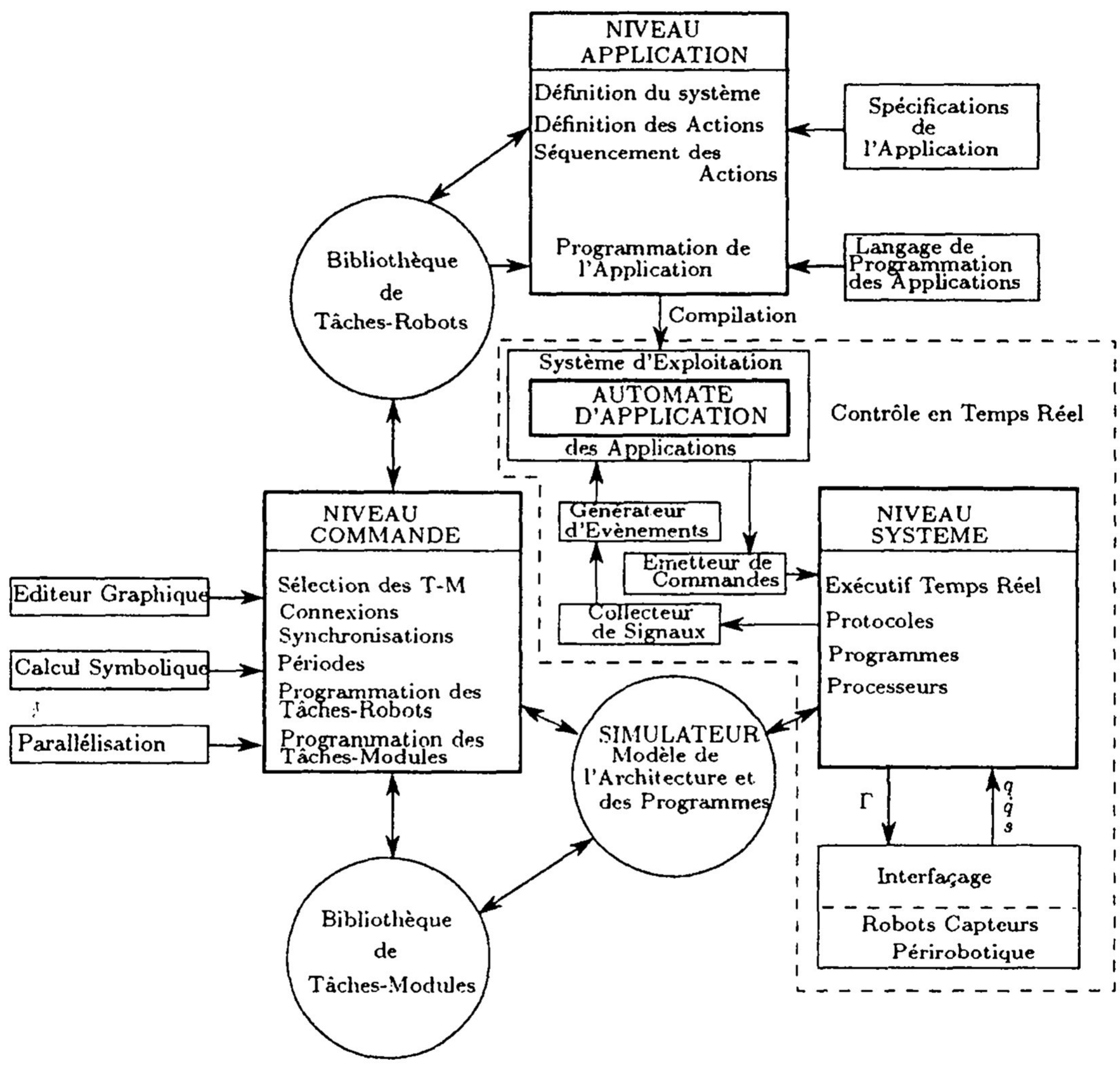

Figure 3 : Architecture générale

- interfacer le matériel
- définir la machine d'éxécution en fonction des contraintes de temps d'éxécution requises par le niveau commande
- répartir le code obtenu sur les ressources de calcul
- gérer l'éxécution de tâches réparties sur un réseau de machines multiprocesseurs

4.3 Niveau Application

Ce niveau est le plus externe du contrôleur, accessible à l'utilisateur du système robotisé. Il se compose de deux sous-systèmes: le langage de programmation des applications (LPA) et son compilateur et le système d'exploitation du niveau Application (SEA).

4.3.1 Langage de Programmation des Applications

Une application robotique , par exemple la réalisation d'une tâche d'assemblage en utilisant successivement un capteur de vision puis un proximètre et enfin un capteur d'efforts, peut être considérée comme la succession dans le temps de l'éxécution d'un certain nombre d'étapes dont chacune est associée à une fonction de tâche et dont l'activation et la terminaison dépendent d'évènements internes ou externes au système de commande.

Pour chaque étape de l'application, l'utilisateur doit pouvoir:

- choisir dans une bibliothèque une Tâche-Robot en fonction de:
  - la fonctionnalité recherchée (par ex., aller d'un point à un autre en ligne droite dans l'espace opérationnel)
  - d' indications sur les caractéristiques de l'algorithme de commande utilisé (par exemple, performances observées sur des trajectoires test ...)
  - la puissance de calcul nécessaire ou disponible
- spécifier des paramètres de la trajectoire désirée dans l'espace de la tâche (profil de vitesse, configurations de départ et d'arrivée, géométrie de la trajectoire ...)
- définir la liste des évènements ou des mesures significatives pour la poursuite de l'application (choix des observateurs)
- spécifier les actions à effectuer sur réception de ces évènements

Les objets manipulés à ce niveau sont donc essentiellement des Tâches-Robots paramétrables et des signaux.
Avec cette structuration du logiciel, l'aspect temporel lié à l'échantillonnage des boucles de commande, en particulier celles utilisant des capteurs extéroceptifs, est contenu uniquement dans la définition et la réalisation des Tâches-Robots et n'interfère pas avec l'aspect temporel lié au séquencement des actions, défini au niveau du langage d'application.
Les entrées du LPA sont donc :

- la description des différentes actions de l'application et de leur enchaînement, obtenue après analyse de l'application par exemple sur une station de travail "hors-ligne"
- une bibliothèque des Tâches-Robots disponibles (construite au niveau commande)

Les sorties du compilateur du LPA seront:

- le programme complet de l'application (liste des T-M et des connexions utilisées par chaque T-R, implantation des différents programmes dans les modules physiques)
- l'Automate d'Application (Tâche-Module spécifique de l'application) gérant l'évolution en temps réel de l'application ; les entrées de ce superviseur sont les évènements et mesures prélevés sur le processus commandé, ses sorties sont les ordres d'activation/désactivation des tâches-modules et des connexions .

Le langage de spécification de l'application doit permettre:

- d'exprimer sans ambiguité le parallélisme sous-jacent au système robotique (plusieurs lois de commande peuvent être actives simultanément lorsque plusieurs robots coopèrent)
- de décrire l'évolution temporelle du système robotisé sous l'influence des différents signaux émis par le système de commande où par l'environnement
- de définir le comportement à adopter face aux situations d'exception.

Un travail de recherche est actuellement en cours au Centre de Mathématiques Appliquées de l'ENSMP afin de définir, à partir d'un exemple d'application, une méthodologie de spécification en ESTEREL d'applications robotiques [11].

Le langage temps réel synchrone ESTEREL[6] possède en effet d'intéressantes caractéristiques en faisant un candidat potentiel pour la spécification d'applications en robotique :

- C'est un langage de haut niveau, permettant de faire une construction claire et modulaire des programmes, où les modifications de spécifications sont faciles à introduire.
- Il permet de manipuler aisément des signaux (attente, émission, actions sur délais, chiens de garde possibles sur n'importe quel signal).
- Il dispose d'une instruction de parallélisme (||) et de deux instructions de sortie de bloc permettant de construire des handlers d'exception (watching et trap).
- Il produit, après compilation, un automate d'état fini séquentiel, introduisant peu d'overhead à l'éxécution.
- L'hypothèse de synchronisme sous jacente permet d'effectuer un certain nombre de preuves et de vérifications à l'aide d'outils associés [41] (détection d'interblocage par exemple), les difficultés de passage du monde physique asynchrone vers le monde abstrait synchrone sont concentrées dans la couche interface de manipulation d'évènements.

Ce travail est actuellement étendu dans la direction de la programmation par objets [12].

D'autres langages ou formalismes tels que ADA, LIPS[14] , ESTELLE[1] , ELECTRE[34] et bien d'autres sont sans doute également utilisables, l'évaluation de leurs possibilités dans ce contexte reste a faire.
Il ne semble en tout cas pas nécessaire d'utiliser à ce niveau un langage spécifiquement robotique (tel qu'un langage de niveau effecteur), la sémantique spécifiquement robotique des actions à réaliser étant présente dans la définition et dans l'interface des Tâches-Robot à mettre en oeuvre.

4.3.2 Système d'Exploitation des Applications

Le SEA est constitué d'un certain nombre d'utilitaires:

- vérification de la cohérence de l'utilisation des ressources du système (par ex., pas d'éxécution simultanée de deux commandes distinctes sur le même robot, vérification du partage des ressources du système lorsque des actions sont déclarées en parallèle ...)

- interfaçage de l'Automate d'Application avec le reste du système (réception des signaux et construction des évènements à destination de l'AA et émission des ordres provenant de celui ci)
- téléchargement du programme dans les différents modules
- gestion du dialogue opérateur (Tâche-Module particulière)
- fonctions de débogage et d'aide à la mise au point (enregistrement de mesures effectuées sur le processus, surveillance de l'activité du contrôleur ...)

4.4 Niveau Commande

Ce niveau concerne un utilisateur averti (de type Automaticien/Roboticien) et lui fournit les outils nécessaires à la conception et à la programmation de Tâches-Robots.
Les Tâches-Robots sont obtenues en activant et en réalisant l'interconnexion logique d'un ensemble de Tâches-Modules (figure 4).

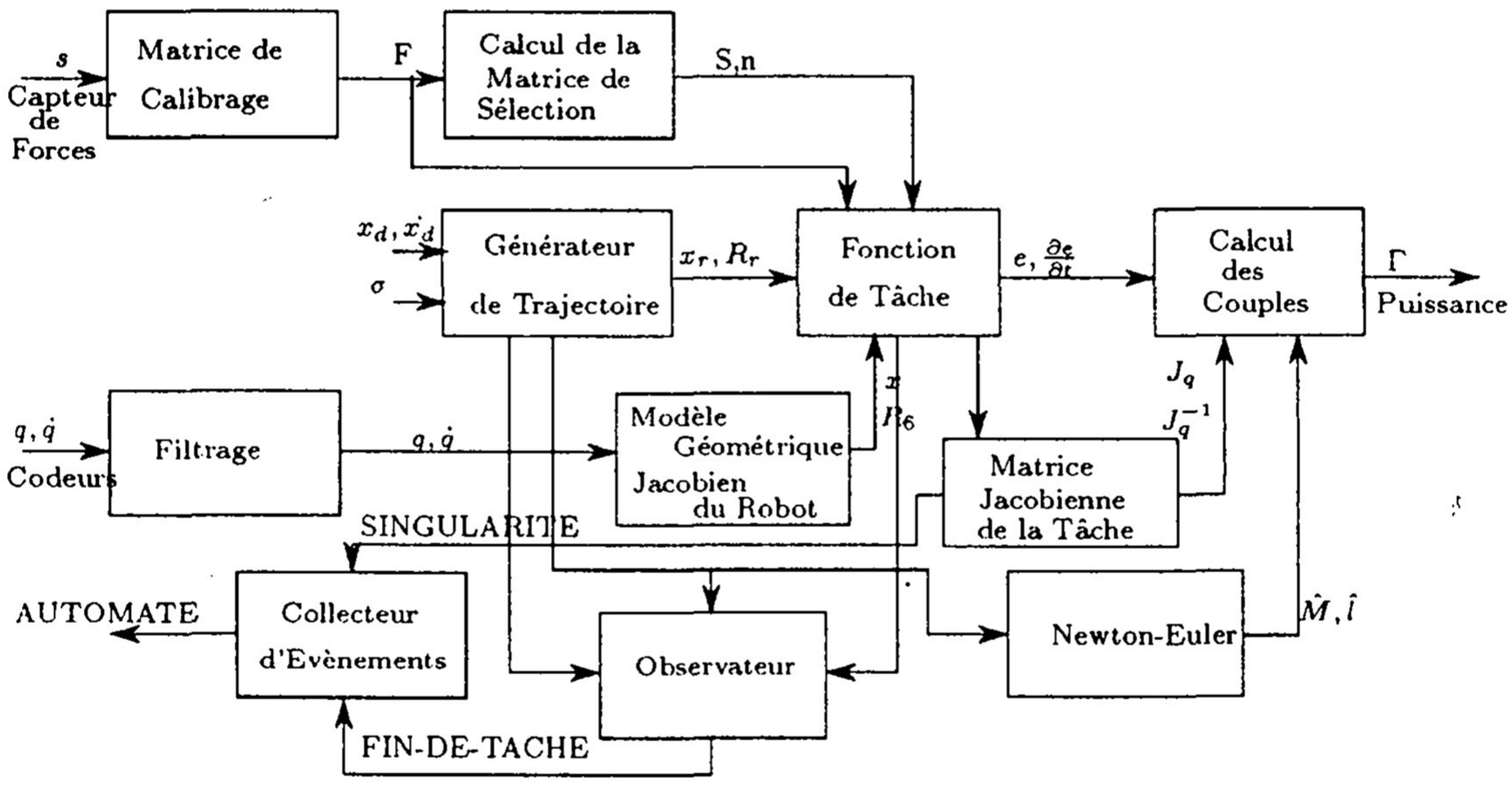

Figure 4 : une Tâche-Robot (commande hybride position/force)

L'élaboration d'une Tâche-Robot nécessite non seulement de synthétiser la loi de commande relative à une fonction de tâche donnée (travail de l'Automaticien), mais également de répartir les Tâches-Modules sur les processeurs disponibles, de définir les relations de synchronisation entre tâches, les périodes d'éxécution ...
Il peut être également nécessaire de programmer les Tâches-Modules nécessaires à une Application et non présentes en bibliothèque.

4.4.1 Programmation des Tâches-Module

Les Tâche-Modules sont des unités de programme indépendantes et pouvant communiquer entre elles par l'intermédiaire de Ports Typés.

Les Ports d'E/S sont virtuels afin de pouvoir réutiliser les tâches dans différents schémas de synchronisation sans retoucher leur code. La communication entre tâches , en particulier l'utilisation de différents modes de synchronisation (Typage des Ports) , invoque des services fournis par la couche synchronisation d'un système de communication s'appuyant sur un éxécutif Temps-réel local et des services de transport sur un bus de terrain pour les tâches distantes [28].

Elles sont écrites en langage de haut niveau (C par exemple) pour des raisons de portabilité, les calculs de commande sont effectués en représentation flottante normalisée et en unités SI .

La conception et la programmation des T-M peuvent être facilitées par l'utilisation d'un certain nombre d'outils:

- En raison du volume de calcul nécessaire à certaines fonctions et aux fortes contraintes temporelles imposées par la dynamique des systèmes commandés, le code de ces tâches pourra être parallélisé de façon à accélérer les temps de calcul et à améliorer la qualité de la commande.
  L'obtention "à la main" de code parallèle efficace est très difficile et il serait donc nécessaire de disposer d'un logiciel de parallélisation de code, paramétrable en fonction de la machine d'éxécution (type et nombre de processeurs).
  Un logiciel de parallélisation de tâches réalisé à l'IRISA [8] pourrait continuer à être développé. Ce logiciel est basé sur le découpage du programme en opérations élémentaires pour lesquelles un temps de calcul peut être défini et sur un algorithme d'ordonnancement utilisant les dépendances entre les différentes opérations. Le placement de tâches obtenu n'est pas optimal (aucun algorithme proposé ne l'est) mais son efficacité, testée sur des algorithmes de transformation de coordonnées et de calcul de modèle dynamique, semble raisonnable pour un temps d'éxécution relativement faible.

- Des outils de calcul symbolique pour faciliter l'écriture des modèles de robots, les transformateurs de coordonnées etc .... Il existe dans ce domaine un certain nombre de logiciels généraux de calcul symbolique (Maple, Macsyma, Reduce et d'autres...), ainsi que des logiciels spécialisés pour la robotique tels que SYMORO [23]).

- Des outils de calibrage des modèles géométriques ou dynamiques [33]. Ces méthodes de calibrage utilisent des logiciels d'identification spécialisés tel que Calibtool en cours de développement dans le projet Prisme de l'Inria [18] associés à des outils de métrologie fournissant les mesures sur le site réel.

4.4.2 Programmation des Tâches-Robots

- Création de Tâches-Robots :
  L'éditeur de Tâches-Robots doit permettre de décrire des objets de type Tâche-Module, Ports et Connexions et d'associer des propriétés à ces objets.
  Une première phase consiste à décrire l'ensemble des Tâches-Modules nécessaires, le réseau des connexions entre ces tâches et un certain nombre de propriétés dépendantes de l'algorithmique utilisée (périodes, synchronisations...). A ce stade, la description de la Tâche-Robot est encore indépendante de la machine cible.
  Dans une deuxième phase, on ajoute un certain nombre de propriétés dépendant de l'implémentation (affectation des processeurs aux tâches...) en vue de produire des programmes éxécutables sur une machine donnée.

  Le produit de la compilation du schéma de T-R est une liste (NETLIST) dans un format à définir (pouvant être par exemple le format standard de systèmes de CAO électronique

EDIF [15]) contenant le noms des composants (T-M et Ports) utilisés et des propriétés associées.

- Propriétés associées aux T-M: nom du programme source, type de processeur utilisé, nombre de processeurs, nom des Ports locaux, Module d'affectation
- Propriétés associées aux Ports: nom du Port, type de données transportées, Entrée ou Sortie, Simple ou Multiple, nom du ou des Ports connectés, type de synchronisation (choisi parmi les 8 possibles, voir 4.5.3), type de service transport

Cette NETLIST est utilisée par le logiciel système (SEA) pour installer les Tâches-Modules et pour configurer les descripteurs de Ports.
La méthodologie de conception d'une Tâche-Robot est décrite dans [21].

- Editeur de Tâches-Robots:
  La construction d'une Tâche-Robot est essentiellement un problème d'Automatique , où les systèmes sont traditionnellement représentés sous forme de bloc-diagrammes et d'équations. On souhaite donc pouvoir programmer les tâches-robots sous cette forme à l'aide d'un éditeur graphique, permettant d'ajouter à l'aspect algorithmique les informations nécessaires à l'implémentation (affectation des Modules, typage des ports, définition des périodes et des synchronisations ...).
  L'éditeur de Tâches-Robots doit permettre de décrire des objets de type Tâche-Module, Ports et Connexions et d'associer des propriétés a ces objets (par ex., affectation d'une T-M à un Module physique, type de synchronisation utilisée sur une Connexion ...).
  Le produit de la compilation du schéma de T-R est une liste (NETLIST) contenant le noms des composants (T-M et Ports) utilisés et des propriétés associées. Le compilateur de T-R doit également vérifier la cohérence de la synchronisation (interblocages, respect des périodes ...)

  L'éditeur graphique développé à l'IRISA pour le langage SIGNAL pourrait être une Interface Homme-Machine adaptée au problème de la programmation des Tâches-Robots .
  Il s'agit d'un éditeur orienté bloc-diagramme permettant de manipuler des boîtes (contenant du code), des ports orientés et des connexions entre ces ports.
  Par rapport à la version actuelle, il faut pouvoir ajouter un certain nombre de propriétés aux objets manipulés (type des ports, période des tâches, affectation des processeurs ...).
  Le calcul d'horloge effectué par le compilateur SIGNAL [5] pourrait sans doute aider à la détection d'incohérences temporelles telles que des interblocages pouvant se produire dans un réseau de Tâches-Module connectées par des primitives de communication bloquantes.
  Cette fonctionnalité en ferait une alternative intéressante face aux nombreux logiciels d'édition graphique existant actuellement, par exemple dans le domaine de la CAO électronique ou dans celui des ateliers de Génie Logiciel.

- Simulateur:
  Il est très difficile d'estimer les performances des commandes obtenues sur ces systèmes très non-linéaires par l'analyse et il est donc nécessaire de simuler l'éxécution des commandes avant leur éxécution sur le site réel; la simulation doit prendre en compte non seulement l'aspect algorithmique du problème mais également les contraintes temporelles induites par l'implémentation sur une architecture cible.
  Un simulateur à évènements discrets contenant un modèle de l'architecture et des programmes permet de vérifier le bon fonctionnement du logiciel sur la machine cible (respect des périodes d'échantillonnage, absence d'interblocage ...); son couplage à un simulateur

en temps continu contenant les modèles de robots et de capteurs permettra une évaluation des lois de commande en tenant compte des contraintes d'implémentation.

Le simulateur est un outil mis à la disposition de l'utilisateur chevronné ou de l'Automaticien responsable de la programmation des T-R pour permettre de qualifier les programmes de commandes éxécutés sur l'architecture cible et de donner des informations sur la qualité de commande obtenue (stabilité, précision ...) intéressant l'utilisateur du système.
Le simulateur SIMPARC [9], développé par le projet Prisme de l'Inria dans le cadre du projet Esprit II ARMS répond à ces besoins: les différents objets constituants l'architecture (Processeurs et Bus) sont modélisés par des Objets programmés en C++, auxquels sont attachés un certain nombre de propriétés: l'une des propriétés attachées aux objets du type processeur est le programme devant être éxécuté sur le processeur réel, enrichi d'information conçernant la durée d'exécution de ses différentes séquences de code. Le déroulement de l'éxécution des différents programmes constituant une application est contrôlée par un moteur de simulation dirigé par les évènements pouvant prendre également en compte les évènements génerés par la partie simulation en temps continu (programme d'intégration des équations différentielles décrivant les robots et les capteurs.) (figure 5)

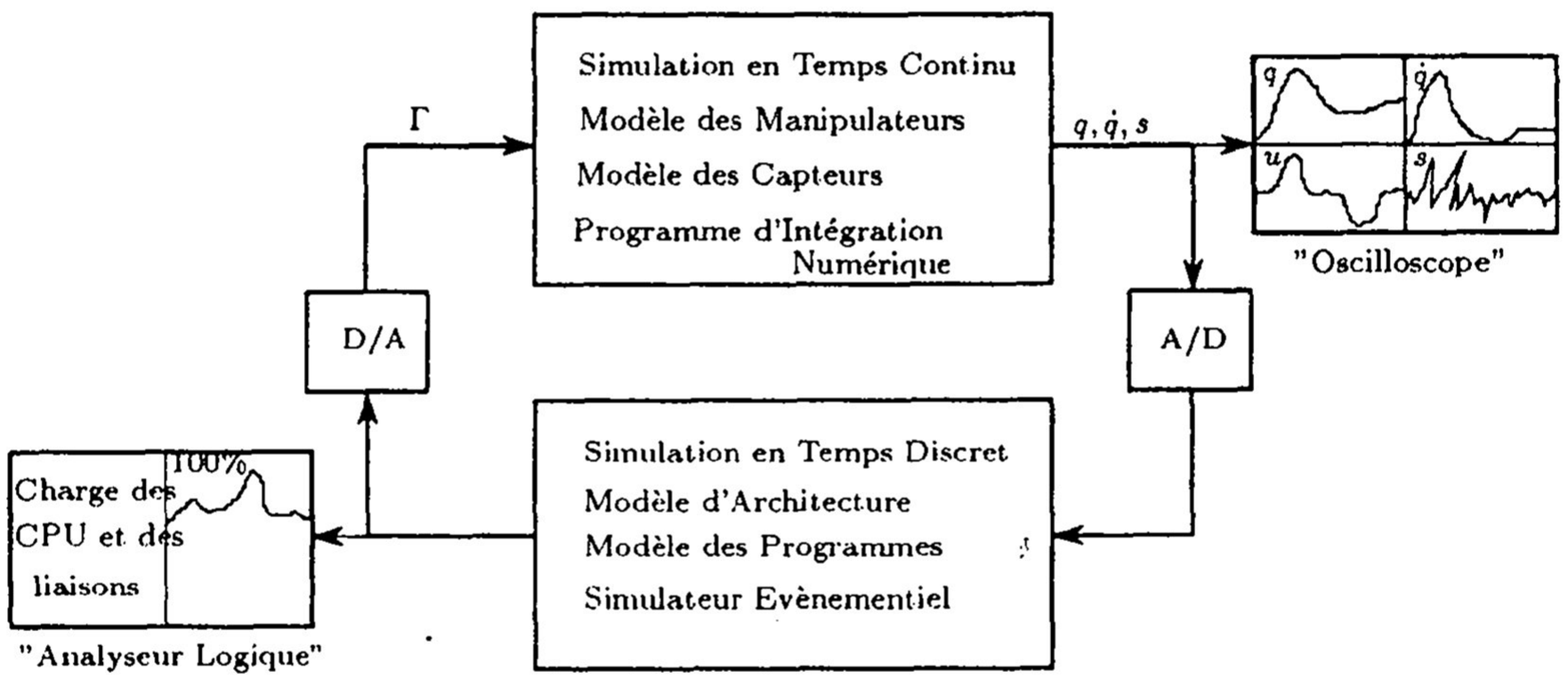

Figure 5 : Une vue générale de SIMPARC

4.5 Niveau Système

Ce niveau concerne le Matériel lui même ainsi que le logiciel système indépendant de l'application. L'intervenant à ce niveau est le concepteur de modules ou l'ingénieur système.
La structure matérielle envisagée est une structure composée d'un certain nombre de modules répartis reliés par un moyen de communication adapté aux contraintes temps réel de l'application (figure 6).

4.5.1 Constitution des Modules

Les modules dédiés aux calculs et au traitement de l'information sont constitués d'un certain nombre de cartes processeurs connectées par un Bus Local tel que les biens connus Multibus ou VME. Ces cartes peuvent être des cartes standard ou spécialisées (Processeurs de traitement de signal, circuits de calcul spécialisés , ...). On pourra également trouver sur le bus des cartes Mémoire et E/S communes et une carte d'interface avec le réseau local.

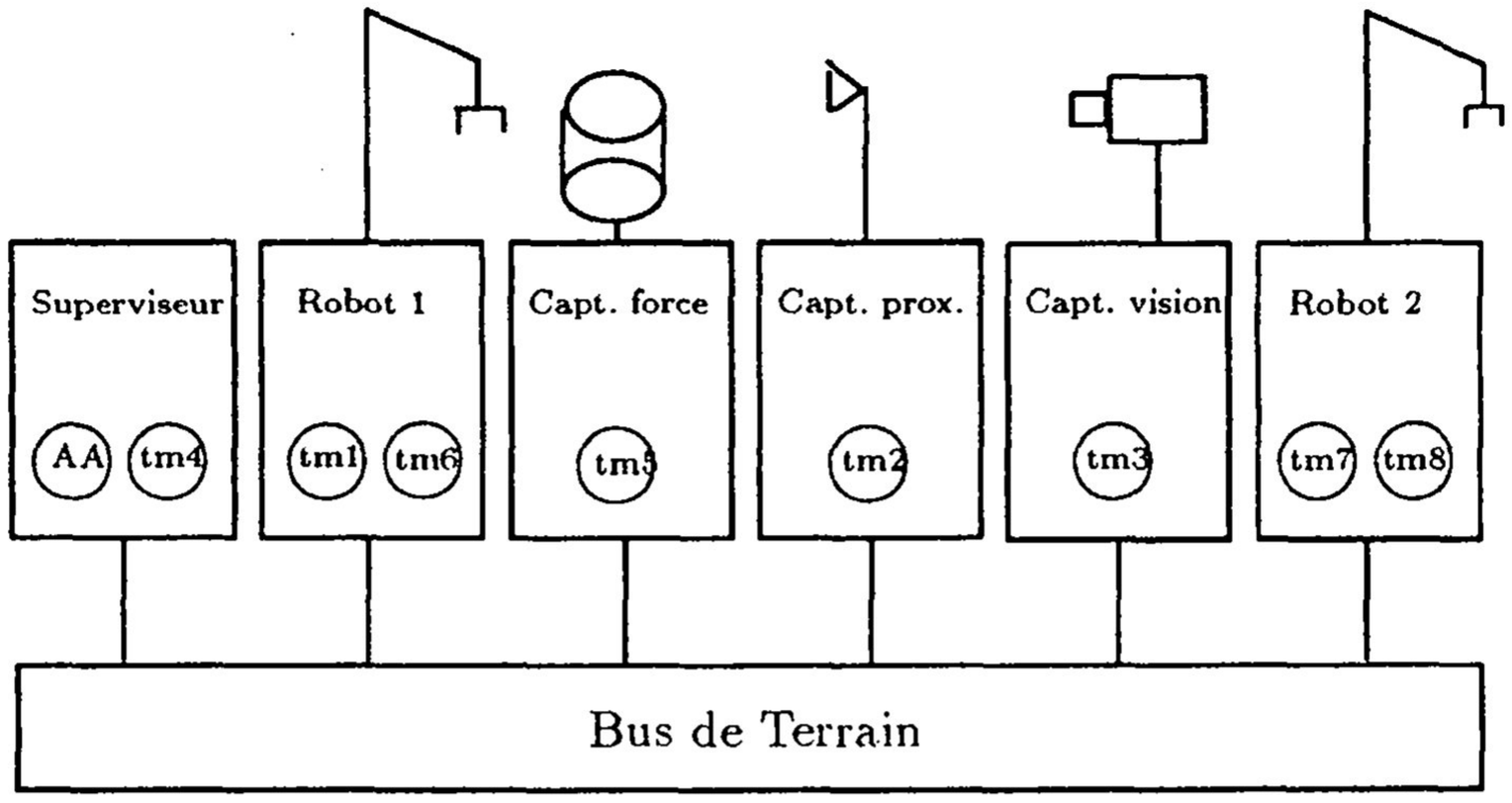

Figure 6 : Architecture matérielle

En raison de la diversité possible des modules constituant le système (modules de calcul, capteurs, caméras munis de leurs cartes de traitement ...), les bus locaux ne seront pas forcément du même type. On peut cependant recommander l'utilisation de Bus Normalisés pour lesquels de nombreuses cartes sont disponibles ...
L'utilisation d'une communication par bus parallèles nous paraît souhaitable pour des raisons de facilité de reconfiguration du système (au moment du changement de la Tâche-Robot en cours) et d'équité du coût de communication entre les tâches élémentaires, comparativement à des solutions plus spécialisées de type par exemple réseau de Transputers, sans doute plus adaptés à traiter des structures algorithmiques plus régulières comme en Traitement de Signal mais pouvant induire un coût de routage des communications important, tant à la configuration qu'à l'éxécution. Des architectures spécialisées de ce type sont cependant tout à fait envisageables en tant que "coprocesseur spécialisé" destiné au traitement à haute vitesse d'une fonction donnée. Chaque carte processeur (autre que les processeurs spécialisés) devra comporter un processeur de technologie récente (68020, 68030, 80386, 88000 ...) associé à un coprocesseur mathématique, une mémoire double accès , un mécanisme de génération d'IT interprocesseurs et en option un bus d'E/S local.

> **REMARQUE:** la communication par mémoire double accès entraîne un risque d'interblocage (cycle RMW sur 68020 par exemple). Il faut donc s'assurer que ce risque est correctement pris en compte. Il est également nécessaire de pouvoir diffuser les Interruptions interprocesseurs (un processeur doit pouvoir en interrompre plusieurs autres simultanément), fonctionnalité nécessaire pour pouvoir éxécuter du code parallélisé.

La puissance de calcul doit pouvoir être ajustée de façon à assurer des périodes d'échantillonnage inférieures à 1 msec pour le calcul du vecteur d'erreur $\hat{e}$ et du vecteur de commande Γ et de l'ordre de 5 à 10 msec pour le rafraichissement de termes non critiques ($\hat{I}, \hat{M}$ ...).

4.5.2 Interfaçage Robotique

On doit pouvoir disposer sur les bus utilisés dans les modules d'un certain nombre de cartes d'interface avec le processus physique :

- mesures de positions à partir de codeurs optiques, synchro-résolveurs, potentiomètres ...
- mesures de vitesses par génératrices tachymétriques, comptage de tops codeurs ...
- mesures d'efforts, de signaux proximétriques, d'images ... Les capteurs utilisés doivent être interfaçés directement sur le bus.
- sorties vers les amplificateurs de puissance des robots (les algorithmes de commande envisagés supposant une commande en couple, les amplificateurs doivent être asservis en courant)

Une tâche de prétraitement peut être associée aux interfaces (par exemple, calcul d'un torseur d'effort dans l'espace désiré à partir de mesures de forces) , les mesures prétraitées sont fournies dans un format adapté à la représentation choisie (par ex., flottant simple précision normalisé, unités SI)

4.5.3 Communication Intermodule

Les modules doivent pouvoir communiquer entre eux grâce a un moyen de communication respectant les contraintes de type temps réel induites par la nature "boucle fermée répartie" des commandes.

Les caractéristiques nécessaires pour les applications envisagées sont les suivantes [7]:

- délai de transmission de bout en bout de quelques centaines de μs
- débit utile de l'ordre de 10 Mbits/sec
- accès au médium décentralisé
- taille moyenne des données transmises de 24 octets
- utilisation d'échéances temporelles sur les messages
- fonctionnement en milieu perturbé, voire même agressif

L'offre industrielle actuelle ne répond pas aux contraintes rencontrées : on y trouve d'un coté des bus de terrain (FIP, Bitbus d'Intel) présentant des débits insuffisants ainsi qu'un contrôle d'accès centralisé compliquant considérablement la gestion de schémas de commande multicadencés et de l'autre des réseaux "haut de gamme" (MAP ...) où l'accumulation des couches logicielles standardisées conduisent à des temps de réponse inacceptables.

Une solution étudiée à l'INRIA [4] [29] est une architecture de communication compactée composée de 3 couches:

Bus de terrain: Au niveau le plus bas de l'architecture, on trouve un bus de terrain permettant le transfert de datagrammes entre les modules , avec un temps de réponse et un débit compatibles avec les contraintes temporelles de l'application. Ce bus est basé sur une épingle en fibre optique et son mécanisme d'accès utilise des priorités dynamiques calculées à partir de l'échéance temporelle des messages (figure 7).

Format de trame (longueur des champs en octets) :
START(1) PRIO(2) SRC-ADDR(1) DEST-ADDR(1) TPDU(4) DONNEES(24 Typ) CRC(2) STOP (1)

DAMNET

Demand Assignment Multiple access NETwork

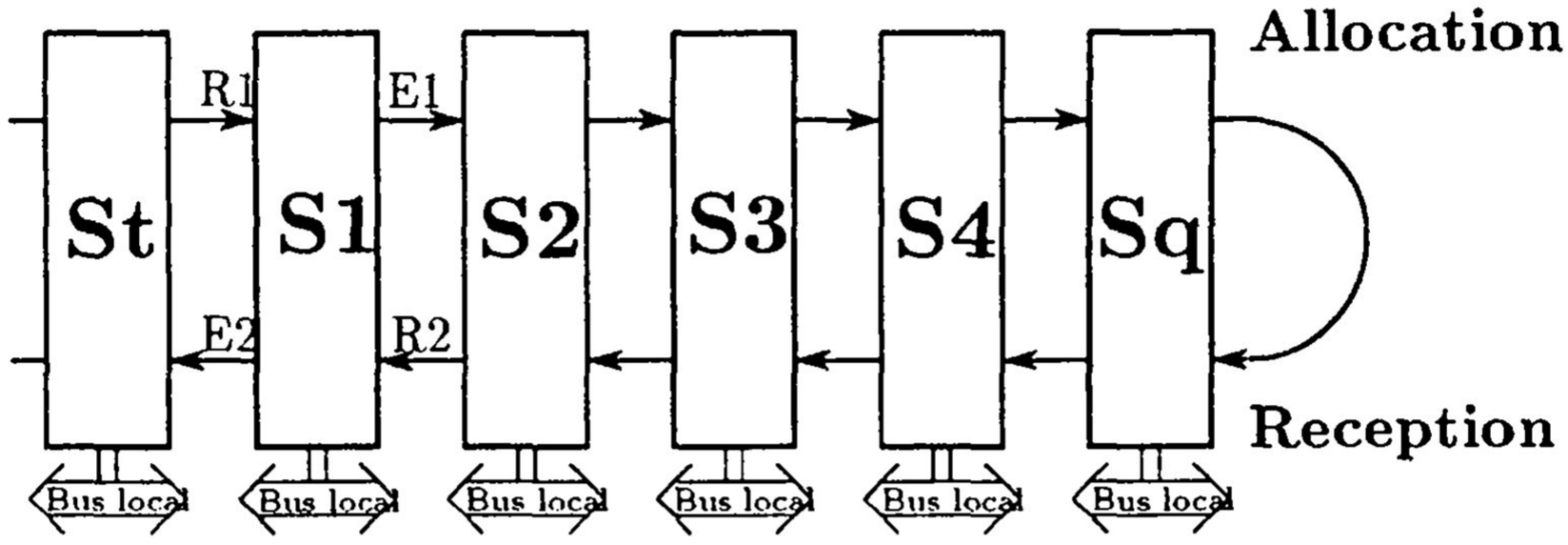

Figure 7 : Le bus de terrain Damnet

Mécanisme d'accès :

Les stations sont toujours prêtes à envoyer une trame, par défaut une trame vide de priorité nulle. A la réception d'un délimiteur de début de trame START sur son port de réception R1, chaque station compare au vol la valeur de la priorité de sa propre trame et celle de la trame incidente, et ne transmet que la trame de plus haute priorité. La trame sortant de la station de queue Sq est donc la plus prioritaire de celles présentes sur l'ensemble du bus. A la réception, les stations destinatrices de cette trame la recopient dans leur tampon ; une station peut avoir gagné localement l'accès au bus et l'avoir perdu en aval: la comparaison du champ SRC-ADDR avec sa propre adresse lui permet de s'en rendre compte et de remettre sa trame en jeu dans le cycle suivant. Celui ci est initialisé par la station de tête St sur réception d'un délimiteur de fin de trame STOP ou sur détection d'erreur.

Caractéristiques et Performances :

Codage par blocs 4B6B
Longueur maximum des messages de 256 octets
Mise en oeuvre des priorités dynamiques de messages
Synchronisation bit permanente par boucle à verrouillage de phase
Synchronisation des trames par violation du code
Extension possible par la tête ou la queue en cours de service
Possibilité de mélanger différents supports (Fibre optique,câble coaxial ...)
Mécanisme d'accès insensible aux caractéristiques physiques du bus (débit, longueur, nombre de stations)

Performances pour une modulation en bande de base à 34 Mbps avec 16 stations déployées sur 100m:
Débit utile avant codage : 22.7 Mbps
Temps de propagation du jeton virtuel : 2 μs
Durée de transfert de 24 octets : 16 μs
Transfert de 32 messages de 24 octets en moins de 500 μs

Couche transport: La couche transport offre un service de datagrammes acquittés, nécessaire pour fiabiliser la transmission de certains messages (alarmes, exceptions, communications utilisant des primitives bloquantes ...). Cette couche est résidente dans le coupleur du réseau.

Couche synchronisation: Cette couche fournit différents services de synchronisation entre les tâches du système, le choix du type de synchronisation est fait en fonction des périodes respectives des tâches communicantes. Huit mécanismes de synchronisation entre tâches suceptibles d'être utilisés dans une Tâche-Robot ont été identifiés:

- communication asynchrone
- communication synchrone (rendez-vous)
- producteur asynchrone et consommateur synchrone
- producteur asynchrone et consommateur synchronisé au futur
- producteur synchrone et consommateur asynchrone
- producteur synchronisé au futur et consommateur asynchrone
- demande/réponse immédiate
- demande/réponse différée

Ces mécanismes sont implémentés grâce à un modèle de communication unique utilisant des primitives de communication "signaler" et "attendre" agissant sur des évènements privés répartis pouvant être de type fugace, mémorisé ou figé appartenant aux ports de communication attachés aux Tâches-Module. [30]

Les protocoles des trois niveaux ont été programmés en ESTEREL et validés soit par simulation soit à l'aide du système de preuve AUTO[26], la conception matérielle des coupleurs est en cours.

4.5.4 Exécutif Temps Réel

Un noyau d'Exécutif temps réel multiprocesseur est présent dans chacun des modules.
Dans tous les cas, en plus des primitives habituelles (évènements, sémaphores, ...) il doit exister une couche communication gérant les Ports Typés associés aux tâches.
Plusieurs solutions sont possibles:

- Un éxécutif réellement multiprocesseur (un processeur Maître et N-1 processeurs esclaves); cette solution permet de gérer des changements de contexte simultanés sur les esclaves et permet donc d'éxécuter des programmes parallèles. Il semble ne pas exister d'offre industrielle correspondante.

- Un éxécutif classique multitâches par processeur, associé à une couche de communication interprocesseurs et à des outils de mise au point. Cette solution ne permet pas de gérer des programmes parallèles et peut poser des problèmes au niveau du partage de ressources critiques telles que l'interface réseau. Par contre , la majorité de l'offre commerciale actuelle tombe dans cette catégorie (OS9, pSOS+, VRTX32, VxWorks et bien d'autres ...), avec une vision généralement Unixienne de l'Interface Homme-Machine.

- Un système d'exploitation temps-réel conçu d'emblée pour être distribué, tel que Mach [32] ou Chorus [35]. En dehors des primitives classiques de manipulation de tâches temps-réel, ces systèmes intègrent dans leur noyau même des outils de communication inter-processus

(IPC). Le système Chorus utilise notamment une notion de communication par portes proche de celle décrite dans [28], la possibilité de grouper ou de faire migrer des portes entre processus pouvant être utilisable lors des commutations de Tâches-Robots.

Il est évidement souhaitable que cet éxécutif soit le plus possible portable, donc écrit au maximum dans un langage de haut niveau courant.

5 Conclusion et Perspectives

Les propositions d'architecture décrites dans ce rapport sont basées sur les travaux entrepris depuis plusieurs années dans le projet Robotique de l'IRISA et dans le projet PRISME de l'INRIA-Sophia dans le domaine de la conception et de la mise en oeuvre de la commande de robots.
La base théorique est fournie par la théorie des fonctions de tâche, théorie générale et unificatrice englobant la plupart des schémas de commande de robots connus à ce jour. Cette théorie permet de plus de traiter de façon systématique les problèmes de commande difficiles liés à la réalisation de tâches redondantes et/ou référencées capteurs.
La mise en oeuvre en temps réel de ces commandes nécessite l'utilisation d'architectures matérielles puissantes, constituées d'un réseau de ressources de calcul connectées par des moyens de communication adaptés et gérés par un système d'exploitation temps-réel réparti.
La structure du logiciel et des moyens de programmation correspondant fait apparaître plusieurs niveaux d'intervention possibles : il apparaît en particulier que les contraintes temporelles liées au séquencement des actions vues par l'utilisateur sont d'une nature différente de celles liées aux boucles de commande et doivent être traitées à des niveaux différents, ce qui n'apparaît pas dans les contrôleurs actuels où bien souvent le point d'entrée unique est un langage "robotique" de niveau effecteur.
L'intégration de ces idées sur site expérimental est en cours de développement sur les ressources de l'atelier robotique de l'INRIA Sophia-Antipolis, avec en particulier comme objectif la mise en oeuvre du système d'exploitation Chorus et du bus de terrain Damnet et le développement du système de conception et de mise au point de Tâches-Robot Orccad.

Bibliographie

[1] Alami R. et al: *Conception en ESTELLE du Pilotage d'une Cellule Flexible d'Assemblage*
Séminaire Franco-Brésilien sur les Systèmes Informatiques Répartis, Florianopolis, 11-14 Sept. 1989.

[2] Allard R., Mack B., Bayoumi M.: *RCTS: A flexible environment for sensor integration and control of robot systems - the distributed processing approach*
Proccedings of the NASA Conference on Space Telerobotics, G.Rodriguez and H. Seraji editors, JPL Publication 89-7, January 1989

[3] Backes P., Hayati S., Hayward V., Tso K.: *The Kali multi-arm robot programming and control environment*
Proccedings of the NASA Conference on Space Telerobotics, G.Rodriguez and H. Seraji editors, JPL Publication 89-7, January 1989

[4] Belmans P.,Borrelly J.J., Mejia M. et Simon D. *A Distributed System for Robotic Applications*
IEEE International Symposium in Intelligent Control, August 1988, Arlington (Virginia).

[5] Benveniste A. , Le Goff B. et Le Guernic P.: *Hybrid Dynamical Systems Theory and the Language SIGNAL*
Rapport de Recherche INRIA n^o 838, Avril 1988.

[6] Berry G., Couronné P. et Gonthier G.: *Programmation synchrone des systèmes réactifs: le langage ESTEREL*
Techniques et Sciences Informatiques, vol 6, n^o4, 1987

[7] Borrelly J.J. et Simon D.: *Specifications of a sublocal network for robotics*
Advanced Seminar on Real Time Local Area Networks, Bandol, Avril 1986

[8] Borrelly J.J. et Le Borgne M.: *Implémentation multiprocesseurs d'algorithmes de commande*
Techniques de la robotique Tome 1:Architectures et Commandes,Hermès,Paris,1988.

[9] Borrelly J.J.: *SIMPARC : SImulator for Multi-Processor Advanced Robot Controller*
ARMS project internal working document, April 1990.

[10] Butner S., Wang Y., Mangaser A. Jordan S.: *Design and Simulation of RIPS: an Advanced Robot Control System*
IEEE International Conference on Robotics and Automation, Philadelphia, 1988

[11] Coste-Manière E., Espiau B.: *A Synchronous Approach for Control Sequencing in Robotic Application*
IEEE Workshop on Intelligent Motion Control, Istambul, August 1990.

[12] Coste-Manière E., Espiau B.: *Modélisation objet pour la commande et la synchronisation en robotique*
Rapport de Recherche INRIA, à paraître

[13] Coulas M.F., Macewen G.H., Marquis G.: *RNet: A Hard Real-Time Distributed Programming System*
IEEE transactions on Computers, vol C-36, n^o 8, August 1987

[14] Dupourque V.: *Les contrôleurs de robots*
Collection Novotique, AFRI, 1986

[15] *Electronic Design Interchange Format Version 2 0 0*
Electronic Industries Association, American Technical Publishers Ltd, May 1987.

[16] Gauthier D., Freedman P., Carayannis G. and Malowany A.S.: *Interprocess Communication for Distributed Robotics*
IEEE Journal of Robotics and Automation, vol RA3, n^o 6, December 1987

[17] Graham J.H.: *Special Computer Architectures for Robotics: Tutorial and Survey*
IEEE Transactions on Robotics and Automation, vol 5, n^o 5, October 1989

[18] Guyot G. *Calibration Simulée d'un Bras Robot à l'aide de Capteurs de Vision*
D.E.A. de l'Université de Nice, Juin 1989

[19] Ish-Shalom J. and Kazanzides P.: *SPARTA: Multiple Signal Processors for High-Performance Robot Control*
IEEE Transactions on Robotics and Automation, vol 5, n^o 5, October 1989

[20] Joubert A.: *Description et simulation d'Algorithmes et d'Architectures distribuées*
Thèse de Doctorat de l'Université de Rennes I, Octobre 1988.

[21] Joubert A., Simon D.: *ORCCAD: towards an Open Robot Controller Computer Aided Design system*
Rapport de Recherche INRIA , à paraître

[22] Kazanzides P., Wasti H., Wolovich W.A.: *A Multiprocessor System for Real-Time Robotic Control: Design and Applications*
IEEE International Conference on Robotics and Automation, Raleigh, 1987

[23] Khalil W.: *Technique de modélisation et de commande dynamique*
Séminaire SYSCOROB: Architectures matérielle et logicielle des contrôleurs en robotique, Collection Novotique, AFRI, 1986

[24] Kim J.J., Blythe D.R., Penny D.A. and Goldenberg A.A.: *Computer Architecture and Low Level Control of the PUMA/RAL System: Work in Progress*
IEEE International Conference on Robotics and Automation, Raleigh, 1987

[25] Lee I., King R.B. and Paul R.P.: *A Predictable Real-Time Kernel for Distributed Multisensor Systems*
IEEE Computer, June 1989

[26] Lecompte V., Madelaine E.et Vergamini D.: *AUTO: Un système de vérification de processus parallèles et communicants*
Rapport Technique INRIA n^o 83 Mars 1987

[27] Lloyd J., Parker M. and McClain R.: *Extending the RCCL Programming Environment to Multiple Robots and Processors*
IEEE International Conference on Robotics and Automation, Philadelphia, 1988

[28] Mejia M.,Simon D.,Belmans P..Borrelly J.J.: *Mecanismes de synchronisation dans un systeme robotique reparti*
Seminaire Franco-Bresilien sur les systemes informatiques repartis, Florianopolis, Bresil, Sept 89

[29] Mejia M., Borrelly J.J., Simon D. *Un Réseau local temps réel pour la Robotique*
Actes du Colloque Francophone sur l'Ingénierie des Protocoles, R. Castanet et O. Rafiq éditeurs, Eyrolles 1988.

[30] Mejia M.: *Contribution à la conception d'un réseau local temps réel pour la Robotique*
Thèse de Doctorat de l'Université de Rennes I, Juin 1989.

[31] Narasimhan S., Siegel D.M., Hollerbach J.M.: *Condor: An Architecture for Controlling the Utah-MIT Dexterous Hand*
IEEE Transactions on Robotics and Automation, vol 5, n^o 5, October 1989

[32] Rashid R.F.: *Threads of a New System*
Unix Review, August 1986

[33] Roth Z., Mooring B. and Ravani B. *An Overview of Robot Calibration*
IEEE journal of robotics and automation, vol. RA-3, nn^o 5, October 1987

[34] Roux O. *Electre: Expression et Modélisation du comportement de Processus Temps-Réel Répartis Communicants*
Thèse de Doctorat de l'Université de Nantes, Décembre 1985

[35] Rozier M. et al.: *CHORUS Distributed Operating Systems*
Computing Systems Journal, vol 1, n^o 4, December 1988.

[36] Samson C.: *Une approche pour la synthèse et l'analyse de la commande des robots manipulateurs rigides*
Rapport de Recherche INRIA 669, Mai 1987.

[37] Samson C.: *Commande de robots manipulateurs rigides.Synthèse et analyse.*
Techniques de la robotique Tome 1:Architectures et Commandes,Hermès,Paris,1988.

[38] Samson C., Espiau B., Le Borgne M.: *Robot Control: the Task-function approach*
sous presse, Oxford University Press

[39] Schwan K., Bihari T., Weide B.W., Taulbee G.: *High-Performance Operating System Primitives for Robotics and Real-Time Control Systems*
ACM Transactions on Computer Systems, Vol 5, n^o 3, August 1987

[40] Shin K.G., Epstein M.E.: *Intertask Communications in an Integrated Multirobot System*
IEEE Journal on Robotics and Automation, vol RA-3, n^o 2, April 1987.

[41] De Simone R., Vergamini D.: *Aboard AUTO*
Rapport de Recherche INRIA n^o 112, Octobre 89.

[42] Stewart D.B., Schmitz D.E., Khosla P.K.: *CHIMERA II: A Real Time Multiprocessing Environment for Sensor Based Control*
IEEE International Symposium on Intelligent Control, Albany, September 1989

Imprimé en France
par
. l'Institut National de Recherche en Informatique et en Automatique .

ISSN 0249-6399