Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Architecture et protocole de communication temps-réel distribué pour un pilote automatique d'automobile

Marlair, Georges; Coemelck, Xavier

Award date:
1995

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 03. Jul. 2025

Georges Marlair & Xavier Coemelck

Architecture et protocole de communication temps-réel distribué pour un pilote automatique d'automobile

Septembre 1995

Mémoire présenté en vue de l'obtention du grade de Licencié et Maître en informatique

Résumé

L'informatisation d'un véhicule automobile peut être considérée comme un système de contrôle/commande tournant dans un environnement distribué et soumis à des contraintes temps-réel.

Après une brève analyse des différents matériels utilisés pour faire de nos voitures des I.V.H.S., nous concentrerons notre étude sur les besoins de communications inter agents indispensables à un tel système.

Profitant des mécanismes de structurations offerts par l'approche objet et en nous basant sur les outils de communications existants (mécanisme de la mémoire partagée et de l'échange de messages) mais insuffisants, nous construirons une architecture adaptée à nos besoins. Dans un premier temps, nous analyserons les outils basiques nécessaires à un tel système; ensuite, nous affinerons notre modèle par des couches additionnelles : celle des agents et une couche prenant en charge les communications dans un environnement distribué à caractère temps-réel ou non. Ce modèle d'architecture permettra de rendre transparents tous les mécanismes de communication inter agents.

Une fois cette architecture établie, nous dresserons le modèle d'un agent exécutif possédant les qualités requises pour une telle application.

Abstract

An I.V.H.S. may be seen (from the computer scientist view) as a control/command system running in a distributed environment that must meet real-time constraints.

We shall, at first, shortly analyze, the different kinds of materials used in I.V.H.S. and then focus our study on the inter agent communication means needed by such a system.

Coupling the object's method (offering interesting stucturation mechanism) and common communication tools used by many systems (the shared memory and the exchange of messages), we shall build, at first, a layer that will give us shared communication objects. Over it, we will build a layer dealing with agents' problems, and then, a third layer, that will cope with real-time constraints and the need of inter agent communication in a distributed environment. This architecture will improve the transparency of inter agent communication mechanisms.

Once this architecture has been designed, we shall drawn an executive agent's model featuring the requested quality of our application.

Remerciements

Les auteurs tiennent à remercier les personnes suivantes sans le concours desquelles cet ouvrage n'aurait certainement pas vu le jour.

Le professeur **Jean Ramaekers** (directeur de ce mémoire) pour sa confiance, ses conseils judicieux et l'attention particulière qu'il a apporté à notre travail.

Le département d'automatique du C.E.R.T. (Centre d'Etude et de Recherche de Toulouse) et son directeur monsieur **Marc Labarrère**, mais aussi, et plus particulièrement, monsieur **Jean-Jacques Henry** qui nous a introduits dans le monde merveilleux des IVHS. Ses connaissances et ses explications nous furent indispensables.

Les membres de la S.O.D.I.T. (SOciété pour le Développement de l'Innovation dans les Transports) : **Laurent Bréheret** et **Frédéric Schetini**, dont l'expérience dans le domaine des transports nous a permis d'éclaircir de nombreux concepts; et Mademoiselle **Véronique Ivansky** pour la sympathie particulière dont elle a fait preuve lors de notre séjour à Toulouse.

Nous remercions enfin monsieur **Jean-Pierre Martin** pour sa maîtrise de l'orthographe et de la stylistique et pour la patience remarquable dont il a fait preuve lors de la phase de correction du présent ouvrage.

Enfin, tous ceux qui ont participé de près ou de loin à la réalisation de ce mémoire et que nous n'avons pas cités.

1. INTRODUCTION

Nous avons voulu axer l'objet de notre étude sur la problématique des systèmes temps-réel de type embarqués, telle qu'elle peut se poser en particulier pour les transports routiers. Pour ce faire, nous avons effectué notre stage de fin d'étude au sein d'une petite société, nouvellement formée, qui s'attachait à proposer des réponses au délicat problème de congestion rencontré par la plupart de nos grandes villes. A la lecture de ce qui se faisait en la matière, le concept de « véhicule intelligent » nous est apparu comme un vecteur de recherche privilégié, tant par les pouvoirs publics que par l'activité privée liée à l'automobile.

A l'instar de ce que peuvent être les autoroutes de l'information pour les secteurs des télécommunications et des services, les voitures intelligentes nous sont présentées comme la révolution de l'automobile de papa. Aussi, nous avons tenté d'en savoir plus en la matière et avons opéré une pseudo « étude étymologique » de ce concept novateur afin d'en mieux cerner la finalité. Et celle-ci s'est imposée à nous de manière éclatante.

Nos voitures qui, par le passé, étaient symboles de liberté et d'évasion deviendront d'ici quelques décennies les composantes individuelles du transport collectif. Les super gestionnaires des mégalopoles de demain disposeront d'un parc d'automobiles particulières dotées de pilotes automatiques implémentés sur base de critères de sécurité, de respect de l'environnement et de satisfaction de la mobilité globale. Pratiquement, imaginez que votre voiture soit dépourvue d'instrument de pilotage (vollant, changement de vitesse, ...); tout au plus vous pouvez lui indiquer vocalement votre destination et le type de trajet choisi, à savoir : le plus rapide, le plus confortable ou encore le plus riche en monuments historiques. Douillettement installé, vous serez véhiculé tel un sénateur par son chauffeur. La voiture sera en communication étroite avec le centre de gestion routière qui lui indiquera à tout moment le trajet à suivre, respectant ainsi vos demandes, mais aussi une logique de satisfaction globale des besoins du réseau. Le pilotage, quant à lui, s'opérera d'une main de velours. Votre véhicule, reconnaissant le type de route empruntée, adaptera automatiquement sa vitesse et, via le centre de gestion, il connaîtra la position des autres usagers; enfin, muni d'une batterie de capteurs de tout type, il identifiera toute modification de son environnement et agira de la manière la plus adéquate.

Pour l'heure, nous n'en sommes pas encore là. Cependant, il faut se rendre à l'évidence que la tendance actuelle est bel et bien d'introduire le concept aérien de pilotage automatique au domaine de la voiture particulière. Nous en voulons pour preuve que certains véhicules de haut de gamme possèdent déjà un régulateur de vitesse intelligent capable d'adapter la vélocité du véhicule en fonction des

caractéristiques statiques et dynamiques de la voie routière utilisée, voire pourquoi pas d'un plafond de consommation.

L'informatisation d'un véhicule revient à réaliser un logiciel de contrôle/commande dans un environnement distribué soumis à des contraintes Temps-Réel. Nous sommes en effet face à de nombreux matériels, systèmes et capteurs concourant à la réalisation d'un but commun : le pilotage automatique d'un véhicule automobile. Quelles sont les caractéristiques d'un tel système ? Tant au niveau du système d'exploitation que du logiciel de contrôle/commande, nous tenterons de répondre à cette question en dressant l'architecture générale du système et en choisissant une approche adéquate pour réaliser le logiciel de contrôle/commande.

La distribution des matériels utilisés ainsi que leur haut besoin de coopération impose à notre système de disposer de mécanismes de communications preformants. En effet, les données issues des différents capteurs seront traitées par des logiciels de contrôle dispersé un peu partout dans le véhicule. De même, les ordres de commandes devront être acheminés vers différents contrôleurs (freiner, démarrer, tourner les routes, ...). La base d'un tel système nous semble donc résider dans les facilités offertes pour communiquer.

Un bref tour d'horizon de ce qui se fait en matière de mécanismes de communication nous apprendra que les outils classiques, proposés par les systèmes existant, sont insuffisants et ne répondent pas exactement à nos attentes. De même, le caractère hautement réactif de l'application envisagée nous amènera à délaisser l'approche procédurale et nous irons chercher dans les langages d'acteurs les capacités de structurations nécessaires à notre projet.

Sur base des mécanismes de communications classiques et en utilisant l'approche objet, nous tenterons de définir des agents particuliers, dotés d'une capacité de communication rendant compte des contraintes de notre environnement : l'extensibilité et l'efficacité. Les communications inter-agents doivent être transparentes au programmeur afin que celui-ci n'ait à résoudre que les problèmes relatifs au logiciel de contrôle/commande. Ces communications doivent s'établir entre des agents coopérant sur une même machine mais aussi dans un environnement distribué et ce, sans dégradation du temps de réponse (cfr. critère Temps-Réel). L'extensibilité du système doit permettre l'adjonction facile de nouveaux types de capteurs, processeurs et équipements. Un fois les objets et les protocoles de communications établis, nous disposerons d'agents capables de communiquer suivant les contraintes postulées : les agents exécutifs Temps-Réel.

Armés de ce type d'agents, les programmeurs « n'auront plus qu'à » choisir les différents agents constituant leur application et à régler les problèmes relatifs au contrôle des événements.

2. CONCEPT DE " VEHICULE INTELLIGENT "

2.1 Définition

Un Véhicule Intelligent est un véhicule doté de matériels et de systèmes de traitements de l'information afin de rencontrer des besoins de plus en plus pressants d'amélioration de l'environnement, de la sécurité et de la mobilité.

Ce concept, qui apparaît successivement en Europe avec les programmes PROMETHEUS (1986) et DRIVE I (1989), aux USA avec la naissance de l'IVHS (1990) et enfin, au Japon avec le programme SSVS initié par le MITI (1992), est le fruit de diverses volontés politiques et économiques dont voici les faits marquants.

2.1.1 Les Chocs Pétroliers

A deux reprises (1973 et 1979), la planète " économie " est secouée par une redoutable onde de choc nommée " choc pétrolier ". Le prix du brut coté sur le marché d'Amsterdam s'envole avec une augmentation de près de 200 %, la facture énergétique explose, la balance des payements chavire. Ce sont nos économies qui vacillent. Le règne de l'or noir est annoncé. L'heure est au rationnement.

Très vite, afin de réduire la consommation pétrolière, les pouvoirs publics vont prendre des mesures qui, pour être draconiennes, auront des effets immédiats. Parmi ces mesures, nous en épinglons deux des plus célèbres touchant le secteur automobile belge :

1. Limitations, annoncées temporaires, de la vitesse sur les axes routiers. Ce sera 60 km/h en ville, 90 km/h sur les nationales et 120 km/h sur autoroute.
2. En 1974, instauration des journées dominicales sans automobiles durant lesquelles nos villes retrouvèrent leur charme suranné de début du siècle.

Le caractère répétitif des chocs pétroliers imposera aussi de prendre des mesures sur le long terme. Et, partant du postulat que des congestions de circulation induisent de fortes hausses de consommation en carburant, la régulation des flux

automobiles s'imposera comme une composante de plus en plus prépondérante dans l'incessante chasse aux gaspillages.

C'est ainsi que dès 1979, les autoroutes allemandes seront le terrain d'expérimentations d'un système de routage de véhicules nommé ALI (Autofahrer Leitung Informationen System) [BRAE, 80], utilisant des boucles magnétiques pour mesurer les débits. Ce système, initié par le gouvernement allemand et développé conjointement par Volkswagen et Blaupunkt, illustre la toute nouvelle synergie qu'entretiennent pouvoirs publics, constructeurs d'automobiles et industries de l'électronique. Plus tard ALI complété par AUTO-SCOUT donnera naissance au projet ALI-SCOUT [VON, 86] testé à grande échelle en 1991 sur le site pilote de Berlin.

Outre l'adoption de système de routage, la régulation automobile peut encore s'opérer à partir d'extensions et améliorations de la voirie, par une meilleure information des usagers via des panneaux à messages variables (PMV) [WHI, 91] ou encore par des messages radio (non encore codés) sur la bande F.M.

Sous la pression des consommateurs, les constructeurs d'automobiles entament des recherches pour améliorer le rendement énergétique de leurs moteurs. De sorte que, progressivement, sur les modèles haut de gamme, les carburateurs feront place aux injections électroniques et, de fait, ce sont les technologies de traitement de l'information qui font leur entrée, par le biais du compartiment moteur, dans les véhicules.

2.1.2 Le parc automobile connaît une très forte croissance

Pays	Années	Nbre Véhicules en circulation (Millions)	Variation cumulée	Nbre Véhicules Produits (Millions)
CEE	1970	73	-	-
	1980	107	47%	-
	1990	146	100%	14

[DEHU, 94]

De 1970 à 1990, le marché européen de l'automobile aura connu une croissance soutenue. Aujourd'hui, l'Europe compte plus de 146 millions de véhicules dont la part de voitures particulières est d'environ 89%.

Le mauvais climat conjoncturel des années 1993-94 a durement frappé cet important secteur économique qui, en 1990, représentait pas moins de 7% du PNB de la Communauté Européenne ou encore, pour le Japon, 4,6 % de son PIB. Ce secteur

fournit ainsi du travail à plus de 4 millions de personnes en Europe, sans compter les activités induites telles que les assurances, les stations services, les garagistes, les centres d'écolages, etc. qui, selon toute vraisemblance, génèrent une masse d'emploi supérieure à l'activité directe. Enfin, ce secteur européen occupe le premier rang mondial devant les Etats-Unis avec respectivement 12 millions et 8 millions de voitures particulières vendues chaque année. C'est pourquoi les pouvoirs politiques se portèrent au chevet des constructeurs d'automobiles. C'est ainsi par exemple que la France, sous l'initiative de son premier ministre E. Balladur, instaura une aide gouvernementale (appellée prime Balladur) à l'achat d'un véhicule neuf. Cependant, pour l'année 1995, tous les indicateurs, notamment les taux d'intérêts et le nombre de nouvelles immatriculations, semblent annoncer le retour à la croissance de ce secteur et ce tout particulièrement pour le marché américain qui connaît une très forte croissance. Remarquons toutefois que le marché belge ne donne pas encore les signes d'une reprise durable.

D'un point de vue macroéconomique, le marché mondial, quant à lui, connaît une progression annuelle de 4 %. On estime par ailleurs que les nouveaux marchés que constituent l'Europe de l'est et l'Asie du sud-est soutiendront une forte demande, qui est respectivement estimée à prés de 3.5 millions et 2 millions de véhicules par an [EUFI, 92].

Mais, comme nous le montrerons dans les points qui suivent, cette explosion du parc automobile a pour effet une aggravation alarmante non seulement du nombre des accidents, mais aussi du niveau de congestion des voies routières et, par voie de conséquence, des effets de la pollution par les hydrocarbures. Des mesures, des réglementations et des initiatives de tout type verront le jour, afin d'éviter l'effondrement de tout un pan de l'économie mondiale. Empruntant les propos d'un prophète, nous pourrions annoncer les technologies de l'information comme étant le futur rédempteur de la voiture.

2.1.2.1 Les accidents

	Années	Nbre d'accidents	Variation cumulée	Nbre de tués	Variation cumulée	Nbre de blessés	Variation cumulée
Japon	1980	476677	-	8760	-	598719	-
	1984	518642	+9%	9262	+6%	644321	+8%
	1989	661363	+39%	11086	+20%	814832	+26%
C.E.E.	1970	-	-	67000	-	-	-
	1990	-	-	46000	-30%	-	-

[PRE, 91]

Comme l'illustre le tableau ci-dessus, le tribut en terme de vie humaine, ce que nous appellerons le coût social, devient de plus en plus moralement intolérable. Mais, il est un autre coût dont il faut aussi tenir compte, que nous qualifierons d'« économique ». Nous vivons la crise de " l'état providence ". Les trésoreries publiques sont au plus mal; il faut impérativement résorber le trou " abyssal " de la Sécurité Sociale, sous peine de la retrouver en situation de cessation de payements. Réduire le nombre de morts et de blessés, et notamment ceux issus des accidents de la route, devient dès lors une nécessité socio-économique.

L'objectif déclaré est donc de rendre plus sûre la conduite d'une automobile. Pour ce faire, il a fallu au préalable connaître les circonstances et les éléments qui ont concouru à ces accidents. Pour l'année 1993, l'INRETS propose la typologie des causes d'accidents suivante :

Fatigue, inattention.	28 %
Distances de sécurité non respectées	14 %
Vitesse excessive.	13 %
Non prise en compte des conditions météo.	8 %
Eclatement de pneumatique	6 %
Collisions avec des piétons	15 %
Chercher son chemin	5 %

[BOU, 94]

Cette étude met en exergue l'opportunité de développer des systèmes d'assistance à la conduite, capables d'aider le conducteur à limiter le nombre d'erreurs directes de conduite (erreurs humaines). Il en résultera aussi nombres d'autres améliorations qui se feront tant dans les véhicules qu'au niveau des infrastructures du réseau.

Véhicule : - Amélioration de la sécurité active, par l'adoption de système de gestion électronique des trains roulants (suspension, freins, etc.).

- Amélioration de la sécurité passive (ceintures de sécurité à prétensionneurs, apparition de zones de déformation, renforcement de la rigidité de certaines structures, barres de protection latérale, etc.).

Réseau : - Pérennité et renforcement des limitations de vitesses.

- Modification du tracé routier par l'adjonction de ronds points, de casse-vitesse et autres aménagements.

- Gestion dynamique des feux de signalisation intégrant la notion de véhicule prioritaire (cfr. PRODYN [HEN, 83]).

- Meilleure information des usagers par l'installation de panneaux à messages variables.

Parallèlement à ces dispositions que nous pourrions qualifier de « matérielles », le législateur tentera un travail de plus longue haleine : le changement de comportement des usagers de la route. Cette action s'articulera autour de la politique dite du « bâton et de la carotte ». En effet, il sera tantôt fait appel à un arsenal répressif (diminution du taux d'alcoolémie autorisé, renforcement des contrôles de vitesse, correctionnalisation des excès de vitesses, etc.), tantôt à des campagnes de sensibilisation au travers des différents média.

Et l'Europe, pour avoir été précurseur et innovatrice en la matière, connaît une régression du nombre de morts parallèlement à une augmentation du parc automobile, alors que, parallèlement, le Japon vit une tendance inverse (cfr. premier tableau page 6).

2.1.2.2 La congestion

Jusqu'à la fin des années 80, les pouvoirs publics des pays de la Communauté Européenne ont répondu à l'augmentation du parc automobile, et donc à la demande de mobilité, par l'augmentation du nombre et des capacités des voiries. Ainsi, depuis

1970, le réseau asphalté européen s'est étendu d'environ 350.000 Km dont 18.000 Km d'autoroutes [EUFI, 92]. Or, cet accroissement de l'offre a un effet directement proportionnel sur la demande. Si bien que l'on prévoit pour 2010, une augmentation de 30% du trafic passager et de 80% du trafic marchandise [RTE, 94]. D'autre part, le coût annuel de la congestion en Europe est évalué par la Communauté Européenne à près de 50 milliards d'Ecus et ce, sans compter les coûts relatifs à la santé, la sécurité et à l'environnent : les gouvernements vont radicalement changer de politique.

L'heure est à la promotion, au détriment de la voiture, des transports publics de masse. Les moyens seront :

1. Optimiser le réseau de transport public : On place les transports en commun en site propre, on développe des systèmes de régulation dynamique de l'offre en fonction de la demande, on tente d'influencer le comportement des usagers par une information fiable des temps de parcours (cfr. le programme SCOPE).
2. Décourager l'automobiliste d'utiliser son véhicule et plus particulièrement dans les grands centres urbains, où les embarras de circulation connaissent des niveaux inquiétants. On rétrécit les voiries, on étend les zones piétonnes. Ainsi, des villes comme Athènes, Rome, Milan et Bologne interdisent toute circulation pendant certains créneaux horaires. A Hanovre, aucun véhicule à moteur, hormis les transports publics, ne peut circuler dans le centre ville. Pour l'heure, la solution du « péage urbain », initié par les pays Scandinaves, semble avoir la faveur de la Communauté Européenne. Mais comme le montre le tableau ci-dessous, ces développements d'infrastructures au sein des villes demanderont d'investir dans les 15 années à venir des sommes considérables.

Cibles	Besoins d'investissements de 1991-2005
Voiries Urbaines	55 Milliards FF
Equipements de Gestion de Flux	13 Milliards FF
Routes & Voiries Nationales en site Urbain	120 Milliards FF
Coût Total (Urbain)	**188 Milliards FF**

[EUFI, 92]

Cependant l'Europe, s'attachant à désenclaver toutes les villes de plus de 250.000 habitants, et aussi à relancer l'économie européenne par une politique Keynésienne de grands travaux [DELO, 94], continuera de développer et financer le réseau autoroutier transeuropéen, soit un investissement de près de 125 milliards d'Ecus pour les dix ans à venir [RTE,94].

2.1.2.3 La pollution

Vers la fin des années 70, la plupart des pays de l'OCDE connaissent une montée en puissance des mouvements écologistes. En Allemagne, où la progression est la plus sensible, les Verts entrent en force au Bundestag. Ils réclament, entre autres choses, la réduction des émissions provenant de la combustion de carburant. Il semble en effet établi que ces émissions seraient responsables de l'effet de serre et des pluies acides affectant de manière inquiétante notamment la Forêt Noire dans la vallée de la Ruhr ou encore les résineux des pays Scandinaves. Ainsi, les automobilistes se trouvent dans le collimateur des écologistes car, comme le tableau ci-dessous nous le montre, la part des rejets polluants pour la France en 1984 due aux transports est de loin la plus importante. Ainsi, de nouvelles mesures affecteront, à nouveau, l'automobile. Elles émaneront tant du secteur public que du secteur privé.

Polluants	Rejets Annuels (Tonnes)	Part des Transports (Tonnes)	Part des Transports (Pourcentage)
CO	6 500 000	5 500 000	84,6 %
Nox	2 485 000	1 085 000	43,7 %
SO2	1 900 000	100 000	5,3 %
Particules	200 000	60 000	30 %
Total	**11 085 000**	**6 745 000**	**60,8 %**

[DEHU, 94]

Par sa directive 85-210/CEE publiée dans le J.O.C.E. en date du 3 avril 1985, la Communauté Européenne imposera pour le 1er octobre 1989, l'utilisation pour les nouveaux véhicules de l'essence sans plomb et des pots catalytiques, ce qui nécessitera la généralisation de l'injection électronique des moteurs. Parallèlement, des efforts considérables seront mis en oeuvre pour promouvoir la recherche et le développement dans les domaines de gestion du trafic et d'aide à la navigation. Ce sera, en 1986, le projet cadre PROMETHEUS regroupant la plupart des constructeurs d'automobiles du continent.

De son côté, le Japon, en 1994, revoit ses normes légales d'émissions de Nox pour les camions à essence et au L.P.G., ainsi que pour les véhicule commerciaux diesels, de sorte que, par exemple, les véhicules de commerce légers devaient, pour 1994, réduire de 35% leurs émissions de Nox [PRE, 91].

Mais c'est aux Etats-Unis, ou plus exactement dans l'état de Californie, que seront prises les mesures les plus lourdes de conséquences pour le secteur automobile. En effet, le gouvernement fédéral édictera la " Zero Emission Act " imposant aux constructeurs automobiles de produire un certain pourcentage de véhicules totalement

non-polluants (aucune émission). Un formidable défi technologique est ainsi imposé aux grands de l'automobile.

Toutefois, le secteur privé aura su rapidement, sous la houlette de gourous publicitaires, transformer cette contrainte en un argument de vente de premier ordre. Un nouveau concept est né : " La voiture écologique ". De telle sorte qu'il devient vital pour les constructeurs, de continuer tous leurs efforts visant à réduire les niveaux de consommation. Les injections et autres boîtiers électroniques de cartographie (gestion de l'allumage) sont de plus en plus performants. « L'ordinateur de bord » devient un élément familier du mobilier d'intérieur. Et les efforts consentis porteront leurs fruits; ainsi, la consommation pétrolière française, après avoir atteint son maximum lors du premier choc pétrolier de 1973 avec près de 112 millions de tonnes et affichant encore en 1979 (deuxième choc pétrolier) un niveau de 106 millions de tonnes, était ramenée en 1984 à 77 millions de tonnes. Soit une réduction significative de 37 % sur 5 ans [FACH, 85].

2.2 Synthèse

Certes, l'histoire de l'automobile a connu bien des mutations au long de son centenaire, mais force est de constater que l'introduction de composants électroniques aura imprimé à cette évolution une courbe exponentielle. A tel point qu'aujourd'hui, il devient ardu de trouver un organe mécanique échappant au contrôle d'un calculateur.

Et cette tendance, nous la devons tant à des initiatives privées qui rivalisent d'ingéniosité pour rendre nos véhicules confortables et attrayants (ce sont par exemple, la climatisation, le régulateur de vitesse, la suspension pilotée, la boîte de vitesse automatique, ...), qu'à des directives publiques (nationales ou supranationales) pour réduire les accidents (cfr. l'airbag aux U.S.A.) et les émissions polluantes (cfr. directive 85-210/CEE publiée dans le J.O.C.E. en date du 3 avril 1985). Outre les apports marginaux liés à « l'art du gadget », il est indéniable que nos véhicules acquièrent toujours plus de compétence et de la sorte, c'est nos voitures qui deviennent intelligentes.

Demain peut-être, nous vivrons l'ère des bureaux mobiles, ultra confortables et sécurisés, qui navigueront avec précision et efficacité dans nos mégalopoles, telles des abeilles autour de leur ruche.

3. FONCTIONS

3.1 Introduction

Le chapitre précédent, qui constitue en quelque sorte un tour d'horizon succinct du concept de véhicule intelligent, nous permet de mettre en exergue les grands changements qui ont touché l'automobile et son environnement. Ce faisant, nous avons pu identifier trois fonctions, liées étroitement au concept de véhicule intelligent, qui pourront être supportées par des systèmes de traitement de l'information.

Nous allons les aborder de manière chronologique. La fonction la plus ancienne est liée aux organes mécaniques (3.2 Gestion des Organes Mécaniques); ensuite, ce furent les fonctions de navigation qui connurent des « réalisations informatiques » (3.3 L'aide à la navigation). Pour l'heure, l'activité se concentre sur l'aide au pilotage (3.4 Aide au pilotage).

3.2 Gestion des Organes Mécaniques

Comme nous l'avons entrevu dans les points qui précèdent, les premières applications de l'informatique à l'automobile, et ce nonobstant les phases industrielles liées à la conception et au montage, ont eu pour objectif le pilotage optimal de certains organes mécaniques tels que le carburateur, l'injection, les freins, etc. Les tendances générales qui ont prévalu à l'informatisation de ces organes mécaniques s'articulent essentiellement autour d'une innovation technologique : le passage de l'électronique analogique à l'électronique digitale.

Décennie 70-80 : Les automobiles haut de gamme voient apparaître, dans leur dotation en équipements, des systèmes de gestion électronique de type monofonction. Nous sommes ici en présence d'une technologie analogique proposant des calculateurs munis d'un grand nombre de composantes à échantillonnages discrets, ayant peu de portabilité d'une année modèle à une autre. Ils offrent de surcroît une médiocre fiabilité suite essentiellement à la présence de soudures et d'importantes variations thermiques provoquées par les amplificateurs. C'est pourquoi, seules des fonctions séparées telles

que l'allumage, l'injection d'essence, les freins ou encore la climatisation automatique pourront être gérées. Notons que l'apport de cette technologie dans le domaine de l'allumage et de l'injection aura déjà un effet bénéfique sur la consommation, soient des gains allant de 0,6 à 1 litre aux 100 km selon le type de conduite.

Décennie 80-90 : L'avènement de la micro-électronique numérique et de l'ingénierie logicielle permettent le développement de nombreux nouveaux systèmes sur les modèles des gammes moyennes et basses. Les importantes avancées en matière de gestion des freins (ABS) permettent au haut de gamme de connaître des systèmes multifonctions tels que le pilotage de la suspension (cfr. la suspension hydractive de Citroën), de la motricité (cfr. ASC de BMW), des roues (cfr. 4 roues directrices chez Honda), de la boîte de vitesses (cfr. Tiptronic chez Porsche). En effet, on retrouvera dans tous ces systèmes une architecture basée sur une centrale sophistiquée de commande des freins.

Les dernières innovations en la matière sont assez significatives d'une certaine tendance d'assistance au pilotage. C'est, par exemple, l'accélérateur électronique moins dangereux qu'une solution mécanique en cas de déformation de l'habitacle, mais permettant aussi d'adjoindre au véhicule une centrale de gestion dynamique de la vitesse (Cfr. nouvelle génération de Cruise Control). C'est aussi le constructeur Suédois Volvo qui, utilisant les enseignements de sa division aéronautique, propose le remplacement du séculaire volant par une commande électronique de type Joystick.

Remarquons enfin que ces efforts d'optimalisation de la gestion des organes mécaniques ont des effets directs sur l'amélioration de la sécurité active et donc en quelque sorte du pilotage.

3.3 L'aide à la navigation

3.3.1 Définitions de la navigation et du guidage

Il est important de bien comprendre la distinction entre le système de navigation et le système de guidage. La **navigation** (ou routage) consiste à assurer la planification

et le suivi des routes à l'aide d'instruments d'acquisition de données et de procédure de traitement de ces données. Le rôle du navigateur est donc double :

- planifier le trajet ;
- localiser à tout moment l'emplacement du véhicule.

Le système de **guidage**, quant à lui, se contente de piloter - à l'aide d'informations de direction - le conducteur pour qu'il puisse suivre les routes planifiées jusqu'à la destination. Le guidage consiste donc à informer dynamiquement (c'est-à-dire délivrer les informations opportunes au moment ad hoc) le conducteur sur base des données dispensées par le navigateur.

3.3.2 Bref historique des systèmes de navigation

Les premiers systèmes d'aide à la navigation sont apparus au début de ce siècle. Ils étaient principalement basés sur la méthode de l'estime. Ainsi, les *Jones Live-map* (1911), grâce à la technique de l'odomètre[1] , permettaient à un conducteur de connaître à tout moment la distance qu'il avait parcourue et celle qu'il lui restait à parcourir pour atteindre son objectif. La route à suivre était tracée sur un disque gradué en kilomètres tournant en fonction de la rotation des roues. Un pointeur fixe permettait d'identifier le kilométrage parcouru sur le disque. Il fallait bien sûr que le conducteur reste sur la route prévue.

Un autre système, développé pour des besoins militaires durant la seconde guerre mondiale, utilisait la technique du compas magnétique[2] intégré à des éléments électroniques. « Le compas orientait une tige mécanique correspondant à la direction du véhicule. La tige était reliée à un ordinateur mécanique qui calculait la distance parcourue en composantes x y à l'aide de l'odomètre et pointait automatiquement le trajet du véhicule sur une carte. L'erreur de précision était de 1 km tous les 50 à 150 km parcourus. » [CHA, 91]

Une seconde génération de systèmes de navigation vit le jour dans les années 70. Ces systèmes utilisent toujours la technique de l'estime mais la complètent par un repérage sur carte. Grâce aux cartes embarquées dans le véhicule, on peut facilement recaler la position de celui-ci puisque l'on sait que ce dernier voyage nécessairement sur une route. Citons par exemple les systèmes ARCS[3] et CARIN. Le système ARCS, élaboré en 1973 par French et Lang, permet de programmer l'itinéraire comme une séquence de vecteurs représentant la carte. Le principe comprend un microprocesseur et un odomètre. Des instructions vocales de routages sont émises au moment adéquat. Le système CARIN, quant à lui, utilise un disque compact pour stocker les cartes

[1] Instrument qui compte le nombre de tours des roues effectué par le véhicule.
[2] Type particulier de boussole indiquant la direction du véhicule.
[3] Automatic Route Control System.

routières ainsi qu'un écran permettant de visualiser la position du véhicule sur le plan routier des environs.

Une alternative au système de cartes embarquées consiste à utiliser des balises positionnées le long des routes. Le véhicule envoie sa destination aux balises et reçoit sa position ainsi que l'itinéraire à suivre. Les différents systèmes de ce type sont basés sur l'initiative américaine dans le cadre du projet ERGS[1] où la communication entre les balises et le véhicule est bidirectionnelle. Commencé vers la fin des années 60, le projet fut abandonné en 1970 car l'infrastructure nécessaire était trop onéreuse. Quelques années plus tard, les Japonais et les Allemands reprenaient le principe pour expérimenter leurs projets respectifs : CACS[2] et ALI. Ce dernier a conduit au système actuel ALI-SCOUT.

3.3.3 Fonctions et structure des systèmes embarqués

Il existe de nombreux systèmes possédant chacun leur propre matériel, leur propre protocole de communication ... Mais force est de constater que, si les matériels sont différents d'un système à l'autre (il n'existe encore aucun standard dans le domaine), les fonctions assurées par les systèmes embarqués sont généralement les mêmes et peuvent se résumer par les cinq points suivants :

- la saisie et la mémorisation de la destination introduite par le conducteur ;
- la localisation du véhicule ;
- la réception, le décodage et la sélection d'informations pertinentes pour le routage du véhicule ;
- la mise à jour régulière de la position du véhicule sur le système de représentation interne de la route à suivre ;
- le guidage du conducteur.

Afin de réaliser ces fonctions, le système embarqué dispose d'un microprocesseur, d'instruments de navigation électroniques (capteurs, odomètres, émetteur/récepteur ...) et d'une interface homme/machine assurant la communication avec le conducteur.

Le conducteur indique sa destination par l'intermédiaire d'un clavier. Le microprocesseur, qui a accès à l'ensemble du réseau routier grâce aux informations stockées sur le CD ROM, prend note de cette destination et calcule l'itinéraire optimal

[1] Electronic Route Guidance System.
[2] Comprehensive Automobile Traffic Control System.

pour l'atteindre. Une fois en route, il actualise l'itinéraire en fonction des informations fournies par le système de navigation (sur base, par exemple, des résultats d'un odomètre ou d'un compas magnétique) et les messages provenant de l'extérieur (messages radiophoniques, localisation par satellite, informations en provenance des balises ...). Le système de guidage communique l'itinéraire et les informations pertinentes via l'écran afin de pouvoir piloter le conducteur sur l'itinéraire sélectionné. On peut donc, en toute généralité, identifier les éléments constitutifs du système embarqué tels que représentés sur le schéma ci-dessous (Fig.3.1.).

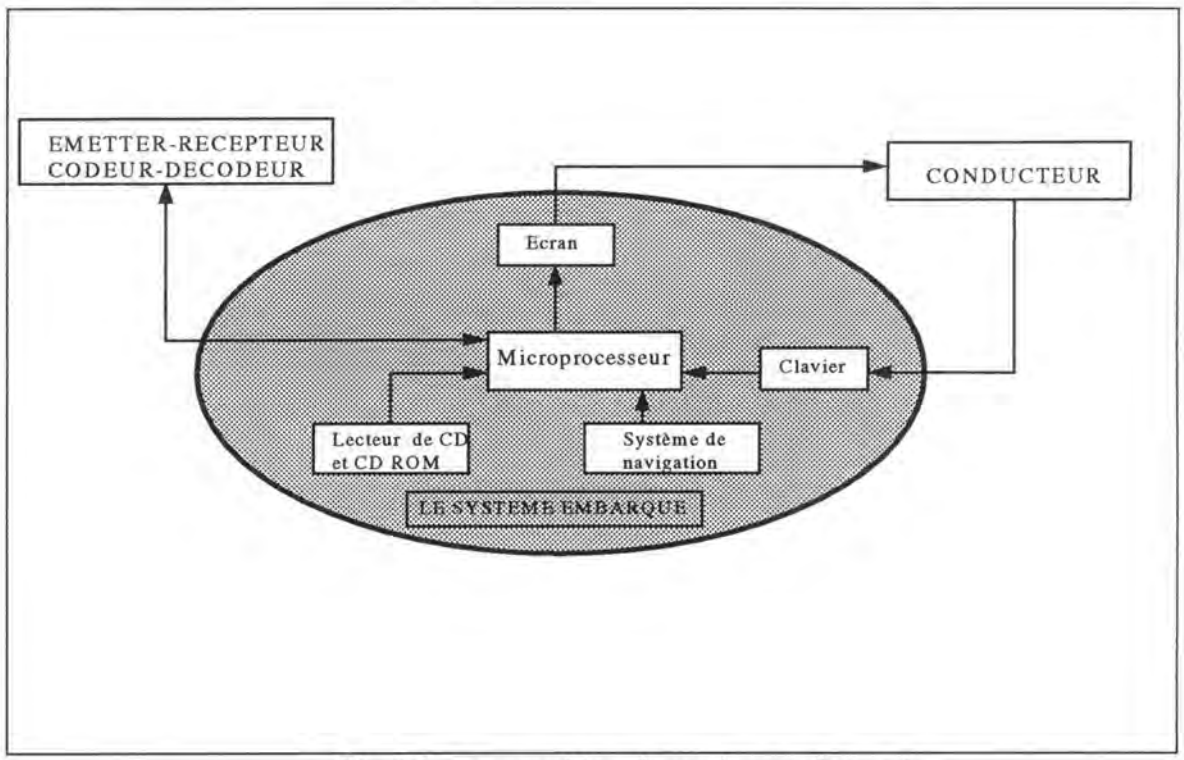


Fig.3.1. : composition du système embarqué

Remarquons toutefois que l'écran et le clavier ne sont pas les seules interfaces disponibles. Des études ergonomiques ont d'ailleurs démontré que des indications auditives concernant le guidage étaient supérieures aux présentations graphiques. En effet, la conduite d'un véhicule et la lecture d'un écran sont des tâches essentiellement visuelles. Il est donc impossible d'exécuter ces deux actions en même temps sans risque d'interférence. Des tests sur les deux types de présentation (visuelle et auditive) ont de plus révélé que les conducteurs se trompaient beaucoup moins souvent de route avec ce dernier.

3.3.4 Les deux techniques pricipales utilisées pour la navigation

Il nous paraît intéressant de consacrer une partie de ce premier chapitre à l'exposition des principales techniques utilisées dans la navigation. Ces techniques conditionnent le matériel utilisé et donc les moyens d'acquérir l'information nécessaire au guidage dynamique. Il existe deux types de technique : le calcul d'estime[1] et la radionavigation.

3.3.4.1 Le calcul d'estime

Le véhicule est équipé de **capteurs de direction** et de **mesureur des distances parcourues** afin de pouvoir réactualiser sa position par rapport à son point de départ. Le point de départ étant connu, il suffit de reporter les déplacements mesurés pour connaître à tout moment la position courante de la voiture.

Les capteurs de distance, ou odomètres simples, se contentent de traduire le nombre de tours effectués par les roues en kilomètres parcourus. Cette information est délivrée au système de navigation pour qu'il puisse effectuer les calculs d'estime.

Les capteurs de direction sont un peu plus compliqués, nous n'entrerons d'ailleurs pas dans les détails de leur conception. Il nous suffit de savoir qu'il en existe de trois types : les capteurs géomagnétiques, les gyroscopes à fibres optiques et les odomètres différentiels. Ces derniers se placent aux roues non motrices et analysent la différence de tours entre la roue droite et la roue gauche pour déterminer la direction prise par la voiture. Cette direction, tout comme la distance parcourue, est envoyée au système de navigation pour qu'il puisse mettre à jour la position du véhicule.

Si les techniques utilisées pour le calcul d'estime ont le mérite d'être peu coûteuses et de ne nécessiter aucune infrastructure extérieure, elles se révèlent de piètre qualité lorsque les distances parcourues deviennent importantes ou lorsque le véhicule effectue trop de manoeuvres. En effet, la précision des capteurs est loin d'être absolue et la position estimée dérive peu à peu de la position réelle. Il est donc nécessaire, à intervalle régulier, de réactualiser la position de la voiture par des techniques comme le map-matching[2].

[1] « Dead reckoning » en anglais.
[2] Le map-matching -ou système de repérage sur carte- suppose que le véhicule se trouve toujours sur une route et recale les données de l'estime par rapport à une carte mémorisée du réseau.

3.3.4.2 Les techniques de radionavigation

Ces techniques utilisent, comme leur nom l'indique, des signaux radiophoniques et permettent une localisation absolue de la voiture. Le système le plus prometteur est sans nul doute le GPS[1]. Sa précision et les limites imposées par le comportement des signaux radio ont obligé les utilisateurs civils[2] à perfectionner le système en le dotant d'aide au sol de façon à ramener la précision de localisation à quelques centimètres d'erreur.

Le principe de base du fonctionnement de la navigation par GPS est relativement simple : 24 satellites gravitent autour de la terre de façon à ce que, en chaque point de celle-ci, six satellites au moins soient au-dessus de l'horizon. Les satellites transmettent constamment des signaux. Ceux-ci sont suivis et décodés par un récepteur passif (ce recepteur n'émet rien vers le satellite).

La localisation d'un véhicule est réalisée grâce à la méthode de **triangulation :** lorsque les positions et les distances séparant (au moins) trois points sont connues, toute localisation relative à ces trois points peut être déterminée par triangulation. Les satellites servent alors de points de repère, puisque, à chaque moment, leur position sur leur orbite est connue - ou du moins, peut être calculée. Remarquons que, en pratique, le récepteur doit recevoir des signaux d'au moins quatre satellites pour opérer une localisation tridimensionnelle.

Puisque, à l'origine, la précision d'une telle localisation pour une application civile laisse à désirer quant à son utilisation pour la navigation routière, les constructeurs ont imaginé plusieurs moyens ou techniques, qui, couplés au système GPS, lui permettent d'obtenir une précision de localisation au centimètre près. Nous proposons dans les deux paragraphes suivant une brève présentation de ces techniques. Les premières sont des améliorations apportées au système GPS lui-même, les secondes sont des augmentations, c'est-à-dire, des appareils qui peuvent être couplés au GPS pour en accroître la précision.

L'amélioration la plus sensible pour la précision repose sur un principe simple : la **correction différentielle**. Deux récepteurs dans la même région feront approximativement la même erreur en mesurant le signal du satellite car les causes majeures d'erreurs sont communes aux deux récepteurs. Ces causes peuvent être par exemple : le délai de transfert du signal à travers l'ionosphère et la troposphère, une incertitude sur la position du satellite, une erreur d'horloge du satellite ou encore l'application d'une S/A[3] ... L'idée consiste alors à placer un **récepteur de référence** à

[1] Global Positionning System.
[2] « La précision d'un tel système est de 100 mètres pour une application civile, mais peut atteindre 10 mètres s'il s'agit d'applications autorisées par le gouvernement ». [Cha91]
[3] Selective Availibility ou dégradation intentionnelle de la précision du GPS par l'armée des Etats-Unis.

un endroit fixe et à mesurer les effets agrégés de ces erreurs. Quand cette mesure est fournie à un autre récepteur, ce dernier peut alors corriger ses données grâce à cette référence. Les gains de précision réalisés avec l'utilisation d'un correcteur différentiel permettent de réduire l'erreur de localisation à 5 mètres[1]. D'autres méthodes, telles que la « Narrow Correlator technology » et le « Carrier Phase Tracking », permettent de réduire l'effet multipath[2] et d'accroître ainsi la précision de localisation à quelques dizaines de centimètres près.

Divers outils peuvent être couplés au GPS afin d'augmenter la robustesse du système de localisation dans les régions où une obsturation du satellite se produit. En effet, lorsque la voiture passe sous un pont ou dans un tunnel, lorsqu'elle évolue entre de hauts buildings ... les ondes transmises par le GPS ne peuvent plus atteindre le récepteur embarqué. Le véhicule se retrouve alors sans aucune indication de localisation. Pour remédier à cet inconvénient, on peut faire appel aux capteurs de direction et de distance dont nous avons déjà parlé. Leur excellente précision sur de petites périodes en font un outil complémentaire - le GPS fournissant la référence absolue de départ pour les odomètres - à la localisation par satellite. Lorsque la période de perte du signal est trop longue pour utiliser des IRU[3], on peut se servir de **pseudolites.** Un pseudolite est une machine qui émet des signaux semblables à ceux du satellite mais à partir du sol. Il faut cependant veiller à ce que les signaux du pseudolite n'interfèrent pas avec ceux du satellite. En effet, à cause des interférences, un récepteur trop près d'une de ces machines ne pourra pas recevoir les informations en provenance d'un satellite. Par contre, un récepteur trop loin d'un pseudolite ne sera pas capable de suivre le signal de ce dernier.

En guise de conclusion, nous pouvons dire que, pour qu'un système GPS civil fournisse une précision suffisante pour être utilisé par un IVHS, les systèmes suivants sont nécessaires :

- Un récepteur GPS embarqué permettant de faire de la correction différentielle, disposant d'un narrow correlator ainsi que d'un carrier phase tracker.
- Un IRU pour permettre une navigation en continu alors que les signaux du GPS n'arrivent plus (tunnels, buildings ...).
- Une base de donnée cartographique précise dans le système de coordonnées utilisées par les signaux GPS afin que le véhicule connaisse à tout moment sa localisation par rapport à la route.

[1] Plus la distance entre la référence et le récepteur augmente, plus la correction devient incertaine. Ainsi, une précision au mètre près n'est possible que lorsque le recepteur est relativement près de la référence.
[2] Erreurs dues aux réflexions des ondes.
[3] Internal Reference Unit tels que les odomètres ...

- Pour les longues périodes d'occultation du signal, des pseudolites doivent être présents.

Afin de mieux visualiser l'architecture et la composition d'un tel système, ses différents composants ont été reportés sur la figure suivante (Fig. 3.2.):

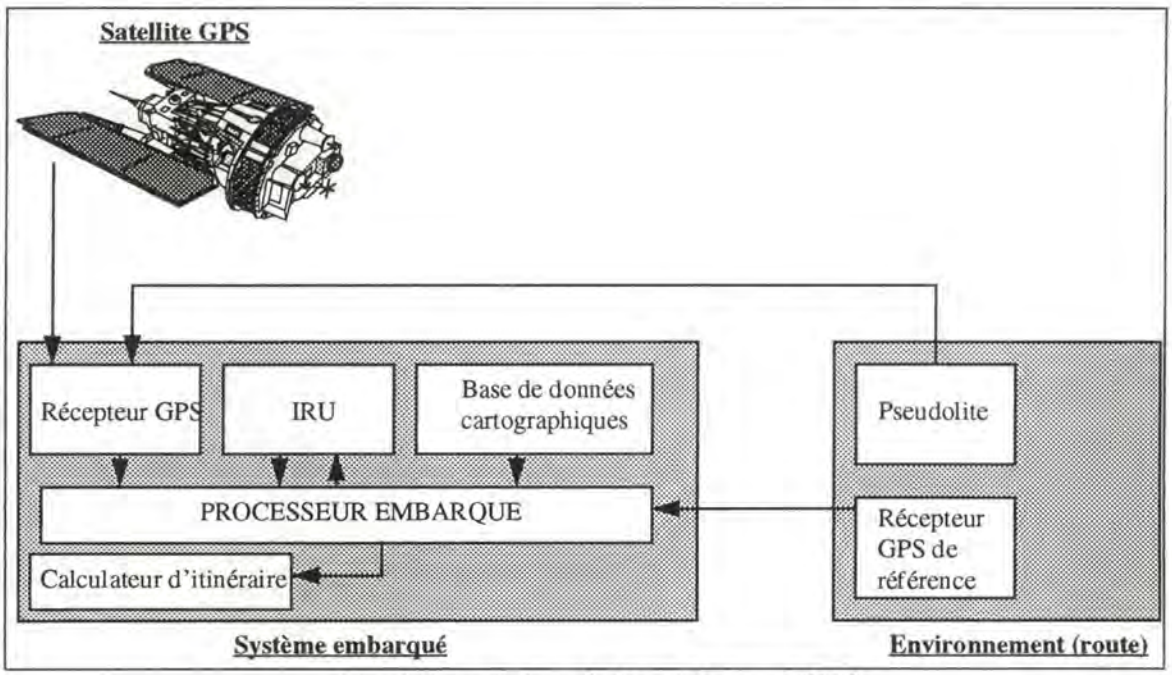


Fig. 3.2. : schéma de l'architecture GPS

3.3.5 L'infrastructure

3.3.5.1 Introduction

Jusqu'à présent, nous avons négligé un aspect important du guidage : son dynamisme. Afin de procurer à tout moment au conducteur un itinéraire optimal par rapport aux critères que ce dernier à choisis -rapidité, confort, coût minimum...- il est impératif d'avertir le navigateur des changements survenus sur le réseau routier emprunté. Les informations concernant les embouteillages, les routes barrées ... ne peuvent pas être insérées dans la base de données cartographiques du système embarqué. Il est donc nécessaire d'organiser un système de communication entre l'environnement et le système de navigation. Ainsi, sur base de ces informations, le navigateur pourra, s'il y a lieu, recalculer un itinéraire mieux adapté aux nouvelles conditions.

Outre le désir d'opérer un routage dynamique, certains systèmes (RACS, ALI-SCOUT...) utilisent des balises servant à la fois à fournir des informations routières mais aussi des communications personnelles, des informations

touristiques ... et des informations ayant pour but de recaler la position du véhicule. De tels systèmes nécessitent donc une infrastructure extérieure au véhicule pour réaliser la tâche de navigation, que celle-ci soit dynamique ou non.

Quel que soit le point de vue considéré, nous pouvons caractériser les types d'infrastructures utilisés suivant deux grandes qualifications : la première consiste à différencier les systèmes à communication unidirectionnelle des systèmes à communication bidirectionnelle, la seconde distinction porte sur le type d'émission, sur une zone ou en un point fixe.

3.3.5.2 Systèmes avec communication unidirectionnelle

Emission sur une zone

Dans cette situation, le système transmet, par radio et sur une zone donnée, des informations concernant cette zone. Tous les véhicules équipés se trouvant alors dans la dite zone reçoivent en même temps les mêmes informations. Les informations proviennent généralement des centres nationaux ou régionaux d'information routière, ceux-ci étant le plus souvent reliés à différents postes de police. Ces informations portent sur les congestions, les incidents de trafic ou les changements du réseau dus notamment aux travaux entrepris sur certains tronçons.

Il existe deux systèmes basés sur ce principe. Il s'agit de AMTICS[1] et de CARMINAT.

Le schéma ci-dessous reprend le principe général ainsi que les différents composants d'un tel système.

[1] Adveanced Mobile Traffic Information and Communication System (Tsugawa et Okamoto, 1989). Système pour lequel une expérience pilote a été menée à Tokyo pendant trois mois en avril 1988. En plus du récepteur terminal, un récepteur micro-ondes est chargé de recueillir des informations de localisation en provenance d'une balise qui servira au principe de navigation.

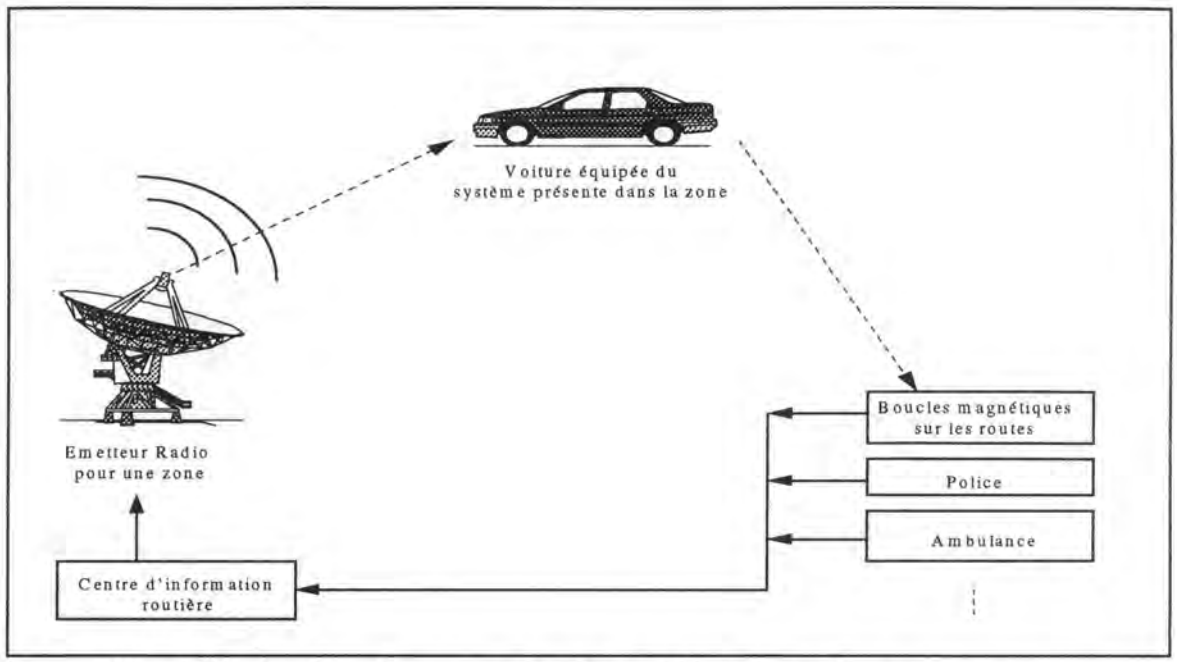


Fig. 3.3. : Système à communication unidirectionnelle sur une zone.

Nous ne présenterons que le projet européen CARMINAT et ce pour deux raisons : premièrement, il utilise un système extrèmement répandu à l'heure actuelle, le RDS ; ensuite, il vient d'être commercialisé par Renault et fait donc l'objet d'une application réelle.

C'est en 1986, sous l'initiative de l'Euréka que le programme CARMINAT voit le jour. Il résulte de la conjonction des trois projets suivants :

- le projet ATLAS[1] qui débute en 1980 et qui est mené par le constructeur Renault et la Télé Diffusion de France (TDF);
- le projet CARIN[2], datant de 1980 et développé par Philips;
- le projet MINERVE[3] développé par la Sagem en 1986.

Son nom est d'ailleurs l'agrégation des premières syllabes des trois projets cités.

L'architecture générale du système peut être qualifiée de modulaire et d'évolutive. En effet, différents modules s'organisent autour d'un processeur d'interface homme/machine contrôlant les échanges d'informations entre le conducteur et les différents modules. Le caractère évolutif est réalisé par un bus de communication sur lequel viennent se greffer les différents sous-systèmes : capteurs, interfaces, calculateurs ... La figure ci-dessous illustre cette architecture (figure 3.4.).

[1] Acquisition par Télé Diffusion de Logiciel Automobile pour les Services.
[2] CAR Information and Navigation.
[3] Média Intelligent pour l'Environnement Routier du Véhicule Européen.

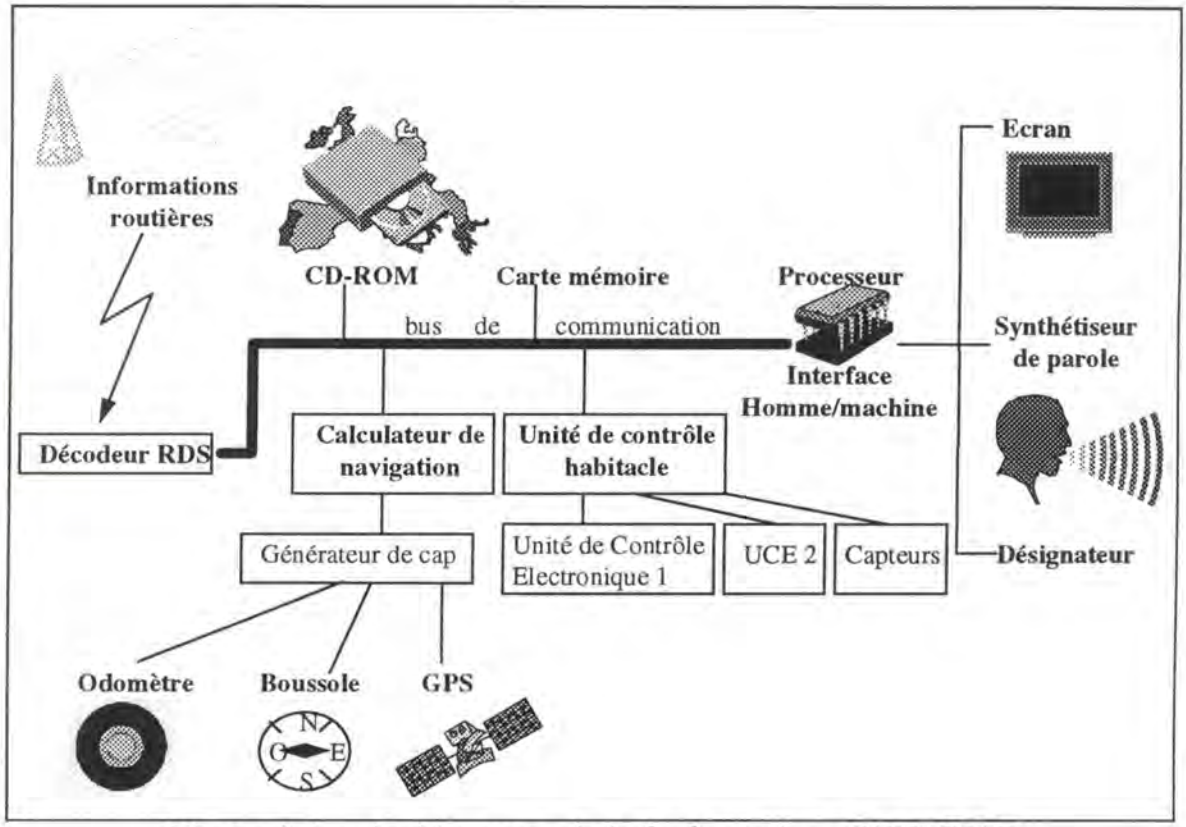


Figure 3.4. : Architecture générale du système CARMINAT

Les messages **RDS**[1] (ou encore **RDS-TMC**[2]) sont transmis sur une fréquence de 57 KHz à un taux de 1187.5 bits/sec., sous forme de trame composées de quatre blocs de 26 bits chacun. Ces messages digitaux sont reçus via un canal FM ou AM et sont bien sûr inaudibles. Le décodeur embarqué possède des tables (TMC look up tables) et un catalogue d'événements lui permettant de déchiffrer les messages reçus selon le protocole ALERT-C. Le lecteur intéressé par le codage des messages RDS et le protocole ALERT-C peut consulter [BRAE,90].

L'unité de contrôle habitacle, qui commande les fonctions du véhicule, est composée d'une base numérique (un PC embarqué avec un processeur 16 bits et 1 Mb de mémoire), d'un logiciel de contrôle écrit en langage C et d'une interface correspondant aux différents capteurs.

La **carte mémoire du véhicule** (16 Kb) est utilisée pour différentes applications (4 Kb) (antivol ...) mais aussi comme base de données (12 Kb) pour suivre l'évolution de la vie du véhicule (anomalies, maintenance, manuel ...).

Le **calculateur de navigation** est relié, via un module d'acquisition de données, à différents capteurs lui permettant de connaître l'orientation (**boussole**), la distance parcourue (**odomètre**), la localisation du véhicule (**GPS**)... mais aussi les informations sur l'état des routes et du trafic (**RDS**) ainsi que les différentes voies utilisables (**CD-ROM**).

L'interface Homme/Machine renseigne le conducteur à l'aide d'informations de guidage tant sous forme auditives que graphiques.

[1] Radio Data System. (La version américaine du RDS s'appelle : le RBDS : Radio Broadcast Data System.)
[2] Traffic Message Chanel

Le **processeur** de commande (de type 68000) est relié au bus de communication et est utilisé pour afficher ou faire entendre les informations de guidage à l'usager.

Le système est donc bien équipé pour assurer les trois fonctions suivantes : la localisation du véhicule (GPS, odomètre, boussole), la planification du trajet (CD-ROM, RDS), et le guidage dynamique sonore et/ou visuel (Interface avec écran et synthétiseur vocal).

Emission en un point fixe

Dans ce type de comunication, les véhicules reçoivent l'information via des balises émettrices se trouvant à des endroits précis du réseau routier.

Le système japonais RACS[1] utilise cette technique et s'est fixé pour but de « dévolopper une balise servant à la fois à fournir des informations routières, recaler la position du véhicule (localisation), permettre des communications personnelles et donner des informations touristiques. »[CHA, 90]

Les balises utilisent un système à micro-ondes permettant un taux de transmission très élevé. La communication entre la balise et le véhicule n'étant possible que sur une zone de 50 mètres autour de la balise, il faut jalonner les routes de balises, celles-ci étant généralement espacées de 2 à 5 km.

3.3.5.3 Systèmes avec communication bidirectionnelle

Emission sur une zone

Les systèmes de comunication bidirectionnelle agissant sur une zone sont basés sur l'utilisation de radios cellulaires. PATHFINDER[2] et SOCRATES[3] appartiennent à cette catégorie.

Le système PATHFINDER utilise le principe suivant : pendant les dix premières secondes de chaque minute, un central transmet l'information à tous les véhicules équipés se trouvant sur la zone; ensuite, durant les cinquante secondes restantes, les véhicules répondent les uns après les autres, transmettant au central leur position, vitesse ... Il est évident qu'un tel système ne peut correctement fonctionner qu'avec un nombre restreint de véhicules.

Le système SOCRATES est un projet Européen qui a débuté le 1er janvier 1989 pour se terminer par une démonstration encourageante sur le site de Göteborg en Suède à la fin 1991. SOCRATES utilise des GSM (Groupe Spécial Mobile) pour communiquer entre les voitures et les centres de contrôle.

[1] Road Automobile Communication System.
[2] Système expérimental américain issu de la coopération entre le Federal HighWay Administration (FHWA), Caltrans et General Motor (GM). PATHFINDER a débuté en 1988.
[3] System Of Cellular Radio for Traffic Efficiency and Safety.

« Par l'intermédiaire du canal SOCRATES, le système transmettra les temps de parcours courants et prédits aux véhicules équipés d'un système de navigation» [CHA, 90] (comme CARIN par exemple). Le calculateur de navigation du véhicule déterminera alors la meilleure trajectoire pour la voiture.

Emission en un point fixe

Dans ce type de système, c'est le centre de contrôle qui se charge du calcul des chemins optimaux et non le calculateur d'itinéraire embarqué. Afin d'accroître la précision de ses calculs, le centre de contrôle reçoit en permanence des informations en provenance des différents véhicules équipés (notamment les temps de parcours de ceux-ci) mais aussi de la police ou d'autres organismes en relation avec le trafic routier.

S'il connaît la destination du véhicule, le système central se contente de communiquer au véhicule le chemin optimal pour atteindre cette destination. C'est le cas notamment du système japonais CACS[1]. Si, par contre, le central ne connaît pas la destination du véhicule, il envoie à ce véhicule l'arbre des plus courts chemins reliant la position du véhicule à toutes les destinations. C'est ce que font des systèmes comme ALI-SCOUT ou AUTOGUIDE.

CACS

CACS est un projet japonais qui a débuté il y a une vingtaine d'années. Il s'est terminé en 1978 par une série d'essais sur une zone expérimentale couvrant 28 km^2 de la ville de Tokyo.

Lorsque le véhicule équipé passe devant une balise, il reçoit un signal lui indiquant qu'il peut communiquer. Le véhicule envoie alors sa demande de routage (celle-ci contient un code pour identifier sa destination) et la balise lui fournit une réponse. Les balises sont de type inductif : elles utilisent des bobines émettrices-réceptrices. Leurs antennes émettent à une fréquence de 105.6 kHz un signal dont la direction de propagation est perpendiculaire à la bobine. Les balises sont programmées pour recevoir des signaux à une fréquence de 175.8 kHz pour autant que le signal à recevoir ait une direction de propagation perpendiculaire à la bobine. La bobine placée à l'intérieur du véhicule doit être positionnée d'une certaine façon par rapport aux antennes au sol. La vitesse de transmission d'un tel système est de 4800 bits/sec.

« Les balises reçoivent les tables des plus courts chemins toutes les 15 minutes de la part du centre de commande. Dans ces tables, les index de destination ont une structure d'arbre, et les branches terminales de l'arbre pointent sur les index d'entrée. Ainsi, lorsqu'un véhicule arrive et donne sa destination, la balise cherche dans la table et lui envoie le routage correspondant à sa destination. » [CHA,90]

[1] Comprehensive Automobile traffic Control System.

Le calcul des chemins optimaux est effectué au centre de commande. Des ordinateurs, mettant continuellement à jour des bases de donnés[1], envoient leurs données à des mini-ordinateurs sur lesquels tournent des simulateurs de trafic[2]. Ce sont ces mini-ordinateurs qui calculent les chemins optimaux et génèrent les tables de routage qui seront envoyées aux balises par l'intermédiaire d'un troisième type d'ordinateurs chargés d'effectuer les communications.

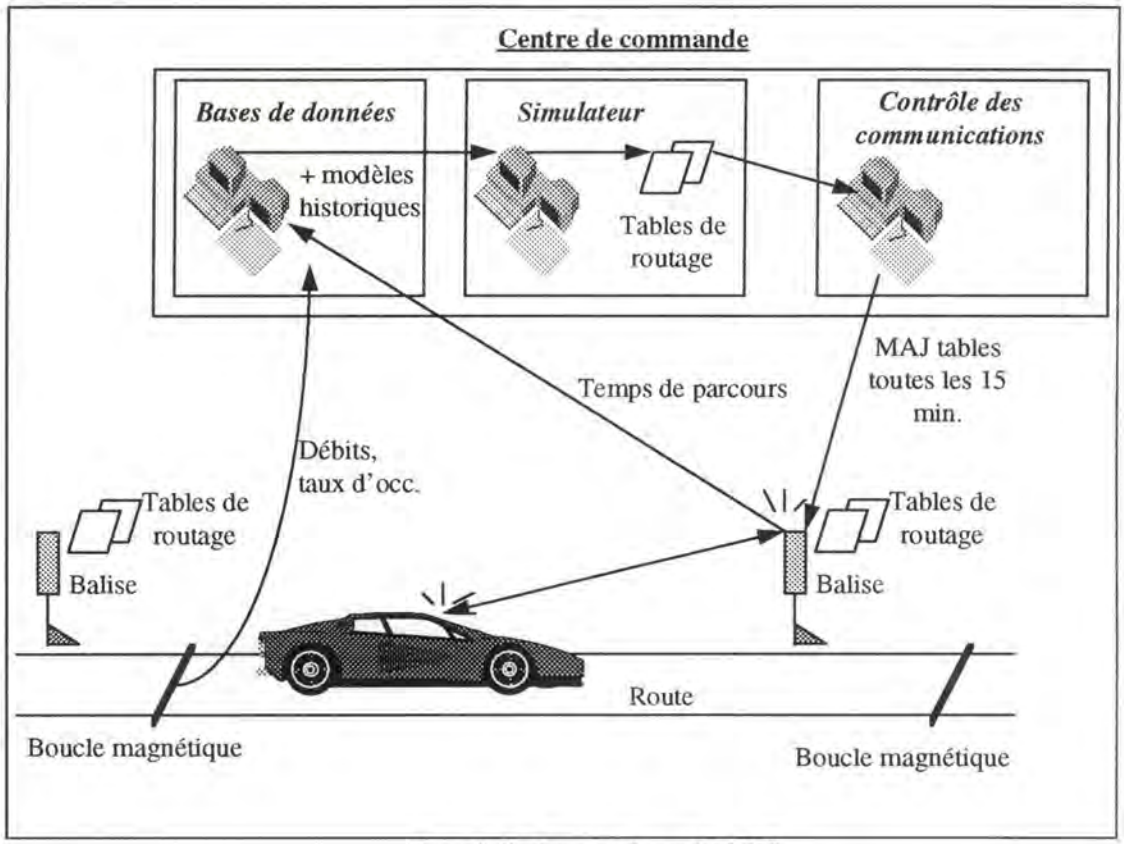


Fig. 3.5. Le système CACS

ALI-SCOUT

Projet allemand rassemblant les expériences de Bosch/Blaupunkt (ALI) et du système AUTO-SCOUT de Siemens, le système ALI-SCOUT utilise des balises à infra-rouges pour guider les véhicules sur un réseau routier donné. Le système embarqué ne possède pas de carte du réseau, seules les coordonnées de l'origine et de la destination sont connues. Un calcul d'estime embarqué, recalé par les balises rencontrées, permet de connaître en permanence la localisation du véhicule.

Lorsque le véhicule équipé passe à proximité d'une balise, il fournit ses temps de parcours à l'ordinateur du centre de commande et reçoit en retour la carte

[1] Les données concernet les temps de parcours , les débits et taux d'occupation des tronçons routiers constituant le réseau.

[2] C'est le mode sofware, basé sur des algorithmes de simulation. Il existe un autre mode : le mode hardware. Le mode hardware utilise une machine spécialement conçue pour simuler le trafic sur un réseau routier : le TNSS (Traffic Network Simulator System). Un TNSS est « un circuit de composants électroniques connectés de façon à représenter un réseau routier. La propagation de pulsations électriques dans le circuit simule le trajet des véhicules »[CHA,90].

routière locale des environs (d'un diamètre n'excédant pas 10 Km) et **l'arbre des plus courts chemins reliant la balise à toutes les destinations**. Le système embarqué analyse alors l'arbre des destinations et en déduit l'itinéraire optimal qu'il transmet au fur et à mesure au conducteur du véhicule.

3.4 Aide au pilotage

Enfin, les avancées faites dans les domaines de gestion des organes mécaniques et de l'aide à la navigation permettent d'offrir au conducteur une véritable aide au pilotage de son véhicule, participant ainsi à l'amélioration de la sécurité active. L'idée ultime liée au concept de véhicule intelligent est d'ailleurs d'introduire le concept aérien des « pilotes automatiques » au domaine des automobiles. Il s'en suivrait un gain de sécurité mais aussi de fluidité par une gestion accrue du trafic.

A la lecture de ce qui se fait en la matière, nous avons observé deux approches distinctes du concept d'aide au pilotage. La première vision propose un module passif qui se cantonne à des fonctions de monitoring et d'information du conducteur (3.4.1). La deuxième approche, quant à elle, se veut plus maximaliste en proposant, lorsque certaines conditions que nous énoncerons seront remplies, une conduite partiellement, voire même totalement automatisée (3.4.2).

3.4.1 Monitoring et information

Les départements de recherche et de développement du groupe Peugeot Citroën, en collaboration avec des organismes publiques français de recherches (Université de Technologie de Compiègne, le CNRS), ont, dans le cadre du projet Pro-Art de PROMETHEUS, développé un copilote capable d'évaluer les risques et conseiller le conducteur des actions possibles lors de manoeuvres. Ce copilote est intégré au sein d'une Peugeot 605 de série.

Il s'agit à proprement parler d'un système expert qui, au travers de capteurs et de sa base de connaissances, réalise l'évaluation de certaines manoeuvres. Il se cantonne à un rôle d'information vocale.

Cependant, nous pensons que, dans la pratique, un tel système révélera vite ses limites. En effet, il est connu de tous que bon nombre de conducteurs commettent des erreurs de conduite. Tantôt elles sont fortuites ou accidentelles, tantôt elles sont commises sciemment (par exemple : griller un feu orange, dépasser les limites légales, non respect des lignes blanches, ...). Et dans ce dernier cas, le conducteur va rapidement vouloir déconnecter cet ordinateur de bord qui lui semble trop tatillon,

cette voix synthétique par trop moralisatrice qui n'aura de cesse de lui rappeler ses ecarts de conduite.

3.4.2 Automation

Nous entendons par « automation », l'exécution totalement automatisée de manoeuvres de conduite par le biais d'une combinaison d'organes mécaniques et de systèmes de traitement de l'information.

3.4.2.1 Partielle

Pour l'heure, les possibilités d'automation ne sont que partielles. Elles ne concernent que certaines manoeuvres ne présentant pas trop de difficultés de modélisation. Notons qu'il ne s'agit encore que de prototypes ne connaissant aucune homologation officielle et, à fortiori, de production de masse.

Il s'agit par exemple des manoeuvres liées au stationnement ou encore au dépassement sur autoroute. Celles-ci, au travers de capteurs et calculateurs, peuvent être entièrement prises en charge par l'ordinateur de bord.

3.4.2.2 Totale

A l'instar de ce qui se fait déjà en robotique, avec par exemple des robots magasiniers, lorsque toutes les manoeuvres liées à la conduite d'un véhicule auront été modélisées et automatisées, il n'y aura plus d'obstacle à l'élaboration d'un véhicule totalement automatisé.

Mais alors qu'un robot évolue dans un univers confiné, clos et totalement connu, la voiture, quant à elle, doit affronter un monde ouvert soumis aux contraintes environnementales et donc aux impondérables. Intuitivement, on aura compris que, pour atteindre cet objectif avec des niveaux de sécurité inhérents au transport de personne, il faudra se doter de structures de traitement et de communication très performantes.

Remarque : Historiquement, le concept de pilotage automatique est issu de l'aéronautique. Mais, là où le réseau routier est fortement saturé et réduit à deux dimensions, les voies aériennes, quant à elles, sont encore

relativement fluides et se projettent en trois dimensions. En outre, il est à noté que les constructeurs aéronautiques sont souvent liés de près ou de loin à des complexes militaro-industriels et connaissent donc des niveaux d'investissement colossaux. C'est la raison pour laquelle les avions connaissent plusieurs longueurs d'avance dans l'aide au pilotage.

3.5 Synthèse

Très vite, au cours de notre état des lieux des efforts mis en oeuvre pour faire de nos automobiles des véhicules intelligents, nous nous sommes rendus compte qu'il s'agissait en définitive d'un problème plus général : l'informatisation d'une voiture. Soit un processus qui consiste à intégrer des systèmes hétérogènes, autonomes mais communiquant et au besoin coopérant dans un environnement très changeant, où le facteur de temps-réel revêt toute son importance.

Il s'agit donc d'une **Architecture Distribuée Temps-Réel**. Nous nous proposons de discuter quel pourrait être le **Système de Commandes** qui piloterait une telle architecture. Plus particulièrement, nous développerons un **Modèle de Représentation des Agents** constitutifs d'un véhicule intelligent, ainsi qu'un **Protocole de Communication** intégrant les contraintes de synchronisme et de réactibilité liées au Temps-Réel.

4. SYSTEME DE COMMANDES TEMPS-REEL DISTRIBUE

4.1 Introduction

Connaissant les fonctions supportées par un "véhicule intelligent" et les matériaux mis en oeuvre afin de réaliser ces fonctions, intéressons-nous maintenant à la configuration informatique nécessaire au support d'un tel pilote automatique.

Dans un premier temps, nous nous attacherons à présenter notre vision d'une architecture distribuée soumise à des contraintes temps-réel (4.2 Architecture Temps-réel Distribuée). Ensuite, nous étudierons brièvement les mécanismes de communication classiques et nous présenterons leurs limites (4.3 Modèles et mécanismes des communications classiques). Enfin, nous nous doterons d'un modèle de représentation et de structuration de données indispensable pour la poursuite de notre étude (4.4 Approche Objet).

4.2 Architecture Temps-réel Distribuée

4.2.1 Qu'est-ce qu'une architecture distribuée ?

Il s'agit d'un ensemble de machines autonomes, dont les sites sont géographiquement dispersés, reliées entre elles par un réseau de communication, et qui concourent à la réalisation d'objectifs communs (fig. 4.1.). La distribution s'applique donc tant au niveau physique (machines), qu'au niveau logiciel du système (système & applications). Ainsi, on parlera aussi de systèmes dits coopératifs. Il faudra dès lors mettre en oeuvre des outils de contrôle et de distribution des capacités de calcul, des traitements et des données.

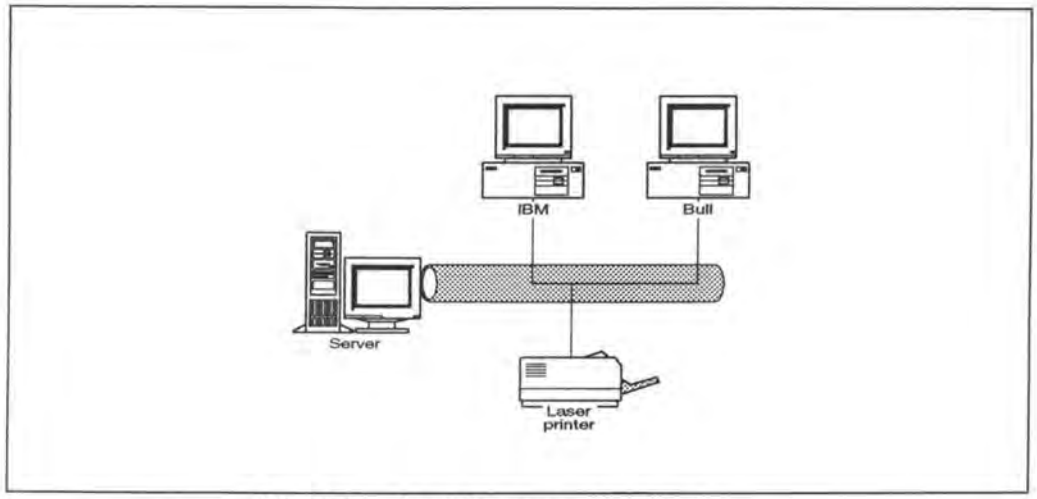


Fig.4.1. : Architecture distribuée

4.2.2 Pourquoi une architecture distribuée ?

Depuis quelques années, l'informatique de gestion semble donner la faveur, et ce au détriment des Main Frames, au développement de réseaux composés de micro-ordinateurs situés de manière éparse dans l'organisation. L'argumentation de ses promoteurs repose sur la réduction des coûts matériels (de l'ordre de 1 à 20) et, en vrac, l'amélioration de l'ergonomie (utilisation des interfaces G.U.I.), la flexibilité, l'extensibilité ou encore de la sécurité. Cependant, le manque crucial de méthodologie efficiente permettra aux détracteurs de mettre en exergue les coûts importants de conception et de développement (coûts humains) inhérents à ces systèmes. De sorte que, actuellement, bon nombre d'entreprises sont dans l'expectative quant au choix de développer et conserver leur système centralisé existant ou de migrer vers une architecture décentralisée (downsizing).

Dans le cas particulier des pilotes automatiques de véhicules, le choix d'une telle architecture (distribuée) s'impose par le caractère même de la distribution physique des matériels présents dans un véhicule. En effet, rappelons que la mise au point d'un pilote automatique se base, pour une partie, sur l'intégration d'équipements déjà existants sur les véhicules et donc, ayant leurs propres fonctions telles que, par exemple, la gestion des freins, de l'injection, de la suspension...

4.2.3 Le niveau physique

Nous définissons le niveau physique comme un ensemble de machines concourant à la réalisation d'un objectif (4.2.3.1) auquel on aura pris soin de greffer des moyens leur permettant de communiquer entre elles et avec le monde extérieur (4.2.3.2).

4.2.3.1 Support de l'application

Il s'agit d'une collection d'équipements dédiés, le plus souvent hétérogènes, possédant leurs propres mémoires et capacités de traitement (C.P.U. + Horloge).

Armés de notre cahier des charges en termes de fonctionnalité requises pour la mise en oeuvre d'un pilote automatique de voiture, nous avons fait le tour de ce que l'état actuel de la technologie pouvait proposer comme solutions matérielles. Nous en dérivons une nomenclature des équipements requis qui s'articule autour de deux classes distinctes.

Premièrement, nous retrouvons les équipements faisant déjà partie de la dotation possible des véhicules de gamme moyenne, ce sont :

Equipements existants
La centrale de gestion des freins.
La centrale d'injection et d'allumage.
La centrale de motricité.
Le Cruise Control.
La suspension pilotée.
Un poste de radio doté du R.D.S.
Une centrale de communication G.S.M.

Deuxièmement, il y a ceux qui seront spécifiquement dédiés à l'automation du pilotage et qu'il convient de greffer, et ce de manière la plus harmonieuse possible, sur les véhicules. Ces matériels peuvent être de très haute technologie, ce sont :

Equipements à ajouter
Des caméras digitales (au nombre de quatre).
Une caméra digitale infrarouge. Un Laser de détection de position et de mouvement.
Une centrale d'auto-localisation G.P.S. couplée à un calculateur d'estime via la centrale de gestion des freins. Une centrale de communication avec le réseau.
Une centrale de communication avec le conducteur composée d'une unité de production et de reconnaissance vocale et d'un incrustateur d'images sur le pare-brise.

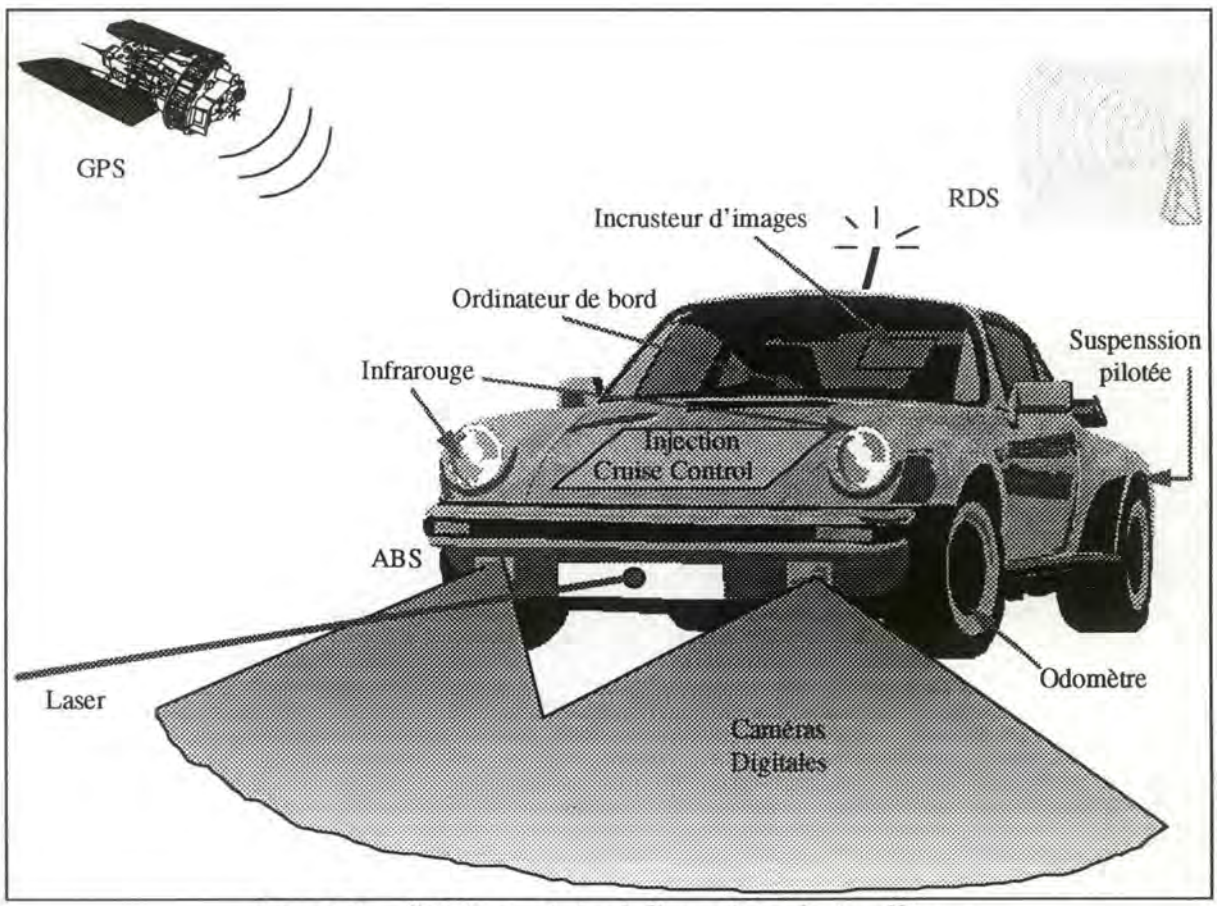


Fig. 4.2. : Schéma général d'un véhicule intelligent.

4.2.3.2 Support de la communication

A l'heure actuelle, les systèmes électriques des voitures de haut de gamme nécessitent le montage de plus de 1000 câbles pour une longueur de près de 2.5 Km. En outre, la tendance à la multiplication des accessoires électriques et/ou électroniques fera, dans les annnées à venir, fortement croître ce chiffre.

Dès lors, la mise en place de réseaux de communication et d'alimentation, à l'instar de ce qui se fait dans le domaine de l'aéronautique, devenait évidente tant en termes :

a) de coûts, avec une réduction globale estimée à plus de 10%, consécutive à des gains sur l'achat de matériaux et sur les frais de montage; mais aussi par une réduction de poids et donc de consommation, si minime fût-elle ;

b) d'efficacité, avec des possibilités simplifiées d'ajouts d'éléments. Ainsi, par exemple, la mise en place d'une alarme ne nécessitera plus la pose de câbles ;

c) mais aussi de sécurité, par le caractère robuste inhérent aux réseaux de type L.A.N.

Le projet "Advanced Palmet" développé conjointement par Mazda et Furakawa Electric propose que le lien physique supportant les besoins en communication d'une voiture soit assuré par un réseau local de type L.A.N. utilisant des paires torsadées blindées et ayant une capacité de 1 Mbytes/sec.

Remarque : La spécification de tels réseaux pourrait à elle seule faire l'objet d'une étude mais là n'est pas notre propos. Cependant, pour plus de détails, nous invitons le lecteur à se référer notamment au projet "Advanced Palmet". [KIM,94]

4.2.3.3 Limites

Pour ce qui concerne le support de l'application, il nous semble que l'état actuel de la technologie permet de rencontrer toutes les attentes. Cependant, comme nous l'avons déjà dit précédemment, la collection d'équipements requis est composée pour une partie de produits de très haute technologie. Ce qui nous permet de mettre l'accent sur les risques liés à la fiabilité de tels matériels et donc au niveau de sécurité du véhicule. Notre critique se base sur deux réflexions.

1. D'une part, les formules développées jusqu'à présent (cfr. PalmNet) ne permettent pas, selon nous, de garantir un niveau de performance adéquat. Cela est principalement dû au type de matériel préconisé (Twisted Pairs) qui,

fort de son faible coût, n'en reste pas moins médiocre en qualité de transmission, et cela est d'autant plus vrai que l'on se trouvera dans un environnement soumis à de fortes variations thermiques et autres distorsions électromagnétiques. Certes, on pourrait blinder les câbles et leur adjoindre des algorithmes plus ou moins complexes de correction d'erreurs; mais alors, la dynamique temps-réel s'en trouverait dangereusement entravée.

2. D'autre part, toute défaillance du réseau doit être impérativement proscrite. A l'instar de ce qui se fait en course automobile où le circuit de frein est dédoublé ou encore dans la construction aéronautique où certains câblages sont sextuplés (Cfr. Airbus A320-330), nous pensons qu'il faudrait prévoir un réseau de secours.

De plus, il existe aussi des risques de type économique liés à la rentabilité des matériels les plus coûteux.

C'est pourquoi, nous préconisons l'adoption d'un double réseau en étoile composé de fibres optiques (cfr. figure 4.3.).

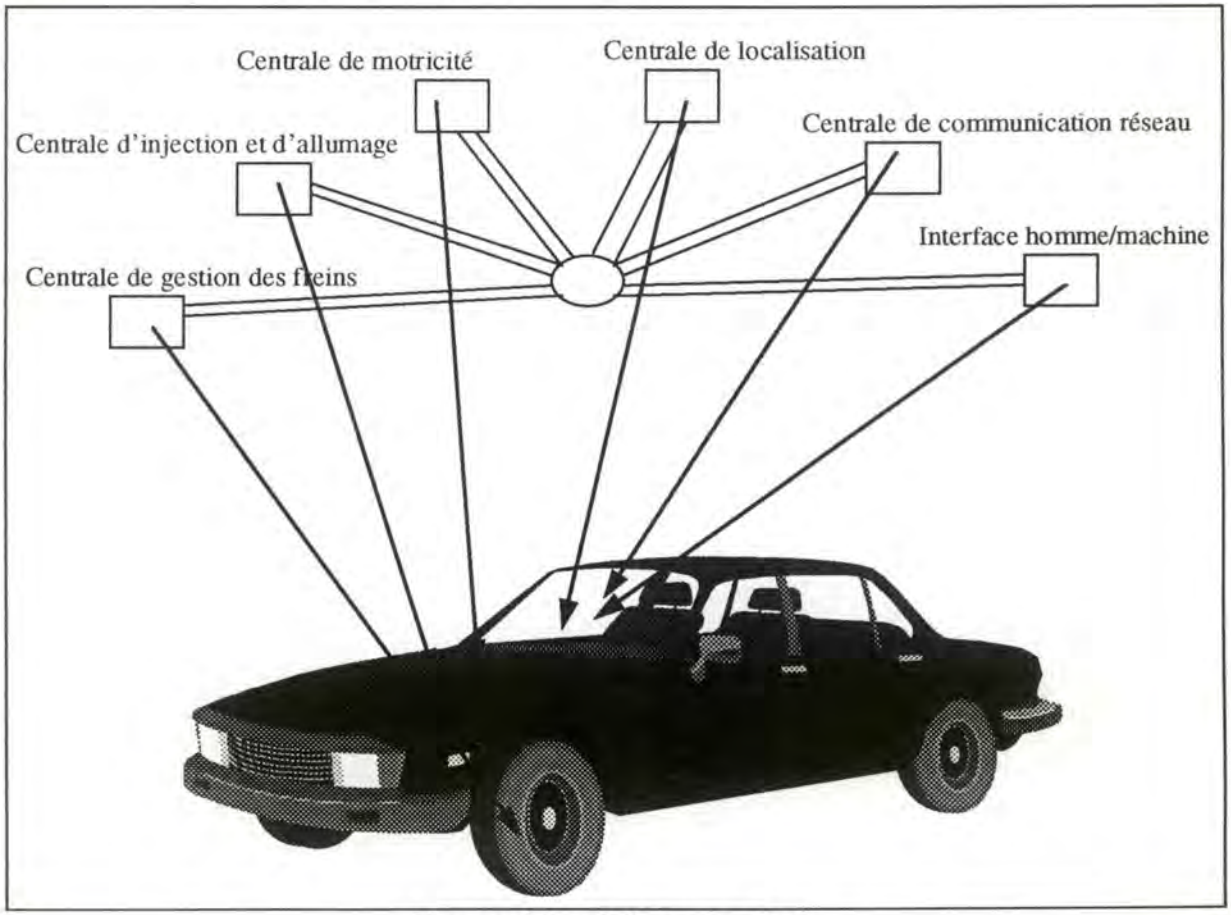


Figure 4.3. : Réseau en étoile

4.2.4 Niveau logiciel

Le niveau logiciel se compose de deux couches. La première est celle de l'Operating System appelé DOS (Distributed Operating System), sur laquelle se greffe la seconde couche : le système de contrôle-commande.

4.2.4.1 Système

Selon une approche hiérarchisée, c'est le niveau le plus bas de l'architecture. Dès lors, un système d'exploitation distribué (DOS) doit proposer :

a) Une large variété de services basiques, liés à la couche système, tels que la protection, la synchronisation, l'intercommunication, la gestion des fichiers, etc.

b) La coordination, au sens large, des éléments de la distribution.

c) L'adéquation de la dynamique du système avec les impératifs dynamiques de la couche applicative. C'est la notion de temps-réel.

En particulier Watson [WATS, 81] caractérisera un DOS par :

"One of the major design goals of a DOS is to provide processes with access to real and abstract objects or ressources in wich the distribued nature of their implementation is hidden as far as practical, and in wich all objects (system or used defined) are names, communicated with, shared and protected uniformly. Real objects are entities such as processors, secondary storage, I/O, devices etc. DOS abstract objects or ressources are entities such as processes, file directories, virtual I/O devices, databases, clocks etc. wich are useful set of basic building blocks for creating higher level objects."

Cette approche nous permet de proposer une vue plus fine des facilités que doit pouvoir supporter la couche système liée à un environnement distribué.

1. Offrir une plate-forme d'extension, facilitant l'adjonction de nouvelles ressources.
2. Donner à l'utilisateur une vue homogène et cohérente du système. C'est la transparence.
3. Proposer des mécanismes de communication tels que le Pipelining mais aussi des modèles liés à la distribution tels que le Client/Serveur.
4. Présenter une méthodologie efficiente d'implémentation.
5. Enfin, supporter des procédures d'administration et de sécurité interdisant par exemple tout accès non autorisé.

Cependant, Watson se réfère à une structuration des données (se référer au point infra 4.4 Approche Objet) dont nous n'avons pas encore fait le postulat. Pour plus de détails, nous vous invitons à la lecture du point 4.4.3 Les O.S. distribués Objets.

4.2.4.2 Contrôle-commande

Il s'agit à proprement parler du système de commande du véhicule, soit un ensemble d'éléments logiciels qui constitue véritablement l'interaction entre les éléments matériels et l'opérateur.

Remarque : L'opérateur humain peut, dans des systèmes hyper-automatisés, être substitué par un système de conduite intégrant des facultés d'adaptabilité et d'apprentissage (cfr. engins spatiaux).

Traditionnellement, dans la littérature, on distingue deux approches du système de contrôle-commande : l'approche fonctionnelle et l'approche comportementale.

4.2.4.2.1 Approche fonctionnelle

L'architecture des systèmes de contrôle-commande de type fonctionnel est basée sur une découpe hiérarchique fonctionnelle. Chaque couche est assignée à la réalisation d'un objectif tel que la perception, la modélisation, l'action, le contrôle, ... Il s'agit d'une découpe d'une fonction décisionnelle (cfr. Mintzberg [MINT,]). Nous pouvons donc identifier les trois sphères fonctionnelles suivantes : la sphère décisionnelle, la sphère exécutive et la sphère matérielle.

1. La **sphère décisionnelle** est représentée par les couches de niveau supérieur du modèle. Il s'agit essentiellement des fonctions de planification et de contrôle du niveau exécutif. Son implémentation est de type rétroactive sur les couches de plus bas niveau liées à l'exécution et fait largement appel aux techniques d'aide à la décision (Recherche Opérationnelle, l'Intelligence Artificielle associées à une interface de qualité avec l'opérateur en terme d'ergonomie avec l'opérateur).

   Il faut aussi prévoir une interface avec la sphère exécutive qui se compose des modules de synchronisation et de supervision.

a) La synchronisation : Il faut pouvoir paramétrer les lois de transition liées au réseau de Pétri de façon à synchroniser l'exécution de primitives.

b) La supervision : Elle permet à l'opérateur (conducteur et/ou système de conduite) de lancer l'exécution de primitives, de les interrompre et d'en connaître le déroulement (OK - Non OK).

2. Le **sphère exécutive,** quant à elle, correspond aux couches de plus bas niveau du système de contrôle-commande où s'exécutent les primitives d'actions sur les organes mécaniques du véhicule et de perception au travers des divers capteurs, ainsi que les injonctions du niveau décisionnel. L'implémentation de ce niveau est du ressort de la robotique. Il a notamment abouti au développement de systèmes d'exploitation distribué (DOS) dans un environnement d'application temps-réel (par exemple : RIPE [MEN, 91]).

   Les éléments principaux appartenant à cette sphère sont un module d'exécution basée sur la logique de Pétri ainsi qu'un module, souvent appelé module de commande, réalisant l'interface avec la sphère matérielle.

3. La **sphère matérielle** est composée de l'ensemble des équipements requis pour l'application (cfr. Support de l'application).

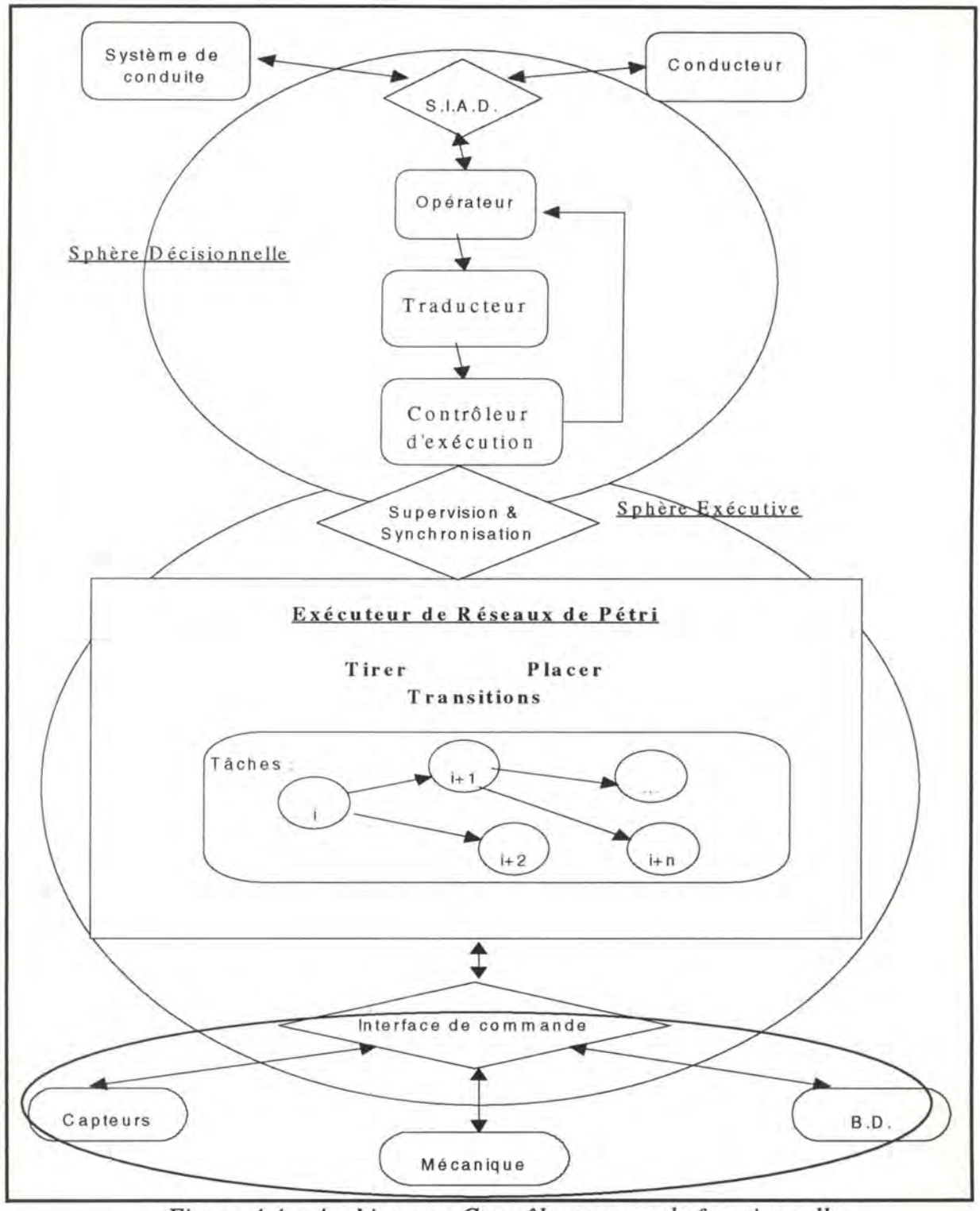


Figure 4.4. : Architecture Contrôle-commande fonctionnelle.

L'architecture appelée **Blackbord** est une application particulière de l'approche fonctionnelle où l'exécuteur de réseau de Pétri se retrouve directement dans la base de données elle-même. C'est donc le gestionnaire de base de données qui déclenche, en fonction d'une logique de places, les différentes procédures. Les procédures

appartenant à un certain niveau fonctionnel n'ont dès lors pas l'initiative de leur communication ni de leur exécution.

Afin d'y remédier, des chercheurs en robotique de l'université de Canergie Mellon, ont développé une architecture de contrôle-commande d'un laboratoire mobile appelé **NavLab** (Navigation Laboratory). Cette architecture se base aussi sur une découpe fonctionnelle, mais cette fois liée à la fonction de pilotage. Elle est composée de cinq sphères, à savoir : le capitaine, la navigation, la perception, le pilotage et le contrôle des organes physiques de conduite et une base de données. Chaque sphère possède ses propres capacités de traitement et interfaces. Sous le contrôle du capitaine, elles ont l'initiative de l'exécution de leurs propres procédures, de sorte que l'ensemble des procédures de l'application s'exécutent en parallèle. Pour ce qui est des traitements nécessitant une coopération, les sphères possèdent aussi l'initiative de la communication via une base de données. L'initiative d'exécution est donc contingentée par le capitaine de sorte que le contrôle s'en retrouve donc hyper centralisé au niveau de la sphère du capitaine qui, en interaction avec le conducteur, représente véritablement le système de conduite de l'application. La base de données apparaît quant à elle comme un goulot d'étranglement.

Il s'agit en fait d'une application du concept de client/serveur avec un serveur exécution de l'application de pilotage automatisé et un serveur de base de données. Cette architecture ne nous semble pas adéquate pour supporter notre application de pilotage. Car, en effet, rappelons encore une fois que ce genre d'application nécessite, pour des raisons de sécurité évidentes, une dynamique proche de celle du temps-réel liée à des impératifs de sécurité, que l'on appelle aussi Hard Real Time.

Il en ressort que les réalisations actuelles (Balckbord et NavLab) liées à l'approche fonctionnelle d'un système de contrôle-commande pour une application Hard Real Time présente des limites rédhibitoires.

4.2.4.2.2 Approche comportementale

Brooks, chercheur au Michigan Institue of Technology nous propose ici une approche pour le moins originale qui se base sur des études comportementales effectuées sur des insectes. On y retrouve une découpe hiérarchisée où les couches correspondent à des niveaux comportementaux. Chaque comportement est la résultante d'une réaction à un ou des stimuli. Il n'y a donc pas de système de conduite ou de planification; tout au plus, on définit quelles sont les réactions produites par le système qui reçoit des événements extérieurs et/ou intérieurs. Il ne faut dès lors pas parler d'actions mais plutôt de réflexes et d'architecture réactive. On effectue une boucle de réactions sur la perception.

Imaginons que le modèle présente plusieurs comportements différents pour un même niveau, il se pose alors le problème de savoir quel comportement il faut adopter en fonction d'un événement. Brooks propose d'y remédier en affectant un et un seul comportement par niveau. Il n'existe dès lors pas de comportement ayant le même degré hiérarchique. Cette découpe hiérarchique constitue une fonction strictement croissante sur l'ensemble des comportements. Chaque comportement est implémenté comme une machine à états finis étendus. Il s'agit en fait d'une extension d'une machine à états finis à laquelle on greffe une série de registres et une horloge interne. Les messages sont stockés dans les registres. Il est possible d'interconnecter plusieurs machines à états finis étendus. Chaque comportement possède la propriété d'inhiber une de ses sorties, voire même détruire une de ses entrées (voir figure 4.5.).

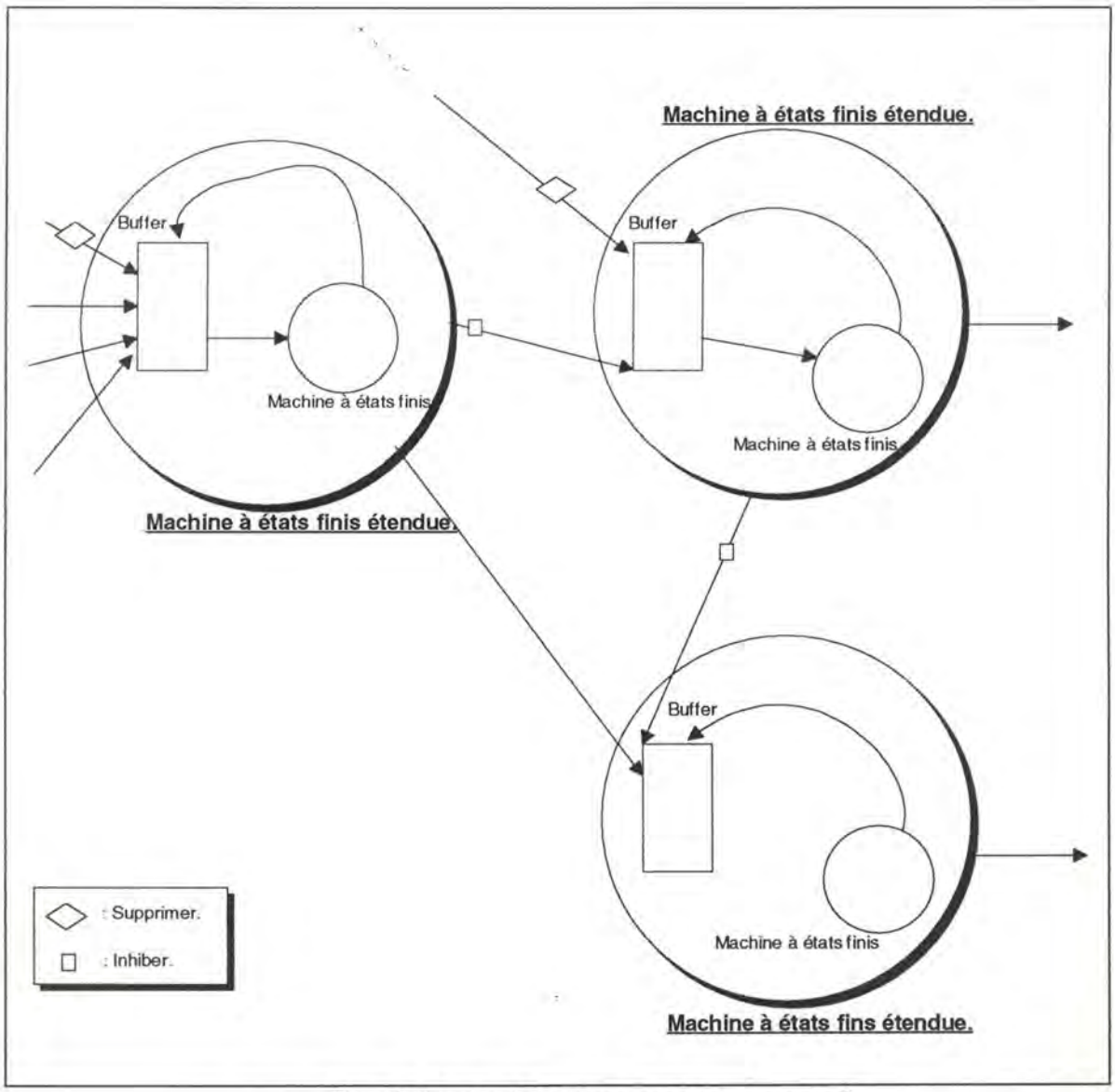


Fig. 4.5. : Architecture comportementale.

4.3 Modèles et mécanismes des communications classiques

4.3.1 Introduction

Lorsque l'on désire utiliser simultanément plusieurs processus, et donc effectuer des traitements en parallèle, nous devons disposer des moyens suivants :

a) une entité pour représenter une activité séquentielle (processus, tâche);

b) un logiciel d'ordonnancement des activités (scheduler);

c) des modèles et des protocoles de communication pour permettre l'échange d'informations entre deux entités parallèles.

Les systèmes d'exploitation classiques, qu'ils soient dotés de capacités de traitement Temps-Réel ou non, offrent toujours les trois facilités exposées ci-dessus. Malheureusement, la mise en oeuvre efficace d'applications qui impliquent l'exécution simultanée de plusieurs processus demande des compétences très poussées à cause du bas niveau des mécanismes de communication et de synchronisation offerts par les différents systèmes.

Puisqu'un de nos objectifs est de faciliter la programmation d'application Temps-Réel, il convient dans un premier temps d'analyser les problèmes engendrés par la communication entre entités parallèles et de montrer les difficultés qu'un programmeur peut rencontrer lorsqu'il désire élaborer des telles applications.

Quand plusieurs processus s'exécutent simultanément afin de réaliser les différentes tâches d'une application, il est possible que ceux-ci entrent en concurrence dans l'usage des ressources disponibles. Nous dirons alors que les relations qu'ils entretiennent sont d'ordre **conflictuel**. Il sera donc nécessaire de gérer les conflits d'accès aux ressources partagées en offrant des mécanismes qui permettent d'assurer l'intégrité des données lors de leur partage. Les processus n'entretiennent cependant pas que des relations d'ordre conflictuel. En effet, il est courant qu'ils participent à la réalisation d'une tâche globale. Dans ce cas, ils doivent coopérer entre eux afin de réaliser cette tâche. Les relations qu'ils entretiennent alors sont des relations de **coopération**. Des mécanismes de synchronisation et de communication sont alors nécessaires afin de maintenir la cohérence des actions entreprises par chaque activité.

Suivant l'architecture des systèmes sur lesquels on travaille, les modèles de communications entre processus peuvent revêtir deux formes[1] : la communication par **données partagées** ou la communication par **échange de messages**.

Dans le cas de la mémoire partagée, plusieurs entités (ou processus) partagent la même structure de données (ou zone mémoire), ils y accèdent en lecture ou en écriture et ce de manière concurrente. Dans un schéma de communication par échange de messages, un processus émetteur crée un message contenant l'information à transmettre et l'envoie, suivant les modalités du modèle d'échange de message, à un ou plusieurs processus destinataires[2].

Quel que soit le modèle utilisé, des protocoles de communication et de synchronisation doivent être définis afin d'assurer le bon déroulement des échanges. Ces protocoles peuvent être de deux classes : **synchrone** ou **asynchrone**. Dans le cas d'un protocole synchrone, un processus émetteur transmet une information et attend un accusé de réception de la part du ou des destinataires avant de poursuivre son activité. Un échange asynchrone, quant à lui, permet à l'émetteur de continuer son activité sans attendre la prise en compte de l'information transmise par le(s) destinataire(s).

Puisque les notions de bases sont maintenant établies, nous pouvons aborder l'analyse des mécanismes de communication et étudier plus en profondeur les problèmes que nous risquons de rencontrer lors de leur mise en oeuvre.

4.3.2 Communication par données partagées

4.3.2.1 Principe

Dans un tel contexte, la ou les données, ne s'échange(nt) pas pour ainsi dire ! En effet, le principe de données partagées utilise le fait que l'on puisse partager la mémoire (et donc les données qui y sont inscrites) entre différents processus. On peut donc classer les entités interagissant au moyen de ce mécanisme en deux catégories :

- les processus lecteurs qui se contentent comme leur nom l'indique de lire les données stockées en mémoire ;
- les processus écrivains qui affectent l'état de la mémoire partagée en y inscrivant (ou simplement en mettant à jour) des données.

[1] Cette distinction n'est pas exclusive. Certains systèmes offrent les deux modes de communication : par messages et par mémoire partagée. (Par exemple, les IPC du système V d'UNIX permettent d'utiliser le mécanisme de la mémoire partagée et celui des messages.)

[2] En fait, l'émetteur transmet le message au système de communication qui l'achemine de manière explicite vers les destinataires.

Remarquons cependant que ces catégories ne sont pas exclusives : un processus peut être tantôt lecteur, tantôt écrivain.

La rapidité d'exécution d'un tel mécanisme d'échange de données en fait un outil intéressant pour les applications en temps-réel. Puisque ce mécanisme est par nature asynchrone et non bloquant, il en résulte un coût de gestion de la mémoire partagée. Il faut en effet pouvoir régler les conflits d'accès des différents processus désirant accéder à ces données afin de conserver l'intégrité et la cohérence de ces dernières. Nous allons donc voir comment opérer une telle gestion en repérant les causes de conflits pour ensuite y apporter une solution adéquate.

4.3.2.2 Problèmes et solutions

Tentons d'imaginer les conséquences de la situation suivante : deux processus accèdent en même temps à la donnée partagée. Si ce sont deux processus lecteurs, il n'y aucun problème puisqu'ils n'affectent pas l'état de la mémoire. Par contre, si au moins un des deux est un processus écrivain, il risque d'affecter l'intégrité et/ou la cohérence des données. La séquence d'instructions où se produit ce conflit est appelée **section critique**. Pour éviter de tels conflits d'accès, il faut disposer d'un mécanisme permettant de réaliser une exclusion mutuelle grâce à laquelle un processus travaillant sur les données partagées en aura l'exclusivité. Ce mécanisme empêchera alors tout autre processus d'accéder aux données pendant leur utilisation.

Le mécanisme d'exclusion mutuelle peut être décrit par deux primitives atomiques[1] : *ENTRER_SECTION_CRITIQUE* et *SORTIR_SECTION_CRITIQUE* bordant la section critique. La première primitive, appelée juste avant d'entrer dans la section critique, interdit à tout autre processus d'accéder aux données partagées. Lorsque notre processus aura exécuté son travail sur les données, et donc accompli la section critique, la seconde primitive avertira les autres processus qu'ils peuvent entrer en section critique et donc accéder aux données partagées.

Maintenant que nous disposons de ce mécanisme, d'autres problèmes risquent de surgir : l'**interblocage** et la situation dite de **famine**. En effet, si le mécanisme d'exclusion mutuelle permet d'assurer la cohérence et l'intégrité des données en allouant l'exclusivité d'accès à un processus, elle ne résout pas tous les problèmes de partage. Au contraire, elle en induit même de nouveaux (mais d'une autre nature). Nous avons vu que, quand un processus exécute une région critique, il bloque, par l'intermédiaire de la primitive *ENTRER_SECTION_CRITIQUE*, les autres processus désirant accéder aux données partagées. Si notre processus ne sort jamais de la région critique[2], il ne pourra pas effectuer la primitive *SORTIR_SECTION_CRITIQUE* et les autres processus attendront indéfiniment

[1] L'atomicité garantit au processus qu'il ne sera pas interrompu lors de l'exécution de cette primitive.
[2] Le processus peut être détruit accidentellement (tué) ou simplement attendre la libération d'une ressource détenue par un des processus bloqué devant la mémoire partagée.

qu'il libère la mémoire partagée. Voilà un cas d'interblocage ! Nous pouvons en déduire une définition plus rigoureuse [RAM ,90] : « L'interblocage est une situation où un processus est définitivement bloqué en attente d'une ressource possédée par un autre processus lui-même définitivement bloqué en attente d'une ressource possédée par le premier.» Les contraintes nécessaires pour que surviennent l'interblocage sont les suivantes :

- l'exclusion mutuelle ;
- quand un processus est bloqué, il garde toutes les ressources précédemment acquises ;
- la non-préemption[1].

La situation de famine est liée aux priorités des processus et à la politique d'ordonnancement de ceux-ci lorsqu'ils attentent la libération d'une ressource (ici, la mémoire partagée). Il se peut en effet qu'un processus de faible priorité n'accède jamais à la ressource partagée. Pour ce faire, il suffit qu'il existe toujours au moins un autre processus de plus haute priorité demandeur de la même ressource et que les demandes d'entrée en section critique ne soient pas mémorisées selon leur ordre d'arrivée.

Ayant à l'esprit les problèmes d'interblocage et de famine, nous pouvons maintenant chercher une solution pour implémenter nos deux primitives d'exclusion mutuelle.

Une première solution [RAM 90] consiste à utiliser un tableau de variables de contrôle : **c[1:n]** et une variable **r** . Chaque élément du tableau est une variable de contrôle pour un processus. La variable **r** contiendra l'identificateur du processus qui travaille sur la ressource partagée, c'est-à-dire son indice dans le tableau. Ainsi, si le processus P_i utilise la ressource (ici, la mémoire partagée), **c[i]** = 2 et **r** = i. A la demande d'entrée dans la section critique, donc dans la primitive *ENTRER_SECTION_CRITIQUE*, on positionne **c[i]** à 1. L'admission dans la section critique est réalisée par un parcours circulaire du tableau **c[1:n]**.

[1] « La ressource n'est réallouée qu'après avoir été libérée par le processus qui l'avait précédemment acquise.» [RAM ,90]

Au niveau global l'algorithme est le suivant :

```
Tableau d'entiers c [ 1 : nproc ] dont chaque cellule est initialisée à 0;
Pointeur r initialisé à 0;

ENTRER_SECTION_CRITIQUE :

        i1 :    c[i] ← 1 ; /* le processus signale son arrivée */
        i2 :    Pour j = r Par pas de 1 Jusque n puis Pour j = 1 Par pas de 1 Jusque r-1 Faire
                {
                    Si j = i Alors i3 ; /* Si le processus n'a pas de prédécesseur dans la file, il rentre dans
                                           la section critique */
                    Si c[j] ≠ 0 Alors i2 ; /* Sinon, il boucle sur l'examen de ses prédécesseurs */
                }
        i3 :    c[i] ← 2 ; /* Il signale qu'il est entré */
        i4 :    Pour j = 1 Par pas de 1 Jusque n Faire
                {
                    Si  (j ≠ i) Et (c[j] = 2) Alors i1 ; /* Il renonce à poursuivre si un autre que lui est déjà
                                                            dans la section critique.  Il recommence alors sa
                                                            requête. */
                }
        i5 :    r ← i ; /* Son numéro est affiché */

SECTION CRITIQUE : {...}

SORTIE_SECTION_CRITIQUE :

i7 :    Si i = n Alors r ← 1 /* Après sa sortie de la section critique, il affiche la nouvelle valeur
                                d'examen du tableau circulaire */
        Sinon r ← i + 1 ,
i8 :    c[i] ← 0 ; /* Il réinitialise à 0 sa variable de contrôle */
```

Il est aisé de démontrer que :

- ***Il y a au plus un seul processus dans la région critique ...***

  Car tout processus P_i dans la section critique a dû exécuter la primitive *ENTRER_SECTION_CRITIQUE* et donc les instructions i4 et i5. Or, pour franchir le test i4, il faut qu'au moment de l'exécution, tous les c[j] soient différents de 2. Il faut cependant tenir compte du fait que plusieurs processus peuvent exécuter i4 en même temps. Mais, ces autres processus ont nécessairement exécuté i3 après P_i. Ils trouvent alors c[i] = 2 lors de l'exécution du test i4 et sont renvoyés à i1.

- ***Aucun processus n'attend indéfiniment ...***

  Puisque le lieu d'attente est constitué par la boucle i1, i2 et que, si aucun nouveau processus n'a entamé la procédure d'entrée, les processus en attente ne peuvent rester indéfiniment dans cette boucle.

> Par contre, « si de nouveaux processus sont arrivés, le processus ne peut être gêné que par ceux qui se sont interposés entre lui et le processus alors dans la section critique. Ceux-là passeront avant lui mais, comme à chaque sortie, l'origine de l'examen se rapproche de lui ...» (cfr. i7) « le processus en attente est assuré de pouvoir pénétrer lui-même dans la section après un temps limité. » [RAM 90]

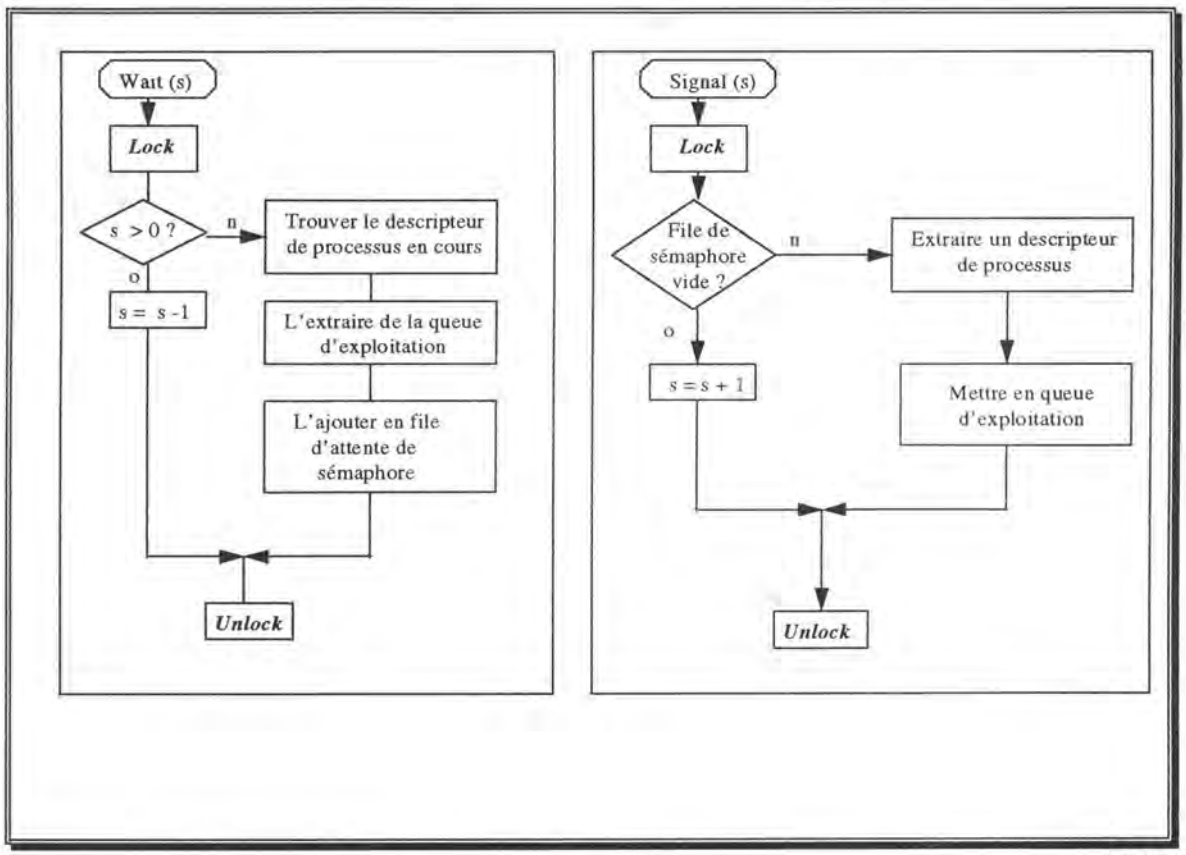
La solution que nous venons de présenter est peut-être un peu lourde et complexe, mais elle a le grand avantage d'être générale dans le sens où elle n'impose aucune contrainte quant au mode d'exécution des primitives. De plus, les instructions utilisées dans l'algorithme se retrouvent sur tous les systèmes puisque tout système propose toujours des instructions de test, d'affectation et de saut. Les tableaux sont aussi un standard que l'on retrouve partout. Ces deux dernières remarques en font un outil puissant et, ce qui n'est pas négligeable, extrêmement portable.

Il existe bien d'autres solutions pour régler les problèmes d'exclusion mutuelle. Nous n'en présenterons cependant qu'une seule : la solution de Dijkstra. Dijkstra proposa d'associer à chaque ressource utilisée par les processus une variable entière et positive appelée « **sémaphore** ». La valeur initiale de cette variable rend compte du nombre de processus pouvant l'utiliser simultanément. Au fur et à mesure que des processus emploient ou délaissent la ressource, le sémaphore diminue ou augmente. Chaque processus désirant utiliser la ressource doit alors vérifier la valeur contenue dans le sémaphore qui y est rattaché afin de savoir s'il a le droit ou non d'accéder à la ressource en question. Dans le cas précis de l'exclusion mutuelle, le sémaphore doit être initialisé à 1.

La valeur d'un sémaphore s ne peut être modifiée que par les deux opérateurs **Wait** (s) et **Signal** (s). L'opérateur Signal accroît la valeur du sémaphore s d'une unité. L'accroissement est indivis[1]. L'opérateur Wait, quant à lui, décroît la valeur du sémaphore d'une unité à condition que le résultat ne devienne pas négatif. Cette condition nous permet d'approcher de plus près le mécanisme d'utilisation des sémaphores. Nous avons dit plus haut que les sémaphores étaient des variables attachées à une ressource ... Ils ne sont pas que cela[2] ! En effet, si un processus exécute l'opération Wait sur un sémaphore dont la valeur est nulle, la valeur du sémaphore ne change pas et le processus doit attendre que l'opération Wait soit réalisable. L'opération Wait implique donc que les processus soient bloqués quand un sémaphore a la valeur zéro et soient libérés quand une opération signal augmente la valeur d'une unité. La façon la plus naturelle de réaliser ceci est d'associer à chaque sémaphore une file d'attente de processus. Voici les deux algorithmes :

[1] C'est à dire dans le cas où plusieurs procesus sont en atttente, un seul processus peut effectuer l'opération wait ou signal. Autrement dit, l'exécution de wait ou signal par un processus est ininterruptible.

[2] Pour être rigoureux nous devrions dire que le sémaphore est la variable positive attachée à une ressource et que son emploi pour contrôler l'accès à cette ressource nécessite l'utilisation d'une file d'attente. Dans la pratique cependant, sémaphore et file d'attente se combinent pour former l'outil de contrôle communément appelé sémaphore.



[LIS,90]

Afin d'assurer l'indivisibilité des deux opérateurs, il convient d'avertir le système que, lorsqu'un processus effectue une action sur un sémaphore, il ne peut être interrompu. Voilà pourquoi les deux primitives sont bordées des instructions ***Lock*** et ***Unlock***. Ces deux instructions réalisent une exclusion mutuelle des processus pour Wait et Signal, mais, contrairement aux sémaphores, leur niveau de réalisation relève du matériel. Dans un environnement monoprocesseur, la réalisation de Lock et Unlock consiste à simplement masquer les interruptions par le biais de l'instruction Lock et à réactiver ces dernières par l'appel Unlock. Par contre, dans un système distribué, la neutralisation du mécanisme d'interruption est inutile puisque deux processus peuvent exécuter Wait et Signal sur des processeurs différents. Dans ce cas, il faut disposer d'un mécanisme hardware particulier : l'instruction *test and set*. Ce mécanisme, comme son nom l'indique, permet de tester et de modifier le contenu d'une zone mémoire en une seule instruction. De plus, durant l'exécution de cette instruction, il est interdit à tout autre processus d'accéder à cette zone mémoire. En fait, ce mécanisme revient simplement à utiliser la zone mémoire particulière comme un sémaphore autorisant ou non l'entrée dans les procédures Wait et Signal.

Maintenant que nous disposons des sémaphores, il est aisé de réaliser l'exclusion mutuelle pour l'accès à la mémoire partagée :

```
sémaphore s initialisé à +1 ;

Wait (s) ;
{
 section critique
}
Signal (s);
```

4.3.2.3 Le modèle Lecteurs/Ecrivains

Considérons un protocole d'accès à une donnée partagée selon le principe de l'exclusion mutuelle stricte. A savoir : aucun processus ne peut accéder à la mémoire partagée si un processus y travaille déjà. Cette restriction détériore grandement les performances de la communication par échange de données car, en effet, deux processus lecteurs peuvent très bien consulter la mémoire partagée sans induire d'incohérence. Le protocole considéré est satisfaisant pour des modèles où la nature de la tâche qu'effectue un processus est inconnue, ou encore, lorsque les processus sont tantôt lecteurs, tantôt écrivains. Par contre, si l'on connaît a priori le type de travail que va effectuer un processus désirant accéder à une mémoire partagée, il est intéressant de considérer un autre protocole d'accès, permettant notamment à plusieurs processus lecteurs d'accéder en même temps à la donnée partagée. Plusieurs variantes d'un tel protocole existent et ont comme points communs les trois règles suivantes [Pad 90] :

- plusieurs lecteurs peuvent consulter en parallèle une donnée partagée ;
- un processus écrivain doit avoir l 'accès exclusif à la donnée partagée ;
- lorsque la donnée partagée est libre (aucun processus n'y accède), un lecteur et un écrivain ont la même priorité.

Quatre primitives permettent donc de décrire les différentes variantes d'un tel protocole : *Demander_lecture*, *Terminer_lecture*, *Demander_écriture*, *Terminer_écriture*. Grâce à ces procédures, nous pouvons décrire les algorithmes suivants :

```
Lecteur ( )
{
        Demander_lecture (ressource);
        /* ... Lecture */
        Terminer_lecture (ressource);
}

Ecrivain ( )
{
        Demander_écriture (ressource);
        /* ... Ecriture */
        Terminer_écriture (ressource);
}
```

Ces procédures permettent de différencier les processus lecteurs des processus écrivains en laissant plusieurs processus lecteurs accéder à la mémoire partagée en même temps alors que leurs homologues écrivains se contentent d'un protocole d'accès strict[1]. Elles ne règlent cependant pas les problèmes de famine ou d'interblocage. Ces derniers sont en effet propres à l'utilisation de sections critiques. On peut cependant les résoudre facilement par la réalisation de procédure d'ordonnancement des processus en attente d'accès à la donnée partagée, ce qui conduit à la définition de plusieurs variantes de ce modèle. Parmi ces variantes, nous avons :

1) **PRIORITE AUX LECTEURS** : si la ressource partagée est occupée par un processus lecteur, toute demande de lecture est prioritaire sur une demande d'écriture. Cette politique peut évidement conduire à des situations de famine pour les processus écrivains.

2) **PRIORITE AUX LECTEURS SANS FAMINE DES ECRIVAINS** : en fin d'écriture, tous les lecteurs en attente sont satisfaits, même s'ils sont arrivés après un écrivain. Par contre, pendant une lecture, toutes les demandes de lecture arrivant avant une demande d'écriture sont satisfaites, alors que, si un lecteur arrive après un écrivain, il est mis en attente jusqu'à la fin de l'écriture.

3) **PRIORITE AUX ECRIVAINS** : cette solution est la symétrique de la solution 1 et peut conduire à une famine des lecteurs.

4) **PRIORITE AUX ECRIVAINS SANS FAMINE DES LECTEURS** : cette solution est la symétrique de la solution 2.

[1] Cette différenciation peut être réalisée grâce à un jeu de sémaphores habilement initialisés. L'accès en région critique pour les écrivains étant réglé par un sémaphore SE initialisé à 1, alors que l'accès en région critique pour les lecteurs est réglé par un sémaphore SL initialisé au nombre de lecteurs. Remarquons que les processus lecteurs doivent pouvoir agir sur le sémaphore SE afin d'éviter à un écrivain d'accéder à la donnée partagée alors qu'un ou plusieurs lecteurs sont en région critique.

5) **LA POLITIQUE FIFO** : les demandes d'accès sont stockées selon le mode First In, First Out (premier entré, premier sorti) tout en permettant à plusieurs lecteurs consécutifs d'accéder en même temps à la mémoire partagée.

Différentes politiques donneront des résultats différents. Ainsi, certaines solutions sont satisfaisantes dans certains cas et d'autres non. Leur implémentation est plus ou moins aisée suivant les mécanismes d'exclusion mutuelle présentés aux paragraphes précédents puisque le problème consiste à partager équitablement les données tout en garantissant leur cohérence.

4.3.3 Communication par échange de messages

4.3.3.1 Principe

Dans le cas d'échange d'informations par mémoire partagée, nous avons vu qu'en fait il n'y a pas d'échange proprement dit mais seulement un accès en lecture ou en écriture à une donnée commune stockée en un endroit connu des différents processus désirant l'utiliser. Nous allons maintenant voir comment réaliser un échange véritable, c'est-à-dire, comment réaliser un transfert réel d'information entre différents processus. Dans ce cas, l'information est encapsulée dans un message qui est transmis explicitement par un processus émetteur à un ou plusieurs processus destinataires.

Insistons dès à présent sur le fait que les processus communicants ne sont pas obligés d'être localisés sur un même processeur. En effet, différents processeurs peuvent être reliés par une liaison quelconque de façon à ce que les processus communicants soient localisés sur ces différents processeurs et communiquent via des messages transitant sur un réseau.

Quelle que soit la situation dans laquelle on se trouve : LAN[1], WAN[2], réseau à diffusion (RDS) ou encore communication par satellite (GPS) ... les systèmes de communications sont généralement construits sur le principe de couches hiérarchiques. Ces couches, plus ou mois nombreuses suivant les protocoles utilisés, permettent une abstraction des traitements réalisés dans les couches inférieures. Ainsi, la couche la plus externe offre une interface virtuelle très proche de l'utilisateur, alors que les couches les plus basses réalisent les fonctions fondamentales du système de communication - émission et réception de messages - en se basant directement sur les caractéristiques du matériel et de l'O.S. utilisés.

[1] Local Area Network ou réseau local.
[2] Wide Area Network ou réseau de taille large (± 100 km)

4.3.3.2 Le modèle producteur / consommateur

Ce modèle met en jeu des processus producteurs d'information qui sont généralement des émetteurs de messages et des processus consommateurs qui sont généralement les destinataires des messages.

Dans le schéma producteur/consommateur classique, l'émetteur produit l'information, l'encapsule dans un message, puis demande au système de communication que ce message soit communiqué à un ou plusieurs processus consommateurs, et ceci à l'aide d'une primitive « *envoyer* ». Le système stocke le message dans une mémoire tampon pour le délivrer par la suite aux consommateurs à l'appel d'une primitive « *recevoir* ».

Les primitives d'envoi et de réception peuvent être bloquantes ou non bloquantes, ce qui permet de classer les protocoles d'échange de messages en quatre modes.

1) Les primitives d'envoi et de réception des messages sont bloquantes

La communication se fait alors sur « rendez-vous », c'est-à-dire que le premier processus arrivé au point de synchronisation attend l'autre. Puisque les deux processus sont en place au moment de l'échange, il n'y a pas besoin de tampon pour stocker les messages.

Ce type de communication permet de concevoir des applications fortement structurées où les instants de communication sont bien connus car déterminables à priori. Par contre, le parallélisme est ici extrêmement limité puisque l'architecture de l'application est complètement figée[1]. De plus, la systématisation des appels avec réponse alourdit les échanges.

2) Les primitives d'envoi et de réception de messages sont non bloquantes

Dans ce cas, une mémoire tampon est nécessaire pour stocker les messages en attente. A l'émission d'un message, ce dernier est stocké dans la mémoire tampon, et la réception d'un message consiste à le récupérer dans cette mémoire.

L'avantage de cette solution réside dans son parallélisme qui est total. Il peut cependant être intéressant de vouloir contrôler le flux des messages, notamment pour des applications Temps-Réel. D'autres mécanismes de synchronisation doivent alors être mis en oeuvre pour permettre aux processus de coordonner leurs actions.

[1] Les modules de communication sont entièrement basés sur le principe de la synchronisation des différents processus à des moments précis. Il en résulte que la moindre modification peut induire des remaniements importants de l'application.

3) La primitive d'envoi est bloquante et la primitive de réception est non bloquante

Ce modèle permet de ne pas bloquer le récepteur si l'émetteur n'est pas prêt. En effet, dans une telle situation, l'émetteur d'un message est bloqué jusqu'à ce que le récepteur l'ait pris et le récepteur ne peut bien sûr recevoir un message que si l'émetteur est prêt à l'envoi.

4) La primitive d'envoi est non bloquante et la primitive de réception est bloquante

A l'émission d'un message, ce dernier est stocké dans une mémoire tampon, et l'appel de la primitive de réception a pour effet de bloquer le processus appelant jusqu'à ce qu'un message soit déposé par un émetteur.

4.3.3.3 Le modèle client / serveur

Le modèle client/serveur permet à deux processus de communiquer de façon similaire à des appels de procédures.

Dans ce modèle, un processus client envoie une requête, sous forme de message, à un processus serveur qui effectue un travail et renvoie une réponse. Ce modèle est synchrone car le client attend la réponse du serveur pour continuer son activité. De plus, plusieurs clients peuvent solliciter un même service et donc envoyer des requêtes au serveur. Dans ce cas, les requêtes reçues par le serveur sont stockées dans une file d'attente pour être traitées, par la suite, selon un ordre précis (FIFO ou préférence suivant l'importance relative des processus ...).

La similitude de ce modèle avec un appel de procédure classique ne se limite pas uniquement au modèle de communication, mais va jusqu'à l'utilisation d'une syntaxe équivalente. L'envoi d'une requête par un client se résume alors à l'exécution d'une procédure. L'exploitation de ce modèle dans un environnement distribué nécessite donc l'implantation d'un appel de procédure à distance, de telle sorte que pour demander un service distant, un client exécute une procédure sans se préoccuper de la génération, l'acheminement et la réception des messages. Toute cette partie est prise en charge par le système de façon à ce que ces traitements soient transparents au client. Des processus intermédiaires, utilisant le modèle de communication par messages, sont alors nécessaires et sont créés soit par un appel de procédure spécifique, soit par un compilateur. De même, du point de vue du serveur, une procédure est déclenchée par un appel classique, permettant la réalisation du service demandé. Ce dernier n'a donc, lui non plus, aucune connaissance du traitement des messages.

Remarquons toutefois que l'implantation de l'appel de procédure à distance pose cependant quelques problèmes et notamment en ce qui concerne le passage des arguments qui ne peut se faire que par valeur. En effet, dans une architecture

distribuée sans mémoire partagée, une adresse sur un site n'a aucune signification sur un autre site, surtout si les processeurs sont hétérogènes.

4.3.3.4 Limites

Il va sans dire que le modèle d'échange de messages est bien adapté aux architectures distribuées. Les inconvénients de ce modèle apparaissent lorsque les processus communicants s'exécutent sur une même machine ou sur deux processeurs ayant une zone de mémoire commune. Dans ces deux derniers cas, tous les intermédiaires nécessaires à une communication par échange de messages tels que la constitution du message d'appel et du message de réponse, le stockage intermédiaire dans un tampon, l'envoi et la réception explicite de messages, font que le coût de la communication est augmenté de façon inutile, détériorant alors les performances du système de communication et donc aussi celles de l'application. Or, les "véhicules intelligents" sont confrontés à des contraintes temps-réel nécessitant un haut niveau de performance (fiabilité, rapidité). De plus, le caractère hétérogène et évolutif des matériels utilisés impose une architecture extensible qui réduira fortement les coûts de mise en oeuvre. Les mécanismes présentés ne permettent malheureusement pas en l'état de réaliser une telle architecture de façon efficiente et au meilleur coût.

4.4 Approche Objet

L'idée novatrice de ce concept est de regrouper une structure de données (langage de classe) et l'ensemble des procédures susceptible de la manipuler, au sein d'une structure innovante de haut niveau appelé Objet (langages d'acteurs).

Alors que les langages de programmation classiques de type procéduraux, tels que le Pascal, définissent un programme comme une suite de procédures, les langages à Objets, quant à eux, proposent une vue plus dynamique en considérant les programmes comme des ensembles d'Objets indépendants dont on définit le comportement et qui communiquent entre eux (langages d'acteurs).

Remarque : Outre la plus grande puissance des méthodologies de développement orienté Objet, l'un des attraits de ces langages est que nous pouvons trouver sur le marché des versions, telles que le Pascal Objet ou encore le C++, supportant bien évidemment le développement de structures à Objets mais aussi la récupération de librairies classiques de compétences procédurales.

4.4.1 Langages de Classes

L'aspect structurel des langages à Objets impose la vue d'un Objet comme une structure de données connaissant un certain comportement. Tous les Objets partageant la même structure et le même comportement appartiennent à une même Classe. Les langages de classes sont généralement caractérisés par **trois notions basiques** : l'instanciation, l'héritage et la communication.

1. **Instanciation**

   Un Objet est une expression physique appelée "instance" du modèle défini par la classe.

2. **Héritage**

   Nous pouvons introduire une structure de graphe dans la définition des classes. Ainsi, une classe de niveau supérieur appelée Mère peut léguer ses propriétés à une classe de niveau inférieur appelée Fille. Celle-ci peut raffiner son héritage par l'adjonction de nouvelles propriétés. Si la classe Fille ne connaît qu'une seule Mère, nous parlerons dès lors d'héritage basé sur une arborescence, sinon il s'agit d'un graphe orienté acyclique.

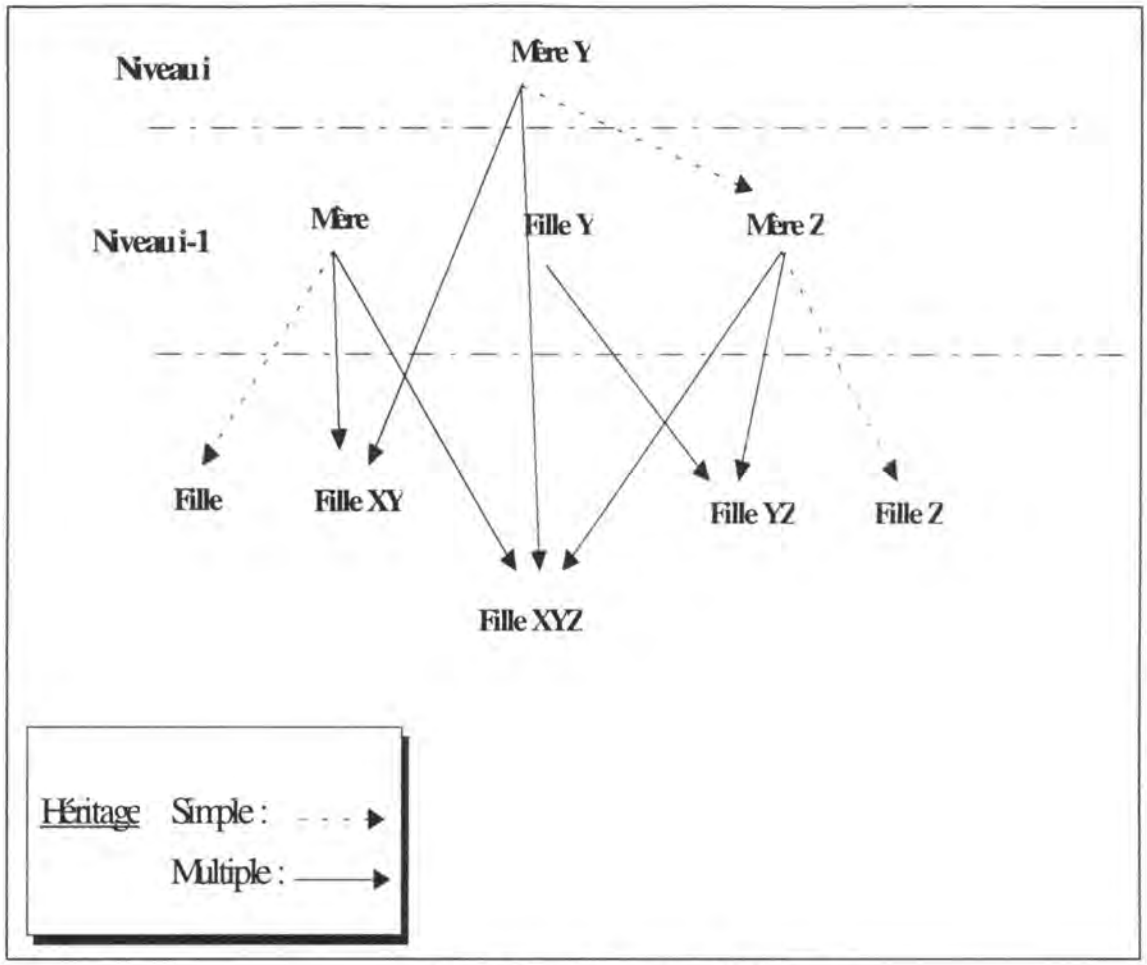


Fig. 4.6. : Exemple d'héritage.

3. Communication

Les Objets communiquent entre eux selon le principe du producteur/consommateur et sont contrôlés par l'envoi synchrone de messages. Comme le montre la figure 4.7. ci-dessous, la communication repose sur des mécanismes d'invocation, locale à un Objet consommateur, des méthodes publiques d'un Objet producteur d'informations, selon un protocole qui est synchrone. Les méthodes publiques de cet Objet sont donc exportées localement vers l'objet consommateur, nous parlerons dès lors d'Objets Exportés. Il est à noté qu'un Objet peut être, selon les cas, producteur ou consommateur d'informations voire même les deux.

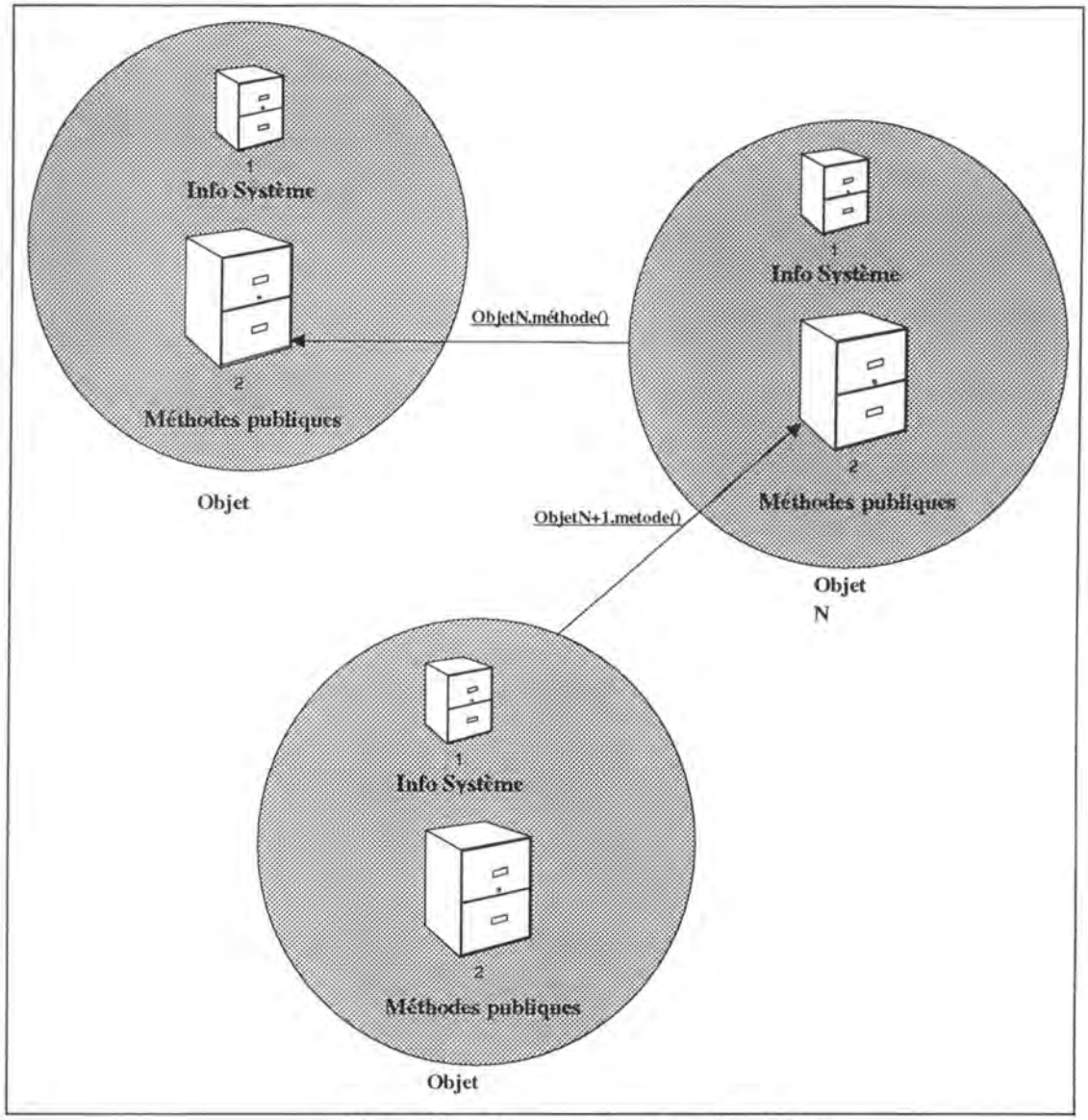


Fig. 4.7. : Communication des Objets.

Les langages de classes, ainsi définis, connaissent **trois grandes propriétés** en terme d'ingénierie logicielle.

1. L'encapsulation

Permettre de regrouper, d'encapsuler au sein d'une entité communicante, des données et les procédures y afférentes. Ce qui offre au développeur un niveau important d'abstraction des données, ainsi qu'une grande modularité de l'architecture, assurant entre autres le caractère réutilisable des Objets.

2. Le polymorphisme

Nous distinguons deux formes de polymorphisme : le Polymorphisme de fonction par lequel il est possible d'appliquer une même fonction à des arguments ayant

des types différents et le Polymorphisme d'héritage où une instance de classe Mère peut être assimilée à une instance de classe Fille.

3. L'extensibilité

Les propriétés d'encapsulation et de polymorphisme permettent aux langages de classes d'offrir un fort degré d'extensibilité. En effet, pour spécialiser ou restreindre le champ d'une application, il suffira de créer une instance Fille connaissant un comportement plus restrictif que l'instance Mère.

4.4.2 Langages d'Acteurs

Nous l'avons dit, un Objet est aussi caractérisé par son comportement, il s'agit de l'aspect actantiel du langage à Objets, où l'Objet est vu comme un acteur autonome et indépendant communiquant par voies asynchrones.

C'est dans le cadre de recherches sur le thème de l'Intelligence Artificielle que le roboticien C. Hewit a le premier proposé le concept d'acteur. Ce modèle lui permit d'apporter la notion de distribution au problème complexe de représentation de la connaissance d'un robot.

Plus tard, d'autres développements (cfr. [POM, 80]) permettront d'enrichir considérablement ce modèle en permettant à une structure d'acteur de traiter plusieurs messages simultanément. De sorte que nous pouvons définir la **structure d'un acteur** comme :

a) L'ensemble des acteurs, appelé **accointances**, qu'il connaît et avec lesquels il communique.

b) Une **file d'attente des messages** issus des accointances implémentée selon un modèle FiFo.

c) Le **comportement** qu'il adoptera lors de la réception de messages des accointances. Ainsi, l'agent connaît une succession de comportements (un et un seul par message) qui viennent se greffer en remplacement des autres. C'est au sein même d'un comportement que sera défini le comportement de remplacement. Cependant le comportement "remplacé" continuera son exécution jusqu'à son terme. Nous sommes donc en présence d'une collection de comportements indépendants mais concurrents s'exécutant en parallèle.

4.4.3 Les O.S. distribués Objets

Appliquons le concept d'Objet aux systèmes d'exploitation. Pour mieux en cerner les mécanismes, nous nous proposons d'envisager un cas précis : le Chorus Object Oriented Layer (C.O.O.L.) développé en France. Il s'agit en réalité d'une extension qui, comme bien souvent, se greffe sur un système d'exploitation Unix (C++) dont la communication se base sur l'invocation d'objets.

Le système C.O.O.L. propose une nomenclature des types. Tout d'abord, le distinguo entre la couche Unix et l'extension s'applique aussi aux Objets, de sorte que nous sommes en présence d'Objets C++ et d'Objets C.O.O.L. Ensuite C.O.O.L. prévoit de définir un contexte à partir d'une collection d'Objets C.O.O.L. synchronisés par des sémaphores et des activités.

Un Objet C.O.O.L. connaît les attributs suivants : le serveur, le mode d'activité et le moniteur. Le serveur est un pointeur vers la mémoire de l'Objet associé à chaque instance du type Objet C.O.O.L. Le mode d'activité est dit actif lorsqu'une activité doit être créée à chaque instance du type Objet C.O.O.L. Le moniteur prévoit l'exécution en zone critique de toutes les opérations exportées par les Objets de ce type.

Un contexte, quant à lui, se voit réaliser par un Acteur (ensemble d'objets C.O.O.L.). Il s'agit en fait d'une mémoire accessible. La communication d'Objets liés à un même contexte se fait par une invocation réalisée par les mécanisme C++, tandis que, pour les Objets distants, l'invocation sera réalisée par des messages synchrones ou non. L'aspect qui nous semble le plus intéressant est d'associer à un message un Objet de sorte que la migration d'Objet corresponde en fait à un problème de plus haut niveau : la communication à distance.

4.4.4 Les systèmes de commandes Objets temps-réels

C'est dans le domaine de la robotique qu'apparurent les premiers systèmes de contrôle commande. Historiquement, les robots ont toujours été définis par leur appartenance à une classe de robots et constituées d'une série de composants : une plate-forme de mobilité, un bras de manipulation, une caméra pour la vision et la reconnaissance de l'environnement fermé et des autres objets [PELL, 90]. Ainsi, la robotique postule l'extistence d'objets coopérants selon un plan préetabli[1] afin d'atteindre un objectif commun.

Il apparaît que la modularité et la fléxibilité de la couche physique peut être transcrite au niveau du système de contrôle commande par l'adoption d'un modèle d'objets logiciels directement associé au modèle d'objets physiques. Parmi les

[1] Ce pourrait être par exemple un réseau de Pétri.

nombreuses approches présentent dans la littérature, il en est une dont la typologie des objets nous intéresse fortement et dont nous présenterons les grandes lignes.

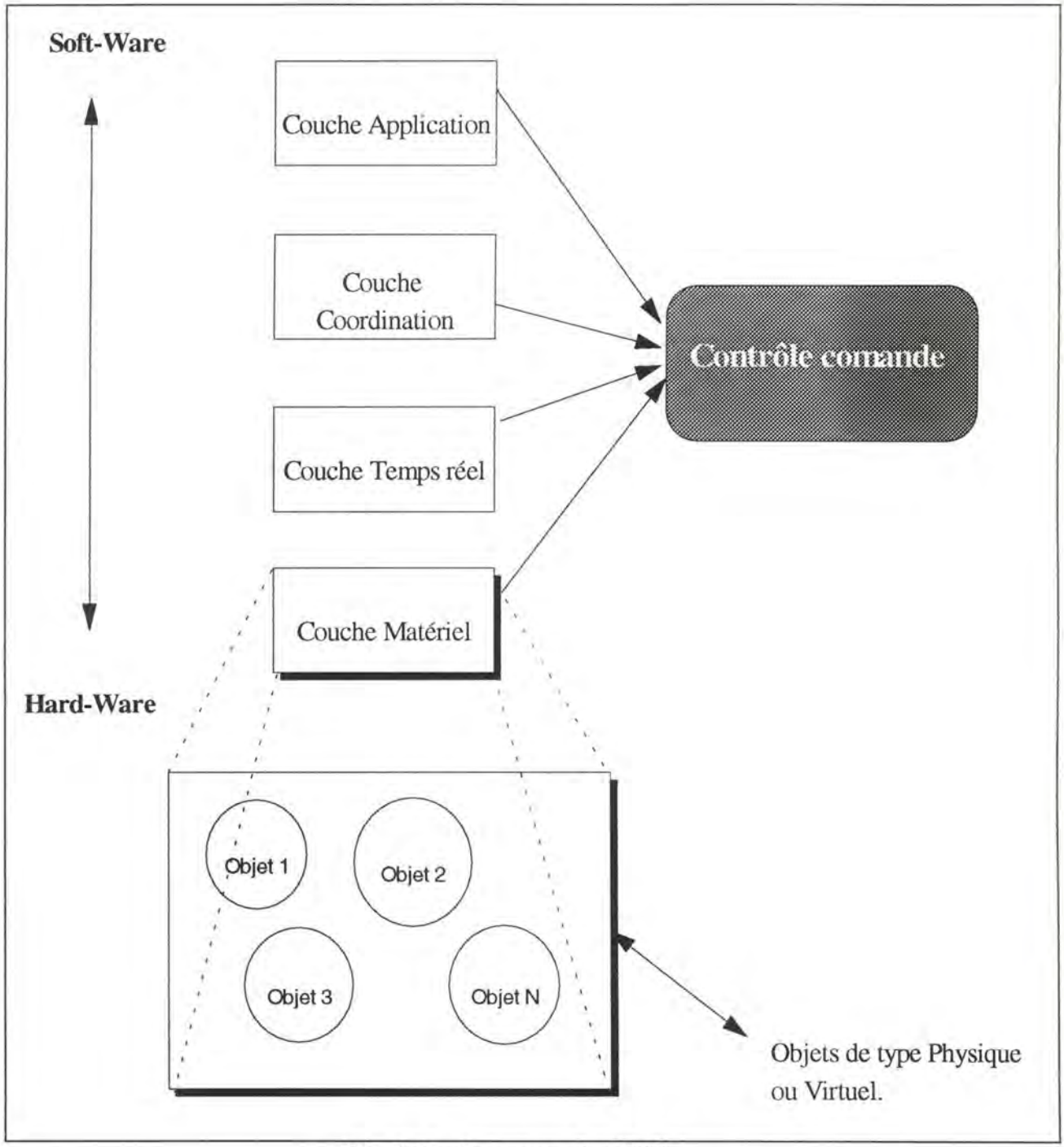


Fig. 4.8. : Structure de RIPE.

Il s'agit d'un environnement orienté objet pour la programmation indépendante de systèmes de contrôle/commande de robots (RIPE) développé sur le système

d'exploitation UNIX V. Son modèle de représentation et de structuration introduit **deux types d'objets** :

1. Les Objets Physiques, ils sont les représentants des entités physiques du robot.

2. Les Objets Virtuels qui ne sont pas des Objets Physiques, mais des entités dédiées à la réalisation de tâches telles que la génération des trajectoires, la gestion des erreurs ou encore le gestionnaire des communications.

Comme nous l'avons déjà ennoncé, la démarche des concepteurs aura été de structurer l'environnement RIPE comme un ensemble de classes logicielles et ce, en relation directe avec les objets physiques inhérents à une application robotique. Ces classes d'objets logicielles connaissent une interface de communication indépendante de l'application et constituée par le langage de programmation RIPL. Comme nous le montre la figure 4.8., l'environnement Ripe connaît **quatre couches** composées d'Objets de type Physique et/ou Virtuel :

1. Couche Application	Contient la définition de l'application, d'une tâche (par exemple, cela peut être la modélisation du monde).
2. Couche Coordination	Contient les programmes[1] de contrôle et de coordination des activités et des équipements.
3. Couche Temps-réel	Contient la commande temps-réel des équipements[2].
4. Couche Matériel	Contient les différents drivers des équipements.

Pour plus de détails et en particulier pour connaître les domaines d'applications de RIPE nous invitons le lecteur à se référer à [Mill, 91].

[1]La programmation se faisant sous UNIX V, le langage utilisé est le C++.
[2] Ici encore l'implémentation se fait en C++, cette fois sur base d'un système d'exploitaion temps-réel de type VxWorks.

4.4.5 Limites

Comme nous l'avons vu précédemment, la méthodologie de conception d'architecture de logiciels issus du concept d'Objet (O.O.A.) offre des avancées précieuses en terme de structuration (langage de classes), d'activités parallèles et de communication asynchrone (langage d'acteurs). Cependant, le modèle Objet ne propose malheureusement pas d'implémenter de manière efficiente la communication par messages connaissant une réponse et une certaine priorité. C'est pourquoi, des extensions telles que ABCL/TIT [YONR,86] ont vu le jour et développent l'idée d'Objets concurrents, encore appelé Objets Actifs.

Un Objet Actif est un objet connaissant une activité qui se déroule séquentiellement. Cette activité est connue au sein du système d'exploitation comme une tâche ou encore un processus.

Dans le cas ABCL/TIT , il s'agit en fait d'Objets actifs communiquant par voie de messages asynchrones. Chaque Objet, que l'on appelle aussi Acteur possède sa propre mémoire sur site abritant son état et son scénario comportemental. La communication, qui, selon les besoins asynchrones, peut être indifféremment synchrone ou anticipée, se fait via une boîte aux lettres attachée à l'agent. Celle-ci est composée de deux files de messages gérées selon un modèle Fifo. Il y a respectivement une file de messages normaux (Commun) et une file de messages prioritaires (Express). Ces derniers ont le pouvoir de courcircuiter le traitement de tout autre message de type ordinaire. Lorsque le dernier message a été traité, l'agent s'endort.

On le voit, il est possible de raffiner cette extension en multipliant les files de messages et donc les niveaux de priorité.

4.5 Synthèse

Dans ce chapitre, nous avons donné un aperçu d'un système distribué et exposé les mécanismes classiques de communication. Il est rapidement apparu que ces mécanismes présentaient des limites par rapports à nos exigences :

1. faiblesses au niveau de la modélisation ;

2. insuffisance des mécanismes de communication classiques.

L'approche objet nous a permis de nous doter d'un modèle de représentation et de structuration des informations manipulées, assurant notamment l'extensibilité et la portabilité du système. Il demeure cependant des lacunes au niveau de la communication.

5. COMMUNICATION PAR OBJETS PARTAGES

5.1 Introduction

Nous avons vu qu'il existe deux techniques de communication entre processus : la mémoire partagée et l'échange de messages. Chacune de ces techniques a ses limites et ses avantages. De manière générale, le choix d'une technique plutôt qu'une autre sera dicté par l'architecture matérielle et les supports de communication existants (tailles des mémoires, types de réseaux, environnement distribué ou non ...).

Ayant également abordé les concepts objets et leurs avantages, il nous paraît dès lors intéressant de réunir les techniques de communications classiques avec les concepts objets afin de mettre sur pied un système de communication par objets partagés qui nous permettra de pallier aux inconvénients rencontrés lors de la reconfiguration d'une application. Il s'avère en effet que, lorsque l'on modifie la répartition des processus ou lorsque l'on adjoint de nouveaux processus à une application, de profondes modifications doivent être opérées, rendant ainsi la reconfiguration extrèmement difficile (l'extensibilité).

Pour réaliser un tel système de communication, il nous paraît opportun de définir les deux types d'objets suivants : les **agents** et les **objets partagés**. « Les agents sont des objets relationnels pourvus d'intentions dans le sens où ils ont des buts, des besoins et des préférences. Un agent peut exécuter un processus dont les effets peuvent changer l'état du système ».[MYL ,95] Les agents représenteront donc l'unité de parallélisme de notre modèle de communication.. Les objets partagés, quant à eux, seront les données manipulées (et/ou échangées) par les agents. La figure 5.1. permet de visuliser les rôles de ces deux objets.

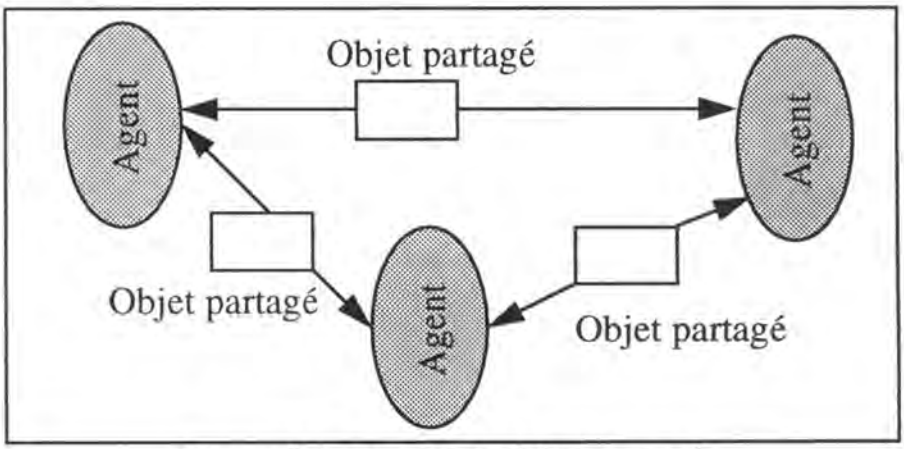


Figure 5.1. : agents et objets partagés

5.2 Modèle d'un agent

Afin de considérer l'agent comme une extension d'un objet dans un environnement parallèle distribué, il nous faut affiner et restreindre légèrement la définition donnée par John Mylopoulos[1]. Nous définirons dès lors un agent comme une entité indépendante caractérisée par les trois composants ci-dessous (fig. 5.2.) :

- le(s) **activité(s)**, représentée(s) par le déroulement du ou des processus déclenché(s) par l'agent ;
- des **données (publiques ou privées)**, ou objets créés pendant le déroulement d'une activité d'un agent, seules les données publiques peuvent être accessibles par d'autres agents ;
- des **moyens de communication**, puisque l'agent est une entité relationnelle, il convient qu'il puisse échanger des informations avec d'autres agents.

Les données privées et publiques représentent l'état de l'agent, elles peuvent être constituées de données locales mais aussi d'autres agents.

Nous retenons la portée de la définition précédente quant aux conséquences d'une activité sur l'environnement. Il convient cependant d'apporter une distinction entre des agents au sein desquels plusieurs activités concurrentes peuvent se dérouler en parallèle et des agents ne comprenant qu'une seule activité. Des problèmes d'intégrité de données peuvent en effet résulter d'un partage de l'état de l'agent par ses diverses activités. Ces problèmes doivent être résolus manuellement par le programmeur et nous n'en tiendrons pas compte dans notre modèle.

L'accès aux données d'un agent n'est possible que via les mécanismes de communications fournis par cet agent.

[1] La définition proposée dans [MYL ,95] est en effet trop générale et le domaine d'application des agents se cantonne trop souvent aux S.I. utilisés dans le cadre d'application pour le business. Les systèmes robotisés connaissent généralement d'autres contraintes.

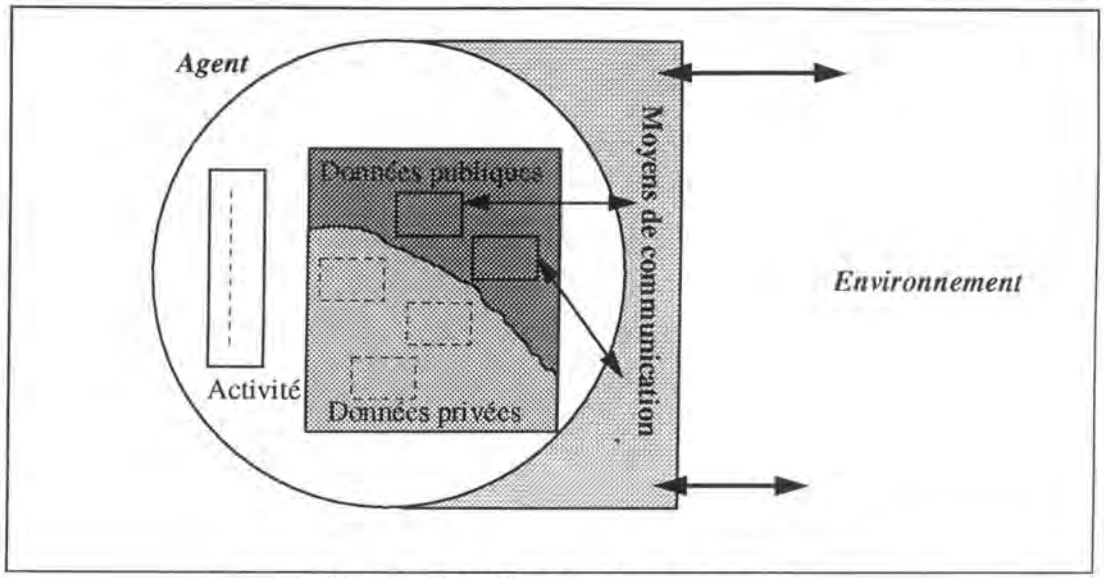


Figure 5.2. Modèle d'un agent

5.3 L'objet partagé

Quel que soit le schéma de communication choisi pour échanger de l'information entre différents processus (Cfr. Chapitre 3), il existe une constante indépendante du mécanisme de communication : **le cycle de vie de l'information**. Toute information échangée possède en effet un cycle de vie qui comporte une phase de génération de l'information par un agent, une phase d'échange ou de transmission de l'information (de l'agent générateur vers un ou plusieurs autres agents), une phase d'exploitation pendant laquelle chacun des agents destinataires utilise l'information pendant une certaine durée qui leur est propre et finalement une phase de destruction de l'information lorsque le dernier agent à l'utiliser décide qu'elle est dorénavant inutile.

Afin d'être transmise, l'information est généralement inscrite sur un support mémoire. Si le mécanisme de communication utilisé est la mémoire partagée, la transmission de l'information est implicite. Par contre, lorsque l'on utilise l'échange de messages, la transmission de l'information est forcément explicite. Dans la méthodologie objet, l'information à échanger est encapsulée par un objet et deviendra accessible par certaines procédures qui constituent l'interface de l'objet en question.

Le principe de la communication par objet partagé consiste à associer un **support mémoire** et un **état de validité** rendant compte de l'état d'avancement de l'information dans son cycle de vie. Dans sa phase de génération, une information sera qualifiée de non valide. Ce n'est qu'au moment de la transmission que son producteur la validera afin de la rendre exploitable par les autres agents consommateurs.

5.4 Protocoles logiques d'échange d'informations par objets partagés

Nous faisons l'hypothèse que nous disposons de plusieurs processeurs sur lesquels s'exécutent de manière concurrente les agents communiquant entre eux. En fait, il y aura autant de processeur qu'il y a d'entités matérielles dans le véhicule (injection, A.B.S., centrale de navigation, centrale de détection etc...). Comme nous sommes dans un environnement distribué, le parallélisme de ces agents est réel. Il peut cependant exister des parallélismes simulés dans le cas d'un agent ayant plusieurs activités concurrentes tournant sur le même site ou tout simplement lorsque plusieurs agents exécutent leur activité dans un environnement mono-processeur. Dans ces deux derniers cas, un scheduler sera nécessaire afin de simuler ce parallélisme.

La communication par objets partagés se déroule en cinq étapes :

1. Création de l'objet partagé qui n'est autre que le support du transfert de l'information.
2. Ecriture et validation de l'information
3. Transmission de l'information.
4. Lecture de l'information.
5. Destruction de l'objet partagé.

Comme l'information est échangée entre un émetteur et un ou plusieurs destinataires, deux cas de figure peuvent se présenter : l'émetteur est producteur de l'information ou un des destinataires est producteur de l'information. Un protocole spécifique correspond à chacune de ces situations (Figures 5.3. et 5.4.). Il est important de bien différencier l'émetteur du producteur et le récepteur du consommateur. De même, nous discernons l'objet partagé - qui n'est autre qu'un support mémoire - de l'information qu'il contient.

Il convient également de noter une différence fondamentale entre le modèle de communication par objets partagés et celui des lecteurs/écrivains. L'objet initialisé n'est jamais modifié alors que, lors de l'utilisation de la mémoire partagée, les processus communicants effectuent souvent des modifications du contenu de la zone mémoire commune. L'objet initialisé n'est jamais modifié pour garantir l'intégrité de l'information durant toute la durée de l'échange. Il va sans dire que cette contrainte réduit les performances temporelles du modèle puisque chaque objet créé est quasi

systématiquement détruit juste après sa création. Nous verrons par la suite comment résoudre ce problème crucial pour les applications temps-réel.

Les mécanismes de communication par objets partagés peuvent être considérés comme des abstractions des mécanismes classiques de communication. Ainsi, les techniques de mémoires partagées ou d'échange de messages peuvent être utilisées sans aucune incidence sur la programmation. L'utilisation de la mémoire partagée aura pour effet de transmettre l'adresse d'un objet partagé alloué en mémoire globale alors que dans le cas d'un échange de messages, l'objet alloué en mémoire locale est encapsulé dans un message pour être transmis par copie. Les objets partagés constituent alors une interface homogène et indépendante de l'implémentation permettant de synchroniser un échange de données entre différents agents.

5.4.1 L'émetteur est producteur de l'information

L'agent émetteur crée un objet partagé, dont le rôle est d'encapsuler l'information. Il valide cette information puis transmet l'objet à un ou plusieurs destinataires. Après l'opération de transmission, l'objet créé n'a plus d'utilité et doit donc être détruit.

Les agents destinataires attendent l'objet partagé. Lorsqu'ils le reçoivent, ils lisent l'information qu'il contient. Une fois la lecture effectuée, l'objet n'a plus aucune utilité pour la communication, il est donc détruit.

Nous avons choisi le formalisme des réseaux de Pétri pour représenter les différentes étapes du protocole (Figure 5.3).

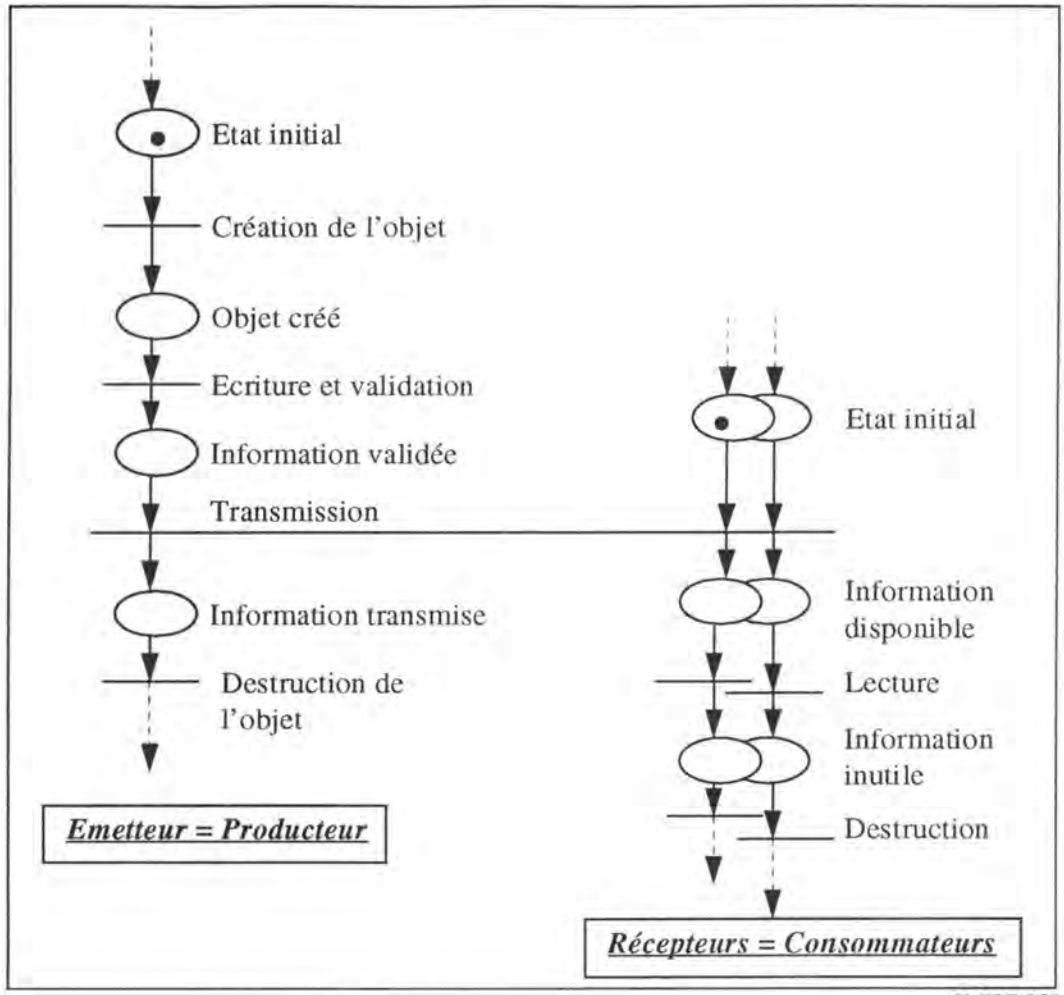


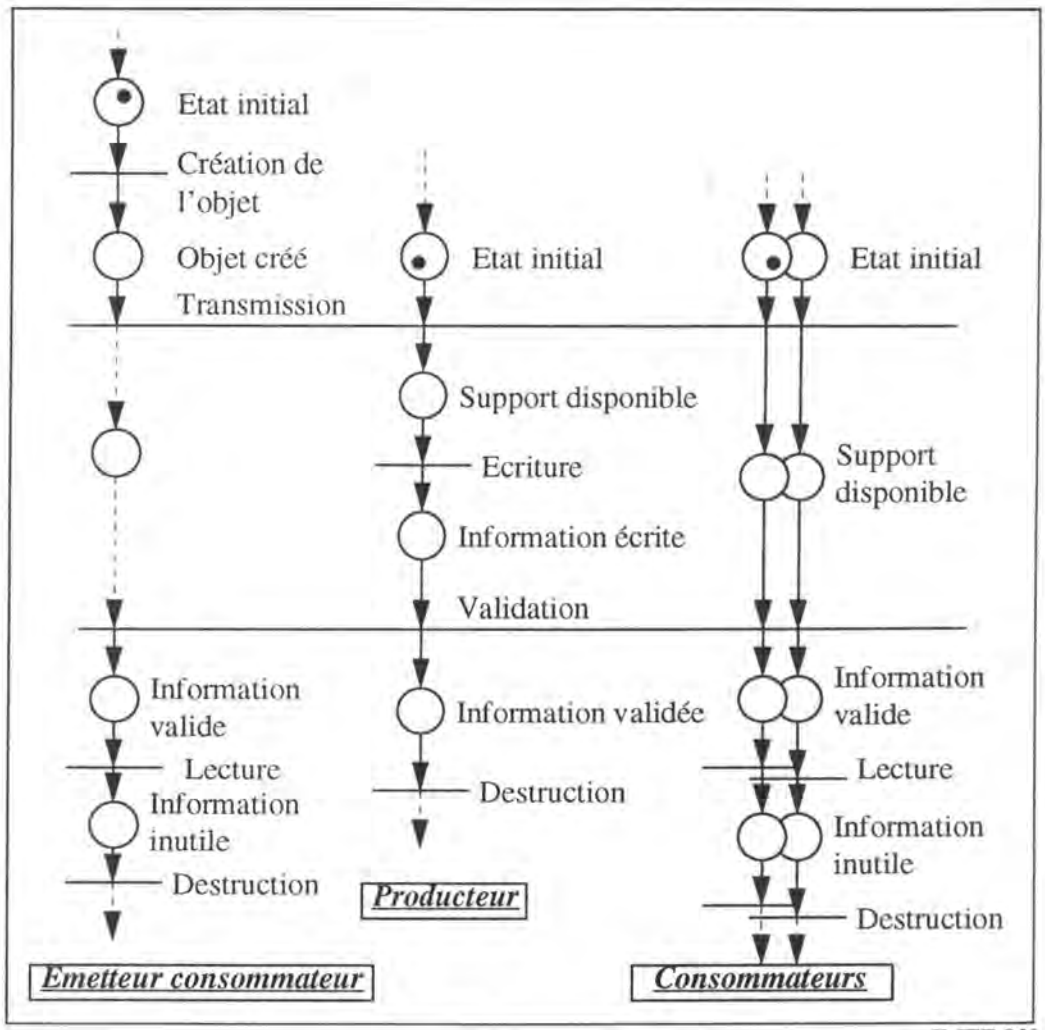
Fig. 5.3. L'émetteur est producteur

5.4.2 Un des destinataires est producteur d'informations

Dans ce cas, l'agent émetteur est demandeur d'informations. Pour ce faire, il crée un objet partagé et le transmet au producteur qui va encapsuler l'information dans l'objet reçu.

Après avoir validé l'information demandée par l'émetteur, l'agent producteur la lui retourne. Notons que l'émetteur peut, sans que le producteur en ait connaissance, transmettre l'objet partagé à d'autres agents consommateurs.

La figure 5.4. représente le déroulement de ce second protocole.



[MEF,93]

Fig. 5.4. Un destinataire est producteur

5.5 La synchronisation des agents

Comme le mécanisme de communication par objets partagés permet entre autres la synchronisation des agents qui échangent des données, il convient de s'interroger sur le comportement de ces agents avant et après la transmission. Or, c'est le protocole utilisé qui conditionnera le comportement des agents lorsqu'ils communiquent entre eux. Il existe de nombreuses variantes de protocoles dont les plus extrêmes sont représentées ci-dessous : figure 5.5.

Dans le protocole synchrone, l'émetteur, avant de transmettre, attend que le récepteur soit prêt. Le récepteur subit la même attente pour être au rendez-vous lorsque la transmission s'effectuera. Une fois l'information transmise, l'émetteur attend de nouveau pour recevoir le compte rendu. Lorsqu'il l'a reçu, il peut poursuivre son activité. Par contre, dans les protocoles complètement asynchrones, l'émetteur

transmet l'information sans se préoccuper de l'état du récepteur, continuant ainsi son activité sans attendre le compte rendu éventuel qui lui serait adressé par le récepteur.

Il en résulte qu'un agent peut avoir un comportement synchrone ou asynchrone tant en émission qu'en réception.

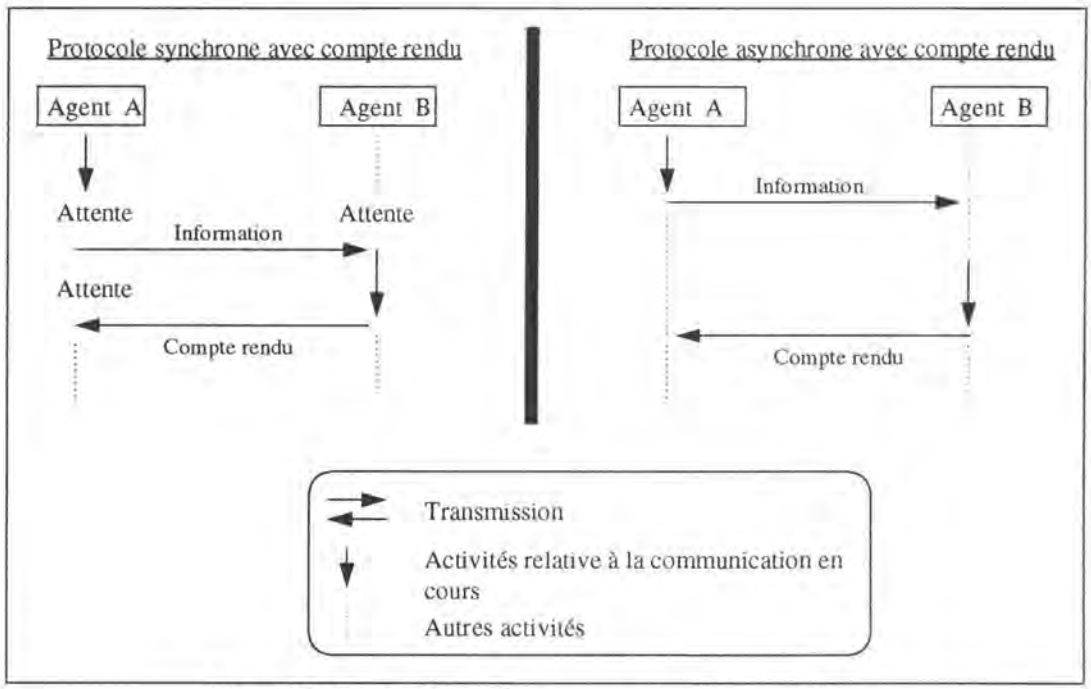


Fig. 5.5. Protocoles de communication extrêmes

Comme nos agents doivent être utilisés pour un système de contrôle commande temps-réel, ils doivent d'une part réagir rapidement et d'autre part être indépendants et autonomes. De plus, comme l'agent devra faire face à différentes situations telles que : identification d'un obstacle sur la trajectoire du véhicule, modification légale de la vitesse, etc.., il est nécessaire qu'il puisse définir son comportement de manière dynamique.

Afin d'assurer le respect de ces trois contraintes, il faut permettre à nos agents de communiquer suivant le protocole le moins contraignant possible, c'est-à-dire le protocole **asynchrone sans compte-rendu**. Il est néanmoins intéressant de permettre aussi à nos agents d'anticiper certaines synchronisations (cfr. caractère temps-réel).

Il nous faut donc définir un mode de communication asynchrone pour chacun des protocoles de communication par objets partagés que nous avons décrits dans la section précédente et déceler les points de synchronisations qui permettront de coordonner les activités des agents (lorsque l'émetteur est producteur et lorsque l'émetteur est consommateur).

Nous dérivons donc directement des deux protocoles deux modes de communication asynchrone sans obligation de réponse (compte rendu). Ces modes de communication par objets partagés sont illustrés ci-dessous (figure 5.6.).

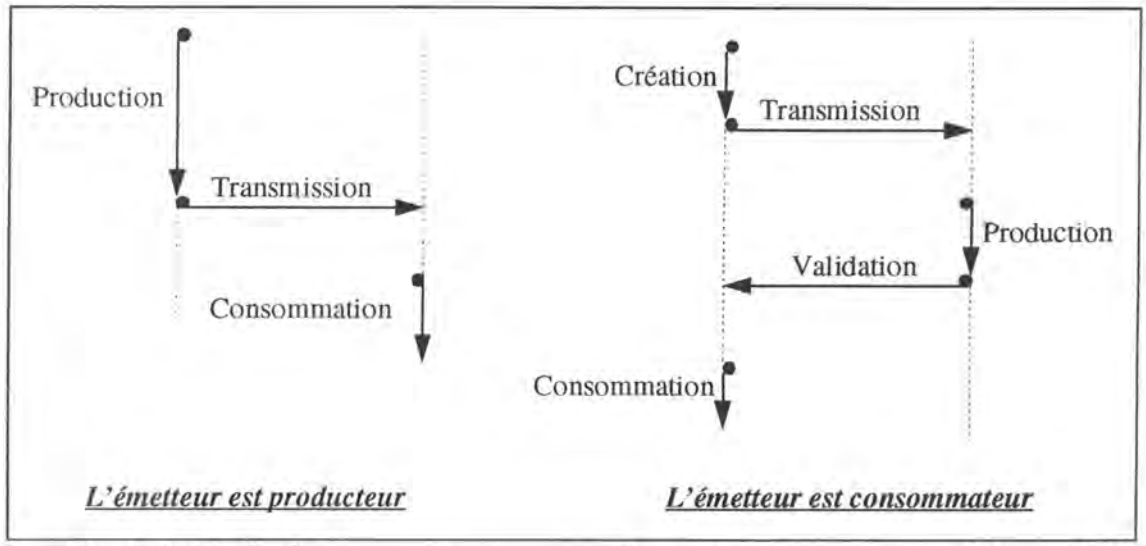


Fig. 5.6. Mode de communication asynchrone (sans compte rendu) par objet partagé

Il existe deux transitions - dans les réseaux de Pétri utilisés pour schématiser les protocoles des communications par objets partagés - qui peuvent être utilisées comme points de synchronisation possibles entre deux agents communiquant par objet partagé. Il s'agit de la **transition de transmission** et de la **transition de validation**. La transition de transmission indique une requête d'un agent émetteur s'adressant à un agent récepteur alors que la transition de validation permet à un agent producteur d'avertir les agents consommateurs que l'information est disponible. Il est important de noter que la sensibilité des agents est un élément important de la communication par objets partagés. En effet, à la transmission d'un objet partagé ou à la validation de l'information qu'il contient, les agents sensibles sont avertis de la disponiblilité de l'information qui les intéresse (respectivement l'objet partagé pour le récepteur et l'information contenue dans l'objet pour le consommateur).

Il nous faut donc réaliser deux synchronisations : la première entre un agent émetteur et un agent récepteur (transition de transmission) et la seconde entre un agent producteur et un agent consommateur (transition de validation). Par contre, si nous considérons l'objet partagé (donc le support mémoire) comme une information à part entière, produite par un émetteur et consommée par un récepteur, nous pouvons n'envisager qu'un seul mécanisme générique basé sur la validation de l'information par son producteur. On peut donc définir complètement le mécanisme de synchronisation par deux primitives complémentaires : une primitive de validation de l'information et une primitive d'attente de la validation d'une information.

Nous proposons la syntaxe en pseudo code suivante comme forme de ces primitives :

Attente(objet_partage1,objet_partage2,objet_partage3, ...);

Validation(objet_partage1,objet_partage2,objet_partage3, ...);

{ Qui retournent la position de l'argument validé. Ces formes ont l 'avantage de réaliser de manière simple et en une seule instruction, une synchronisation multiple avec en plus une notion de priorité attachée à la place occupée par l'objet partagé dans la liste des arguments. De plus, cette syntaxe n'étant pas conforme à l'approche objet, elle n'oblige pas le programmeur à adjoindre les méthodes validation et attente dans la définition de son objet.}

Nous décrirons plus en détail les mécanismes de synchronisation dans le chapitre suivant (6.5 Quid de la synchronisation ?).

5.6 Problèmes liés à la manipulation des objets partagés et leur solution : la manette

Si le mécanisme de communication par objets partagés est une abstraction des mécanismes classiques de communication, il n'en demeure pas moins qu'il faut utiliser l'un de ces derniers pour réaliser la transmission de l'objet partagé. Nous avons vu qu'il existe deux principes : la mémoire partagée et l'échange de messages. Lors d'un échange de messages, si les agents communicants ne disposent pas d'une mémoire commune, l'objet partagé est recopié dans le contexte des agents destinataires. Par contre, si les agents s'exécutent sur un même processeur ou sur plusieurs processeurs disposant d'une mémoire commune, il est beaucoup plus intéressant d'utiliser le mécanisme de mémoire partagée. L'objet est alors alloué en mémoire globale et est transmis par référence. Il est en effet beaucoup plus efficace de transférer une référence plutôt que de copier l'objet tout entier.

Ainsi, dans une communication par objets partagés, le programmeur manipulera les objets par l'intermédiaire de références tant au niveau de l'émetteur qu'au niveau du destinataire. C'est le système d'exploitation qui se chargera d'encapsuler les objets

dans des messages et d'opérer les transmissions explicites entre sites distincts de façon à ce que toutes ces tâches soient transparentes au programmeur.

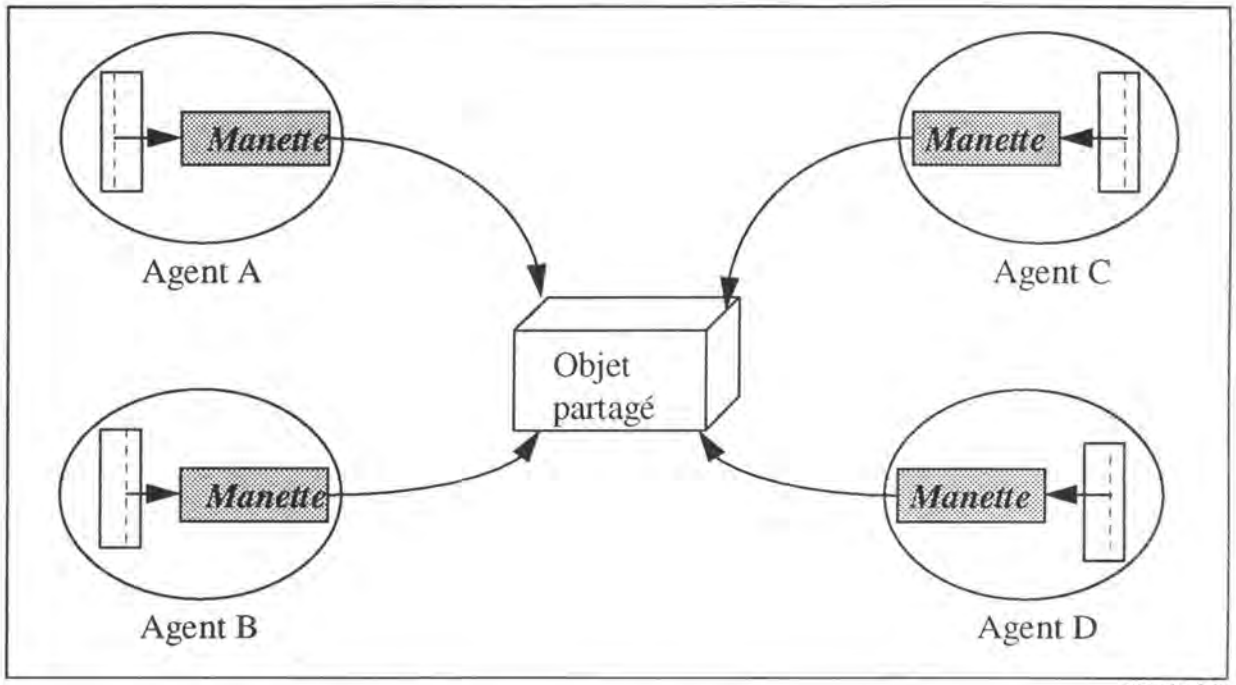
L'utilisation des références pose cependant des problèmes de gestion de la mémoire lorsque, dans une communication par objets partagés, les agents créent et détruisent presque simultanément les objets qu'ils se partagent. Un objet ne peut être détruit que lorsque l'information qu'il encapsule n'est plus d'actualité, c'est-à-dire lorsque le dernier consommateur a pu lire l'information qui lui a été transmise.

Or, dans notre modèle, les agents évoluent en parallèle et sont indépendants les uns des autres. Comment donc identifier à priori le dernier utilisateur d'une information ? La solution consiste à obliger chaque agent à demander la destruction de l'objet lorsqu'il ne s'en sert plus et à opérer la véritable destruction lorsque le dernier utilisateur fait appel au destructeur. Cette solution[1] peut être réalisée efficacement grâce à des objets de références : **les manettes**.

Une manette est une référence instanciée sous la forme d'un objet local dans le contexte d'un agent. Cette référence représente un lien sur un objet partagé alloué en mémoire globale (cfr. Figure 5.7.). Les manettes peuvent donc être vues comme des sémaphores de niveau supérieur permettant de réaliser une gestion saine de la mémoire dynamique. A chaque objet partagé est associé un compteur de références (initialisé à 1 lors de la création de l'objet, incrémenté de 1 lorsqu'une référence est dupliquée et décrémenté de 1 lorsqu'une référence est détruite). L'objet ne sera effectivement détruit et la mémoire libérée que si le compteur de référence est à 1. L'utilisation d'une manette ne se limite pas au seul contrôle de la destruction d'un objet, elle peut être utilisée pour vérifier le respect de certaines conditions, notamment pour la duplication d'une référence. Par exemple, une copie ne sera réalisée que si la variable destinée à recueillir la référence à copier est NUL ...

Il faut toutefois insister sur le fait que les opérations de création, de copie et de destruction doivent être considérées comme des sections critiques devant être réalisées en exclusion mutuelle (Cfr. Utilisation des sémaphores au chapitre précédent).

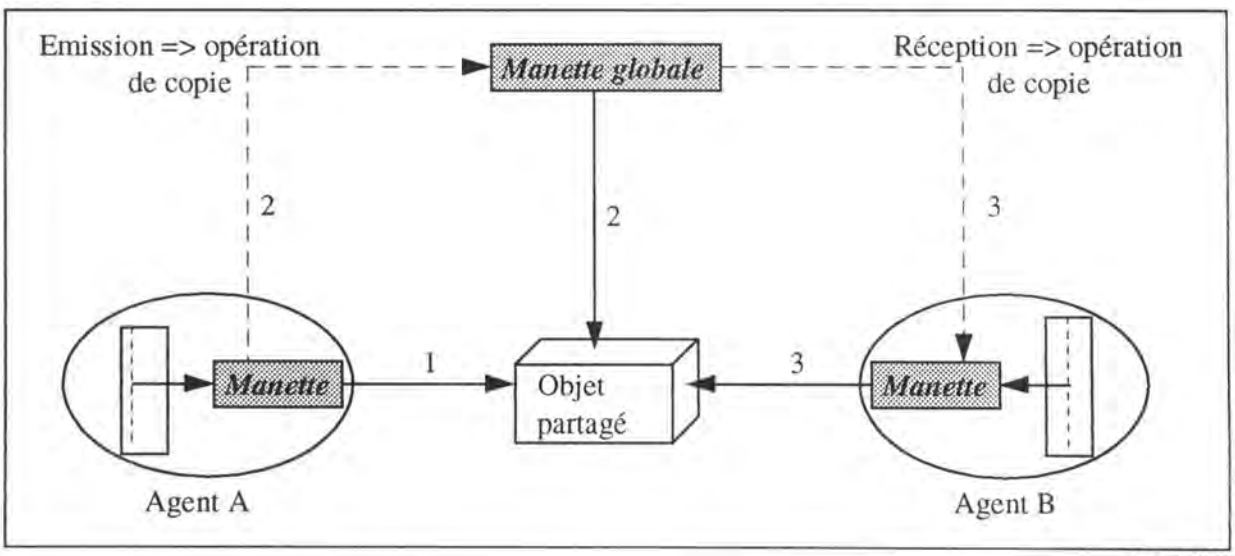
[1] Dans un environnement distribué constitué de plusieurs processeurs ne disposant pas d'une mémoire commune, une référence à une zone mémoire pour un processeur n'a aucune signification pour un autre processeur. Dans ce cas, seul le mécanisme d'échange de message, et donc de copie complète de l'objet partagé dans l'environnement local du destinataire, pourra être utilisé. L'accès aux objets partagés par référence est donc impossible et l'utilisation de manettes s'avère inutile.



[MEF,93]

Fig. 5.7. L'utilisation des manettes pour la manipulation des objets partagés

Nous pouvons utiliser les manettes pour créer un premier niveau d'abstraction pour notre objet de communication. Il nous suffit de créer une manette globale qui n'est autre qu'un objet de référence alloué dans une mémoire commune aux différents agents désirant communiquer (figure 5.8.). Les agents viendront alors lire et écrire leurs références à l'aide d'une opération de copie de la manette[1]. Rappelons que cet objet de communication nécessite un mécanisme de mémoire partagée, il n'est donc pas applicable dans un environnement distribué.



[MEF,93]

Fig. 5.8. communication par manette globale

[1] l'opération de copie doit être atomique.

La communication relative à une manette globale se déroule en trois étapes. (Les numéros de ces étapes ont été reportés sur la figure 5.9.) La première étape (1) a pour effet de créer un objet partagé ce qui se résume à : (a) allouer une zone mémoire globale destinée à accueillir l'information, (b) instancier une manette locale (celle de l'agent A) contenant une référence sur la zone allouée en mémoire, (c) initialiser le compteur de référence de la manette à 1. La seconde étape (2) consite à émettre l'objet partagé. Comme nous l'avons dit précédemment, l'émission est en fait une copie de la référence locale (manette de l'agent A) dans la manette globale. Ce dernier pointe dorénavant sur l'objet partagé créé par l'agent A. Comme il y a eu copie, le compteur de référence est incrémenté d'une unité, il est donc égale à deux. La dernière étape (3) consiste en la réception de l'objet partagé envoyé par l'émetteur. C'est également une copie de référence, mais cette fois, de la référence globale (manette globale) dans une référence locale (manette de l'agent récepteur B). Suite à cette nouvelle copie, le compteur de référence est de nouveau incrémenté et le nombre de références passe ainsi à trois.

Comme l'objet de communication est une référence, les primitives d'attente et de validation permettant de synchroniser les agents sont toujours d'actualité et peuvent donc être utilisées pour synchroniser l'émetteur et le récepteur. Il suffit en effet de considérer la référence globale comme une information écrite par l'émetteur. Après une écriture, cette information (référence globale) est validée, ce qui a pour effet d'activer les agents qui l'attendaient.

Lorsque l'information n'est plus utilisée par un agent, celui-ci peut détruire l'objet partagé. Il ne fera en fait qu'une destruction de sa référence locale. Ce qui n'a comme effet que de détruire le lien qui associe sa manette à la manette globale. Cette destruction n'aura donc aucune incidence sur les autres liens entretenus par les agents utilisant le même objet partagé.

5.7 Un objet de communication pour un environnement distribué : le canal

Un canal représente une **abstraction d'une voie de communication** admettant une **capacité de stockage** pour mémoriser des objets partagés selon le mode **FIFO**. Un canal doit toujours être **attaché** à l'agent **destinataire**.

Il convient d'éclaircir quelque peu cette définition succinte du canal tant au niveau de la politique de stockage qu'au point d'attache obligatoire de notre objet de communication.

Il est évident qu'un canal peut être uni ou bidirectionnel. S'il est unidirectionnel, il peut, en toute généralité, être attaché tant à l'émetteur qu'au destinataire. S'il est bidirectionnel, il sera bien sûr attaché aux deux agents. Un lien est toujours obligatoire puisqu'il permet au système de communication de prendre en charge de manière automatique la synchronisation des agents par rapport aux données échangées. Or, dans les protocoles de communication par objets partagés que nous avons exposés, le destinataire doit être averti quand l'objet partagé a été envoyé par l'émetteur. Il en résulte qu'un canal de communication sera attaché à l'agent destinaire.

Le canal permet donc à l'agent auquel il est attaché de recevoir des informations alors qu'il n'est pas à l'écoute. Un agent peut bien sûr posséder plusieurs canaux ayant des priorités différentes. Notons dès à présent que les priorités sont associées aux canaux et non aux objets partagés. Ces priorités sont spécifiées par l'agent destinataire et ce durant l'appel de la primitive d'attente. Ces priorités sont donc dynamiques et, par conséquent, c'est l 'acteur destinataire qui est le seul juge de l'importance à accorder aux informations qu'il reçoit. Puisque, tout objet partagé arrivant sur le même canal possède une priorité identique à celle accordée à ce canal, les objets partagés pourraient être mémorisés selon le mode FIFO.

Pour des raisons d'efficacité, il nous paraît judicieux de limiter le modèle de communication par canal à un système point à point. Il apparaît en effet que bon nombre d'interactions entre agents dans les systèmes opératoires (et temps-réel en particulier) sont de type point à point. Ensuite, la mise en oeuvre de mécanismes de diffusion est complexe et de fiabilité discutable, surtout dans des environnements distribués.

Dans notre modèle, un agent peut donc créer plusieurs canaux de priorité différente (figure 5.9.).

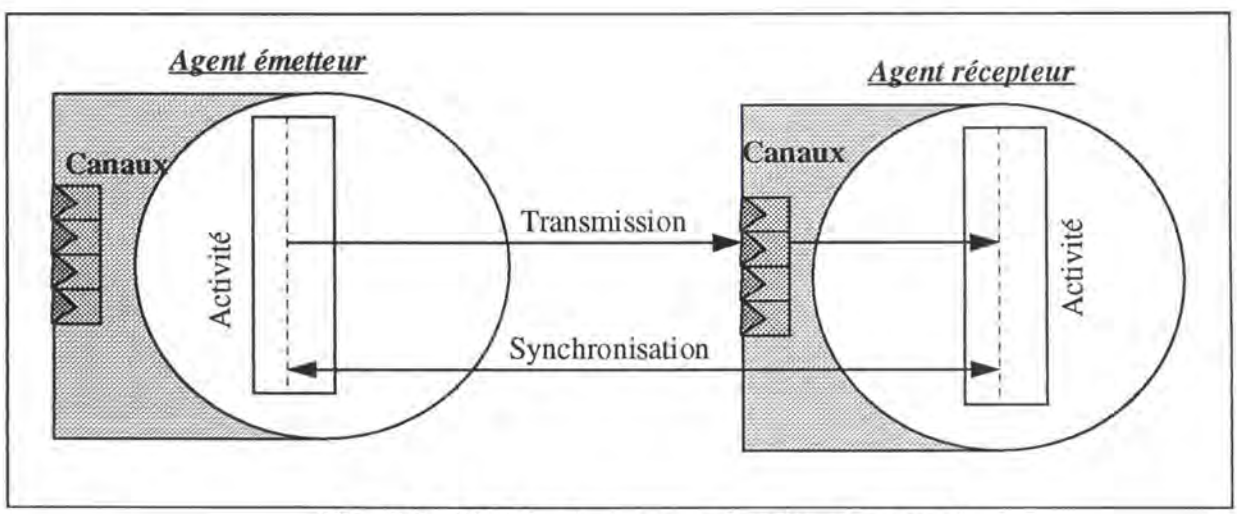


Fig. 5.9. Modèle de communication par canal

Contrairement au modèle de communication par référence globale, la communication par canal comprend trois phases : (1) l'ouverture du lien, (2) une phase de communication, (3) la fermeture du lien.

L'objet canal est créé par l'agent récepteur[1] et doit être considéré comme un objet conventionnel pouvant donc être manipulé par référence. Il nous faut alors disposer des données suivantes afin de publier le canal nouvellement créé : la référence de l'objet partagé de type canal et un identificateur représentant le nom de la référence.

Lorsqu'une demande d'ouverture de lien est faite par l'agent émetteur, celui-ci fait appel aux services du catalogue auquel il transmet le chemin d'accès au canal. Le catalogue lui transmet en retour la référence du canal sur lequel les objets partagés doivent être envoyés. La communication peut alors avoir lieu.

A la fin de la communication, il faut fermer le lien. Cette opération consiste à simplement détruire la référence communiquée par le catalogue à l'émetteur.

Il est possible d'implémenter les canaux suivant différents principes. L'objet partagé peut être alloué en mémoire locale et transmis par copie ou alloué en mémoire globale et transmis par référence. Le canal peut aussi rendre l'utilisation des messages (obligatoire si les deux agents communicants sont distants dans l'espace) totalement transparente au programmeur. Le canal est donc un mécanisme d'abstraction encapsulant différentes techniques internes de mise en oeuvre, tels la mémoire partagée ou l'échange de messages, ces derniers pouvant être utilisés indifféremment pour réaliser l'échange des objets partagés. Ainsi, selon la dispersion des agents, on optera pour un mécanisme de mémoire partagé (le plus rapide) ou, si c'est vraiment obligatoire, pour l'échange de messages.

5.8 Synthèse

Sur base des mécanismes de communication classique et de l'approche objet, nous avons dressé l'ébauche d'un protocole de communication par objets partagés qui garantit des échanges asynchrones entre les agents communicants. En outre, leurs activités sont synchronisées par des primitives de validation et d'attente.

L'utilisation des objets manettes pour manipuler les objets partagés nous a permis de rendre transparent au programmeur la gestion des problèmes classiques de manipulation des références et de réaliser une gestion saine de la mémoire dynamique.

[1] Afin de faire savoir aux autres acteurs intéressés qu'un canal vient d'être créé, l'agent récepteur doit faire appel aux services d'un acteur système, « le catalogue », auquel il transmet les données permettant d'identifier le canal créé. Pour le moment, considérons qu'un catalogue gère une arborescence constituée de noms de canaux auxquels est rattaché le nom de la machine sur laquelle s'exécute l'agent qui a créé le canal. Nous détaillerons plus avant cet acteur système dans le chapitre suivant (7.5.3 L'objet Catalogue).

Nous avons défini deux objets de commmunication : la manette globale (permettant à plusieurs agents résidant sur le même processeur d'échanger des données) et le canal (qui permet une communication entre agents quel que soit leur lieu d'exécution).

Il nous reste cependant à offrir une solution efficace aux problèmes de distribution et d'hétérogénéité, nos modèles se situant en effet à un niveau d'abstraction équivalent à l'échange explicite de messages, ce qui nous éloigne du modèle de base de la communication objet qui se fait par invocation.

6. ARCHITECTURE DE L'OIGNON A OBJETS DE COMMUNICATIONS

6.1 Introduction

Sur base des principes précédemment énoncés, nous allons, dans les points qui suivent, proposer une architecture de communication que nous nommerons : « Oignon à objets de communication ». Dans un premier temps nous donnerons les idées qui ont présidé à son élaboration (6.2); ensuite, nous détaillerons quelque peu cet oignon (6.3 & 6.4); enfin, il sera question des problèmes soulevés par le besoin de synchronisation (6.5).

6.2 Idée

Afin de proposer un design d'architecture de notre système de communication qui présente un haut degré d'extensibilité et de portabilité, nous allons tirer parti de la métaphore de l'oignon et des concepts de modélisation orientés objet définis au point 4.4 Approche Objet.

6.2.1 La métaphore de l'oignon

L'extensibilité, garante d'un système ouvert et évolutif, est le fruit de la découpe hiérarchique de l'oignon. En effet, le noyau central est rendu extensible par la possibilité d'adjoindre un ensemble de couches supplémentaires qui connaissent des niveaux d'abstraction différents. Ainsi, une couche de niveau N sera définie de manière plus précise par la couche de niveau N-1 et définira l'interface avec le niveau N+1. Et c'est justement ces interfaces successives qui permettront de garantir la portabilité de cette architecture : il ne sera en effet possible de réduire les lourdes procédures d'interfaçage des matériels -souvent hétérogènes- de la couche basique de niveau zéro que grâce à elles. Cette couche, qui en réalité n'est autre que le système d'exploitation, doit être capable de supporter l'ordonnancement des tâches et la gestion dynamique de la mémoire, des interruptions, des événements, etc. Notre système de communication ne commence donc véritablement qu'à la couche de niveau 1.

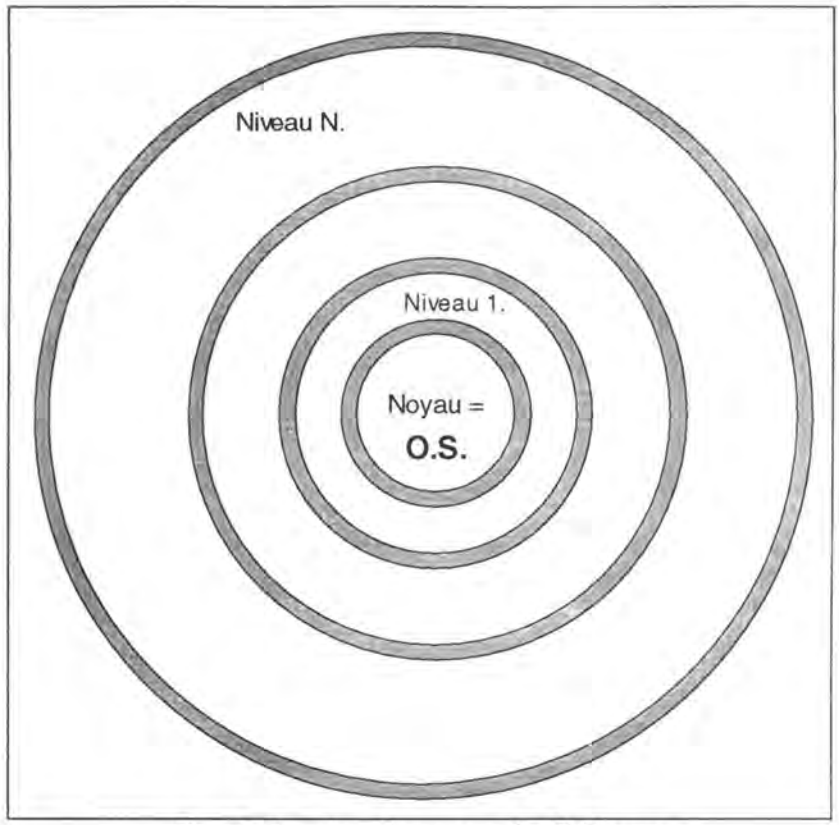


Fig. 6.1. : Architecture de l'Oignon

6.2.2 Apports de l'orientation objet

A cette découpe en niveaux hiérarchiques, nous allons adjoindre la puissance de la conception orientée objet. Il s'agira donc de définir une collection d'objets, de sortes que la spécification d'une couche de notre oignon devient :

a) La spécification des objets basiques constitutifs de la couche (remarquons qu'il réaliseront l'interface avec la couche inférieure).

b) La spécification de l'interface de ces objets.

c) La spécification du protocole d'utilisation de ces différents objets.

d) La spécification du mode de dérivation, des objets et protocoles de la couche inférieure, qui a présidé à la réalisation des objets de cette couche.

Dans le souci d'offrir de la portabilité à notre solution d'architecture de communication, il nous faut aussi veiller à ce qu'elle soit la plus indépendante possible

du système d'exploitation choisi. Pour ce faire, il nous faut considérer les objets de communication réalisant l'interface entre les deux premiers niveaux de l'oignon (niveau 0 et 1). Ces objets sont nécessaires à la réalisation des mécanismes de communication de l'oignon à objets de communication. L'implémentation de ces objets dépend bien entendu du type de système d'exploitation qui a été retenu comme pierre angulaire de l'architecture. Si une migration doit avoir lieu au niveau de ce système, il faudra selon toute vraisemblance recompiler ces objets. Mais afin de ne pas devoir envisager « la recompilation » de l'ensemble des objets constituant l'oignon, nous proposons d'opérer l'encapsulation de ces objets qui deviendront les objets de base de l'oignon. Pour plus de détails quant à ces encapsulations, nous invitons le lecteur à se référer au point 6.3 « Les objets basiques ».

Remarque : L'apport de la conception orientée objet permet aussi de garantir dans une certaine mesure l'extensibilité de l'architecture, mais cette fois au niveau d'une couche de l'oignon par l'adjonction possible d'un nouveau type d'objet.

Il s'agit donc d'une architecture découpée en couches hiérarchiques et composée d'objets réalisant des mécanismes de communication basés sur le modèle producteur/consommateur. Cette structuration nous permettra de définir une application comme un ensemble d'objets concurrents qui communiquent par le biais d'une mémoire partagée appelée objet de communication.

Dans les points qui suivront, nous nous proposerons de donner une vue plus fine des couches de niveau 1 et 2 qui représentent véritablement le noyau de communication de notre architecture. Ce noyau, suivant les principes énoncés au point 6.2.1, se greffera directement sur le système d'exploitation (couche de niveau zéro).

La question que nous devons nous poser, à ce stade de la réflexion, est de savoir quels sont les objets dont nous avons besoin pour réaliser ce noyau de communication.

6.3 Les objets basiques (niveau 1)

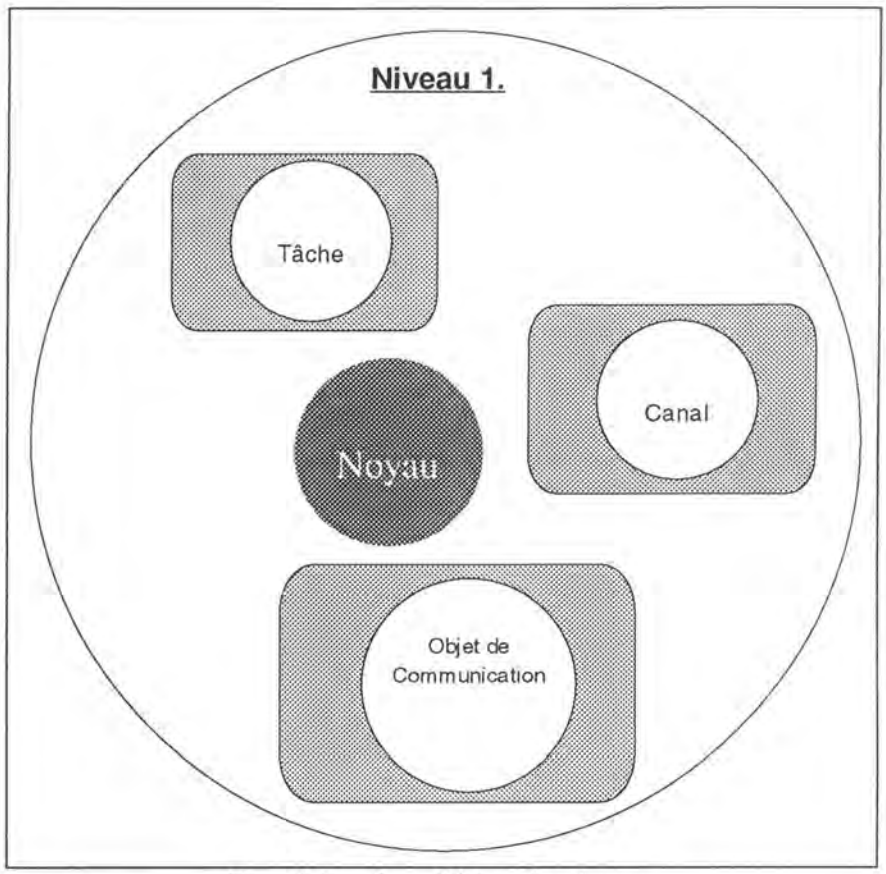


Fig. 6.2. : Les Objets basiques.

Comme le nom générique de cette section l'indique, et comme nous pouvons le constater sur la figure 6.2., nous retrouvons ici les objets constituant la base même du noyau de communication. Ils sont au nombre de trois : La tâche, l'objet de communication et le canal.

6.3.1 L'objet Tâche

Il s'agit de l'objet nécessaire à la gestion des activités en fonction de l'état de l'information. Cet objet devra supporter les fonctions classiques liés à la gestion d'activités, à savoir : la création, l'activation, la suspension et la destruction d'une activité. Pour ce faire, nous créons un type d'objet **tâche** dont l'interface est composée des quatre fonctions ou méthodes susnommées.

Il est à noter que le terme **tâche** doit être pris dans son acception large, c'est-à-dire un ensemble d'instructions séquentielles rendant compte d'un programme, d'une routine ou encore d'une interruption; en cela, il se rapproche du vocable « processus » sous UNIX.

Comme il a été dit précédemment, l'objet **tâche** sera encapsulé afin d'augmenter la portabilité de l'architecture. La prise en charge de cet objet sera intégralement de la responsabilité du noyau de communication, de sorte que l'utilisateur (cfr. programmeur) ne pourra en avoir l'accès. La création d'une tâche sera donc le résultat de l'instanciation, par un Agent, d'un objet de type tâche[1].

6.3.2 L'objet de communication

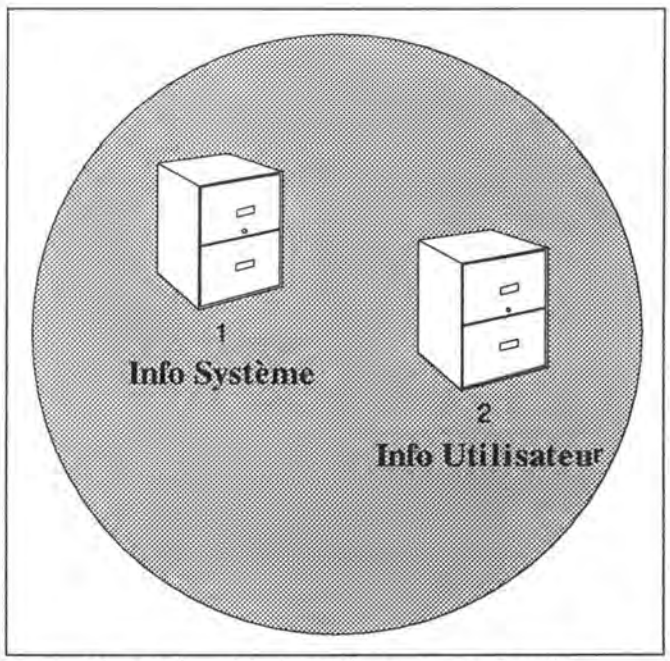


Fig. 6.3. : L'objet de communication

Les objets de communication sont les unités informationnelles non réutilisables que s'échangent les agents. C'est l'information partagée[2]. Ils sont donc stockés dans la mémoire partagée du système. La création d'un objet de communication, comme le montre la figure ci-dessus (Fig. 6.3.), se fait en deux étapes :

1. Le programmeur encapsule l'information voulue. Il s'agit de l'information utilisateur. Ensuite, le programmeur doit prévenir le système, par un code d'implémentation particulier de l'application communicante (instruction, macro, etc.

[1] Voir infra 6.4 Les Agents (niveau 2).
[2] Cfr. supra 4.3.2 Communication par données partagées.

qui constitue l'interface de cet objet avec le programmeur), de l'existence de ce nouvel objet de communication. Ainsi, le système pourra manipuler et gérer les objets de communication qui lui seront déclarés.

2. C'est à ce moment que le noyau de communication encapsulera, au sein de l'objet de communication nouvellement instancié et déclaré, les informations nécessaires au système. Au minimum, ces informations doivent contenir :

   - Une clef de validité (booléen) permettant un contrôle d'accès de l'objet de communication. La validation ne peut être faite que par le producteur de l'objet de communication. Elle sera aussi un mécanisme de synchronisation entre agents sur une donnée échangée, car seuls les objets de communication qui auront été validés pourront être consommés[1].

   - La liste des agents[2] qui doivent être prévenus lorsque l'information contenue dans l'objet de communication sera validée. Il s'agit en réalité de la collection des consommateurs sensibles à cette information. Nous identifierons ces « clients » par leur objet tâche.

   - Une adresse permettant au système d'exploitation de stocker l'objet de communication soit en mémoire locale, soit en mémoire globale. Il s'agit donc d'un pointeur alloué à la gestion de la mémoire dynamique.

   L'information nécessaire sera transparente, inaccessible à l'utilisateur. En effet, l'utilisation des mécanismes de la mémoire partagée génère dans bien des cas des sections critiques, où il y a lieu de mettre en oeuvre des mécanismes du système d'exploitation afin de générer des exclusions mutuelles. Cela est notamment vrai dans le cas de l'accès aux données de la clef de validité d'un objet de communication.

Enfin, l'objet de communication ainsi constitué connaît deux points d'entrées liés au type d'information utilisateur ou système.

Remarque : Les objets de communication appartiennent à la famille d'objets partagés dont la vie peut se résumer à : création, consommation et -c'est quasiment toujours le cas- destruction. Il n'y a donc pas de réutilisation de ces objets. Cette succession de manipulations ne permet malheureusement pas de rencontrer notre exigence d'une dynamique proche du celle du temps-réel. C'est pourquoi, nous suggérons que les utilisateurs de cette architecture, en d'autres termes les programmeurs, prévoient lors de l'initialisation de leurs applications, c'est-à-dire lors du seul moment où le temps-réel n'est pas exigé, de créer une collection d'objets de communications qui seront

[1]Voir infra 6.5 Quid de la synchronisation ?
[2]Voir infra 6.4 Les Agents (niveau 2).

typés selon les besoins estimés et qui seront stockés au sein d'un objet canal[1] origine. De la sorte, la création et la destruction d'un objet de communication seront respectivement le déchainâge et le chaînage de cette objet à son canal origine. Ainsi, nous pensons offrir des mécanismes efficaces de gestion de la mémoire dynamique.

6.3.3 L'objet Canal

L'objet **canal** est le support de base d'une communication de type producteur/consommateur entre un émetteur et un récepteur. Cet objet doit supporter :

1. La gestion de la queue d'objets de communication en attente de consommation. En effet, il va sans dire qu'un agent peut se voir adresser une collection d'objets de communication. Or, il ne peut les consommer que selon un mode séquentiel, c'est-à-dire, un à la fois. Afin de prévenir toute perte d'information, il faudra bien évidemment veiller à gérer, mémoriser cette multitude d'objets en attente.
2. Les transferts unidirectionnels d'objets de communication entre l'émetteur et le ou les récepteurs.

L'objet **canal** sera considéré par l'utilisateur (le programmeur) comme un mécanisme de stockage et de transfert d'objets de communication. Ces objets étant déclarés partagés, l'objet **canal** le sera tout autant.

D'une part, l'interface de l'objet **canal** sera composée de quatre méthodes classiques liées à la gestion des files d'attentes (cfr. la pile). Ces méthodes sont : créer la queue, vider la queue, ajouter un objet de communication et enlever un objet de communication. Notons que ces méthodes se doivent d'être :

1. Atomiques, afin d'assurer l'exclusion mutuelle.
2. Non bloquante, pour garantir une communication asynchrone. Pour ce faire, nous préconisons (voir figure 6.5.) que la gestion de l'objet Canal se fasse au travers d'une file d'attente des objets de communication à laquelle on adjoindra une seconde file comprenant les demandes de consommation issues du récepteur. Ainsi, un Agent peut faire une demande de consommation non bloquante d'un objet de communication, qui sera satisfaite dès que l'objet en question sera déposé et validé par l'émetteur.

[1]Voir infra 6.3.3 L'objet Canal.

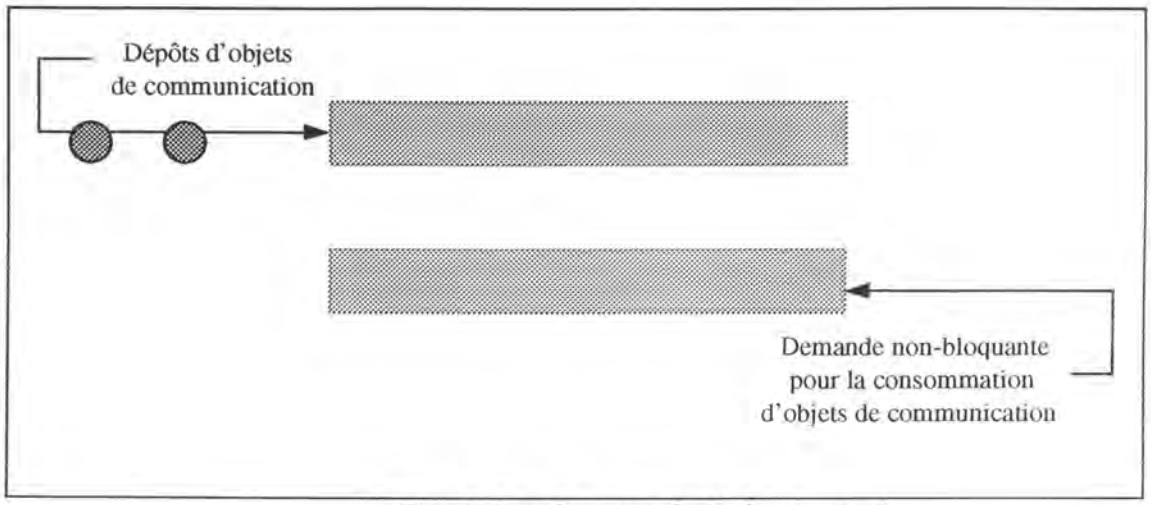


Fig. 6.5. : Gestion de l'objet canal

D'autre part, cet interface doit permettre à un agent de se mettre en attente d'un ou plusieurs objets de communication stockés sur un ou plusieurs canaux. Pour ce faire, il faut adjoindre un niveau de priorité à la notion de demande de consommation. Chaque **canal** regroupe donc des communications connaissant le même niveau de priorité tandis que la consommation proprement dite des objets de communication contenus dans un **canal** sera fonction du contexte d'exécution. Il pourra s'agir d'une logique temporelle de type LIFO ou bien FIFO, et/ou d'une "logique mixte".

Ici aussi, l'objet **canal** sera encapsulé, alors que son implémentation dépendra de la couche zéro (boîte aux lettres ou files d'attente).

6.4 Les Agents (niveau 2)

Maintenant que nous avons opéré l'identification des objets de base, nous pouvons passer à un niveau d'abstraction plus élevé et, par conséquent, définir la couche de niveau 2 (voir figure 6.6.). Celle-ci est composée par les objets de type **Agent** directement dérivés du concept d'acteur[1]. Un agent est en quelque sorte un objet qui joue le rôle d'un environnent d'activité composé de deux types d'objets appartenant à la couche inférieure de niveau 1. Il s'agit d'un objet tâche et 1 ou N objets canaux.

[1] Cfr. 4.4.2 Langages d'Acteurs.

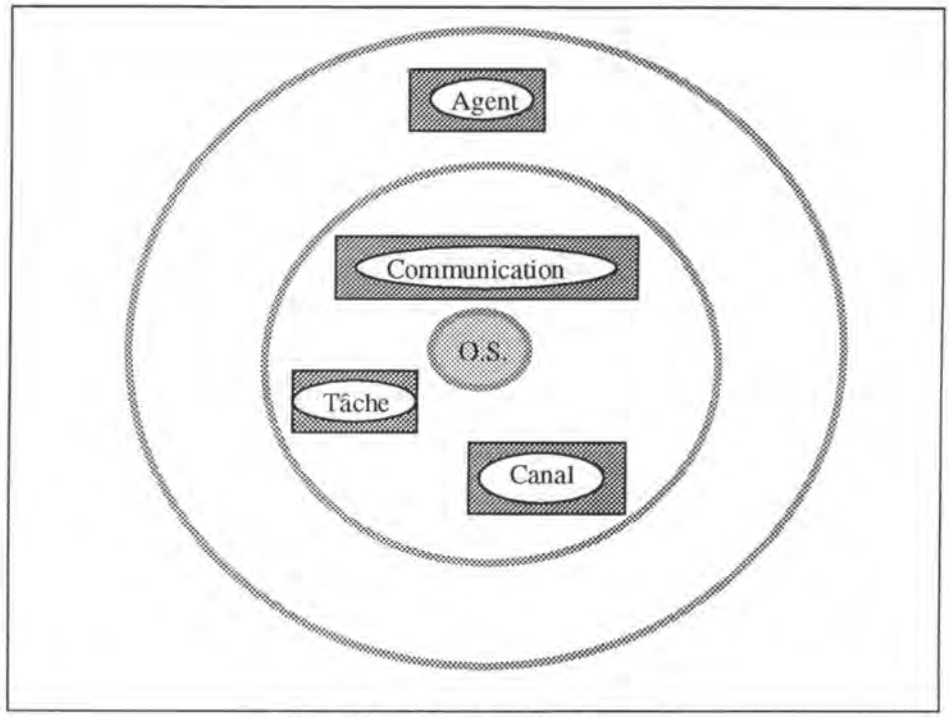


Fig. 6.6. : Couche de niveau 2

Les activités qui s'exécutent dans cet environnement sont en fait les processus séquentiels définis par les méthodes attachées à ces objets.

La vie d'un agent commence par l'instanciation d'un objet de type Agent. Lors de cette création, le programmeur se doit de définir, au sein du constructeur de l'Agent, une série de renseignements. Il s'agit :

1. l'activité de l'agent : c'est la section de code qui renseigne de la gestion de l'agent, elle contient la définition d'objets particuliers à l'agent, mais aussi la définition d'objets partagés ou encore de canaux de communication ;
2. la priorité de l'agent.

Cette définition pourrait, en pseudo-code, ressembler à ceci :

Activation_Def (code)
{Définition de l'activité de l'Agent }
Agent_Constr (priorite, Activation_Def, A)
{création d'un Agent de priotité "priority", dont l'activité est passée par paramètre et dont le point d'entrée est A}

Ensuite, l'Agent se fera connaître du système et donc de l'environnent de communication par la création d'un objet **tâche** de niveau inférieur. Pour rappel, cette création est faite par le système de communication, elle est donc transparente au programmeur.

6.5 Quid de la synchronisation ?

L'architecture que nous venons de présenter propose une communication entre agents par le biais d'objets de communication qui sont déclarés partagés. Il faudra donc synchroniser Ces agents communicants.

- D'une part, nous avons les agents récepteurs qui veulent se placer en attente de consommation et donc de validation d'un ou plusieurs objets de communication. Pour ce faire, les agents exécutent une primitive d'attente qui pourrait en pseudo-code ressembler à ceci :

Attente (ident, objet_liste, time-out, cléf, code)

{ La primitive Attente envoie l'identification de l'agent ainsi que la liste des objets sur lesquels l'agent veut se mettre en attente, la priorité des objets est fonction de leur place dans la liste. Cette primitive indique aussi le délai maximum d'attente au-delà duquel l'agent est prévenu, réveillé. Le résultat est un entier correspondant à la place de l'objet dans la liste qui a été validé (= 0 si time-out = true). Si plusieurs objets de communication sont validés simultanément, la clef retournée sera celle de l'objet le plus prioritaire. Attente retourne aussi un code dont la valeur indique une anomalie possible }

A l'invocation de cette primitive :

1. Le champs relatif aux agents consommateurs de l'objet de communication sera mis à jour. Ces agents seront identifiés par le processus représentant leur activité, c'est-à-dire l'objet **tâche** (paramètre Ident de la primitive Attente).
2. La méthode de suspension de l'objet **tâche** sera invoquée, ce qui aura pour effet immédiat d'endormir l'activité de consommation de l'agent récepteur jusqu'à la validation de l'objet de communication ou encore la survenance de l'événement time-out.

- D'autre part, nous avons le producteur qui valide un ou plusieurs objets de communication, autorisant par la même les agents sensibles à consommer ces objets. La primitive de validation pourrait en pseudo-code ressembler à ceci :

> ***Validation (objet_liste, code)***
>
> ***{ Avec objet_liste, la liste des objets validé et code analogue à la primitive Attente}***

A l'invocation de la primitive Validation, le champs clef de validité du ou des objets de communication validés est mis à vrai. Et l'objet tâche du ou des agents en attente est réveillé par un événement (méthode Activation du type d'objet tâche).

6.6 Synthèse

Nous avons défini un protocole de communication, en somme général, entre des agents concurrents. Il est basé sur un modèle producteur/consommateur asynchrone et implémenté à partir de canaux de communication unidirectionnels qui émettent et réceptionnent des objets de communication partagés. Cependant, il nous faut encore prendre en compte les notions de distribution (hétérogénéité) et de temps-réel.

Remarque : Afin de connaître une gestion optimale de la mémoire dynamique, nous pourrions tenir compte des enseignements liés au concept de manette détaillé au chapitre précédent. Ainsi, il suffirait de créer un objet de type Manette.

7. COMMUNICATION DANS UN ENVIRONNEMENT DISTRIBUE TEMPS-REEL

7.1 Introduction

Suite à la présentation de l'architecture de l'oignon à objets de communication, nous nous proposons d'en donner une extension qui puisse intégrer les impératifs de distribution et de temps-réel inhérents à notre application de pilotage automatique d'un véhicule.

Rappelons une fois encore, que cette application est le résultat de la coopération d'une collection de matériels, qui le plus souvent sont hétérogènes, et de leurs logiciels dédiés. Ils s'agit donc réellement d'une application distribuée tant sur le plan physique, qu'au niveau du logiciel. Cependant, l'architecture de communication que nous avons présentée au chapitre 6 « Architecture de l'oignon à Objets de Communications », ne permettra tout au plus à ces applications coopérantes que de communiquer. Mais, compte tenu de leur caractère hétérogène, elles auront bien du mal à se comprendre. De plus, l'environnement particulier[1] dans lequel devront se dérouler ces communications nous permet au minimum de mettre en doute leur fiabilité et donc de postuler l'existence d'erreurs de transmission d'informations.

C'est pourquoi, après l'introduction d'un nouveau concept (7.2 Nouveau concept : Les Objets Vues), il nous faudra affiner notre première architecture de communication (7.5 « Architecture de l'oignon à Objets Exportés et Objets Vues ») et proposer un protocole de communication par Objets Vues qui soit asynchrone, non-bloquant, de type producteur/consommateur et qui puisse supporter des fonctions de contrôle d'erreur ainsi que de la traduction automatique des informations échangées (7.6 « Protocole de communication par Objets Exportés et Objets Vues »). Nous disposerons alors de tous les éléments nécessaires à la réalisation d'un nouveau concept : l'agent éxecutif. Ce dernier devant évoluer dans un environnement distribué temps-réel (7.7.3 « Modèle d'un agent exécutif »).

[1] Cet environnement est composé du véhicule, son pilote et de son infrastructure directe (route, autres usagers, etc.). Il sera soumis à des contraintes importantes de fortes variations de températures , de champs éléctromécaniques et autres intempéries.

7.2 Nouveau concept : Les Objets Vues

La notion d'Objets Vues est directement dérivée de l'objet exporté et de la technique de « l'extract » (aussi appelé de la vue) utilisée dans les technologies des bases de données.

Pour rappel, et comme nous l'avons vu au point 4.4.1 « Langages de Classes », la communication de base du modèle d'objet s'appuie sur la notion de l'objet exporté. Ainsi, lorsqu'un objet consommateur invoque une méthode publique appartenant à un objet préalablement instancié, il voit son activité suspendue jusqu'à la terminaison complète de la méthode. L'instanciation d'un objet se fait localement à celui-ci.

Remarque : l'identification de l'objet exporté est locale à un site.

Cependant, il nous semble intéressant d'enrichir le concept d'objet exporté avec un mécanisme d'invocation à distance de méthodes dans un environnement distribué. Ainsi, comme nous le montre la figure 7.1., l'objet exporté est désormais composé des ensembles classiques d'informations privées (système) et publiques, auxquels on adjoint un ensemble de méthodes dont l'invocation peut être distante. Elles sont donc exportables. Les informations (méthodes) publiques et exportables représentent respectivement l'interface locale et l'interface distante de L'objet exporté.

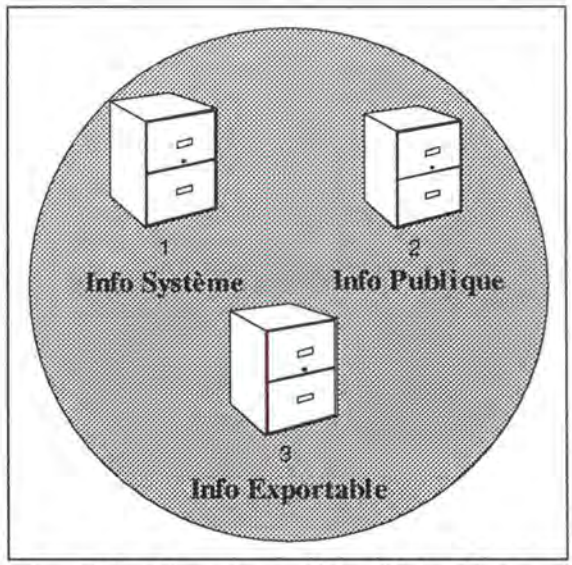


Fig. 7.1. : Structure d'un Objet Exporté

Il était encore nécessaire de pouvoir différencier le type d'invocation que l'objet exporté pouvait connaître, c'est-à-dire qu'il était nécessaire d'introduire le concept de distance dans l'invocation d'une méthode. Pour ce faire, nous introduisons le concept d'Objet Vue qui sera l'image fidèle d'un objet instancié dans un contexte distant.

Avec ceci de particulier que ses informations locales seront identiques à celles exportables de L'objet exporté modèle (cfr. Figure 7.2.).

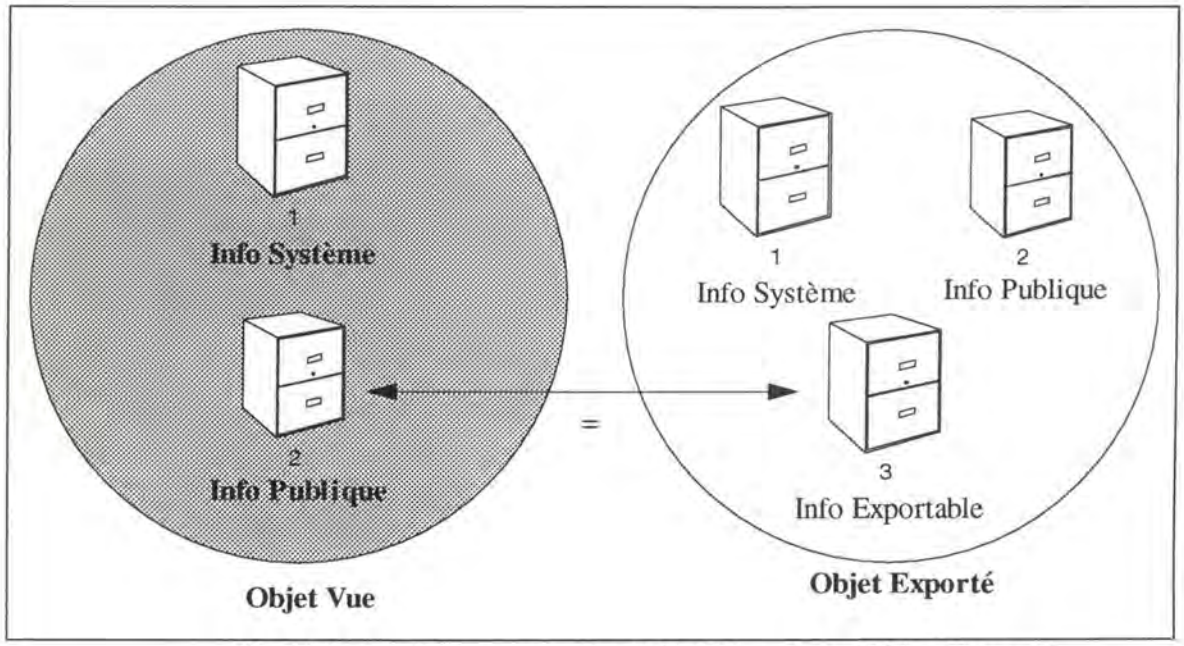


Fig. 7.2. : Relation entre un Objet Vue et un Objet Exporté

Ainsi, lorsqu'un consommateur désire communiquer avec un objet qui a été préalablement instancié dans un contexte distant, il lui suffit d'instancier localement son corollaire objet vue. Ensuite, il ne lui reste plus qu'à invoquer les méthodes publiques voulues dans cet objet vue. Cependant la réalisation de ces méthodes, comme nous le montre la figure ci-dessous (Fig. 7.3.), se déroulera bien évidemment au sein du contexte d'origine de l'objet exporté[1].

[1]Voir infra 7.5 « Architecture de l'oignon à Objets Exportés et Objets Vues ».

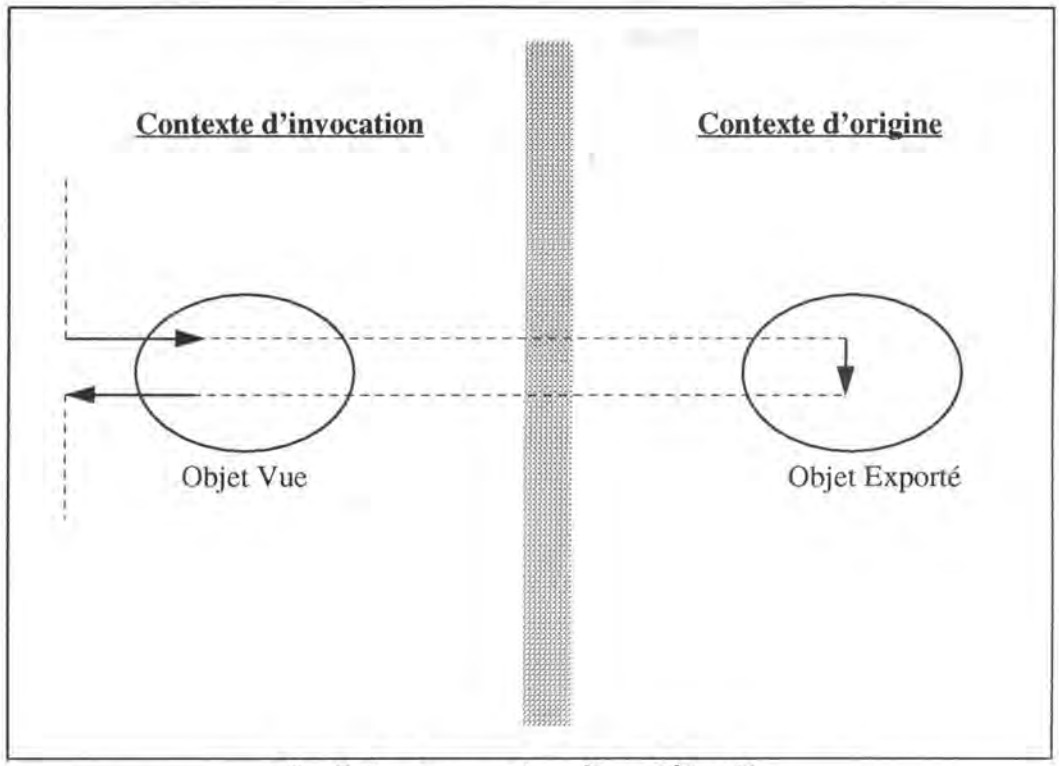


Fig. 7.3. : Invocation d'un Objet Vue.

Il est en outre remarquable de constater que cette identité des méthodes trouvera une réalisation simple et efficace au travers des mécanismes de polymorphisme[1] de fonctions offerts par le modèle Objet.

7.3 Problèmes liés au temps-réel

La principale lacune du protocole de communication du modèle Objet est le caractère synchrone des échanges. Pratiquement, lorsqu'un objet consommateur instancie dans son contexte (localement) un objet producteur d'informations, toute invocation d'une des méthodes de cet objet exporté a pour effet de suspendre l'activité de l'objet consommateur. Ceci peut aussi s'appliquer aux objets vues et leurs (pseudo) invocations locales (voir figure 7.3.). Or, dans le cadre de notre application d'un pilote automatique de véhicule, le système doit être réactif, c'est-à-dire être à tout moment à l'écoute de son environnement. Tout blocage d'activité, en particulier lié à une procédure de communication entre agents, est donc à proscrire.

C'est à la réalisation de cette tâche que les mécanismes liés à la communication par objets partagés, que pour rappel nous nommons objets de communication dans notre structure d'oignon, vont nous être du plus grand secours. En effet, nous allons encapsuler les invocations dans des données partagées de sortes que l'invocation d'une méthode d'un objet vue devienne non-bloquante. Le consommateur, en attente du service, ne verra pas son activité suspendue mais restera en contact avec l'objet

[1]Pour rappel, il s'agit de la posibilité de greffer la même interface (méthodes publiques) à des objets différents.

producteur afin de connaître le déroulement de la méthode invoquée. De plus, un même consommateur pourra connaître plusieurs liaisons avec le même ou avec d'autres producteurs. Un autre puissant aspect de cette méthode est que le producteur, à l'instar de son consommateur, ne voit pas son activité bloquée par l'invocation distante ou locale d'une de ses méthodes publiques ou exportables. Ce qui lui permet de gérer et répondre « simultanément » aux différentes requêtes. Pour ce faire, il devra gérer une queue des invocations en attente. Aussi, il lui sera loisible d'interrompre ses services aux consommateurs pour accomplir des tâches qui lui sont plus urgentes (par exemple réagir à un stimulus de son environnement). Pour réaliser cette dernière propriété, il suffira d'affecter un canal à chaque invocation[1].

Cette introduction de mécanisme d'encapsulation et de partage des données seront bien évidemment transparents au programmeur, et le consommateur procédera de la même façon à l'invocation locale et distante. De plus celles-ci seront non-bloquantes.

7.4 Problèmes liés à la distribution (hétérogénéité)

Afin de répondre au déficit d'unicité dans la représentation des données, du temps ou encore de la désignation des services, nous incorporons à notre modèle une horloge universelle ainsi qu'un traducteur automatique des échanges.

En fait de traducteur, il s'agira d'une méthode privée de l'agent producteur de l'information. La traduction se fera donc localement à celui-ci, en un langage convenu (par exemple ASCII) lors de l'ouverture du lien entre les parties communicantes. Remarquons que cette méthode ne permet pas de garantir un temps de services constant et indépendant du lieu (site) du consommateur. En effet, dans le cas d'une communication entre agents tournant sur des matériels et sous des logiciels homogènes, la phase de traduction n'est plus nécessaire. Le temps de service connaît donc des variations.

7.5 Architecture de l'oignon à Objets Exportés et Objets Vues

Cette architecture est une extension de l'architecture de l'oignon à Objets de Communication que nous avons vue au point 6. Elle en reprendra donc les trois premières couches. Le concept d'Agent constituera le contexte d'exécution de notre architecture; alors, les Objets de Communication et Objets Canaux permettront de

[1]Pour rappel, on peut associer à un canal un niveau de priorité.

réaliser les mécanismes de partage d'objets et donc de garantir la synchronisation des Agents Consommateurs et Producteurs.

A ces trois premières couches, nous en grefferons une troisième (couche de niveau 3). Intuitivement et au vu de ce qui a été dit précédemment, le lecteur pourra aisément en identifier les composantes. Il s'agit des Objets Exportés (7.5.1), Objets Vues (7.5.2) mais aussi L'objet Catalogue (7.5.3).

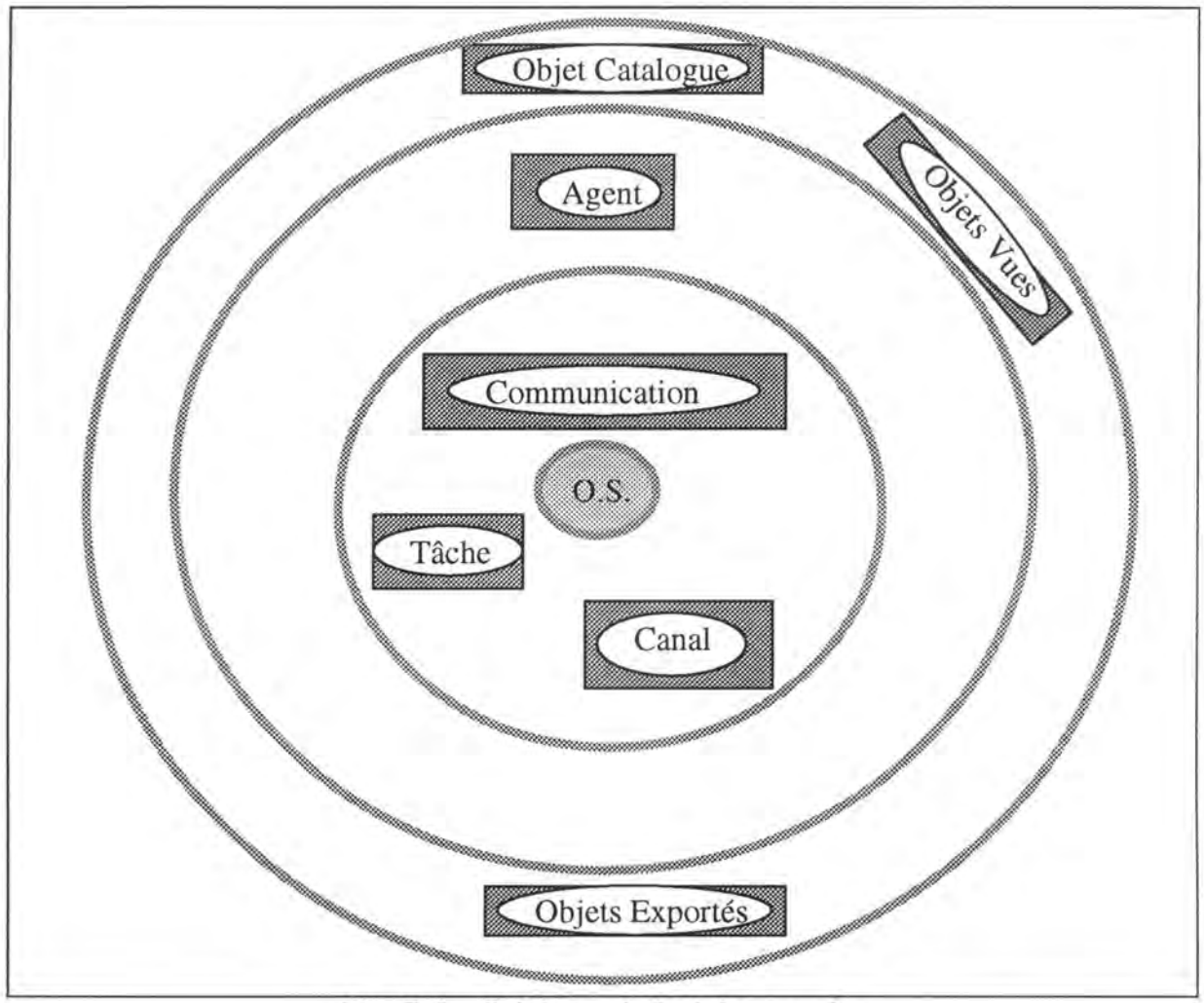


Fig. 7.4. : Schéma général des couches

7.5.1 Objets Exportés

Ayant déjà précédemment abordé le concept des objets exportés, il ne nous reste plus qu'à envisager de quelle manière nous pourrons spécifier l'interface distante (méthodes exportables) de l'objet exporté.

La solution nous paraît assez banale, il s'agira lors de la définition d'une classe, c'est à dire lors de la définition d'un type d'objet, de spécifier la collection de méthodes qui appartiendront à l'interface distante. Bien entendu, cette spécification sera valide pour toutes les instances de cette classe[1], elle sera statique.

7.5.2 Objets Vues

Puisque la définition des interfaces distantes des objets exportés se font de manière statique, il nous sera possible de demander au compilateur de créer autant de classes d'objets vues qu'il y aura de classes d'objets exportés. Et à chaque instanciation distante d'un objet exporté, le système instanciera localement au consommateur un objet vue qui le représentera.

7.5.3 L'objet Catalogue

Comme nous l'avons vu[2], les objets exportés sont identifiés localement à leur site. Ceci ,bien évidemment, ne permet donc pas de localiser un tel objet au niveau de l'architecture tout entière. C'est pourquoi, il faudra affecter à chaque objet exporté un identifiant global. Ce peut être, par exemple, la concaténation de l'identifiant local de l'objet et du nom du site - de la machine - sur lequel il s'exécute.

A l'instar de ce que peuvent réaliser les systèmes de gestion de fichiers, nous implémenterons notre catalogue sous forme d'une arborescence d'identifiants d'objets exportés réalisant par la même occassion une identification de tous les services publics et privés de l'application distribuée et de leurs accès respectifs. Chaque machine connaîtra donc un catalogue reprenant tous les services qu'elle offre et des pointeurs vers les catalogues accessibles à partir de ce site. Comme on peut le voir dans la figure

[1]Cfr. définition d'une classe d'un langage de classe.
[2]Voir supra 7.2 « Nouveau concept : Les Objets Vues ».

7.5., accéder à un service consiste simplement à parcourir le catalogue de la machine locale afin d'en connaître la localisation.

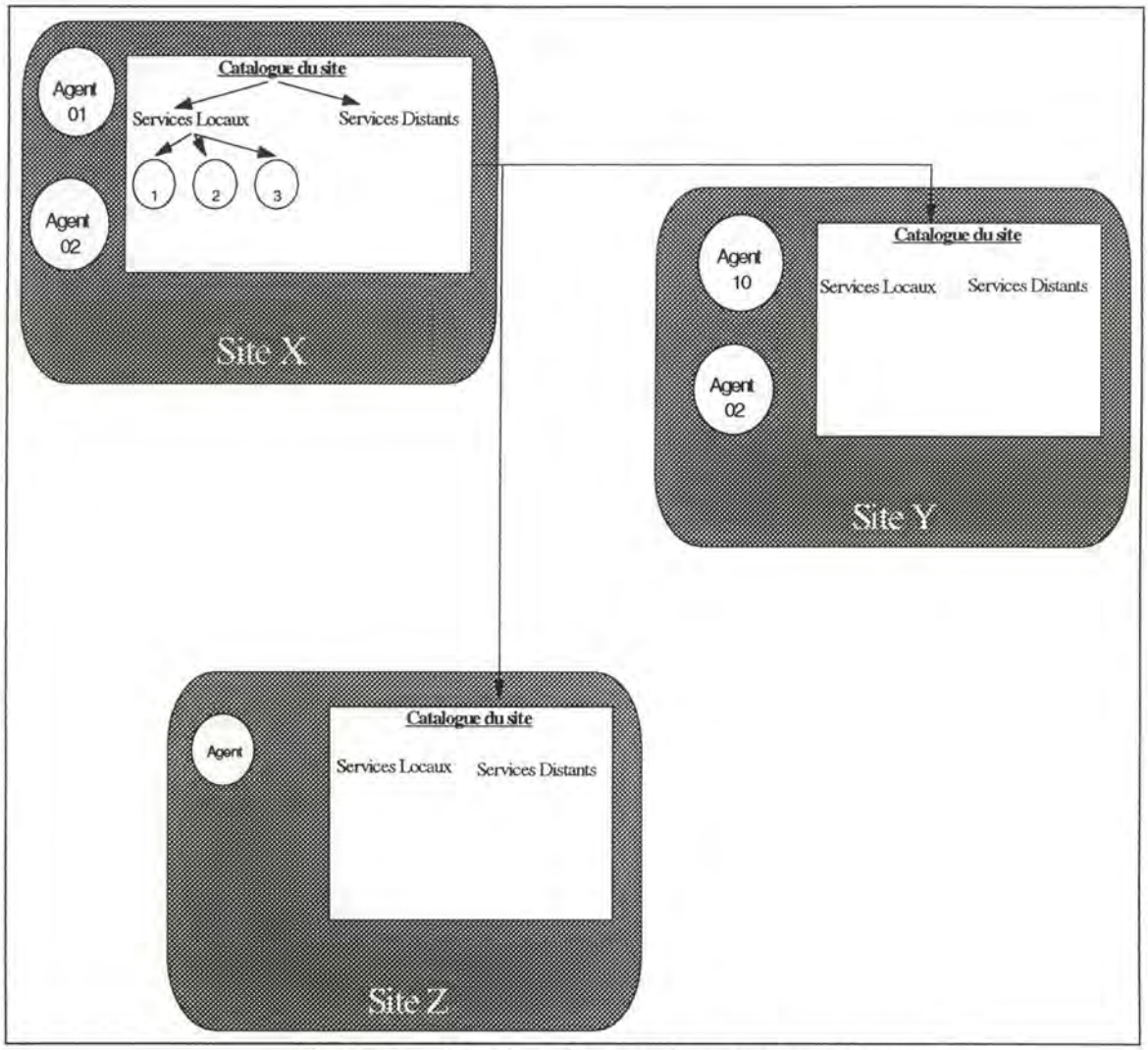


Fig. 7.5. : Gestion des services par catalogue

L'interface de l'objet catalogue est directement accessible au programmateur et composée de deux méthodes :

1. "Pub" que nous détaillons plus au point 7.6.2 « Création d'un service ».
2. La localisation et l'ouverture d'un lien entre l'agent consommateur et l'agent producteur du service (voir infra 7.6.3 « Ouverture d'un lien »).

Remarque : Le catalogue global connaît lui aussi une distribution, puisqu'il est le résultat du chaînage d'un ensemble de catalogues locaux.

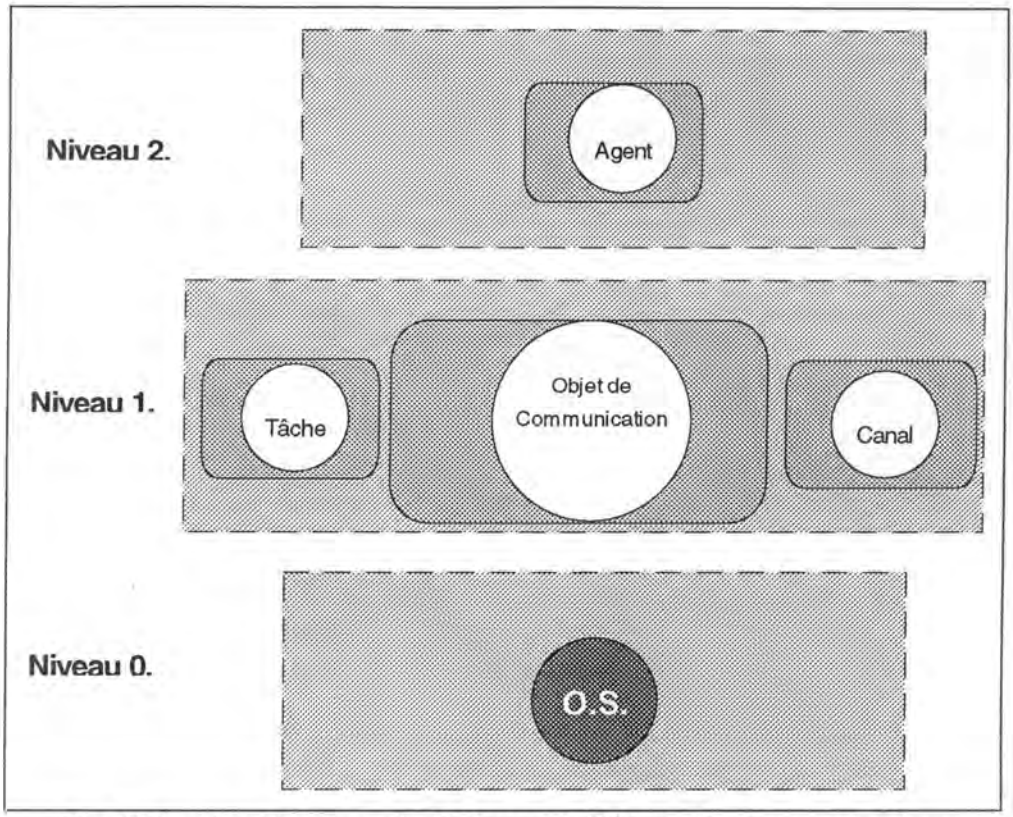


Fig. 7.6. : Architecture de l'oignon à Objets de communication.

7.6 Protocole de communication par Objets Exportés et Objets Vues

Après avoir répondu aux problèmes liés aux environnements distribués (7.3) et temps-réel (7.4), nous nous attacherons à donner la dynamique, le scénario de communication entre deux agents.

7.6.1 Dynamique de communication

Pour qu'une communication puisse avoir lieu entre deux agents, il faut que les agents producteurs d'informations se fassent connaître des consommateurs potentiels par la création d'un service (7.6.2). Ensuite, le consommateur pourra réaliser un lien avec le producteur (7.6.3). La consommation du service peut dés lors commencer (7.6.4). Ensuite, il faut détruire le lien (7.6.5) et éventuellement le service (7.6.6).

7.6.2 Création d'un service

La création d'un service se résume à la création, par instanciation, d'un Objet Exporté. Cette création est le fruit d'un agent producteur d'informations. Comme nous l'avons déjà vu, l'agent producteur veillera à affecter un canal à toutes les méthodes exportables. Alors que la publication de l'Objet Exporté sera supporté par une requête de L'objet Catalogue du système dont nous donnerons la structure au point suivant.

En pseudo code, cette requête ou fonction pourrait ressembler à ceci :

Pub (Objet, liste_Méthodes, liste_Canaux)

{ Objet identifie l'Objet Exporté, alors que liste_Méthodes et liste_Canaux sont respectivement deux pointeurs vers la liste des méthodes rendues publiques et la liste des canaux qui leur sont affectés. Rappelons que l'ordre de ces listes détermine la priorité des méthodes. }

7.6.3 Ouverture d'un lien

Afin de réaliser une communication entre deux agents (l'un producteur et l'autre consommateur de services), ceux-ci doivent préalablement ouvrir un lien. Deux cas de figures peuvent se présenter :

1. Le client ne connaît pas les arguments nécessaires (par exemple : l'identificateur de l'Objet Exporté source) à la création de l'Objet Vue contenant des services qu'il désire obtenir. Ils pourront lui être fournis par un service de l'Objet Catalogue. C'est alors seulement qu'il pourra créer l'Objet Vue (point 2).
2. Le client, muni des arguments requis, crée l'Objet Vue. Cet objet, par le biais d'une de ses méthodes, se mettra en rapport avec l'Objet Catalogue afin d'ouvrir le lien avec le producteur. Cette ouverture du lien permettra en particulier de fixer le type de représentation des données choisi pour communiquer.

7.6.4 Transfert d'informations

Comme nous le montre la figure 7.9., la communication se fait de manière asynchrone et non-bloquante. Il y a donc principalement deux étapes :

1. L'invocation d'une méthode de l'Objet Vue. Pour rappel, les termes de l'invocation vont être encapsulés au sein d'un objet de communication qui est partagé. Cet objet va emprunter le canal qui lui est assigné en fonction de sa priorité. Le producteur de la méthode invoquée reçoit cet objet et retourne un écho de bonne réception et acceptation de l'invocation. Notre agent consommateur peut, en attendant l'exécution du service, vaquer à d'autres occupations.
2. La réalisation effective du service. On peut en outre imaginer d'ajouter à cette réalisation un écho du déroulement.

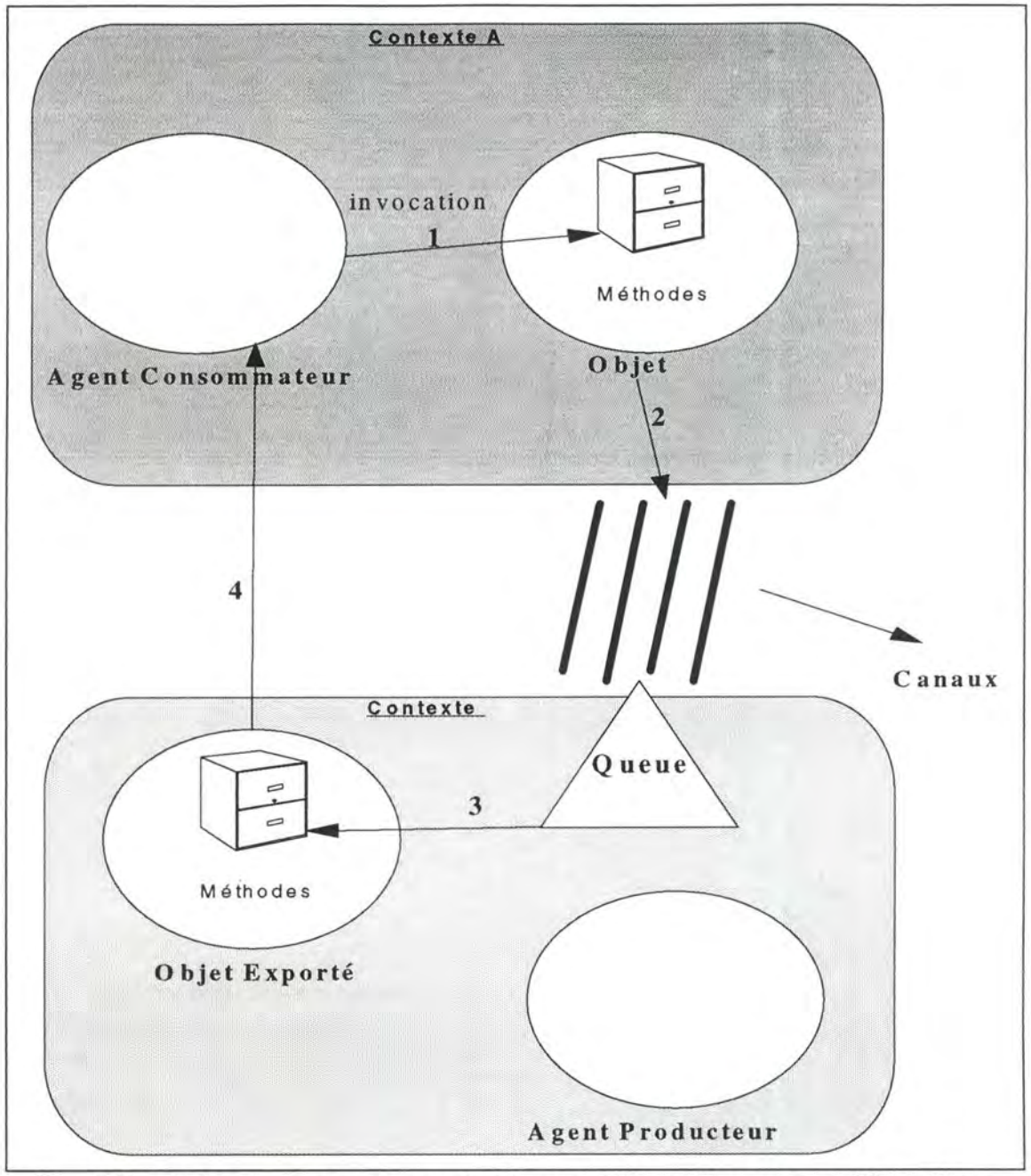


Fig. 7.7. : Mécanisme de communication par Objets Exportés et Objets Vues.

7.6.5 Fermeture du lien

La destruction d'un lien de communication se fera, de manière explicite, par l'exécution de la méthode de destruction de l'Objet Vue.

7.6.6 Destruction du service

Si un objet producteur estime que ces services sont devenu obsolètes, il lui est loisible de les détruire et ce par la destruction sine die de l'Objet Exporté. Remarquons que les canaux utilisés par cet Objet ne doivent pas nécessairement être détruits, mais peuvent tout simplement être réalloués à l'objet producteur afin de supporter la communication d'un autre Objet Exporté.

7.7 Modèle d'un agent exécutif temps-réel

7.7.1 Introduction

La décomposition d'un système de contrôle en agents intelligents coopérant afin de réaliser un objectif global a amené au paradigme multi-agents (voir multi-robots) qui se focalise le plus souvent sur les capacités décisionnelles et cognitives des agents aux dépens de l'aspect exécutif de ceux-ci. On constate donc souvent l'abandon de cette approche originale lorsque l'on s'intéresse à la mise en oeuvre des agents et donc à leur aspect exécutif.

Implémenter un agent exécutif pose de nombreux problèmes dus essentiellement à la diversité des matériaux (tant hardware que software). Il faut en effet faire face à des représentations de données qui varient suivant le type de machines ou de langage utilisés, mettre en oeuvre de façon simultanée des applications compilées et interprétées, synchroniser un ensemble d'agents suivant des contraintes temps-réel, et surtout, utiliser différents équipements ayant chacun leur propre structure de commande.

Nous avons donc fusionné l'approche multi-agents et la conception orientée objet de telle sorte que les logiciels de contrôle/commande puissent être structurés selon les principes de la conception orientée objet tout en étant perçus par les systèmes décisionnels comme un ensemble d'agents autonomes et indépendants.

Nous présenterons dans ce chapitre le modèle d'un agent exécutif permettant de concevoir une application de contrôle commande temps-réel sous forme d'agents autonomes pouvant s'intégrer facilement au sein d'un groupe.

7.7.2 Classification des systèmes multi-agents

Les systèmes multi-agents sont utilisés pour résoudre deux classes de problèmes : les systèmes multi-experts et les systèmes multi-robots [Fer, 88].

- Les systèmes multi-experts utilisent des agents qui coopèrent afin de résoudre un problème global. Pour communiquer, ces agents utilisent une mémoire commune. Dans cette mémoire commune, ou « blackboard », chaque agent lit et écrit des messages concernant ses buts et ses intentions.
- Dans les systèmes multi-robots, les agents sont des entités physiques (capteurs, processeurs ...) ou abstraites (lois de commande ...) qui agissent sur leur environnement ou sur eux-mêmes. Contrairement aux systèmes multi-experts, les agents d'un système multi-robots ne connaissent l'environnement qu'au travers d'une perception propre liée à leurs moyens de communication.

Chaque système peut être construit suivant un modèle à grain fin ou à gros grain. La granularité des agents est fonction de la tendance que l'on veut donner au modèle.[Fer, 88]

- Les systèmes à forte granularité sont issus de l'école dite cognitive et utilisent des agents intelligents et fortement autonomes. Ces agents sont donc capables de réaliser individuellement des tâches complexes.
- Les systèmes à grains fins correspondent à l'école réactive [Bro ,89]. Dans un tel système, l'agent est considéré comme un noeud aux fonctionnalités très simples se limitant à des réactions de type stimulus/action. Le comportement intelligent n'est alors produit que par l'interaction de plusieurs agents non intelligents. Un exemple extrême de cette tendance se retrouve dans les réseaux de neurones.

La figure suivante (fig. 7.8.) synthétise notre classification sommaire des systèmes multi-agents.

Systèmes multi-agents			
Systèmes multi-experts		**Systèmes multi-robots**	
Grain fin	Gros grain	Grain fin	Gros grain

Fig. 7.8. Classification des systèmes multi-agents

7.7.3 Modèle d'un agent exécutif

La classe de problèmes qui nous intéresse est bien celle des systèmes multi-robots et de leurs logiciels de contrôle-commande. Nous préconisons une approche réactive (donc à grains fins) dans laquelle un agent sera plus ou moins intelligent et admettra surtout une forte capacité de communication lui permettant de s'intégrer à un groupe déjà constitué de différents agents.

Il apparaît qu'un système de contrôle/commande résulte de la synergie de disciplines nombreuses que nous pouvons regrouper en deux domaines : celui de la commande et celui de la conduite (figure 7.11.). Ces deux domaines présentent des approches différentes, voire contradictoires, puisqu'au niveau de la conduite, le paradigme multi-agents est extrêmement répandu alors qu'au niveau de la commande on utilise plus souvent une approche procédurale utilisant comme unité de décomposition une fonctionnalité de l'agent et non l'agent lui-même. Notre but est de rapprocher ces deux domaines afin de créer un agent exécutif de commande dans un environnement Temps-Réel distribué. Nous pensons qu'un tel agent facilitera la communication entre les logiciels issus des deux domaines.

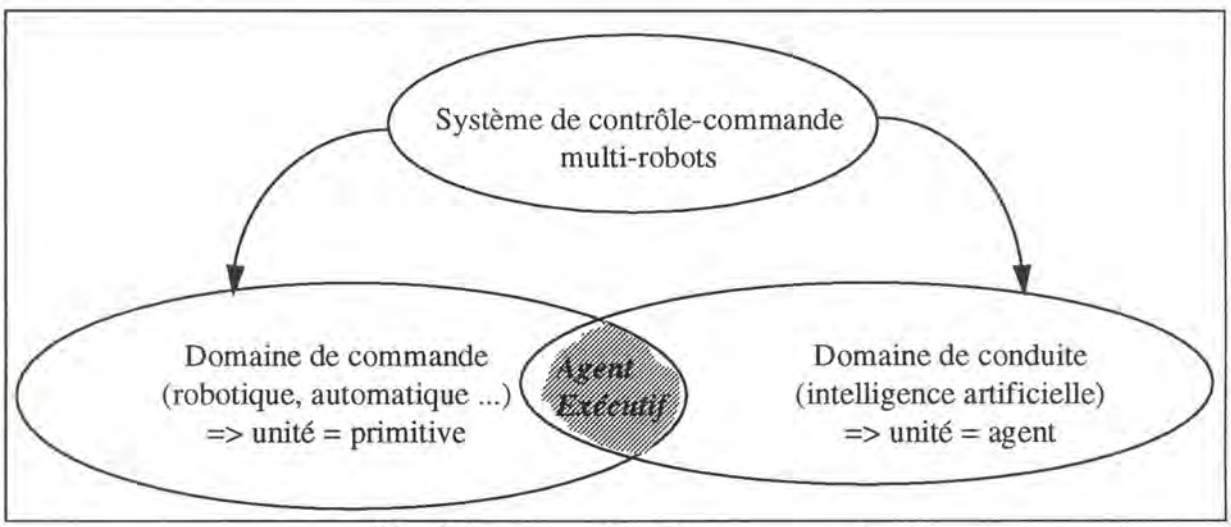


Fig. 7.9. : Domaine d'un agent exécutif

Pour construire un tel modèle, il est nécessaire de dégager les paramètres pertinents d'un agent en tant qu'entité individuelle et en tant qu'individu dans un groupe. Nous savons que, dans les systèmes multi-robots, le contrôle peut être centralisé, réparti ou mixte et que les agents doivent être assez autonomes pour pouvoir réaliser individuellement certaines opérations tout en dépendant d'un agent de contrôle pour les opérations plus complexes. Nous ne pouvons par contre faire aucune hypothèse quant au nombre d'agents utilisés, ni quant à leur niveau de granularité. En effet, le modèle doit permettre d'implémenter aussi bien des agents à forte granularité que des agent à grain fin[1]. Nous pouvons toutefois considérer que tous les agents sont spécialisés, leurs spécialités étant connues de tous ceux avec qui ils coopèrent.

Comme les agents doivent coopérer afin de réaliser une tâche globale, ils doivent échanger des informations et se synchroniser pour coordonner leurs actions. Comme nous considérons les différents équipements et logiciels composant l'application multi-robot comme des agents, ceux-ci doivent disposer de moyens de communication pour pouvoir dialoguer entre eux. D'autre part, comme ces équipements évoluent dans un environnement dynamique, ils doivent réagir rapidement à certains changements de cet environnement. Par conséquent, la communication entre agents doit pouvoir se faire de manière asynchrone pour rendre compte du caractère temps-réel de certains événements.

Il s'agit donc d'une architecture relationnelle orientée (figure 7.12.)[2] où chaque client connaît les services qui lui sont utiles mais sans connaître ses clients. Nous pouvons ainsi garantir l'évolution du système.

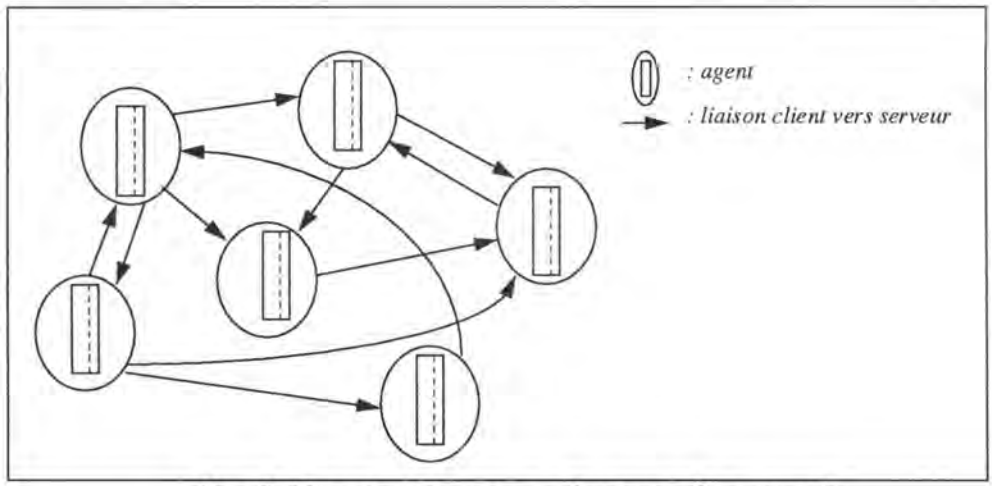


Fig. 7.12. : Architecture relationnelle orientée

[1] Par exemple, nous aurons des agents très simples pour la surveillance des capteurs (caméras, récepteur RDS, odomètres ...) et des agents beaucoup plus complexes, à même de planifier leurs actions, pour le contrôle et la conduite du véhicule.

[2] Remarquons que le schéma n'exclut pas l'éventualité d'un bouclage où un agent peut être à la fois client et serveur.

Dans une telle architecture et suite aux considérations de la page précédente, nous pensons que le modèle fonctionnel d'un agent exécutif admettant les quatre capacités suivantes (figure 7.13.) : **spécialisation**, **délégation**, **contrôle** et **communication** représentera parfaitement notre agent exécutif.

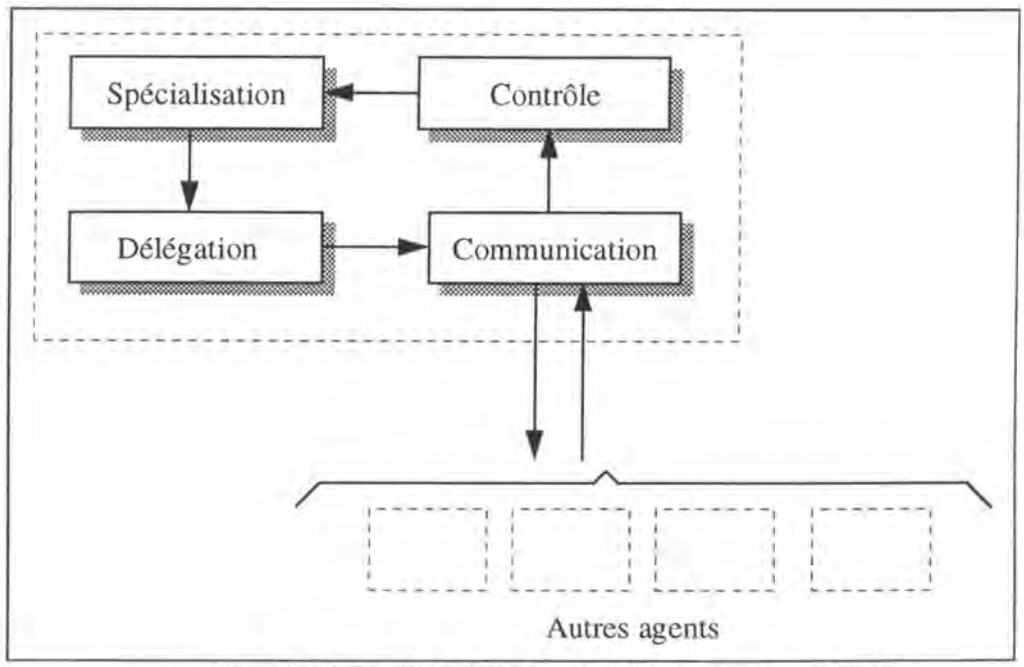


Fig. 7.13. modèle fonctionnel d'un agent exécutif

La capacité de spécialisation traduit les compétences et le savoir-faire de l'agent dans un ou plusieurs domaines. La spécialisation d'un agent est caractérisée par des mécanismes internes et une interface externe. De plus, cette spécialité doit être accessible à d'autres agents[1] pouvant être distants et/ou hétérogènes.

La capacité de contrôle représente le caractère indépendant de l'agent, ce qui lui permet d'acquérir un certaine autonomie. L'agent est en effet le seul maître quant à la gestion de ses compétences. Ainsi, l'importance qu'il accorde à une requête est de son unique appréciation. Il est donc de son ressort de suspendre le traitement de la requête courante[2] pour tenir compte d'une requête plus urgente.

La spécialisation de l'agent ne lui permettant pas de couvrir tous les domaines de compétences, grâce à sa capacité de délégation, l'agent délègue certaines tâches à d'autres agents dont les compétences sont indispensables pour réaliser ses objectifs. Au niveau de l'implémentation, pour déléguer, un agent doit connaître les spécialités de ses accointances[3]. Ces spécialités sont représentées par des objets exportés,

[1] L'objet exporté défini au chapitre précédent peut être considéré comme le modèle d'implémentation d'une spécialité d'un agent. Les méthodes exportées représentent alors les différents services qu 'un agent peut offrir.

[2] Rappelons que nos agents n'ont jamais qu'une seule activité. (Cfr. Modèle de communication par objets partagés.)

[3] Les accointances sont les agents auquels il délègue une tâche.

accessibles à distance par l'intermédiaire des objets images. Nous pouvons donc considérer les objets images comme les représentants des accointances d'un agent. L'objet vue encapsulant les mécanismes de localisation, de transport et d'accès à l'objet réel, la communication entre un agent et ses accointances peut alors se faire par simple invocation et indépendamment de leur localisation.

Enfin, la capacité de communication d'un agent lui permet de s'intégrer dans un groupe mais aussi de s'ouvrir sur le monde extérieur. Au niveau de l'implémentation, ceci est réalisé par des objets partagés ou par des objets vues. Les communications sont bien sûr asynchrones et toujours indépendantes de la localisation.

8. CONCLUSION.

8.1 Bilan

Après avoir cerné le concept de « véhicule intelligent » et, de manière plus générale la problématique liée à la mobilité (situation actuelle et enjeux), il nous est apparu qu'une des voies de recherche largement privilégiée par des autorités tant nationales (U.S.A.,France,Japon, ...) que supra-nationales (C.E.E.) réside dans la réalisation -à court terme- d'aides au pilotage et -à long terme- d'un pilote automatique de voiture.

Du point de vue informatique, la diversité des matériels utilisés, leur dispersion au sein du véhicule (centrale de gestion des freins, des trains roulants, navigateur ...) nous a amenés à proposer un support constitué d'un réseau local en fibres optiques. Afin de rencontrer un niveau de sécurité élevé, inhérent au transport de personnes, nous proposons en outre que ce L.A.N. connaisse une structure en étoile dédoublée.

S'agissant du système de pilotage, des conclusions similaires s'imposent : nous requérons donc que son architecture soit distribuée, extensible et portable tout en respectant des contraintes temps-réel. Cette notion de coopération nécessite des outils de communication universels et performants permettant un asynchronisme non blocant.

Très vite, lors du rescencement des mécanismes classiques de communication proposés par les systèmes existant, il nous est apparu qu'aucun d'eux ne satisfaisait à nos exigences.

Afin de respecter les impératifs d'extensibilité et de portabilité maximale, nous nous sommes dotés d'une méthodologie de structuration des données adéquate : l'approche objet.

Pour réaliser des communications efficaces, nous avons d'abord envisagé un modèle de communication par objets partagés garantissant des échanges asynchrones. La création d'un objet **manette** nous a permis de rendre transparente au programmeur la gestion des objets partagés tout en gérant la mémoire dynamique de façon optimale. La définition d'un second objet, le **canal**, nous a permis, contrairement à la manette, des communications inter agents dans un environnement distribué.

L'architecture à oignon de communication de notre système est directement dérivée du modèle de communication par objets partagés. Cependant, elle n'intègre pas encore les impératifs de distribution et de temps-réel.

Afin de palier à ces dernières insuffisances (distribution et temps-réel), nous avons créé un objet **vue** permettant d'invoquer des services à distance et nous affectons à chaque invocation un canal dont la priorité rend compte du caractère temps réel ou non du service requis.

Nous disposons dès lors d'une architecture constituée de quatre couches (l'O.S., la couche des objets basiques, la couche agent, la couche des objets vues et exportés). Cette architecture permet des communications inter agents dans un environnement distribué connaissant des contraintes temps-réel.

Les agents utilisés dans notre systèmes sont des agents exécutifs. Ils disposent des quatre capacités suivantes : spécialisation,délégation,contrôle et communication, leur permetant de s'intégrer facilement dans une architecture relationnelle distribuée. L'approche que nous préconisons pour leur implémentation est de type réactif, elle permettra d'ailleurs une meilleure extensibilité du système.

8.2 Perspectives

Conscients que notre travail de fin d'étude s'inscrivait dans une démarche prospective du concept de véhicule intelligent, nous avons pris soin de relever les grands domaines d'études suceptibles d'enrichir ce concept novateur. En effet, il apparaît que l'informatisation d'un véhicule n'est pas l'apanage des seuls informaticiens, mais constitue plutôt un carrefour pluridisciplinaire.

Nous proposons de dresser un bref aperçu des possibilités de recherches dans des domaines scientifiques tels que : la robotique, l'économie, la psychologie, l'épistémologie et ,bien entendu, l'informatique.

8.2.1 Robotique/électronique

Tant au niveau de l'infrastructure qu'à celui du système embarqué, il reste encore beaucoup à faire. Il s'avère en effet que l'équipement nécessaire à la réalisation du concept de véhicule intelligent sera de plus en plus important, volumineux, lourd et coûteux. En particulier, le nombre de câbles connaîtra une véritable explosion, il serait donc pertinent de développer des mécanismes de multiplexage se greffant sur un backbone de communication et d'alimentation de type LAN.

De nombreux mécanismes de perception tels que boucles magnétiques à induction, émissions radio, balises, centres de contrôles, satellites ... doivent être mis en oeuvre afin de réaliser un guidage dynamique efficient. Cette infrastructure naissante nécessite encore de nombreuses procédures d'intégration et d'optimisation des différents matériels.

8.2.2 Economie

Nos voitures, devenant de plus en plus intelligentes, seront de plus en plus coûteuses. Il faut donc se poser la question de la viabilité économique d'un tel type de véhicule. Cette démarche est loin d'être simple. En effet, il faut prendre en compte d'innombrables facteurs tels que :

- le coût des investissements en infrastructure ;
- le coût du véhicule ;
- le marché potentiel ;
- les réductions de consommation, de pollution ;
- l'accroissement de la mobilité ;
- la réduction du nombre d'accidents et par conséquent des primes d'assurances ;
- un redéfinition de l'équation transports publics/transports privés .

En outre, il nous semble que la volonté politique de la Direction Générale 13 des Communautés Européennes (Transports) est de promouvoir de larges projets abondant dans le sens du véhicule intelligent. Cette optique s'inscrit dans une politique de type Keynésienne[1] de relance de l'économie Européenne par des grands travaux.

Pour l'heure, le problème principal est d'évaluer quelle serait la masse critique d'utilisateurs à atteindre pour rendre le projet viable. Pour ce faire, il est clair que, à l'instar du développement du Minitel en France, les autorités nationales et supra-nationales (C.E.E.) devront mettre en oeuvre des aides massives à l'investissement et à la consommation d'un tel produit.

8.2.3 Psychologie

Le caractère hautement interactif d'applications telles que le guidage nécessite encore de nombreuses recherches pour offrir des interfaces de qualité. Les guidages vocaux et visuels de CARMINAT sont certes un bon point de départ mais le synthétiseur de parole est encore trop rudimentaire et l'enplacement de

[1] Cfr. Le « Livre Blanc » de J. Delors.

l'écran ne respecte pas les critères ergonomiques en vigueur. C'est pourquoi, nous préférons l'approche prônée par l'incrusteur d'images sur le pare-brise. Il reste cependant un long chemin à parcourir avant d'apprécier au mieux la charge mentale liée à une activité de pilotage. Ensuite, seulement, il sera possible d'atteindre une standardisation nécessaire de ce type d'interface [SMI, 89] (Cfr. la métaphore du mini-monde pour les PC).

8.2.4 Epistémologie

Nous abordons ici un domaine très sensible mais curieusement fort peu développé dans la littérature. Nous épinglerons principalement deux pierres d'achoppement.

Premièrement, l'autolocalisation des véhicules ne constitue-t-elle pas une grave entrave à nos libertés individuelles? Pour expliciter notre propos, nous prendrons un exemple réel qui nous a été révélé lors d'un entretien avec un frabicant d'outils d'autolocalisation. Celui-ci, contacté par une société de surveillance nocturne d'entrepôts, proposa de doter les véhicules de centrale GPS, afin de permetre au dispachting de localiser et de gérer toutes ses équipes. Il apparut rapidement que cet outil était largement voué à la surveillance des agents. Ceux-ci trouvèrent facilement la parade. En effet, en délaissant leur véhicule aux abords des entrepôts, ils pouvaient vaquer à d'autres occupations. Le fabricant proposa dès lors de coudre le récepteur GPS sur la tenue de travail des agents. Nous aurons aisément compris qu'il leur suffira de disposer leurs vêtements aux alentours des lieux à surveiller. Mais la parade ultime du fabricant et de la direction ne sera-t-elle pas de procéder à une implantation cutanée du récepteur? Réalité ou science- fiction, c'est à nous d'en décider!

Deuxièmement, depuis le début de notre étude nous avons parlé de véhicule intelligent. Il conviendrait cependant de s'interroger avec le plus grand sérieux sur la portée réelle que revêt un tel qualificatif. Qu'est-ce que l'intelligence ? Et à fortiori, qu'entendons-nous par « voiture intelligente »?

Si cela se limite aux applications actuelles d'aide à la navigation et d'aide au pilotage pour des manoeuvres simples et peu risquées, une défaillance du système ou un imprévu n'aura pas de conséquences désastreuses. Le terme « intelligence », constituant un abus de langage, ne devrait pas être remis en cause.

Par contre, est-il raisonnable de postuler un système intelligent qui se substituerait complètement au conducteur ? Nous savons qu'il est impossible de formaliser complètement tous les actes d'un conducteur et surtout de prendre en compte la complexité d'un environnement ouvert (le réseau routier). Tout au plus,

arriverons-nous à développer un bon modèle. Mais, comme par essence, un modèle est toujours imparfait, avons- nous le droit de réaliser un tel système ?

8.2.5 Informatique

Un pilote automatique s'inscrit dans le cadre des outils informatiques d'aide à la décision. Ce type d'outil étant le fruit de différentes branches de l'informatique telles que la gestion des bases de données, l'intelligence artificielle, la recherche opérationnelle ... , il s'agira donc de mettre en oeuvre de nombreuses synergies afin de réaliser pleinement notre pilote automatique.

Nous pourrions utiliser les enseignements de l'intelligence artificielle afin de doter notre véhicule de bases de données intelligentes utilisant des mécanismes d'auto-apprentissage. Ainsi, par exemple, le constructeur Porsche propose sur son modèle phare une boîte de vitesses électronique (Tiptronic) capable de s'ajuster aux habitudes du conducteur.

Notre étude se limitant à une réflexion sur le système devant supporter un pilote automatique de voitures, il faudra encore s'attacher au développement du logiciel de pilotage lui-même et des problèmes de méthodologie afférents. Au niveau, par exemple, de la spécification des besoins, en d'autres termes, l'établissement du cahier des charges, il est nécessaire de disposer d'une méthodologie et d'un outil de conception adéquats au contexte d'exécution de l'application envisagée. Des langages de spécifications tels Albert [DUB, 94], Statemate [HAR, 88] semblent répondre à ces exigences.

9. TABLE DES SIGLES

A

ALERT-C	Protocole utilisé par les système RDS (écrit en langage C).
ALI	Autofahrer Leitund Informations System.
AMTICS	Adveanced Mobile Traffic Information and Communication System.
ARCS	Automatic Route Control System.
ATLAS	Acquisition par Télé Diffusion de Logiciels Automobiles pour les Services.

C

CACS	Comprehensive Automobile Traffic Control System.
CARIN	CAR Information and Navigation.
CNRS	Centre national de recherche scientifique (france).

D

DRIVE (I & II)	Dedicated Road Safety Systems and Intelligent Vehicles in Europe. Lancé en 1989, le programme DRIVE à été reconduit par la direction générale des télématiques (DG13) en 1992 et prendra fin en mars 1995. La DG13 initie, coordonne et finance la recherche issu de consortiums, à concurrence de 50% pour le privé et 100 % pour les organismes publiques.
DOS	Distributed Operating System.

E

ERGS	Electronic Route Guidance System.

F

FHWA	Federal High Way Administration

G

GPS	Global Positionning System.

I

IRU	Internal Reference Unit.
IVHS	Intelligent Vehicule Higway System.

J

J.O.C.E.	Journal Officiel de la Communauté Européenne.

L

LAN	Local Area Network.

M

MINERVE	Média Intelligent pour l'Environnement Routier du Véhicule Européen.
MITI	Ministry of International Trade & Industry (Japon).

O

O.O.A.	Objet-Oriented Analysis.

P

PATHFINDER	Projet expérimental mené aux U.S.A. par le FHWA, Caltrans et G.M.
PROMETHEUS	Programme for a European Traffic with Highest Efficiency and Unprecendented Safety lunched in 1986. Ce programme est issue de la direction générale des transports de la Communauté européenne (DG7). D'une durée de 8 ans, il fut lancé en 1986 et regroupa essentiellement les constructeurs d'automobiles européens.
PMV	Panneaux à Messages Variables.

R

RBDS	Radio Broadcast Data System
RDS	Radio Data System.

S

S/A	Selective Availibility.
SCOPE	Projet Européen pour l'amélioration des réseaux de transport public. Lancé en Janvier 1992 avec un budget de 4.5 million d'ECU.
SOCRATES	System for Cellular Radio for Traffic Efficiency and Safety.
SSVS	Super Smart Vehicule System (Japon).

T

TMC	Traffic Message Chanel.
TNSS	Traffic Network Simulator System.

W

WAN	Wide Area Network

10. BIBLIOGRAPHIE

[BOU, 94] Boussuge J, Valade JM. «Projected Impact on Security of Driving-Aid Systems, Based on Known and Simplified Motorway Security Models » In Proceedings of the First World Congress on Applications of Transport Telematics and Intelligent Vehicle-Highway System (ed) Ertico, volume 4, pp. 1892-1899, 1994.

[BRAE, 80] Braegas P. « Function, Equipement, and Field Testing of a Route Guidance and Information System for Drivers (ALI)» In : Transaction on Vehicular Technology, IEEE, vol. VT-29, pp. 216-225, 1980.

[BRAE, 90] Braegas Peter, Ducheck R. « L'utilisation de la radio (RDS TMC) dans les systèmes d'aide à la circulation » In Electronique Automobile Juin-juillet 1990.

[BRO,89] R.A. Brooks, « Robust Layered Control System for a Mobile Robot ». Artificial Intelligence at MIT, Expanding Frontiers, vol. 2 (Chap. 24). Winston and Shellard Editor, 1989.

[CAT, 95] Catling Ian, Harris Richard : « The Socrates Project : An Overview Of Progress In The ATT Program » In Proceedings of the First World Congress on Applications of Transport Telematics and Intelligent Vehicule-Highway System (ed) Ertico, volume 5, 1995.

[CHA, 91] Charbonnier Christine « Le routage dynamique des véhicules » . Thèse de doctorat ENSAE, 1991.

[DEHU, 94] De Hestru M, Dupuis M. « Direct Impact Transport Modelling about Atmospheric Rejects and Noise in Commautte Urbaine de Lille » In Proceedings of the First World Congress on Applications of Transport Telematics and Intelligent Vehicule-Highway System (ed) Ertico, volume 2, pp. 782-783, 1994.

[DELO, 94] Delors J. « Livre Blanc »

[DUB, 94] Dubois Eric, Du Bois Philippe, Dubrun Frédéric, Petit Michaël : « The ALBERT Course Vol.1 : The Language ». University of Namur, 1994.

[EMM, 94] Emmerink Carla, Schulte Hans, « Route Planning & Route Guidance in thePhilips In-car Navigation System CARIN » In Proceedings of the First World Congress on Applications of Transport Telematics and Intelligent Vehicule-Highway System (ed) Ertico, volume 1, 1994.

[FACH, 85] Fache J. « Les carburants pour moteurs à allumage commandé : Perpespctives d'évolution » In L'Ingénieure de l'Automobile, pp. 43-44, 1985.

[FEB, 88] J. Ferber et M. Ghallab, « Problématique des univers multi-agents intelligents », Actes des journées internationales de l'IA. Toulouse 1988.

[Gil, 95] Gilchrist P., Günther B., Demery D.A. « General Packet Radio Service on GSM for ATT » In Proceedings of the First World Congress on Applications of Transport Telematics and Intelligent Vehicule-Highway System (ed) Ertico, volume 5, 1995.

[HAR, 88] Harel D., Lachover H., Naamad A., Pnueli A., Politi M., Sherman R., Shtul-Trauring A. : « STATEMATE : A Working Environment for the Development of Complex Reactive System ». IEEE, pp396-406, 1988.

[HEN, 83] Henry JJ, Farges JL, Tuffal J. « The PRODYN Real Time Traffic Algorithm » In Klamt, Lauber (eds) Fourth IFAC Conference on Control in Transportation, Baden-Baden, 1983.

[HOOF, 89] Hoofman G. LISB « An Individual Route Guidance and Information System in Berlin » In Perrin J-P. (ed) Control, Computers, Communications in Transportation, IFAC. Symposia series 1990, N 12, Pergamon Press, pp. 265-268, 1989.

[KIM, 94] H. Kimura, Y. Himono, Y. Matsuda, N. Hiwa « The Development of the Advanced Protocol for Automative for Automative Local Area Multiplexing Network (Advanced PALMNET), » SAE Paper, Paper. No. 940365, Feb. 1994.

[LIS, 90] Lister A.M. « Principes fondamentaux des systèmes d'exploitation ». Edition Eyrolles 1990.

[MEF, 93] Meftouh F., « Systèmes de commande temps-réel multi-agents ». Thèse de doctorat de l'ENSAE. 1993.

[MEN, 91] Mennox R, Miller J. « An Objet-oriented Environment for Robot System Architecture » in Control Systems, IEEE, volume 2, pp. 13-23, 1991.

[MILL, 91] Miller D.J. and Mennox R.C. « An Object Oriented Environment for Robot Systems Architectures », IEEE, Control System, Vol.11, Number 2, pp. 14-23. February 1991.

[MYL, 95] Mylopoulos John : « Conceptual Modelling for I.S. Engineering ». Chaire internationale. FUNDP Namur, 1995

[PAD, 90] G. Padiou et A. Sayah. « Techniques de synchronisation pour les applications parallèles ». Cepadues Edition, Toulouse 1990.

[PRE, 91] « Rapport sur la Recherche et le Developpment dans les Transports Terrestres au Japon » (ed) Seric, septembre 1991.

[RAM, 90] Ramaekers Jean « Systèmes d'exploitation matière approfondie ». Notes du cours de deuxième licence et maîtrise en Informatique (FUNDP) ,1990.

[RAN,94] Galijan Randal C., Gilkey James Y., Turner Richard P. « System Architecture and Test Results of a Carrier Phase GPS - Based Lateral & Longitudinal Automated Highway System- » In Proceedings of the First World Congress on Applications of Transport Telematics and Intelligent Vehicule-Highway System (ed) Ertico. Vol. 1 . pp127-134.,1994

[RTE, 94] « La Route TransEuropéenne ». Commission Européenne : Direction Générale des Transports, 1994.

[SAR, 90] Sarignac A. & Challe M.P. « Le programme CARMINAT ». In Electronique Automobile (juin-juillet 1990).

[SMI, 89] Smiley Alison « Mental Workload and Information Management » in IEEE . pp 435 - 438, 1989

[VON, 86] Von Tomkewitsch R. « An Universal Guidance and Information System for Road Traffic ». Conference publication N 260, The Institution of Electrical Engineers, pp. 22-29, 1986.

[WATS, 81] Watson R. W. « Distributed System Architecture Model » in Lampson, 1981.

[WHI, 91] « White Book for Variable Signs Application ». DRIVE VAMOS, 1991.

[YONR, 86] Yonezawa A. « Object-Oriented Concurrent Programming in ABCL/1 » In Proceedings of the first OOPSLA, pp. 258-269, 1986.

[ZIJ, 95] Zijderhand F., « Socrates : the open system and it's applications » In Proceedings of the First World Congress on Applications of Transport Telematics and Intelligent Vehicule-Highway System (ed) Ertico, volume 5, 1995.

11. TABLE DES MATIERES