Université de NANCY I — UER de Mathématiques

Centre de Recherche en Informatique de NANCY (CRIN)
BP 239 54506 VANDOEUVRE-LES-NANCY Cedex

CONCEPTION DES APPLICATIONS INFORMATIQUES REPARTIES EN COMMANDE DE PROCEDES INDUSTRIELS : UNE DEMARCHE, DES OUTILS.

présentée et soutenue publiquement le 5 Mai 1987
devant la commission d'Examen
A l'Université de NANCY I
pour l'obtention du grade de

DOCTEUR ES-SCIENCES MATHEMATIQUES
(Mention : Informatique)

par

JACQUES LONCHAMP
Maître de Conférences à l'Université de METZ

Rapporteurs : MM **Jacques MOSSIERE**, *Université de GRENOBLE I*
Michel RAYNAL, *Université de RENNES I*

Examinateurs : MM **Jean-Claude DERNIAME**, *Université de NANCY I*
Président

Marion CREHANGE, *Université de NANCY II*
Guy-René PERRIN, *Université de BESANCON*
Jean-Pierre THOMESSE, *INPL NANCY*
Robert VALETTE, *CNRS LAAS TOULOUSE*

Université de NANCY I — UER de Mathématiques

Centre de Recherche en Informatique de NANCY (CRIN)
BP 239 54506 VANDOEUVRE-LES-NANCY Cedex

CONCEPTION DES APPLICATIONS INFORMATIQUES REPARTIES EN COMMANDE DE PROCEDES INDUSTRIELS : UNE DEMARCHE, DES OUTILS.

THESE

présentée et soutenue publiquement le 5 Mai 1987
devant la commission d'Examen
A l'Université de NANCY I
pour l'obtention du grade de

DOCTEUR ES-SCIENCES MATHEMATIQUES
(Mention : Informatique)

par

JACQUES LONCHAMP
Maître de Conférences à l'Université de METZ

Rapporteurs : MM **Jacques MOSSIERE**, *Université de GRENOBLE I*
Michel RAYNAL, *Université de RENNES I*

Examinateurs : MM **Jean-Claude DERNIAME**, *Université de NANCY I*
Président

Marion CREHANGE, *Université de NANCY II*
Guy-René PERRIN, *Université de BESANCON*
Jean-Pierre THOMESSE, *INPL NANCY*
Robert VALETTE, *CNRS LAAS TOULOUSE*

Je tiens à remercier en premier lieu les membres du jury.

J'ai souhaité que Michel RAYNAL, éminent spécialiste des systèmes répartis, soit rapporteur de ce travail. Sa lecture critique de mes premières propositions et le temps qu'il a consacré à les commenter, ont permis de les améliorer considérablement.

Jacques MOSSIERE a accepté d'être rapporteur de cette thèse. La connaissance approfondie qu'il a du génie logiciel le désignait naturellement pour cette évaluation, dont je le remercie.

Jean-Claude DERNIAME me fait l'honneur de présider le jury. Je rends hommage tout d'abord à travers lui à tous ceux qui ont contribué à ma formation à l'informatique et à la recherche au sein de l'Université de NANCY I. Par ailleurs, mon travail doit beaucoup à l'équipe qu'il a su créer et animer avec Jean-Pierre THOMESSE et Guy-René PERRIN, autour du thème de la conception des applications réparties dans le cadre de l'ATP sur le parallélisme. Mes remerciements vont à tous.

J'ai souhaité la présence de Robert VALETTE dans ce jury. Ma participation au projet Automatique et Robotique Avancées, bénéficiant d'un contrat financé par le CNRS et les industriels membres du groupement, a été l'élément déterminant pour l'orientation de mes travaux. Il représente ici tous ceux que j'ai côtoyés dans ce cadre, car j'ai particulièrement apprécié les nombreux échanges de vues que nous avons eus.

Marion CREHANGE dirige mes travaux depuis de nombreuses années. Sans ses encouragements, la qualité de nos relations, son soutien constant et ses excellents conseils, ce travail n'aurait certainement pas abouti. Je l'en remercie très sincèrement.

Je tiens à associer à ces remerciements Martial LALLIER qui a réalisé une grande part des outils logiciels de ce travail ainsi que tous ceux qui ont contribué à la réalisation matérielle soignée de cet ouvrage.

Mes remerciements vont aussi à tous mes amis de l'IAE et de l'IUT de l'Université de METZ. La qualité de mon environnement quotidien a joué un rôle certain dans la réussite de mes travaux de recherche.

Enfin, comment ne pas joindre à ces remerciements mon épouse, ma fille, mes parents et mes proches; je leur dédie à tous ce travail.

RESUME :

La première partie de la thèse établit après analyse, une liste de caractéristiques significatives pour la classe de problèmes visée : applications de commande de procédés industriels et plus particulièrement, applications de commande de systèmes de production du type cellule ou atelier flexible. En fonction de leur adéquation à ces caractéristiques est retenu un ensemble de concepts et d'idées dans les domaines suivants : architectures logicielles, démarches de conception et techniques de spécification.

La seconde partie définit la démarche méthodique proposée, s'appuyant sur cet ensemble de concepts et d'idées. A chaque étape de la démarche sont décrits d'une part les outils (langages de spécification) et d'autre part leur mode d'emploi (méthodes de construction, validations, environnement de conception). Puis la faisabilité de la méthode est illustrée sur quelques exemples non triviaux.

La troisième partie vise à évaluer la contribution : sur la base d'un exemple commun pour les Réseaux de PETRI et par simple argumentation au regard d'autres méthodes et outils proches.

La proposition englobe un noyau immédiatement et concrètement utilisable (architecture logicielle -proche de celle de CONIC, ARGUS, etc. -, outil de spécification graphique des structures, outil de spécification programmatoire au niveau détaillé, éditeur / contrôleur interactif de ces

ABSTRACT :

The first section of the thesis analyses distributed process control software requirements; software for the control of production systems like flexible manufacturing systems or robotized assembly cells are especially studied. Our choices of a software architecture, a design method and specification tools are based on these problems requirements.

The second section defines the design method. At each step, the tools (specification languages) and their use (partitioning methods, validations, design environment) are described. The feasibility of the method is illustrated by several non trivial examples.

The third section evaluates the proposal : on the basis of a common example for Petri Nets and by argumentation for other methods and tools.

The thesis contains a kernel, immediatly and concretely usable (a software architecture -roughly similar to those proposed in CONIC, ARGUS, etc.-, a graphical language for structural specification, a pseudo language for detailed specification, an interactive editor and controller for these descriptions, the translation in ADA of the concepts, etc.) with more formal and original extensions (a formal behavioural specification language based on relationships among interface events-derived from CHEN and YEH's work-, related validations, a systematic partitioning method, prototyping tools, etc).

descriptions, traduction en ADA des concepts) autour duquel se greffent un ensemble de propositions d'outils intellectuels et logiciels de plus haut niveau (outil de spécification formelle des comportements en termes des relations entre événements à l'interface - dérivé des travaux de CHEN et YEH-, validations associées, méthode de partitionnement systématique, outil de prototypage, etc).

Nous prétendons qu'il s'agit d'une tentative réaliste de rapprochement entre les visions théoriques et pratiques de la conception.

MOTS CLES :

CONCEPTION ; APPLICATIONS REPARTIES ; COMMANDE DE PROCEDES INDUSTRIELS ; STRUCTURATION ; SPECIFICATION ; VALIDATION ; METHODE ; OUTILS

This proposal is claimed to be a realistic attempt to bridge the gulf between practical and theorical approaches of design.

INDEX TERMS :

DESIGN ; DISTRIBUTED SYSTEMS ; PROCESS CONTROL SYSTEMS ; STRUCTURATION ; SPECIFICATION ; VALIDATION ; METHOD ; TOOLS

TABLE DES MATIERES

TROISIEME PARTIE

INTRODUCTION

1. LE CONTEXTE DE L'ETUDE :

Cette étude a été conduite au cours des années 80 à 85 dans le cadre d'un contrat liant le Centre de Recherches en Informatique de NANCY (CRIN) et le Groupement d'Intérêt Scientifique "Automatique et Robotique Avancées" (ARA), ayant regroupé sous l'égide du CNRS une vingtaine de laboratoires, en majorité d'automatique, et des partenaires industriels (RENAULT, BULL, TELEMECANIQUE, ADEPA, etc.).

Orientée à l'origine vers la spécification du système d'information d'un atelier flexible, en vue de l'évaluation de sa logique et l'évaluation quantitative de son comportement [LON82a, LON82b] - dans le prolongement du projet MAESTRO [DUF80] -, son thème a été redéfini, après le désengagement des deux autres chercheurs initialement impliqués. Le sujet sur la conception des applications informatiques réparties en commande de procédés industriels a été retenu car il s'intégrait mieux aux préoccupations communes des équipes de recherche en génie logiciel du CRIN telles qu'elles ont pu être définies lors de synthèses, dans le contexte de l'ATP sur le parallélisme en particulier [CDF83].

Ce travail individuel a largement tiré bénéfice de cet environnement; il s'inscrit dans un continuum de travaux menés au CRIN, depuis une quinzaine d'années, en particulier dans les domaines de :

- la modularité [DER74, MIN79, ...],
- la communication [JUL81, PDT83, PER85, ...],
- la structuration des applications réparties [THO80, DER 83, ZAK84, ...].

Quelques communications [LON82b, LON83b] et rapports internes [LON82a, LON82c], ainsi que les contributions aux journées annuelles ARA [LON82d, LON83a], illustrent la démarche empirique suivie où la confrontation aux cas d'application proposés aux membres du pôle "Ateliers Flexibles" d'ARA a permis de dégager progressivement les idées directrices, de sélectionner et d'affiner les apports "théoriques" les mieux adaptés au domaine d'application considéré.

2. LE PLAN DE LA THESE :

Dans la première partie, nous dressons après analyse, une liste de caractéristiques significatives pour la classe de problèmes visée : applications de commande de procédés industriels et plus particulièrement, applications de commande de systèmes de production du type cellule ou atelier flexible.

Nous procédons ensuite à un choix, en fonction de leur adéquation à ces caractéristiques, d'un ensemble de concepts et d'idées dans les domaines suivants : modèles de

structuration, démarches de conception et techniques de spécification des applications réparties et temps réel.

La seconde partie définit la démarche méthodique retenue, s'appuyant sur cet ensemble de concepts et d'idées. A chaque étape de la démarche, nous décrivons d'une part les outils (langages de spécification) et d'autre part leur mode d'emploi (méthodes de construction, validations, environnement de conception). Puis nous illustrons la faisabilité de la méthode sur quelques exemples non triviaux.

La troisième partie vise à évaluer notre contribution : lorsque c'est possible par comparaison avec des propositions de même nature sur la base d'exemples communs ou, plus généralement, par argumentation au regard de travaux proches dans les domaines de l'automatique, des systèmes répartis et des systèmes temps-réel.

BIBLIOGRAPHIE :

[CDF83] CREHANGE, DERNIAME, FINANCE, JARAY, LONCHAMP, PERRIN, THOMESSE, ZAKARI
Du cahier des charges à une solution -entités communicantes, liaisons et communications-.
Rapport ATP Parallélisme,CRIN,1983

[DER74] DERNIAME J.C.
Le projet CIVA - un système de programmation modulaire.
Thèse d'Etat, NANCY I, 1/74

[DER83] DERNIAME, ZAKARI
Langage de conception pour les applications réparties en conduite de procédés.
Actes BIGRE 83, CAP D'AGDE, 10/83 (pp 363,376)

[DUF80] DUFOURD J.F.
Maquettes pour évaluer les systèmes d'information des organisations.
Thèse d'Etat, NANCY I, 1/80

[JUL81] JULLIAND J.
Expression des communications entre processus d'un programme parallèle par des types abstraits.
Doctorat de Spécialité, BESANCON, 5/81

[LON82a] J.LONCHAMP, J.F.DUFOURD, M.CREHANGE
Rapport d'activités 82 de l'équipe "ateliers flexibles".
Rapport 82-R-012, CRIN

[LON82b] J.LONCHAMP, J.F.DUFOURD, M.CREHANGE
Abstraction tools for the logical specification of an information system for a flexible workshop.
Proc. IFIP Work. Conf. APMS 82, BORDEAUX, 8/82, (pp 341,349)

[LON82c] J.LONCHAMP
Spécification et validation de l'architecture logique globale des applications temps-réel réparties.
Rapport 82-R-092, CRIN

[LON82d] J.LONCHAMP, J.F.DUFOURD, M.CREHANGE
Etude du système d'information d'un atelier flexible.
Actes Premières journées ARA, POITIERS, 9/82, (pp 121, 130)

[LON83a] J.LONCHAMP
Un outil d'aide à la conception des systèmes informatiques des ateliers flexibles.
Secondes journées ARA, BESANCON, 11/83

[LON83b] J.LONCHAMP
Structuration logique en termes d'agents communicants des applications réparties de commande/contrôle de processus.
Actes BIGRE 83, CAP D'AGDE, 10/83 (pp 255,273)

[MIN79] MINOT R.
ATM : un système de fabrication de programmes basé sur les concepts de modularité et de type abstrait.
Doctorat de Spécialité, NANCY I, 3/79

[PDT83] PERRIN, DERNIAME, THOMESSE
Communication based design of distributed systems.
Conv. Informatique Latine, BARCELONE, 6/83

[PER85] PERRIN G.R.
La communication : un outil pour la spécification, la construction et la vérification de systèmes parallèles.
Doctorat d'Etat, NANCY I, 10/85

[THO80] THOMESSE J.P.
SYGARE : une structuration pour la conception d'application en temps réel et réparties.
Doctorat d'Etat, NANCY I, 1980

[ZAK84] ZAKARI A.
FLEXI : langage de conception d'applications de conduite de procédés industriels.
Doctorat 3ème cycle, NANCY I, 3/84

PREMIERE PARTIE

CHAPITRE 1 :

ANALYSE DE LA CLASSE DE PROBLEMES

RESUME :

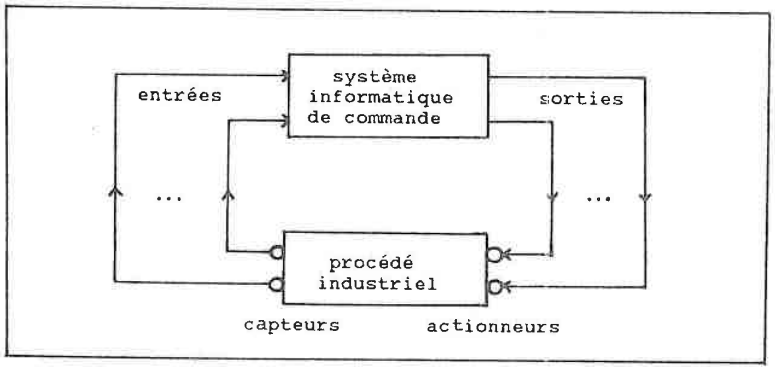
Ce premier chapitre présente la classe de problèmes considérée et diverses approches de description.

Il propose, après analyse, une liste de dix caractéristiques significatives, sur lesquelles se fonderont nos choix ultérieurs.

1. DEFINITION GENERALE :

Notre étude concerne les applications de commande de procédés industriels et plus particulièrement de commande de systèmes de production, du type cellule ou atelier flexible. Ces systèmes flexibles de production se caractérisent par la complexité de leur pilotage, liée à l'automatisation complète de la manutention et du transport des pièces (et parfois des outils). Ils ont la faculté de fabriquer simultanément différents types de produits grâce à la souplesse des systèmes de transport et à l'adaptativité des postes de travail automatisés [HUT77]. Ils s'appliquent aussi bien à l'usinage qu'à l'assemblage de produits en petite et moyenne série (la grande série relevant des chaînes de transfert spécialisées).

De manière générale, un système informatique de commande de procédé industriel est intimement lié au procédé qu'il pilote : par le biais de capteurs, il suit l'évolution du procédé et, en particulier, l'évolution de grandeurs caractéristiques de son fonctionnement (ex : température, position, volume, etc.). A partir de ces entrées, il élabore en sortie des commandes vers des actionneurs agissant sur le procédé, en vue de contraindre les grandeurs caractéristiques à rester dans un domaine de variation donné, ce qui équivaut à contraindre le procédé à respecter certaines propriétés ou un comportement donné. L'élaboration des sorties à partir des entrées doit satisfaire des contraintes : temporelles du type temps de réponse, liées à la sûreté de fonctionnement, etc.



système industriel informatisé

Fig. 1.1.

2. DIVERSES FORMES DE DESCRIPTION :

D'un point de vue théorique, on peut modéliser le procédé comme un ensemble d'objets, chaque objet pouvant prendre un ensemble d'états, et un ensemble de transformations liées aux lois naturelles (physico-chimiques) d'évolution du procédé et qui conduisent à des changements d'état des objets. Si l'on veut modéliser le système industriel complet (procédé et système informatique de commande), on doit ajouter les opérations commandées par le système informatique, elles aussi génératrices de changements d'états des objets. La spécification d'un système industriel et des propriétés qu'il doit respecter peut alors inclure l'énumération [LEL83] :

- des objets,
- des états des objets,
- des opérations (transformations naturelles ou commandées),
- des contraintes sur les objets, états et opérations (ex: contraintes temporelles sur les durées autorisées de certains états ou de certaines opérations, relations temporelles entre opérations, etc.),
- des probabilités acceptables d'occurrences d'erreurs sur les états et opérations,
- des opérations de "récupération" associées aux erreurs.

On imagine bien, combien la détermination exhaustive des objets, des états et surtout des propriétés peut être ardue pour un système réel. Les tentatives de description qui vont dans ce sens restent souvent au niveau de spécifications très informelles (ex : projet MOST [LIA81]).

Les automaticiens utilisent classiquement une forme de description centrée sur les variables caractéristiques du procédé, les actions à déclencher sur le procédé et les lois de commande qui les mettent en relation (ex : [LAD82], spécifications fonctionnelles par GRAFCET [BLA79]). C'est une description très "procédurale" (dynamique) : la frontière entre l'expression de la loi de commande avec ses variables et l'expression de sa représentation informatique (variables informatiques, algorithmes, etc.) n'est pas toujours claire.

Nous préférons une description à l'interface du système de commande et du procédé : l'application est définie globalement par **la relation qu'elle maintient entre ses entrées et ses sorties en vue de contraindre le procédé commandé à conserver un comportement donné**. Il s'agit à la fois d'une "vue externe", satisfaisante pour les utilisateurs, et d'un point de départ idéal pour le concepteur de l'application, spécialement dans l'optique d'utilisation d'un modèle en réseau d'entités communicantes, comme nous le montrerons dans la suite.

Sont décrits :

- les entrées du système de commande,
- ses sorties,
- les relations entre entrées et sorties, sous une forme que nous voulons **statique et abstraite**,

- les contraintes sur les entrées, les sorties et les relations, imposées par le procédé et l'environnement en général.

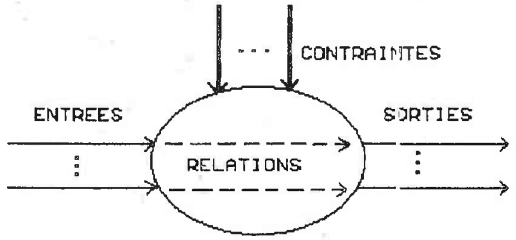


Fig.1.2.

3. ELEMENTS D'UNE DESCRIPTION A L'INTERFACE DU SYSTEME DE COMMANDE ET DU PROCEDE :

a) Les entrées :

Parmi les types d'entrées les plus courantes on peut citer : les mesures de grandeurs caractéristiques (périodiques ou à la demande), les notifications de changement d'état d'objets physiques et autres événements survenant aléatoirement, les requêtes ou les consignes en provenance de l'environnement humain (réglages, interrogations,...).

b) Les sorties :

Parmi les types de sorties les plus courantes on peut citer : les commandes aux actionneurs, les alarmes, les réponses et accusés de réception aux requêtes et consignes des opérateurs, l'alimentation des bases de données destinées à la gestion de production.

c) Les relations :

Le nombre des relations et la complexité des coordinations qu'elles impliquent, augmentent très vite avec la taille des systèmes. Par ailleurs beaucoup de tâches annexes à la régulation proprement dite, sont dévolues aux systèmes informatiques : acquisition et stockage de données, mise en forme de résultats (pour les opérateurs, les gestionnaires), optimisation, etc.

Pour répondre à cette complexité, on rencontre fréquemment des descriptions organisées comme un ensemble hiérarchisé de sous-systèmes logiques communicants :

- sous-systèmes de contrôle local (boucles de régulation simples), au niveau bas,

- sous-systèmes de coordination, au niveau intermédiaire (ex : contrôle d'une machine complexe),
- sous-systèmes de supervision de "secteur" (atelier, département, usine), au niveau supérieur, chargés de l'ordonnancement, l'optimisation, la mise en forme des données pour le management, etc.

On peut noter avec [VAL81] qu'une telle hiérarchisation est prise en défaut dans le cas des ateliers flexibles où la commande du système de transport dépend aussi bien du niveau bas (ex : arrivée d'un chariot à un plot de détection) que du niveau supérieur (ex : choix d'ordonnancement de la production).

d) Les contraintes générales :

Tout système de conduite de procédé en temps-réel est soumis par nature à un ensemble de contraintes générales [LEL83] :

- d'exactitude : contrairement aux applications de gestion, pour lesquelles les sorties erronées peuvent être "récupérées" par annulation ou compensation, les applications de commande en temps réel sont tenues à l'exactitude, car les récupérations transparentes d'erreurs ne sont pas en général possibles;
- de promptitude : pour certaines tâches critiques la rapidité d'obtention d'une sortie peut conditionner sa validité;
- de disponibilité : les taux requis s'approchent souvent des 100%; ces taux sont envisageables grâce aux techniques de prévention et de tolérance aux fautes;
- de flexibilité : une bonne flexibilité fonctionnelle est assurée si l'ajout, la modification ou la suppression de fonctionnalités sont aisés; la flexibilité opérationnelle recouvre les adaptations dynamiques du système en cours de fonctionnement.

L'acuité des contraintes varie beaucoup et, même dans le domaine des procédés industriels, les fourchettes de variation sont assez larges [PRI81] :

- temps de réponse aux alarmes	sidérurgie	: 0,01s
	agro-alimentaire	: 1s
- degré de disponibilité exigé	chimie	: 99,5%
	papier-carton	: 98,8%
- nombre de points d'E/S	raffinage	: 1200/1960
	papier-carton	: 35/90

4. CARACTERISTIQUES SIGNIFICATIVES :

L'analyse des applications de commande de systèmes de production nous permet de dégager un certain nombre de caractéristiques en rapport avec la structuration de ces applications et avec le choix de la méthode de conception.

a) Caractéristiques en rapport avec la structuration des applications :

(1) l'environnement de ces applications est fortement parallèle : le procédé comporte de nombreuses entités et activités évoluant en parallèle.

(2) la structure de cet environnement est stable : les entités et activités du procédé sont en nombre sensiblement constant.

(3) l'activité est continue.

(4) la répartition géographique peut être importante.

(5) les exigences de promptitude sont le plus souvent peu élevées, mais les exceptions à cette règle, comme les alarmes, doivent être absolument prises en considération.

(6) les volumes d'informations nécessaires pour la coordination des sous-systèmes logiques sont le plus souvent faibles.

(7) la multiplicité des composantes de nature diverses du système industriel, composantes mécaniques, informatiques, humaines, etc., rend indispensable l'étude de plusieurs modes de marche : mode de fonctionnement normal, modes de fonctionnements dégradés.

b) caractéristiques en rapport avec le choix de la méthode de conception :

(8) le concepteur doit tenir compte de multiples contraintes de réalisation économiques, techniques (ex : contraintes environnementales), informatiques (ex : matériels ou logiciels préexistants); peu de systèmes sont construits "ex nihilo".

(9) certains systèmes, tels les ateliers flexibles, présentent une grande complexité, y compris algorithmique, qui met en défaut les outils de description et de réalisation classiques du domaine.

(10) les "utilisateurs" (ingénieurs spécialistes du procédé ou de l'instrumentation, opérateurs de conduite, automaticiens) ont des cultures techniques propres (non informatiques).

Ces caractéristiques guideront notre choix d'un modèle de structuration, d'une démarche de conception et d'une technique de spécification. Notons que plusieurs d'entre elles, comme le haut degré de parallélisme de l'environnement (1), la répartition géographique (4) ou la localité de nombreux traitements (6), permettent de justifier le recours aux architectures réparties.

BIBLIOGRAPHIE :

[BLA79] BLANCHARD M.
Comprendre, maîtriser et appliquer le GRAFCET
CEPADUES Ed., TOULOUSE, 1979

[HUT77] HUTCHINSON G.K.
The control of flexible manufacturing system : required information and algorithm structures.
IFAC Symp. on Information-Control problems in Manufacturing technology, TOKYO, 10/77

[LAD82] LADET P.
Contribution à l'étude des systèmes informatiques répartis pour la commande de procédés industriels.
Thèse d'Etat, GRENOBLE, 1982

[LEL83] LE LANN G.
Sur le traitement réparti temps-réel.
Actes IFIP 83, PARIS, (pp 213,225)

[LIA81] LIAIS E.
Méthode et outils de spécification temps-réel.
Thèse Doct. Ing., TOULOUSE UPS, 7/81

[PRI81] PRINCE, SLOMAN
Communication requirements of distributed computer control system.
IEE Proc., E-128, 1/81 (pp 21,34)

[VAL81] VALETTE R.
Spécification et validation de la synchronisation des tâches dans un atelier flexible de masticage.
Rapport LAAS 81.I.38, 10/81

CHAPITRE 2 :

CHOIX ESSENTIELS

RESUME :

Ce deuxième chapitre explicite les choix essentiels réalisés dans trois domaines clés :
- la structuration des applications,
- la démarche méthodique,
- les outils de spécification.

Ces choix s'appuient sur l'analyse de l'état de l'art pour ces questions et sur les caractéristiques significatives de la classe de problèmes, établies au premier chapitre. Nous considérons en effet qu'il doit y avoir primauté de la classe de problèmes et de l'étude de ses caractéristiques sur le choix d'une méthode et primauté du choix d'une méthode sur le choix des outils de spécification, qui doivent s'accorder à eux.

1. CHOIX DU MODELE DE STRUCTURATION DES APPLICATIONS :

1.1. GENERALITES SUR LA "PROGRAMMATION REPARTIE" :

La caractéristique fondamentale de ce que l'on peut appeler la "programmation répartie" ou les "langages de programmation d'architectures réparties" [COR81] est de tenir compte, dans une certaine mesure, des particularités de l'architecture support pour obtenir une programmation efficace tout en restant de "haut niveau". L'absence de variables partagées entre les entités qui structurent les applications est le critère de base qui nous autorise à classer dans cette famille des langages comme CSP [HOA78], DP [BRI78], CONIC [KRA83], ARGUS [LIS79], NIL [PST83], etc.

Leurs concepts peuvent être analysés dans la perspective historique de l'évolution des langages dans les domaines de la modularité, du parallélisme, de la communication et la synchronisation. On trouvera des études conséquentes de ces évolutions dans de nombreux ouvrages, comme par exemple [HOM84]. Nous nous contentons ici d'évoquer brièvement les principales variantes au sein de cette famille de langages, pour ce qui concerne le choix des entités structurantes et la nature des interactions entre ces entités, et de justifier nos choix en fonction des propriétés des applications visées énoncées au § 4 du chapitre 1.

1.2. LES ENTITES :

L'examen des propositions en programmation répartie nous permet de distinguer deux types de structurations en entités :

- le premier à base de processus séquentiels (ex : CSP , DP, PLITS [FEL79]),

- le second à base d'entités regroupant un ensemble de processus résidant sur un même site où ils peuvent se partager des ressources communes; nous les appellerons "modules de KRAMER" ou "modules" en l'absence d'ambiguïté (ex : CONIC, ARGUS, MARS [KOP82], [DER83], NIL).

Le premier type ne permet pas d'exprimer explicitement dans le programme des **contraintes de localisation**. Par contrainte de localisation nous entendons la nécessité de résider ou de ne pas résider sur un même site, un site étant constitué d'un ou plusieurs processeurs se partageant une mémoire commune. Pour les applications de commande de procédés cette faculté est essentielle en raison des différences importantes d'efficacité et de sûreté entre des traitements purement locaux à un site et des traitements répartis sur plusieurs sites.

Le second type de structuration, que nous retenons, permet au contraire de rendre compte des **partages de ressources**, de l'existence de **couplages** temporels ou informationnels exceptionnellement élevés - cf. caractéristiques (5), (6) du chap. 1 § 4 - et de certaines **contraintes d'implantation** - cf. (4), (8) -.

L'indépendance vis à vis de la répartition physique, essentielle pour la flexibilité de la solution, demeure, dans la mesure où les sites physiques sur lesquels seront implantés les modules ne sont pas fixés; en effet, si un module ne peut être réparti, il est possible d' installer plusieurs modules sur un même site physique : les modules apparaissent comme des **"sites virtuels"**. LISKOV résume parfaitement cette philosophie pour ses "gardiens", similaires aux modules de KRAMER : « a programmer can conceive of a distributed program as a set of abstract nodes, each of which perform a meaningful task for its application. Intraguardian activity is local and inexpensive (since it all takes place at a single physical node); interguardian processing is likely to be more costly, but the possibility of this added expense is evident in the program structure ».

D'autres points de discussion sont ouverts quant à l'interêt de prévoir : une gestion dynamique des entités (créations, suppressions), une migration dynamique des entités (changements automatiques de localisation à l'exécution), une distinction entre entités actives et entités passives. Nous répondons négativement à ces interrogations. Pour les deux premiers points en raison du caractère essentiellement statique des applications -cf. propriété (2)-. De plus, l'existence d'entrées et sorties externes attache de nombreuses entités à des capteurs et actionneurs reliés le plus souvent à un seul site physique, ce qui rend difficiles les migrations. Notons que la migration dynamique sous le contrôle du système d'exploitation à des fins d'optimisation dont il est question ici n'a rien à voir avec la migration sous le contrôle de l'utilisateur à des fins de reconfiguration, prévue dans CONIC. Cette forme achevée mais coûteuse de flexibilité opérationnelle [MAG83] est intéressante dans certains cas particuliers; dans d'autres, l'exploitation d'une bibliothèque de versions préétablies selon les différents modes de marche -cf. (7)-, doit suffire.

1.3. LES INTERACTIONS ENTRE ENTITES :

L'unique moyen d'interaction entre modules réside dans l'échange de messages qui symbolise le recours au système de communication du système réparti. Nous préconisons de retenir une riche palette de types de communication asynchrones et synchrones correspondant aux "besoins naturels" de communication répertoriés pour les applications de commande de procédés. Ce point sera détaillé au chapitre 4.

Le concept de port [BAL71] permet de définir proprement l'interface visible des modules : les processus et les

objets internes sont cachés au profit des seuls ports par l'intermédiaire desquels des communications peuvent s'établir. Au niveau du port certaines propriétés comme le sens de l'échange et le type de message admissible sont précisés. De l'intérieur également, seuls les ports locaux sont visibles : ils abstraient l'environnement du module. L'utilisation de désignations indirectes au niveau des actions de communication, grâce aux ports, et la séparation de l'expression des chemins de communication de l'expression de leur utilisation, sont des gages de modularité et de flexibilité.

La notion d' "autonomie des modules" [LIS79] est également importante pour les applications de commande de procédés : c'est le module qui décide localement des communications à prendre en compte et des délais d'attente maxima au delà desquels les actions de communication bloquantes seront supposées avoir échoué. Ceci permet de prendre en compte les défaillances du système de communication et des autres modules et contribue donc à la sûreté et la flexibilité opérationnelle des applications.

Pour l'essentiel, notre choix de modèle de structuration correspond à celui de CONIC; les principaux concepts sont repris sur la figure 2.1. Nous préciserons dans la suite les quelques différences de détail concernant les liaisons, les actions de communication, les priorités, etc.

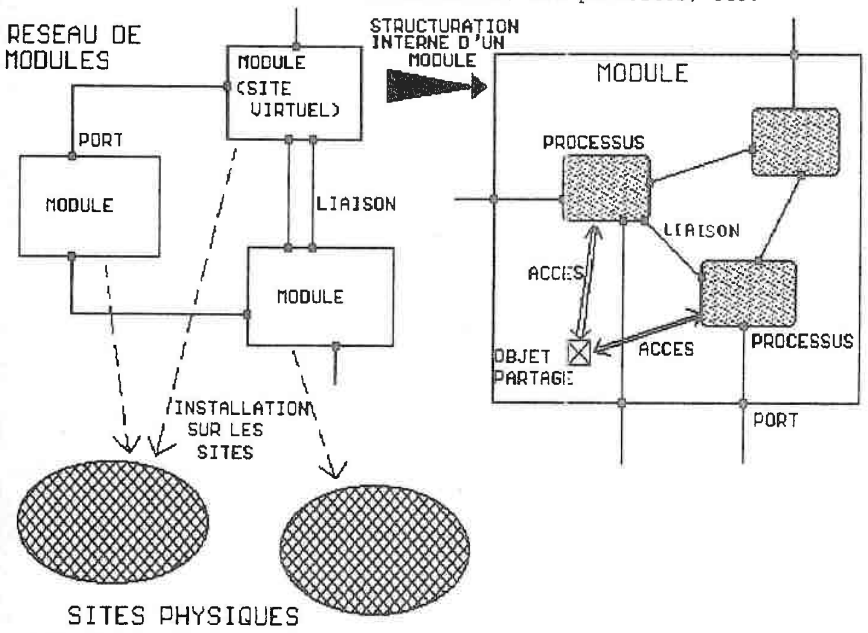


Fig. 2.1.

2. IDEES DIRECTRICES POUR L'ETABLISSEMENT D'UNE DEMARCHE METHODIQUE DE CONCEPTION :

2.1. DELIMITATION DE LA PHASE DE CONCEPTION :

Le découpage classique du cycle de vie des applications retient comme trois premières phases :

- la phase de définition : elle a pour but de spécifier "ce que fait le système", en décrivant d'une manière ou d'une autre une liste de propriétés requises ou "requirements" (*).
- la phase de conception : elle a pour but de définir "comment le système fait ce qu'il doit faire", c'est à dire la solution adoptée ("design").
- la phase de réalisation : elle comporte l'écriture dans le langage de programmation choisi, la mise au point et l'installation des logiciels sur la configuration matérielle cible.

Le caractère imprécis des frontières entre ces phases a souvent été souligné. L'élaboration des propriétés requises peut nécessiter une décomposition du problème qu'il est bien difficile de distinguer d'un "début de conception" (cf. la notion de "theoretical implementation" de [PAR77]). La frontière entre conception et réalisation dépend du niveau d'abstraction retenu pour la spécification de la solution, autrement dit des degrés de liberté laissés aux programmeurs d'application.

Nous partirons de l'établissement d'une spécification partiellement formalisée des propriétés requises pour aboutir à une spécification détaillée de la solution retenue, ne laissant aux programmeurs que des choix d'implantation **sur un même site** d'algorithmes ou de structures de données abstraites.

Pour ce qui concerne la détermination de la configuration matérielle cible, peu de travaux systématisent la réflexion. C'est le cas dans [CAL82], pour des architectures multi-microordinateurs bâties à partir de cartes standards (cartes UC, cartes mémoires, bus, contrôleurs E/S, etc.). Les critères d'acceptation d'une architecture tiennent dans un taux d'activité acceptable pour les processeurs (< 100%), et le respect des contraintes temporelles. Pour les établir il faut connaître les fréquences maxima de déclenchement de tous les traitements et leurs durées maxima d'exécution : on propose soit de les estimer en fonction des algorithmes et de la connaissance des cartes constituantes, soit de procéder à

(*) Nous n'utilisons pas le mot "spécification" dans son acception restrictive qui concerne précisément l'expression formalisée des propriétés requises. Nous l'utilisons pour désigner toute description, même informelle, présentant un certain degré d'abstraction par rapport au code exécutable.

des tests sur un processeur de référence et d'estimer les temps sur les processeurs cibles, grâce à des coefficients de puissance établis par "benchmarks". Les choix d'architecture sont guidés par des critères de construction comme :

- la modularité : assemblage de cartes standards et de modules logiciels standards ou préexistants,
- la minimisation des coûts : minimisation du nombre des composants,
- la sûreté de fonctionnement : introduction de redondances matérielles et fonctionnelles pour atteindre les niveaux de disponibilité souhaités, etc.

Dans notre contexte plus général, une systématisation de cette nature apparaît peu réaliste :

- les architectures sont beaucoup plus complexes et hétérogènes (micros, minis, ordinateurs universels, automates programmables, processeurs spécialisés); les estimations de temps d'éxécution en sont d'autant compliquées;
- les critères de construction sont souvent multiformes (ex : certains processeurs sont intégrés à des matériels imposés ou à des applications préexistantes, d'autres sont spécifiques et à déterminer -cf. propriété (8)-);
- les critères économiques ne peuvent se réduire "à la minimisation du coût des processeurs par le taux d'activité maximum";
- toute systématisation risque de surévaluer les critères facilement quantifiables, alors que les architectures réparties visent aussi à répondre à des besoins qualitatifs (flexibilité, simplicité,...).

Nous nous restreignons donc dans ce travail à la conception d'une architecture logicielle compatible avec une architecture matérielle, soit imposée soit définie en parallèle par une réflexion "ad hoc" de nature économique et technique. Seules apparaîtront dans nos descriptions finales, les identifications des sites physiques et les assignations des modules à ces sites.

2.2. LE CHOIX DES ETAPES :

La démarche retenue comporte deux étapes principales que constituent **la structuration de l'application en un réseau de modules et la conception détaillée des modules**, sur le modèle général du "programming-in-the-large/programming-in-the-small" [DRE75]. La première étape oblige le concepteur à exprimer et donc à maîtriser l'organisation générale de toute son application avant de commencer à en spécifier les détails. Ceci est particulièrement important en raison de la complexité de certaines applications -cf.(9)-.

L'étape de structuration en un réseau de modules peut se décomposer en deux stades :

- **"structuration logique" en un réseau d'agents**, qui se fait de manière totalement indépendante de l' implanta-

tion. Nous utilisons le vocable d'"agent" pour parler de sous-systèmes logiques de structure quelconque.

-**"structuration organique" en un réseau de modules** (sites virtuels), qui permet d'exprimer les choix essentiels d'organisation de la solution.

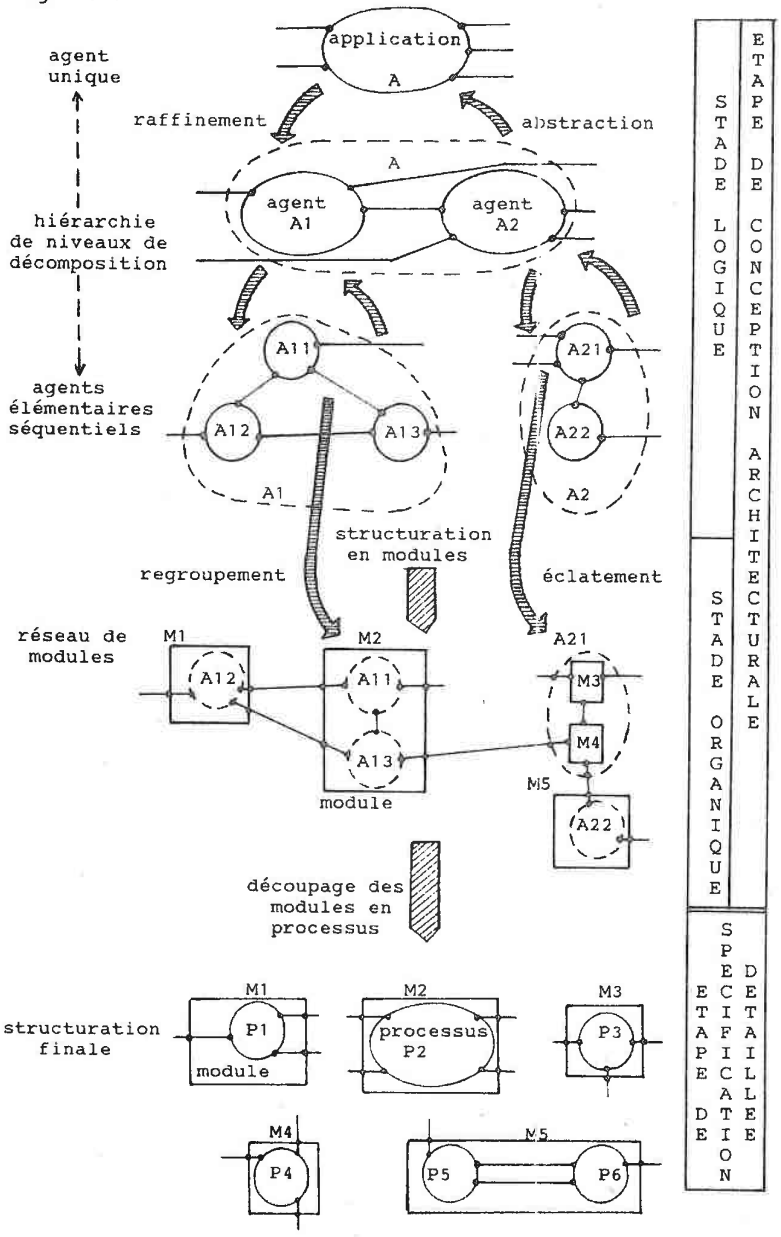
Le passage par la structure logique peut se justifier d'au moins deux façons :
- une solution "conceptuelle" facilite le dialogue avec les "utilisateurs" non informaticiens -cf.propriété (10)-
- il s'agit parfois du seul moyen réaliste d'exprimer les propriétés requises de systèmes dont la complexité -cf. (9)- empêche que ce puisse être fait au niveau global d'une "boîte noire" unique (agent unique).

La scission en deux stades reflète également la bivalence inhérente à la notion de module de KRAMER :
- unité de découpage logique : « a module performs some local monitoring and control function (e.g. a pump controller) »,
- unité de répartition et de configuration : « a module is the smallest software component which can be distributed or replaced » [KRA83].

Nous proposons de **pousser la décomposition logique jusqu'à l'obtention d'un réseau d'agents élémentaires séquentiels**. Ceci doit nous permettre de mettre en évidence clairement (c'est à dire en termes des seules communications par la quantification des flux de messages), les couplages entre agents élémentaires afin d'étayer la réflexion sur leur regroupements en modules faiblement couplés entre eux et à forte cohérence interne -cf.(5),(6)-. La grande majorité des modules correspondront en effet à des regroupements d'agents élémentaires "voisins" dans le réseau même si, dans quelques cas particuliers (par exemple à cause de contraintes d'implantation -cf.(8)-), un agent élémentaire peut être scindé en plusieurs modules.

Le passage d'un agent global modélisant l'ensemble de l'application au réseau des agents élémentaires peut se conduire de manière descendante (par raffinements successifs), ascendante (abstractions successives) ou mixte. Nous proposons une méthode, pour systématiser ce passage, fondée sur la prise en compte du parallélisme de l'environnement, c'est à dire du parallélisme entre entités du monde réel -cf.(1)-. Nous postulons en effet qu'une bonne structuration d'une application de commande de procédés doit refléter pour l'essentiel ce type de parallélisme, pour des raisons de lisibilité et de facilité d'évolution; on est proche de la notion de structuration sous-tendue par une modélisation de la réalité environnante, introduite par JACKSON [JAC78] et des techniques de développement "orientées objets" [BOO86]. Nous montrons dans ce qui suit que la systématisation du découpage peut être poussée très loin dans cette optique. La figure 2.2. illustre cette succession d'étapes.

Fig. 2.2.



Enfin, nous mettons l'accent sur l'importance des spécifications graphiques des structures; beaucoup de choix de détail du modèle sont dictés par un souci de non ambiguïté au niveau de la représentation graphique : "ce qui se conçoit bien ... se dessine clairement", pensons nous. Nous le justifions également par la culture spécifique des utilisateurs concernés -cf.(10)-.

3. IDEES DIRECTRICES POUR LE CHOIX D'UN OUTIL DE SPECIFICATION:

Comme la structure de ce chapitre le montre, il y a pour nous **primauté de la classe de problèmes et de l'étude de ses caractéristiques sur le choix d'une méthode et primauté du choix d'une méthode sur le choix des outils de spécification**, qui doivent s'accorder à eux. Nous examinons donc les caractéristiques générales requises pour un bon outil de spécification, puis les caractéristiques requises dérivant du domaine d'application et enfin les caractéristiques requises dérivant de la démarche de conception.

3.1. CARACTERISTIQUES GENERALES REQUISES :

Elles ont souvent été analysées; nous reprenons ici l'essentiel de celles indiquées par le ADA-Europe Working Group on Formal Methods for Specifications and Development [BAR84]. Une spécification doit être :

- formelle (pour être précise et permettre des validations rigoureuses),
- compréhensible (enseignable à des non mathématiciens),
- abstraite (pour éviter les sur-spécifications et les biais à l'implantation),
- expressive (pour permettre la description de propriétés variées : de vivacité, de sûreté, temps-réel, etc.),
- utilisable pour le "programming in the large" (et pas seulement au niveau du programme final),
- traitable (pour réaliser effectivement des validations et dériver des implantations),
- testable (car les validations ont des limites).

3.2. CARACTERISTIQUES REQUISES EN FONCTION DU DOMAINE D' APPLICATION :

Nous souhaitons que la forme de spécification retenue s'accorde au mieux avec la vision que peut avoir un utilisateur de son application.

En nous fondant sur notre définition générale des applications de commande de procédé (**l'application est définie par la relation qu'elle maintient entre ses entrées et ses sorties en vue de contraindre le procédé qu'elle commande à**

conserver un comportement donné), nous donnons la préférence à une spécification en termes d'**événements à l'interface** (entrées et sorties sur les ports externes) et de **relations entre ces événements**. La description de ces relations devra bien entendu respecter les propriétés de formalité, d'abstraction et d'expressivité évoquées au paragraphe précédent.

Plus précisément, la description des conséquences de certains événements externes ou de certaines mesures de valeurs en fonction de l'état actuel du système, conséquences correspondant pour l'essentiel à des déclenchements d'actions, nous semble à la base de toute spécification des systèmes de commande. Il nous apparaît essentiel de pouvoir exprimer ces propriétés de la manière la plus directe et la plus naturelle possible : donc avec la notion de **déclenchement** (ou précédence causale) entre événements à l'interface et la notion d'état du système; cet état résulte de tous les événements à l'interface qui ont **précédé** l'instant considéré. Précédence causale, précédence temporelle et sa traduction en forme de valeurs de variables d'état (qui mémorisent des histoires d'événements à l'interface), constitueront donc les ingrédients de base de notre outil de spécification.

3.3. CARACTERISTIQUES REQUISES EN FONCTION DE LA DEMARCHE DE CONCEPTION :

Nous devons pouvoir établir des spécifications à différents niveaux :

- agent global unique, correspondant à l'application toute entière,
- réseau d'agents de la structure logique,
- réseau de modules de la structure organique.

Chaque entité, quelle que soit sa nature, sera donc caractérisée par :

- son **interface**, en termes de ports et de types de messages admissibles,
- la **description formelle de son comportement**, en termes de relations entre événements sur les ports,
- la **spécification de son implantation éventuelle en un réseau d'entités communicantes**; ceci établit le lien entre deux niveaux de description [BOC83].

Un point fondamental sera la vérification de la cohérence entre la spécification du comportement de l'entité globale et le comportement de son implantation.

BIBLIOGRAPHIE :

[BAL71] BALZER R.M.
PORTS : a method for dynamic interprograms communication and job control.
AFIPS Conf., Vol 38, 1971, (pp 485,489)

[BAR84] BARRINGER H.
Formal specification techniques for parallel and distributed systems. A short review.
Proc. ADATEC/ADA Europe Joint Conf., BRUXELLES, 11/84 (pp 281,294)

[BOC83] BOCHMANN, RAYNAL
Structured specification of communicating systems.
IEEE Trans.on Computers, Vol C-32, no 2, 2/83 (pp 120,133)

[BOO86] BOOCH G.
Object-oriented development.
IEEE Trans. on Soft. Eng., Vol SE-12, no 2, 2/86 (pp 211, 221)

[BRI78] B. HANSEN
Distributed processes : a concurrent programming concept.
CACM, Vol 21, 11, 11/78 (pp 934,941)

[CAL82] CALVEZ J.P.
Une méthodologie de conception des systèmes multi-micro-ordinateurs pour les applications de commande en temps réel.
Thèse d'Etat, NANTES, 11/82

[COR81] CORNAFION
Systèmes informatiques répartis.
DUNOD, 1981

[DER83] DERNIAME, ZAKARI
Langage de conception pour les applications réparties de conduite de procédés.
BIGRE 83, CAP D'AGDE, 10/83 (pp 363,376)

[DRE76] DE REMER, KRON
Programming-in-the-large versus programming-in-the-small.
IEEE Trans. on Soft. Eng., Vol SE-2, no 2, 6/76 (pp 80,86)

[FEL79] FELDMAN J.A.
High level programming for distributed computing.
CACM, Vol 22, no 6, 6/79 (pp 353,368)

[HOA78] HOARE C.A.R.
Communicating sequential processes.
CACM, VOL 21, no 8, 8/78 (pp 666,677)

[HOM84] HOMMEL G.
Language constructs for distributed programs.
LNCS no 190, Springer-Verlag, 1984 (pp 287,342)

[JAC78] JACKSON M.A.
Information systems : modeling, sequencing and transformations.
Proc.3th Conf.on Soft.Eng.,1978

[KOP82] KOPETZ, LOHNERT, MERKER, PAUTHNER
High level programming of distributed process control systems.
Real-time-data'82, VERSAILLES, 11/82 (pp 33,40)

[KRA83] KRAMER, MAGEE, SLOMAN, LISTER
CONIC : an integrated approach to distributed computer control systems.
IEE Proc.,Vol 130,1,1983

[LIS79] LISKOV B.
Primitives for distributed computing.
Proc. 7th Symp. operating systems principles, 12/79 (pp 33,42)

[MAG83] MAGEE, KRAMER
Dynamic system reconfiguration for distributed real-time systems.
RR DOC 83/10, Imp. College, LONDON, 3/83

[PAR77] PARNAS D.L.
The use of precise specifications in the development of software.
Information processing 77, North Holland Pub., 1977 (pp 861, 867)

[PST83] PARR, STROM
NIL : a high-level language for distributed system programming.
IBM Syst.Journ.,Vol 22,1/2,1983

DEUXIEME PARTIE

CHAPITRE 3 :

CONCEPTION DE L'ARCHITECTURE GLOBALE DE L'APPLICATION

RESUME :

Ce troisième chapitre propose un outil de spécification satisfaisant les caractéristiques requises exposées au chapitre 2. Il permet de décrire les aspects structuraux (conformes au modèle retenu : entités, ports et liaisons) et comportementaux (des entités et liaisons).

Son mode d'emploi (construction de la spécification et validations associées), pour la phase de structuration logique et la phase de structuration organique, sont ensuite décrits et illustrés. Au niveau logique, une méthode de partitionnement systématique de l'application en agents élémentaires est en particulier exposée. Le chapitre se termine par une réflexion sur la nature d'un environnement de conception adapté; la réalisation d'un outil d'édition graphique et de contrôle des structures est présentée.

1. L'OUTIL DE SPECIFICATION :

1.1. DESCRIPTION DES ENTITES (AGENTS OU MODULES) ET DES LIAISONS ENTRE ENTITES :

1.1.1. Les entités :

Chaque entité est identifiée par sa nature (agent ou module) et son nom ; tous les noms d'entités de même nature doivent être distincts.

L'interface d'une entité est constitué d'un ensemble de ports. Chaque port est identifié par un nom et caractérisé par sa catégorie (cf. ci-après) et un ou deux types de messages admissibles. Les noms de ports sur une même entité doivent être distincts; par contre deux ports sur deux entités différentes peuvent porter le même nom (on les distingue dans ce cas grâce à la "notation pointée" <nom-entité>.<nom-port>).

Il y a quatre catégories de ports selon le type d'action de communication qui y prend place :

- les ports d'entrée (**E-PORT**), pour les réceptions de messages d'un type donné,

- les ports de sortie (**S-PORT**), pour les émissions de messages d'un type donné,

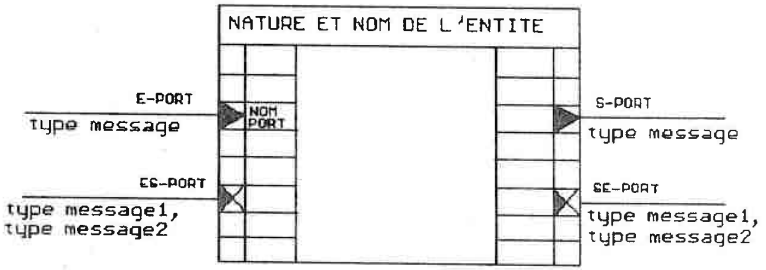
- les ports de sortie-entrée (**SE-PORT**), pour les émissions de requêtes d'un type donné avec attente des réponses d'un type donné,

- les ports d'entrée-sortie (**ES-PORT**), pour les réceptions de requêtes d'un type donné avec émission des réponses d'un type donné après traitement.

Les notations textuelles et graphiques que nous utiliserons dans la suite sont les suivantes :

```
{agent }nom-entité
{module}
     interface
       E-PORT nom-port : type-message;
       S-PORT nom-port : type-message;
       ES-PORT nom-port : type-message1 REPONSE type-message2;
       SE-PORT nom-port : type-message1 REPONSE type-message2;
     comportement
        liste de propriétés "comportementales" (cf. § 1.2.)
     implantation
        réseau d'entités communicantes (cf. § 1.3.)
fin nom-entité;
```

Fig. 3.1.

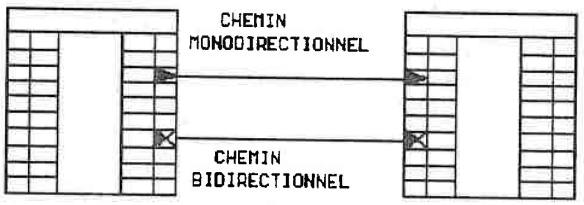


1.1.2. Les liaisons :

Les chemins de communication qui lient les ports des entités constituent la trame des réseaux. Par eux passent toutes les interactions entre entités.

On appelle chemin de communication monodirectionnel, un chemin liant un port de sortie à un port d'entrée; on appelle chemin de communication bidirectionnel, un chemin liant un port de sortie-entrée à un port d'entrée-sortie (ce sont les seuls assemblages autorisés).

Fig. 3.2.



On appelle liaison un ensemble de chemins de communication ayant un même port de sortie ou de sortie-entrée comme origine. On distingue :

- les liaisons bipoints, à un seul chemin
- les liaisons multipoints, à plusieurs chemins.

Une communication est une utilisation d'une liaison (tout ou partie de ses chemins) pour un échange d'information. On distingue les communications "1 vers 1" qui impliquent deux interlocuteurs et les communications "1 vers n" (ou diffusions) qui impliquent n+1 interlocuteurs sur une liaison comportant au moins n chemins.

Un de nos critères pour obtenir une structuration claire et compréhensible est l'absence d'ambiguïté au niveau de la représentation graphique; elle nous conduit à ajouter les restrictions de forme suivantes :

- les ports de sortie ou de sortie-entrée ne peuvent être origine que d'une seule liaison.
  Exemple : sans cette règle la fig. 3.3.ci-contre pourrait être interprétée comme une liaison multipoint ou comme 3 liaisons bipoints.
  ♦
  Par contre, un port d'entrée ou d'entrée-sortie peut être partagé entre plusieurs liaisons.

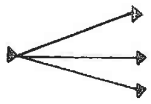


Fig. 3.3.

- une liaison multipoint à p chemins véhicule uniquement des communications "1 vers 1" ou des communications "1 vers p"; on parle respectivement de liaison multipoint utilisée "en sélection" ou "en diffusion"; une * près du port origine repère les diffusions. Ceci supprime toute ambiguïté sur les interlocuteurs effectifs des communications prenant place sur ces liaisons.

- seules les liaisons monodirectionnelles accueillent des diffusions; en effet, une diffusion dans le cas bidirectionnel nous apparaît être une forme de transaction peu claire (ex: type du message réponse ?) et probablement peu utile.

En résumé, nous distinguons 5 types de liaisons dont nous récapitulons les notations textuelles et graphiques :

- liaison monodirectionnelle bipoint :
  ⟨nom-S-PORT⟩ **vers** ⟨nom-E-PORT⟩

Fig. 3.4.a.

- liaison bidirectionnelle bipoint :
  ⟨nom-SE-PORT⟩ **vers** ⟨nom-ES-PORT⟩

Fig. 3.4.b.

- liaison monodirectionnelle multipoint en diffusion :
  ⟨nom-S-PORT⟩ **vers** ⟨nom-E-PORT$_1$⟩ **et** ... **et** ⟨nom-E-PORT$_n$⟩

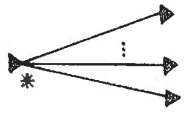


Fig. 3.4.c.

- liaison monodirectionnelle multipoint en sélection :
  <nom-S-PORT> **vers** <nom-E-PORT$_1$> **ou** ... **ou** <nom-E-PORT$_n$>

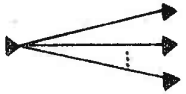


Fig. 3.4.d.

- liaison bidirectionnelle multipoint en sélection :
  <nom-SE-PORT> **vers** <nom-ES-PORT$_1$> **ou** ... **ou** <nom-ES-PORT$_n$>

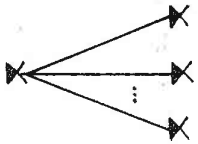


Fig. 3.4.e.

1.2. DESCRIPTION FORMELLE DU COMPORTEMENT DES ENTITES :

1.2.1. Les événements :

Les ports sont le siège d'événements atomiques (ils surviennent complètement ou pas du tout) et instantanés (de durée nulle). Il s'agit, selon le type des ports :
- d'émissions de messages (ports de sortie),
- de réceptions de messages (ports d'entrée),
- d'émissions de requêtes et de réceptions de réponses (ports de sortie-entrée),
- de réceptions de requêtes et d'émissions de réponses (ports d'entrée-sortie).

Tous les événements sur un port sont strictement ordonnés.

Notations :

- **EV**(<nom-port>) représente l'ensemble des événements sur un port ; **EV**(**ENT**<nom-entité>) représente l'ensemble des événements sur une entité ; **EV**(**SYS**) représente l'ensemble des événements du système étudié.

- <nom-événement>ε**EV**(<nom-port>), représente un événement sur un port (on notera dans la suite tous les noms d'événements en minuscules et tous les noms de ports en majuscules).

- <nom-événement>.**msg** représente le message associé à l'événement et <nom-événement>.**msg**.<nom-champ>, un champ de ce message. Les différents champs et leurs types (entier, réel, chaîne de caractères, type construit par énumération) peuvent être précisés dans une déclaration de type de message : **type-message** <nom-type-message>
  <nom-champ$_1$> : <type>;
  ...
  <nom-champ$_n$> : <type>;
  fin;

Les types construits par énumération peuvent être spécifiés par : **type** <nom-type> = (<liste-valeurs>);

- **ord**(<nom-événement>) est un entier positif qui représente le numéro d'ordre de l'événement sur le port.

- sur un port d'entrée-sortie ou de sortie-entrée on peut distinguer l'émission (sortie) et la réception (entrée) par :
  <nom-événement>ε**EV**(<nom-port>(**E**)) pour l'entrée,
  <nom-événement>ε**EV**(<nom-port>(**S**)) pour la sortie.

- sur un port origine d'une liaison multipoint ou extrémité de plusieurs liaisons on peut spécifier des événements relatifs à un des chemins en adjoignant au nom du port un indice associé au chemin :
  <nom-événement>ε**EV**(<nom-port>(<indice>))
  <nom-événement>ε**EV**(<nom-port>($\{\mathbf{E}\atop\mathbf{S}\}$,<indice>))

Lorsque deux ports sont liés par un chemin de communication, cette liaison est "transparente", c'est à dire que **tout événement sur le port origine se confond avec un événement sur le port extrémité** (la communication est sûre et instantanée). On peut bien entendu spécifier explicitement des liaisons "non transparentes", grâce à une entité intermédiaire ayant les propriétés voulues.

1.2.2. Les relations de précédence entre événements :

a) la précédence causale :

Deux événements a et b sont en précédence causale, notée a=>b, si l'événement a est la cause directe ou indirecte (par un enchaînements de causalités) de b : "a déclenche b".

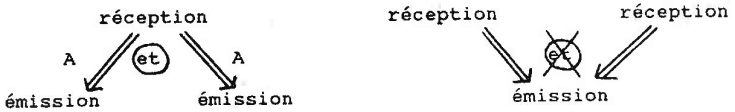
Les propriétés de la précédence causale sont :

- l'antiréflexivité : ~(a=>a) (ou ~ représente la négation logique),
- l'antisymétrie : si a=>b alors ~(b=>a)
- la transitivité : si a=>b et b=>c alors a=>c

Nous notons $a \overset{A}{\Rightarrow} b$ une causalité locale à l'entité A (c'est à dire ne mettant en jeu aucune communication avec d'autres entités).
Habituellement, il s'agit d'une réception qui déclenche directement une émission. Beaucoup plus rarement on peut trouver des causalités entre émissions, directes ou non et donc des causalités indirectes entre réceptions et émissions. Une causalité locale entre réceptions n'a pas de sens.

Une réception peut être la cause de plusieurs émissions, même directement. Par contre, toute émission a une cause effective unique même si elle possède des causes potentielles multiples.

Fig. 3.5.



b) la précédence temporelle :

La relation de précédence temporelle est la relation d'ordre partiel définie de la manière suivante [GRE77,LAM78] : sur le même site, deux événements a et b sont en précédence temporelle, notée a->b, si a survient avant b selon l'horloge du site; si a et b sont sur des sites différents, on peut assurer que a->b lorsque a est l'émission d'un message et b sa réception, après une communication de durée non nulle. Pour beaucoup d'événements sur des sites différents, il est impossible de dire si l'un précède l'autre ou réciproquement : on dit qu'ils sont concurrents (notation : a // b).
Exemple :

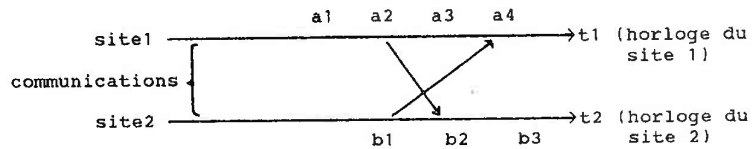


Fig. 3.6. on a : a1->a2, a2->b2, a1->b3, ...
a2 // b1, a3 // b3, a4 // b2, ...

♦

Nous notons a<->b la simultanéité de a et b; il peut s'agir d'événements simultanés sur le même site ou d'une émission et d'une réception pour une communication de durée nulle. Pour tout couple d'événements a et b, on a :
a->b ou b->a ou a<->b ou a // b (ou exclusifs)

Les propriétés de la précédence temporelle sont :
- l'antiréflexivité : ~(a->a),
- l'antisymétrie : si a->b alors ~(b->a),
- la transitivité : si a->b et b->c alors a->c.

1.2.3. Le modèle "simplifié" et le modèle "complet" :

La relation de précédence temporelle, telle que nous venons de la définir, est parfaitement adaptée à la spécification du comportement des modules (sites virtuels sans horloge commune). Elle l'est moins pour les agents de la phase logique, auxquels la notions de site est étrangère.

Nous proposons de moduler la "finesse" du modèle selon le stade de la démarche et le type de validation à réaliser.

a) le modèle "simplifié" :

Au niveau logique, nous proposons un modèle "simplifié" où l'on fait abstraction du temps "lié à l' implantation" pour ne considérer que le "temps réel de l' environnement" [GUE80]. C'est à dire que tous les aspects temporels liés à l' implantation (durée des traitements, durée des communications) sont abstraits, éliminant ainsi les problèmes liés à la répartition et à la multiplicité des horloges. Deux événements en relation de causalité sont confondus (sauf temporisation explicite selon le temps réel). Seules subsistent les précédences temporelles liées aux instants d'occurrence d'événements externes datés dans le temps réel de l'environnement. Dans ce modèle, le système évolue de manière instantanée à chaque événement à l'interface (ou lors d'événements internes rythmés par le temps réel). Tous les événements sont distincts et ordonnés dans ce temps réel.

La relation qui lie précédence causale et précédence temporelle s'énonce, dans cette optique :
a=>b implique ~(b->a) et plus précisément a<->b sauf si une temporisation selon le temps réel est explicitement indiquée, auquel cas a->b.

Dans ce modèle, la cohérence d'une implantation par rapport à la spécification globale se vérifie par la **conservation** dans le réseau des propriétés "comportementales" spécifiées au niveau global (les conséquences de la répartition ayant été gommées).

b) le modèle "complet" :

La notion de site apparaît au niveau organique , ce qui nous incite à utiliser le modèle complet de CHEN & YEH [CHE83], en vue de spécifier et de valider **d'autres types de propriétés**. Les événements sont toujours atomiques et instantanés mais la relation de causalité n'implique plus que les événements soient confondus : des délais existent qui modélisent les temps de traitement et de communication. Les événements non ordonnés par la précédence temporelle sont concurrents.

Ce niveau de description nous permet par exemple de spécifier et valider des propriétés sur les chaînes d'événements en relation de causalité, ce que le modèle "simplifié" ne permet pas, tous les événements en relation de causalité y étant confondus (cf. spécification et validation d'un protocole du type "bit alterné" entre modules, au § 2.2).

La relation qui lie précédence causale et précédence temporelle s'énonce, dans cette optique : a=>b implique a->b .

Remarque : la proposition de CHEN manque de précision sur certains points. C'est ainsi que les traitements des messages sont définis comme des événements atomiques et instantanés au même titre que les émissions et réceptions de messages. Mais rien ne précise comment se situent ces traitements par rapport aux événements de communication. Prenons le cas de deux événements de réception et d'émission en relation de causalité sur une même entité A (événements a et b tels que : a=>b). L'état de A devrait être évalué au moment de l'événement "traitement de a.**msg**", en fonction de tous les événements antérieurs. Or on s'aperçoit, au vu des exemples (cf. propriété NW11 de [CHE83]), que l'état du système est défini au moment de la réception du message déclencheur et non au moment de son traitement.

Nous ne considérons ici que les événements de communication et posons a priori que l'état du système est défini au moment de la réception du message déclencheur, ce qui rapproche modèle "simplifié" et modèle "complet".

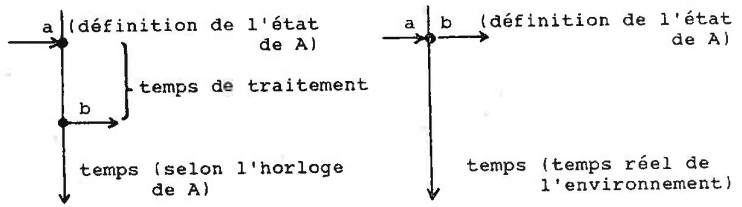


Fig. 3.7.

♦

1.2.4. La spécification des propriétés "comportementales" des entités :

Le comportement des entités est spécifié sous la forme d'un ensemble de propriétés exprimées à l'aide :
- des événements à l'interface de l'agent,
- des messages associés aux événements,
- d'opérateurs (par priorités décroissantes) :
  + unaires (quantification et négation) : ∀,∃,~
  + relationnels : ε,=,≡,=>,->,<->,//
  + logiques : v,∩
  + d'implication : #>,<#>

Ces propriétés peuvent être soit des propriétés invariantes ("safety properties" [OWI82]), soit des propriétés d'accessibilité ("liveness properties"), grâce à la précédence causale, soit des propriétés sur les données ("data oriented properties").

Parmi les propriétés les plus courantes, on peut citer :

- les déclenchements inconditionnels de sorties sur des entrées :
ex: $\forall a(a\varepsilon \mathbf{EV}(A) \#> \exists b(b\varepsilon \mathbf{EV}(B) \cap a=>b))$
S

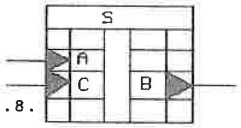


Fig.3.8.

- les déclenchements inconditionnels exclusifs de sorties sur des entrées :
ex: $\forall a(a\varepsilon \mathbf{EV}(A) <\#> \exists b(b\varepsilon \mathbf{EV}(B) \cap a=>b))$
S

- les déclenchements conditionnels de sorties sur des entrées :
ex: $\forall a(a\varepsilon \mathbf{EV}(A) \cap P \#> \exists b(b\varepsilon \mathbf{EV}(B) \cap a=>b))$
S
où P est un prédicat quelconque.

- les déclenchements conditionnels exclusifs de sorties sur des entrées :
ex: $\forall a(a\varepsilon \mathbf{EV}(A) \cap P <\#> \exists b(b\varepsilon \mathbf{EV}(B) \cap a=>b))$
S

- les ensembles de toutes les causes possibles (conditionnelles ou inconditionnelles) de sorties :
ex: $\forall b(b\varepsilon \mathbf{EV}(B) \#> (\exists a(a\varepsilon \mathbf{EV}(A) \cap P \cap a=>b) \text{ v } \exists c(c\varepsilon \mathbf{EV}(C) \cap Q \cap c=>b))$
S

- les propriétés sur les valeurs des messages :
ex: $\forall a,\forall b(a\varepsilon \mathbf{EV}(A) \cap b\varepsilon \mathbf{EV}(B) \cap a=>b \#> b.\mathbf{msg}=F(a.\mathbf{msg}))$
S
où F est une fonction quelconque.

- l'absence de duplication en sortie :

ex: ∀a,∀b1,∀b2(a∈**EV**(A)∩b1∈**EV**(B)∩b2∈**EV**(B)∩a=>b1∩
S
a=>b2 #> b1≡b2)
S

- la conservation de l'ordre :

ex: ∀a1,∀a2,∀b1,∀b2(a1∈EV(A)∩a2∈EV(A)∩b1∈EV(B)∩
b2∈**EV**(B)∩a1=>b1∩a2=>b2 #> (a1->a2∩b1->b2) v (a2->a1∩
S S
b2->b1) v (a1≡a2∩b1≡b2))

Il est également possible de spécifier des propriétés "temps-réel", grâce à la fonction **intervalle** qui donne le temps écoulé entre 2 événements.

Exemple: horloge H

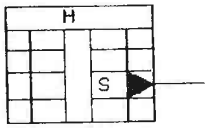


Fig.3.9.

∀s1,∀s2(s1∈**EV**(S)∩s2∈**EV**(S)∩s1=>s2∩**ord**(s2)=**ord**(s1)+1
H
#>**intervalle**(s1,s2)=5)

♦

Les prédicats associés aux déclenchements conditionnels sont souvent pénibles à écrire; en effet, l'état de l'entité au moment de la réception du message déclencheur (et plus généralement tout "effet de bord" entre propriétés) n'est exprimable que par le biais de références à l'historique des événements passés et parfois aux valeurs des messages de cet historique. En outre, les mêmes expressions se retrouvent souvent dans plusieurs propriétés. Pour alléger l'écriture des propriétés, nous proposons d'introduire des variables d'état dont les valeurs reflètent (mémorisent) une certaine histoire passée : elles sont utilisées lorsqu'on le juge utile dans les prédicats; leur définition en termes du modèle est donnée séparément et leur détermination peut s'appuyer sur des outils classiques tels les diagrammes de transitions.

Dans la même optique de simplification et d'amélioration de la lisibilité des spécifications, on pourrait également introduire des abréviations pour les propriétés courantes, regrouper les propriétés en chapitres logiques, introduire des commentaires ou même enrichir le modèle en introduisant des conjonctions d'événements déclencheurs, par exemple. Nous n'avons pas approfondi pour l'instant ces questions.

Exemple : spécification d'un chariot "autodirectif", c'est à dire d'un chariot qui suit automatiquement un guide noyé dans le sol (agent AUTO); il transforme des messages du type "position" en messages du type "correction-direction", à condition que le pilotage automatique ait été mis en marche par un message du type "commande" valant "on"; quand le pilotage automatique est arrêté par une commande valant "off", le chariot ne réagit plus aux positions reçues en entrée.

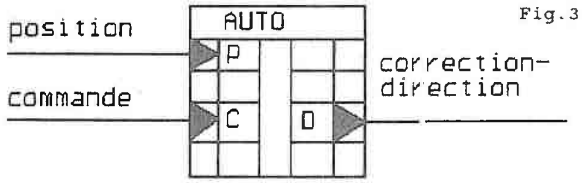


Fig.3.10.

agent AUTO
interface
E-PORT P : position ;
E-PORT C : commande ;
S-PORT D : correction-direction ;
comportement
(A1) ∀p(p∈**EV**(P)∩∃c(c∈**EV**(C)∩c.**msg**=on∩c->p)∩~∃c1(c1
∈**EV**(C)∩c1.**msg**=off∩c->c1->p) <#> ∃d(d∈**EV**(D)∩p=>
d)) (~)
(A2) ∀p,∀d(p∈**EV**(P)∩d∈**EV**(D)∩p=>d #> d.**msg**=F(p.**msg**))
fin AUTO;
type-message commande
val : tycde;
fin;
Type tycde = (on,off);

Il est possible d'introduire une variable d'état ETAT-PILOTE qui indique l'état de fonctionnement du pilote automatique au moment d'un événement t quelconque :

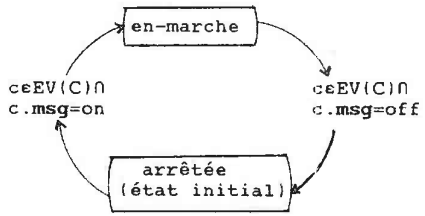
ETAT-PILOTE=en-marche **quand** t : ∃c(c∈**EV**(C)∩c.**msg**=on∩
c->t)∩~∃c1(c1∈**EV**(C)∩c1.**msg**=off∩c->c1->t)

ETAT-PILOTE=arrêté **quand** t : ~∃c(c∈**EV**(C)∩c->t)v
(∃c(c∈**EV**(C)∩c.**msg**=off∩c->t)∩~∃c1(c1∈**EV**(C)∩
c1.**msg**=on∩c->c1->t))

Avec cette variable, la propriété (A1) se réécrit :

(A1) ∀p(p∈**EV**(P)∩ETAT-PILOTE=en-marche **quand** p <#> ∃d
(d∈**EV**(D)∩p=>d))

Le diagramme de transitions d'états que l'on peut associer à la variable ETAT-PILOTE est le suivant :



♦ Fig.3.11.

(~) a->b->c abrège a->b ∩ b->c ; c.**msg** est utilisé pour c.**msg**.val comme le message commande n'a qu'un seul champ.

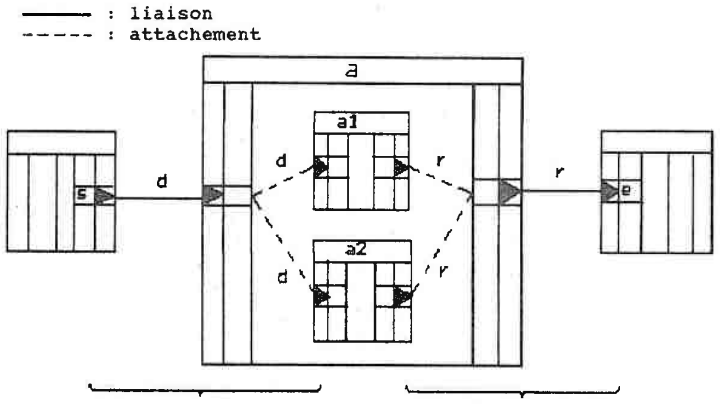
1.3. DESCRIPTION DE L'IMPLANTATION DES ENTITES ET VALIDATIONS ASSOCIEES :

1.3.1. Implantation d'une entité en un réseau d'entités communicantes :

Lorsque l'on décompose une entité en un réseau d'entités on doit respecter l'identité des ports de l'entité englobante et des ports externes du réseau. On est donc souvent amené à introduire des entités artificielles pour distribuer et concentrer des messages ayant plusieurs destinataires ou plusieurs origines dans le réseau. La notion (notation) d'attachement nous permet d'en faire l'économie. Tous les ports de l'entité à raffiner sont attachés à un ou plusieurs ports de même type du réseau (avec possibilité de sélection ou de diffusion) : ils deviennent de simples points de continuation des chemins de communication; en entrée, en cas d'attachement multiple, il y a changement de profil de la liaison (ex : bipoint à multipoint en sélection ou diffusion, multipoint à n chemins en diffusion à multipoint à n+m chemins en diffusion); en sortie, en cas d'attachement multiple, il y a multiplication de liaisons de mêmes caractéristiques. Les notations graphiques et textuelles des attachements sont introduites dans l'exemple ci-dessous.

Exemple :

Fig. 3.12.



la liaison bipoint ainsi "prolongée" équivaut à une liaison multipoint à 2 chemins en sélection.

la liaison bipoint ainsi "prolongée" équivaut à 2 liaisons bipoints avec le même port de réception.


```
agent a
     interface
       E-PORT x : d;
       S-PORT y : r;
     comportement
          ...
     implantation
       agent a1;
          interface
            E-PORT x : d;
            S-PORT y : r;
          comportement
            ...
       fin a1;
       agent a2;
          interface
            E-PORT u : d;
            S-PORT v : r;
          comportement
            ...
       fin a2;
       liaisons
       attachements
           a.x à a1.x ou u;
           a1.y à a.y;
           v à a.y;
fin a;
```

♦

Une certaine cohérence doit bien sûre être maintenue en cas d'attachements successifs à plusieurs niveaux : en aucun cas un chemin de communication "prolongé" par des attachements ne doit mélanger diffusions et sélections (cf. règles du § 1.1.2.) ni comporter plus d'un "tronçon" du type liaison (les autres étant du type attachement). L'outil de manipulation des spécifications devra contrôler l'application de ces règles (cf. § 2.3.2).

1.3.2. Validation de la cohérence d'une implantation :

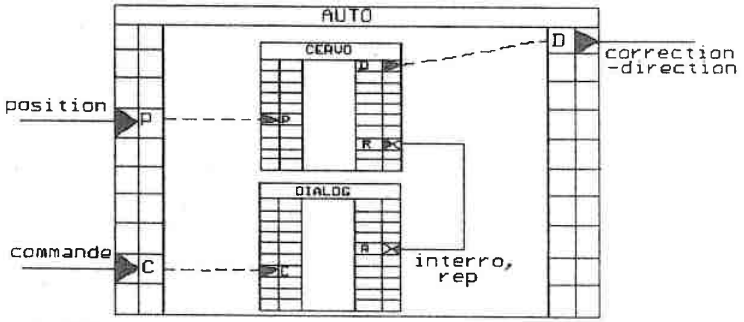
a) principes :

Nous cherchons à prouver la cohérence de l'implantation vis à vis du comportement global de l'entité; ceci implique en particulier que toutes les liaisons entre entités du réseau aient été définies et qu'elles véhiculent les bonnes informations, que les attachements soient adéquats et que les propriétés des entités du réseau se composent correctement au regard des propriétés globales. Dans le modèle "simplifié", cohérence est synonyme de conservation des propriétés et nous cherchons à retrouver, à partir des propriétés des entités du réseau, les propriétés de l'agent global.

Ces preuves sont établies propriété par propriété : elles peuvent se comparer à des preuves de théorèmes [CHE83].

Exemple : supposons que l'agent AUTO, déjà spécifié précédemment, soit décomposé en deux agents DIALOG (qui assure le dialogue avec l'opérateur) et CERVO (qui calcule la trajectoire).

Fig.3.13.




```
agent AUTO
  interface
    E-PORT P : position ;
    E-PORT C : commande ;
    S-PORT D : correction-direction ;
  comportement
     (A1) ∀p(pεEV(P)∩]c(cεEV(C)∩c.msg=on∩c->p)∩~]c1(c1ε
          EV(C)∩c1.msg=off∩c->c1->p) <#> ]d(dεEV(D)∩p=>d))
     (A2) ∀p,∀d(pεEV(P)∩dεEV(D)∩p=>d #>d.msg=F(p.msg))
  implantation
     agent DIALOG
       interface
         E-PORT C : commande;
         ES-PORT R : interro REPONSE rep;
       comportement
         (D1) ∀r(rεEV(R(E))∩]c(cεEV(C)∩c.msg=on∩c->r)∩
              ~]c1(c1εEV(C)∩c1.msg=off∩c->c1->r) <#> ]r1
              (r1εEV(R(S))∩r=>r1∩r1.msg=marche))
         (D2) ∀r(rεEV(R(E))∩(~]c(cεEV(C)∩c->r) v (]c(cε
              EV(C)∩c.msg=off∩c->r)∩~]c1(c1εEV(C)∩c1.msg=on
              ∩c->c1->r))) <#> ]r1(r1εEV(R(S))∩r=>r1∩r1.msg=
              arrêt))
     fin DIALOG;
     agent CERVO
       interface
         E-PORT P : position
         S-PORT D : correction-direction;
         SE-PORT R : interro REPONSE rep;
       comportement
        (C1) ∀p(pεEV(P) <#> ]r(rεEV(R(S))∩p=>r))
        (C2) ∀r(rεEV(R(E))∩r.msg=marche <#> ]d(dεEV(D)∩
             r=>d))
        (C3) ∀r,∀d(rεEV(R(E))∩dεEV(D) ∩r=>d #> ]p(pεEV
             (P)∩ord(p)=ord(r))∩d.msg=F(p.msg))
     fin CERVO;
```



```
     liaisons
       CERVO.R vers DIALOG.R;
     attachements
       AUTO.C à DIALOG.C;
       AUTO.P à CERVO.P;
       CERVO.D à AUTO.D;
fin AUTO;
type-message commande
     val : tycde;
fin;
type-message réponse
     val : tyrep;
fin;
Type tycde = (on,off);
Type tyrep = (marche,arrêt);
```

Montrons à titre d'exemple, et de manière semi-formelle, la conservation de la propriété (A1) dans l'implantation de AUTO :

(a) tout pεEV(AUTO.P) se confond avec un pεEV(CERVO.P), car les deux ports sont attachés.

(b) ∀p(pεEV(AUTO.P) <#>]r(rεEV(CERVO.R(S))∩p=>r)), d'après (a) et (C1).

(c) tout rεEV(CERVO.R(S)) se confond avec un rεEV(DIALOG.R(E)), car les deux ports sont liés.

(d) ∀p(pεEV(AUTO.P) <#>]r(rεEV(DIALOG.R(E))∩p=>r)), d'après (b) et (c).

(e) tout cεEV(DIALOG.C) se confond avec un cεEV(AUTO.C), car les deux ports sont attachés.

(f) rεEV(DIALOG.R(E)) <-> pεEV(AUTO.P), d'après (d) et la règle liant => et -> dans le modèle "simplifié".

(g) ∀p(pεEV(AUTO.P)∩]c(cεEV(AUTO.C)∩c.msg=on∩c->p)∩~]c1(c1εEV(AUTO.C)∩c1.msg=off∩c->c1->p) <#>]r(rεEV(DIALOG.R(S))∩p=>r∩r.msg=marche)), d'après (d),(e),(f),(D1) et la transitivité de =>.

(h) tout rεEV(DIALOG.R(S)) se confond avec un rεEV(CERVO.R(E)), car les deux ports sont liés.

(i) ∀p(pεEV(AUTO.P)∩]c(cεEV(AUTO.C)∩c.msg=on∩c->p)∩~]c1(c1εEV(AUTO.C)∩c1.msg=off∩c->c1->p) <#>]r(rεEV(CERVO.R(E))∩p=>r∩r.msg=marche)), d'après (g) et (h).

(j) ∀p(pεEV(AUTO.P)∩]c(cεEV(AUTO.C)∩c.msg=on∩c->p)∩~]c1(c1εEV(AUTO.C)∩c1.msg=off∩c->c1->p) <#>]d(dεEV(CERVO.D)∩p=>d)), d'après (i),(C2) et la transitivité de =>.

(k) tout dεEV(CERVO.D) se confond avec un dεEV(AUTO.D), car les deux ports sont attachés.

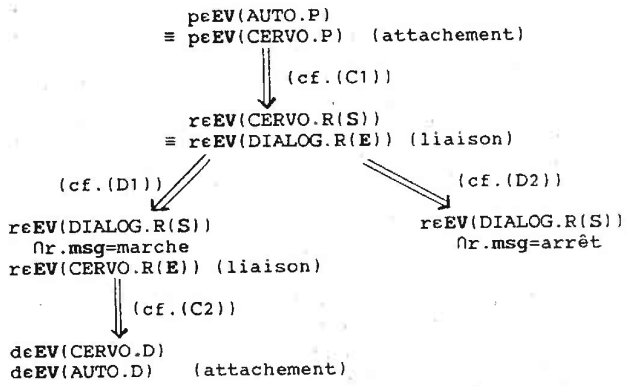
(l) ∀p(pεEV(AUTO.P)∩]c(cεEV(AUTO.C)∩c.msg=on∩c->p)∩~]c1(c1εEV(AUTO.C)∩c1.msg=off∩c->c1->p) <#>]d(dεEV(AUTO.D)∩p=>d)), d'après (j) et (k).

♦ C.Q.F.D.

b) Démarche pratique :

Les démonstrations des propriétés classiques (déclenchements inconditionnels exclusifs ou non, déclenchements conditionnels exclusifs ou non, ensemble de causalités et absence de duplication - cf. § 1.2.4.), sont assez standardisées et leur pratique est facilitée par la construction du graphe de tous les enchaînements de causalités dans le réseau.

Exemple : Fig.3.14.



Les schémas de preuve pour ces propriétés classiques et pour des graphes sans cycles sont les suivants :

(I) dans le cas d'une propriété globale quelconque qui se retrouve à l'identique dans une des entités du réseau, il suffit de vérifier que les ports qui se correspondent sont attachés.

(II) dans le cas d'une propriété du type déclenchement inconditionnel non exclusif (resp. exclusif), il faut que les ports impliqués dans la propriété au niveau global soient attachés à des ports dans le réseau et :
+ qu'il existe une suite de déclenchements inconditionnels non exclusifs (resp. exclusifs) les reliant dans le graphe,
+ ou qu'il existe plusieurs suites de déclenchements inconditionnels ou non les reliant et dont le conditionnement d'ensemble soit équivalent au prédicat vrai (resp. avec vérification de l'exclusivité du déclencheur).

(III) dans le cas d'une propriété du type déclenchement conditionnel non exclusif (resp. exclusif), il faut que les ports impliqués dans la propriété au niveau global soient attachés à des ports dans le réseau et qu'il existe une ou des suites de déclenchements les reliant dans le graphe et dont le conditionnement d'ensemble soit équivalent au conditionnement de la propriété globale (resp. avec vérification de l'exclusivité du déclencheur).

(IV) dans le cas d'une propriété sur l'ensemble des causalités possibles d'une sortie, il faut que les ports impliqués dans la propriété au niveau global soient attachés à des ports dans le réseau et qu'il existe une

ou des suites de déclenchements dans le graphe permettant de remonter du port de sortie au(x) port(s) d'entrée, avec les mêmes éventuels conditionnements.

(V) dans le cas d'une propriété de déclenchement sans duplication, il suffit de montrer qu'il lui correspond dans le graphe un enchaînement de déclenchements sans duplication.

Nous rencontrerons dans la suite de la thèse des exemples de validations selon ces schémas standards et des validations plus spécifiques (par exemple, sur un graphe des causalités avec cycle, au § 2.2.2).

Un démonstrateur, centré sur la vérification des enchaînements de causalités, écrit en PROLOG, est également en cours de réalisation.

2. MODE D'EMPLOI DE L'OUTIL :

2.1. EMPLOI DE L'OUTIL LORS DE LA PHASE DE STRUCTURATION LOGIQUE :

2.1.1. Les différents niveaux d'utilisation et les stratégies de validation :

L'outil de spécification peut être utilisé au niveau de l'agent unique qui modélise l'ensemble de l'application. Mais il n'est pas toujours possible de décrire "ex abrupto" une application de grande taille comme une boîte noire unique. Dans ce cas, le concepteur peut procéder à un premier découpage "macroscopique" en plusieurs sous-systèmes de tailles plus raisonnables qui constituent autant de points de départ de la spécification.

Au cours de la phase logique, le concepteur procède à la décomposition de l'application en un réseau d'agents élémentaires. Il peut travailler en une ou en plusieurs étapes, de manière descendante (par raffinements successifs), ascendante (par abstractions successives) ou mixte. A chaque niveau de décomposition l'outil de spécification peut être utilisé.

Pour les vérifications de cohérence, deux stratégies sont possibles :

- vérifier uniquement le niveau terminal en agents élémentaires par rapport à l'agent global unique : il suffit de vérifier que chaque propriété de l'agent global se retrouve au niveau terminal; le nombre de vérifications est minimum mais chacune peut être très lourde à réaliser;
- vérifier chaque niveau par rapport au niveau immédiatement supérieur : les vérifications sont beaucoup plus nombreuses mais aussi beaucoup plus simples à réaliser.

2.1.2. Elaboration de la spécification globale :

a) la démarche proposée :

L'établissement des propriétés de l'agent global (ou d'un sous-système) à partir de la description informelle de l'application est un travail difficile. Nous proposons, pour "dégrossir" le problème, de rechercher tout d'abord les propriétés essentielles (les causalités) et les variables d'état utiles pour caractériser l'état du système lors de ces déclenchements, sans entrer dans le détail de leur formulation. Nous proposons de réfléchir en termes de relations de **déclenchement**, de **mémorisation** et de **contribution**, selon le canevas suivant :

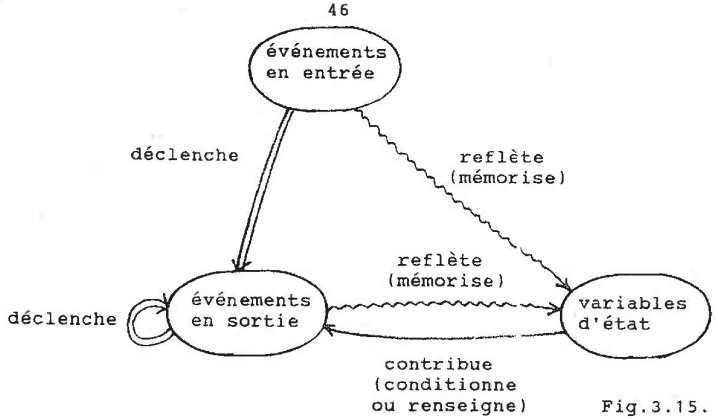


Fig.3.15.

Les variables d'état reflètent (mémorisent) une certaine histoire en termes des événements en entrée et en sortie; elles contribuent aux événements en sortie, en les conditionnant ou en leur apportant leurs valeurs (relation "renseigne"). Après avoir établi les listes de ces relations, le concepteur peut affiner la définition des variables d'état complexes en s'aidant de diagrammes de transitions d'états (cf. § 1.2.4.). Il est alors en mesure de donner leur spécification précise dans le modèle, puis celle des propriétés principales de l'agent.

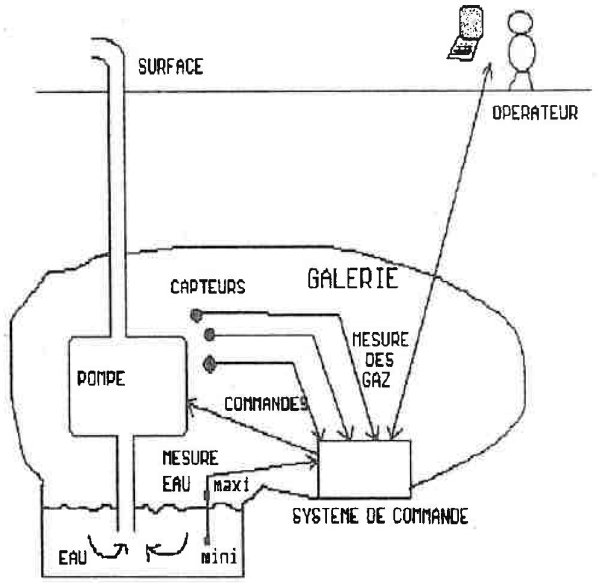
b) un exemple : la pompe de KRAMER [KRA80].

Il s'agit du système de commande d'une pompe de drainage située dans une galerie d'une mine de charbon et de contrôle de son environnement atmosphérique. La pompe, après avoir été mise en état de fonctionnement par un ordre de l'opérateur en surface, marche automatiquement; démarrage lorsque l'eau est mesurée au dessus d'un niveau maximum et arrêt lorsque l'eau est mesurée en dessous d'un niveau minimum. Pour des raisons de sécurité, la pompe ne peut pas démarrer ou continuer à fonctionner lorsque le pourcentage de méthane dans l'air dépasse un certain seuil. Des capteurs de méthane, de monoxyde de carbone et de ventilation renseignent périodiquement le système. L'opérateur en surface peut à tout instant interroger le système pour connaître l'état de l'atmosphère. Des alarmes, en cas de dépassement des seuils relatifs aux gaz et à la ventilation, sont émises vers la surface.

Remarque : le programme en CONIC est notre définition du problème. C'est à dire que notre spécification rend compte **dans tous ses détails du comportement induit par ce programme** (ex: si la pompe est arrêtée à cause du méthane, seul un ordre de l'opérateur peut la faire repartir). On pourrait bien entendu définir beaucoup plus simplement une application de commande d'une pompe, mais ce ne serait pas **LA** POMPE DE KRAMER.

♦

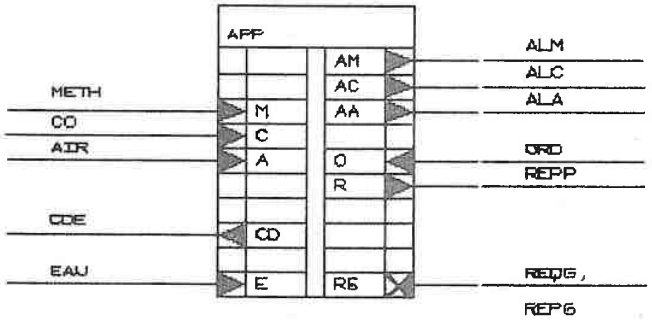
Fig. 3.16,



Le système étant de taille modeste, le concepteur peut en rendre compte par un agent unique (APP). Le concepteur définit les types de messages externes en entrée et sortie. Remarquons au passage l'importance de ce choix qui fixe en fait le niveau de la commande considérée (certains messages peuvent abstraire un niveau de commande locale de plus bas niveau) :

- les messages des types EAU, METH, CO, AIR véhiculent les valeurs mesurées par les capteurs,
- les messages des types ALA, ALM, ALC sont les signaux purs d'alarme pour l'air, le méthane et le CO,
- les messages du type REQG précisent la nature du gaz à tester (méthane, co, air),
- les messages du type REPG comportent la nature du gaz et son état (normal, anormal),
- les messages du type CDE donnent la nature de la commande à la pompe (marche, arrêt),
- les messages du type ORD précisent la nature de l'ordre de l'opérateur (prêt, couper, requête),
- les messages du type REPP indiquent l'état de la pompe (coupée, prête, en-marche, arrêtée-eau, arrêtée-méthane).

Fig. 3.17.



Les relations de déclenchements, de contribution et de mémorisation retenues sont les suivantes :

Relations de déclenchement (⟹) :

no	événements déclencheurs	événements déclenchés	commentaires
r1	a1εEV(A)	a2εEV(AA)	alarmes sur mesures des gaz
r2	cεEV(C)	aεEV(AC)	
r3	mεEV(M)	aεEV(AM)	
r4	oεEV(O)	rεEV(R)	réponse à un ordre de l'opérateur
r5	r1εEV(RG(E))	r2εEV(RG(S))	réponse à une requête sur un gaz
r6	mεEV(M), eεEV(E), oεEV(O)	cεEV(CD)	commande au moteur de la pompe

Relations de contribution (⟶) :

no	variables	événements concernés	commentaires
r7	ETAT-METHANE, ETAT-POMPE	cεEV(CD)	conditionnent la mise en route ou l'arrêt de la pompe
r8	ETAT-POMPE	rεEV(R)	"renseigne" la réponse
r9	ETAT-METHANE, ETAT-AIR, ETAT-CO	rεEV(RG(S))	"renseignent" la réponse

Relations de mémorisation (⟿) :

no	événements caractéristiques	variables	commentaires
r10	mεEV(M)	ETAT-METHANE	
r11	aεEV(A)	ETAT-AIR	
r12	cεEV(C)	ETAT-CO	
r13	oεEV(O), mεEV(M), eεEV(E) cεEV(CD)	ETAT-POMPE	cf. l'analyse détaillée ci-après

La variable ETAT-POMPE, qui présente une certaine complexité, est disséquée à l'aide du diagramme de transitions de la figure 3.18.

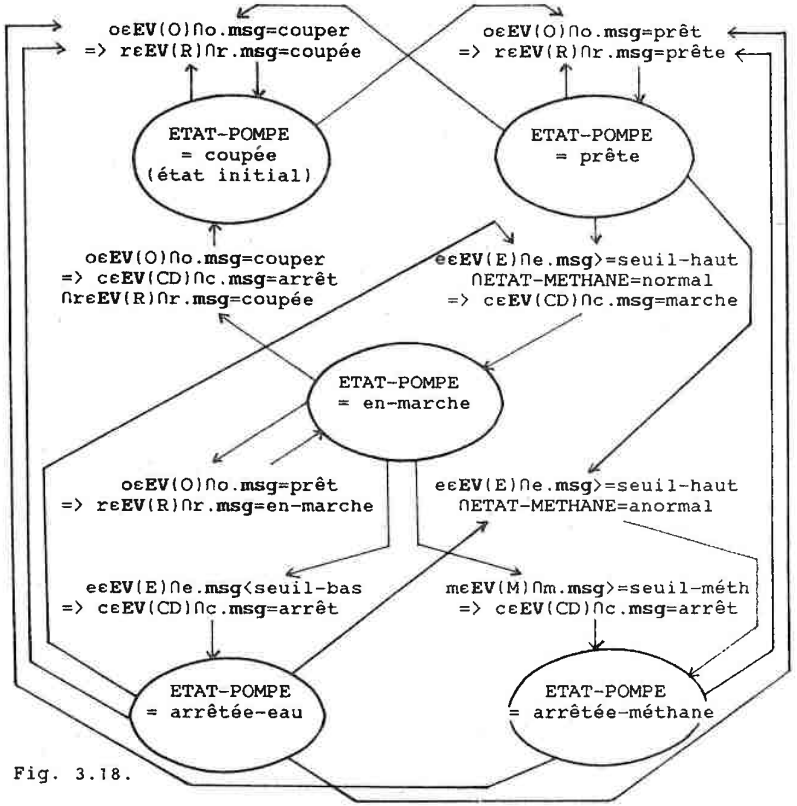


Fig. 3.18.

Les variables d'état peuvent alors être spécifiées de la manière suivante :

ETAT-METHANE = normal **quand** t équivaut à :
∃m1(m1εEV(M)∩m1->t∩m1.msg<seuil-méth)∩~∃m2(m2εEV(M)∩m1->m2->t) (˜)

ETAT-METHANE = anormal **quand** t équivaut à :
∃m1(m1εEV(M)∩m1->t∩m1.msg>=seuil-méth)∩~∃m2(m2εEV(M)∩m1->m2->t)

ETAT-AIR et ETAT-CO sont spécifiées exactement sur le même modèle.

ETAT-POMPE = en-marche **quand** t équivaut à :
∃c1(c1εEV(CD)∩c1->t∩c1.msg=marche)∩~∃c2(c2εEV(CD)∩c1->c2->t)

ETAT-POMPE = coupée **quand** t équivaut à :
~∃o1(o1εEV(O)∩o1->t∩o1.msg=prêt) v (∃o1(o1εEV(O)∩o1->t∩o1.msg=couper)∩~∃o2(o2εEV(O)∩o1->o2->t∩o2.msg=prêt))

ETAT-POMPE = arrêtée-eau **quand** t équivaut à :
∃c1(c1εEV(CD)∩c1->t∩c1.msg=arrêt)∩~∃c2(c2εEV(CD)∩c1->c2->t)∩∃e1(e1εEV(E)∩e1=>c1)∩~∃o1(o1εEV(O)∩o1.msg=(prêt v couper)∩c1->o1->t)∩~∃e2(e2εEV(E)∩e2.msg>=seuil-haut∩c1->e2->t)

ETAT-POMPE = arrêtée-méthane **quand** t équivaut à :
(∃c1(c1εEV(CD)∩c1->t∩c1.msg=arrêt)∩~∃c2(c2εEV(CD)∩c1->c2->t)∩∃m1(m1εEV(M)∩m1=>c1)∩~∃o1(o1εEV(O)∩o1.msg=(prêt v couper)∩c1->o1->t)) v (∃e3(e3εEV(E)∩e3.msg>=seuil-haut∩e3->t)∩ETAT-METHANE=anormal **quand** e3∩ETAT-POMPE=(prête v arrêtée-eau) **quand** e3∩~∃o3(o3εEV(O)∩o3.msg=(prêt v couper)∩e3->o3->t))

ETAT-POMPE = prête **quand** t équivaut à :
∃o1(o1εEV(O)∩o1->t∩o1.msg=prêt)∩~∃r1(r1εEV(R)∩r1.msg=en-marche∩o1=>r1)∩~∃o2(o2εEV(O)∩o2.msg=couper∩o1->o2->t)∩~∃e1(e1εEV(E)∩e1.msg>=seuil-haut∩o1->e1->t)

Enfin, la spécification formelle du comportement de l'agent APP est établie comme suit, en adjoignant aux propriétés de causalité (A1) à (A9), des propriétés sur les valeurs des sorties (A11) à (A18), sur les ensembles de causalités possibles pour les sorties ayant plusieurs types de déclencheurs potentiels (A10), sur la non-duplication (A19) :

(˜) Avant le premier message sur M, C, A, on suppose que les variables d'état ont un état indéfini (ex: ETAT-METHANE= indéfini **quand** t équivaut à : ~∃m1(m1εEV(M)∩m1->t)).

agent APP

comportement

(A1) ∀o(o∈**EV**(O) ⟨#⟩ ∃r(r∈**EV**(R)∩o=>r))

(A2) ∀r1(r1∈**EV**(RG(E)) ⟨#⟩ ∃r2(r2∈**EV**(RG(S))∩r1=>r2))

(A3) ∀m(m∈**EV**(M)∩PA3 ⟨#⟩ ∃a(a∈**EV**(AM)∩m=>a))
avec PA3 : m.**msg**>=seuil-méth
(A4) ∀a(a∈**EV**(A)∩PA4 ⟨#⟩ ∃a1(a1∈**EV**(AA)∩a=>a1))
avec PA4 : a.**msg**>=seuil-air
(A5) ∀c(c∈**EV**(C)∩PA5 ⟨#⟩ ∃a(a∈**EV**(AC)∩c=>a))
avec PA5 : c.**msg**>=seuil-co

(A6) ∀m(m∈**EV**(M)∩PA6 #> ∃c(c∈**EV**(CD)∩m=>c∩c.**msg**=arrêt))
avec PA6 : m.**msg**>=seuil-méth∩ETAT-POMPE=en-marche **quand** m
(A7) ∀e(e∈**EV**(E)∩PA7 #> ∃c(c∈**EV**(CD)∩e=>c∩c.**msg**=arrêt))
avec PA7 : e.**msg**<seuil-bas∩ETAT-POMPE=en-marche **quand** e
(A8) ∀o(o∈**EV**(O)∩PA8 #> ∃c(c∈**EV**(CD)∩o=>c∩c.**msg**=arrêt))
avec PA8 : o.**msg**=couper∩ETAT-POMPE=en-marche **quand** o

(A9) ∀e(e∈**EV**(E)∩PA9 ⟨#⟩ ∃c(c∈**EV**(CD)∩e=>c∩c.**msg**=marche))
avec PA9 : e.**msg**>=seuil-haut∩ETAT-METHANE=normal **quand** e
∩ ETAT-POMPE=(arrêtée-eau v prête) **quand** e

(A10) ∀c(c∈**EV**(CD)∩c.**msg**=arrêt #> ∃m(m∈**EV**(M)∩PA6∩m=>c) v
∃e(e∈**EV**(E)∩PA7∩e=>c) v ∃o(o∈**EV**(O)∩PA8∩o=>c))

(A11) ∀o,∀r(o∈**EV**(O)∩r∈**EV**(R)∩o=>r∩PA11 #> r.**msg**=coupée)
avec PA11 : o.**msg**=couper v (o.**msg**=requête∩ETAT-POMPE=
coupée **quand** o)
(A12) ∀o,∀r(o∈**EV**(O)∩r∈**EV**(R)∩o=>r∩PA12 #> r.**msg**=en-marche)
avec PA12 : o.**msg**=(requête v prêt)∩ETAT-POMPE=en-marche
quand o
(A13) ∀o,∀r(o∈**EV**(O)∩r∈**EV**(R)∩o=>r∩PA13 #> r.**msg**=arrêtée-
eau)
avec PA13 : o.**msg**=requête∩ETAT-POMPE=arrêtée-eau **quand** o
(A14) ∀o,∀r(o∈**EV**(O)∩r∈**EV**(R)∩o=>r∩PA14 #> r.**msg**=arrêtée-
méthane)
avec PA14 : o.**msg**=requête∩ETAT-POMPE=arrêtée-méthane
quand o
(A15) ∀o,∀r(o∈**EV**(O)∩r∈**EV**(R)∩o=>r∩PA15 #> r.**msg**=prête)
avec PA15 : (o.**msg**=requête∩ETAT-POMPE≠ en-marche **quand**
o) v (o.**msg**=requête∩ETAT-POMPE=prête **quand** o)

(A16) ∀r1,∀r2(r1∈**EV**(RG(E))∩r2∈**EV**(RG(S))∩r1=>r2∩PA16 #>
r2.**msg**.gaz=méthane∩r2.**msg**.état= {anormal, normal})
avec PA16 : r1.**msg**=méthane∩ETAT-METHANE={anormal, normal} **quand** r1
(A17) ∀r1,∀r2(r1∈**EV**(RG(E))∩r2∈**EV**(RG(S))∩r1=>r2∩PA17 #>
r2.**msg**.gaz=co∩r2.**msg**.état= {anormal, normal})
avec PA17 : r1.**msg**=co∩ETAT-CO={anormal, normal} **quand** r1

(A18) ∀r1,∀r2(r1∈**EV**(RG(E))∩r2∈**EV**(RG(S))∩r1=>r2∩PA18 #>
r2.**msg**.gaz=air∩r2.**msg**.état= {anormal, normal})
avec PA18 : r1.**msg**=air∩ETAT-AIR={anormal, normal} **quand** r1
(A19) ∀x,∀y1,∀y2(x∈**EV**(X)∩y1,y2∈**EV**(Y)∩x=>y1∩x=>y2 #>
y1≡y2)
X et Y étant des ports quelconques de l'agent.

fin APP

2.1.3. Une approche empirique de la structuration logique :

Après avoir établi la spécification globale, le concepteur peut procéder, de manière descendante, ascendante ou mixte, à la structuration logique de son application. S'agissant de la phase logique, seuls des critères indépendants des questions de mise en oeuvre doivent être retenus : découpage fonctionnel, selon les "objets" du monde réel, selon le parallélisme de l'environnement (cf. § 2.1.4.), etc.

Des conseils très généraux et des "normes" empiriques peuvent être proposées :

- la démarche ascendante est préférable pour les parties bien connues ou partagées par de nombreux sous-systèmes alors que la démarche descendante est préférable pour analyser les parties moins bien connues [NEU82],
- un bon agent élémentaire doit être de taille "raisonnable",
- un bon raffinement génère un réseau de 3 à 6 agents (cf. les "normes" de SADT [DIC77],
- etc.

Si le cahier des charges suggère un découpage (cf. structuration fonctionnelle hiérarchisée du chapitre 1 § 3.c), cela peut constituer un guide précieux.

Exemple : un scénario (arbitraire) pour la pompe de KRAMER.

A chaque étape nous donnons uniquement la description graphique de la structure.

étape (a) : (cf. Fig. 3.19.)

Le concepteur implante l'agent APP par deux agents PPE et ATM qui correspondent aux deux fonctions principales qu'il retient du cahier des charges : la commande de la pompe (avec gestion de la variable ETAT-POMPE) et le contrôle de l'atmosphère (avec gestion des variables ETAT-METHANE, ETAT-CO, ETAT-AIR). En fonction des dépendances listées au § 2.1.2, le concepteur établit les communications internes au réseau :

- arrêt de la pompe sur alarme-méthane (le message ALM est donc diffusé vers PPE en plus de l'extérieur),
- démarrage de la pompe nécessitant la connaissance de l'ETAT-METHANE (messages REQM, REPM).

étape (b) : (cf. Fig. 3.20.)

Changeant d'optique, le concepteur décide ensuite d'exprimer par trois agents, les trois fonctions similaires de contrôle d'un gaz; comme les actions de scrutation des gaz avec émission d'une éventuelle alarme et de réponse aux requêtes concernant ce gaz ne lui paraissent pas devoir être parallélisées, il considère ces trois agents (CAIR, CCO, CMETH) comme étant des agents élémentaires.

Fig. 3.19.

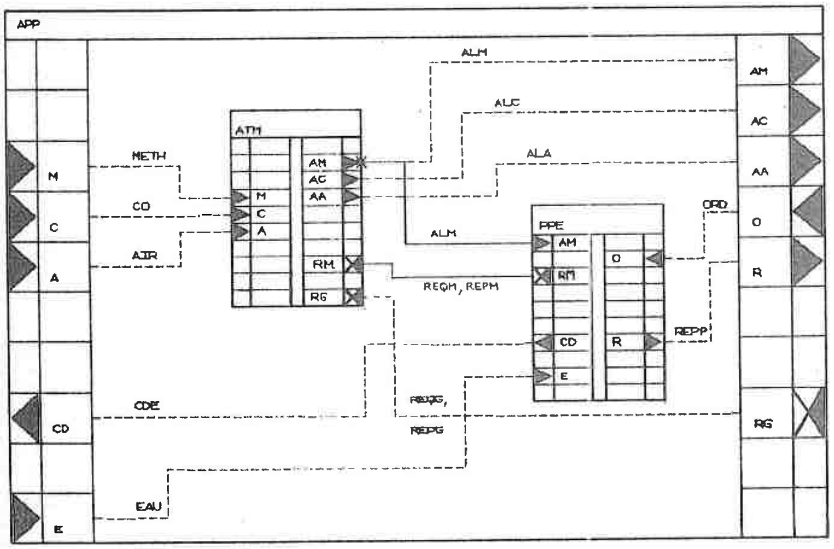
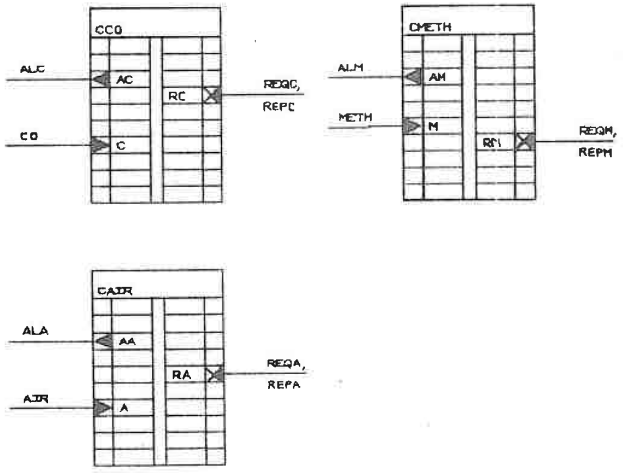


Fig. 3.20.



étape (c) : (cf. Fig. 3.21.)

Le concepteur note que l'agent ATM introduit à l'étape (a), abstrait tout ce qui concerne le contrôle des gaz. Il encapsule donc les agents CAIR, CCO et CMETH dans ATM dont ils constituent une implantation, avec un agent NOEUD chargé d' aiguiller les requêtes (selon le nom du gaz dans le message REQG) et de concentrer les réponses.

Les étapes (b) et (c) illustrent une phase de conception ascendante.

étape (d) : (cf. Fig. 3.22.)

L'examen des actions à réaliser au sein de l'agent PPE conduit le concepteur à un dernier raffinement qui sépare et parallélise le test de l'eau vis à vis des seuils mini et maxi de la commande de la pompe proprement dite (agents TEAU et POMPE comme implantation de PPE). Le message DEAU indique la détection d'un niveau d'eau hors seuils.

Aucun autre traitement n'apparaît logiquement parallélisable au concepteur : sa structure logique en agents élémentaires est donc établie.

Fig. 3.21.

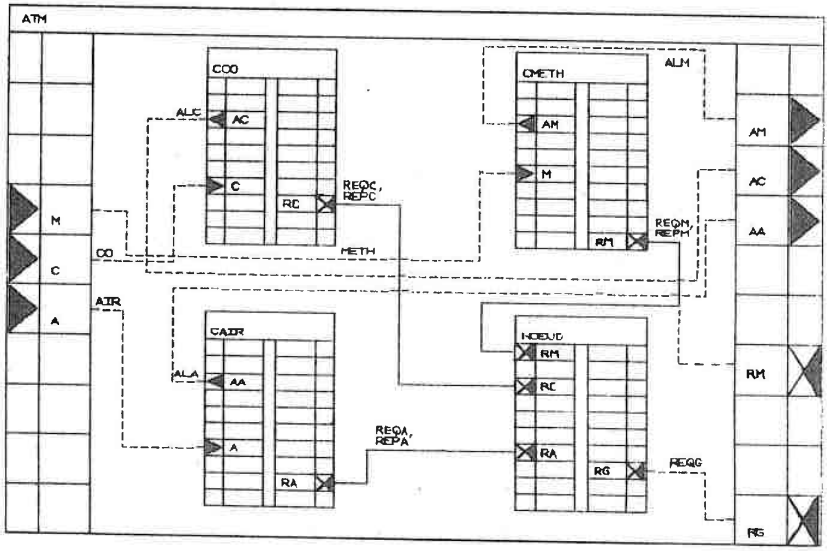
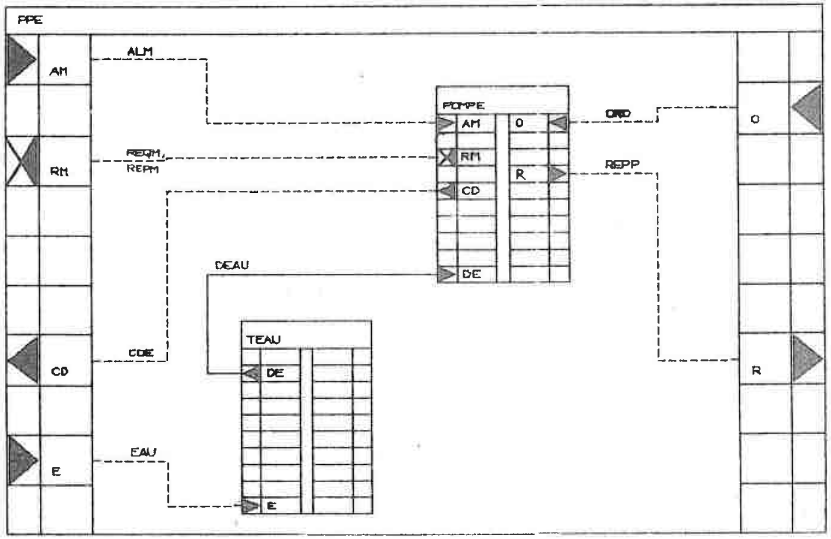


Fig. 3.22.



2.1.4. Une proposition de systématisation de la structuration logique :

Nous proposons dans ce paragraphe une méthode systématique de partitionnement(*) en agents élémentaires. Une telle assistance à la conception est intéressante dans la mesure où:

+ elle s'appuie sur un petit nombre de critères (afin que l'interprétation et la modification éventuelle de ses résultats soit aisée),
+ elle ne nécessite qu'un petit nombre de données, faciles à déterminer,
+ elle peut être automatisée.

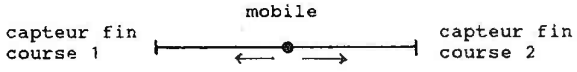
Les applications qui nous intéressent s'inscrivent dans un environnement fortement parallèle, c'est à dire que de nombreux composants du procédé évoluent en parallèle. Nous **postulons qu'une bonne structuration d'une application de ce type, doit refléter pour l'essentiel ce type de parallélisme pour des raisons de lisibilité et de facilité d'évolution** [JAC 78], [BOO86]. Nous proposons une approche systématique fondée sur la prise en compte du parallélisme de l'environnement (ou "parallélisme de situation"). Nous ne prétendons pas que ce type d'approche est supérieur aux autres (approche fonctionnelle classique, approche par les "flots de données" [GOM84] ou les "flots de contrôle" [EST86], etc.) et l'idée même d'un classement absolu nous semble peu crédible (quels critères? quelles mesures? quel échantillon représentatif pour appliquer ces mesures?). Nous constatons que cette approche présente quelques qualités et notre propos est d'apporter une aide au concepteur dans ce cadre. Selon nous, l'approche qui bénéficie du maximum d'outils finit souvent par s'imposer comme la "meilleure" (cf. le modèle individuel au travers de la méthode MERISE [TAR83] dans le domaine de la conception des systèmes d'information).

Le parallélisme de l'environnement est capturé en termes des simultanéités possibles, ou à l'inverse des exclusions temporelles, entre entrées et sorties externes du système de commande : en effet, la commande d'entités parallèles implique des entrées et sorties simultanées; inversement, l'existence d'entrées ou sorties temporellement exclusives reflète le plus souvent la commande séquentielle d'une même entité élémentaire du procédé.

Exemples :

- d'entrées en exclusion temporelle : les messages en provenance des capteurs de fin de course d'un mobile sur un rail.

Fig.3.23.



(*) le lecteur nous pardonnera ce néologisme dérivé de l'anglais "partitioning" que nous préférons à "découpage" (trop restrictif) ou "structuration" (trop général).

- de sorties en exclusion temporelle : dans le problème du mixer (problème test de l'EWICS-TC11 [LUD83]), les commandes aux robinets sont temporellement exclusives (le robinet A est ouvert jusqu'à ce que le niveau de la cuve C atteigne la valeur "moy"; le robinet A est alors fermé et B est ouvert jusqu'à ce que le niveau de la cuve C atteigne "max"; B est alors fermé et C ouvert, jusqu'à ce que la cuve C soit vide; A est alors réouvert, etc.

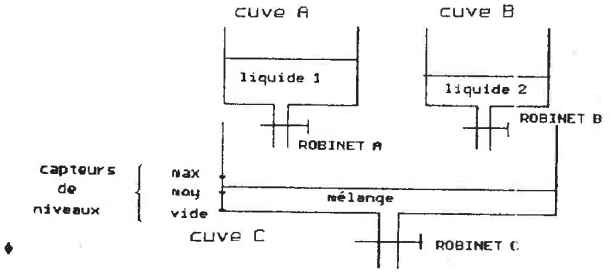


Fig.3.24.

a) description de la méthode - la première étape :
α) construction du graphe des dépendances entre E/S :
α1) on construit un graphe biparti complet entre entrées et sorties externes avec des poids 0. ex:

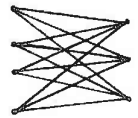
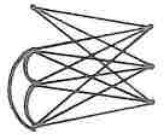


Fig.3.25.

α2) on ajoute des arcs liant entre elles, les entrées et les sorties en exclusion temporelle, avec des poids 1. ex:

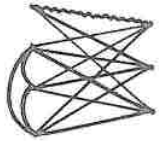


α3) on donne des poids particuliers aux arcs liant les entrées et les sorties en relation de causalité :

{ ∞ s'il s'agit d'entrées et sorties sur un même port bidirectionnel,
{ 1 sinon.

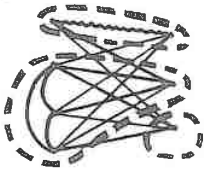
Légende : { poids 0 : ——
{ poids 1 : ▬▬
{ poids ∞ : ≈≈≈

ex:



ß) exploitation du graphe : on recherche le **minimum de composantes complètes telles que le poids total des arcs inter composantes soit minimum**. Chaque composante correspond à un agent élémentaire muni de ses entrées et sorties externes. Les arcs inter composantes entre entrées et sorties correspondent à des communications du type "déclenchement à distance". Il peut y avoir une ou plusieurs solutions.

ex:



b) justifications et commentaires :

Dans une composante complète tous les sommets sont liés entre eux; en particulier, toutes les entrées et toutes les sorties; il n'y a donc **pas possibilité de parallélisme entre entrées et sorties d'un même agent**. Notons que le nombre minimum de composantes est égal à max(n1,n2) où n1 (resp. n2) est le nombre de groupes d'entrées (resp. de groupes de sorties) en exclusion temporelle. Augmenter ce nombre reviendrait à inclure du parallélisme non imposé par l'environnement ou parallélisme de conception : ce sera bien entendu envisagé, mais ultérieurement.

La minimisation du poids total des arcs inter-composantes relève évidement de la recherche d'un **couplage faible entre entités à cohésion forte**, caractéristique généralement retenue comme critère de bon découpage modulaire. Les poids ont été fixés de manière standard pour rendre compte de la cohésion logique entre entrées et sorties (ils pourraient être affinés si nécessaire) :

- les poids 0 correspondent à des entrées et sorties sans relation de causalité mais qui peuvent néanmoins se retrouver sur un même agent en conséquence d'autres décisions ("cohérence coïncidentale" dans [SOK80]);
- les poids intermédiaires 1 privilégient le regroupement d'entrées et de sorties en relation de causalité. Ces relations permettent une évaluation approximative des couplages et de la cohésion;
- les poids ∞ correspondent aux entrées et sorties sur un même port bidirectionnel, qui ne peuvent pas être dissociées, par définition.

c) application à l'exemple de la pompe de KRAMER :

L'exemple est "pauvre" dans la mesure où les n1=6 entrées et les n2=6 sorties sont toutes avec simultanéité possible (nous étudierons un exemple plus riche au chapitre 5 § 1.3.c). Les relations de causalité du tableau de la page 48 donnent le graphe de dépendance suivant (pour des raisons de lisibilité, les arcs de poids 0 liant chaque entrée à chaque sortie ont été omis) :

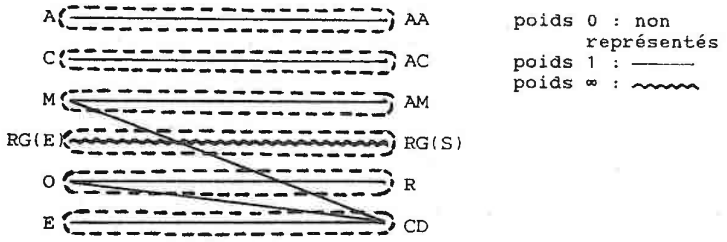


Fig. 3.26.

La solution optimum est celle indiquée sur le graphe, avec 6 agents élémentaires (A1 à A6) et deux déclenchements à distance de CD par M (type de message MCD) et de CD par O (type de message OCD). La figure 3.27. représente cette structure.

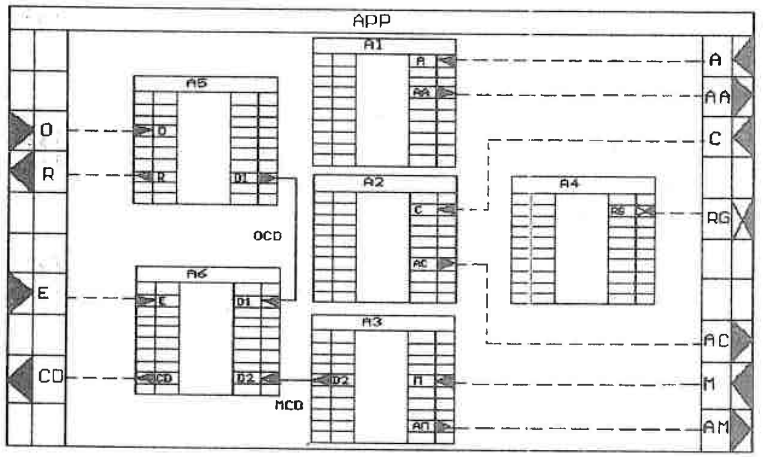


Fig. 3.27.

d) description de la méthode - la deuxième étape :

Manquent à cette structure les communications internes résultant des relations de mémorisation et de contribution (conditionnement et renseignement - cf. § 2.1.2).

On peut envisager de systématiser la mise en évidence de ces communications internes. Pour ce faire, il faut localiser les variables d'état sur les agents : pour la relation de mémorisation, il faut donner une pondération de l'importance de chaque facteur de mise à jour par rapport aux autres (quand il y en a plusieurs). Ceci permet de **localiser la variable d'état sur l'agent où sont localisés les facteurs de mise à jour les plus importants**; la pondération d'un facteur pourra par exemple s'appuyer sur les fréquences des mises à jour résultant de chaque facteur. En cas d'égalité des pondérations totales de plusieurs agents, le concepteur devra affiner ses choix.

Toute contribution non locale d'une variable d'état à une sortie se traduit par une requête/réponse. Toute mise à jour non locale se traduit par une communication. Des informations supplémentaires permettent parfois d'éviter de créer des communications redondantes.
Exemple : si une variable est impliquée à la fois dans une contribution non locale et une mise à jour (correspondant au

cas où une sortie peut être effectuée si une variable présente une certaine valeur et que cette sortie entraîne du même coup la mise à jour de la variable), une requête/réponse peut suffire à assurer les deux fonctions. ♦

e) application à la pompe de KRAMER :

Il faut saisir :
- le tableau des relations de contribution de la page 48,
- le tableau des relations de mémorisation complété comme suit :

événements caractérisant la maj	variables	pondérations
meEV(M)	ETAT-METHANE	
aeEV(A)	ETAT-AIR	
ceEV(C)	ETAT-CO	
oeEV(O) meEV(M) eeEV(E) ceEV(CD)	ETAT-POMPE	1 1 4 5

ETAT-METHANE, ETAT-AIR, ETAT-CO sont localisés sur les agents d'accueil de M, A et C. ETAT-POMPE est localisé sur l'agent d'accueil de CD et E (poids 9), de préférence aux agents d'accueil de O (poids 1) et de M (poids 1).

Il en résulte une mise à jour à distance de ETAT-POMPE sur O (type de message MAJO) et des requêtes/réponses associées aux relations de contribution r7, r8 et r9 (types de messages REQPPE/REPPE, REQM/REPM, REQA/REPA, REQC/REPC). Soit la structure logique complète de la figure 3.29.

Cette structure est légèrement différente de celle obtenue empiriquement et représentée par la Fig. 3.23. Mais les différences s'expliquent et se résorbent aisément :
- l'analyse du comportement de l'agent A5 montre au concepteur que celui-ci ne peut réaliser aucun traitement effectif et ne fait que répercuter des messages vers un autre agent : la parallélisation résultant de la prise en compte de simultanéités d'entrées et de sorties apparaît dans ce cas "factice" et les agents ainsi liés peuvent être fusionnés (A5 et A6);
- au contraire, les agents POMPE et TEAU de la structure logique empirique sont ici fusionnés : la décision de les scinder relevait en effet du parallélisme de conception qui n'est pas pris en compte par la méthode à ce niveau;
- enfin, les messages du type MCD (déclenchement de CD sur M) peuvent être confondus avec les messages du type ALM, car ils ont le même contenu et les mêmes conditions de déclenchement.

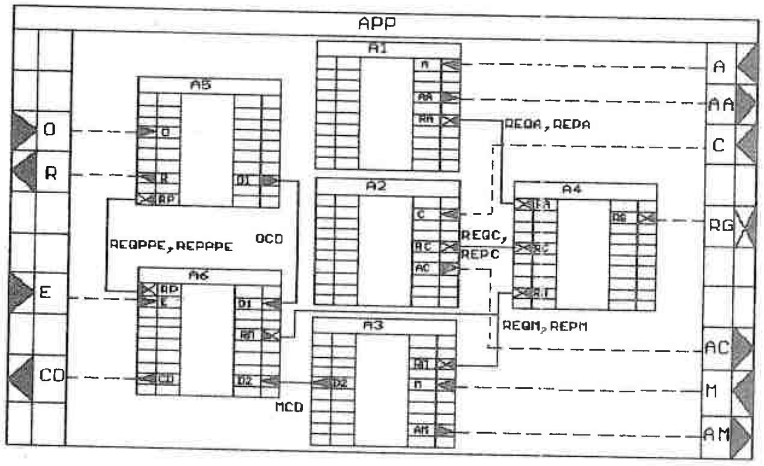


Fig. 3.28.

f) conclusion :

La méthode est efficace dès lors que le parallélisme de l'environnement est significatif. Elle ne nécessite que des informations simples, à la base de toute analyse (liste des relations de dépendance, liste des simultanéités des entrées et sorties externes). Comme toutes les méthodes de partitionnement proposées dans la littérature (ex : [SHA86], [MAR81]), elle fournit une proposition de découpage au concepteur, constituant un point de départ pour sa réflexion.

2.1.5. Validation de la structuration logique :

La décomposition de l'agent global en un réseau d'agents élémentaires, qu'elle s'inscrive dans une démarche empirique, comme au § 2.1.3., ou dans la méthode systématique du § 2.1.4., doit s'accompagner d'une validation conduite selon les principes exposés au § 1.3.2. Cette validation peut s'effectuer en une fois (réseau des agents élémentaires par rapport aux propriétés de l'agent global) ou niveau par niveau (dans le cas d'une approche empirique par raffinements successifs -cf. § 2.1.1.).

Nous présentons, pour compléter l'exemple de la pompe de KRAMER, la validation de la structure obtenue au § 2.1.3. au regard des propriétés de l'agent global APP. Pour ce faire il nous faut successivement :

(a) spécifier l'implantation ,
(b) vérifier que les variables d'état utilisées dans la spécification globale peuvent être réparties et gérées sur les agents de son implantation (dans le cas de la méthode systématique les localisations sont établies),
(c) spécifier formellement ces agents (avec utilisation des variables d'état sur les agents où elles sont localisées),
(d) construire les graphes d'enchaînement des causalités pour les propriétés qui ne se retrouvent pas à l' identique dans un des agents de son implantation,
(e) vérifier la conservation de ces propriétés en s' aidant de ces graphes, comme indiqué au § 1.3.2.

a) implantation de APP :

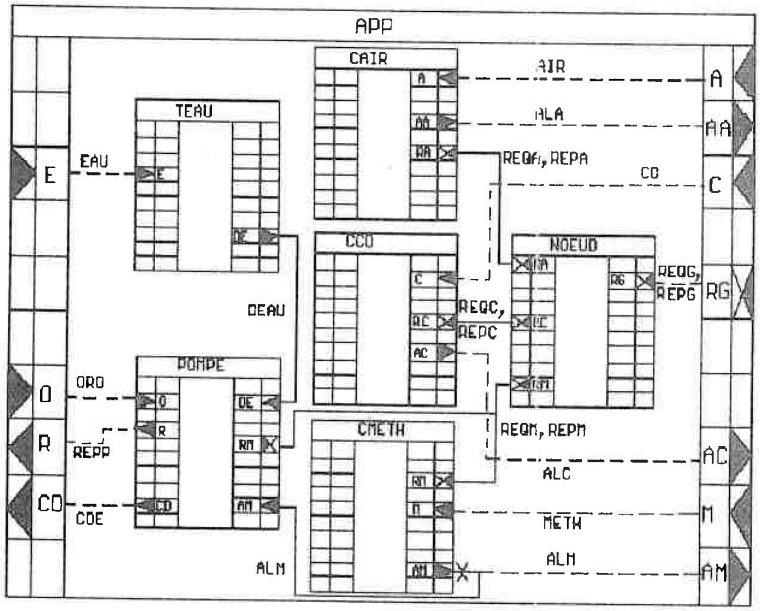
```
agent APP
  interface
    ...
  comportement   } (cf. descriptions données au § 2.1.2.)
    ...
  implantation
    agent POMPE
      interface
        E-PORT O : ORD;
        E-PORT DE : DEAU;
        E-PORT AM : ALM;
        S-PORT R : REPP;
        S-PORT CD : CDE;
        SE-PORT RM : REQM réponse REPM;
    fin POMPE;
    agent TEAU
      interface
        E-PORT E : EAU;
        S-PORT DE : DEAU;
    fin TEAU;
    agent CMETH
      interface
        E-PORT M : METH;
        S-PORT AM : ALM
        ES-PORT RM : REQM réponse REPM
    fin CMETH;
    agent CAIR
      interface
        E-PORT A : AIR;
        S-PORT AA : ALA;
        ES-PORT RA : REQA réponse REPA;
    fin CAIR;
    agent CCO
      interface
        E-PORT C : CO;
        S-PORT AC : ALC;
        ES-PORT RC : REQC réponse REPC;
    fin CCO;
    agent NOEUD
      interface
        ES-PORT RG : REQG réponse REPG;
        SE-PORT RM : REQM réponse REPM;
```



```
        SE-PORT RC : REQC réponse REPC;
        SE-PORT RA : REQA réponse REPA;
    fin NOEUD;
    liaisons
      TEAU.DE vers POMPE.DE;
      CMETH.AM vers POMPE.AM;
      POMPE.RM vers CMETH.RM;
      NOEUD.RA vers CAIR.RA;
      NOEUD.RC vers CCO.RC;
      NOEUD.RM vers CMETH.RM;
    attachements
      POMPE.CD à APP.CD;
      APP.O à POMPE.O;
      POMPE.R à APP.R;
      APP.E à TEAU.E;
      APP.M à CMETH.M;
      APP.A à CAIR.A;
      APP.C à CCO.C;
      CMETH.AM à APP.AM;
      APP.RG à NOEUD.RG;
      CAIR.AA à APP.AA;
      CCO.AC à APP.AC;
fin APP;
```

Ou encore, pour ceux qui, comme nous, préfèrent les spécifications graphiques des structures :

Fig. 3.29.



b) répartition des variables d'état :

ETAT-METHANE ne dépend que des événements sur M; elle peut être localisée sur CMETH. Le raisonnement est le même pour ETAT-AIR sur CAIR et ETAT-CO sur CCO.

Pour ETAT-POMPE, on note au vu du diagramme des transition d'états, que sa gestion nécessite la connaissance :
- des ordres de l'opérateur (port O),
- des valeurs hautes et basses de l'eau (port DE),
- des valeurs hautes du méthane (port AM),
- de l'état du méthane (port RM).

ETAT-POMPE peut donc être localisée sur l'agent POMPE.

c) spécification formelle des agents élémentaires :

Beaucoup de propriétés sont identiques à celles exprimées au niveau global; on les notera en rappelant le numéro de la propriété dans APP.

```
agent TEAU
  comportement
     (T1) ∀e(e∈EV(E)∩PT1 <#> ∃d(d∈EV(DE)∩e=>d∩d.msg=haut))
          avec PT1 : e.msg>=seuil-haut
     (T2) ∀e(e∈EV(E)∩PT2 <#> ∃d(d∈EV(DE)∩e=>d∩d.msg=bas))
          avec PT2 : e.msg<seuil-bas
     (T3) ≡ (A19)
fin TEAU;

agent POMPE
  comportement
     (P1) ≡ (A1)
     (P2) ∀a(a∈EV(AM)∩PP2 #> ∃c(c∈EV(CD)∩a=>c∩c.msg=
          arrêt))
          avec PP2 : ETAT-POMPE=en-marche quand a
     (P3) ∀d(d∈EV(DE)∩PP3 #> ∃c(c∈EV(CD)∩d=>c∩c.msg=
          arrêt))
          avec PP3 : d.msg=bas∩ETAT-POMPE=en-marche quand d
     (P4) ∀d(d∈EV(DE)∩PP4 <#> ∃r(r∈EV(RM(S))∩d=>r))
          avec PP4 : d.msg=haut∩ETAT-POMPE=(arrêtée-eau v
          prête) quand d
     (P5) ≡ (A8)
     (P6) ∀r(r∈EV(RM(E))∩PP6 <#> ∃c(c∈EV(CD)∩r=>c∩c.msg=
          marche))
          avec PP6 : r.msg=normal
     (PP7) ∀c(c∈EV(CD)∩c.msg=arrêt #> ∃a(a∈EV(AM)∩PP2∩
          a=>c) v ∃d(d∈EV(DE)∩PP3∩d=>c) v ∃o(o∈EV(O)∩PP5∩
          o=>c))
     (PP8) ≡ (A11)
     (PP9) ≡ (A12)
     (PP10) ≡ (A13)
     (PP11) ≡ (A14)
     (PP12) ≡ (A15)
     (PP13) ≡ (A19)
fin POMPE;
```

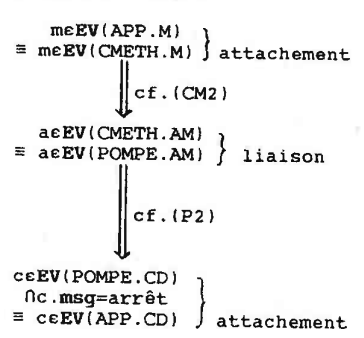


```
agent CMETH
  comportement
     (CM1) ∀i∈[1..2],∀r1(r1∈EV(RM(E,i)) <#> ∃r2(r2∈
           EV(RM(S,i))∩r1=>r2))
     (CM2) ≡ (A3)
     (CM3) ∀i∈[1..2],∀r1,∀r2(r1∈EV(RM(E,i))∩r2∈
           EV(RM(S,i))∩r1=>r2∩PCM3 #> r2.msg={anormal} )
                                              {normal }
           avec PCM3 : ETAT-METHANE = {anormal} quand r1
                                      {normal }
     (CM4) ≡ (A19)
fin CMETH;
agent CAIR
  comportement
     (CA1) ∀r1(r1∈EV(RA(E)) <#> ∃r2(r2∈EV(RA(S))∩r1=>r2))
     (CA2) ≡ (A4)
     (CA3) ∀r1,∀r2(r1∈EV(RA(E))∩r2∈EV(RA(S))∩r1=>r2∩PCA3
           #> r2.msg={anormal} )
                     {normal }
           avec PCA3 : ETAT-AIR = {anormal} quand r1
                                  {normal }
     (CA4) ≡ (A19)
fin CAIR;
agent CCO
  comportement
     (CC1) ∀r1(r1∈EV(RC(E)) <#> ∃r2(r2∈EV(RC(S))∩r1=>r2))
     (CC2) ≡ (A5)
     (CC3) ∀r1,∀r2(r1∈EV(RC(E))∩r2∈EV(RC(S))∩r1=>r2∩PCC3
           #> r2.msg={anormal} )
                     {normal }
           avec PCC3 : ETAT-CO = {anormal} quand r1
                                 {normal }
     (CC4) ≡ (A19)
fin CCO;
agent NOEUD
  comportement
     (N1) ∀r1(r1∈EV(RG(E)) <#> ∃r2(r2∈EV(RG(S))∩r1=>r2))
     (N2) ∀r1(r1∈EV(RG(E))∩PN2 <#> ∃r2(r2∈EV(RA(S))∩
          r1=>r2))
          avec PN2 : r1.msg=air
     (N3) ∀r1(r1∈EV(RG(E))∩PN3 <#> ∃r2(r2∈EV(RC(S))∩
          r1=>r2))
          avec PN3 : r1.msg=co
     (N4) ∀r1(r1∈EV(RG(E))∩PN4 <#> ∃r2(r2∈EV(RM(S))∩
          r1=>r2))
          avec PN4 : r1.msg=méthane
     (N5) ∀r1(r1∈EV(RA(E)) <#> ∃r2(r2∈EV(RG(S))∩r1=>r2))
     (N6) ∀r1(r1∈EV(RC(E)) <#> ∃r2(r2∈EV(RG(S))∩r1=>r2))
     (N7) ∀r1(r1∈EV(RM(E)) <#> ∃r2(r2∈EV(RG(S))∩r1=>r2))
     (N8) ∀r1,∀r2(r1∈EV(RA(E))∩r2∈EV(RG(S))∩r1=>r2 #>
          r2.msg.nature=air∩r2.msg.état=r1.msg)
     (N9) ∀r1,∀r2(r1∈EV(RC(E))∩r2∈EV(RG(S))∩r1=>r2 #>
          r2.msg.nature=co∩r2.msg.état=r1.msg)
     (N10) ∀r1,∀r2(r1∈EV(RM(E))∩r2∈EV(RG(S))∩r1=>r2 #>
          r2.msg.nature=méthane∩r2.msg.état=r1.msg)
     (N11) ≡ (A19)
fin NOEUD;
```

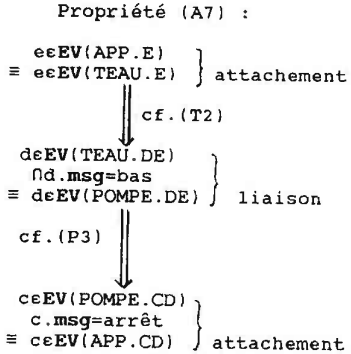
d) construction des graphes des enchaînements de causalités :

Il y a 7 propriétés de APP qui ne se retrouvent pas à l'identique dans un agent de son implantation, soient (A6), (A7), (A9), (A10), (A16), (A17) et (A18). Pour les 3 relations de déclenchement, nous construisons les graphes des enchaînements de causalités :

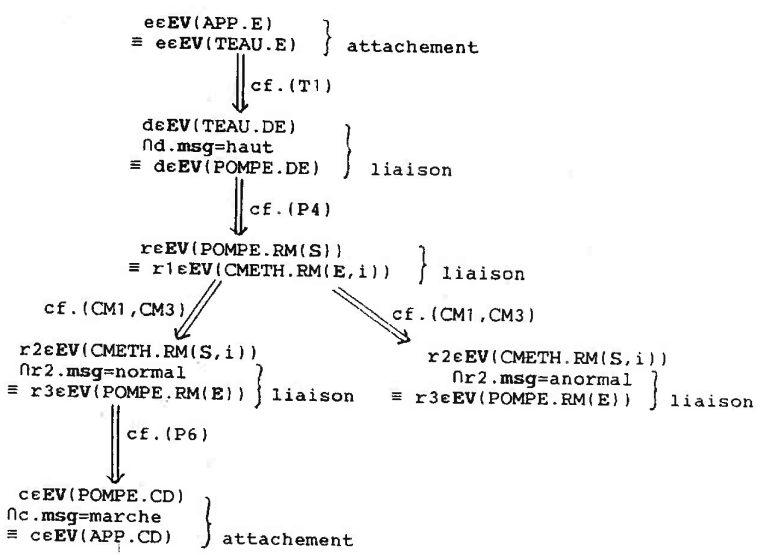
Propriété (A6) :



Propriété (A7) :



Propriété (A9) :



e) vérification de la conservation des propriétés dans l'implantation :

Pour les propriétés qui se retrouvent à l'identique dans un des agents de l'implantation il suffit de vérifier que les ports correspondants aux deux niveaux sont bien attachés; c'est le cas de : (A1),(A2),(A3),(A4),(A5) (A8),(A11),(A12), (A13),(A14),(A15).

Pour la propriété (A6) et au vu du graphe du § d, il suffit de vérifier que la conjonction des conditions de (CM2) et (P2), à savoir m.**msg**>=seuil-méth ∩ ETAT-POMPE=en-marche **quand** a, équivaut bien à la condition de (A6) ; c'est vrai, car m<->a d'après la définition du modèle "simplifié".

Pour la propriété (A7) et au vu du graphe du § d, il suffit de vérifier que la conjonction des conditions de (T2) et (P3), à savoir e.**msg**<seuil-bas ∩ d.**msg**=bas ∩ ETAT-POMPE= en-marche **quand** d, équivaut bien à la condition de (A7); c'est vrai, car :

+ d<->e,
+ d.**msg**=bas est vrai ssi e.**msg**<seuil-bas.

Pour la propriété (A9) et au vu du graphe du § d, il suffit de vérifier que la conjonction des conditions de (T1), (P4), (CM3) et (P6), à savoir e.**msg**>=seuil-haut ∩ d.**msg**=haut∩ ETAT-POMPE=(arrêtée-eau v prête) **quand** d ∩ ETAT-METHANE=normal **quand** r1∩r2.**msg**=normal, équivaut bien à la condition de (A9); c'est vrai, car :

+ c<->d<->r1<->r2<->r3,
+ d.**msg**=haut est vrai ssi e.**msg**>=seuil-haut,
+ r2.**msg**=normal est vrai ssi ETAT-METHANE=normal quand r1.

La propriété (A10) est vérifiée car les causalités possibles de ceEV(CD)∩c.**msg**=arrêt sont conservées (cf. conservation des propriétés (A6),(A7),(A8)) et qu'aucune causalité nouvelle n'apparaît dans l'agent POMPE.

La propriété (A16) est vérifiée car r1.**msg**=méthane implique qu'une requête est adressée de NOEUD à CMETH (cf. (N4)) et ETAT-METHANE = {anormal, normal} implique qu'une réponse {anormal, normal} est envoyée de CMETH à NOEUD (cf. (CM3)) et répercutée sur RG(S) (cf.(N8)).

Même démonstration pour les propriétés (A17), (A18).

Enfin, la propriété (A19) de non-duplication en sortie est conservée car elle est vraie dans tous les agents du réseau.

2.2. EMPLOI DE L'OUTIL LORS DE LA PHASE DE STRUCTURATION ORGANIQUE :

2.2.1. Règles de structuration en modules et validation de la cohérence :

Le passage de la structure logique en termes d'agents à la structure organique en termes de modules s'effectue le plus souvent par regroupements d'agents élémentaires en un même module pour bénéficier de la localisation sur un même site et plus exceptionnellement par division d'agents élémentaires en plusieurs modules. La réflexion s'appuie sur :

- le couplage entre agents : il peut être de nature temporelle ou informationnelle. Un fort couplage informationnel signifie un échange important de messages entre agents. Le regroupement au sein d'un module autorisera le partage éventuel d'informations par les futurs processus. Un fort couplage temporel signifie une contrainte de temps stricte sur un type de communication; il apparaît comme une conséquence des contraintes de temps à l'interface (temps de réponse).

- la "proximité logique" des agents (participation à un même sous-système logique) : il est souhaitable en effet que les modules conservent une certaine unité.

- les contraintes d'implantation : la localisation de telle fonction ou de telle ressource sur un site peut être imposée; le découpage en modules doit en tenir compte; ce cas peut conduire à l'éclatement d'un agent élémentaire.

- les contraintes de sûreté : elle peuvent se traduire par l'introduction de nouvelles entités (ex: des redondances fonctionnelles implantées sur des sites distincts) ou de nouvelles communications (ex: des acquittements dans le cadre de protocoles de communication).

Il ne nous apparaît pas possible à ce niveau, en raison de la multiplicité des facteurs spécifiques, de proposer une méthode de découpage systématique. Nous donnons cependant quelques indications pour apprécier le couplage entre agents :

- on peut essayer de quantifier les fréquences des communications internes à partir des fréquences des communications externes et des probabilités estimées de déclenchement;

- on peut rechercher les couplages temporels les plus élevés au vu des contraintes de temps de réponse à l'interface;

- la force des couplages devant conduire à un regroupement est difficile à quantifier de manière générale; on peut prendre en compte les couplages les plus forts en évitant de dépasser, par ces regroupements, une taille "raisonnable" pour les modules : la seule mesure objective de complexité des modules à ce niveau réside dans **la taille de l'interface des modules, c'est à dire le nombre des ports**, qu'il semble préférable de limiter approximativement à la douzaine.

La structuration étant établie, il est possible de donner la spécification formelle du comportement des modules et d'en valider la cohérence par rapport aux spécifications initiales.

Exemple : structuration organique de la pompe de KRAMER.

L'exemple n'est guère significatif dans la mesure où notre liberté est totale : aucune contrainte d'implantation ni d'élément quantitatif ne sont connus.

Au vu de l'énoncé informel du problème, on peut supposer un couplage informationnel relativement élevé entre l'agent qui surveille le niveau d'eau et celui qui gère la pompe et un couplage temporel notable sur l'alarme méthane entre l'agent qui contrôle le méthane et celui qui contrôle la pompe. On peut donc proposer de regrouper ces trois agents (c'est à dire tous ceux qui contribuent directement au fonctionnement de la pompe) sur un premier module (FP), le second accueillant les autres activités de surveillance des gaz (SG).

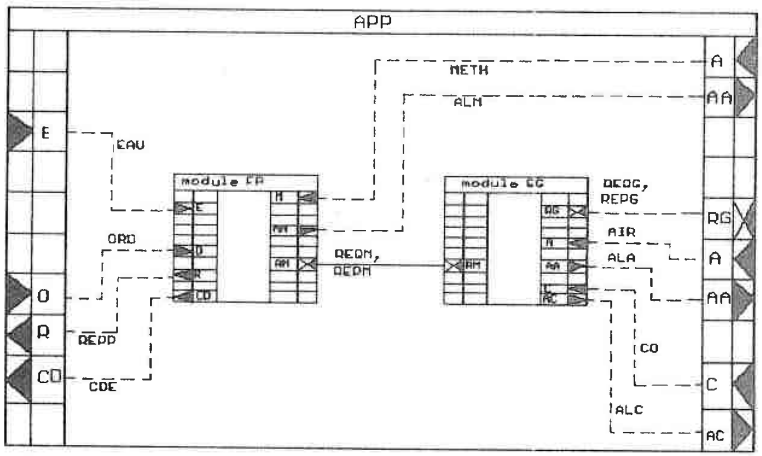


Fig. 3.30.

La solution retenue par KRAMER est différente : elle correspond à notre premier découpage logique de la figure 3.19. en un module de gestion de la pompe et un module de gestion de l'environnement atmosphérique (méthane,air,co); l'alarme sur le méthane s'y fait donc comme une communication inter modules (l'évolution des teneurs devant être lente, on peut supposer qu'il suffit de prendre une petite marge de sécurité par rapport au seuil dangereux pour que les délais impliqués par les communications deviennent négligeables).
♦

2.2.2. Autres validations de propriétés à l'aide du modèle "complet" :

Le modèle "complet" de CHEN et YEH permet une description beaucoup plus "réaliste" des systèmes répartis que le modèle "simplifié". On y prend en compte, d'une certaine manière, les durées des traitements et des communications. Seuls certains événements peuvent être ordonnés, sur la base des horloges locales aux sites ou comme conséquence des communications; les autres événements sont concurrents. Ces caractéristiques permettent d'exprimer et de valider des propriétés plus fines, et souvent liées à des choix d'implantation, comme par exemple :

- l'exclusion mutuelle entre deux événements quelconques a et b, qui s'exprime par :
  (a->b) v (b->a)
  (dans le modèle "simplifié" tous les événements étaient exclusifs ou confondus, par construction);

- la conservation de l'ordre des messages dans une relation de causalité entre entrée et sortie qui s'exprime par :
  ∀x1,∀x2,∀y1,∀y2(x1,x2∈**EV**(X)∩y1,y2∈**EV**(Y)∩x1=>y1∩x2 =>y2 #) (x1->x2∩y1->y2) v (x2->x1∩y2->y1) v (x1≡x2∩y1≡y2)
  (dans le modèle "simplifié" x1 et y1, x2 et y2 étaient confondus, par définition);

- plus généralement des propriétés sur les chaînes d'événements en relation de causalité, comme nous le verrons plus loin dans la validation d'un protocole de communication entre modules (ils étaient tous confondus dans le modèle "simplifié").

Ces validations sont beaucoup plus complexes que les validations de cohérence dans le contexte du modèle "simplifié". Nous les illustrons par l'étude d'un cas complet qui met en évidence la puissance du modèle.

a) Présentation du cas :

Il est tiré de [GOF84] et concerne l'automatisation de la reconnaissance des véhicules en tête de lignes de production dans l'industrie automobile (montage, ferrage,

sellerie) et la commande de ces lignes en fonction du véhicule reconnu.

Les fonctionnalités qui y sont décrites sont très restreintes ce qui permet de traiter complètement l'exemple. Comme il s'agit de l'extension d'un système existant, beaucoup de contraintes d'implantation doivent être respectées.

En entrée de chaque ligne de production, on trouve un dispositif d'identification automatique du véhicule (lecture par laser d'un code à barres situé sur le véhicule). En cas de lecture incorrecte, l'information doit être saisie manuellement par un opérateur depuis un terminal d'atelier en entrée de la ligne. En cas d'anomalie (identification manuelle incorrecte, véhicule non prévu dans les bases de données, passage impossible à enregistrer) un message donnant l'ordre de retirer le véhicule de la ligne apparaît sur le terminal, pour l'opérateur. A partir de l'identification du véhicule, il y a diffusion "en bord de ligne" des informations associées au véhicule et nécessaires à sa fabrication; ces informations sont trouvées dans des bases de données; elles sont destinées aux robots, automates programmables, etc. (forme codée) et aux terminaux d'affichage (forme textuelle). Le système doit permettre d'éditer des états sur le parc de véhicules traité sur la ligne; les passages en tête de ligne sont donc enregistrés.

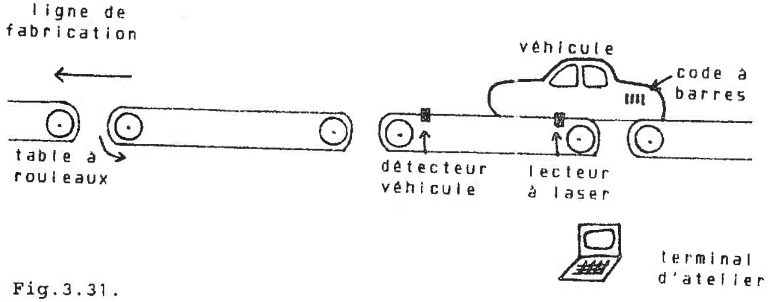


Fig.3.31.

La configuration informatique est définie au départ et constituée par un réseau local reliant des matériels sur chaque ligne de fabrication et un mini-ordinateur central. Pour chaque ligne, on trouve :
- un micro de saisie, auquel sont connectés le détecteur, le lecteur à laser et le terminal d'atelier en tête de ligne,
- un micro de gestion de la ligne, doté d'un disque,
- un micro de diffusion, situé dans l'atelier et qui met en forme et diffuse les informations aux équipements.

Les informations à diffuser, appelées "appros" (approvisionnements), sont situées soit dans une base de données locale à la ligne (sur le micro de gestion de la ligne) soit dans la base de données centrale (sur le mini

central), selon que la destination du véhicule était connue au départ ou non. Les passages sont également enregistrés sur le mini central.

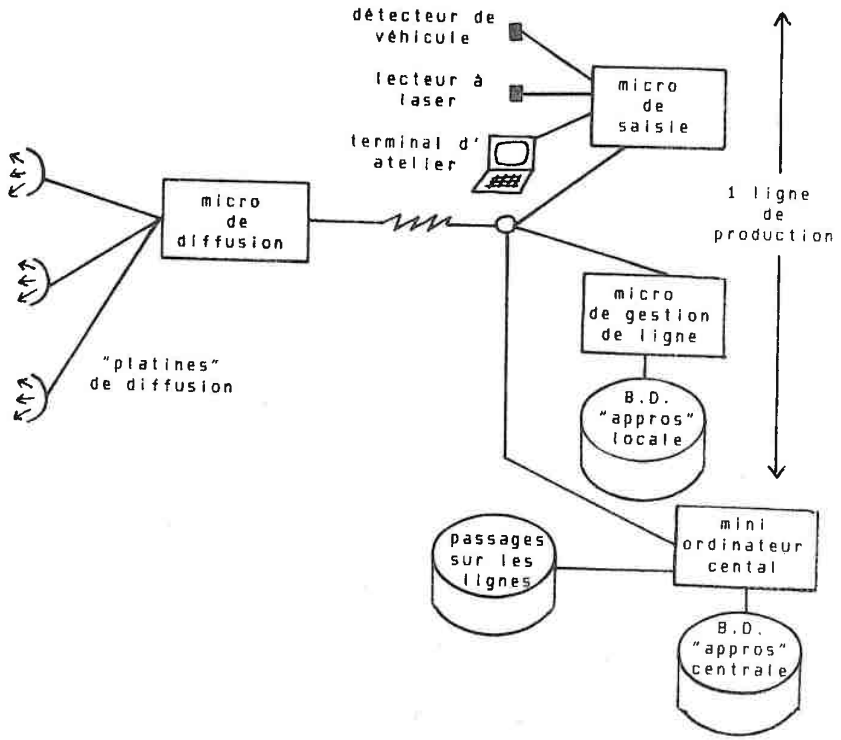


Fig. 3.32.

Sur les différentes lignes, les matériels sont hétérogènes. Par contre les fonctionnalités sont identiques et indépendantes. Nous spécifions donc le suivi et la commande d'une seule ligne de production.

b) Spécification de l'agent global :

A partir de ce cahier des charges, le premier travail consiste à définir les entrées et sorties externes et à les attacher à un agent unique S, figurant l'application toute entière :

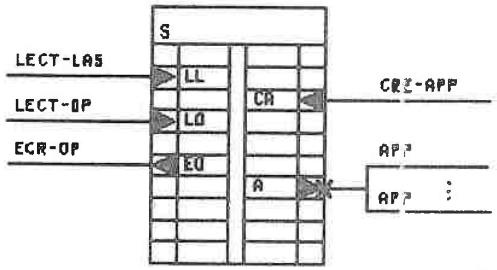


Fig. 3.33.

Dans cette figure :
- LECT-LAS contient l'identification du véhicule qui passe en tête de ligne, lue par laser,
- LECT-OP contient l'identification du véhicule, introduite par l'opérateur,
- ECR-OP contient un message à l'opérateur ("manuel" pour une demande de saisie manuelle de l'identification ou "retirer" pour une demande de retrait de véhicule de la ligne),
- CRE-APP contient un "appro" qui doit être rangé dans une base de données,
- APP est le message à diffuser vers les équipements de la ligne.

Remarque : la mise en marche du laser sur la détection du véhicule a été abstraite.

Les relations de dépendance entre ces entrées, ces sorties et la "variable d'état" BASE-APPROS, sont les suivantes :

Relations de déclenchement :

événements déclencheurs	événements déclenchés
l1εEV(LL), l2εEV(LO)	aεEV(A)
l1εEV(LL), l2εEV(LO)	eεEV(EO)

Relations de mémorisation :

événements	variables
cεEV(CA)	BASE-APPROS

Relations de contribution :

variables	événements
BASE-APPROS	e∈EV(EO)
BASE-APPROS	a∈EV(A)

De manière plus précise, les fonctionnalités peuvent être spécifiées comme suit :

```
agent S
  interface
     E-PORT LL : LECT-LAS;
     E-PORT LO : LECT-OP;
     E-PORT CA : CRE-APP;
     S-PORT EO : ECR-OPP;
     S-PORT A : APP;
  comportement
     (S1) ∀l(l∈EV(LL)∩PS1 <#> ∃e(e∈EV(EO)∩l=>e∩e.msg=
          "manuel"))
          avec PS1 : ~normal-ident(l.msg)
     (S2) ∀l(l∈EV(LL)∩PS2 #> ∃e(e∈EV(EO)∩l=>e∩e.msg=
          "retirer"))
          avec PS2 : normal-ident(l.msg)∩[~∃c(c∈EV(CA)∩
          dans-base(c.msg,l.msg)) v ~enregistr-pass(l.msg)]
     (S3) ∀l(l∈EV(LO)∩PS3 #> ∃e(e∈EV(EO)∩l=>e∩e.msg=
          "retirer"))
          avec PS3 : ~normal-ident(l.msg) v (normal-ident
          (l.msg)∩[~∃c(c∈EV(CA)∩c->l∩dans-base(c.msg,l.msg))
          v ~enregistr-pass(l.msg)])
     (S4) ∀l(l∈EV(LO)∩PS4 #> ∃a(a∈EV(A)∩l=>a))
          avec PS4 : normal-ident(l.msg)∩∃c(c∈EV(CA)∩c->l∩
          dans-base(c.msg,l.msg))∩enregistr-pass(l.msg)
     (S5) ∀l(l∈EV(LL)∩PS5 #> ∃a(a∈EV(A)∩l=>a))
          avec PS5 : normal-ident(l.msg)∩∃c(c∈EV(CA)∩c->l∩
          dans-base(c.msg,l.msg))∩enregistr-pass(l.msg)
     (S6) ∀a(a∈EV(A) #> ∃l1(l1∈EV(LL)∩PS5∩l1=>a) v ∃l2
          (l2∈EV(LO)∩PS4∩l2=>a))
     (S7) ∀e(e∈EV(EO)∩e.msg="retirer" #> ∃l1(l1∈EV(LL)∩PS2
          ∩l1=>e) v ∃l2(l2∈EV(LO)∩PS3∩l2=>e))

     + normal-ident(x) est vrai ssi le message en paramètre x,
       du type LECT-LAS ou LECT-OP, est conforme et faux
       sinon;
     + dans-base(x,y) est vrai ssi il existe un appro créé par
       un message x du type CRE-APP qui corresponde au message
       y du type LECT-LAS ou LECT-OP et faux sinon;
     + enregistr-pass(x) est vrai ssi le passage associé au
       message x du type LECT-LAS ou LECT-OP a pu
       être correctement enregistré et faux sinon.
fin S;
```


c) La structuration logique :

Le graphe des dépendances rend compte des exclusions temporelles entre LL et LO en entrée, EO et A en sortie et des causalités :

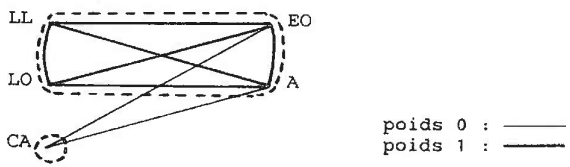


Fig. 3.34.

Le découpage en deux agents est évident : A1 regroupe LL, LO, EO, A et A2 accueille CA. Au vu de la relation de mémorisation, BASE-APPROS est associée à l'agent d'accueil de CA (A2). Au vu de la relation de contribution, une requête/réponse pour l'accès à BASE-APPROS sur EO ou A est introduite (messages REQ-APP, REP-APP).

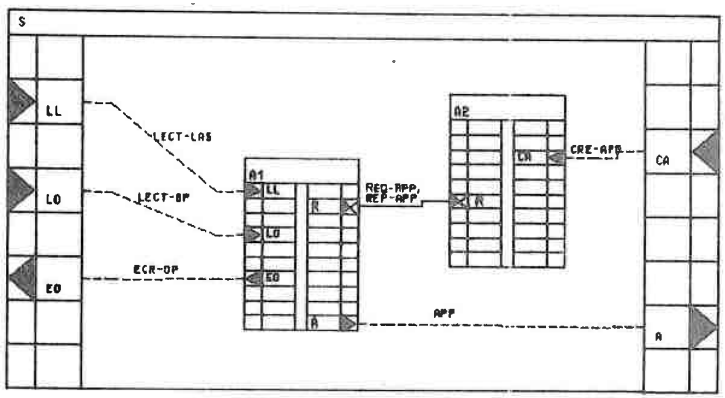


Fig. 3.35.

La simplicité de la structure logique nous incite à établir d'abord la structure organique et à procéder directement à la validation de cette dernière vis à vis de la spécification globale (agent S).

d) La structuration organique :

Pour tenir compte des contraintes d'implantation, les agents A1 et A2 doivent être scindés en plusieurs modules:
- A1 en 4 modules :

. M1 : gestion de la saisie sur le micro de saisie,
. M2 : gestion du fichier des passages sur le mini,
. M3 : contrôle des appros sur le micro de la ligne,
. M4 : diffusion des appros sur le micro de diffusion,

- A2 en 2 modules :
  . M5 : gestion centrale des appros sur le mini,
  . M6 : gestion locale des appros sur le micro de la ligne.

Cet éclatement induit de nouvelles communications internes :

- déclenchement du contrôle des appros sur M3 après la saisie sur M1 (messages D-CONT),
- appel à la base de données des appros sur M5 et M6 lors du contrôle des appros sur M3 (messages REQ-APP,REP-APP),
- appel à la gestion du fichier des passages sur M2 lors du contrôle des appros sur M3 (messages REQ-PASS,REP-PASS),
- déclenchement de ECR-OPP sur M1 en cas d'anomalie du contrôle des appros sur M3 (messages D-ECR),
- déclenchement de la diffusion sur M4 après contrôle sur M3 (messages D-DIF),
- transfert des appros reçus de la gestion centrale sur M5 vers la gestion locale sur M6 quand la destination des véhicules est connue (messages TR-APP).

Les figures 3.36. et 3.37. donnent les décompositions des agents A1 et A2.

Fig. 3.36.

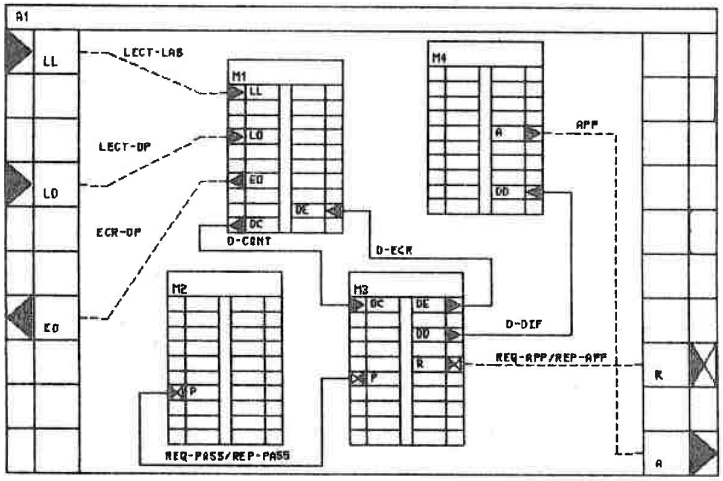
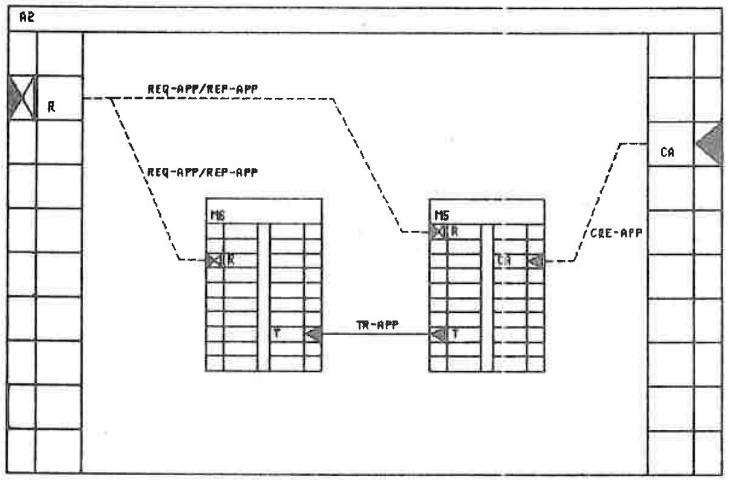


Fig. 3.37.



Les spécifications des modules et la validation de cohérence vis à vis de la spécification globale sont données à l'annexe I. Elles n'apportent rien de nouveau par rapport aux exemples déjà traités si ce n'est une complexité plus grande des graphes d'enchaînement des causalités et l'illustration de l'intérêt de leur usage pour conduire la validation.

e) Validation complémentaire :

Une contrainte d'implantation supplémentaire indique que la liaison entre le micro de la ligne et le micro de diffusion n'est pas sûre de fonctionnement et qu'un protocole doit être intégré au niveau de l'application pour pallier cet inconvénient [GOF84] : l'émetteur M3 envoie répétitivement chaque message vers le destinataire M4 jusqu'à ce qu'il reçoive un acquittement en provenance du destinataire; pour éviter les duplications, un numéro d'ordre entier est associé à chaque message : le destinataire traite un message seulement si son numéro d'ordre n'est jamais apparu auparavant; mais il acquitte tous les messages en renvoyant leur numéro d'ordre; l'émetteur ne change de message que lorsque le précédent a été acquitté.

Une entité intermédiaire F nous permet de spécifier les propriétés de la liaison fiable que nous cherchons à établir entre M3 et M4 :

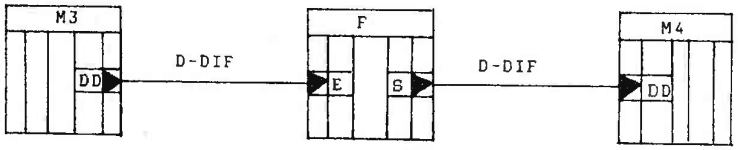
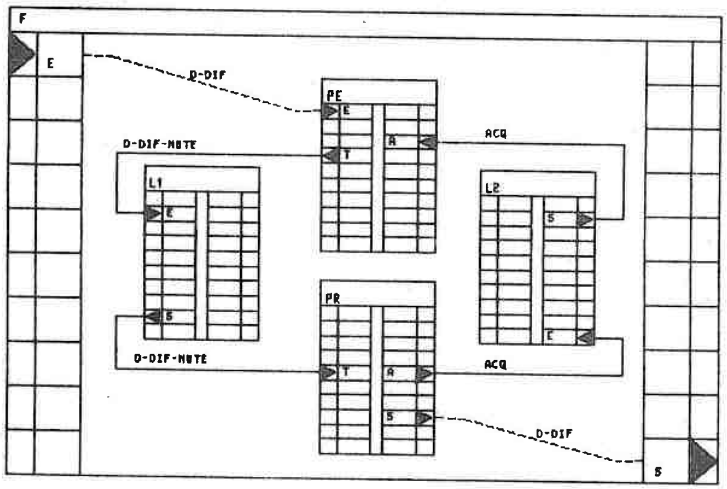


Fig.3.38.

```
entité F
  interface
     E-PORT E : D-DIF;
     S-PORT S : D-DIF;
  comportement
     (F1) ∀e(e∈EV(E) #> ∃s(s∈EV(S)∩e=>s))
          (pas de perte de message)
     (F2) ∀s(s∈EV(S) #> ∃e(e∈EV(E)∩e=>s))
          (pas de message spontané)
     (F3) ∀e,∀s1,∀s2(e∈EV(E)∩s1,s2∈EV(S)∩e=>s1∩e=>s2 #>
          s1≡s2)
          (pas de duplication de message)
     (F4) ∀e,∀s(e∈EV(E)∩s∈EV(S)∩e=>s #> s.msg=e.msg)
          (conservation du contenu)
     (F5) ∀e1,∀e2,∀s1,∀s2(e1,e2∈EV(E)∩s1,s2∈EV(S)∩
          e1=>s1∩e2=>s2 #> (e1->e2∩s1->s2) v (e2->e1∩s2->s1) v
          (e1≡e2∩s1≡s2))
          (conservation de l'ordre)
fin F;
```

La réalisation de F met en jeu quatre entités PE, PR, L1, L2 qui représentent respectivement : le protocole d'émission (il sera inclus dans le module M3), le protocole de réception (il sera inclus dans le module M4) et les deux lignes non fiables de M3 vers M4 (qui achemine les messages) et de M4 vers M3 (qui achemine les acquittements). Soit la figure 3.39. et la spécification d'implantation qui suit.

Fig. 3.39.

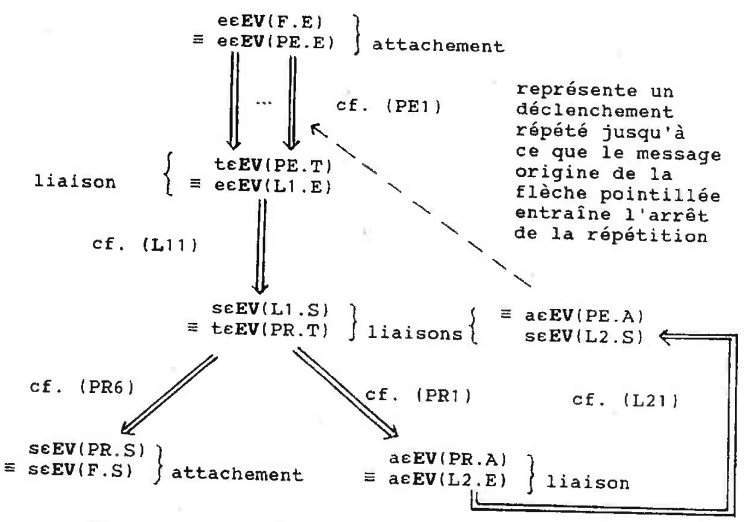



```
implantation
  entité PE
    interface
      E-PORT E : D-DIF;
      E-PORT A : ACQ;
      S-PORT T : D-DIF-NOTE;
    comportement
      (PE1) ∀e(e∈EV(E) #> ∃t(t∈EV(T)∩e=>t∩[∃a(a∈EV(A)∩
            a.msg=ord(e)) v {∀t1(t1∈EV(T)∩e=>t1) #> ∃t2(t2
            ∈EV(T)∩e=>t2∩t1->t2)}]))
             (on a soit un acquittement a qui correspond à e ou
             alors une émission sur T répétée à l'infini).
      (PE2) ∀e,∀t(e∈EV(E)∩t∈EV(T)∩e=>t #> t.msg.no=ord(e)∩
            t.msg.app =e.msg)
      (PE3) ∀e,∀t(e∈EV(E)∩t∈EV(T)∩e=>t #> ∀k(k∈ℕ∩
            k<ord(e) #> ∃a(a∈EV(A)∩a.msg=k∩a->t)))
             (avant une émission tous les acquittements précé-
            dents doivent avoir été reçus)
      (PE4) ∀t(t∈EV(T) #> ∃e(e∈EV(E)∩e=>t))
  fin PE;
  entité PR
    interface
      E-PORT T : D-DIF-NOTE;
      S-PORT A : ACQ;
      S-PORT S : D-DIF;
    comportement
      (PR1) ∀t(t∈EV(T) #> ∃a(a∈EV(A)∩t=>a))
      (PR2) ∀t,∀a(t∈EV(T)∩a∈EV(A)∩t=>a #> a.msg=t.msg.no)
      (PR3) ∀a(a∈EV(A) #> ∃t(t∈EV(T)∩t=>a))
      (PR4) ∀t1,∀t2,∀a1,∀a2(t1,t2∈EV(T)∩a1,a2∈EV(A)
            t1=>a1∩t2=>a2 #> (t1->t2∩a1->a2) v (t2->t1∩a2->a1)
            v (t1≡t2∩a1≡a2))
      (PR5) ∀t,∀a1,∀a2(t∈EV(T)∩a1,a2∈EV(A)∩t=>a1∩t=>a2
            #> a1≡a2)
      (PR6) ∀t(t∈EV(T)∩~∃t1(t1∈EV(T)∩t1->t∩t1.msg.no=
            t.msg.no) #> ∃s(s∈EV(S)∩t=>s))
      (PR7) ∀t,∀s(t∈EV(T)∩s∈EV(S)∩t=>s #> s.msg=t.msg.app)
      (PR8) ∀t1,∀t2,∀s1,∀s2(t1,t2∈EV(T)∩s1,s2∈EV(S)∩
            t1=>s1∩t2=>s2 #> (t1->t2∩s1->s2) v (t2->t1∩s2->s1)
            v (t1≡t2∩s1≡s2))
      (PR9) ∀t,∀s1,∀s2(t∈EV(T)∩s1,s2∈EV(S)∩t=>s1∩t=>s2
            #> s1≡s2)
      (PR10) ∀s(s∈EV(S) #> ∃t(t∈EV(T)∩t=>s))
  fin PR;
  entité L1
    interface
      E-PORT E : D-DIF-NOTE;
      S-PORT S : D-DIF-NOTE;
    comportement
      (L11) ∀e1(e1∈EV(E) #> [{∀e2(e2∈EV(E)∩e1->e2∩e2.msg=
            e1.msg) #> ∃e3(e3∈EV(E)∩e2->e3∩e3.msg=e1.msg)}
            #> ∃e,∃s(e∈EV(E)∩s∈EV(S)∩e.msg=e1.msg∩e=>s)])
             (si une infinité de messages identiques entrent
            dans L1 au moins un en sort)
      (L12) ∀s(s∈EV(S) #> ∃e(e∈EV(E)∩e=>s))
      (L13) ∀e,∀s(e∈EV(E)∩s∈EV(S)∩e=>s #> s.msg=e.msg)
  fin L1;
```

```
entité L2
  interface
    E-PORT E : ACQ;
    S-PORT S : ACQ;
  comportement
    (L21) ≡ (L11)
    (L22) ≡ (L12)
    (L23) ≡ (L13)
fin L2;
liaisons
    PE.T vers L1.T;
    L1.S vers PR.T;
    PR.A vers L2.E
    L2.S vers PE.A;
attachements
    F.E à PE.E;
    PR.S à F.S;
```

Le graphe des enchaînements de causalités est le suivant :

Fig. 3.40.



Il faut tout d'abord prouver que le réseau R, qui implante F, est sans blocage, c'est à dire que la "condition d'arrêt" de (PE1) (∃a(aεEV(A)∩a.msg=ord(e)) devient toujours vraie.

Nous le démontrons en deux étapes :

(1) si un nombre non borné de messages identiques sont émis sur PE.T, alors, un nombre non borné d'entre eux arrivent sur PR.T.

En effet, si un nombre non borné de messages identiques sont émis sur PE.T :

+ d'après (L11), au moins un message arrive sur PR.T,
+ après cet événement, un nombre non borné de messages identiques sont encore émis sur PE.T,
+ d'après (L11), un second message au moins arrive sur PR.T, etc. C.Q.F.D.

(2) tout message émis sur PE.T reçoit sur PE.A un acquittement avec son numéro d'ordre.

En effet, supposons qu'il n'y ait pas d'acquittement :

+ d'après (PE1), un nombre non borné de messages de même numéro sont émis sur PE.T,
+ d'après (1), un nombre non borné d'entre eux arrivent sur PR.T,
+ tout message qui arrive sur PR.T est acquitté sur PR.A, avec son numéro d'ordre, d'après (PR1 et PR2),
+ d'après (L21), au moins un de ces acquittements arrive sur PE.A avec ce numéro d'ordre. C.Q.F.D.

La vérification de la conservation dans R des propriétés (F1) à (F5) peut se conduire comme suit :

Pour (F1) ∀e(eεEV(F.E) #> ∃s(sεEV(F.S)∩e=>s)) :

+ tout eεEV(F.E) est un eεEV(PE.E), car les deux ports sont attachés,
+ soit τ = {tεEV(PR.T)|e=>t}; cet ensemble est non vide et borné d'après (1) et (2).
+ soit t* ε τ, le plus ancien élément de τ :
  ∀t'εEV(PR.T)∩t'->t* #> t'.msg.no ≠ t*.msg.no, car : e=>t* ∩ ~(e=>t')
  Donc : ∃s(sεEV(PR.S)∩t* =>s), d'après (PR6)
+ tout sεEV(PR.S) est un sεEV(F.S), car les deux ports sont attachés C.Q.F.D.

Pour (F2) ∀s(sεEV(F.S) #> ∃e(eεEV(F.E)∩e=>s)) :

+ tout sεEV(F.S) est un sεEV(PR.S), car les deux ports sont attachés,
+ ∀s(sεEV(PR.S) #> ∃t(tεEV(PR.T)∩t=>s)), d'après (PR10),
+ tout tεEV(PR.T) est un tεEV(L1.S), car les deux ports sont liés,
+ ∀s(sεEV(L1.S) #> ∃e(eεEV(L1.E)∩e=>s)), d'après (L12),
+ tout eεEV(L1.E) est un tεEV(PE.T), car les deux ports sont liés,
+ ∀t(tεEV(PE.T) #> ∃e(eεEV(PE.E)∩e=>t)), d'après (PE4),
+ tout eεEV(PE.E) est un eεEV(F.E), car les deux ports sont attachés C.Q.F.D.

Pour (F3) ∀e,∀s1,∀s2(eεEV(F.E)∩s1,s2εEV(F.S)∩e=>s1∩e=>s2 #> s1≡s2) :

+ tout eεEV(F.E) est un eεEV(PE.E),
  tout s1εEV(F.S) est un s1εEV(PR.S),
  tout s2εEV(F.S) est un s2εEV(PR.S),
  car les ports sont attachés,
+ eεEV(PE.E)=>s1εEV(PR.S) #> ∃t1(t1εEV(PR.T)∩e=>t1∩

t1=>s1) (seul enchaînement de causalités possibles) ∩ t1.**msg.no=ord**(e), d'après (PE2) et (L13),

+ e∈**EV**(PE.E)=>s2∈**EV**(PR.S) #> ∃t2(t2∈**EV**(PR.T)∩e=>t2∩ t2=>s2) (seul enchaînement de causalités possibles) ∩ t2.**msg.no=ord**(e), d'après (PE2) et (L13),

+ d'après (PR8) on a : (t1->t2∩s1->s2) v (t2->t1∩s2->s1) v (t1≡t2∩s1≡s2); les 2 premiers cas contredisent (PR6) pour le deuxième message s. Donc : s1≡s2 C.Q.F.D.

Pour (F4) ∀e,∀s(e∈**EV**(F.E)∩s∈**EV**(F.S)∩e=>s #> s.**msg**=e.**msg**) :

+ propriété triviale d'après (PE2), (L13), (PR7) et les attachements.

Pour (F5) conservation de l'ordre des messages sur F.E et F.S:

+ en raison des attachements, tout e∈**EV**(F.E) est un événement e∈**EV**(PE.E) et tout événement s∈**EV**(PR.S) est un événement s∈**EV**(F.S);

+ e1∈**EV**(PE.E)=>s1∈**EV**(PR.S) #> ∃t11(t11∈**EV**(PE.T)∩e1=>t11 ∩∃t12∈**EV**(PR.T)∩t11=>t12∩t12=>s1 (seul enchaînement de causalités possibles) et ∀t1∈EV(PR.T)∩t1≠ t12∩e1=>t1 #> ~(t1->t12), d'après (PR6), (PE2);

+ e2∈**EV**(PE.E)=>s2∈**EV**(PR.S) #> ∃t21(t21∈**EV**(PE.T)∩e1=>t21 ∩∃t22∈**EV**(PR.T)∩t21=>t22∩t22=>s2 (seul enchaînement de causalités possibles);

+ si on suppose e1->e2 : ∃t1∈**EV**(PR.T), ∃a1∈**EV**(PE.A) tels que : e1=>t1=>a1, d'après (2) a1->t21, d'après (PE3); Donc : t12->(ou<->)t1->a1->t21->t22

+ t12=>s1, t22=>s2 et t12->t22 #> s1->s2, d'après (PR8) C.Q.F.D.

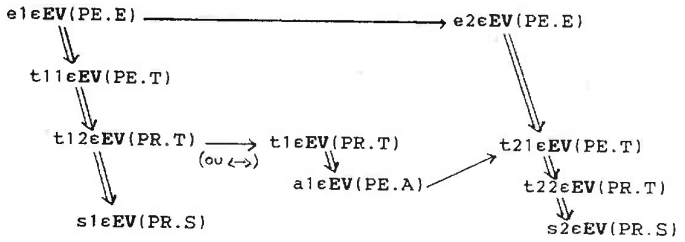


Fig. 3.41.

+ on peut refaire la même démonstration pour e2->e1 et e2≡e1.

Le réseau réalise donc bien le protocole voulu.

2.3. ELEMENTS D'UN ENVIRONNEMENT DE CONCEPTION :

2.3.1. Généralités :

Au vu de ce qui précède, il apparaît que les tâches principales du concepteur sont les suivantes :

- conception des structures,
- spécification des comportements,
- conduite des validations.

Des outils de natures diverses pourraient l'aider dans la réalisation de ces tâches :

- (1) un "éditeur/contrôleur" de structures pouvant intégrer à certains stades un "constructeur" de structures (cf. la méthode de partitionnement logique);
- (2) un "éditeur/contrôleur" des spécifications de comportement;
- (3) un outil de "validation automatique" ou "d'assistance à la validation";

La validation automatique du point (3) apparaît quelque peu utopique si on l'aborde dans toute sa généralité, bien qu'en théorie, s'agissant de calcul des prédicats du premier ordre, les outils de base existent. Nous avons engagé la réalisation d'un démonstrateur partiel écrit en PROLOG et centré sur la vérification des enchaînements de causalités.

N'ayant pas approfondi la définition syntaxique d'un langage de spécification de comportement complet et agréable d'utilisation, nous n'avons pas abordé le point (2).

Nous avons par contre réfléchi aux outils appropriés pour la manipulation des descriptions structurelles du point (1) et nous en avons réalisé des maquettes [LAL84, LAL87]. Ces manipulations liées au "programming-in-the-large" nous semblent assez différentes des manipulations classiques de code liées au "programming-in-the-small". En cours de conception, l'utilisateur doit gérer une (ou plusieurs) hiérarchie(s) de réseaux d'entités communicantes, ces réseaux étant plus ou moins complètement définis à un instant donné.

La description textuelle d'une telle structuration hiérarchique peut se faire de différentes manières; par exemple :

- par imbrication complète des implantations à profondeur quelconque (cf. fig. 3.42.a); les raffinements se traduisent alors par des insertions de texte;
- par descriptions séparées des différentes implantation à profondeur 1 (cf. fig.3.42.b); les raffinements se traduisent alors par des adjonctions de descriptions.

Fig. 3.42.a

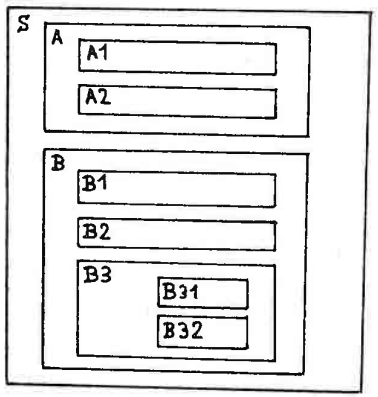
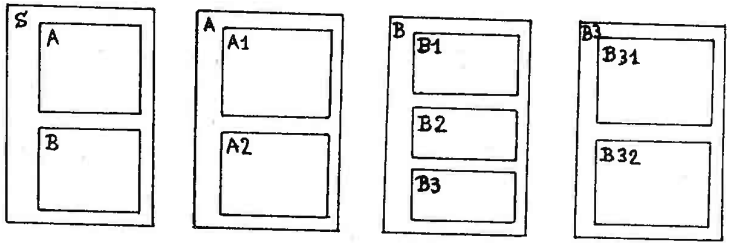


Fig. 3.42.b



Dans les deux cas, la manipulation par un éditeur classique est très pénible et ne permet pas d'avoir une vue d' ensemble du travail en cours (dans le cas a) ou de parcourir facilement la hiérarchie (cas b). Un éditeur syntaxique [DON79, MEL82, MEL85] apporte une nette amélioration, dans la mesure où il permet une circulation aisée dans une description hiérarchisée et qu'il permet d'extraire d'une hiérarchie une entité quelconque et de n'en représenter que les implantations à profondeur limitée (par exemple les implantations à profondeur 1 comme sur la figure 3.42.b). Dans la pratique, l'utilisation de ces possibilités reste d'un maniement lourd et l'agrément pour l'utilisateur extrêmement limité.

L'amélioration de l'interface utilisateur passe à l' évidence par une représentation graphique des structures : sous la forme d'une représentation a posteriori de structures préalablement définies ou, de préférence, sous la forme d'une interaction graphique complète lors de la définition des structures. L'intérêt porté par beaucoup à une méthode comme SADT [DIC77] provient à notre avis autant de l'existence de tels outils (ex: [LLS83]) que des qualités intrinsèques de la méthode elle même.

L'éditeur graphique interactif devra offrir les fonctionnalités de base suivantes :

- le travail de manière descendante, ascendante ou mixte (et donc la gestion de plusieurs hiérarchies indépendantes devant fusionner ultérieurement),
- les mises à jour multi-niveaux automatiques (car une modification à un niveau d'une hiérarchie peut se répercuter à d'autres niveaux),
- le travail en termes d'agents puis de modules,
- des commandes simples et puissantes de navigation dans les hiérarchies,
- l'intégration d'un maximum de contrôles immédiats à chaque opération et d'une commande de vérification différée (lorsque la structure est présumée être complète).

Le paragraphe suivant donne les grandes lignes de la maquette d'outil réalisée, dont on peut apercevoir quelques productions dans ce qui précède (ex : figures 3.19 à 3.22).

2.3.2. Une réalisation - l'éditeur graphique et vérificateur des structures :

Le concepteur travaille de manière interactive grâce à un ensemble de commandes que nous ventilerons en quatre catégories :

- commandes de création et de modification d'éléments de structure : en plus de la modification de l'image à l'écran, ces commandes entraînent la mise à jour de la structure de données qui représente la (les) hiérarchie(s) de réseaux, avec les caractéristiques (graphiques en particulier) de tous leurs éléments. Une modification à un niveau de la hiérarchie peut se répercuter à d'autres niveaux (ex: la suppression d'un agent entraîne celle de tous ses éventuels raffinements).
- commandes de déplacement :
  + dans la(les) hiérarchie(s),
  + dans un réseau dont la taille excède celle de l'écran
- commandes de documentation,
- commandes diverses : validation différée, sauvegardes et éditions, fin de session, etc.

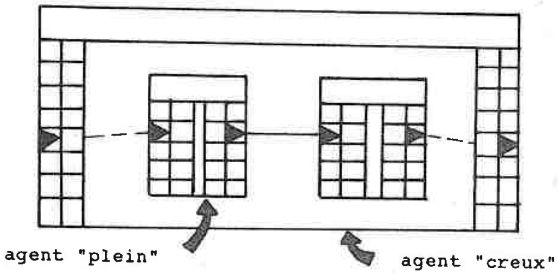
a) commandes de création et de modification d'éléments de structure :

(a1) **créer-agent**(nom-agent)+localisation par le curseur.

Nous appelons "agent plein", un agent dont on ne perçoit à l'écran que l'interface et non la constitution interne; le terme s'oppose à "agent creux", comme l'illustre la figure 2.43. (il s'agit d'une notion relative à une image,

un agent apparaissant "plein" à un niveau i de la hiérarchie et "creux" au niveau i+1).

Fig.3.43.



Cette commande fait apparaître, sur le réseau en cours de définition, le gabarit d'un agent "plein" de taille standard (20 ports), son coin inférieur gauche prenant la place du curseur. Si le gabarit s'inscrit correctement dans le réseau, l'utilisateur valide la proposition : l'agent est complètement dessiné et son nom est inscrit; sinon, le gabarit peut être déplacé à volonté.

La commande contrôle que le nom d'agent n'existe pas déjà.

(a2) **créer-port**(nom-agent, nom-port, type)+localisation par le curseur dans une des cases libres de l'agent.

La création n'est possible que si l'agent ("creux" ou "plein") est à l'écran; en fonction du type (E, S, ES, SE), le graphisme correspondant apparaît dans la case désignée, avec le nom du port.

La commande vérifie que le nom n'existe pas déjà sur l'agent.

Cette création peut se répercuter au niveau hiérarchique supérieur (si la création a été effectuée sur un agent "creux") ou au niveau hiérarchique inférieur (si la création a été effectuée sur un agent "plein").

(a3) **créer-liaison**(nom-agent$_1$, nom-port$_1$, nom-agent$_2$, nom-port$_2$, nom(s)-type(s)-messages)+localisation par le curseur des points de cassure du tracé et de l'emplacement des noms de types de messages.

Le mot "liaison" désigne en fait ici, un chemin de communication. L'agent$_1$ ("plein") doit être à l'écran; l'agent$_2$ ("plein") peut ne pas y être, mais les deux agents doivent appartenir au réseau à l'écran. L'utilisateur indique les points de cassure (cf. Fig.2.44.) du chemin qui n'est définitivement tracé qu'après validation.

La cohérence des types de ports liés est vérifiée.

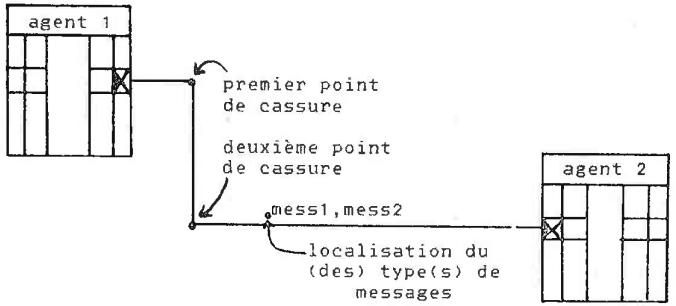


Fig.3.44.

(a4) **créer-attachement**(nom-agent$_1$, nom-port$_1$, nom-agent$_2$, nom-port$_2$, nom(s)-type(s)-messages)+localisation par le curseur des points de cassure du tracé et de l'emplacement des noms de types de messages.

Un des agents est "plein", l'autre est "creux". L'agent$_1$ doit être à l'écran. Le tracé se fait comme pour une liaison mais en pointillés.

Le test de compatibilité des types des ports est réalisé.

(a5) **raffiner**(nom-agent)

Crée un agent "creux" occupant tout l'écran, en reprenant dans les cases en bordure les ports déjà crées au niveau "plein".

L'agent doit exister et ne pas avoir déjà été raffiné.

Si la taille de l'agent "creux" n'est pas suffisante pour décrire son implantation, on peut l'agrandir et déplacer la fenêtre-écran sur lui (cf.ci-après).

(a6) **encapsuler**(nom-agent)

Le réseau à l'écran est encapsulé dans l'agent nommé. Si cet agent préexistait, il apparaît comme agent "creux" avec ses ports (cela correspond à l'assemblage de deux morceaux de la hiérarchie); sinon un agent "creux" sans ports apparaît.

L'agent nommé ne doit pas avoir été raffiné; un agent ne peut être encapsulé dans un de ses fils.

(a7) **renommer-agent**(ancien-nom,nouveau-nom)

L'agent concerné doit exister et se trouver à l'écran ("plein" ou "creux"); le nouveau nom ne doit pas exister déjà dans l'application.

La modification est répercutée si nécessaire dans la hiérarchie.

(a8) **renommer-port**(nom-agent,ancien-nom,nouveau-nom)
Le port doit exister et être à l'écran; le nouveau nom ne doit pas déjà exister sur l'agent.
La modification est répercutée si nécessaire dans la hiérarchie.
Remarque : pour changer les autres caractéristiques d'un port (type), d'une liaison ou d'un attachement (types de messages), il faut actuellement les détruire et les recréer.

(a9) **supprimer-agent**(nom-agent)
L'agent "plein" doit être à l'écran. Après un message d'avertissement, sont supprimés :
- l'agent nommé,
- ses ports et toutes les liaisons et attachements s'y rapportant,
- toute la hiérarchie de décomposition de l'agent.

(a10) **supprimer-port**(nom-agent,nom-port)
Le port doit être à l'écran. Après un message d'avertissement, sont supprimés :
- le port (éventuellement à deux niveaux de la hiérarchie),
- toutes les liaisons et attachements s'y rapportant.

(a11) **supprimer-liaison**(nom-agent$_1$,nom-port$_1$,nom-agent$_2$,nom-port$_2$)
La liaison doit être à l'écran.

(a12) **supprimer-attachement**(nom-agent$_1$,nom-port$_1$,nom-agent$_2$,nom-port$_2$)
L'attachement doit être à l'écran.

(a13) **déplacer-agent**(nom-agent)+localisation par le curseur
Le gabarit de l'agent plein apparaît à l'endroit repéré par le curseur. Après validation, l'agent complet est dessiné avec ses ports; l'ancien agent disparaît de même que toutes les liaisons et attachements s'y rapportant. Pour chaque liaison et attachement, le système demande de redéfinir son nouveau tracé (ensemble de points de cassure).
Le déplacement n'est possible qu'au sein d'un même réseau.
Remarque : il n'est pas proposé actuellement de commande de déplacement de port, de liaison ou d'attachement; l'utilisateur doit supprimer l'élément et le recréer.

(a14) **agrandir-agent**(nom-agent)
L'agent "plein" doit être à l'écran. Un gabarit apparaît, correspondant à l'ajout d'une hauteur standard vers le bas (20 ports supplémentaires). Après validation, l'agent est complètement tracé.
Plusieurs agrandissements successifs sont possibles. L'agent "creux" agrandi a une taille supérieure à celle de l'écran.

(a15) **déclarer-module**(nom-agent)
Déclare qu'un agent est un module. Cet agent ne doit pas lui-même être inclus dans un module ou inclure un module.

Cette commande n'est acceptée que lorsqu'il n'y a plus qu'une seule hiérarchie.

(a16) **dupliquer-agent**(nom-agent)+localisation par le curseur
Le gabarit de l'agent "plein" apparaît à l'endroit repéré par le curseur. Après validation, l'agent complet est dessiné avec ses ports; le nouvel agent hérite de tous les éventuels raffinements de l'agent d'origine.

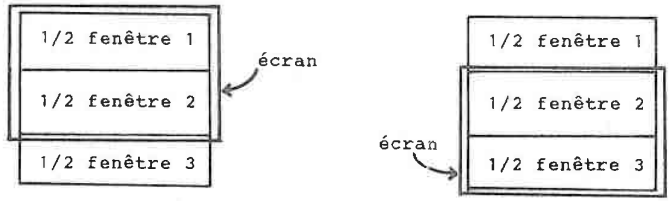
b) commande de déplacement :

(b1) **monter**
(b2) **descendre**
Ces commandes permettent de se déplacer dans les divers niveaux d'une hiérarchie.

(b3) **changer**
Permet de changer de hiérarchie (liste circulaire).

(b4) **haut**
(b5) **bas**
Ces commandes permettent de déplacer la fenêtre-écran sur un réseau de taille supérieure à celle de l'écran. Un recouvrement d'une demi fenêtre est conservé à chaque déplacement en particulier pour faciliter le tracé des liaisons et attachements.

Fig.3.45.



état initial ---> commande **bas** ---> état final

Ces commandes de déplacement peuvent être appelées de l'intérieur d'autres commandes (a1, a3, a4, a13,...).

(b6) **ailleurs**
Permet de créer une nouvelle hiérarchie indépendante (provisoirement) des précédentes.

c) commandes de documentation :

(c1) **créer-texte**({nom-agent | nom-type-message}) :
Permet de créer un texte à l'aide d'un éditeur général ou spécialisé et de l'attacher à un agent ou à un type de message, à des fins de documentation.

(c2) **voir-texte**(nom-agent | nom-type-message) :

Permet de visualiser le texte associé à l'entité désignée.

(c3) **lister**(agents | ports | liaisons | attachements | messages)

Permet de lister les entités désignées avec leurs caractéristiques.

d) commandes diverses :

(d1) **contrôler**

Effectue les contrôles différés et affiche des diagnostics de mauvaise structuration.

(d2) **sauvegarder**(nom-fichier)
(d3) **charger**(nom-fichier)
(d4) **écraser**(nom-fichier)
(d5) **catalogue**

Ces commandes permettent la gestion des fichiers d'applications.

(d6) **hard-copy**

Permet de sortir sur imprimante l'image à l'écran (sans le menu et la zone de dialogue).

(d7) **arrêt**

Fin de session.

Remarque : la sortie textuelle est réalisée à l'aide de l'éditeur syntaxique avec lequel l'éditeur graphique est interfacé (cf. chapitre 4 § 2.2).

Une maquette d'outil, qui implante toutes les commandes sauf celles de documentation, a été réalisée [LAL84]. Développée au CRIN sur MINI6, elle est écrite en FORTRAN A autour de l'IGL TEKTRONIX. Elle a été portée sous MULTICS au Centre Inter Régional d'Informatique de Lorraine, pour bénéficier d'un environnement matériel et logiciel plus riche. Mais sa cible naturelle reste une station de travail scientifique.

BIBLIOGRAPHIE :

[BOO86] BOOCH G.
Object oriented development.
IEEE Trans.on Computers, Vol SE-12, no 2, 2/86 (pp 211,221)

[CHE83] CHEN, YEH
Formal specification and verification of distributed systems
IEEE Trans. on Soft. Eng., Vol SE-9, no 6, 1983

[DIC77] DICKOVER, MAC GOWAN, ROSS
Software design with SADT.
Proc. Annual Conf.of the ACM, SEATLE, 77, (pp 99, 114)

[DON79] DONZEAU-GOUGE, HUET,KAHN
The MENTOR program manipulation system.
Rapport IRIA, 10/79

[EST86] ESTRIN, FENCHEL, RAZOUK, VERNON
SARA : Modelling, Analysis and Simulaticn Support for Design of Concurrent Systems.
IEEE Trans.on Soft.Eng., Vol SE-12, no 2, 2/86 (pp 293,311).

[GOF84] GOFFIC P.
Architecture de logiciels répartis. Spécification et simulation; Thèse Doct. Ing., NANCY, 6/84

[GOM84] GOMAA H.
A software design method for real-time systems.
CACM, Vol 27, no 9, 9/84 (pp 938,949)

[GRE77] GREIF I.
A language for formal problem specification.
CACM, Vol 20, no 12, 12/77 (pp 931, 935)

[GUE80] LE GUERNIC, RAYNAL
L'expression de la communication dans les langages.
Dans cours "synchronisation et communication dans un ensemble de processus communicants", RENNES, 10/81, (pp 127,171)

[JAC78] JACKSON M.A.
Information systems : modeling, sequencing and transformations; Proc. 3th Conf. on Soft. Eng., 1978

[KRA80] KRAMER, LISTER, MAGEE, SLOMAN
Distributed process control systems : programming and configuration; RR 80/12, Imp. College, LONDRES, 5/80

[LAL84] LALLIER M.
Un outil graphique d'aide à la conception de systèmes répartis; DEA, NANCY I, 9/84

[LAL87] LALLIER M.
Doctorat de 3ème cycle (a paraître).
NANCY I, 1987

[LAM78] LAMPORT L.
Time, clocks and the ordering of events in a distributed system; CACM, Vol 21, no 7, 7/78 (pp 558, 565)

[LUD83] LUDEWIG J.
ESPRESO : a system for process control software specification; IEEE Trans. on Soft. Eng., Vol SE-9, no 4, 7/83 (pp 427, 436)

[LLS83] LISSANDRE, LAGIER, SKALLI
SAS : un système d'assistance aux spécifications.
Actes journées BIGRE, CAP D'AGDE, 10/83, (pp 46, 67)

[MAR81] MARCOGLIESE, NOVARESE
Module and data allocation methods in distributed systems
Proc. 2nd Int. Conf. on Distrib. Computing Syst., 4/81, (pp 50, 59)

[MEL85] MELESE, MIGOT, VEROVE
The MENTOR-V5 documentation
INRIA, RT043, 1/85

[MEL82] MELESE B.
METAL, un langage de spécification pour le système MENTOR
TSI, Vol1, 4, 7-8/82 (pp 275, 286)

[OWI82] OWICKI, LAMPORT
Proving liveness properties of concurrents programs.
ACM Trans. on Prog. Lang. and Syst., Vol 4, no 3, 82 (pp 455,495)

[SHA86] SHATZ S.
A partitioning algorithm for distriuted software systems design; Information sciences, no38, Elsevier Pub., 1986 (pp 165, 180)

[SOK80] SOK S.
Evolutivité du logiciel.
Thèse 3ème cycle, NANCY I, 9/80

[TAR83] TARDIEU, ROCHFELD, COLLETTI
La méthode MERISE - Principes et outils.
Ed. d'Organisation, PARIS, 1983

CHAPITRE 4 :

CONCEPTION DETAILLEE DE L'APPLICATION

RESUME :

Ce quatrième chapitre présente tout d'abord les éléments d'un outil de spécification "programmatoire" au niveau détaillé : organisation interne des modules et types de communication (dont nous donnons la spécification formelle en termes de notre modèle).

La concrétisation de cet outil au travers de l'utilisation du langage ADA ou de la définition d'un pseudo-code est ensuite étudiée et discutée.

Enfin, le mode d'emploi de l'outil (règles de structuration, validations et tests) est abordé.

1. ELEMENTS D'UN OUTIL DE DESCRIPTION DETAILLEE :

La première phase de la démarche de conception (phase de définition de l'architecture globale), nous a permis de structurer l'application en un réseau de modules. L'objectif de la deuxième phase, dite de "conception détaillée", est de donner une description précise du contenu des modules, en termes des composants élémentaires (processus séquentiels) et de leurs interactions (communications de divers types, partage d'objets, interruptions), tout en continuant à faire abstraction du détail des calculs purement locaux et des implantations des structures de données. En ce sens il s'agit toujours de "programming-in-the-large" et non de "programming-in-the-small" [DRE76].

Nous étudions dans ce paragraphe, les principaux concepts pour la spécification détaillée des applications de commande de procédés industriels, en restant au niveau des principes généraux et sans chercher à donner des constructions linguistiques précises.

1.1. ORGANISATION INTERNE DES MODULES :

Chaque module est constitué d'un ensemble de processus séquentiels susceptibles de s'exécuter en parallèle. En raison des propriétés des applications visées, ces processus sont permanents (leur durée de vie est celle de l'application), cycliques et en nombre fixe -cf. propriétés (2),(3) du chapitre 1 § 4-. Aucun mécanisme ne doit impliquer la terminaison des processus (ex: exceptions dans le corps des processus).

Les interactions entre processus n'appartenant pas au même module se font exclusivement par des communications, via les ports du module; pour des raisons de lisibilité un seul processus peut consommer des messages sur un port d'entrée; en sortie, par contre, plusieurs processus peuvent produire des messages sur un même port. En raison du caractère parallèle de leur environnement -cf. propriété (1)-, les processus doivent pouvoir choisir de manière indéterministe (ou selon des priorités statiques) entre plusieurs sollicitations en entrée et ne les accepter que si une condition liée à leur état est vérifiée (choix indéterministe avec gardes). Enfin, les processus doivent pouvoir se débloquer localement d'une communication bloquante (notion d'autonomie).

Les interactions entre processus d'un même module doivent pouvoir se faire sans restrictions, en particulier par accès à des structures de données communes, étant donné que tout se passe sur le même site.

Des priorités statiques sont attachées aux processus afin d'organiser leur ordonnancement en cas de conflit pour l'utilisation du (ou des) processeur(s) du site. Dans chaque module, des processus peuvent être dédiés à la prise en compte

des interruptions externes. Il nous apparaît souhaitable que l'exécution d'un tel processus d'interruption puisse s'accompagner du blocage de tous les processus "normaux" du module (et d'éventuels processus d'interruption moins prioritaires) jusqu'à la fin de la prise en compte de l'interruption. Ces processus d'interruption devraient également pouvoir replacer les processus du même module dans un état donné à leur reprise (cf. par exemple le mécanisme de réinitialisation -"reset"- proposé dans [KOP82] qui correspond à l'effacement des éventuels messages d'entrée en attente, à la réinitialisation des objets locaux et du pointeur d'exécution). Ceci, afin de permettre un contrôle strict du comportement des modules en cas d'interruption, même sur un site multi-processeurs.

1.2. TYPES DE COMMUNICATION :

1.2.1. Les besoins généraux en communication des applications réparties en commande de procédés:

Quelques publications analysent spécifiquement les besoins en communication de ces applications [KRA81, KOP82]. Elles considèrent essentiellement les échanges à l'interface du système de commande et de son environnement et dressent des listes de propriétés requises. Le tableau 4.1. de la page suivante résume les résultats des deux études précédemment citées.

On peut faire ressortir à partir de ces études les besoins fondamentaux suivants :

-(a) l'asynchronisme (émetteur non bloqué),
-(b) les échanges multi-destinataires (en diffusion ou sélection),
-(c) les échanges bidirectionnels du type requête/réponse,
-(d) la consommation non exhaustive des messages,
-(e) les délais maximum d'attente sur les opérations de communication bloquantes ("time-out").

Afin de décrire précisément nos choix nous revenons tout d'abord sur le concept général de communication afin d'en mettre en lumière les différents aspects.

Tableau 4.1.

AUTEUR	CATEGORIE DE COMMUNICATION		PROPRIETES
KRAMER	"status" (rapport d'état)	"periodic status" ou "event status"	du type notification (+ monodirectionnelle, + 1 ou n destinataire(s), + émetteur non bloqué, + récepteur bloqué en attente de message (timeout possible), + consommation du dernier reçu)
		"query status"	du type requête/réponse (+ bidirectionnelle, + 1 destinataire, + émetteur bloqué en attente de réponse (timeout possible), + récepteur bloqué en attente de message (timeout possible), + consommation exhaustive)
	alarme		du type notification et d'urgence forte.
	commande	avec 2 réponses (début + fin de l'action commandée)	du type requête/réponse et notification.
		avec 1 réponse (fin action)	du type requête/réponse.
KOPETZ	mesure de valeur		+ monodirectionnelle, + 1 ou n destinataire(s), + émetteur non bloqué, + récepteur bloqué en attente de message (timeout possible) + filtrage possible selon la durée de validité, le contenu du message, le nom de l'émetteur, etc. + consommation du dernier reçu avec remise.
	rapport d'événement		idem précédent sauf + consommation exhaustive
	alarme		idem précédent et urgence forte.
	commande		idem précédent sauf urgence
	requête/réponse		+ bidirectionnelle, + 1 ou n destinataire(s), + émetteur bloqué en attente de réponse (timeout possible), + filtrage possible, + consommation exhaustive.

1.2.2. Le concept de communication aux divers stades de la conception :

Le concept général de communication a fait l'objet de très nombreux travaux [CAR82, DPT82, MAC82, PER85, ...]. On est frappé par la multitude de caractéristiques très hétérogènes que l'on peut attacher au concept. En rapprochant les trois premières références par exemple, on peut établir une liste de 16 caractéristiques, sans prétendre à une quelconque exhaustivité (ni orthogonalité) :

- (i) "symetricity" [CAR82] : degré de connaissance des noms des interlocuteurs
  + communication symétrique : chaque interlocuteur doit connaître le nom de l'autre,
  + communication asymétrique : le récepteur peut ne pas connaître le nom de l'émetteur.

- (ii) "concurrency" [CAR82] : degré de parallélisme entre émetteur et récepteur; les deux extrêmes étant :
  + l'exécution parallèle : l'émetteur continue dès que le message est composé,
  + l'attente de terminaison : l'émetteur est bloqué jusqu'à ce que le récepteur ait fini le traitement du message.

- (iii) "determinism" [CAR82] : possibilité de choix indéterministe entre plusieurs actions de communication dans le corps des entités communicantes ou non.

- (iv) "multiplexing" [CAR82] : possibilité que plus d'un émetteur ou récepteur participe à un échange; il y a 3 cas de figure principaux :
  + sans,
  + récepteurs multiples pour une même émission,
  + émetteurs potentiels multiples pour une même réception.

- (v) "message content" [CAR82] : nature des messages; il y a de nombreuses possibilités comme :
  + valeur d'un certain type de données (ex : chaînes de caractères),
  + valeur d'un type quelconque de données,
  + liste d'arguments et ses variantes : paramètres d'appel de procédures à distance (cf.DP [BRI78]), liste de couples nom-valeur (cf.PLITS [FEL79]), etc.

- (vi) "handshaking" [CAR82] : échange de messages additionnels transparents au niveau application, pour contrôler la communication; par exemple :
  + demande de réception (récepteur vers émetteur),
  + demande d'émission (émetteur vers récepteur),
  + acquittement de réception (récepteur vers émetteur).

- (vii) "ordering" [CAR82] : garantie sur l'ordre de réception par rapport à l'ordre d'émission. Soit :

  + pas de garantie d'arrivée,
  + garantie d'arrivée mais pas de garantie d'ordre,
  + ordre relatif garanti (c'est à dire pour chaque triplet émetteur-récepteur-échange),
  + ordre absolu garanti (l'ordre d'émission est conservé même s'il y a plusieurs émetteurs possibles; nécessite une forme de temps global).

- (viii) "queueing" [CAR82] : stockage des messages ; soit :
  + pas de stockage,
  + tampon de taille bornée,
  + tampon de taille non bornée.

- (ix) "queueing discipline" [CAR82] : gestion du tampon; soit :
  + pas de discipline garantie,
  + dans l'ordre d'arrivée (FIFO),
  + manipulation du tampon à discrétion par le receveur pour prélever les messages dans un ordre spécifique (ex: selon contenu des messages).

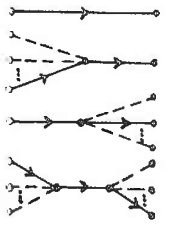
- (x) localisation des points d'échange dans les entités [DPT82] :
  + explicite par des primitives de communication dans le corps des entités,
  + implicite en entrée ou en sortie de l'entité (cf. SYGARE [THO80], acteurs [HEW77])

- (xi) mode de désignation des partenaires [DPT82] :
  + direct, par les noms des entités qui communiquent,
  + indirect, par exemple par des ports.

- (xii) mono ou bi-directionnalité [DPT82] : dans le cas bidirectionnel, il s'agit de deux échanges au niveau application (et non à des niveaux différents -cf.(vi)-.

- (xiii) "sémantique" de l'échange [DPT82] :
  + déterministe s'il existe une relation statique entre les flots de données produites et consommées,
  + indéterministe, sinon.

- (xiv) "profils de communication" [MAC82] :
  + 1 vers 1 :
  + 1 parmi n vers 1 :
  + 1 vers n :
  + 1 parmi n vers 1 parmi m :



- (xv) "selectivité" des réceptions [MAC82] :
  + réceptions inconditionnelles,
  + selon le type de message,

+ selon le nom de l'émetteur,
+ selon l'état interne du récepteur,
+ selon le contenu du message.

- (xvi) signification de l'acquittement [MAC82] :
+ message arrivé au destinataire,
+ message accepté par le destinataire,
+ message traité par le destinataire,
+ message accepté par le système de communication local de l'émetteur, chargé de l'acheminement.

Une première manière de clarifier le concept est de l'examiner à différents niveaux d'abstraction. La nature même de notre travail nous incite à le considérer au niveau de la spécification architecturale puis au niveau de la spécification détaillée.

Au premier niveau l'aspect essentiel est la liaison, qui traduit un besoin de communication et précise :
- les interlocuteurs potentiels (cf. (iv), (xiv)),
- la nature de l'échange, mono ou bi-directionnel (cf.(xi)),
- le (ou les) type(s) de messages échangés (cf.(v)).

Dans le langage de spécification au niveau logique (modèle simplifié), les liaisons sont "transparentes", c'est à dire que les événements d'émission et de réception sont confondus; il s'agit d'une communication "idéale", instantanée et sûre. Cette vision est cohérente avec la volonté de s'abstraire de tous les aspects liés à l'implantation. Au niveau organique, dans le modèle "complet", il est possible de spécifier de manière plus réaliste les caractéristiques des liaisons inter sites (perte possible de messages, conservation ou non de l'ordre, délais, etc.).

Au second niveau, celui de la spécification détaillée, apparaissent les interlocuteurs effectifs (processus émetteur et récepteur(s)); l'aspect essentiel est le **type des communications**, caractérisé en particulier par les synchronisations exigées des interlocuteurs et le mode de gestion des tampons en cas d'asynchronisme (cf. (ii), (viii), (ix), (xiii), (xvi)). Nous montrons au paragraphe suivant comment le modèle "complet" de comportement permet de spécifier formellement ces divers types de communication. Dans l'optique d'une description "programmatoire", des **primitives de communication** permettent de décrire les échanges selon ces types. On aborde alors les aspects linguistiques des primitives (cf. (iii), (xii), (xv)) et des désignations (cf. (i), (x)).

Le dernier niveau, qui nous intéresse moins, est celui de la mise en oeuvre des primitives au niveau système (cf. (vi)).

Au paragraphe suivant nous rappelons brièvement nos choix en matière de liaisons et exposons nos choix en matière de types de communication, en nous appuyant sur les propriétés requises indiquées au § 1.2.1.

1.2.3. Nos choix :

Nos cinq types de liaison du chapitre 3 § 1.1.2. (monodirectionnelle bipoint, bidirectionnelle bipoint, monodirectionelle multipoint en diffusion, monodirectionnelle multipoint en sélection et bidirectionnelle multipoint en sélection) satisfont les propriétés requises (b) et (c).

An niveau le plus macroscopique, nous distinguons trois types de communication :

- les communications **asynchrones sur les liaisons monodirectionnelles** : l' émetteur est non bloqué (c'est à dire qu'il confie son message au système de communication local chargé de l'acheminer et qu' il poursuit son exécution); chaque récepteur n'émet ni réponse ni acquittement; les messages reçus peuvent s'accumuler et il faut préciser la méthode de gestion du tampon de chaque récepteur.

- les communications **synchrones sur les liaisons monodirectionnelles** : l'émetteur est bloqué en attente d'un acquittement de prise en compte émis par chaque récepteur.

- les communications **synchrones sur les liaisons bidirectionnelles** : l'émetteur est bloqué en attente d'un message de réponse émis par le destinataire de sa requête.

On distingue donc à ce niveau d'observation trois types d'émission et trois types de réception principaux :

- émission non bloquante : $E\bar{B}$
- émission bloquante en attente d'acquittement : EBA
- émission bloquante en attente de réponse : EBR

- réception avec réponse : RR
- réception avec acquittement : RA
- réception sans acquittement : $R\bar{A}$

Leur assemblage est régi par les règles de compatibilité suivantes :

$E\bar{B}$ est compatible avec $R\bar{A}$
EBA est compatible avec RA
EBR est compatible avec RR

Au niveau d'observation le plus fin nous subdivisons les types d'émission de la manière suivante :

- émission non bloquante sur une liaison monodirectionnelle bipoint ($E\bar{B}1$),
- émission non bloquante sur une liaison monodirectionnelle multipoint en diffusion ($E\bar{B}n$),

- émission non bloquante sur une liaison monodirectionnelle multipoint en sélection (EB1/n), le choix du chemin pouvant être explicite ou aléatoire,

- émission bloquante en attente d'acquittement sur une liaison monodirectionnelle bipoint, avec ou sans time-out (EBA1 et EBA1(tmax)),
- émission bloquante en attente d'acquittement sur une liaison monodirectionnelle multipoint en diffusion, avec ou sans time-out (EBAn et EBAn(tmax)),
- émission bloquante en attente d'acquittement sur une liaison monodirectionnelle multipoint en sélection, avec ou sans time-out (EBA1/n et EBA1/n(tmax)) et avec choix explicite ou aléatoire du chemin,

- émission bloquante en attente de réponse sur une liaison bidirectionnelle bipoint, avec ou sans time-out (EBR1 et EBR1(tmax)),
- émission bloquante en attente de réponse sur une liaison bidirectionnelle multipoint en sélection, avec ou sans time-out (EBR1/n et EBR1/n(tmax)) et avec choix explicite ou aléatoire du chemin.

Les réceptions sont toutes bloquantes mais on peut construire des réceptions non bloquantes avec un time-out à zéro. Il est nécessaire de distinguer les réceptions sur des ports partagés entre plusieurs liaisons (réceptions "depuis 1 parmi n") où le choix du chemin peut être explicite ou aléatoire, et les réceptions sur des ports non partagés (réceptions "depuis 1") :

- réception avec réponse depuis 1, avec ou sans time-out (RR1 et RR1(tmax)),
- réception avec réponse depuis 1 parmi n, avec ou sans time-out et choix du chemin aléatoire ou explicite (RR1/n et RR1/n(tmax)),

- réception avec acquittement depuis 1, avec ou sans time-out (RA1 et RA1(tmax)),
- réception avec acquittement depuis 1 parmi n, avec ou sans time-out et choix du chemin aléatoire ou explicite (RA1/n et RA1/n(tmax)),

- réception sans acquittement depuis 1, sans time-out et politique de gestion du tampon en FIFO, LIFO, HASARD, DERNIER RECU SANS REMISE, DERNIER RECU AVEC REMISE (RA1(politique)) ou avec time-out et politique FIFO, LIFO, HASARD, DERNIER RECU SANS REMISE (RA1(politique,tmax)),
- réception sans acquittement depuis 1 parmi n, avec ou sans time out, choix du chemin aléatoire ou explicite, et politique de gestion prise dans les listes ci-dessus (RA1/n et RA1/n(tmax)).

Cette large palette de possibilités permet bien entendu de satisfaire les cinq besoins fondamentaux du § 1.2.1. Nous nous attachons au paragraphe suivant à expliciter formellement le contenu de ces types de communication.

1.2.4. Spécification formelle des types de communication retenus :

Nous proposons une spécification ascendante : les types d'émission et de réception élémentaires (EX1,RX1) sont donnés d'abord. A partir d'eux, les types d'émissions complexes (EXn, EX1/n) et de réceptions complexes (RX1/n) sont construits. Enfin, par interconnexion de types d'émission et de réception compatibles via des liaisons dont nous rappelons les spécifications courantes (type LF, pour liaison fiable et type LF, pour liaison non fiable), nous pouvons construire tous les types de communications possibles.

Exemple : Type de communication asynchrone avec diffusion vers 3 récepteurs et :
. réception en FIFO avec time-out sur le premier (liaison LF),
. réception en dernier reçu avec remise sur le deuxième (liaison LF),
. réception en FIFO sans time-out sur le troisième (liaison LF).

♦

Remarque : pour faciliter la lecture des spécifications nous nous interdisons les références à des propriétés qui ne se retrouvent pas sur la même page et les répétons quand nécessaire (exception faite pour les propriétés générales de conservation de l'ordre et de non duplication qui se retrouvent presque partout et sont simplement nommées). Nous reprenons, pour présenter les émissions et réceptions, la classification détaillée du § 1.2.3.

♦

a) émission non bloquante vers 1 ($E\bar{B}1$) :

En D, arrivent les messages de début d'émission avec la valeur v à transmettre et de F, partent les signaux de fin d'émission;

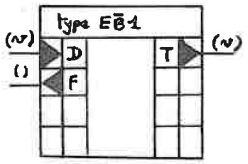
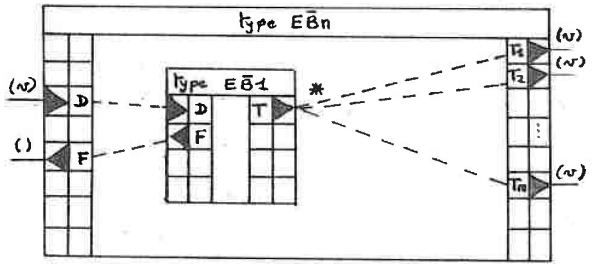


Fig.4.1.

(1) $\forall d(d\in EV(D) \langle\#\rangle \exists t(t\in EV(T)\cap d=>t))$
(2) $\forall d(d\in EV(D) \langle\#\rangle \exists f(f\in EV(F)\cap d=>f))$
(émetteur non bloqué)
(3) $\forall d,\forall t(d\in EV(D)\cap t\in EV(T)\cap d=>t \#>$
t.msg=d.msg)
(4) $\forall d1,\forall d2(d1,d2\in EV(D)\cap ord(d2)=ord(d1)+1$
$\#> \exists f(f\in EV(F)\cap d1->f->d2))$
(opérations d'émission en séquence)
(5) $\forall x1,\forall x2,\forall y1,\forall y2(x1,x2\in EV(X)\cap y1,y2\in EV(Y)\cap x1=>y1\cap$
$x2=>y2 \#> (x1->x2\cap y1->y2)\vee(x2->x1\cap y2->y1)\vee(x1\equiv y1\cap y1\equiv y2))$
(pour tous ports X et Y sur l'agent; cette propriété sera nommée "conservation-ordre(X,Y)" dans la suite).
(6) $\forall x,\forall y1,\forall y2(x\in EV(X)\cap y1,y2\in EV(Y)\cap x=>y1\cap x=>y2 \#> y1\equiv y2)$
(pour tous ports X et Y sur l'agent; cette propriété sera nommée "non-duplication(X,Y)" dans la suite).

b) émission non bloquante vers n ($E\bar{B}n$) :



(rappel : tout événement sur T se confond avec n événements sur $T_1, \ldots, T_n$)

Fig.4.2.

c) émission non bloquante vers 1 parmi n ($E\bar{B}1/n$) :

L'entité de type S1 réalise la sélection soit selon la valeur du message reçu sur D, soit au hasard (si le message contient la valeur indéfinie $\perp$)

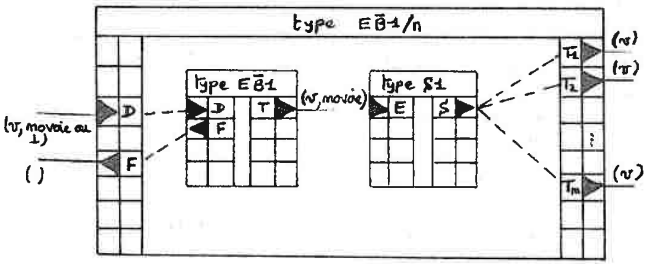


Fig.4.3.

(rappel : le port S peut être décrit comme n sous-ports

port S)

Type S1 :
(1) $\forall e(e\in EV(E)\cap P1 \langle\#\rangle \exists ie[1,n],\exists s(s\in EV(S(i))\cap e=>s))$
P1 : e.msg.novoie=i $\vee$ (e.msg.novoie=$\perp \cap$ i=hasard(1,n))
où hasard(1,n) délivre un nombre au hasard entre 1 et n.
(2) $\forall ie[1,n],\forall e,\forall s(e\in EV(E)\cap s\in EV(S(i))\cap e=>s \#> s.msg=$
e.msg.v)
(3) conservation-ordre(E,S)
(4) non-duplication(E,S)

d) émission bloquante en attente d'acquittement vers 1 :

d1) sans time-out (EBA1) : en cas de perte de l'acquittement il y a blocage infini; le cas où la liaison dupliquerait l'acquittement est prévu (grâce aux numéros d'ordre qui circulent)
(1) $\forall d(d\in EV(D) \langle\#\rangle \exists t(t\in EV(T)\cap d=>t))$
(2) $\forall a(a\in EV(A)\cap P2 \langle\#\rangle \exists f(f\in EV(F)\cap a=>f))$
P2 : $\exists d(d\in EV(D)\cap d->a)\cap a.msg.no=ord(d)\cap$
$\tilde{}\exists a1(a1\in EV(A)\cap d->a1->a\cap a1.msg.no=ord(d))$
(3) $\forall d,\forall t(d\in EV(D)\cap t\in EV(T)\cap d=>t \#> t.msg.v=$
d.msg$\cap$t.msg.no=ord(d))
(4) $\forall d1,\forall d2(d1,d2\in EV(D)\cap ord(d2)=ord(d1)+1$
$\#> \exists f(f\in EV(F)\cap d1->f->d2))$
(5) conservation-ordre(X,Y)
(6) non-duplication(X,Y)

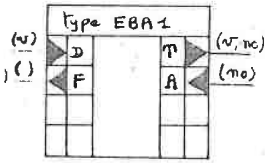


Fig.4.4.

d2) avec time-out (EBA1(tmax)) : la liaison issue du port O porte un signal si le délai d'attente maximum tmax est dépassé (d'éventuels acquittements postérieurs seront ignorés); le type horloge (HORL) envoie des signaux à chaque unité de temps (unité utilisée pour la mesure des time-out).

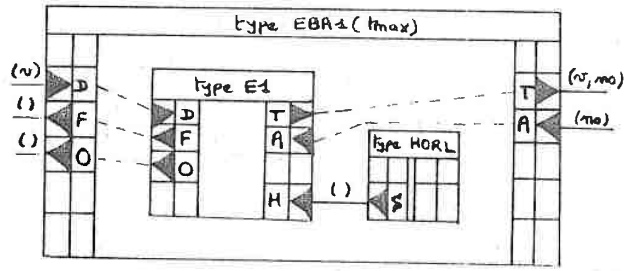


Fig.4.5.

type E1, identique à EBA1 sauf :
(2) $\forall a(a\in EV(A)\cap P2 \langle\#\rangle \exists f(f\in EV(F)\cap a=>f))$
P2 : $\exists d(d\in EV(D)\cap d->a)\cap a.msg.no=ord(d)\cap\tilde{}\exists a1(a1\in EV(A)\cap$
$d->a1->a\cap a1.msg.no=a.msg.no)\cap intervalle(a,d)<tmax$
(5) $\forall d1,\forall d2(d1,d2\in EV(D)\cap ord(d2)=ord(d1)+1 \#> \exists f(f\in EV(F)\cap$
$d1->f->d2)\vee\exists o(o\in EV(O)\cap d1->o->d2))$
(7) $\forall h(h\in EV(H)\cap P7 \langle\#\rangle \exists o(o\in EV(O)\cap h=>o))$
P7 : $\exists d(d\in EV(D)\cap d->h)\cap\tilde{}\exists a,f(a\in EV(A)\cap f\in EV(F)\cap d->a->h\cap$
$a=>f)\cap\tilde{}\exists h1,o1(h1\in EV(H)\cap o1\in EV(O)\cap d->h1->h\cap h1=>o1)\cap$
intervalle(d,h)>tmax

type HORL:
(1) ∀s1(s1∈EV(S) ⟨#⟩ ∃s2(s2∈EV(S)∩s1=>s2))
(2) ∀s1,∀s2(s1,s2∈EV(S)∩s1=>s2 #> intervalle(s1,s2)=1)

e) émission bloquante en attente d'acquittement vers n :

e1) sans time-out (EBAn) : il faut un acquittement de la part de chaque destinataire;

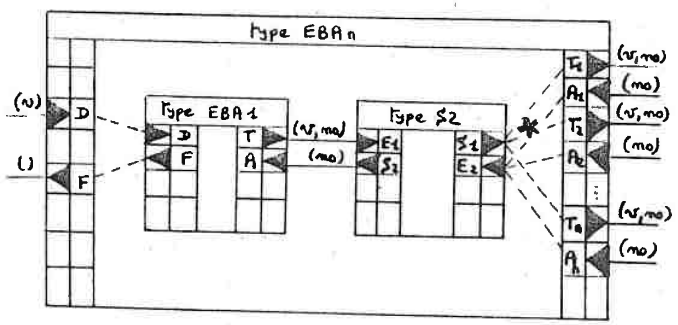
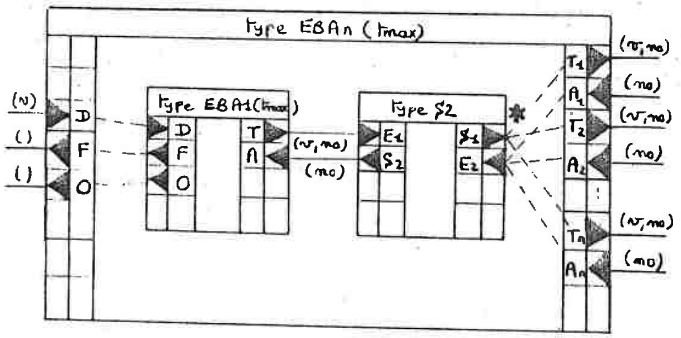


Fig.4.6.

type S2:
(1) ∀i∈[1,n],∀e(e∈EV(E1) ⟨#⟩ ∃s(s∈EV(S1(i))∩e=>s))
(2) ∀i∈[1,n],∀e(e∈EV(E2(i))∩P2 ⟨#⟩ ∃s(s∈EV(S2)∩e=>s))
P2 : ∃e1(e1∈EV(E2(i+1 mod n))∩e1->e∩e1.msg.no=e.msg.no)
∩∃e2 ... ∩∃en-1(en-1∈EV(E2(i+n-1 mod n))∩en-1->e∩
en-1.msg.no= e.msg.no)
(3) ∀i∈[1,n],∀e,∀s(e∈EV(E1)∩s∈EV(S1(i))∩e=>s #> s.msg=
e.msg)
(4) ∀i∈[1,n],∀e,∀s(e∈EV(E2(i))∩s∈EV(S2)∩e=>s #> s.msg=
e.msg)
(5) conservation-ordre(X,Y)
(6) non-duplication(X,Y)

e2) avec time-out (EBAn(tmax)) :

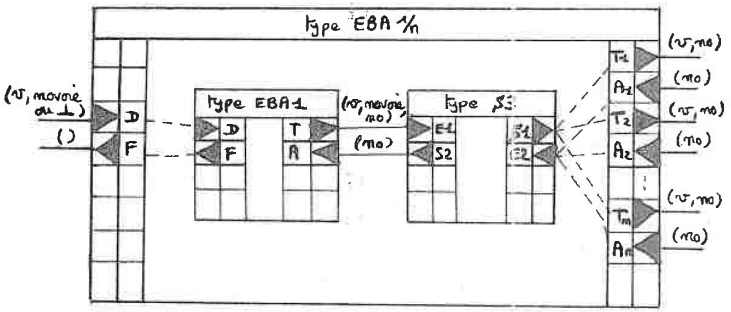
Fig.4.7.



f) émission bloquante en attente d'acquittement vers 1 parmi n:

f1) sans time-out (EBA1/n) :

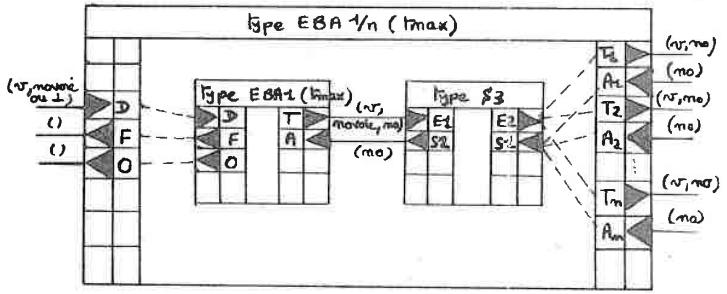
Fig.4.8.



type S3 :
(1) ∀e(e∈EV(E1)∩P1 ⟨#⟩ ∃s(s∈EV(S1(i))∩e=>s))
P1 : e.msg.novoie=i ∨ (e.msg.novoie=⊥∩i=hasard(1,n))
(2) ∀i∈[1,n],∀e(e∈EV(E2(i)) ⟨#⟩ ∃s(s∈EV(S2)∩e=>s))
(3) ∀i∈[1,n],∀e,∀s(e∈EV(E1)∩s∈EV(S1(i))∩e=>s #> s.msg.v=
e.msg.v∩s.msg.no=e.msg.no)
(4) ∀i∈[1,n],∀e,∀s(e∈EV(E2(i))∩s∈EV(S2)∩e=>s #> s.msg=
e.msg)
(5) conservation-ordre(X,Y)
(6) non-duplication(X,Y)

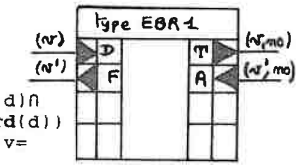
f2) avec time-out (EBA1/n(tmax)) :

Fig.4.9.



g) émission bloquante en attente de réponse vers 1 :

g1) sans time-out (EBR1) :

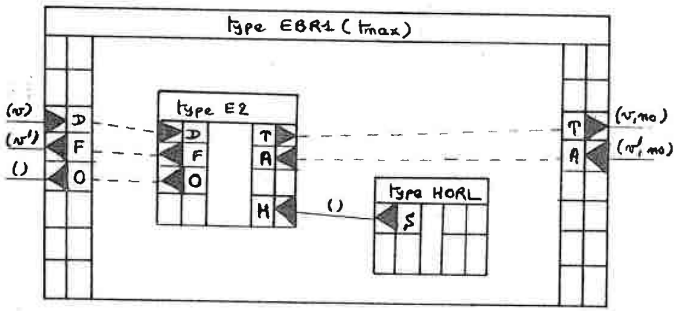


(1) ∀d(d∈**EV**(D) ⟨#⟩]t(t∈**EV**(T)∩d=>t))
(2) ∀a(a∈**EV**(A)∩P2 ⟨#⟩]f(f∈**EV**(F)∩a=>f))
P2 :]d(d∈**EV**(D)∩d->a)∩a.**msg.no**=**ord**(d)∩
~]a1(a1∈**EV**(A)∩d->a1->a∩a1.**msg.no**=**ord**(d))
(3) ∀d,∀t(d∈**EV**(D)∩t∈**EV**(T)∩d=>t #⟩ t.**msg**.v=
d.**msg**∩t.**msg**.no=**ord**(d))
(4) ∀a,∀f(a∈**EV**(A)∩f∈**EV**(F)∩a=>f #⟩ f.**msg**=
a.**msg**.v')
(5) ∀d1,∀d2(d1,d2∈**EV**(D)∩**ord**(d2)=**ord**(d1)+1
#⟩]f(f∈**EV**(F)∩d1->f->d2))
(6) **conservation-ordre**(X,Y)
(7) **non-duplication**(X,Y)

Fig.4.10.

g2) avec time-out (EBR1(tmax)) :

Fig.4.11.

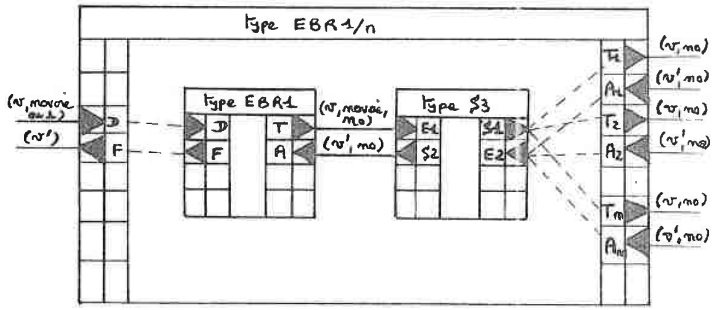


type E2 : identique à EBR1 sauf
(2) ∀a(a∈**EV**(A)∩P2 ⟨#⟩]f(f∈**EV**(F)∩a=>f))
P2 :]d(d∈**EV**(D)∩d->a)∩a.**msg.no**=**ord**(d)∩~]a1(a1∈**EV**(A)∩
d->a1->a∩a1.**msg.no**=a.**msg.no**)∩**intervalle**(a,d)<tmax
(5) ∀d1,∀d2(d1,d2∈**EV**(D)∩**ord**(d2)=**ord**(d1)+1 #⟩]f(f∈**EV**(F)∩
d1->f->d2) ∨]o(o∈**EV**(O)∩d1->o->d2))
(8) ∀h(h∈**EV**(H)∩P8 ⟨#⟩]o(o∈**EV**(O)∩h=>o))
P8 :]d(d∈**EV**(D)∩d->h)∩~]a,f(a∈**EV**(A)∩f∈**EV**(F)∩d->a->h∩
a=>f)∩~]h1,o1(h1∈**EV**(H)∩o1∈**EV**(O)∩d->h1->h∩h1=>o1)∩
intervalle(d,h)>tmax

h) émission bloquante en attente de réponse vers 1 parmi n :

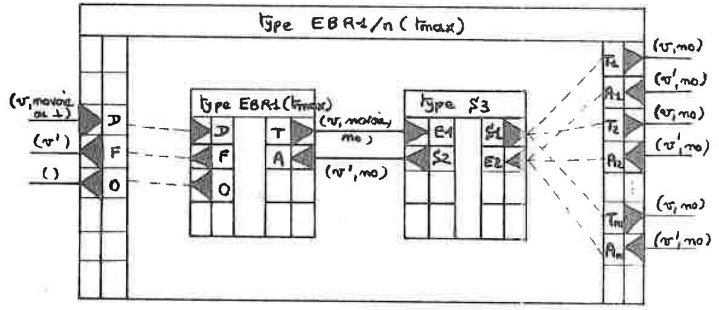
h1) sans time-out (EBR1/n) :

Fig.4.12.



h2) avec time-out (EBR1/n(tmax)) :

Fig.4.13.



i) réception avec réponse depuis 1 :

i1) sans time-out (RR1) : D accueille les demandes de réception; sur F sont transmises les valeurs reçues et R accueille les réponses.

(1) ∀d(d∈**EV**(D)∩P1 #> ∃f(f∈**EV**(F)∩d=>f))
P1 : ∃t(t∈**EV**(T)∩t->d)∩~∃t1(t1∈**EV**(T)∩t1->t∩t1.**msg**.no=t.**msg**.no)∩~∃f1(f1∈**EV**(F)∩t->f1->d)
(2) ∀t(t∈**EV**(T)∩P2 #> ∃f(f∈**EV**(F)∩t=>f))
P2 : ∃d(d∈**EV**(D)∩d->t)∩~∃f(f∈**EV**(F)∩d->f->t)∩~∃t1(t1∈**EV**(T)∩t1->t∩t1.**msg**.no=t.**msg**.no)
(3) ∀r(r∈**EV**(R) <#> ∃a(a∈**EV**(A)∩r=>a))
(4) ∀d,∀f(d∈**EV**(D)∩f∈**EV**(F)∩d=>f∩P1 #> f.**msg**=t.**msg**.v)
(5) ∀t,∀f(t∈**EV**(T)∩f∈**EV**(F)∩t=>f #> f.**msg**=t.**msg**.v)
(6) ∀r,∀a(r∈**EV**(R)∩a∈**EV**(A)∩r=>a∩∃t(t∈**EV**(T)∩t->r)∩~∃t1(t1∈**EV**(T)∩t->t1->r)∩t1.**msg**.no>t.**msg**.no) #> a.**msg**.v'=r.**msg**.v' ∩ a.**msg**.no=t.**msg**.no)
(7) ∀f(f∈**EV**(F) #> ∃d(d∈**EV**(D)∩P1∩d=>f) v ∃t(t∈**EV**(T)∩P2∩t=>f))
(8) ∀d1,∀d2(d1,d2∈**EV**(D)∩**ord**(d2)=**ord**(d1)+1 #> ∃f(f∈**EV**(F))∩∃r(r∈**EV**(R))∩d1->f->r->d2)
(9) conservation-ordre(X,Y)
(10) non-duplication(X,Y)

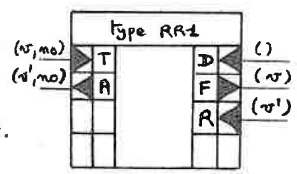
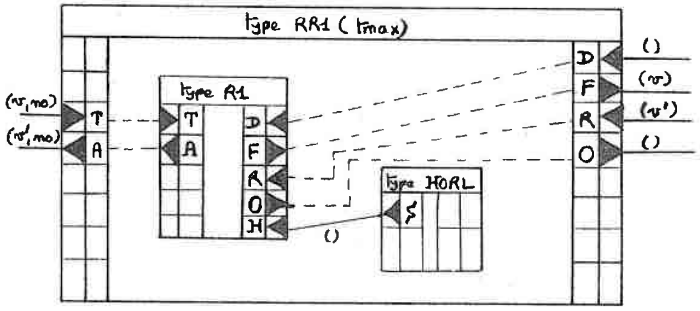


Fig.4.14.

i2) avec time-out (RR1(tmax)) :

Fig.4.15.



type R1, identique à RR1 sauf :
(2) ∀t(t∈**EV**(T)∩P2 #> ∃f(f∈**EV**(F)∩t=>f))
P2 : ∃d(d∈**EV**(D)∩d->t)∩~∃f(f∈**EV**(F)∩d->f->t)∩~∃t1(t1∈**EV**(T)∩t1->t∩t1.**msg**.no=t.**msg**.no)∩**intervalle**(d,t)<tmax
(8) ∀d1,∀d2(d1,d2∈**EV**(D)∩**ord**(d2)=**ord**(d1)+1 #> (∃f(f∈**EV**(F))∩∃r(r∈**EV**(R))∩d1->f->r->d2) v ∃o(o∈**EV**(O)∩d1->o->d2)
(11) ∀h(h∈**EV**(H)∩P11 <#> ∃o(o∈**EV**(O)∩h=>o))
P11 : ∃d(d∈**EV**(D)∩d->h)∩~∃d1(d1∈**EV**(D)∩d->d1->h)∩~∃f(f∈**EV**(F)∩d->f->h)∩~∃h1(h1∈**EV**(H)∩d->h1->h∩**intervalle**(d,h1)>tmax)∩**intervalle**(d,h)>tmax

j) réception avec réponse depuis 1 parmi n :

j1) sans time-out,sur une voie choisie au hasard (RR1/n) : le numéro de la voie réceptrice est indiqué avec le message reçu.
(1) ∀d(d∈**EV**(D)∩P1 #> ∃f(f∈**EV**(F)∩d=>f))
P1 : ∃i∈[1,n],∃t(t∈**EV**(T(i))∩t->d)∩~∃t1(t1∈**EV**(T(i))∩t1->t∩t1.**msg**.no=t.**msg**.no)∩~∃a(a∈**EV**(A(i))∩t->a->d∩a.**msg**.no=t.**msg**.no)
(2) ∀i∈[1,n],∀t(t∈**EV**(T(i))∩P2 #> ∃f(f∈**EV**(F)∩t=>f))
P2 : ∃d(d∈**EV**(D)∩d->t)∩~∃f(f∈**EV**(F)∩d->f->t)∩~∃t1(t1∈**EV**(T(i))∩t1->t∩t1.**msg**.no=t.**msg**.no)
(3) ∀r(r∈**EV**(R)∩P3 <#> ∃a(a∈**EV**(A(i))∩r=>a))
P3 : ∃f(f∈**EV**(F)∩f->r)∩~∃f1(f1∈**EV**(F)∩f->f1->r)∩f.**msg**.novoie=i
(4) ∀d,∀f(d∈**EV**(D)∩f∈**EV**(F)∩d=>f∩P1 #> f.**msg**.v=t.**msg**.v∩f.**msg**.novoie=i)
(5) ∀i∈[1,n],∀t,∀f(t∈**EV**(T(i))∩f∈**EV**(F)∩t=>f #> f.**msg**.v=t.**msg**.v∩f.**msg**.novoie=i)
(6) ∀i∈[1,n],∀r,∀a(r∈**EV**(R)∩a∈**EV**(A(i))∩r=>a∩∃t(t∈**EV**(T(i))∩t->r)∩~∃t1(t1∈**EV**(T(i))∩t->t1->r∩t1.**msg**.no>t.**msg**.no) #> a.**msg**.v'=r.**msg**.v'∩a.**msg**.no=t.**msg**.no)
(7) ∀f(f∈**EV**(F) #> ∃d(d∈**EV**(D)∩P1∩d=>f) v ∃t,∃i∈[1,n](t∈**EV**(T(i))∩P2∩t=>f))
(8) ∀d1,∀d2(d1,d2∈**EV**(D)∩**ord**(d2)=**ord**(d1)+1 #> ∃f(f∈**EV**(F))∩∃r(r∈**EV**(R))∩d1->f->r->d2))
(9) conservation-ordre(X,Y)
(10) non-duplication(X,Y)

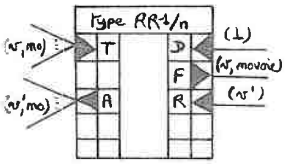


Fig.4.16.

j2) avec time-out,sur une voie choisie au hasard RR1/n(tmax) :

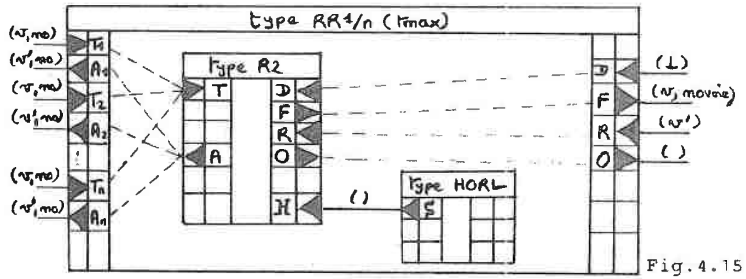
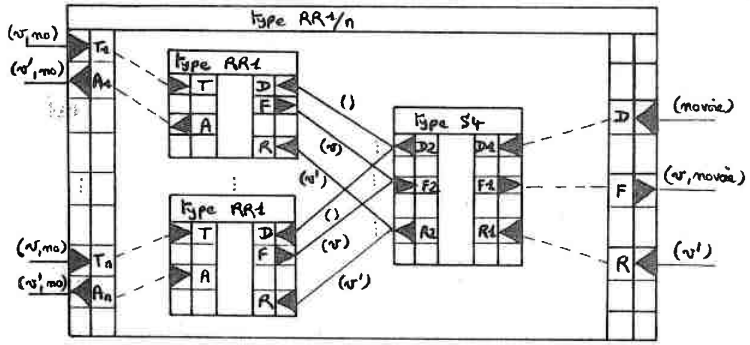


Fig.4.15.

type R2, identique à RR1/n sauf :
(2) ∀t,∀i∈[1,n](t∈**EV**(T(i))∩P2 <#> ∃f(f∈**EV**(F)∩t=>f))
P2 : ∃d(d∈**EV**(D)∩d->t)∩~∃f(f∈**EV**(F)∩d->f->t)∩~∃t1(t1∈**EV**(T(i))∩t1->t∩t1.**msg**.no=t.**msg**.no)∩**intervalle**(d,t)<tmax
(8) ∀d1,∀d2(d1,d2∈**EV**(D)∩**ord**(d2)=**ord**(d1)+1 #> (∃f(f∈**EV**(F))∩∃r(r∈**EV**(R))∩d1->f->r->d2) v ∃o(o∈**EV**(O)∩d1->o->d2)
(11) ∀h(h∈**EV**(H)∩P11 <#> ∃o(o∈**EV**(O)∩h=>o))
P11 : ∃d(d∈**EV**(D)∩d->h)∩~∃d1(d1∈**EV**(D)∩d->d1->h)∩~∃f(f∈**EV**(F)∩d->f->h)∩~∃h1(h1∈**EV**(H)∩d->h1->h∩**intervalle**(d,h1)>tmax)∩**intervalle**(d,h)>tmax

j3) sans time-out, avec sélection d'une voie :

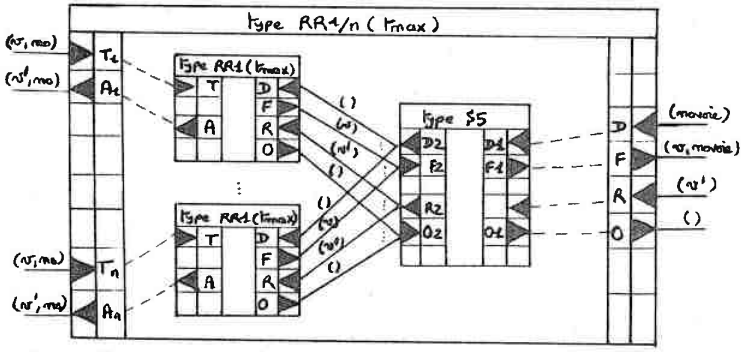
Fig.4.18.



type S4 :
(1) ∀d(d∈EV(D1)∩P1 <#> ∃d1(d1∈EV(D2(i))∩d=>d1))
P1 : d.msg=i
(2) ∀i∈[1,n],∀f(f∈EV(F2(i)) <#> ∃f1(f1∈EV(F1)∩f=>f1))
(3) ∀r(r∈EV(R1)∩P3 <#> ∃r1(r1∈EV(R2(i))∩r=>r1))
P3 : ∃d(d∈EV(D1)∩d->r)∩~∃d1(d1∈EV(D1)∩d->d1->r)∩
d.msg=i
(4) ∀i∈[1,n],∀f,∀f1(f∈EV(F2(i))∩f1∈EV(F1)fe=>f1 #>
f1.msg.v=f.msg.v∩f1.msg.novoie=i)
(5) ∀i∈[1,n],∀r,∀r1(r∈EV(R1)∩r1∈EV(R2(i))∩r=>r1 #>
r1.msg=r.msg)
(6) conservation-ordre(X,Y)
(7) non-duplication(X,Y)

j4) avec time-out, avec sélection d'une voie :

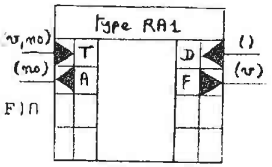
Fig.4.19.



type S5, identique à S4 sauf :
(8) ∀i∈[1,n],∀o(o∈EV(O2(i)) <#> ∃o1(o1∈EV(O1)∩o=>o1))

k) réception avec acquittement depuis 1 :

k1) sans time-out (RA1) :

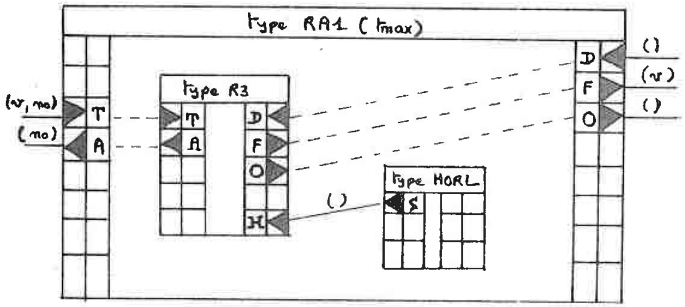


(1) ∀d(d∈EV(D)∩P1 #> (∃f(f∈EV(F)∩d=>f)∩
∃a(a∈EV(A)∩d=>a)))
P1 : ∃t(t∈EV(T)∩t->d)∩~∃t1(t1∈EV(T)∩
t1->t∩t1.msg.no=t.msg.no)∩~∃f1(f1∈EV(F)∩
t->f1->d)
(2) ∀t(t∈EV(T)∩P2 #> (∃f(f∈EV(F)∩t=>f)∩
∃a(a∈EV(A)∩t=>a)))
P2 : ∃d(d∈EV(D)∩d->t)∩~∃f(f∈EV(F)∩
d->f->t)∩~∃t1(t1∈EV(T)∩t1->t∩t1.msg.no=
t.msg.no)
(3) ∀d,∀f,∀a(d∈EV(D)∩f∈EV(F)∩a∈EV(A)∩d=>f∩d=>a∩P1 #>
f.msg=t.msg.v ∩ a.msg=t.msg.no)
(4) ∀t,∀f,∀a(t∈EV(T)∩f∈EV(F)∩a∈EV(A)∩t=>f∩t=>a #> f.msg=
t.msg.v ∩ a.msg=t.msg.no)
(5) ∀f(f∈EV(F) #> ∃d(d∈EV(D)∩P1∩d=>f) v ∃t(t∈EV(T)∩P2
∩t=>f))
(6) ∀a(f∈EV(A) #> ∃d(d∈EV(D)∩P1∩d=>a) v ∃t(t∈EV(T)∩P2
∩t=>a))
(7) ∀d1,∀d2(d1,d2∈EV(D)∩ord(d2)=ord(d1)+1 #> ∃f(f∈EV(F)∩
d1->f->d2))
(8) conservation-ordre(X,Y)
(9) non-duplication(X,Y)

Fig.4.20.

k2) avec time-out (RA1(tmax)) :

Fig.4.21.

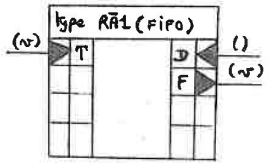


type R3, identique à RA1 sauf :
(2) ∀t(t∈EV(T)∩P2 #> (∃f(f∈EV(F)∩t=>f)∩∃a(a∈EV(A)∩
t=>a)))
P2 : ∃d(d∈EV(D)∩d->t)∩~∃f(f∈EV(F)∩d->f->t)∩~∃t1(t1∈
EV(T)∩t1->t∩t1.msg.no=t.msg.no)∩intervalle(d,t)<tmax
(7) ∀d1,∀d2(d1,d2∈EV(D)∩ord(d2)=ord(d1)+1 #> ∃f(f∈EV(F)
∩d1->f->d2) v ∃o(o∈EV(O)∩d1->o->d2))
(10) ∀h(h∈EV(H)∩P10 <#> ∃o(o∈EV(O)∩h=>o))
P10 : ∃d(d∈EV(D)∩d->h)∩~∃d1(d1∈EV(D)∩d->d1->h)∩~∃f
(f∈EV(F)∩d->f->h)∩~∃h1(h1∈EV(H)∩d->h1->h∩intervalle
(d,h1)>tmax)∩intervalle(d,h)>tmax

1) réception sans acquittement depuis 1 :

11) sans time-out et tampon géré en FIFO (RÃ1(FIFO)) :

Fig.4.22.



(1) ∀d(d∈**EV**(D)∩P1 #> ∃f(f∈**EV**(F)∩d=>f))
P1 : ∃t(t∈**EV**(T)∩**ord**(t)≥**ord**(d))
(2) ∀t(t∈**EV**(T)∩P2 #> ∃f(f∈**EV**(F)∩t=>f))
P2 : ∃d(d∈**EV**(D)∩d->t)∩~∃f(f∈**EV**(F)∩d=>f)∩~∃t1(t1∈**EV**(T)∩d->t1->t∩t1=>f)
(3) ∀t,∀f(t∈**EV**(T)∩f∈**EV**(F)∩t=>f #> f.**msg**=t.**msg**)
(4) ∀d,∀f(d∈**EV**(D)∩f∈**EV**(F)∩d=>f∩∃t(t∈**EV**(T)∩**ord**(t)=**ord**(f) #> f.**msg**=t.**msg**)
(5) ∀f(f∈**EV**(F) #> ∃d(d∈**EV**(D)∩P1∩d=>f) v ∃t(t∈**EV**(T)∩P2∩t=>f))
(6) ∀d1,∀d2(d1,d2∈**EV**(D)∩**ord**(d2)=**ord**(d1)+1 #> ∃f(f∈**EV**(F)∩d1->f->d2))
(7) **conservation-ordre**(X,Y)
(8) **non-duplication**(X,Y)

12) sans time-out et tampon géré en LIFO (RÃ1(LIFO)) :
identique à RA1(FIFO) sauf :
(4) ∀d,∀f(d∈**EV**(D)∩f∈**EV**(F)∩d=>f∩∃t1(t1∈**EV**(T)∩t1->d)∩~∃t2(t2∈**EV**(T)∩t1->t2->d)∩∃t(t∈**EV**(T)∩**ord**(t)=**maximum-unique**(1,**ord**(t1)) #> f.**msg**=t.**msg**)
où maximum-unique(1,n) donne le plus grand entier entre 1 et n non encore choisi auparavant.

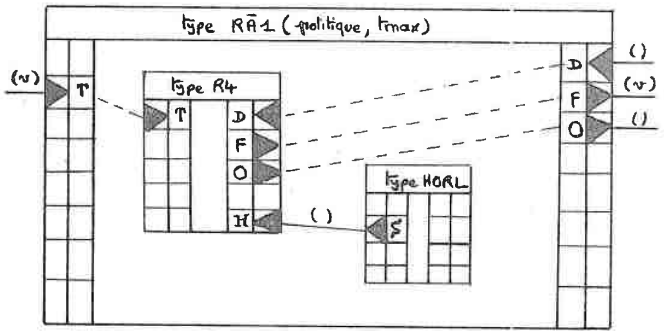
13) sans time-out et tampon géré au hasard (RÃ1(HASARD)) :
identique à RA1(FIFO) sauf :
(4) ∀d,∀f(d∈**EV**(D)∩f∈**EV**(F)∩d=>f∩∃t1(t1∈**EV**(T)∩t1->d)∩~∃t2(t2∈**EV**(T)∩t1->t2->d)∩∃t(t∈**EV**(T)∩**ord**(t)=**hasard-unique**(1,**ord**(t1)) #> f.**msg**=t.**msg**)
où hasard-unique(1,n) donne un entier au hasard entre 1 et n non encore choisi auparavant.

14) sans time-out et tampon géré en dernier reçu avec remise (RÃ1(DRR)) :
identique à RA1(FIFO) sauf :
(1) ∀d(d∈**EV**(D)∩P1 #> ∃f(f∈**EV**(F)∩d=>f))
P1 : ∃t(t∈**EV**(T)∩t->d)
(4) ∀d,∀f(d∈**EV**(D)∩f∈**EV**(F)∩d=>f∩∃t(t∈**EV**(T))∩~∃t1(t1∈**EV**(T)∩t->t1) #> f.**msg**=t.**msg**)

15) sans time-out et tampon géré en dernier reçu sans remise (RÃ1(DR̄R̄)) :
identique à RA1(FIFO) sauf :
(1) ∀d(d∈**EV**(D)∩P1 #> ∃f(f∈**EV**(F)∩d=>f))
P1 : ∃t(t∈**EV**(T)∩t->d)∩~∃t1(t1∈**EV**(T)∩t->t1->d)∩~∃f1(f1∈**EV**(F)∩t->f1->d)
(4) ∀d,∀f(d∈**EV**(D)∩f∈**EV**(F)∩d=>f∩∃t(t∈**EV**(T))∩~∃t1(t1∈**EV**(T)∩t->t1) #> f.**msg**=t.**msg**)

16) avec time-out et quelque soit la politique de gestion du tampon (RÃ1(FIFO ou LIFO ou HASARD ou DRR̄, tmax)) :

Fig.4.23.



type R4, identique au type correspondant sans time-out, sauf :
(2) ∀t(t∈**EV**(T)∩P2 #> ∃f(f∈**EV**(F)∩t=>f))
P2 : ∃d(d∈**EV**(D)∩d->t)∩~∃f(f∈**EV**(F)∩d=>f)∩~∃t1(t1∈**EV**(T)∩d->t1->t)∩**intervalle**(d,t)<tmax
(6) ∀d1,∀d2(d1,d2∈**EV**(D)∩**ord**(d2)=**ord**(d1)+1 #> (∃f(f∈**EV**(F)∩d1->f->d2) v (∃o(o∈**EV**(O)∩d1->o->d2))
(9) ∀h(h∈**EV**(H)∩P9 <#> ∃o(o∈**EV**(O)∩h=>o))
P9 : ∃d(d∈**EV**(D)∩d->h)∩~∃d1(d1∈**EV**(D)∩d->d1->h)∩~∃f(f∈**EV**(F)∩d->f->h)∩~∃h1(h1∈**EV**(H)∩d->h1->h∩**intervalle**(d,h1)>tmax)∩**intervalle**(d,h)>tmax

m) réception avec acquittement depuis 1 parmi n :

m1) sans time-out, sur une voie choisie au hasard (RA1/n) :

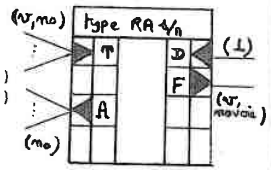
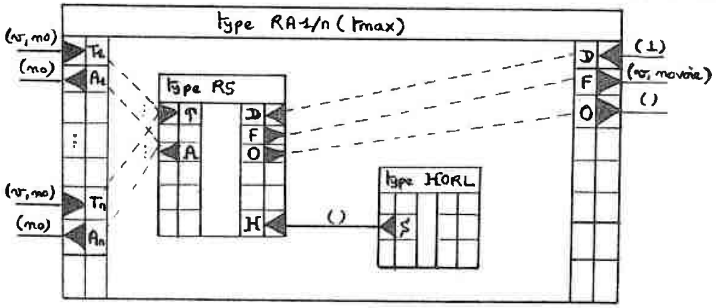


Fig.4.24.

(1) ∀d(d∈EV(D)∩P1 #> ∃f(f∈EV(F)∩d=>f)∩
∃a(a∈EV(A(i))∩d=>a))
P1 : ∃i∈[1,n],∃t(t∈EV(T(i))∩t->d)∩
~∃t1(t1∈EV(T(i))∩t1->t∩t1.msg.no=t.msg.no)
∩~∃a(a∈EV(A(i))∩t->a->d∩a.msg.no=t.msg.no)
(2) ∀i∈[1,n],∀t(t∈EV(T(i))∩P2 #> ∃f(f∈EV(F)∩
t=>f)∩∃a(a∈EV(A(i))∩t=>a))
P2 : ∃d(d∈EV(D)∩d->t)∩~∃f(f∈EV(F)∩
d->f->t)∩~∃t1(t1∈EV(T(i))∩t1->t∩t1.msg.no=
t.msg.no)
(3) ∀i∈[1,n],∀d,∀f,∀a(d∈EV(D)∩f∈EV(F)∩a∈EV(A(i))∩d=>f∩
d=>a∩P1 #>f.msg=t.msg.v∩f.msg.novoie=i∩a.msg=t.msg.no)
(4) ∀i∈[1,n],∀t,∀f,∀a(t∈EV(T(i))∩f∈EV(F)∩a∈EV(A(i))∩
t=>f∩t=>a #> f.msg.v=t.msg.v∩f.msg.novoie=i∩a.msg=
t.msg.no)
(5) ∀f(f∈EV(F) #> ∃d(d∈EV(D)∩P1∩d=>f) v ∃i∈[1,n],∃t(t∈
EV(T(i))∩P2∩t=>f))
(6) ∀i∈[1,n],∀a(f∈EV(A(i)) #> ∃d(d∈EV(D)∩P1∩d=>a) v ∃i∈
[1,n],∃t(t∈EV(T)∩P2∩t=>a))
(7) ∀d1,∀d2(d1,d2∈EV(D)∩**ord**(d2)=**ord**(d1)+1 #> ∃f(f∈EV(F)∩
d1->f->d2))
(8) conservation-ordre(X,Y)
(9) non-duplication(X,Y)

m2) avec time-out, sur une voie choisie au hasard (RA1/n(tmax)) :

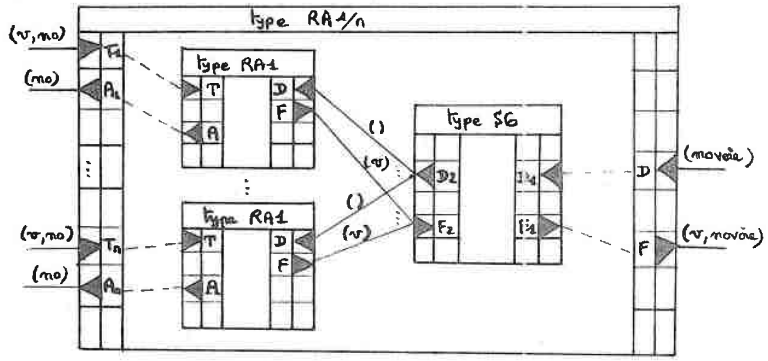
Fig.4.25.



type R5, identique à RA1/n sauf :
(2) ∀i∈[1,n],∀t(t∈EV(T(i))∩P2 <#> ∃f(f∈EV(F)∩t=>f)∩∃a(a∈
EV(A(i))∩t=>a))
P2 : ∃d(d∈EV(D)∩d->t)∩~∃f(f∈EV(F)∩d->f->t)∩~∃t1(t1∈
EV(T(i))∩t1->t∩t1.msg.no=t.msg.no)∩**intervalle**(d,t)<tmax
(7) ∀d1,∀d2(d1,d2∈EV(D)∩**ord**(d2)=**ord**(d1)+1 #> ∃f(f∈EV(F)
∩d1->f->d2) v ∃o(o∈EV(O)∩d1->o->d2))
(10) ∀h(h∈EV(H)∩P10 <#> ∃o(o∈EV(O)∩h=>o))
P10 : ∃d(d∈EV(D)∩d->h)∩~∃d1(d1∈EV(D)∩d->d1->h)∩~∃f
(f∈EV(F)∩d->f->h)∩~∃h1(h1∈EV(H)∩d->h1->h∩**intervalle**
(d,h1)>tmax)∩**intervalle**(d,h)>tmax

m3) sans time-out, avec sélection d'une voie :

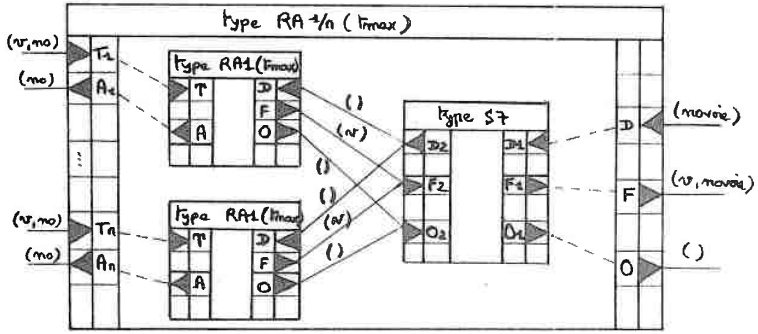
Fig.4.26.



type S6 :
(1) ∀d(d∈EV(D1)∩P1 <#> ∃d1(d1∈EV(D2(i))∩d=>d1))
P1 : d.msg=i
(2) ∀i∈[1,n],∀f(f∈EV(F2(i)) <#> ∃f1(f1∈EV(F1)∩f=>f1))
(3) ∀i∈[1,n],∀f,∀f1(f∈EV(F2(i))∩f1∈EV(F1)∩f=>f1 #>
f1.msg.v=f.msg.v∩f1.msg.novoie=i)
(4) conservation-ordre(X,Y)
(5) non-duplication(X,Y)

m4) avec time-out, avec sélection d'une voie :

Fig.4.27.

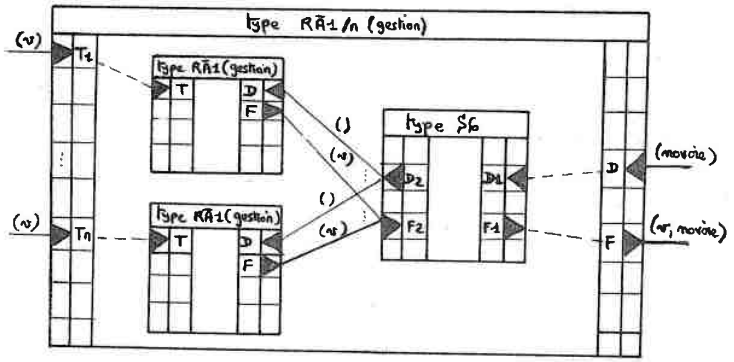


type S7, identique à S6 sauf :
(6) ∀i∈[1,n],∀o(o∈EV(O2(i)) <#> ∃o1(o1∈EV(O1)∩o=>o1))

n) réception sans acquittement depuis 1 parmi n :

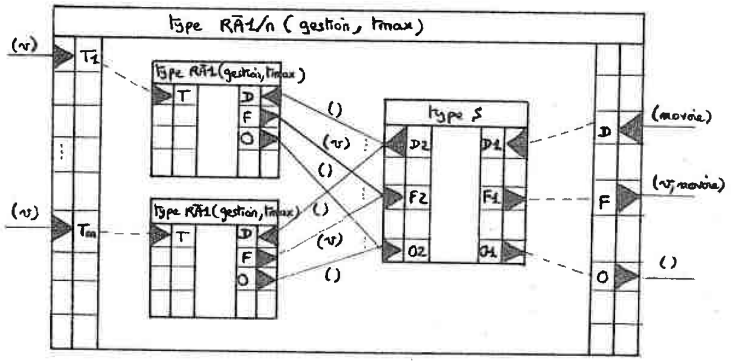
n1) sans time-out, avec sélection d'une voie (RĀ1/n) :
La gestion du tampon peut être FIFO, LIFO, HASARD, DRR ou DRR̄.

Fig.4.28.



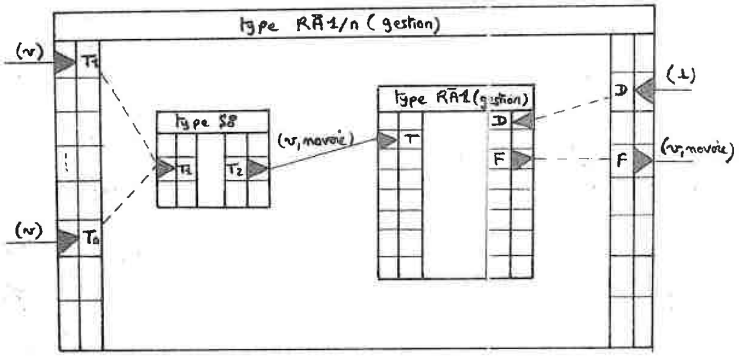
n2) avec time-out, avec sélection d'une voie (RĀ1/n(tmax)) :
La gestion du tampon peut être FIFO, LIFO, HASARD ou DDR̄.

Fig.4.29.



n3) sans time-out, sur une voie choisie au hasard :
La gestion du tampon peut être FIFO, LIFO, HASARD, DDR ou DDR̄.

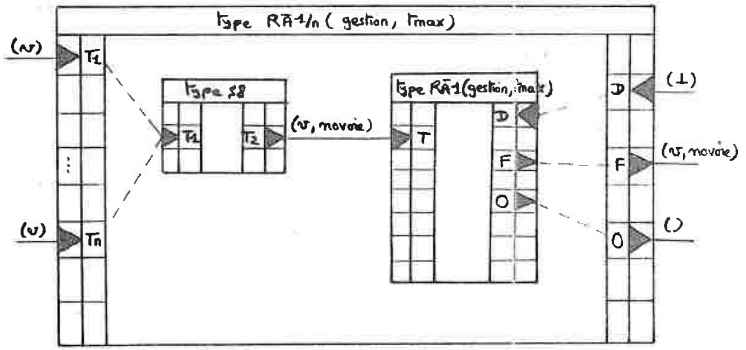
Fig.4.30.



type S8 :
(1) ∀i∈[1,n],∀t(t∈**EV**(T1(i)) <#> ∃t1(t1∈**EV**(T2)∩t=>t1))
(2) ∀i∈[1,n],∀t,∀t1(t∈**EV**(T1(i))∩t1∈**EV**(T2)∩t=>t1 #>
t1.**msg**.v=t.**msg** ∩ t1.**msg**.novoie=i)
(3) conservation-ordre(X,Y)
(4) non-duplication(X,Y)

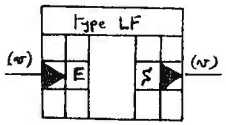
n4) avec time-out, sur une voie choisie au hasard :
La gestion du tampon peut être FIFO, LIFO, HASARD ou DRR̄.

Fig.4.31.



o) spécification des types de liaisons :

Fig.4.32.



Le type "liaison fiable" (LF), est défini par :

(1) Ve(eεEV(E) <#>]s(sεEV(S)∩e=>s)]
(2) Ve,Vs(eεEV(E)∩sεEV(S)∩e=>s #>
s.msg=e.msg)
(3) conservation-ordre(E,S)
(4) non-duplication(E,S)

La définition d'une liaison "non fiable" a été donnée chapitre 3 § 2.2.2. (entité L1).

p) spécification d'un type de communication : nous reprenons à titre d'exemple le type décrit au début de ce § 1.2.4. :

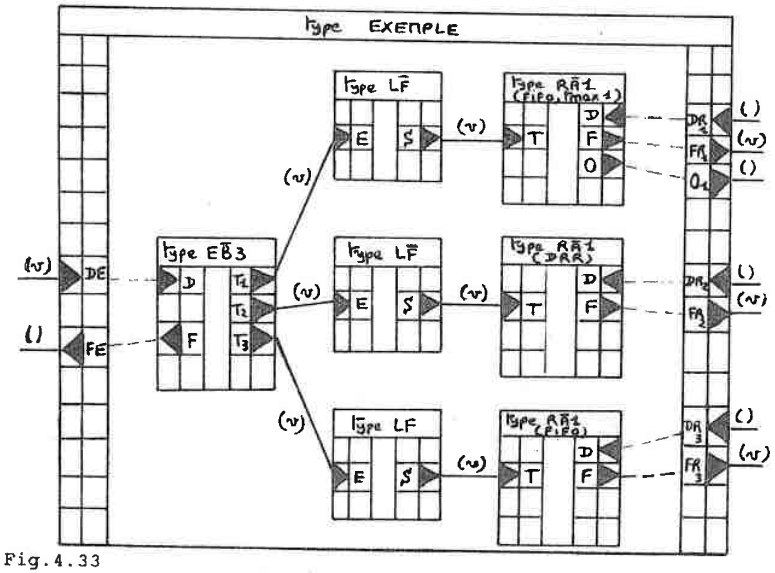


Fig.4.33

Cette forme de spécification nous paraît intéressante par :
- sa modularité,
- la possibilité de raffiner certaines entités en implantations "proches de la mise en oeuvre" dont on peut valider la cohérence vis à vis des propriétés de l'entité de départ (c'est ce que nous avons fait au chapitre 3 § 2.2.2. pour l'implantation d'une liaison fiable à l'aide de liaisons non fiables et d'un protocole adéquat),
- l'unicité du formalisme pour la spécification des entités et des "systèmes de communication" qui les relient, porteuse de promesses pour la définition d'un langage de spécification abstrait au niveau détaillé.

2. STRATEGIES POUR LA CONCRETISATION DE L'OUTIL DE DESCRIPTION DETAILLEE :

Le langage de spécification des comportements, nous apparaît peu satisfaisant comme moyen d'expression à ce niveau : il permet bien de décrire les processus séquentiels, leurs communications (cf. paragraphe précédent) et des objets partagés par les processus (en tant qu'entités, les règles d'accès étant exprimées comme des propriétés sur les communications "matérialisant" ces accès); par contre sa lourdeur, déjà sensible pour spécifier les types de communication, et son inaptitude à rendre compte d'aspects fondamentaux pour les applications de commande de procédés, comme les interruptions ou les exceptions, sont gênants. C'est pourquoi, nous estimons qu'une description "programmatoire" est mieux adaptée à ce niveau.

Le vecteur d'une telle description peut être soit un langage de programmation de "haut niveau", soit un pseudo-code spécifique dont on trouve de nombreux exemples dans le domaine de la commande de procédés (ex: ESPRESO [LUD83], SPECTRE [MOI81], [CAL82]).

Il n'existe pas de langage de programmation de haut niveau largement diffusé qui intègre le modèle de structuration que nous avons retenu. Les langages qui s'en approchent, tels CONIC, MARS, ARGUS ou NIL, sont des prototypes. Il n'est pas dans nos objectifs (et nos moyens) de créer de toutes pièces un nouveau langage prototype. La voie la plus immédiate consiste à se tourner vers le langage ADA [ICH79], souvent étudié sous l'angle de la spécification au niveau du "programming-in-the-large" [PYL84, BOO81, ...].

Nous examinons donc brièvement les deux solutions qui consistent à recourir au langage ADA ou à utiliser un pseudo-code "ad hoc".

2.1. EMPLOI DU LANGAGE ADA, COMME LANGAGE DE SPECIFICATION DETAILLEE :

Le langage ADA ne s'inscrit pas a priori dans la catégorie des langages de "programmation répartie", car le modèle qui le sous-tend suppose une mémoire commune partageable par toutes les entités actives de l'application [STA85]. Cependant, une abondante littérature envisage son utilisation dans un tel contexte réparti, selon diverses approches :

- écriture d'un programme ADA par site, l'exécutif réparti se chargeant des interactions entre ces programmes; les inconvénients sont nombreux :
  - . la communication inter sites est moins sûre que la communication à l'intérieur des programmes,
  - . la conception est liée très tôt à une configuration matérielle donnée,

. tout changement de configuration matérielle ou toute modification d'allocation entraîne une refonte importante des programmes,
. on ne peut tester l'application sur une configuration mono site, etc.

- écriture d'un programme ADA unique, structuré comme un ensemble d'entités (tâches [SCH81]) ou "noeuds virtuels" [DAR84]), interagissant exclusivement par appels d'entrées; l'exécutif réparti gère les appels d'entrées à distance. Les inconvénients tiennent aux fortes limitations d'usage apportées au langage (ni variables, ni procédures communes, ni passage de pointeurs, etc.). Par contre, l'utilisateur peut s'abstraire de la configuration matérielle cible.

- utilisation de systèmes automatisés de "répartition du code source"; dans [ARM85], un préprocesseur ("automated software distributor"), modifie un programme ADA quelconque pour l'adapter à une répartition donnée. Dans [COR83, 84], un programme complémentaire écrit dans un langage de répartition ("ADA Program Partitionning Language") s'ajoute au programme ADA et un compilateur spécial ("distributed ADA compiler") traite les deux descriptions pour produire un code exécutable sur un environnement spécifique ("distributed ADA runtime system"). Les affectations d'entités définies par le programme de répartition peuvent être multiples (gestion de copies multiples) et dynamiques (objectif de "survivabilité" en cas de pannes).

Dans notre contexte, il s'agit de décrire une application en conservant un certain degré d'abstraction (ex: spécification de sous-programmes et de structures de données sans les corps correspondants), en rendant compte des choix structurels (modules, processus, ports, liaisons, etc.), des types de communication, des aspects "temps-réel" définis au premier paragraphe de ce chapitre. Ce peut être fait, en respectant un certain style de programmation et des restrictions d'usage, selon la seconde approche évoquée ci-dessus.

a) expression des choix structurels :

A chaque module on associe un paquetage ("package"); à chaque processus, une tâche. Les types globaux à toute l'application (ex: types de messages des liaisons entre modules) figurent dans un (ou plusieurs) paquetage(s), importé(s) dans les "paquetages modules" grâce à la clause **with** :

with <nom(s) paquetage(s) des types globaux>
package <nom module>
    <déclarations objets partagés par les processus>
    <déclarations des processus (tâches)>
end <nom module>.

Les notions de port et de liaison n'ont pas de contrepartie directe en ADA : dans un échange, la tâche appelante désigne explicitement la tâche appelée (l'inverse n'étant pas vrai). Nous proposons de localiser ces références dans des "unités interface" (procédure ou tâche), assimilables aux ports, préservant ainsi une certaine indépendance entre les tâches de l'application [MOR83] :

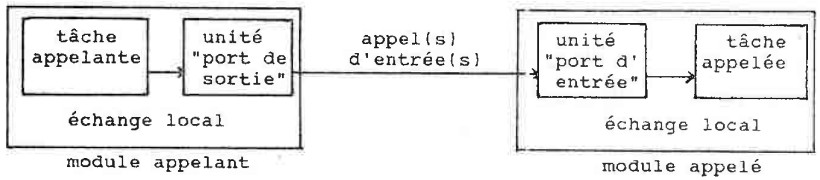


Fig.4.34.

b) expression des types de communication :

Il s'agit des communications entre modules à réaliser exclusivement à l'aide d'appels d'entrées. Nous donnons à l'annexe II, les schémas de traduction des types d'émission et de réception selon l'organisation de la figure 4.34. et en suivant la nomenclature du paragraphe 1.2.3. Les quelques difficultés à résoudre concernent :

- les émissions non bloquantes (cf. Annexe II, § a),
- la prise en compte des possibilités de perte et de duplication des informations transmises (cf. Annexe II, § d),
- la notion de délai d'attente sur les opérations bloquantes : en effet, le délai maximum d'attente d'un appel d'entrée temporisé en ADA concerne la prise en compte de l'entrée et non la fin du traitement associé, par l'appelé; ceci rend possible un blocage de l'appelant à la suite d'une défaillance de l'appelé, ce qui contredit notre hypothèse d'autonomie des modules. Il est possible de contourner cette difficulté en faisant appel aux propagations explicites d'exceptions (cf. Annexe II, § d).

c) l'expression des aspects temps-réel et des aspects divers :

Les priorités statiques sur les ports d'entrée et les priorités statiques associées aux processus peuvent être rendues respectivement au niveau du choix indéterministe (cf. [LEV82]) et grâce au pragma **priority**.

Par l'instruction **for...use at...**, il est possible d'associer des entrées à des interruptions externes (dispositifs physiques ou programmes de bas niveau d'interfaçage) et de programmer des processus d'interruption. Certains mécanismes comme le blocage de tous les processus normaux du module et leur réinitialisation doivent être gérés au niveau des programmes de bas niveau d'interfaçage.

Les exceptions non récupérées dans le corps des tâches ADA entraînent leur arrêt (passage à l'état accompli). Pour programmer des processus permanents, il est nécessaire de récupérer toutes les exceptions par une clause **others**, associée au bloc inclus dans la boucle infinie de chaque processus.

d) évaluation :

Le modèle retenu pour les applications que nous visons est beaucoup plus simple que celui qui sous-tend le langage ADA. Il en découle une impressionnante liste de règles et de restrictions d'usage concernant la structuration en tâches et leur gestion, leurs interactions, l'utilisation de nombreux mécanismes tels les exceptions ou les pointeurs. On peut juger préférable, même dans le cas d'une programmation en ADA, d'introduire un pseudo-code cohérent avec le modèle retenu et à partir duquel sera généré le squelette du code définitif.

2.2. EMPLOI D'UN PSEUDO-CODE SPECIFIQUE - LE "LANGAGE DE SPECIFICATION DETAILLE" (LSD) :

a) Caractéristiques :

Nous donnons à l' Annexe III une proposition de définition d'un tel langage. Il a été conçu :

- pour focaliser l'attention sur l'essentiel et laisser de côté l'accessoire (ex : absence d'arithmétique, comme dans [LUD82]),
- pour faciliter une mise en oeuvre en ADA; tous les concepts peuvent se traduire en ADA (nous l'avons déjà montré pour les caractéristiques principales du modèle),
- en reprenant un maximum de constructions linguistiques usuelles dans les langages récents,
- en prévoyant de manière systématique des paragraphes de description libre (en langue naturelle ou dans des formalismes au choix).

b) discussion :

Le LSD ne doit pas apparaître comme un niveau supplémentaire de langage, compliquant la démarche, mais au contraire comme un moyen de simplifier celle-ci en fournissant un moyen d'expression adéquat. Le concepteur utilise différents moyens, adaptés à chaque phase et à la classe de problèmes visée :

- l'expression graphique pour définir l'organisation structurelle de l'application,
- le langage de spécification abstrait des comportements pour conduire les validations formelles,
- le LSD, pour décrire les traits essentiels de la structuration interne des modules

Reste, à l'issue de la conception, une phase de programmation complémentaire dans le langage cible, accessible à un "programmeur PASCAL standard" (pour l'essentiel, écriture

d'implantations de structures de données et d'algorithmes purement locaux à un site).

Par ailleurs, nous évoquerons dans la suite l' intérêt que peut présenter la construction de maquettes de simulation exécutables en pseudo-parallélisme sur un système classique. Le LSD peut servir également de point de départ à la génération du squelette de programme dans le langage de simulation choisi.

Enfin, nous avons pu réfléchir, sur la base du LSD, aux outils de manipulation et de contrôle des spécifications qui pourraient être proposés au concepteur dans le prolongement de ceux décrits au chapitre 3 § 2.3.2.

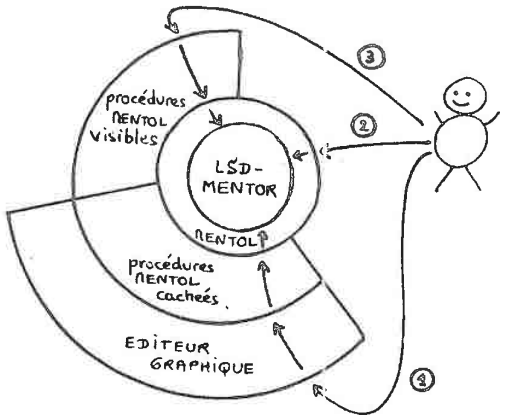
c) l'outil de gestion des spécifications :

L'intégration de l'éditeur/vérificateur graphique des structures comme une couche interface d'un éditeur syntaxique LSD-MENTOR a été réalisée (sous MULTICS au CIRIL) [LAL87] - cf. Annexe IV -. Le concepteur dispose donc d'un éditeur/vérificateur mixte (graphique et textuel) avec lequel il peut travailler de trois manières :

(1) en utilisant les commandes de l'éditeur graphique; c'est le seul moyen qui lui est donné de spécifier les aspects structuraux;
(2) en utilisant le langage de manipulation d'arbres MENTOL, pour la spécification détaillée;
(3) en utilisant des procédures MENTOL prédéfinies, qui facilitent la définition et la manipulation de certaines entités structurées de la spécification détaillée.

Le passage de la représentation graphique aux arbres syntaxiques se fait grâce à des procédures MENTOL cachées de l'utilisateur et dont l'éditeur graphique génère des appels, dans des fichiers de liaison. Ce passage à la spécification détaillée n'est autorisé que si l'application apparaît complète (c'est à dire si la commande de contrôle différé n'a pas détecté d'erreurs).

Fig. 4.35.



Plus précisément l'éditeur graphique peut fonctionner selon trois modes :

- le mode "création", lors de la création initiale de l'application; une commande **compiler** permet de compiler la structure complète vérifiée en un arbre MENTOR équivalent;
- le mode "mise à jour globale", dans lequel on passe dès chargement d'une application préexistante; l'arbre syntaxique est modifié après un ensemble de commandes de modification "contrôlables". En effet les commandes graphiques sont soit :
  + sans effet sur la spécification détaillée (ex: commandes de déplacement),
  + avec effet "contrôlable", c'est à dire directement interprétable (ex: créer-liaison, supprimer-port)
  + avec effet considéré comme "incontrôlable", car différé; c'est le cas de la seule commande **ailleurs** (cf. Chap. 3 § 2.3.2.) qui ouvre une nouvelle hiérarchie de structures devant fusionner ultérieurement à celle existant déjà.
- le "mode mise à jour locale", purement graphique; l'application devra alors être compilée ou recompilée ultérieurement; on y passe à la demande ou en cas d' utilisation de la commande **ailleurs** lors d'une mise à jour.

L'éditeur textuel fonctionne selon trois modes homologues :

- le mode "création", par exécution du fichier de liaison produit par l'éditeur graphique lors d'une compilation;
- le mode "mise à jour globale", par exécution du fichier de liaison produit par l'éditeur graphique après mise à jour d'une application préexistante;
- le mode "mise à jour locale", lorsque le concepteur travaille sur les aspects de la spécification détaillée non contrôlés par l'éditeur graphique . De très nombreux contrôles immédiats et différés relatifs à la sémantique du LSD sont réalisés dans ce mode.

Concluons, en remarquant que le fait d'interdire toute modification structurelle en dehors de l'éditeur graphique assure le maintien de la cohérence entre la représentation graphique et la spécification détaillée. La représentation graphique apparaît alors comme une **"vue externe"** hiérarchisée ("en 3-D"), documentant le programme, dont le texte n'est pas alourdi par la description de cette structure hiérarchique établie lors de la conception ("en 2-D").

3. MODE D'EMPLOI DE L'OUTIL DE DESCRIPTION DETAILLEE :

3.1. REGLES DE STRUCTURATION INTERNE DES MODULES :

Le concepteur dispose, pour chaque module, d'une décomposition en agents élémentaires séquentiels obtenue à la phase de structuration logique (chapitre 3).

Plusieurs facteurs peuvent l'inciter à modifier ce découpage. En particulier :

- le parallélisme de conception : pour des raisons d'efficacité de traitement (parallélisation) ou de modularité (taille des processus), le concepteur peut décider de scinder un agent élémentaire en un réseau de processus. Le parallélisme de conception peut également impliquer, à l'inverse, une diminution du parallélisme décrit au niveau logique; dans ce cas, le concepteur estime que la fusion d'agents élémentaires (et souvent la sérialisation de communications externes non temporellement exclusives) est acceptable et permet d'obtenir une structure plus simple et raisonnablement efficace;

- l'introduction de processus d'interruption (et éventuellement de processus à réinitialiser après interruption), qui n'avait bien entendu pas été envisagée au niveau logique;

- la prise en compte des contraintes temporelles critiques qui touchent certaines fonctionnalités et qui peuvent entraîner leur isolement dans des processus qui recevront des priorités élevées ([GOM84]);

- le couplage avec des périphériques d'entrée ou de sortie : ceux-ci évoluant à des vitesses qui leur sont propres, il est préférable que des processus spécifiques assurent les interfaçages ([GOM84]);

- l'introduction de processus de "service" pour réaliser, par exemple, la gestion de ressources et discipliner les accès à celles-ci.

En limitant au strict nécessaire ces transformations, nous pensons que l'on préservera la lisibilité de l' application.

Au contraire de la phase de structuration organique, l'introduction de redondances des communications et des entités ne se pose pas en général à ce niveau; en effet, il est question ici de traitements et de communications locales à un site : les communications y sont beaucoup plus sûres et les redondances locales de traitements ont peu d'interêt.

Les choix des types de communication doivent se faire en parallèle avec le découpage en processus , car la

connaissance des interlocuteurs effectifs est un paramètre de choix important. Les critères d'efficacité et de sûreté sont à considérer : ils s'avèrent par nature contradictoires, comme dans le cas du choix entre émission non bloquante et émission bloquante en attente d'acquittement. L'expérience du concepteur joue ici un rôle prépondérant pour apprécier les conséquences prévisibles de telle ou telle option.

3.2. VALIDATION DE LA COHERENCE DE LA SPECIFICATION DETAILLEE VIS A VIS DES PROPRIETES COMPORTEMENTALES :

Nous présentons deux approches partielles qui nous semblent susceptibles d'être effectivement pratiquées par les concepteurs.

3.2.1. Selon la structure de la spécification détaillée :

Lorsqu'un module est implanté exclusivement sous la forme de processus normaux interagissant par des communications, c'est à dire sans faire appels aux mécanismes pour lesquels nous ne disposons pas de contrepartie dans le modèle de spécification des comportements, il est possible d'utiliser la technique classique de validation du comportement du réseau de processus au regard du comportement du module. Puis la spécification formelle du comportement de chaque processus peut être rapprochée du schéma de contrôle de ce processus réduit aux actions de communication et aux choix (conditionnelles, choix indéterministes). Beaucoup de propriétés classiques (déclenchements inconditionnels ou conditionnels, ensemble des causalités possibles, propriétés sur les valeurs, etc.) peuvent être vérifiés par cette simple confrontation avec les schémas de contrôle ou avec la spécification détaillée.

Exemple : un déclenchement inconditionnel sera vérifié si la réception et l'émission correspondantes sont dans la même branche ou sous-branche du schéma de contrôle.

♦

On trouvera à l'annexe V, une validation selon cette approche de la spécification détaillée du module de contrôle de la pompe, proposé par KRAMER (cf. § 2.2.1).

3.2.2. Par analyse des traces d'événements à l'interface des modules :

Dans les cas plus complexes, il est possible de recourir à une autre technique, beaucoup plus lourde, qui consiste à simuler le fonctionnement de l'application (ou du module) et à enregistrer la trace des événements à l'interface ([ECA85]). L'analyse de ces traces permet de détecter d'éventuels comportements ne respectant pas les

propriétés requises et d'apporter, en leur absence, un certain degré de confiance dans la solution retenue. Il faut :

- simuler le comportement de l'environnement de l'application (ou du module) étudié,

- traduire la spécification détaillée dans un langage se prêtant à la simulation (en pseudo-parallélisme sur une configuration classique pour plus de simplicité).

On trouvera dans [GOF84], un travail mené au CRIN et dont certains aspects pourraient être repris dans cette optique : il s'agit de la construction en SIMONE [BEZ76], de maquettes de simulation d'applications structurées selon un modèle semblable au nôtre à quelques détails près, essentiellement en vue d'étudier les conséquences des défaillances possibles des sites et des liaisons.

Le travail de l'utilisateur pourrait être sensiblement allégé par la génération automatique, à partir de la spécification détaillée en LSD, du squelette de l'application dans le langage de simulation (comme pour la production du code définitif).

BIBLIOGRAPHIE :

[ARM85] ARMITAGE, CHELINI
ADA software on distributed targets : a survey of approaches
ADA Letters, Vol 4, no 4, 2/85 (pp 32,37)

[BEZ76] BEZIVIN, KAUBISCH, LEROY, NEUT, RANNOU
SIMONE : manuel de référence.
Rapport SFER/IRIA, 1976

[BOO81] BOOCH G.
Describing software design in ADA.
ACM SIGPLAN notices, 1981

[BRI78] B. HANSEN
Distributed processes : a concurrent programming concept.
CACM, Vol 21, no 11, 11/78 (pp 934, 941)

[CAL82] CALVEZ J.P.
Une méthodologie de conception des systèmes multi-micro-ordinateurs pour les applications de commande en temps réel.
Thèse d'Etat, NANTES, 11/82

[CAR82] CARSON G.S.
Message-based distributing computing.
Real-time Syst. Symp., LOS ANGELES, 12/82 (pp 170, 183)

[COR83] CORNHILL D.
A survivable distributed computing system for embedded application programs written in ADA.
ADA Letters, 11-12/83

[COR84] CORNHILL D.
Partitioning ADA programs for execution on distributed systems.
Honeywell Syst. and Research Center, MINNEAPOLIS, 1984

[DAR84] DARPRA, MADERNA, STAMMERS
Using ADA and APSE to support distributed multi-micro-processor targets.
ADA Letters, Vol 3, no 6, 5-6/84

[DPT82] DERNIAME, PERRIN, THOMESSE
Communication based design of distributed systems.
Rapport 82-R-022, CRIN, 1982

[DRE75] DE REMER, KRON
Programming-in-the-large versus programming-in-the-small.
IEEE Trans.on Soft.Eng., Vol SE-2, no 2, 6/76 (pp 80,86)

[ECA75] ECAULT C.
Une expérience dans la spécification et la validation des systèmes distribués.
Thèse 3ème cycle, RENNES 1, 1985

[FEL79] FELDMAN J.A.
High level programming for distributed computing.
CACM, Vol 22, no 6, 6/79 (pp 353,368)

[GOF84] GOFFIC P.
Architecture de logiciels répartis. Spécification et simulation.
Thèse Doct. Ing., NANCY I, 6/84

[GOM84] GOMAA H.
A software design method for real-time systems.
CACM, Vol 27, no 9, 1984 (pp 938,949)

[HEW77] HEWITT, ATKINSON
Paralelism and synchrnization in actors systems.
Proc. 4th ACM Symp. on Principles of Prog. Lang., 1/77 (pp 267,280)

[ICH79] ICHBIAH et ALL
Preliminary ADA reference manual and Rationale for the design of the ADA programming language.
SIGPLAN Notices, Vol 14, no 16, 1979

[KOP82] KOPETZ, LOHNERT, MERKER, PAUTHNER
High level programming of distributed process control systems.
Real-time data'82, VERSAILLES, 11/82 (pp 33,40)

[KRA81] KRAMER, MAGEE, SLOMAN
Intertask communication primitives for DCCS.
Proc 2nd Int.Conf.on Distr.Computing Syst., PARIS, 4/81 (pp 404,411)

[LAL87] LALLIER M.
Thèse 3ème cycle, NANCY I, à paraître, 1987

[LEV82] LEVERRAND D.
Le langage ADA - manuel d'évaluation.
DUNOD, 1982

[LUD83] LUDEWIG J.
ESPRESO: a system for process control software specification
IEEE Trans.on Soft.Eng., Vol SE-9, no 4, 7/83 (pp 427,436)

[MAC82] MAC LEOD, RODD
Interprocess communication primitives for distriuted process control.
Proc.IFAC Soft.for process control, MADRID, 10/82 (pp 51,59)

[MOI81] MOISAN S.
SSP : l'analyseur de projets.
Thèse Doct. Ing., TOULOUSE UPS, 11/81

[MOR83] MORETON T.
The programming of CONIC systems in ADA.
RR doc 83/14, Imperial College, LONDON, 8/83

[PER85] PERRIN G.R.
La communication : un outil pour la spécification, la construction et la vérification de systèmes parallèles.
Thèse d'Etat, NANCY I, 10/85

[PYL84] PYLE I.
Limits on the use of ADA for specifications.
Proc.3th Joint ADA Europe/ADA Tec Conf., BRUXELLES, 6/84 (pp 251,260)

[SCH81] SCHUMAN, CLARKE, NIKOLAOU
Programming distributed applications in ADA : a first approach.
Proc.Int.Conf. on Parallel Processing, 1981

[STA85] STAMMERS R.A.
ADA on distributed hardware.
Concurrent Languages in Distr. Syst., IFIP 85, Elsevier Pub. (pp 35,40)

[THO80] THOMESSE J.P.
SYGARE : une structuration pour la conception d'applications en temps réel et réparties.
Thèse Doct. Etat, NANCY I, 1980

TROISIEME PARTIE

CHAPITRE 5 :

EVALUATION DE LA PROPOSITION

RESUME :

Ce cinquième chapitre vise à évaluer notre proposition. Tout d'abord par une confrontation, sur la base d'un cas (la commande d'une cellule flexible d'assemblage), avec une approche par les Réseaux de PETRI colorés. Puis, en comparant notre outil de spécification et notre démarche avec d'autres outils et démarches proches.

1. CONFRONTATION SUR UN CAS AVEC UNE APPROCHE PAR LES RESEAUX DE PETRI COLORES :

1.1. INTRODUCTION :

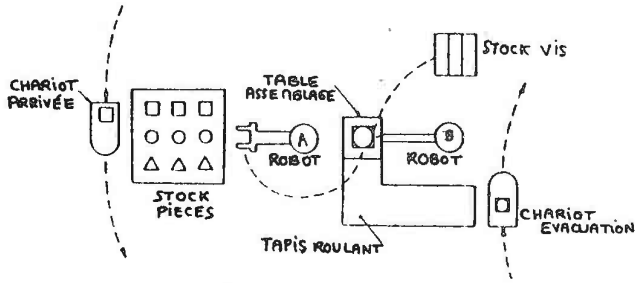
Dans le contexte du projet ARA (cf. Introduction), la spécification des applications de commande des systèmes de production flexibles est abordée par les autres équipes s'intéressant à ce thème en s'appuyant sur les Réseaux de PETRI colorés ([CVM82], [MAY82], [ALL83], ...). Il nous a semblé intéressant de confronter sur un exemple les deux approches. Le cas choisi, extrait d'une communication au congrès AFCET 85 [DVC85], modélise par un réseau de PETRI coloré (RPC), le niveau de la coordination des tâches locales de l' application; c'est à dire que sont exclus, la description détaillée de la commande locale des robots, au niveau bas, et la description des algorithmes d'ordonnancement de la production, au niveau supérieur. Nous rendons compte de ces délimitations par le choix de nos communications externes et en discuterons ultérieurement.

Nous présentons successivement :
- l'énoncé du cas,
- l'établissement de notre spécification logique,
- la confrontation de sa traduction en termes de RPC avec le réseau "cible" (proposé dans [DVC85]), pour quelques observations factuelles,
- nos conclusions au niveau des principes.

1.2. LE CAS :

Il s'agit d'une cellule flexible d'assemblage, qu' illustre la figure 5.1. En fonction des demandes reçues, le robot A prend une pièce dans le stock de pièces, la pose sur la table d' assemblage, puis prend une seconde pièce dans le stock, la pose également et les assemble; le robot B, après avoir pris des vis adaptées dans le stock de vis, fixe les deux éléments assemblés qui sont finalement déchargés sur le tapis roulant. Les deux robots peuvent bien entendu évoluer en parallèle.

Fig. 5.1.

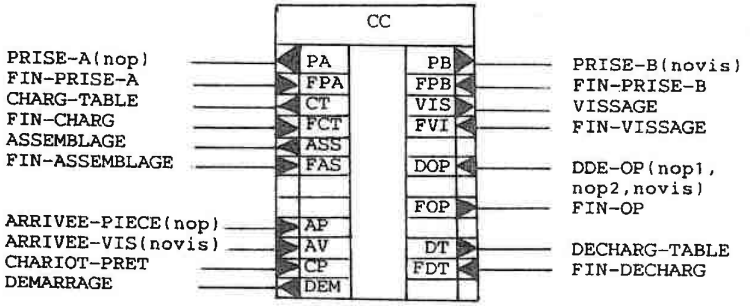


1.3. ETABLISSEMENT DE NOTRE SPECIFICATION LOGIQUE :

a) définition de l'agent global :

Nous modélisons tout d'abord l'application de commande par un agent unique (CC) munis d'entrées et sorties externes dont le choix est révélateur des limites apportées au problème et du niveau d'observation de la commande.

Fig. 5.2.



PRISE-A(nop) est un ordre au robot A de prendre la pièce de référence nop;
FIN-PRISE-A est le compte rendu de fin de cette opération;
CHARG-TABLE, FIN-CHARG, ASSEMBLAGE, FIN-ASSEMBLAGE sont des ordres au robot A et des compte rendus de fin, pour le chargement de la table d'assemblage et l'assemblage.
PRISE-B(novis) est un ordre au robot B de prendre les vis de référence novis;
FIN-PRISE-B est le compte rendu de fin de cette opération;
VISSAGE, FIN-VISSAGE, DECHARG-TABLE, FIN-DECHARG sont des ordres au robot B et des comptes rendus de fin, pour le vissage et le déchargement de la table ;
DDE-OP(nop1,nop2,novis) est une demande de travail en provenance de l'ordonnancement et qui précise les références des composants;
FIN-OP indique à l'ordonnancement la fin d'un travail;
ARRIVEE-PIECE(nop) indique une arrivée en stock de la pièce de référence nop;
ARRIVEE-VIS(novis) idem pour les vis;
CHARIOT-PRET indique qu'un chariot d'évacuation des pièces terminées est prêt à être chargé;
DEMARRAGE donne l'ordre de mise en marche à un tel chariot, après chargement;

b) définition des relations :

Nous "dégrossissons" le problème en termes des relations de causalité, de contribution et de mémorisation. Les variables d'état, introduites pour faciliter la spécification, concernent soit l'état de sous-systèmes (ETAT-A pour le robot A, ETAT-B pour le robot B, ETAT-TABLE pour la table d'assemblage) soit la disponibilité de "ressources" fournies par l'extérieur (EN-STOCK-P pour les pièces, EN-STOCK-V pour les vis, CHARIOT-PRET pour les chariots, DEMANDE-PIECE-EN-ATTENTE pour les demandes de pièces, DEMANDE-VIS-EN-ATTENTE pour les demandes de vis).

Relations de causalité :

Nos	événements déclencheurs	événements déclenchés	commentaires
r1	dɛEV(DOP)	pɛEV(PA)	début de prise de pièce sur une demande d'opération
r2	fɛEV(FAS)	pɛEV(PA)	début de prise de pièce sur fin d'assemblage précédent
r3	fɛEV(FPA)	cɛEV(CT)	chargement de la table sur fin de prise de pièce
r4	fɛEV(FDT)	cɛEV(CT)	chargement de la table sur fin de déchargement
r5	fɛEV(FCT)	pɛEV(PA)	début de prise de 2ème pièce sur fin de chargement
r6	aɛEV(AP)	pɛEV(PA)	début de prise de pièce sur arrivée en stock
r7	fɛEV(FCT)	aɛEV(ASS)	début d'assemblage sur fin de chargement de la table
r8	fɛEV(FAS)	vɛEV(VIS)	début de vissage sur fin d'assemblage
r9	fɛEV(FPB)	vɛEV(VIS)	début de vissage sur fin de prise de vis
r10	fɛEV(FVI)	dɛEV(DT)	début de déchargement sur fin de vissage
r11	cɛEV(CP)	dɛEV(DT)	début de déchargement sur arrivée d'un chariot
r12	fɛEV(FDT)	dɛEV(DEM)	démarrage du chariot sur fin de déchargement
r13	fɛEV(FDT)	fɛEV(FOP)	message de fin d'opération sur fin de déchargement
r14	dɛEV(DOP)	pɛEV(PB)	prise de vis sur demande d'opération
r15	fɛEV(FVI)	pɛEV(PB)	prise de vis sur fin de vissage
r16	aɛEV(AV)	pɛEV(PB)	prise de vis sur arrivée de vis en stock

Relations de contribution :

Nos	variables	événements	commentaires
r17	ETAT-A	pεEV(PA) aεEV(ASS) cεEV(CT)	l'état du robot A conditionne: la prise de pièce, le début de l'assemblage, le chargement de la table
r18	ETAT-B	pεEV(PB) vεEV(VIS)	l'état du robot B conditionne: la prise des vis, le début du vissage
r19	EN-STOCK-P	pεEV(PA)	l'état du stock de pièces conditionne la prise de pièce
r20	EN-STOCK-V	pεEV(PB)	l'état du stock de vis conditionne la prise de vis
r21	DEMANDE-PIECE-EN-ATTENTE	pεEV(PA)	la présence de demandes conditionne la prise de pièce
r22	DEMANDE-VIS-EN-ATTENTE	pεEV(PB)	la présence de demandes conditionne la prise de vis
r23	CHARIOT-PRET	dεEV(DT)	la disponibilité d'un chariot conditionne le déchargement de la table
r24	ETAT-TABLE	cεEV(CT) vεEV(VIS) dεEV(DT)	l'état de la table conditionne: le début de chargement de la table, le début du vissage, le déchargement de la table

Relations de mémorisation :

Nos	événements	variables	commentaires
r25	pεEV(PA),fεEV(FPA), cεEV(CT),fεEV(FCT), fεEV(FAS)	ETAT-A	cf. diagramme d'états ci-après
r26	pεEV(PB),fεEV(FPB), vεEV(VIS),fεEV(FVI)	ETAT-B	cf. diagramme d'états ci-après
r27	aεEV(AP),fεEV(PA)	EN-STOCK-P	arrivée/départ d'une pièce du stock
r28	aεEV(AV),fεEV(PB)	EN-STOCK-V	arrivée/départ d'une vis du stock
r29	dεEV(DOP),pεEV(PA)	DEMANDE-PIECE-EN-ATTENTE	arrivée d'une demande d'opération/prise de pièce
r30	dεEV(DOP),pεEV(PB)	DEMANDE-VIS-EN-ATTENTE	arrivée d'une demande d'opération/prise de vis
r31	cεEV(CP),fεEV(DT)	CHARIOT-PRET	arrivée/occupation d'un chariot
r32	cεEV(CT),fεEV(FAS), vεEV(VIS),fεEV(FVI), dεEV(DT),fεEV(FDT)	ETAT-TABLE	cf. diagramme d'états ci-après

c) spécification formelle des variables d'état :

Le diagramme d'états de ETAT-A εst le suivant :

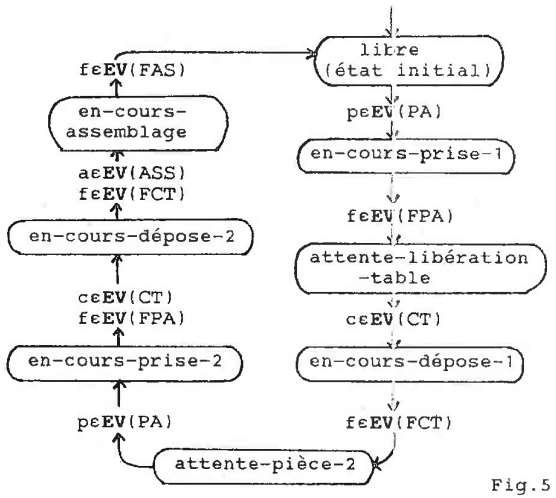


Fig.5.3.

Soit la spécification formelle suivante :

ETAT-A=libre **quand** t : ~∃p(pε**EV**(PA)∩p->t) v (∃f(fε**EV**(FAS)∩ f->t)∩~∃f1(f1ε**EV**(FAS)∩f->f1->t)∩~∃p(pε**EV**(PA)∩f->p->t))

ETAT-A=en-cours-prise-1 **quand** t : (~∃f(fε**EV**(FAS)∩f->t)∩∃p (pε**EV**(PA)∩p->t)∩~∃f1(f1ε**EV**(FPA)∩p->f1->t)∩**ord**(p)=1) v (∃f (fε**EV**(FAS)∩f->t)∩~∃f1(f1ε**EV**(FAS)∩f->f1->t)∩∃p(pε**EV**(PA)∩ f->p->t)∩~∃f2(f2ε**EV**(FPA)∩p->f2->t)∩**ord**(p)=**ord**(f)x2+1)

ETAT-A=attente-libération-table **quand** t : (~∃f(fε**EV**(FAS)∩ f->t)∩∃f1(f1ε**EV**(FPA)∩f1->t)∩~∃c(cε**EV**(CT)∩f1->c->t)∩ **ord**(f1)=1) v (∃f(fε**EV**(FAS)∩f->t)∩~∃f1(f1ε**EV**(FAS)∩f->f1 ->t)∩∃f2(f2ε**EV**(FPA)∩f->f2->t)∩~∃c(cε**EV**(CT)∩f2->c->t)∩ **ord**(f2)=**ord**(f)x2+1)

ETAT-A=en-cours-dépose-1 **quand** t : (~∃f(fε**EV**(FAS)∩f->t)∩∃c (cε**EV**(CT)∩c->t)∩~∃f1(f1ε**EV**(FCT)∩c->f1->t)∩**ord**(c)=1) v (∃f(fε**EV**(FAS)∩f->t)∩~∃f1(f1ε**EV**(FAS)∩f->f1->t)∩∃c(cε**EV** (CT)∩f->c->t)∩~∃f2(f2ε**EV**(FCT)∩c->f2->t)∩**ord**(c)=**ord**(f)x2+1)

ETAT-A=attente-pièce-2 **quand** t : (~∃f(fε**EV**(FAS)∩f->t)∩∃f1 (f1ε**EV**(FCT)∩f1->t)∩~∃p(pε**EV**(PA)∩f1->p->t)∩**ord**(f1)=1) v (∃f(fε**EV**(FAS)∩f->t)∩~∃f1(f1ε**EV**(FAS)∩f->f1->t)∩∃f2(f2ε**EV** (FCT)∩f->f2->t)∩~∃p(pε**EV**(PA)∩f2->p->t)∩**ord**(f1)=**ord**(f)x2+1)

ETAT-A=en-cours-prise-2 **quand** t : (~∃f(fε**EV**(FAS)∩f->t)∩∃p (pε**EV**(PA)∩p->t)∩~∃f1(f1ε**EV**(FPA)∩p->f1->t)∩**ord**(p)=2) v (∃f(fε**EV**(FAS)∩f->t)∩~∃f1(f1ε**EV**(FAS)∩f->f1->t)∩∃p(pε**EV** (PA)∩f->p->t)∩~∃f2(f2ε**EV**(FPA)∩p->f2->t)∩**ord**(p)=**ord**(f)x2+2)

ETAT-A=en-cours-dépose-2 **quand** t : (~∃f(f∈**EV**(FAS)∩f->t)∩∃f1(f1∈**EV**(FPA)∩f1->t)∩~∃f2(f2∈**EV**(FCT)∩f1->f2->t)∩**ord**(f1)=2) v (∃f(f∈**EV**(FAS)∩f->t)∩~∃f1(f1∈**EV**(FAS)∩f->f1->t)∩∃f2(f2∈**EV**(FPA)∩f->f2->t)∩~∃f3(f3∈**EV**(FCT)∩f2->f3->t)∩**ord**(f1)=**ord**(f)x2+2)

ETAT-A=en-cours-assemblage **quand** t : (~∃f(f∈**EV**(FAS)∩f->t)∩∃f1(f1∈**EV**(FCT)∩f1->t)∩~∃f2(f2∈**EV**(FAS)∩f1->f2->t)∩**ord**(f1)=2) v (∃f(f∈**EV**(FAS)∩f->t)∩~∃f1(f1∈**EV**(FAS)∩f->f1->t)∩∃f2(f2∈**EV**(FCT)∩f->f2->t)∩~∃f3(f3∈**EV**(FAS)∩f1->f3->t)∩**ord**(f1)=**ord**(f)x2+2)

Le diagramme d'états de ETAT-B est le suivant :

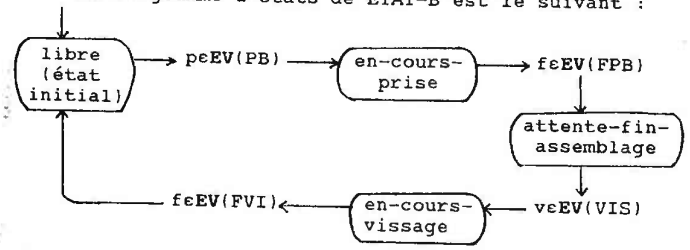


Fig.5.4.

Soit la spécification formelle suivante :

ETAT-B=libre **quand** t : ~∃p(pε**EV**(PB)∩p->t) v (∃f(fε**EV**(FVI)∩f->t)∩~∃f1(f1ε**EV**(FVI)∩f->f1->t)∩~∃p(pε**EV**(PB)∩f->p->t))

ETAT-B=en-cours-prise **quand** t : ∃p(pε**EV**(PB)∩p->t)∩~∃f(fε**EV**(FPB)∩p->f->t)

ETAT-B=attente-fin-assemblage **quand** t : ∃f(fε**EV**(FPB)∩f->t)∩~∃v(vε**EV**(VIS)∩f->v->t)

ETAT-B=en-cours-vissage **quand** t : ∃v(vε**EV**(VIS)∩v->t)∩~∃f(fε**EV**(FVI)∩v->f->t)

Le diagramme d'états de ETAT-TABLE est le suivant :

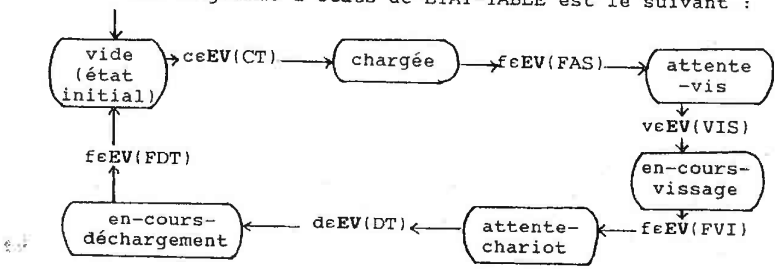


Fig.5.5.

Soit la spécification formelle suivante :

ETAT-TABLE=vide **quand** t : ~∃c(cε**EV**(CT)∩c->t) v (∃f(fε**EV**(FDT)∩f->t)∩~∃c(cε**EV**(CT)∩f->c->t))

ETAT-TABLE=chargée **quand** t : ∃c(cε**EV**(CT)∩c->t)∩~∃f(fε**EV**(FAS)∩c->f->t)

ETAT-TABLE=attente-vis **quand** t : ∃f(fε**EV**(FAS)∩f->t)∩~∃v(vε**EV**(VIS)∩f->v->t)

ETAT-TABLE=en-cours-vissage **quand** t : ∃v(vε**EV**(VIS)∩v->t)∩~∃f(fε**EV**(FVI)∩v->f->t)

ETAT-TABLE=attente-chariot **quand** t : ∃f(fε**EV**(FVI)∩f->t)∩~∃d(dε**EV**(DT)∩f->d->t)

ETAT-TABLE=en-cours-déchargement **quand** t : ∃d(dε**EV**(DT)∩d->t)∩~∃f(fε**EV**(FDT)∩d->f->t)

Les spécifications formelles des autres variables sont les suivantes :

EN-STOCK-P(np)=vrai **quand** t : (∃a(aε**EV**(AP(np))∩a-t)∩~∃p(pε**EV**(PA(np))∩a->p->t)) v (∃a(aε**EV**(AP(np))∩a->t)∩∃p(pε**EV**(PA(np))∩p->t)∩~∃p1(p1ε**EV**(PA(np))∩p->p1->t)∩**ord**(a)>**ord**(p)) (on suppose que les arrivées en stock sont unitaires)

EN-STOCK-P(np)=faux **quand** t : ~∃a(aε**EV**(AP(np))∩a->t) v (∃a(aε**EV**(AP(np))∩a->t)∩~∃a1(a1ε**EV**(AP(np))∩a->a1->t)∩∃p(pε**EV**(PA(np))∩p->t)∩**ord**(p)=**ord**(a))

EN-STOCK-V(novis)=vrai **quand** t : (∃a(aε**EV**(AV(novis))∩a->t)∩~∃p(pε**EV**(PB(novis))∩a->p->t)) v (∃a(aε**EV**(AV(novis))∩a->t)∩∃p(pε**EV**(PB(novis))∩p->t)∩~∃p1(p1ε**EV**(PB(novis))∩p->p1->t)∩**ord**(a)>**ord**(p)) (on suppose que les entrées et les sorties de stock se font par égales quantités)

EN-STOCK-V(novis)=faux **quand** t : ~∃a(aε**EV**(AV(novis))∩a->t) v (∃a(aε**EV**(AV(novis))∩a->t)∩~∃a1(a1ε**EV**(AV(novis))∩a->a1->t)∩∃p(pε**EV**(PB(novis))∩p->t)∩**ord**(p)=**ord**(a))

CHARIOT-PRET=vrai **quand** t : ∃c(cε**EV**(CP)∩c->t)∩~∃d(dε**EV**(DT)∩c->d->t)

CHARIOT-PRET=faux **quand** t : ~∃c(cε**EV**(CP)∩c->t) v (∃d(dε**EV**(DT)∩d->t)∩~∃c(cε**EV**(CP)∩d->c->t))

DEMANDE-PIECE-EN-ATTENTE(np)=vrai **quand** t : ∃d(dε**EV**(DOP)∩d->t)∩∃p(pε**EV**(PA)∩p->t)∩~∃p1(p1ε**EV**(PA)∩p->p1->t)∩{[**ord**(d)=**ord**(p)/2+1∩np=(d.**msg**.nop1 v d.**msg**.nop2)] v [**ord**(d)=(**ord**(p)+1)/2∩np=(d.**msg**.nop1 v d.**msg**.nop2)∩np≠ p.**msg**]}
(np est une des 2 pièces du prochain assemblage ou la pièce restante de l'assemblage en cours)

DEMANDE-PIECE-EN-ATTENTE=faux **quand** t : ~∃d(dε**EV**(DOP)∩d->t) v (∃d(dε**EV**(DOP)∩d->t)∩~∃d1(d1ε**EV**(DOP)∩d->d1->t)∩∃p(pε**EV**(PA)∩p->t)∩~∃p1(p1ε**EV**(PA)∩p->p1->t)∩**ord**(p)=**ord**(d)x2)

DEMANDE-VIS-EN-ATTENTE(nv)=vrai **quand** t : ∃d(d∈**EV**(DOP)∩d->t) ∩∃p(p∈**EV**(PB)∩p->t)∩~∃p1(p1∈**EV**(PB)∩p->p1->t)∩**ord**(d)=**ord**(p) +1 ∩d.**msg**.novis=nv

DEMANDE-VIS-EN-ATTENTE=faux **quand** t : ~∃d(d∈**EV**(DOP)∩d->t) v (∃d(d∈**EV**(DOP)∩d->t)∩~∃d1(d1∈**EV**(DOP)∩d->d1->t)∩∃p(p∈ **EV**(PB)∩p->t)∩~∃p1(p1∈**EV**(PB)∩p->p1->t)∩**ord**(p)=**ord**(d))

d) spécification formelle de l'agent CC :

(1) ∀i∈[1..2],∀d(d∈**EV**(DOP)∩P1 #> ∃p(p∈**EV**(PA)∩d=>p∩p.**msg**= d.**msg**.nop_i))
avec P1 : ~DEMANDE-PIECE-EN-ATTENTE **quand** d∩ETAT-A=libre **quand** d∩EN-STOCK-P(d.**msg**.nop_i) **quand** d

(2) ∀np∈NOP,∀f(f∈**EV**(FAS)∩P2 #> ∃p(p∈**EV**(PA)∩f=>p∩p.**msg**= np))
avec P2 : DEMANDE-PIECE-EN-ATTENTE(np) **quand** f∩EN-STOCK-P(np) **quand** f
(NOP est l'ensemble des références de pièces)

(3) ∀f(f∈**EV**(FPA)∩P3 #> ∃c(c∈**EV**(CT)∩f=>c))
avec P3 : ETAT-TABLE=vide **quand** f∩ETAT-A=en-cours-prise-1 **quand** f

(4) ∀f(f∈**EV**(FPA)∩P4 #> ∃c(c∈**EV**(CT)∩f=>c))
avec P4 : ETAT-A=en-cours-prise-2 **quand** f

(5) ∀f(f∈**EV**(FDT)∩P5 #> ∃c(c∈**EV**(CT)∩f=>c))
avec P5 : ETAT-A=attente-libération-table **quand** f

(6) ∀np∈NOP,∀f(f∈**EV**(FCT)∩P6 #> ∃p(p∈**EV**(PA)∩f=>p∩p.**msg** =np))
avec P6 : ETAT-A=**en-cours-dépose-1** **quand** f∩DEMANDE-PIECE-EN-ATTENTE(np) **quand** f∩EN-STOCK-P(np) **quand** f

(7) ∀a(a∈**EV**(AP)∩P7 #> ∃p(p∈**EV**(PA)∩a=>p∩p.**msg**=a.**msg**))
avec P7 : ETAT-A=attente-pièce-2 **quand** a∩DEMANDE-PIECE-EN-ATTENTE(a.**msg**) **quand** a

(8) ∀a(a∈**EV**(AP)∩P8 #> ∃p(p∈**EV**(PA)∩a=>p∩p.**msg**=a.**msg**))
avec P8 : ETAT-A=libre **quand** a∩DEMANDE-PIECE-EN-ATTENTE(a.**msg**) **quand** a

(9) ∀f(f∈**EV**(FCT)∩P9 <#> ∃a(a∈**EV**(ASS)∩f=>a))
avec P9 : ETAT-A=en-cours-dépose-2 **quand** f

(10) ∀f(f∈**EV**(FAS)∩P10 #> ∃v(v∈**EV**(VIS)∩f=>v))
avec P10 : ETAT-B=attente-fin-assemblage **quand** f

(11) ∀f(f∈**EV**(FPB)∩P11 #> ∃v(v∈**EV**(VIS)∩f=>v))
avec P11 : ETAT-TABLE=attente-vis **quand** f

(12) ∀f(f∈**EV**(FVI)∩P12 #> ∃d(d∈**EV**(DT)∩f=>d))
avec P12 : CHARIOT-PRET **quand** f

(13) ∀c(c∈**EV**(CP)∩P13 #> ∃d(d∈**EV**(DT)∩c=>d))
avec P13 : ETAT-TABLE=attente-chariot **quand** c

(14) ∀f(f∈**EV**(FDT) <#> ∃d(d∈**EV**(DEM)∩f=>d))

(15) ∀f(f∈**EV**(FDT) <#> ∃f1(f1∈**EV**(FOP)∩f=>f1))

(16) ∀d(d∈**EV**(DOP)∩P16 #> ∃p(p∈**EV**(PB)∩d=>p∩p.**msg**= d.**msg**.novis))
avec P16 : ~DEMANDE-VIS-EN-ATTENTE **quand** d∩ETAT-B=libre **quand** d∩EN-STOCK-V(d.**msg**.novis) **quand** d

(17) ∀nv∈NOV,∀f(f∈**EV**(FVI)∩P17 #> ∃p(p∈**EV**(PB)∩f=>p∩p.**msg**= nv))
avec P17 : DEMANDE-VIS-EN-ATTENTE(nv) **quand** f∩EN-STOCK-V (nv) **quand** f (NOV est l'ensemble des références de vis)

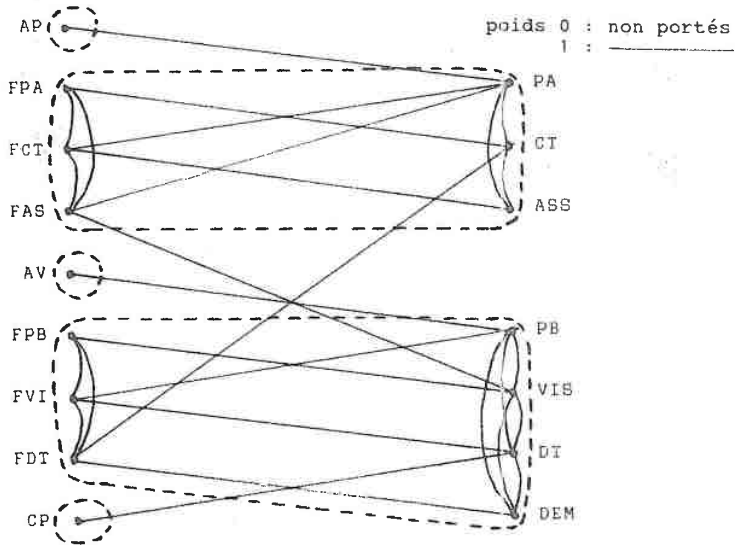
(18) ∀a(a∈**EV**(AV)∩P18 #> ∃p(p∈**EV**(PB)∩a=>p∩p.**msg**=a.**msg**))
avec P18 : DEMANDE-VIS-EN-ATTENTE(a.**msg**) **quand** a∩ETAT-B=libre **quand** a

auxquelles s'ajoutent les ensembles de causalités possibles de PA, PB, CT, VIS, DT et la propriété de non duplication.

e) structuration logique de l'application :

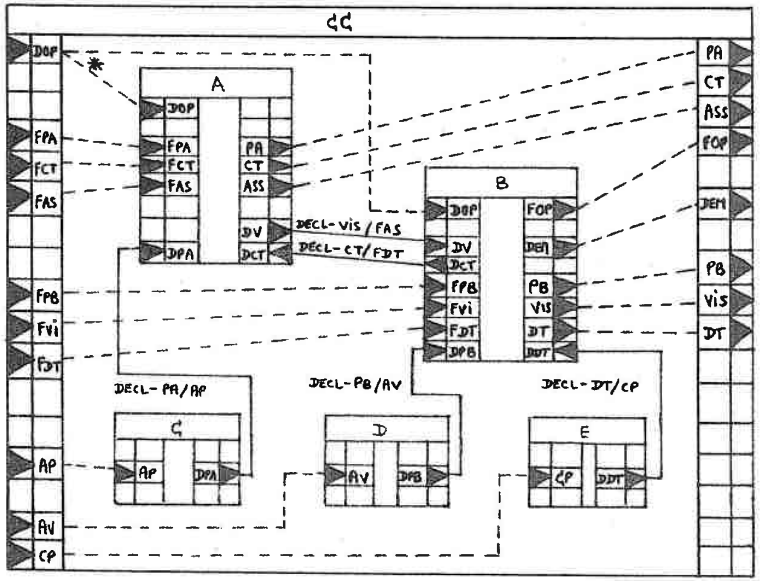
Nous appliquons notre méthode de partitionnement. Les ensembles d'entrées externes temporellement exclusives sont les suivants : (FPA, FCT, FAS), (FPB, FVI, FDT), (AP), (AV), (CP). Les ensembles de sorties externes temporellement exclusives sont les suivants : (PA, CT, ASS), (PB, VIS, DT, DEM). De ces ensembles et du tableau des relations de causalité, on déduit le graphe de la figure 5.6., où est tracé le découpage optimal.

Fig.5.6.



Il résulte de ce graphe, la structuration de la figure 5.7, avec 5 déclenchements à distance correspondant aux 5 arcs inter composantes (types de messages DECL-PA/AP, DECL-PB/AV, DECL-DT/CP, DECL-VIS/FAS, DECL-CT/FDT)

Fig.5.7.



Les communications internes vers le sous-système d'ordonnancement (DOP, FOP) sont attachées aux agents existants:

- FOP à B (cf. relation de causalité r13),
- DOP à A et B (cf. relations de causalité r1 et r14).

Les variables d'état sont ensuite affectées aux agents en fonction des relations de mémorisation, selon la méthode décrite au chapitre 3 § 2.1.4 :

- ETAT-A est localisée sur A, comme tous les messages qui permettent sa mise à jour (PA, FPA, CT, FCT, FAS);
- ETAT-B est localisée sur B, comme tous les messages qui permettent sa mise à jour (PB, FPB, VIS, FVI);
- EN-STOCK-P est localisée sur C, car nous donnons une pondération supérieure à AP qu'à PA (faisant ainsi de C le gestionnaire du stock de pièces); même chose pour EN-STOCK-V sur D (gestionnaire du stock de vis) et CHARIOT-PRET sur E (gestionnaire des chariots);
- DEMANDE-PIECE-EN-ATTENTE est localisée sur A, comme tous ses messages de mise à jour (DOP, PA); de même pour DEMANDE-VIS-EN-ATTENTE, sur B (DOP, PB);

- ETAT-TABLE est localisée sur B, car nous donnons des pondérations identiques à tous les messages (4 pour B, 2 pour A).

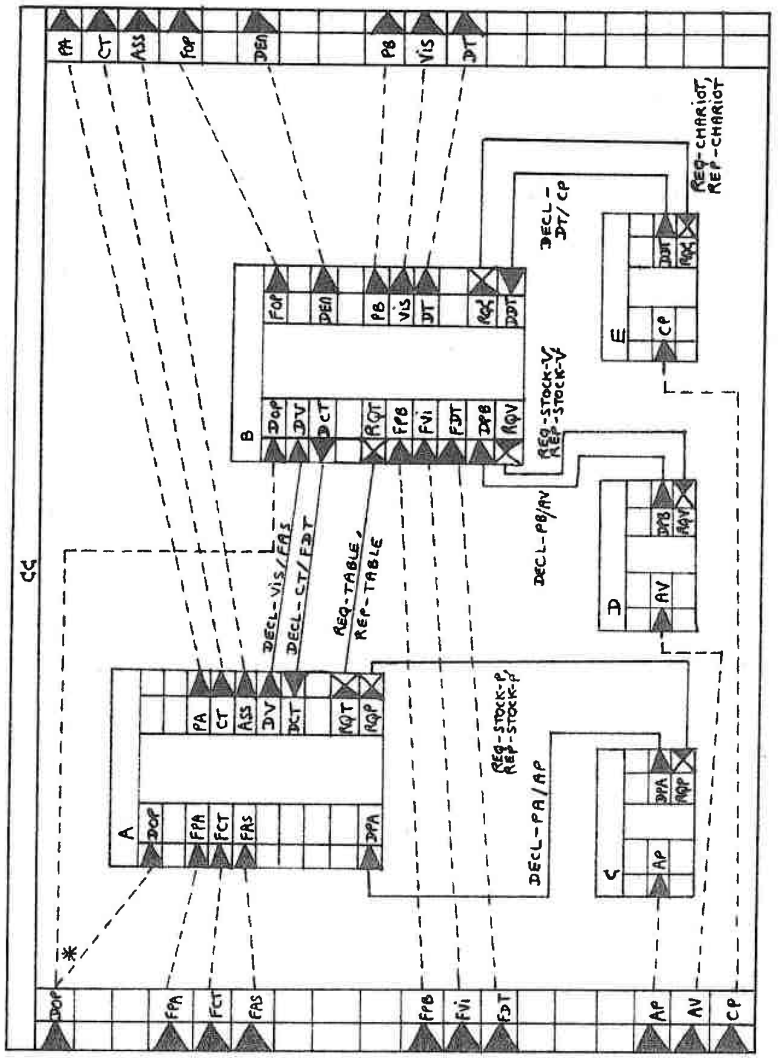
Il en résulte des mises à jour à distance de EN-STOCK-P depuis A (PA), de EN-STOCK-V depuis B (PB), de CHARIOT-PRET depuis B (DT) et de ETAT-TABLE depuis A (CT).

Au vu des relations de contribution il est nécessaire d'avoir des consultations à distance de EN-STOCK-P depuis A (PA), de EN-STOCK-V depuis B (PB), de CHARIOT-PRET depuis B (DT) et de ETAT-TABLE depuis A (CT).

Les contributions et les mises à jour sont conjointes (cf. § 2.1.4.d. Chapitre 3) et 4 requêtes/réponses suffisent à les assurer : REQ-STOCK-P/REP-STOCK-P, REQ-STOCK-V/REP-STOCK-V, REQ-CHARIOT/REP-CHARIOT et REQ-TABLE/REP-TABLE. Soit la structure logique complète de la figure 5.8.

La conception peut se poursuivre par la spécification du comportement des agents élémentaires et la validation de cohérence suivant la procédure classique.

Fig.5.8.



1.4. CONFRONTATION AVEC LA SPECIFICATION PAR RPC :

La correspondance entre les éléments de notre spécification et le réseau "cible" (cf. fig.5.9.) est assez facile à établir.

Les événements de communication se retrouvent soit sous la forme de places partagées (communications "asynchrones") soit sous la forme de tirs forcés de transitions par des variables de synchronisation (communications "synchrones"). Nous discuterons de cette dualité de traductions ultérieurement.

Exemple :

Evénements sur les ports :	places partagées :	transitions et variables de synchronisation :
PA	DPRISE1	
FPA		FIN-PRISE1
CT	DMETTRE1, DMETTRE2	
FCT		FIN-METTRE
ASS	DASSEMB	
FAS		FIN-ASSEMB
AP	SP	
AV	SV	
CP	CD	
DEM		LIB-CHARG
PB	DPRISE2	
FPB		FIN-PRISE2
VIS	DV	
FVI		FIN-VISS
DOP	DEM-PILOT	
FOP		FIN-DEM
DT	DECH	
FDT		FIN-DECH

♦

Aux variables d'état ETAT-A, ETAT-B, ETAT-TABLE, caractéristiques de sous-systèmes du procédé, correspondent des ensembles de places liées en boucle avec une marque dans l'état initial.

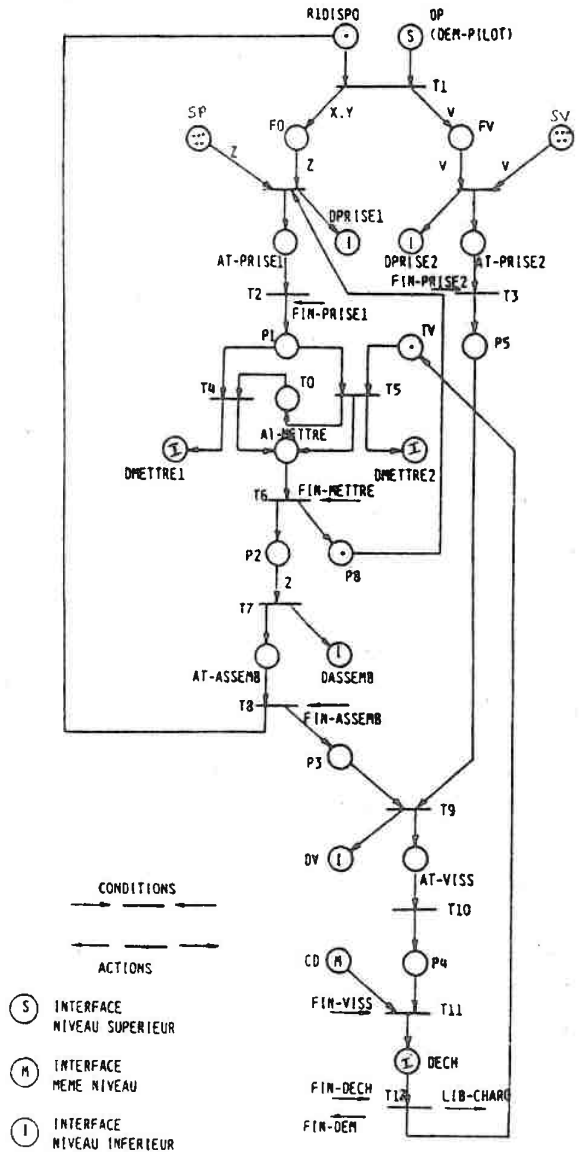
Exemple :

valeurs de ETAT-TABLE :	places correspondantes :
vide (état initial)	TV
chargée	A1-METTRE, P2, AT-ASSEMB
attente-vis	P3
en-cours-vissage	AT-VISS
attente-chariot	P4
en-cours-déchargement	DECH

♦

Nous montrons au paragraphe suivant comment l'examen systématique des correspondances nous permet de mettre

Fig.5.9.



en évidence un certain nombre d'erreurs de spécification dans le réseau cible. Nous généralisons ensuite notre propos en indiquant les réticences que nous inspirent les Réseaux de Petri, comme support d'une méthode de conception.

a) considérations factuelles liées à l'exemple :

Le tableau des relations de causalité nous indique que les prises de vis (événements sur PB) peuvent être déclenchées par les demandes d'opération (relation r14), les arrivées de vis en stock (relation r16) et les fins de vissage par le robot B (relation r15). **Cette dernière possibilité n'est pas rendue dans le réseau cible** où la prise de vis (DPRISE2) n'est pas conditionnée par la fin du vissage précédent (FIN-VISS). En conséquence, on peut demander au robot B de visser et de prendre une vis en même temps : il suffit qu'à la fin d'un assemblage une demande d'opération soit présente dans DEM-PILOT et que la vis correspondante soit en stock ! (un arc devrait lier la transition T10 à une place matérialisant la disponibilité du robot B, elle même conditionnant, avec FV et SV, le déclenchement de DPRISE2).

Lorsque l'on recherche la correspondance entre les valeurs de ETAT-B et les places du réseau, on observe une seconde erreur (plus anecdotique) :

Valeurs de ETAT-B :	Places :
libre	absente (cf.ci-dessus)
en-cours-prise	AT-PRISE2
attente-fin-assemblage	P5
en-cours-vissage	AT-VISS et P4
attente-chariot	?
en-cours-déchargement	DECH

L'absence de place correspondant à l'état attente-chariot résulte du mauvais placement de FIN-VISS qui devrait être remonté sur la transition T10; on aurait alors :

en-cours-vissage	AT-VISS
attente-chariot	P4

On retrouverait ainsi la structuration présente partout ailleurs dans le réseau (cf. DPRISE1/AT-PRISE1/FIN-PRISE1, DMETTRE/AT-METTRE /FIN-METTRE, DASSEMB/AT-ASSEMB/FIN-ASSEMB, DPRISE2/AT-PRISE2/ FIN-PRISE2, etc.).

Lorsque l'on examine la définition de la variable DEMANDE-VIS-EN-ATTENTE on trouve une troisième erreur de spécification, beaucoup plus subtile. D'après notre spécification DEMANDE-VIS-EN-ATTENTE(nv) n'est vrai que si nv est le numéro de vis associé à la demande d'opération de rang immédiatement supérieur à la dernière prise de vis (cf. **ord**(d)=**ord**(p)+1)). Autrement dit **il doit s'agir de la plus ancienne demande de vis non satisfaite**. Or le réseau cible n' interdit pas la présence de 2 marques dans la place FV, correspondant à 2 demandes de vis successives non satisfaites : l'ordre d' arrivée de ces marques n'est pas connu (il suffit

qu'une première opération soit prise en compte; que les vis soient indisponibles; que l'assemblage soit effectué; qu'une deuxième opération soit prise en compte et ses vis également indisponibles). On risque alors de visser un assemblage avec des vis destinées à l'assemblage suivant si celles-ci arrivent les premières dans le stock ! (la solution est de remplacer FV par un tampon FIFO à 2 places).

Une approche analytique, ordonnée et progressive, telle que nous la prônons, nous semble en mesure de minimiser le nombre de telles erreurs.

b) considérations générales liées au formalisme :

L'utilisation des Réseaux de PETRI pour la conception des applications de commande de procédés nous inspire les réflexions suivantes :

- les RP ne permettent pas de décrire l'existence d'un échange sans faire un choix de mise en oeuvre (synchrone/asynchrone); à certains niveaux de la démarche nous pensons préférable de ne raisonner qu'en termes d'existence d'échanges et les RP obligent à des sur-spécifications.

- les RP ne permettent pas, sauf dans des cas élémentaires, de spécifier les "data-oriented properties" (propriétés sur les contenus des messages, comme par exemple la spécification des valeurs des réponses aux requêtes des opérateurs); d'autres propriétés que nous utilisons souvent (comme la conservation de l'ordre) posent également problème.

- plus généralement, et même en faisant appel aux abréviations (Réseaux de PETRI Colorés), certains problèmes apparaissent difficiles à modéliser dans la pratique en raison de leur complexité.
Exemple : dans le cas des ateliers flexibles dont le système de transport des pièces est très peu flexible (ex: l'atelier de masticage des carrosseries [VAL81], avec des tables de transfert à rouleaux et seulement 2 aiguillages), les choix d'itinéraires et les choix d'affectation des pièces aux machines sont complètement spécifiés à l'aide de RPC [VAL81, ALL83]. Lorsque le système de transport devient réellement flexible et donc beaucoup plus complexe, les algorithmes de choix d'itinéraires ou d'affectation ne sont plus modélisés [MAY82]. La "théorisation" qui consiste à définir une séparation entre règles opératoires (les séquencements d'opérations, représentables en RP) et règles de décision (le "coordinateur", représenté par d'autres moyens) apparaît un peu comme un discours de circonstance, reconnaissant cet état de fait. Les conséquences n'en sont pas négligeables : la portée des validations (ex: non blocage), souvent mises en avant pour justifier le recours aux RP, en est très amoindrie; il faudrait par exemple pouvoir prouver que la conjonction des choix au niveau local (modélisés par RP) et au niveau décisionnel

(non modélisés par RP) n'est pas génératrice de nouveaux blocages.
♦
- même lorsque les ambitions sont limitées à la "modélisation de la synchronisation des tâches de l'application de commande" ([CVM82], [ALL83]), on doit noter l'hétérogénéité des descriptions :
  + au niveau de la commande locale des machines, les tâches sont des ordres élémentaires au matériel,
  + alors qu'au niveau de la coordination globale du système, les tâches sont des modules de programmes, voire des programmes tout entiers [VAL81].

L'unité d'outil n'est qu'apparente et les interprétations qui devront être réalisées seront très différentes aux différents niveaux. Il apparaît très simplificateur de parler de "synthèse directe" [ALL83] (c'est à dire de compilation des RP pour produire un code exécutable), particulièrement dans le domaine des ateliers flexibles où la complexité de la coordination est une caractéristique fondamentale.

- les RP constituent surtout un outil de représentation (et de validation dans certains cas) d'une solution; mais pas plus que l'organigramme en programmation classique, ils n'aident à son élaboration. On souhaiterait une démonstration complète de l'adéquation des RP comme support d'une démarche **progressive**, **descendante** et **constructive** de conception, à l'image de ce que nous proposons ici. Notons enfin, que la primauté accordée à la représentation graphique des RP, parfaite pour des exemples d'école, ne nous semble pas constituer un avantage pour les problèmes réels un peu complexes.

2. COMPARAISON AVEC D'AUTRES METHODES ET OUTILS DE CONCEPTION:

a) les méthodes :

Dans le domaine des méthodes de conception d'applications à vocation répartie et temps-réel plus ou moins marquée, on peut distinguer des contributions plutôt "pratiques" et des contributions plutôt "théoriques".

Même dans le registre "pratique", peu d'entre elles abordent le problème dans sa globalité, depuis les propriétés requises jusqu'à l'implantation. Nous avons déjà cité [CAL82] qui propose un cadre méthodologique en cinq niveaux :

- niveau de la description externe (spécification de l'application),
- niveau de la description fonctionnelle (description "topologique" des éléments constitutifs - modules, actions, ports, variables partagées, événements -),
- niveau de la description opérationnelle (qui ajoute au précédent la description algorithmique des actions élémentaires),
- niveau de la description exécutive (qui ajoute au précédent la description de la structure d'exécution et de l'intégration de la spécification opérationnelle sur elle),
- niveau de la réalisation.

On peut regretter dans ce travail, l'absence de tout outil de formalisation des propriétés requises, au premier niveau et de tout outil de spécification abstraite du comportement des entités retenues, au second niveau. Une critique du même ordre peut être adressée à [DER83].

On trouve des travaux assez complets mais où la répartition ne joue pas un rôle prépondérant : il s'agit de systèmes anciens de développement d'applications parallèles adaptés plus ou moins aux exigences de la répartition (ex: SARA [EST86], fondé sur les "UCLA-graphs", COSY [LAU79, LAU80], fondé sur les expressions de chemin).

Beaucoup d'approches "pratiques" s'appuient sur :

- des outils descriptifs classiques, comme les diagrammes du type "data flow" [DEM78, GOM84, ...] ou "control flow" [ALF77, EST86, YAU83, ...],
- des listes de "règles" de structuration. A titre d'exemple, on trouve dans [GOM84] une liste de six critères pour structurer en tâches les applications temps-réel, à partir de diagrammes du type "data flow" (ensemble de "transformateurs élémentaires" de données irrigués par des flots de données) :
  - "dépendance d'E/S" (un transformateur qui accepte des données d'un périphérique ou lui en envoie, travaille à la vitesse de ce périphérique et doit donc constituer une tâche séparée),
  - "fonction critique en temps" (ces transformateurs doivent constituer des tâches séparées pour recevoir des priorités élevées),

  - "fonction intensive en calcul" (à l'inverse, ces transformateurs doivent constituer des tâches séparées à faibles priorités),
  - "cohésion fonctionnelle" (les transformateurs ayant des fonctions proches sont regroupés pour éviter le coût des communications qui résulteraient de leur séparation),
  - "cohésion temporelle" (les transformateurs qui doivent être exécutés en conséquence d'un même stimulus - ex : une interruption - sont regroupés),
  - "exécution périodique" (les transformateurs périodiques constituent des tâches périodiques).

Ces critères recoupent pour l'essentiel ceux exprimés aux différents stades de notre démarche.

Les contributions "théoriques" décrivent en général un processus de structuration des applications par mise en oeuvre de transformations rigoureuses sur une description formalisée des propriétés attendues :

- structuration par décomposition des propriétés invariantes des applications [KRC78, GOK79],
- structuration par décomposition de spécifications en sous-spécifications les plus indépendantes possibles (par exemple en "couches" non communicantes dans [ELF82]),
- synthèse de processus communicants à partir de spécifications en logique temporelle [WOL82, MAW84], etc.

La difficulté réside dans la mise en oeuvre effective de ces idées sur des applications réelles, en raison de leur nature même (ex : prendre comme point de départ l'"invariant global" de l'application, en vue de le décomposer, comme dans [KRC78], nous apparaît assez irréaliste) ou des restrictions qui les accompagnent.

Notre proposition se veut, tout au contraire, globale et effectivement utilisable.

b) les outils :

Le foisonnement des outils de spécification est très important [BAR84]. Nous nous limitons ici à une brève discussion de trois formalismes à base d'événements à l'interface, que nous avons écartés au profit du modèle de CHEN & YEH : les traces, les expressions de comportement, les logiques temporelles classiques.

Les traces [HOA78, MIS81, ZOH81] sont des suites d'événements à l'interface; les propriétés des systèmes sont exprimées à l'aide de ces suites et d'opérations portant sur elles. Elles supposent un ordre total sur les événements et donc une forme de temps unique, ce qui les rapproche de notre modèle "simplifié". La puissance de ce modèle apparaît insuffisante : il est difficile de rendre compte de certains comportements auxquels ne correspondent pas des suites bien définies (ex : une liaison non fiable avec pertes possibles de messages, duplications possibles, etc.). Par ailleurs, les propriétés de vivacité ("liveness properties") sont difficiles à exprimer en termes des seules traces [MIS81].

Les expressions de comportement (CCS [MIL80], LOTOS [BRI86]) visent à décrire tous les comportements possibles en termes d'événements à l'interface, grâce aux opérateurs de séquence, de choix, de composition séquentielle et parallèle de processus (eux-mêmes définis en termes d'expressions de comportement), de récursivité, etc. Fondamentalement, il nous semble préférable pour notre classe de problèmes, d'exprimer plutôt les relations entre événements en entrée et événements en sortie et les quelques effets de bord entre elles, que l'ensemble des séquences d'événements possibles à l'interface; en effet, l'existence de très nombreux événements indépendants à l'interface rend impraticable une description sous forme d'un processus unique : il faut recourir à une décomposition en un ensemble de processus. Le modèle induit donc une certaine forme de structuration, toujours délicate à intégrer dans une démarche de conception, alors que notre proposition permet de conserver une description en "boîte noire" unique pour des systèmes beaucoup plus complexes. Par ailleurs, il nous a semblé préférable d'autoriser des références explicites aux états plutôt que de nous limiter exclusivement aux séquences d'événements à l'interface.

Les logiques temporelles "classiques" construites autour des opérateurs temporels $\Box$ ("maintenant et toujours") et $\Diamond$ ("maintenant ou plus tard") [OWI82, LAM83] : ici également, la recherche de propriétés invariantes utilisant ces opérateurs temporels nous apparaît beaucoup moins immédiate que la recherche des relations entre entrées et sorties. De manière générale, les invariants nous semblent plutôt des outils de vérification a posteriori d'une solution déjà établie. On a en outre souvent souligné le manque de clarté des formules comportant plusieurs opérateurs, en raison du caractère implicite et lié au contexte de l'expression du temps [SCH82].

Notre choix se fonde d'une part, sur la meilleure adéquation aux problèmes considérés (naturellement décrits comme des transformations de données de contrôle en commandes) et d'autre part, sur la simplicité, la souplesse et l'efficacité suffisante du modèle de CHEN & YEH.

BIBLIOGRAPHIE :

[ALF77] ALFORD M.W.
A requirement engineering methodology for real-time processing requirements.
IEEE Trans.on Soft. Eng., Vol SE-3, no 1, 1/77 (pp 60,69)

[ALL83] ALLA, LADET
Spécification de la commande des ateliers flexibles. Utilisation des Réseaux de PETRI colorés.
2èmes Journées ARA, BESANCON, 11/83

[BAR84] BARRINGER H.
Formal specification techniques for parallel and distributed systems. A short review.
Proc. ADATEC/ADA Europe Joint Conf., BRUXELLES, 11/84 (pp 281,294)

[BRI86] BRINKSMA E.
A tutorial on LOTOS.
Protocol Specif., Testing and Verification, Elsevier Pub., 1986, (pp 171,194)

[CAL82] CALVEZ J.P.
Une méthodologie de conception des systèmes multi-micro-ordinateurs pour les applications de commande en temps réel.
Thèse d'Etat, NANTES, 11/82

[CVM82] COURVOISIER, VALETTE, MAYEUX
Spécification de la synchronisation fonctionnelle interne des systèmes de conduite d'ateliers flexibles.
1ères Journées ARA, POITIERS, 9/82 (pp 131,140)

[DEM78] DE MARCO T.
Structured Analysis and system specification.
Yourdon Press, NEW-YORK, 78

[DER83] DERNIAME, ZAKARI
Langage de conception pour les applications réparties en conduite de procédés.
Actes Bigre 83, Le Cap d'Agde, 10/83 (pp 363,376)

[DVC85] DESCLAUX, VALETTE, COURVOISIER
Un langage de description paramétré pour la spécification de la commande d'ateliers de production automatisée.
Congrès AFCET Automatique 85, TOULOUSE, 10/85 (pp 111,123)

[ELF82] ELRAD, FRANCEZ
Decomposition of distributed programs into communication-closed layers.
Report RC9760, IBM Watson Research Center, 82

[EST86] ESTRIN, FENCHEL, RAZOUK, VERNON
SARA : Modeling, Analysis, and Simulation Support for Design of Concurrent Systems.
IEEE Trans.on Soft.Eng., Vol SE-12, no 2, 2/86 (pp 293,311)

[GOK79] GOLDSACK, KRAMER
The use of invariants in the application-oriented specification of real-time control systems.
Research Report, Imperial College, LONDON, 1979

[GOM84] GOMAA H.
A software design method for real-time systems.
CACM, Vol 27, no 9, 9/84 (pp 938,949)

[HOA78] HOARE C.A.R.
A model for communicating processes.
Comp.Lab., OXFORD University, 12/78

[KRC78] KRAMER, CUNNINGHAM
Towards a notation for the functionnal design of distributed processing systems.
Proc.IEEE Int.Conf.on Parallel Proces., 78 (pp 69,76)

[LAM83] LAMPORT L.
Specifying concurrent program modules.
ACM Trans.on Prog.Lang. and Syst., Vol 5, no 2, 83 (pp 190,222)

[LAU79] LAUER, TORRIGIANI, SHIELDS
COSY : a system specification language based on paths and processes.
Acta Informat., 12/79

[LAU80] LAUER, SHIELDS, BEST
Design and analysis of highly parallel and distributed systems.
LNCS86, SPRINGER-VERLAG, 80

[MAW84] MANNA, WOLPER
Synthesis of communicating pocesses from temporal logic specifications.
ACM TOPLAS, Vol 6, no 1, 84 (pp 68,93)

[MAY82] MAYEUX D.
Modélisation et validation de la synchronisation des tâches du logiciel de commande d'un système de transport dans un atelier flexible.
Rap. LAAS-PASTELS 82-0-16, TOULOUSE, 3/82

[MIL80] MILNER R.
A calculus for communicating systems.
LNCS, no 92, Sringer-Verlag, 1980

[MIS81] MISRA, CHANDY
Proofs of networks of processes.
IEEE Trans.on Soft.Eng., Vol SE-7, no 4, 7/81 (pp 417,426)

[OWI82] OWICKI, LAMPORT
Proving liveness properties of concurrent programs.
ACM Trans.on Prog.Lang.and Sys., Vol 4, no 3, 82 (pp 455,495)

[SCH82] SCHWARTZ, MELLIAR-SMITH
From state machines to temporal logic : specification methods for protocol standards.
IEEE Trans.on Communications, 12/82

[VAL81] VALETTE R.
Spécification et validation de la synchronisation des tâches dans un atelier flexible de masticage.
Rap. LAAS-PASTELS, 81-I-38, TOULOUSE, 10/81

[WOL82] WOLPER P.L.
Synthesis of communicating processes from temporal logic specifications.
Rep. STAN-CS 82-925, STANFORD Univ.,1982

[YAU83] YAU, CAGLAYAN
Distributed Software System Design Representation using PETRI Nets.
IEEE Trans.on Soft. Eng., Vol SE-9, no 6, 11/83 (pp 733,745)

[ZOH81] ZHOA, HOARE
Partial correctness of communicating sequential processes.
Proc. IEEE, 2nd Int. Conf. on Distrib.Comp.Syst., Paris, 4/81

CONCLUSION

En conclusion de ce mémoire, nous récapitulons les caractéristiques essentielles de nos propositions et indiquons les orientations envisagées pour la prolongation du travail.

Cette contribution peut se caractériser :

- comme étant l'aboutissement d'une démarche empirique, ayant consisté à confronter les **besoins d'un domaine d'application précis** avec "l'état de l'art" dans les domaines de la structuration des applications réparties, des méthodes de conception et des outils de spécification;

- par la **primauté accordée à la méthode**, c'est à dire au processus progressif et ordonné de résolution des divers problèmes que pose la conception; l'outil de spécification retenu est celui qui nous est apparu le mieux correspondre aux exigences de cette méthodes et au contexte de sa mise en oeuvre;

- par sa **globalité**, qui consiste à prendre en compte, ou au moins à mettre en perspective, toutes les étapes de l'activité de conception.

- par son **"réalisme"**, qui consiste à proposer autour d'un noyau immédiatement et concrètement utilisable (architecture logicielle, démarche empirique descendante, outil graphique de spécification structurelle, spécification détaillée en ADA ou en pseudo-code, éditeurs/contrôleurs pour ces descriptions, traduction des concepts en ADA, etc.) un ensemble d'outils intellectuels et logiciels de plus haut niveau et d'utilisation facultative (spécifications formelles de comportement, validations associées, outils d'assistance aux validations ,partitionnement systématique, outils de prototypage, etc.).

Lorsque l'on dresse le bilan du travail, force est de constater que les propositions sont plus conséquentes et mieux établies pour les étapes initiales de la démarche que pour les étapes organiques et de spécification détaillée. Deux raisons expliquent ce déséquilibre :

- les (trop rares) cas que l'on trouve dans la littérature, comme les cas étudiés conjointement par les différentes équipes de recherche dans le contexte d'ARA, n'intègrent jamais les caractéristiques précises qui permettraient d'étayer les choix concernant la structuration définitive (fréquences des messages externes, quantification des contraintes temporelles à l'interface, contraintes d'implantation, etc.); ceci rend difficile l'investigation et l'illustration des résultats;

- plus fondamentalement, nous avons souhaité éviter de privilégier une forme trop précise de mise en oeuvre. Beaucoup de problèmes qui se posent au niveau de la spécification détaillée ne sont pas indépendants de cette

mise en oeuvre et même s'ils le sont dans une certaine mesure, ce ne peut être démontré qu'en entrant dans le détail de plusieurs formes d'implantation possibles.

Les perspectives pour la prolongation et l'approfondissement du travail sont dès lors claires : il faudra aborder rapidement l'expérimentation en vraie grandeur, après avoir défini un cadre effectif de mise en oeuvre. Le moment nous apparaît opportun, car nous avons constaté une certaine stabilisation des idées et des outils, lors des derniers exemples que nous avons traités. Par un effet de rétroaction souvent décrit, on peut espérer de cette expérimentation un enrichissement de nos méthodes et outils, y compris aux niveaux les plus en amont de la démarche. La disponibilité effective d'un ensemble de logiciels pour la gestion et le contrôle des spécifications, devrait constituer une aide appréciable pour la réalisation de ces expérimentations dans les meilleurs conditions.

D'autres champs de recherches, tels par exemple la prévision, dès le début du cycle de développement, du "dimensionnement" des moyens de communication (réseaux locaux) et de traitement sur la base des spécifications en entitées communicantes, mériteraient d'être abordés également en dépit de leur complexité (soulignée au chapitre 2).

Nous espérons de ce travail, qu'il contribuera à combler un peu le fossé qui sépare théorie et pratique, en particulier dans le domaine de la conception. Fossé très sensible au niveau des échanges vécus dans le cadre du projet ARA, entre "sensibilités informaticiennes" et "sensibilités automaticiennes", d'une part, "préoccupations universitaires" et "préoccupations industrielles", d'autre part.

==========

ANNEXES :

ANNEXE I

Spécification formelle des modules et validation de la cohérence de l'exemple du § 2.2.2.c. du chapitre 3.

a) La spécification formelle des modules est la suivante :

```
module M1
    interface
        E-PORT LL : LECT-LAS;
        E-PORT LO : LECT-OP;
        E-PORT DE : D-ECR;
        S-PORT EO : ECR-OP;
        S-PORT DC : D-CONT;
    comportement
        (M11) ∀l(l∈EV(LL) ∩ P_M11 <#> ∃e(e∈EV(EO) ∩ l=>e ∩
              l.msg=manuel))
              avec P_M11 : ~normal-ident(l.msg)
        (M12) ∀l(l∈EV(LL) ∩ P_M12 #> ∃d(d∈EV(DC) ∩ l=>d ∩
              d.msg=l.msg))
              avec P_M12 : normal-ident(l.msg)
        (M13) ∀l(l∈EV(LO) ∩ P_M13 #> ∃e(e∈EV(EO) ∩ l=>e ∩
              e.msg=retirer))
              avec P_M13 : ~normal-ident(l.msg)
        (M14) ∀l(l∈EV(LO) ∩ P_M14 #> ∃d(d∈EV(DC) ∩ l=>d ∩
              d.msg=l.msg))
              avec P_M14 : normal-ident(l.msg)
        (M15) ∀d(d∈EV(DE) #> ∃e(e∈EV(EO) ∩ d=>e ∩ e.msg=
              retirer))
        (M16) ∀d(d∈EV(DC) #> ∃l1(l1∈EV(LL) ∩ P_M12 ∩ l1=>d) ∨
              ∃l2(l2∈EV(LO) ∩ P_M14 ∩ l2=>d))
        (M17) ∀e(e∈EV(EO) ∩ e.msg=retirer #> ∃l1(l1∈EV(LO) ∩
              P_M13 ∩ l1=>e) ∨ ∃d(d∈EV(DE) ∩ d=>e))
fin M1;

module M2
    interface
        ES-PORT P : REQ-PASS réponse REP-PASS;
    comportement
        (M21) ∀p(p∈EV(P(E)) ∩ P_M21 <#> ∃p1(p1∈EV(P(S)) ∩ p=>p1 ∩
              p1.msg=non))
              avec P_M21 : ~enregistr-pass(p.msg)
        (M22) ∀p(p∈EV(P(E)) ∩ P_M22 <#> ∃p1(p1∈EV(P(S)) ∩ p=>p1 ∩
              p1.msg=oui))
              avec P_M22 : enregistr-pass(p.msg)
fin M2
```

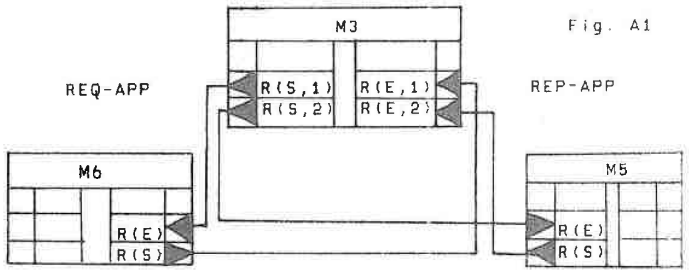


Fig. A1

```
module M3  (cf.fig. A1 où figurent les sous-ports)
    interface
        E-PORT DC : D-CONT;
        S-PORT DE : D-ECR;
        S-PORT DD : D-DIF;
        SE-PORT P : REQ-PASS réponse REP-PASS;
        SE-PORT R : REQ-APP réponse REP-APP;
    comportement
        (M31) ∀d(d∈EV(DC) ⟨⇏⟩ ∃r(r∈EV(R(S,1))∧ d=>r ∧ r.msg=
              d.msg))
        (M32) ∀r(r∈EV(R(E,1)) ∧ P_M32 ⇏ ∃p(p∈EV(P(S)) ∧ r=>p))
              avec P_M32 : r.msg.rep=oui
        (M33) ∀r(r∈EV(R(E,1)) ∧ P_M33 ⟨⇏⟩ ∃r1(r1∈EV(R(S,2))∧
              r=>r1))
              avec P_M33 : r.msg.rep=non
        (M34) ∀r(r∈EV(R(E,2))∧ P_M34 ⇏ ∃p(p∈EV(P(S)) ∧ r=>p))
              avec P_M34 : r.msg.rep=oui
        (M35) ∀r(r∈EV(R(E,2)) ∧ P_M35 ⇏ ∃d(d∈EV(DE) ∧ r=>d))
              avec P_M35 : r.msg.rep=non
        (M36) ∀p(p∈EV(P(E))∧ P_M36 ⟨⇏⟩ ∃d(d∈EV(DD) ∧ p=>d))
              avec P_M36 : p.msg.rep=oui
        (M37) ∀p(p∈EV(P(E))∧ P_M37 ⇏ ∃d(d∈EV(DE) ∧ p=>d))
              avec P_M37 : p.msg.rep=non
        (M38) ∀d(d∈EV(DE) ⇏ ∃p(p∈EV(P(E))∧ P_M37 ∧p=>d)∨
              ∃r(r∈EV(R(E,2)) ∧P_M35 ∧r=>d))
        (M39) ∀p(p∈EV(P(S)) ⇏ ∃r1(r1∈EV(R(E,1))∧P_M32 ∧ r1=>p)∨
              ∃r2(r2∈EV(R(E,2)) ∧ P_M34 ∧ r2=>p))
        (M3A) ∀r,∀p(r∈EV(R(E,1)) ∧ p∈EV(P(S))∧r=>p ⇏  ∃d(d∈
              EV(DC) ∧ d->r ∧ ~ ∃d1(d1∈EV(DC)∧ d->d1->r) ∧p.msg=
              d.msg))
        (M3B) ∀r1,∀r2(r1∈EV(R(E,1)) ∧ r2∈EV(R(E,2))∧ r1=>r2 ⇏
               ∃d(d∈EV(DC)∧ d->r1 ∧ ~∃d1(d1∈EV(DC)∧ d->d1->r1)∧
              r2.msg=d.msg))
        (M3C) ∀r,∀p(r∈EV(R(E,2))∧ p∈EV(P(S) ∧ r=>p ⇏  ∃d(d∈
              EV(DC)∧ d->r ∧ ~ ∃d1(d1∈EV(DC) ∧d->d1->r)∧ p.msg=
              d.msg))
        (M3D) ∀p,∀d(p∈EV(P(E))∧ d∈EV(DD) ∧ p=>d ⇏  ∃r1(r1∈
              EV(R(E,1)) ∧ r1=>p ∧d.msg=r1.msg.val)∧~∃r2(r2∈
              EV(R(E,2))∧ r2=>p ∧ d.msg=r2.msg.val))
fin M3;

module M4
    interface
        E-PORT DD : D-DIF;
        S-PORT A : APP;
    comportement
        (M41) ∀d(d∈EV(DD) ⟨⇏⟩ ∃a(a∈EV(A)∧ d=>a ∧a.msg=d.msg))
fin M4;

module M5
    interface
        E-PORT CA : CRE-APP;
        S-PORT T : TR-APP;
        SE-PORT R : REQ-APP réponse REP-APP;
```



```
    comportement
        (M51) ∀r(r∈EV(R(E))∧ P_M51 ⟨⇏⟩ ∃r1(r1∈EV(R(S))∧ r=>r1 ∧
              r1.msg.rep=oui))
              avec P_M51 :  ∃c(c∈EV(CA) ∧ c->r ∧ dans-base(c.msg,
               r.msg))
        (M52) ∀r(r∈EV(R(E))∧ P_M52 ⟨⇏⟩  ∃r1(r1∈EV(R(S))∧ r=>r1 ∧
              r1.msg.rep=non))
              avec P_M52 :  ~∃c(c∈EV(CA) ∧ c->r ∧
               dans-base(c.msg,r.msg))
        (M53) ∀r,∀r1(r∈EV(R(E))∧ r1∈EV(R(S))∧r=>r1 ∧ r1.msg.rep=
              oui ⇏ ∃c(c∈EV(CA) ∧ c->r ∧ dans-base(c.msg,r.msg) ∧
              r1.msg.val=c.msg))
        (M54) ∀c(c∈EV(CA)∧ P_M54 ⟨⇏⟩ ∃t(t∈EV(T)∧ c=>t ∧ t.msg=
              c.msg))
              avec P_M54 : ~dest-cette-ligne(c.msg)
              dest-cette-ligne est vrai si le message du type
              CRE-APP est destiné à cette ligne et faux sinon.
fin M5

module M6
    interface
        E-PORT T : TR-APP;
        ES-PORT R : REQ-APP réponse REP-APP;
    comportement
        (M61) ∀r(r∈EV(R(E))∧ P_M61 ⟨⇏⟩ ∃r1(r1∈EV(R(S))∧ r=>r1 ∧
              r1.msg.rep=oui))
              avec P_M61 :  ∃t(t∈EV(T) ∧ t->r ∧dans-base(t.msg,
              r.msg))
        (M62) ∀r(r∈EV(R(E))∧ P_M62 ⟨⇏⟩ ∃r1(r1∈EV(R(S)) ∧ r=>r1 ∧
              r1.msg.rep=non))
              avec P_M62 : ~∃t(t∈EV(T)∧t->r∧ dans-base(t.msg,
              r.msg))
        (M63) ∀r,∀r1(r∈EV(R(E)) ∧ r1∈EV(R(S))∧r=>r1∧ r1.msg.rep=
              oui ⇏  ∃t(t∈EV(T) ∧ t->r ∧ dans-base(t.msg,r.msg)∧
               r1.msg.val=t.msg))
fin M6;
type-message D-ECR
fin;

type-message D-CONT
    id : chaîne;
fin;

type-message REQ-APP
    id : chaîne;
fin;

type-message REP-APP
    rep : tyon;
    val : chaîne;
fin;
type tyon = (oui, non);

type-message REQ-PASS
    id : chaîne;
fin;

type-message REP-PASS
    rep : tyon;
fin;

type-message D-DIF
    app : chaîne;
fin;

type-message TR-APP
    app : chaîne;
fin;
```

b) La validation s'établit classiquement. Nous cherchons tout d'abord tous les enchaînements possibles de causalités dans le réseau de modules R à partir des entrées de S; la figure A2 les précise avec leurs conditionnements dans les encadrés.

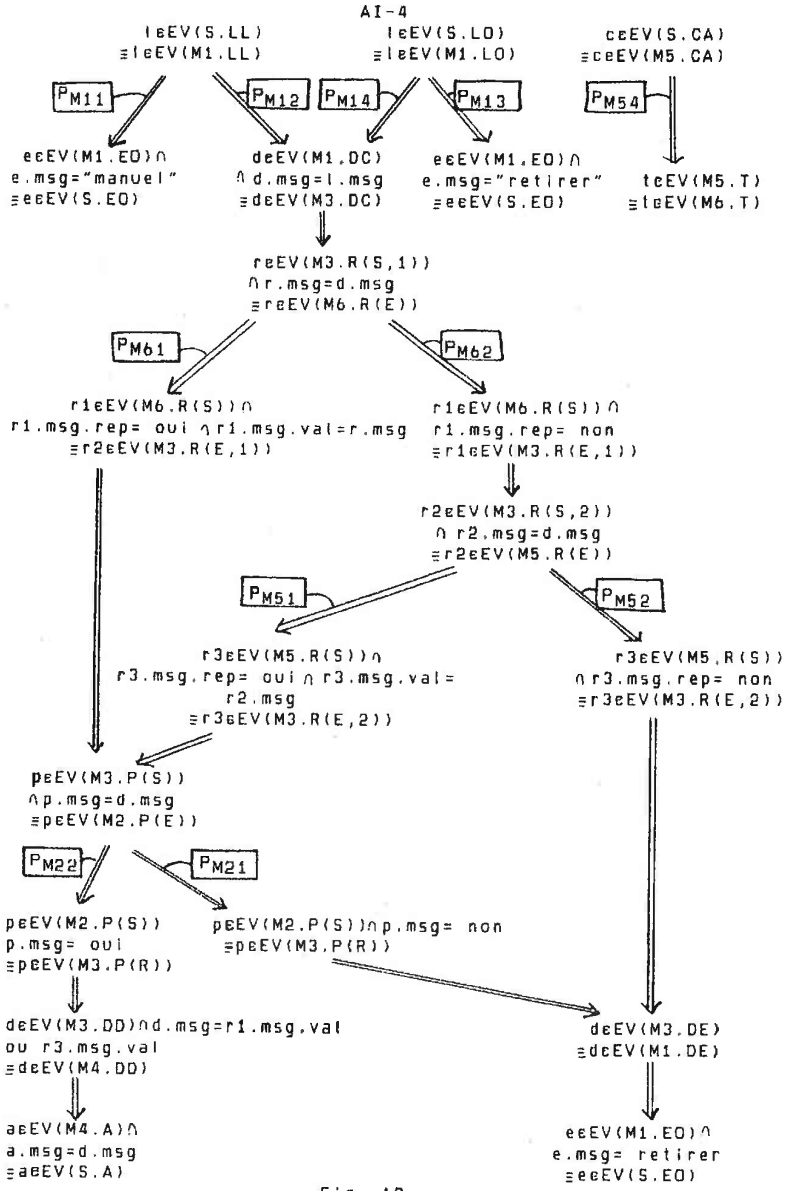


Fig. A2

Nous montrons ensuite que toutes les propriétés de S sont conservées dans le réseau R :

-(S1) Cette propriété se retrouve en (M11); comme les ports correspondants sont attachés sa conservation est vérifiée;

-(S2) Il faut que les ports impliqués dans la propriété au niveau de S soient attachés à des ports de R et qu'il existe une (des) suite(s) de déclenchements dans R qui les relient et dont le conditionnement soit équivalent au conditionnement de la propriété dans S. Au vu de la fig. A2, on note plusieurs chemins liant tεEV(S.LL) à eεEV(S.EO) avec e.msg=retirer; au sein de l'ensemble des 13 chemins possibles, le conditionnement global peut s'écrire : $P_{M12} \wedge (P_{M21} \vee P_{M52})$.
Soit :
normal-ident(l.msg) ∧ (~∃c(cεEV(M5.CA) ∧ c->r2 ∧ dans-base(c.msg,r2.msg))∨ ~enregistr-pass(p.msg))
Or :
+ p.msg = d.msg = l.msg (cf.fig. A2),
+ cεEV(M5.CA) ≡ cεEV(S.CA) (car 2 ports attachés),
+ r2<->r1<->r<->d<->l (événements en relation de causalité donc simultanés),
+ r2.msg = d.msg = l.msg (cf.fig. A2).
Donc ce conditionnement est bien équivalent à P_{S2} :
normal-ident(l.msg)∧ (~∃c(cεEV(S.CA)∧ c->l ∧dans-base (c.msg,l.msg))∨ ~enregistr-pass(l.msg))
C.Q.F.D.

-(S3) La démonstration est semblable avec un chemin possible en plus, dont le conditionnement est :
~normal-ident(l.msg) (cf.fig.A.2).

-(S4) Au vu de la figure A2, on relève que le conditionnement des chemins liant tεEV(S.LO) à aεEV(S.A) peut s'écrire :
$P_{M14} \wedge (P_{M61} \vee P_{M51}) \wedge P_{M22}$.
Soit :
normal-ident(l.msg)∧ (∃t(tεEV(M6.T) ∧ t->r ∧ dans-base (t.msg,r.msg))∨ ∃c(cεEV(M5.CA) ∧ c->r2 ∧ dans-base(c.msg, r2.msg)))∧ enregistr-pass(p.msg)
Or:
+ tεEV(M6.T) ≡ tεEV(M5.T)
+ ∀t(tεEV(M5.T) #> ∃c(cεEV(M5.CA)∧ c=>t ∧ c.msg= t.msg)) (cf. M55)
Donc:
∃t(tεEV(M6.T) ∧ t->r ∧ dans-base(t.msg,r.msg) #> ∃c(cεEV(M5.CA) ∧ c->r ∧ dans-base(c.msg,r.msg))).
Comme:
+ r.msg = l.msg
+ cεEV(M5.CA) ≡ cεEV(S.CA)
+ r2<->r1<->r<->d<->e
+ r2.msg = l.msg
+ p.msg = d.msg = l.msg
le conditionnement ci-dessus est bien équivalent à P_{S4} :

normal-ident(l.**msg**) ∧∃c(c∈**EV**(S.CA)∩ c->l ∩
dans-base(c.**msg**,l.**msg**))∩ enregistr-pass(l.**msg**)
C.Q.F.D.

-(S5) la démonstration est semblable, avec l∈**EV**(S,LL) à la place de l∈**EV**(S,LO).

-(S6) Il faut que les ports impliqués dans la propriété au niveau de S soient attachés à des ports dans R et qu'il existe des suites de déclenchement dans R permettant de remonter du port de sortie aux ports d'entrée et dont le conditionnement soit le même que celui dans S. La figure A2 nous montre ces chemins. L'équivalence des conditionnements a déjà été montrée dans (S4) et (S5).

-(S7) la démonstration est semblable à la précédente. La figure A2 nous donne les chemins concernés.

ANNEXE II

Schémas de traduction en ADA des types d'émission et de réception. cf. chapitre 4 § 2.1.b.

a) Schéma de traduction de l'émission non bloquante vers 1 ($E\bar{B}1$)

L'unité "port de sortie" est une tâche qui permet de ne bloquer la tâche appelante que très brièvement, le temps de lui passer le message à transmettre (les deux tâches sont sur le même site et la tâche "port de sortie" est dédiée à cet effet) :

Tâche appelante :

```
        :
        :
    S.$NB_ENVOYER(msg);  (tous les noms introduits pour les
        :                besoins de la traduction commencent
        :                par $; msg est le message du type M
                         à transmettre)
```

Tâche "port de sortie" :

```
    TASK S IS
      ENTRY $NB_ENVOYER($X : IN M);
    END S;
    TASK BODY S IS
      $M : M;
    BEGIN
      LOOP
        ACCEPT $NB_ENVOYER($X : IN M) DO
            $M := $X;
        END $NB_ENVOYER;
            :
            :
        -- transmission de $M au module appelé;
            :
            :
      END LOOP;
    END S;
```

Pour réaliser la transmission de $M au module appelé, on peut penser à un appel d'entrée $TRANSMETTRE vers une tâche dédiée "port d'entrée" (E) qui consommera $M et rendra la main immédiatement. Soit l'appel :

```
Module-appelé.E.$TRANSMETTRE($M);   (E est le nom du port d'
                                     entrée auquel est lié S)
```

Mais il s'agit d'un appel potentiellement à distance, qui peut être perdu ou retardé, bloquant ainsi S et par voie de conséquence la tâche appelante lors de sa prochaine émission. Ceci n'est pas acceptable comme traduction générale d'une émission non bloquante : il faut donc pouvoir tamponner l'émission ; comme tout échange à distance est bloquant, la seule issue est la création dynamique de tâches locales $MESS_A_TRANSM qui seront bloquées mais qui libèreront S et donc la tâche appelante. On a donc prévu un

tableau de pointeurs vers ces tâches (dont la taille correspond à la taille d'un tampon à l'émission). Ce tamponnage est réalisé par une procédure $TAMPONNER, appelée par S et située dans un paquetage $TAMPON;

```
$TAMPON.$TAMPONNER($M)
```

Le paquetage $TAMPON s'écrit :

```
PACKAGE $TAMPON IS
  PROCEDURE $TAMPONNER($X:IN M);
END $TAMPON;
PACKAGE BODY $TAMPON IS
  TASK TYPE $MESS_A_TRANSM IS
     ENTRY $CONFIER($X : IN M);
  END $MESS_A_TRANSM;
  TASK BODY $MESS_A_TRANSM IS
     $M : M;
  BEGIN
     ACCEPT $CONFIER($X : IN M) DO
       $M := $X;
     END $CONFIER;
     Module-appelé.E.$TRANSMETTRE($M);  -- Toutes ces
                -- transmissions seront prises en compte
                -- dans leur ordre de création.
  END $MESS_A_TRANSM;
  TYPE $POINTEUR IS ACCESS $MESS_A_TRANSM;
  $TAILLE : CONSTANT INTEGER := taille-du-tampon;
  $TABLEAU : ARRAY(1..$TAILLE) OF $POINTEUR;
  $INDICE : INTEGER RANGE 1..$TAILLE :=1;
  PROCEDURE $TAMPONNER($X : IN M) IS
  BEGIN
     IF NOT $TABLEAU($INDICE).ALL'TERMINATED
          THEN ABORT $TABLEAU($INDICE)
      ENDIF;   -- en cas de débordement, le message le
               -- plus ancien est donc perdu
     $TABLEAU($INDICE):=NEW $MESS_A_TRANSM;
     $TABLEAU($INDICE).$CONFIER($X);
     $INDICE:=$INDICE MOD $TAILLE +1;
  END $TAMPONNER;
END $TAMPON;
```

b) schéma de traduction de l'émission non bloquante vers n ($\overline{EB}$n)

La tâche appelante est identique; la tâche "port de sortie en diffusion" (S) comporte n appels de procédure $TAMPONNER successifs au lieu d'un seul, dans n paquetages $TAMPONi :

```
$TAMPON1.$TAMPONNER($M);
$TAMPON2.$TAMPONNER($M);
      :
$TAMPONn.$TAMPONNER($M);
```

Pour ce faire, le paquetage $TAMPON aura été déclaré en générique :


```
GENERIC $TAMPON IS          -- REMARQUE :la taille du tampon
                            -- pourrait être passée en
                            -- paramètre.
      .
      .
      .
END $TAMPON;
```

n exemplaires en seront générés dans la tâche S :

```
TASK BODY S IS
  PACKAGE $TAMPON1 IS NEW $TAMPON;
      :
  PACKAGE $TAMPONn IS NEW $TAMPON;
      :
END S;
```

c) schéma de traduction de l'émission non bloquante vers 1 parmi n ($\overline{EB}$1/n) :

Le destinataire est choisi par la tâche appelante (éventuellement au hasard); le message est transmis à la tâche S, qui le tamponne en fonction de l'identité de son destinataire.

Paquetage du module ou paquetage des objets globaux à l'application :

```
SUBTYPE $TYPID IS INTEGER RANGE 1..n; (identification
                                        des destinataires)
      :
```

Tâche appelante :

```
I : $TYPID;
      :
-- détermination du destinataire I,éventuellement au
S.$NB_ENVOYER(msg,I);                          -- hasard
      :
```

Tâche "port de sortie" :

```
TASK S IS
  ENTRY $NB_ENVOYER($X : IN M, $Y : IN $TYPID);
END S;
TASK BODY S IS
  PACKAGE $TAMPON1 IS NEW $TAMPON;
      :
  PACKAGE $TAMPONn IS NEW $TAMPON;
  $M : M;
  $I : $TYPID;
BEGIN
  LOOP
     ACCEPT $NB_ENVOYER($X : IN M, $Y : IN $TYPID) DO
          $M := $X;
          $I := $Y;
     END $NB_ENVOYER;
```

```
        CASE $I IS
             WHEN id1 => $TAMPON1.$TAMPONNER($M);
                  :
             WHEN idn => $TAMPONn.$TAMPONNER($M);
        END CASE;
      END LOOP;
    END S;
```

d) schéma de traduction de l'émission bloquante en attente de réponse vers 1, avec ou sans time-out (EBR1[(t)]) :

Dans cette traduction et les suivantes nous avons prévu la possibilité de perte et de duplication de message. Les traductions se simplifient si ce n'est pas géré au niveau application. L'unité "port de sortie/entrée" est une procédure (car la tâche appelante est de toute façon bloquée) qui gère le délai d' attente sur la réception de la réponse de fin de service. Les éléments entre [ ] ne sont présents qu'en cas de time-out.

Tâche appelante :

```
[DECLARE
$TIMEOUT : EXCEPTION;
BEGIN]
  SE(msg1,msg2 [ ,t]);
  [TMT1;
EXCEPTION                          -- prise en compte de l'
  WHEN $TIMEOUT => TMT2;           -- exception créée et
END ;]                             -- propagée par SE;
```

Procédure "port de sortie/entrée" :

```
PROCEDURE SE($X: IN M1,$Y: OUT M2[,$T: IN DURATION]) IS
   USE $NUM_SE; -- paquetage de numérotation des
                -- messages sur le port SE;
  [$LIMITE : CALENDAR.TIME;]
  $N : INTEGER;
BEGIN
  [$LIMITE := CALENDAR.CLOCK + $T;]
  SELECT
     Module-appelé.ES.$DEBUT_SERVICE($X,$SUIVANT);
          -- ES est le nom du port auquel SE est lié;
          -- le message $X est déposé avec son no;
     LOOP
       SELECT
         Module-appelé.ES.$FIN_SERVICE($Y,$N);
          -- attente du message $Y en retour et du no;
         EXIT WHEN $TEST($N);
       [OR DELAY $LIMITE-CALENDAR.CLOCK;
          RAISE $TIMEOUT;          -- timeout sur durée de
          EXIT;]                   -- service excessive.
       END SELECT;
     END LOOP;
  [OR DELAY $T;
     RAISE $TIMEOUT;]              -- timeout sur dépôt
  END SELECT;                      -- de $X impossible.
 [EXCEPTION                       -- propagation de l'
  WHEN $TIMEOUT => RAISE;]        -- exception à la tâche
END SE;                           -- appelante.
```



```
PACKAGE $NUM_SE IS
     FUNCTION $TEST($X : IN INTEGER) RETURN BOOLEAN;
     FUNCTION $SUIVANT RETURN INTEGER;
END $NUM_SE;
PACKAGE BODY $NUM_SE IS
     $N : INTEGER := 0;
     FUNCTION $SUIVANT RETURN INTEGER IS
          $N := ($N + 1) MOD n; -- numérotation cyclique;
          RETURN $N;
     END $SUIVANT;
     FUNCTION $TEST($X : IN INTEGER) RETURN BOOLEAN IS
          RETURN $X = $N;
     END $TEST;
END $NUM_SE;
```

e) schéma de traduction de l'émission bloquante en attente de réponse vers 1 parmi n, avec ou sans time-out (EBR1/n[(t)]) :

Le destinataire est choisi par la tâche appelante (éventuellement au hasard) et transmis à SE :

Tâche appelante :

```
      :
I : $TYPID;
      :
-- détermination du destinataire I (éventuellement au
[DECLARE                                  -- hasard)
  $TIMEOUT : EXCEPTION;
BEGIN]
  SE(msg1,msg2 [,t],I);
   TMT1;
[EXCEPTION
  WHEN $TIMEOUT => TMT2;
END;]
```

Procédure "port de sortie/entrée en sélection" (SE) :

```
PROCEDURE SE($X: IN M1, $Y:OUT M2 [,$T: IN DURATION]
,$I: IN $TYPID) IS
  USE $NUM_SE;
  $N : INTEGER;
  [$LIMITE : CALENDAR.TIME;]
BEGIN
  [$LIMITE := CALENDAR.CLOCK + $T ;]
  CASE $I IS
     WHEN id1 => même traduction qu'en d) pour le
                 module appelé1
           :
     WHEN idn => même traduction qu'en d) pour le
                 module appelén
  END CASE;
           :
END SE;
```

f) schéma de traduction de l'émission bloquante en attente d'acquittement vers 1, avec ou sans time-out (EBA1[(t)]) :

Même traduction qu'en d) avec msg à la place de msg1, S à la place de ES et supression de msg2 et de $Y (pas de message en réponse).

g) schéma de traduction de l'émission bloquante en attente d'acquittement vers 1 parmi n, avec ou sans time-out (EBA1/n[(t)]) :

Même traduction qu'en e) avec msg à la place de msg1, Si à la place de ESi et suppression de msg2 et de $Y.

h) schéma de traduction de l'émission bloquante en attente d'acquittement vers n, avec ou sans time-out (EBAn[(t)]) :

La tâche appelante passe la main à une procédure S qui lance n tâches $Si gérant en parallèle les n émissions bloquantes et les interroge sur leur terminaison ou leur time-out éventuel.

Tâche appelante :

```
    [DECLARE
         $TIMEOUT : EXCEPTION;
    BEGIN]
         S(msg [,t]);
         [TMT1;
    EXCEPTION
         WHEN $TIMEOUT => TMT2;
    END;]

    PROCEDURE S($X : IN M [, $T : IN DURATION]) IS
         [$T1, ... , $Tn : BOOLEAN;]
    BEGIN
         $S1.$INIT($X [,$T]);
         :
         $Sn.$INIT($X [,$T]);
         $S1.$FIN[($T1)];
         :
         $Sn.$FIN[($Tn)];
         [IF $T1 OR .... OR $Tn THEN RAISE $TIMEOUT;]
    END;

    TASK $Si IS
         ENTRY $INIT($X : IN M [,$T : IN DURATION]);
         ENTRY $FIN [($Y : OUT BOOLEAN)];
    END $Si;
    TASK BODY $Si IS
         USE $NUM_S;
         $M : M;
         [$T1 : DURATION; $LIMITE : CALENDAR.TIME; $TIMEOUT :
         BOOLEAN;]
         $N : INTEGER;
```



```
    BEGIN
      LOOP
         [$TIMEOUT := FALSE;]
         ACCEPT $INIT($X : IN M [,$T : IN DURATION]) DO
           $M := $X;
           [$T1 := $T;]
         END $INIT;
         [$LIMITE := CALENDAR.CLOCK+$T1;]
         SELECT
           module-appelé-i.E.$DEBUT_SERVICE($M,$SUIVANT);
           LOOP
              SELECT
                module-appelé-i.E.$FIN_SERVICE($N);
                EXIT WHEN $TEST($N);
              [OR DELAY $LIMITE-CALENDAR.CLOCK;
                $TIMEOUT := TRUE;
                EXIT;]
              END SELECT;
           END LOOP;
           [OR DELAY $T
              $TIMEOUT := TRUE;]
         END SELECT;
         ACCEPT $FIN [($Y : OUT BOOLEAN) DO
           $Y := $TIMEOUT;
         END $FIN;]
      END LOOP;
    END $Si;
```

i) schéma de traduction de la réception avec réponse depuis 1, avec ou sans time-out (RR1[(t)]) :

L'unité "port d'entrée/sortie" est une tâche qui accepte la demande de début de service, puis demande le service à la tâche appelée, puis accepte la demande de fin de service.

Tâche ES:

```
         TASK ES IS
              ENTRY $DEBUT_SERVICE($X:IN M1, $N0:IN INTEGER);
              ENTRY $FIN_SERVICE($Y:OUT M2, $N1:OUT INTEGER);
         END ES;
         TASK BODY ES IS
              $M1:M1; $M2:M2;
              $N:INTEGER; $PRECED:INTEGER:=0;
         BEGIN
              LOOP
                ACCEPT $DEBUT_SERVICE($X:IN M1,N0:IN INTEGER) DO
                   $M1:=$X;$N:=$N0;
                END $DEBUT_SERVICE;
                IF  $N>$PRECED THEN $PRECED:=$N; --écarte mes-
                   tâche-appelée.ES($M1,$M2); --sages dupliqués
                   ACCEPT $FIN_SERVICE($Y:OUT M2, $N1:OUT
                     INTEGER) DO
                     $Y:=$M2; $N1:=$N;
                   END $FIN_SERVICE;
                ENDIF;
              END LOOP;
         END ES;
```

Tache appelée:

```
ENTRY ES(msg1:IN M1,msg2:OUT M2);
      :
SELECT
  ACCEPT ES(msg1:IN M1,msg2:OUT M2) DO
    TMT1;        --service (avec; msg2:= ....);
  END ES;
[OR DELAY t;
  TMT2;]
END SELECT;
      :
```

j) schéma de traduction de la réception avec réponse depuis 1 parmi n, avec ou sans time-out (RR1/n[(t)]) :

Il faut n tâches ESi identiques à ES du paragraphe précédent. La tâche appelée s' écrit :

```
ENTRY ES(1..n)(msg1:IN M1,msg2:OUT M2); -- famille d'
      :                                 -- entrées;
I:$TYPID;
      :
 -- détermination de l'appelant I (éventuellement au
                                      hasard)
SELECT
  ACCEPT ES(I)(msg1:IN M1,msg2:OUT M2)
      DO TMT1;
  END ES(I);
  [OR DELAY t;
      TMT2;]
END SELECT;
```

Chaque tâche ESi appelle ES(i).

k) schéma de traduction de la réception avec acquittement depuis 1, avec ou sans time-out (RA1[(t)]) :

La traduction est la même qu'en i) avec :
{msg à la place de msg1, E à la place de ES,
{suppression de msg2, $Y, $M2

l) schéma de traduction de la réception avec acquittement depuis 1 parmi n, avec ou sans time-out (RA1/n[(t)]) :

La traduction est la même qu'en j) avec :
{msg à la place de msg1, Ei à la place de ESi,
{suppression de msg2

m) schéma de traduction de la réception sans acquittement depuis 1, avec ou sans time-out (RA̅1([t,] gestion)) :

m1) gestion en FIFO :

Tâche appelée :


```
ENTRY E(msg:IN M);
:
SELECT
  ACCEPT E(msg:IN M) DO
     TMT1;
  END E;
  [OR DELAY t;
     TMT2;]
END SELECT;
:
```

Tâche "port E" : elle prend en compte les messages les uns après les autres sur l'entrée $TRANSMETTRE où ils sont gérés en FIFO, et les propose à la tâche appelée.

```
TASK E IS
  ENTRY $TRANSMETTRE($X:IN M);
END E;
TASK BODY E IS
  $M:M;
BEGIN
  LOOP
     ACCEPT $TRANSMETTRE($X:IN M) DO
       $M:=$X;
     END $TRANSMETTRE;
     tâche-appelée.E($M);
  END LOOP;
END E;
```

m2) gestion en LIFO, HASARD, DRR ou DR̅R̅ :

La tâche appelée est identique. La tâche "port E" doit toujours proposer le dernier message reçu et ne peut donc rester bloquée en attente de prise en compte d'un message. Il faut donc recourir comme au paragraphe a) ci-avant, à la gestion dynamique d'une tâche $MESSAGE_ELU qui reçoit le message à transmettre et se met en attente de consommation par la tâche appelée. Si un message devant être élu selon la politique de gestion en vigueur survient, la tâche $MESSAGE_ELU est créée par E ,si elle n'existait pas ou avortée et recréée, sinon. Lorsqu'un message est consommé par la tâche appelée, la tâche $MESSAGE_ELU le fait savoir à la tâche E (par $SUIVANT), pour qu'un nouveau message lui soit proposé.

```
TASK E IS
     ENTRY $TRANSMETTRE($X:IN M);
     ENTRY $SUIVANT;
END E;
TASK BODY E IS
     $M:M;
BEGIN
     LOOP
       SELECT
          ACCEPT $TRANSMETTRE($X:IN M) DO
            $M:=$X;
          END $TRANSMETTRE;
          --rangement de $M dans le tampon; extraction
          --selon la politique du message à passer(dans
          --$M) et passage à la tâche $MESSAGE_ELU;
```

```
            OR ACCEPT $SUIVANT;
               --extraction du tampon du prochain message à
               --passer selon la politique, s'il y en a un,
               --(dans $M); passage à la tâche $MESSAGE_ELU;
            END SELECT;
         END LOOP;
      END E;
```

Le passage se fait en appelant une procédure $PASSER dans le paquetage $PASSAGE :

```
$PASSAGE.$PASSER($M).
```

Le paquetage $PASSAGE s'écrit :

```
      PACKAGE $PASSAGE IS
           PROCEDURE $PASSER($X:IN M);
      END $PASSAGE;
      PACKAGE BODY $PASSAGE IS
           TASK TYPE $MESSAGE_ELU IS
                ENTRY $CONFIER($X:IN M);
           END $MESSAGE_ELU;
           TASK BODY $MESSAGE_ELU IS
                $M:M;
           BEGIN
                ACCEPT $CONFIER($X:IN M) DO
                  $M:=$X;
                END $CONFIER;
                tâche-appelée.E($M);
                E.$SUIVANT;
           END $MESSAGE_ELU;
           TYPE $ACCES IS ACCESS $MESSAGE_ELU;
           $ELU:$ACCES;
           PROCEDURE $PASSER($X:IN M) IS
           BEGIN
                IF NOT $ELU.ALL'TERMINATED
                  THEN ABORT $ELU;
                END IF;
                $ELU:=NEW $MESSAGE_ELU;
                $ELU.$CONFIER($X);
           END $PASSER;
      END $PASSAGE;
```

n) schéma de traduction de la réception sans acquittement depuis 1 parmi n, avec ou sans time-out (RÃ1/n([t,] gestion)) :

Il faut n tâches Ei, identiques à E du paragraphe m). La tâche appelée s'écrit :

```
      ENTRY ES(1..n)(msg:IN M);
            :
      I:$TYPID;
            :
      -- détermination de l'appelant I (éventuellement au
                                          -- hasard);
```



```
SELECT
 ACCEPT ES(I)(msg:IN M) DO
   TMT1;
END ES(I);
[OR DELAY t;
   TMT2;]
END SELECT;
```

Chaque tâche ESi appelle ES(i).

ANNEXE III

Proposition de définition d'un langage de spécification détaillé (LSD).

cf. chap. 4 § 2.2.

Toutes les notions structurantes nommées (applications, modules, processus, types de messages, etc.) seront décrites selon le modèle suivant :

```
NOTION nom-notion
     [PRESENTATION                                  [....] : clause
          texte de présentation]                          facultative.
     DESCRIPTION
          description formelle
FIN-NOTION
```

Le texte de présentation facultatif peut prendre diverses formes selon les notions concernées et les préférences du concepteur :

- texte en langage naturel,
- spécification en termes des relations de précédence pour les modules ou processus,
- pré et post conditions pour les fonctions, sous-programmes, opérations, etc.

a) Description d'une application :

```
APPLICATION nom-application
     [PRESENTATION
          texte de présentation]
     DESCRIPTION
          MODULES
               liste-descriptions-modules
          LIAISONS
               liste-descriptions-liaisons (externes
               et inter-modules)
          MESSAGES
               liste-descriptions-types-messages
               (utilisés sur les liaisons ci-dessus)
          TYPES
               liste-déclarations-types-données
               (utiles pour décrire les types de messages
               ci-dessus)
          [REPARTITION
               liste des sites et des assignations de
               modules aux sites ]
FIN-APPLICATION
```

b) Les types de messages :

```
MESSAGE nom-message
     [PRESENTATION
          texte de présentation]
     DESCRIPTION
          {VIDE          }                    {....}
          {liste-champs  }                    {....} : choix.
FIN-MESSAGE                                   {....}
```

Un champ est décrit classiquement par :
nom-champ : type-données
ou par :
CAS nom-champ$_1$ (un champ de type scalaire du message)
QUAND liste-valeurs$_1$ =>nom-champ$_2$: type-données$_1$
QUAND liste-valeurs$_2$ =>nom-champ$_2$: type-données$_2$
...
[**AUTREMENT** nom-champ$_2$: type-données$_n$]
FIN-CAS
Exemple :
CAS no-produit
QUAND 1, 2 => besoin : **REEL**
QUAND 3, 4 => besoin : **BOOLEEN**
AUTREMENT besoin : **ENTIER**
FIN-CAS
Fin exemple.

c) Les liaisons :

Une liaison n'est pas nommée; sa description prend l'une des trois formes suivantes :
- liaison bipoint :
  désignation-port **VERS** désignation-port
- liaison multipoint en diffusion :
  désignation-port **VERS** désignation-port **ET** ... **ET** désignation-port
- liaison multipoint en sélection :
  désignation-port **VERS** [nom-chemin :] désignation-port **OU** ... **OU** [nom-chemin :] désignation-port

Désignation-port prend une des formes suivantes :
- nom-port
- nom-module.nom-port
- **EXTERIEUR** (les liaisons externes sont fictivement terminées par un port **EXTERIEUR**).

Dans une liaison multipoint en sélection tous les chemins sont nommés ou aucun ne l'est; dans le premier cas, la sélection d'un chemin précis au niveau des primitives de réception est possible, alors que dans le second cas, la sélection est nécessairement indéterministe.

d) Les modules :
MODULE nom-module
[**PRESENTATION**
texte de présentation]
DESCRIPTION
INTERFACE
liste-descriptions-ports-module
CORPS
DECLARATIONS
liste-déclarations (constantes, variables, types, fonctions, procédures, utilisables par tous les processus du module)
PROCESSUS
liste-descriptions-processus
LIAISONS
liste-descriptions-liaisons (internes aux modules)

ATTACHEMENTS
liste-desriptions-attachements
MESSAGES
liste-descriptions-types-messages (ceux utilisés sur les liaisons internes)
FIN-MODULE

Un attachement est décrit par :
- nom-E(ou ES)-port de module **A** nom-E(ou ES)-port de processus
- ou nom-S(ou SE)-port de processus **A** nom-S(ou SE)-port de module.

e) Les ports de modules :

Ils établissent uniquement l'interface externe des modules (les propriétés attachées aux ports seront précisées au niveau des processus accueuillant ces ports).

Il y a quatre types de ports :
- port d'entrée :
  E-PORT nom-port : type-message
- port de sortie :
  S-PORT nom-port : type-message
- port d'entrée/sortie : (correspond à un service rendu)
  ES-PORT nom-port : type-message1 **REPONSE** type-message2
- port de sortie/entrée : (correspond à une demande de service)
  SE-PORT nom-port : type-message1 **REPONSE** type-message2

f) Les processus :

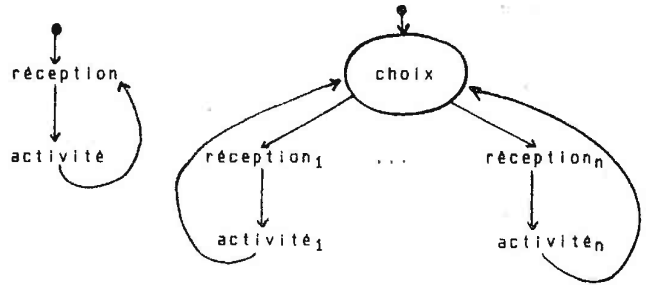
On distingue processus "normaux" et processus d'interruption. Leur durée de vie est celle de l'application.

Un processus normal est décrit par :
PROCESSUS nom-processus [**PRIORITE** entier>=0]
[**PRESENTATION**
texte de présentation]
DESCRIPTION
INTERFACE
liste-descriptions-ports-processus
CORPS
DECLARATIONS
liste-déclarations (constantes, variables, types, procédures, fonctions, exceptions)
TRAITEMENTS
traitements
FIN-PROCESSUS

Les priorités des processus servent à résoudre les éventuels conflits d'allocation des processeurs d'un site aux processus prêts à s'exécuter (c'est à dire non bloqués dans une action de communication ou dans une phrase de délai). L'ordonnancement des processus normaux peut être préemptif ou non.

La partie traitements comporte une boucle infinie qui inclut une ou plusieurs "activités", déclenchées par une réception de message. Lorsqu'il y a plusieurs "activités",

l'aiguillage entre elles se fait par une phrase de choix (cf. §11); chaque "activité" peut comporter éventuellement des choix et réceptions, c'est à dire des "sous-activités" :



Un processus d'interruption est décrit par :

PROCESSUS INTERRUPTION nom-processus [**PRIORITE** entier>=0]
Même description que pour un processus normal si ce n'est que la clause **INTERRUPTION** est associée aux E-PORT d'interruption.
FIN-PROCESSUS

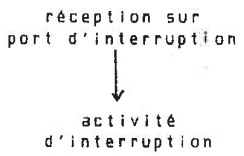
Une interruption est matérialisée par l'arrivée d'un message sur un port d'interruption :

E-PORT nom-port : type-message **INTERRUPTION**

Chaque port d'interruption est associé à un type d'interruption physique et un type d'interruption physique ne peut être associé qu'à un seul port d'interruption. Le détail de la liaison entre message d'interruption et interruption physique n'est pas spécifié à ce niveau (il peut cependant être décrit informellement dans la partie **PRESENTATION** du processus d'interruption.

Implicitement, la priorité d'un processus d'interruption est supérieure à celle de tous les processus normaux : la priorité qui peut figurer, permet de hiérarchiser les interruptions de plusieurs processus d'interruption.

La partie traitements comporte une réception sur le port d'interruption suivie de l'"activité" d'interruption :



g) Les ports de processus : (hors ports d'interruption)

Par rapport à la description d'un port de module on trouve en plus, pour les E-PORT et les ES-PORT, une éventuelle clause de priorité et pour les E-PORT une éventuelle clause de consommation (traduisant l'existence d'un tampon et fixant sa politique de gestion).

. **E-PORT** nom-port : type-message [**PRIORITE** entier>=0]
[**CONSOMMATION** { **DERNIER** { **AVEC** } **REMISE** | **FIFO** | **LIFO** | **HASARD** | ... }]
{ **SANS** }

. **ES-PORT** nom-port : type-message1 [**PRIORITE** entier>=0]
REPONSE type-message2

La priorité peut être utile lorsqu'il y a des choix entre plusieurs réceptions dans le corps des processus (cf. primitive **SELECT** ci-après).

La clause de consommation est utile lorsqu'une émission non bloquante crée une accumulation de messages sur un port d'entrée. La consommation peut être :

- non exhaustive : le dernier message reçu seulement est consommé avec ou sans remise (dans ce dernier cas le même message peut être consommé plusieurs fois); les autres messages sont détruits.
- exhaustive : tous les messages sont consommés selon diverses politiques (FIFO, LIFO, HASARD, ...); un tampon de taille suffisante est supposé exister. Dans la réalité ceci ne peut pas toujours être garanti et au niveau des langages de programmation le cas du tampon plein doit être considéré (blocage de l'émetteur, écrasement, exception,...)

Dans une liaison interne à un module, un port de processus peut être désigné par :
- nom-port
- nom-processus.nom-port
(**EXTERIEUR** ne doit pas apparaître ici).

h) Les déclarations :

Elles peuvent apparaître dans n'importe quel ordre :
- déclaration de constante symbolique :
CONST nom-constante : type-données [:= valeur]
- déclaration de variable :
VAR nom-variable : type-données [:= valeur]
L'affectation d'une valeur est possible pour les types scalaires. Les notations classiques jouent pour l'écriture des constantes.
- déclaration de type :
En plus des types prédéfinis **ENTIER**, **REEL**, **BOOLEEN**, **CHAINE**, on peut déclarer
  - des types structure
TYPE nom-type : **ENUMERATION** liste-valeurs **FIN-TYPE**

- des type énumération
  TYPE nom-type : **STRUCTURE** liste-champs **FIN-TYPE**
- des types abstraits
  TYPE nom-type
  [PRESENTATION
  texte descriptif **]**
  DESCRIPTION
  liste-opérations
  FIN-TYPE
  Une opération est caractérisée par son "profil" et un texte libre pour définir informellement sa sémantique :
  OPERATION
  [PRESENTATION
  texte descriptif **]**
  PROFIL
  domaine -> domaine
  FIN-OPERATION
  où domaine s'écrit : nom-type ou nom-type_1 X...X nom-type_n.

- déclaration d'exception :
  Elle peut apparaître au niveau d'un module ou d' une fonction ou procédure. Ceci fixe la portée de son nom, c'est à dire la partie de l'application où l'exception peut être prise en compte grâce à un récupérateur portant son nom. Ailleurs, elle peut être récupérée anonymement en même temps que d'autres exceptions (cf. § i13, ci après).
  EXCEPTION nom-exception

- déclaration de fonction et procédure :
  FONCTION nom-fonction (liste-paramètres-formels typés)
  RETOURNE nom-type
  [PRESENTATION
  texte descriptif**]**
  DESCRIPTION
  DECLARATIONS
  liste-déclarations (constantes, variables, types, fonctions, procédures, exceptions locales à la fonction)
  TRAITEMENTS
  traitements
  FIN-FONCTION
  où les paramètres formels typés s'écrivent
  nom-paramètre : type-données

  Dans son code, le résultat de la fonction est désigné par nom-fonction et se manipule comme une variable locale ayant pour type celui indiqué après **RETOURNE**.

  PROCEDURE nom-procédure (paramètre-formel$_1$ {**ENTREE** | **SORTIE** | **MAJ**} : type-données$_1$, ..., paramètre-formel$_n$ {**ENTREE** | **SORTIE** | **MAJ**} : type-données$_n$)
  [PRESENTATION
  texte descriptif**]**

  DESCRIPTION
  DECLARATIONS
  liste-déclarations (constantes, variables, types, fonctions, procédures, exceptions locales à la procédure)
  TRAITEMENTS
  traitements
  FIN-PROCEDURE

i) **Les traitements :**
Une partie traitements (ou code) est constituée par un ensemble de phrases séparées par des ";".
On distingue les types de phrases suivants :

i1) phrase de commentaire :
% texte de commentaire**%**

i2) phrase vide :
RIEN

i3) phrase conditionnelle :
SI condition **ALORS** code1**[SINON** code2**]**

i4) phrase de cas:
CAS variable (de type scalaire)
QUAND liste-valeurs$_1$ => code$_1$
QUAND liste-valeurs$_2$ => code$_2$
...
[AUTREMENT code$_n$**]**
FIN-CAS

i5) phrases répétitives :
REPETER code **FIN-REPETER**

TANT QUE condition **REPETER** code **FIN-REPETER**

REPETER code **JUSQUA** condition **FIN-REPETER**

i6) phrases d'appel :
nom-procédure(liste-paramètres-effectifs)

nom-opération(liste-paramètres-effectifs)

Les paramètres effectifs sont des constantes, des variables ou des appels de fonctions.
Peuvent être appelées des fonctions, procédures et opérations déclarées dans le processus concerné ou des fonctions, procédures et opérations déclarées au niveau du module englobant le processus concerné.

i7) phrase d'affectation :
variable := {constante | variable | nom-fonction(liste-paramètres-effectifs)}

Le langage n'offre pas d'arithmétique; les calculs sont décrits informellement dans des paragraphes, procédures ou

fonctions. L'affectation est cependant utile pour faire évoluer les variables d'état.

Les conditions sont elles aussi réduites en conséquence : variables ou fonctions à résultat booléen ou expressions de comparaison (=,≠,>=,<=,<,>,**ET**,**OU**,**NON**).

i8) phrase d'attente :

ATTENDRE délai

où délai est un réel positif ou nul représentant un nombre de secondes.

i9) phrase de retour (en fin de procédure ou fonction) :

RETOURNER

i10) phrases de communication :

- émission bloquante en attente de réponse ou d'acquittement:

B-ENVOYER message **SUR** { nom-S-PORT [**VOIE** nom-chemin] / nom-SE-PORT [**VOIE** nom-chemin] **ATTENDRE** nom-message }
[**ALORS** code1 **QUAND DELAI** délai => code2
FIN-B-ENVOYER]

où message s'écrit : { nom-message / ( ) / (liste-valeurs-scalaires) } pour un signal

Un délai maximum de blocage peut être fixé. Le déblocage se fait sur réception du (ou des) acquittement(s) de prise en compte ou de la réponse. En cas de dépassement du délai, les messages ou acquittements qui surviennent postérieurement et avant la prochaine émission, sont détruits.

- émission non bloquante :

NB-ENVOYER message **SUR** nom-S-PORT [**VOIE** nom-chemin]

- réception bloquante :

B-RECEVOIR nom-message **SUR** { nom-E-PORT [**VOIE** nom-chemin] [**AVEC ACQUITTEMENT**] / nom-ES-PORT [**VOIE** nom-chemin] **ALORS** code **REPONDRE** message }

Un délai maximum d'attente peut être fixé en imbriquant la réception dans une phrase de CHOIX avec délai, le déblocage se faisant sur réception d'un message.

Dans toutes les phrases de réception les messages ont implicitement le type spécifié au niveau du port.

i11) phrase de choix indéterministe :

Il s'agit d'une construction essentielle pour le type d'applications considéré :

CHOIX
QUAND garde1 => code$_1$
QUAND garde2 => code$_2$
...
[**QUAND DELAI** délai => code$_n$] } "branches"
FIN-CHOIX

Une garde est constituée d'une condition booléenne (facultative) et d'une phrase de réception. La garde ne peut pas porter sur le contenu des messages en réception.

Exemple :

```
CHOIX
     QUAND PLEIN<n B-RECEVOIR PRODUIRE SUR P
          => ..............
     QUAND PLEIN>0 B-RECEVOIR CONSOMMER SUR C
          => ..............
FIN-CHOIX
```

Fin exemple.

Une branche est ouverte si l'expression booléenne est vraie (si aucune branche n'est ouverte une exception standard est déclenchée). Une branche est franchissable si elle est ouverte et si la phrase de réception est débloquée. Si plusieurs branches sont simultanément franchissables le choix se fait en fonction des priorités associées aux ports ou à défaut de manière indéterministe. La branche de DELAI est exécutée lorsqu'aucune autre branche n'est franchissable après expiration du délai. Un délai nul permet de construire une réception non bloquante.

i12) phrase de signalement d'exception :

SIGNALER nom-exception

Peut apparaître dans la partie traitements des sous-programmes et des processus ou dans les récupérateurs d'exception (pour propager explicitement une exception).

i13) le récupérateur d'exceptions :

```
EXCEPTION[liste-noms-exception]
     [PRESENTATION
          texte de présentation]
     DESCRIPTION
          code
FIN-EXCEPTION
```

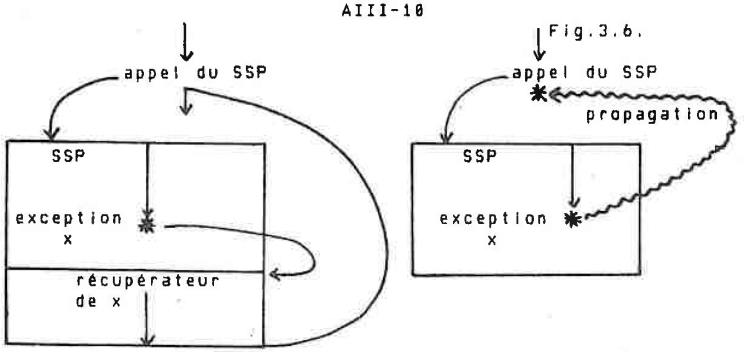
Le mot EXCEPTION est suivi d'un nom ou d'une liste de noms d'exceptions visibles. Si un récupérateur ne porte pas de nom, il récupère toutes les exceptions non récupérées par ailleurs de l'unité à la fin de laquelle il se trouve. Il y a un récupérateur au plus par nom d'exception et par unité.

Les exceptions peuvent se produire dans la partie traitements des sous-programmes et des processus. Le LSD ne permet pas de spécifier un traitement particulier en cas d'exception dans un récupérateur (implicitement l'exception est ignorée).

Le schéma de récupération varie suivant la nature de l'unité dans laquelle se produit l'exception :

- pour les sous-programmes, le schéma est du type "terminaison" (sur le modèle ADA):

Fig.3.6.

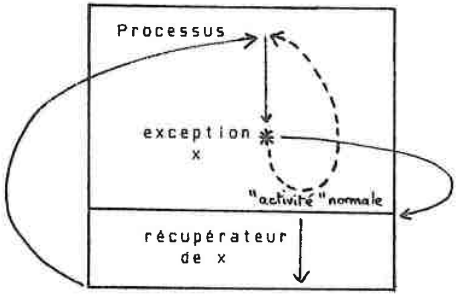


En présence d'un récupérateur de x, le SSP est terminé et le contrôle est rendu à l'appelant, après l'appel.

En absence d'un récupérateur de x, le SSP est terminé et l'exception est propagée à l'appelant au point d'appel.

- pour les processus, le schéma est du type "reprise";

Fig.3.7.



En présence d'un récupérateur de x, celui-ci est exécuté et le processus est repris après abandon de "l'activité" en cours. En absence de récupérateur de x, "l'activité" est abandonnée (nous avons rejeté la notion de terminaison normale ou anormale des processus).

i14) phrase de réinitialisation :
REINITIALISER liste-processus (des processus normaux du même module)

Ne peut apparaître que dans un processus d'interruption.

j) Les fonctions prédéfinies :

Notons quelques fonctions spécifiques telles que :

TEMPS qui délivre le temps courant du site concerné

ID-PROCESSUS qui délivre le nom du processus
ID-MODULE qui délivre le nom du module

k) La répartition des modules sur les sites

REPARTITION
SITES
liste-noms-sites
ASSIGNATIONS
liste-assignations

Chaque assignation s'écrit :
nom-module **SUR** nom-site

Dans les systèmes où la répartition peut être modifiée dynamiquement, il s'agit des assignations initiales.

Un module ne peut être assigné qu'à un seul site; un site peut accueillir plusieurs modules.

ANNEXE IV

Programme METAL définissant le "LSD réduit".
cf. chapitre 4 § 2.2.b.

Le "LSD réduit" correspond à la partie du LSD décrivant le réseau de modules qui est gérée au niveau graphique. Il a été utilisé pour tester l'interfaçage des deux éditeurs.

```
*definition de LSD.metal
*juin 85

definition of LSD is
  rules
    <LSD>::=<APPLIC>;
      <APPLIC>
    <APPLIC>::=application <EN_TETE><DESCR_APPLIC>fin_application;
      applic(<EN_TETE>,<DESCR_APPLIC>)
    <EN_TETE>::=<NOM>;
      <NOM>
    <EN_TETE>::=<NOM> presentation <TEXTE>;
      en_tete1(<NOM>,<TEXTE>)
    <DESCR_APPLIC>::=description <MOD_LIAIS><MESS_TYPES>;
      descr_applic2(<MOD_LIAIS>,<MESS_TYPES>)
    <DESCR_APPLIC>::=description <MOD_LIAIS><MESS_TYPES><REPART>;
      descr_applic1(<MOD_LIAIS>,<MESS_TYPES>,<REPART>)

  abstract syntax
    APPLIC::=applic;
    EN_TETE::=en_tete1 nom;
    DESCR_APPLIC::=descr_applic1 descr_applic2;
    applic->EN_TETE DESCR_APPLIC;
    en_tete1->NOM TEXTE;
    descr_applic1->MOD_LIAIS MESS_TYPES REPART;
    descr_applic2->MOD_LIAIS MESS_TYPES;

chapter NOM
  rules
    <NOM>::=%IDENTIFICATEUR;
      nom=atom(%IDENTIFICATEUR)
    <TEXTE>::=%CHAINE_CARACTERES;
      texte=atom(%CHAINE_CARACTERES)
    <NOM_POINTE>::=<NOM>#.<NOM>;
      nompointe(<NOM>,<NOM>)
    <EXTERIEUR>::=exterieur;
      exterieur=atom('exterieur')

  abstract syntax
    NOM::=nom;
    TEXTE::=texte;
    COMMENT::=comment;
    metavar->implemented as IDENTIFIER;
    comment->implemented as STRING;
    comment_s->COMMENT*...;
    exterieur->;
    nompointe->NOM NOM;
    nom->implemented as STRING;
    texte->implemented as STRING;
    meta->;

end chapter;

chapter MOD_LIAIS
  rules
    <MOD_LIAIS>::=modules <L_MOD> liaisons <L_LIAI>;
      mod_liais(<L_MOD>,<L_LIAI>)
    <L_MOD>::=;
      l_mod=list(())
    <L_MOD>::=<L_MOD>#;<MODULE>;
```



```
      l_mod=post(<L_MOD>,<MODULE>)
    <L_LIAI>::=;
      l_liai=list(())
    <L_LIAI>::=<L_LIAI>#;<LIAISON>;
      l_liai=post(<L_LIAI>,<LIAISON>)

  abstract syntax
    MOD_LIAIS::=mod_liais;
    L_MOD::=l_mod;
    L_LIAI::=l_liai;
    mod_liais -> L_MOD L_LIAI;
    l_mod -> MODULE*...;
    l_liai -> LIAISON*...;

end chapter;

chapter MESS_TYPES
  rules
    <MESS_TYPES>::=messages <L_MESS> types <L_TYPES>;
      mess_types(<L_MESS>,<L_TYPES>)
    <L_MESS>::=;
      l_mess=list(())
    <L_MESS>::=<L_MESS>#;<MESSAGE>;
      l_mess=post(<L_MESS>,<MESSAGE>)
    <L_TYPES>::=;
      l_types=list(())
    <L_TYPES>::=<L_TYPES> #; <TYPE>;
      l_types=post(<L_TYPES>,<TYPE>)
    <MESSAGE>::=message <EN_TETE> description <DESCR_MESS> fin_message;
      mess(<EN_TETE>,<DESCR_MESS>)

  abstract syntax
    MESS::=mess;
    MESS_TYPES::=mess_types;
    L_MESS::=l_mess;
    L_TYPES::=l_types;
    mess_types->L_MESS L_TYPES;
    l_mess->MESS*...;
    l_types->TYPE*...;
    mess->EN_TETE DESCR_MESS;

end chapter;

chapter LIAISON
  rules
    <LIAISON>::=<BIP>;
      <BIP>
    <LIAISON>::=<MULTIP_SEL>;
      <MULTIP_SEL>
    <LIAISON>::=<MULTIP_DIFF>;
      <MULTIP_DIFF>
    <BIP>::=<DES_PORT> vers <DES_PORT>;
      bip(<DES_PORT>.1,<DES_PORT>.2)
    <MULTIP_SEL>::=<DES_PORT> vers <POR1> ou <LPOR1>;
      multip_sel(<DES_PORT>,<POR1>,<LPOR1>)
    <MULTIP_DIFF>::=<DES_PORT> vers <DES_PORT> et <LPOR>;
      multip_diff(<DES_PORT>.1,<DES_PORT>.2,<LPOR>)

  abstract syntax
    LIAISON::=bip multip_sel multip_diff;
```

```
    bip->DES_PORT DES_PORT;
    multip_sel->DES_PORT POR1 LPOR1;
    multip_diff->DES_PORT DES_PORT LPOR;

end chapter;

chapter DES_PORT
  rules
    <DES_PORT>::=<NOM>;
      <NOM>
    <DES_PORT>::=<NOM_POINTE>;
      <NOM_POINTE>
    <DES_PORT>::=<EXTERIEUR>;
      <EXTERIEUR>
    <DES_POR1>::=<NOM> #: <DES_PORT>;
      despor1(<NOM>,<DES_PORT>)

  abstract syntax
    DES_PORT::=nom nompointe exterieur;
    despor1->NOM DES_PORT;

end chapter;

chapter PORT
  rules
    <LPOR1>::=<POR1>;
      lpor1=list((<POR1>))
    <LPOR1>::=<LPOR1> ou <POR1>;
      lpor1=post(<LPOR1>,<POR1>)
    <LPOR>::=<DES_PORT>;
      lpor=list((<DES_PORT>))
    <LPOR>::=<LPOR> et <DES_PORT>;
      lpor=post(<LPOR>,<DES_PORT>)
    <POR1>::=<NOM>;
      <NOM>
    <POR1>::=<NOM_POINTE>;
      <NOM_POINTE>
    <POR1>::=<EXTERIEUR>;
      <EXTERIEUR>
    <POR1>::=<DES_POR1>;
      <DES_POR1>

  abstract syntax
    LPOR1::=lpor1;
    LPOR::=lpor;
    POR1::=DES_PORT despor1;
    lpor1->POR1+...;
    lpor->DES_PORT+...;

end chapter;

chapter MODULE
  rules
    <MODULE>::=module <EN_TETE> <DESCR_MOD> fin_module;
      module(<EN_TETE>,<DESCR_MOD>)
    <DESCR_MOD>::=description <INTERFACE> <CORPS>;
      descr_mod(<INTERFACE>,<CORPS>)
    <INTERFACE>::=interface <L_PORTS_M>;
      <L_PORTS_M>
    <L_PORTS_M>::=;
```



```
      l_ports_m=list(())
    <L_PORTS_M>::=<L_PORTS_M> #; <PORTS_M>;
      l_ports_m=post(<L_PORTS_M>,<PORT_M>)
    <PORT_M>::=<E>;
      <E>
    <PORT_M>::=<S>;
      <S>
    <PORT_M>::=<ES>;
      <ES>
    <PORT_M>::=<SE>;
      <SE>
    <E>::=e_port <NOM> #: <NOM>;
      e(<NOM>.1,<NOM>.2)
    <S>::=s_port <NOM> #: <NOM>;
      s(<NOM>.1,<NOM>.2)
    <ES>::=es_port <NOM> #: <NOM> reponse <NOM>;
      es(<NOM>.1,<NOM>.2,<NOM>.3)
    <SE>::=se_port <NOM> #: <NOM> reponse <NOM>;
      se(<NOM>.1,<NOM>.2,<NOM>.3)

  abstract syntax
    MODULE::=module;
    DESCR_MOD::=descr_mod;
    INTERFACE::=l_ports_m;
    PORT_M::=e s es se;
    module->EN_TETE DESCR_MOD;
    descr_mod->INTERFACE CORPS;
    l_ports_m->PORT_M*...;
    e->NOM NOM;
    s->NOM NOM;
    es->NOM NOM NOM;
    se->NOM NOM NOM;

end chapter;

chapter ESSAI
  rules
    <CORPS>::=corps;
      corps=atom('corps')
    <DESCR_MESS>::=descr_mess;
      descr_mess=atom('descr_mess')
    <TYPE>::=type;
      type=atom('type')
    <REPART>::=repart;
      repart=atom('repart')

  abstract syntax
    corps->;
    descr_mess->;
    type->;
    repart->;

end chapter;

chapter POINTS_D_ENTREE
  rules
    <LSD>::=#[APPLIC]<APPLIC>;
      <APPLIC>
    <LSD>::=#[APPLIC]<METAVAR>;
      <METAVAR>
```

```
<METAVAR>:::=%METAVAR;
  metavar=atom(%METAVAR)
<LSD>:::=#[EN_TETE]<EN_TETE>;
  <EN_TETE>
<LSD>:::=#[EN_TETE]<METAVAR>;
  <METAVAR>
<LSD>:::=#[DESCR_APPLIC]<DESCR_APPLIC>;
  <DESCR_APPLIC>
<LSD>:::=#[DESCR_APPLIC] <METAVAR>;
  <METAVAR>
<LSD>:::=#[NOM] <NOM>;
  <NOM>
<LSD>:::=#[NOM] <METAVAR>;
  <METAVAR>
<LSD>:::=#[TEXTE] <TEXTE>;
  <TEXTE>
<LSD>:::=#[TEXTE] <METAVAR>;
  <METAVAR>
<LSD>:::=#[MOD_LIAIS] <MOD_LIAIS>;
  <MOD_LIAIS>
<LSD>:::=#[MOD_LIAIS] <METAVAR>;
  <METAVAR>
<LSD>:::=#[L_MOD] <L_MOD>;
  <L_MOD>
<LSD>:::=#[L_MOD] <METAVAR>;
  <METAVAR>
<LSD>:::=#[L_LIAI] <L_LIAI>;
  <L_LIAI>
<LSD>:::=#[L_LIAI] <METAVAR>;
  <METAVAR>
<LSD>:::=#[MESS_TYPES] <MESS_TYPES>;
  <MESS_TYPES>
<LSD>:::=#[MESS_TYPES] <METAVAR>;
  <METAVAR>
<LSD>:::=#[MESS] <MESSAGE>;
  <MESSAGE>
<LSD>:::=#[MESS] <METAVAR>;
  <METAVAR>
<LSD>:::=#[L_MESS] <L_MESS>;
  <L_MESS>
<LSD>:::=#[L_MESS] <METAVAR>;
  <METAVAR>
<LSD>:::=#[L_TYPES] <L_TYPES>;
  <L_TYPES>
<LSD>:::=#[L_TYPES] <METAVAR>;
  <METAVAR>
<LSD>:::=#[LIAISON] <LIAISON>;
  <LIAISON>
<LSD>:::=#[LIAISON] <METAVAR>;
  <METAVAR>
<LSD>:::=#[DES_PORT] <DES_PORT>;
  <DES_PORT>
<LSD>:::=#[DES_PORT] <METAVAR>;
  <METAVAR>
<LSD>:::=#[LPOR1] <LPOR1>;
  <LPOR1>
<LSD>:::=#[LPOR1] <METAVAR>;
  <METAVAR>
<LSD>:::=#[LPOR] <LPOR>;
  <LPOR>
```



```
<LSD>:::=#[LPOR] <METAVAR>;
  <METAVAR>
<LSD>:::=#[POR1] <POR1>;
  <POR1>
<LSD>:::=#[POR1] <METAVAR>;
  <METAVAR>
<LSD>:::=#[MODULE] <MODULE>;
  <MODULE>
<LSD>:::=#[MODULE] <METAVAR>;
  <METAVAR>
<LSD>:::=#[DESCR_MOD] <DESCR_MOD>;
  <DESCR_MOD>
<LSD>:::=#[DESCR_MOD] <METAVAR>;
  <METAVAR>
<LSD>:::=#[INTERFACE] <L_PORTS_M>;
  <L_PORTS_M>
<LSD>:::=#[INTERFACE] <METAVAR>;
  <METAVAR>
<LSD>:::=#[PORT_M] <E>;
  <E>
<LSD>:::=#[PORT_M] <METAVAR>;
  <METAVAR>
<LSD>:::=#[PORT_M] <S>;
  <S>
<LSD>:::=#[PORT_M] <METAVAR>;
  <METAVAR>
<LSD>:::=#[PORT_M] <ES>;
  <ES>
<LSD>:::=#[PORT_M] <METAVAR>;
  <METAVAR>
<LSD>:::=#[PORT_M] <SE>;
  <SE>
<LSD>:::=#[PORT_M] <METAVAR>;
  <METAVAR>

end chapter;

end definition
```

ANNEXE V

Exemple de validation de la spécification détaillée.
cf. chap.4 § 3.2.1.

Nous nous limitons au module M-POMPE. Nous choisissons, comme structuration interne la structure logique de la figure 2.20. La validation de la décomposition a déjà été établie au chapitre 3 § 2.1.3. où sont données les spécifications formelles de TEAU et POMPE. Nous donnons ci-dessous la spécification détaillée en LSD de M-POMPE, puis la confrontation des schémas de programme correspondants et des spécifications formelles.

b) Spécification détaillée :

```
APPLICATION KRAMER
  PRESENTATION
    Gestion d'une pompe souterraine et de son environnement
    atmosphérique.
  DESCRIPTION
    MODULES
      MODULE M-POMPE
         PRESENTATION
           Module qui gère la pompe.
         DESCRIPTION
           INTERFACE
              E-PORT E : EAU;
              E-PORT AM : ALM;
              E-PORT O : ORD;
              S-PORT CD : CDE;
              S-PORT R : REPP;
              SE-PORT RM : REQM REPONSE REPM;
           CORPS
              DECLARATIONS
              PROCESSUS
                PROCESSUS TEAU
                   PRESENTATION
                     Gère le niveau d'eau.
                   DESCRIPTION
                     INTERFACE
                        E-PORT E : EAU CONSOMMATION
                          DERNIER SANS REMISE;
                        S-PORT DE : D-EAU;
                     CORPS
                        DECLARATIONS
                          CONST SEUIL-HAUT : REEL;
                          CONST SEUIL-BAS : REEL;
                        TRAITEMENTS
                         REPETER
                          B-RECEVOIR M1 SUR E;
                          SI M1.VAL)=SEUIL-HAUT
                           ALORS NB-ENVOYER (HAUT) SUR DE
                           SINON SI M1.VAL<SEUIL-BAS
                            ALORS NB-ENVOYER (BAS) SUR DE
                                  FIN-SI
                          FIN-SI
                         FIN-REPETER
                FIN-PROCESSUS
```

```
PROCESSUS POMPE
   PRESENTATION
     Gère la pompe.
   DESCRIPTION
     INTERFACE
        E-PORT AM : ALM;
        E-PORT DE : D-EAU CONSOMMATION
                    DERNIER SANS REMISE;
        E-PORT O : ORD;
        S-PORT R : REPP;
        S-PORT CD : CDE;
        SE-PORT RM : REQM REPONSE REPM;
     CORPS
        DECLARATIONS
          CONST MAX : REEL;
          VAR ETAT-POMPE : ETAT := COUPEE;
        TRAITEMENTS
         REPETER
          CHOIX
           QUAND B-RECEVOIR () SUR AM AVEC
           ACQUITTEMENT =>
             SI ETAT-POMPE = EN-MARCHE
             ALORS B-ENVOYER (ARRET) SUR CD
             FIN-SI;
             ETAT-POMPE := ARRETEE-METHANE
           QUAND B-RECEVOIR M1 SUR DE =>
             CAS M1.NIVEAU
              QUAND HAUT => MARCHE-POSSIBLE
              QUAND BAS =>
               SI ETAT-POMPE=EN-MARCHE
               ALORS B-ENVOYER (ARRET) SUR CD;
                  ETAT-POMPE := ARRETEE-EAU
               FIN-SI
             FIN-CAS
           QUAND B-RECEVOIR M3 SUR O =>
             CAS M3.OR
              QUAND REQUETE => RIEN
              QUAND PRET =>
                SI ETAT-POMPE ≠ EN-MARCHE
                ALORS ETAT-POMPE:=PRETE
                FIN-SI
              QUAND COUPER =>
                SI ETAT-POMPE=EN-MARCHE
                ALORS B-ENVOYER (ARRET) SUR CD
                FIN-SI;
                ETAT-POMPE:=COUPEE
             FIN-CAS
           NB-ENVOYER (ETAT-POMPE) SUR R
          FIN-CHOIX
         FIN-REPETER;
         PROCEDURE MARCHE-POSSIBLE
          PRESENTATION
           Tests de l'état de la pompe et du
           méthane avant démarrage éventuel.
          DESCRIPTION
           DECLARATIONS
```



```
                    CORPS
                     SI ETAT-POMPE=PRETE OU ETAT-POMPE=
                      ARRETEE-EAU
                     ALORS
                       B-ENVOYER (METHANE) SUR RM
                       ATTENDRE M2 ALORS
                        SI M2=NORMAL
                        ALORS B-ENVOYER (MARCHE) SUR CD;
                         ETAT-POMPE := EN-MARCHE
                        SINON ETAT-POMPE=ARRETEE-METHANE
                        FIN-SI
                       QUAND DELAI MAX =>
(cf. KRAMER)            ETAT-POMPE:=ARRETEE-METHANE
                       FIN-B-ENVOYER
                     FIN-SI
          FIN-PROCESSUS
        LIAISONS
          EAU.DE VERS POMPE.DE;
        ATTACHEMENTS
          AM A AM;
          E A E;
          O A O;
          R A R;
          RM A RM;
          CD A CD;
        MESSAGES
          MESSAGE D-EAU
              DESCRIPTION
                NIVEAU:NIVO;
          FIN-MESSAGE;
        TYPES
          TYPE NIVO : ENUMERATION HAUT,BAS FIN-TYPE;
FIN-MODULE;
   %L'autre module M-ENVMT n'est pas décrit%
LIAISONS
   %Non décrites%
MESSAGES
   MESSAGE EAU
     DESCRIPTION
        VAL : REEL;
   FIN-MESSAGE;
   MESSAGE ALM
     DESCRIPTION
        VIDE;
   FIN-MESSAGE;
   MESSAGE ORD
     DESCRIPTION
        OR:QUOI;
   FIN-MESSAGE;
   MESSAGE CDE
     DESCRIPTION
        CD : NATURECD;
   FIN-MESSAGE;
   MESSAGE REPP
     DESCRIPTION
        REP : ETAT;
   FIN-MESSAGE;
```

```
        MESSAGE REQM
          DESCRIPTION
             VIDE;
         FIN-MESSAGE;
        MESSAGE REPM
          DESCRIPTION
             ETAG : NORMALITE;
         FIN-MESSAGE;
    TYPES
      TYPE QUOI : ENUMERATION PRET,COUPER,REQUETE FIN-TYPE;
      TYPE NORMALITE : ENUMERATION NORMAL,ANORMAL FIN-TYPE;
      TYPE NATURECD : ENUMERATION MARCHE,ARRET FIN-TYPE;
      TYPE ETAT : ENUMERATION COUPEE,PRETE,EN-MARCHE,
                  ARRETEE-EAU,ARRETEE-METHANE FIN-TYPE;
FIN-APPLICATION
```

c) Vérification de la cohérence des descriptions :

La vérification de cohérence de la spécification détaillée vis à vis des propriétés, s'établit trivialement en confrontant les propriétés et les schémas des processus réduits aux communications et à leurs conditionnements :

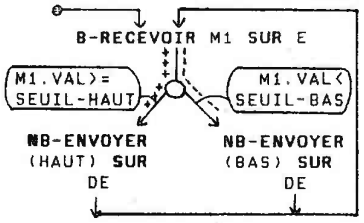
Propriétés

Schéma du processus tiré de la spécification détaillée

processus TEAU :

(T1) $\forall e(e \in EV(E) \wedge e.msg < seuil\text{-}bas \Leftrightarrow \exists d(d \in EV(DE) \wedge e \Rightarrow d \wedge d.msg = bas))$

(T2) $\forall e(e \in EV(E) \wedge e.msg = seuil\text{-}haut \Leftrightarrow \exists d(d \in EV(DE) \wedge e \Rightarrow d \wedge d.msg = haut))$

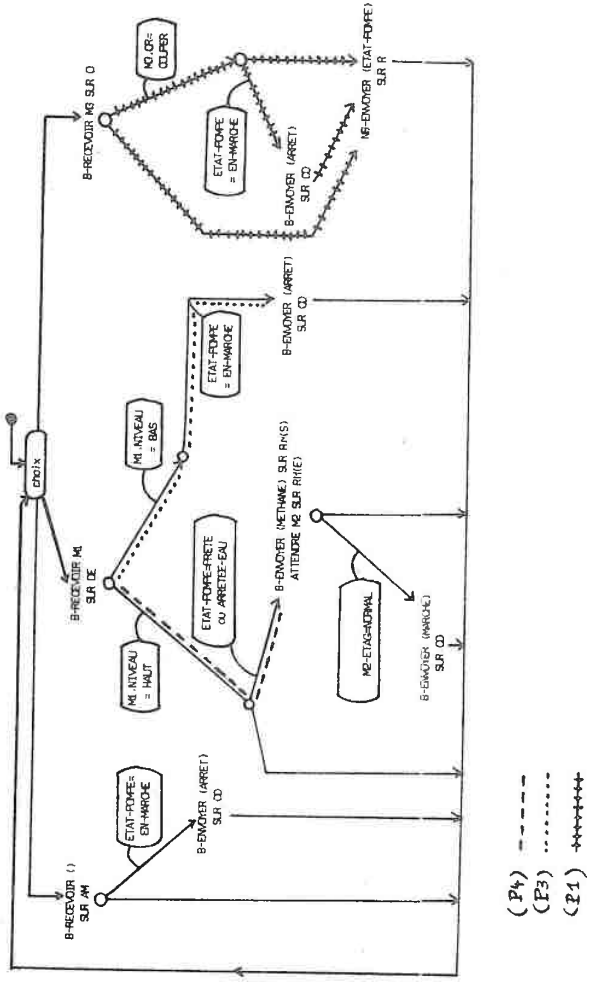


(T1) se retrouve dans la branche de droite (-----) ,(T2) dans la branche de gauche (++++++).

processus POMPE :

La vérification du respect des propriétés sur le schéma du processus, est illustrée par les 3 figures suivantes

Fig. AV-1.

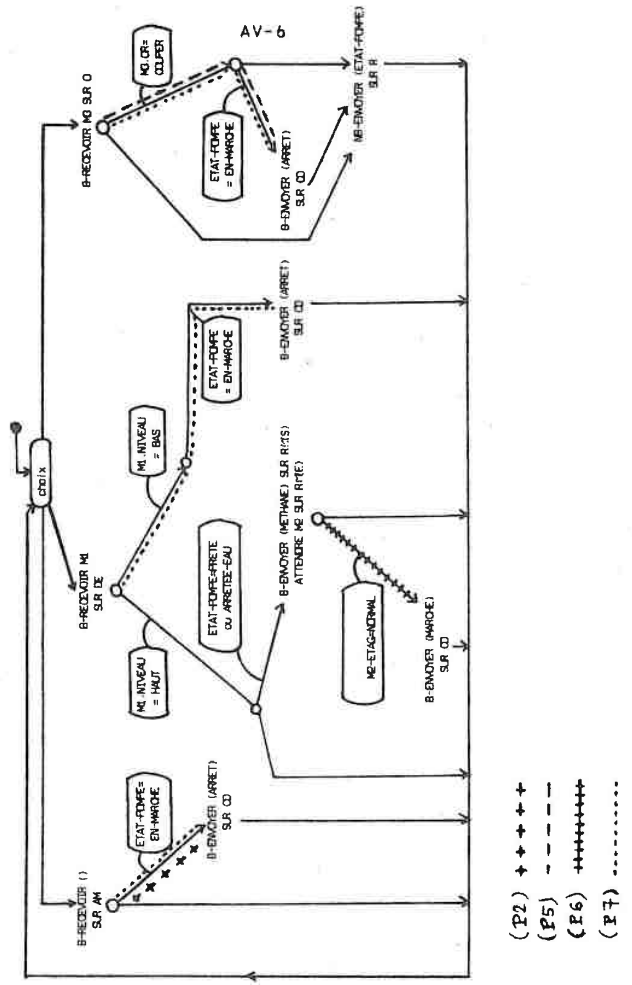


(P1) $\forall o(o \in EV(O) \Leftrightarrow \exists r(r \in EV(R) \wedge o \Rightarrow r))$

(P3) $\forall d(d \in EV(DE) \wedge d.msg = bas \wedge ETAT\text{-}POMPE = en\text{-}marche$ **quand** $d \Leftrightarrow \exists c(c \in EV(CD) \wedge d \Rightarrow c \wedge c.msg = arrêt))$

(P4) $\forall d(d \in EV(DE) \wedge d.msg = haut \wedge (ETAT\text{-}POMPE = prête$ **quand** $d \vee ETAT\text{-}POMPE = arrêtée\text{-}eau$ **quand** $d) \Leftrightarrow \exists r(r \in EV(RM(S)) \wedge d \Rightarrow r \wedge r.msg = méth))$

Fig.AV-2.



(P2) $\forall a(a\in EV(AM)\wedge$ETAT-POMPE=en-marche **quand** a $\#\rangle\ \exists c(c\in EV(CD)\wedge a=\rangle c\wedge c.\mathbf{msg}=arrêt))$

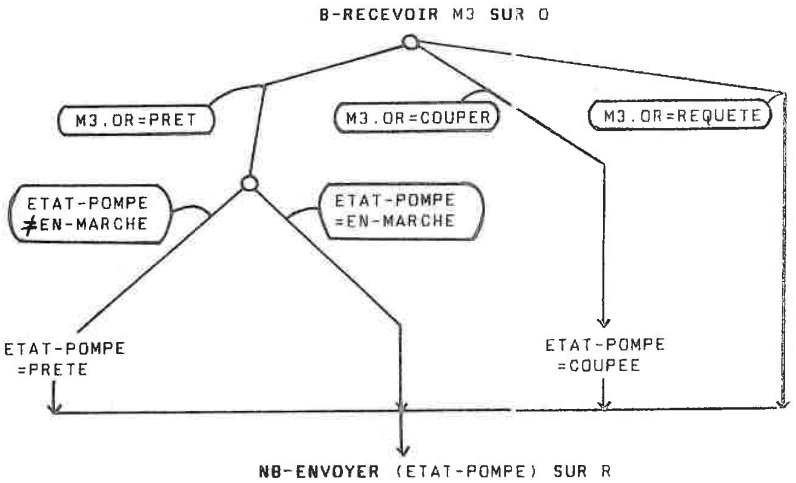
(P5) $\forall o(o\in EV(O)\wedge o.\mathbf{msg}=couper\wedge$ETAT-POMPE=en-marche **quand** o $\#\rangle\ \exists c(c\in EV(CD)\wedge o=\rangle c\wedge c.\mathbf{msg}=arrêt))$

(P6) $\forall r(r\in EV(RM(E))\wedge r.\mathbf{msg}=$normal $\langle\#\rangle\ \exists c(c\in EV(CD)\wedge r=\rangle c\wedge c.\mathbf{msg}=$marche))

(P7) $\forall c(c\in EV(CD)\wedge c.\mathbf{msg}=arrêt\ \#\rangle\ \exists d(d\in EV(DE)\wedge d.\mathbf{msg}=bas\wedge$ETAT-POMPE = en-marche **quand** $d\wedge d=\rangle c)\vee\exists o(o\in EV(O)\wedge o.\mathbf{msg}=couper\wedge$ETAT-POMPE=en-marche **quand** $o\wedge o=\rangle c)\vee\exists a(a\in EV(AM)\wedge$ ETAT-POMPE= en-marche **quand** $a\wedge a=\rangle c))$

Fig.AV-2.

Les propriétés (P8) à (P12) se vérifient grâce au schéma de processus détaillé suivant :



(P8) $\forall o(o\in EV(O)\wedge(o.\mathbf{msg}=couper\vee(o.\mathbf{msg}=requête\wedge$ETAT-POMPE=coupé **quand** o)) $\langle\#\rangle\ \exists r(r\in EV(R)\wedge o=\rangle r\wedge r.\mathbf{msg}=coupé))$

(P9) $\forall o(o\in EV(O)\wedge(o.\mathbf{msg}=requête\vee o.\mathbf{msg}=prêt)\wedge$ETAT-POMPE=en-marche **quand** o $\langle\#\rangle\ \exists r(r\in EV(R)\wedge o=\rangle r\wedge r.\mathbf{msg}=$en-marche))

(P10) $\forall o(o\in EV(O)\wedge o.\mathbf{msg}=requête\wedge$ETAT-POMPE=arrêtée-eau **quand** o $\langle\#\rangle\ \exists r(r\in EV(R)\wedge o=\rangle r\wedge r.\mathbf{msg}=arrêtée-eau))$

(P11) $\forall o(o\in EV(O)\wedge o.\mathbf{msg}=requête\wedge$ETAT-POMPE=arrêtée-méthane **quand** o $\langle\#\rangle\ \exists r(r\in EV(R)\wedge o=\rangle r\wedge r.\mathbf{msg}=arrêtée-méthane))$

(P12) $\forall o(o\in EV(O)\wedge((o.\mathbf{msg}=prête\wedge$ETAT-POMPE$\neq$en-marche **quand** o) $(o.\mathbf{msg}=requête\wedge$ETAT-POMPE=prête **quand** o)) $\langle\#\rangle\ \exists r(r\in EV(R)\ o=\rangle r\wedge r.\mathbf{msg}=prête))$

On peut donc affirmer que la spécification détaillée est cohérente avec les propriétés formelles initiales tirées du cahier des charges.

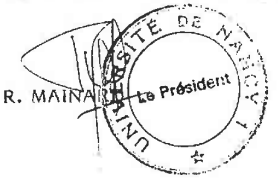
NOM DE L'ETUDIANT : *LONCHAMP Jacques*

NATURE DE LA THESE : *Doctorat d'Etat ès sciences*

VU, APPROUVE ET PERMIS D'IMPRIMER 394

NANCY, le 24 MARS 1987

LE PRESIDENT DE L'UNIVERSITE DE NANCY I



R. MAINARD

RESUME :

La première partie établit une liste de caractéristiques significatives pour la classe de problèmes visée. En fonction de leur adéquation à ces caractéristiques est retenu un ensemble de concepts et d'idées dans les domaines des architectures logicielles, des démarches de conception et des techniques de spécification. La seconde partie détaille la proposition avec d'une part les outils (langages de spécification) et d'autre part leur mode d'emploi (méthodes de construction, validations, environnement de conception). La faisabilité de la méthode est illustrée sur quelques exemples non triviaux. La troisième partie vise à évaluer la contribution.

La proposition englobe un noyau immédiatement et concrètement utilisable (architecture logicielle, outil de spécification graphique des structures, outil de spécification programmatoire au niveau détaillé, éditeur /contrôleur interactif de ces descriptions, traduction en ADA des concepts) autour duquel se greffe un ensemble de propositions d'outils intellectuels et logiciels de plus haut niveau (outil de spécification formelle des comportements, validations associées, méthode de partitionnement systématique, etc).

MOTS CLES :

CONCEPTION ; APPLICATIONS REPARTIES ; COMMANDE DE PROCEDES INDUSTRIELS ; STRUCTURATION ; SPECIFICATION ; VALIDATION ; METHODE ; OUTILS