Institut National Polytechnique
de Lorraine

E.N.S.E.M

THESE

présentée à

L'I.N.P.L

pour l'obtention du titre de

DOCTEUR INGENIEUR

SPECIALITE INFORMATIQUE

par

Philippe NONN

Ingenieur ENSEM

SUJET :

LE SYSTEME D'EXPLOITATION DANS LE PROJET SYGARE

Soutenue publiquement le 16 novenbre 1978 devant la commission d'examen

Membres du jury

Président	:	M. C PAIR
Examinateurs	:	M. C CAMOZZI
		M. J.C DERNIANE
		M. M GRIFFITHS
		M. R HUSSON
		M. J.P THOMESSE

Institut National Polytechnique
de Lorraine

E.N.S.E.M

THESE

présentée à

L'I.N.P.L

pour l'obtention du titre de

DOCTEUR INGENIEUR

SPECIALITE INFORMATIQUE

par

Philippe NONN

Ingenieur E N S E M

SUJET :

LE SYSTEME D'EXPLOITATION DANS LE PROJET SYGARE

Soutenue publiquement le 16 novenbre 1978 devant la commission d'examen

Membres du jury

Président	:	M. C PAIR
Examinateurs	:	M. C CAMOZZI
		M. J.C DERNIANE
		M. M GRIFFITHS
		M. R HUSSON
		M. J.P THOMESSE

AVANT-PROPOS

Nous remercions vivement Monsieur le Professeur PAIR, Président de l'Institut National Polytechnique de Lorraine, qui a bien voulu patronner notre travail, et qui nous fait l'honneur de présider ce jury.

Nous exprimons notre gratitude à Monsieur le Professeur HUSSON pour son accueil au sein de l'équipe automatique du Laboratoire d'Electronique, d'Electrotechnique et d'Automatique de l'E.N.S.E.M., et pour l'intérêt qu'il nous porte en participant à ce jury.

Nous remercions Monsieur le Professeur DERNIAME pour les nombreuses suggestions et critiques constructives qu'il nous a formulées, tout au long de notre travail, et lors de la rédaction de ce mémoire.

Que Monsieur le Professeur GRIFFITHS, Directeur de l'Institut Universitaire de Calcul Automatique de Nancy, et Monsieur CAMOZZI, Responsable du Département Logiciel de la S.E.M.S., trouvent ici l'expression de nos remerciements pour l'honneur qu'ils nous font en participant à ce jury.

Nous remercions Monsieur THOMESSE, Responsable du projet SYGARE, pour ses conseils de tous les instants et pour l'ambiance amicale qu'il n'a cessé d'entretenir,

Nous exprimons notre sincère reconnaissance à Madame DALBOURG, et au personnel du D.P.I.C. de l'Institut National Polytechnique de Lorraine, qui ont assuré avec soin et diligence la réalisation matérielle de ce mémoire.

Nous remercions enfin tous nos collègues du Laboratoire E.E.A. de l'E.N.S.E.M. et du Centre de Recherches en Informatique de Nancy pour leurs conseils et leurs encouragements amicaux.

TABLE DES MATIERES

CHAPITRE VII : GESTION DES ENTREES-SORTIES

CHAPITRE VIII : L'ARCHITECTURE DU SYSTEME D'EXPLOITATION

INTRODUCTION

Les dispositifs mis en oeuvre pour la régulation et la commande des systèmes industriels ont longtemps été conçus isolément pour résoudre un type de problème bien particulier correspondant à une plage de fonctionnement d'une partie de l'installation nécessairement très restreinte.

Ils font en outre appel aux différentes ressources technologiques disponibles sur le marché :

- mécanique,
- électrotechnique,
- électronique.

L'association d'éléments très divers ne permet pas un entretien ou une évolution aisée de l'application : la modification des caractéristiques de l'un deux entraîne, en général, celle d'autres éléments, de telle sorte que l'on hésite souvent à changer de point de fonctionnement pour s'adapter à des conditions nouvelles.

L'introduction des systèmes informatisés a permis le regroupement sur un même calculateur (dénomé de ce fait : "calculateur industriel") de la plupart des traitements relatifs à une application et l'introduction de nouvelles fonctions, jusqu'alors irréalisables, en raison de la multiplicité et de l'hétérogénéité des dispositifs de régulation mis en jeu :

- transmission d'informations,
- synchronisation,
- édition d'un journal, de statistiques ...

Il devenait alors possible de concevoir une exploitation de l'application industrielle adaptée aux différents paramètres principaux.

Le récent essor de l'électronique numérique, avec dans une première étape, l'introduction des automates programmables, puis très rapidement, l'avènement des microprocesseurs, a permis de répartir les points de décision et de concevoir des capteurs "intelligents" réalisant un pré-traitement des informations recueillies.

De cette façon, on a eu la possibilité d'accroître la rapidité des traitements effectués, de diminuer les temps de réponse, d'améliorer le taux d'utilisation des matériels coûteux et d'introduire des redondances aux endroits critiques afin d'améliorer la fiabilité du dispositif.

Cependant, chacun des composants du système étant choisi pour son adaptation à traiter le problème qu'on lui assigne, il possède des qualités qui le distinguent des autres éléments :

- structure interne de la machine,
- variété de l'ensemble des instructions disponibles,
- rapidité du traitement,
- organisation et représentation interne des informations à traiter,
- périphériques connectés,
- taille de l'espace mémoire disponible,
- langages de programmation supportés,
- système d'exploitation (lorsqu'il existe).

La réunion de tels éléments pour former un réseau hétérogène alors qu'ils n'ont pas été conçus initialement pour travailler ensemble pose un grand nombre de problèmes tant au niveau conception qu'au niveau entretien de l'application. L'insuffisance des moyens mis à la disposition du concepteur d'une application répartie est à l'origine de cette étude.

Nous avons en effet voulu démontrer l'intérêt et la faisabilité d'un SYstème de Gestion d'Applications temps-réel Réparties. Nous nous sommes plus particulièrement attachés à cette étude du point de vue système d'exploitation et réseau tandis qu'Alain Cochet-Muchy s'intéressait à la production de programmes exécutables pour un tel système.

Notre conception d'une application répartie est basée sur sa décomposition logique en éléments plus petits : les modules. Le chapitre I présente les caractéristiques de la décomposition d'une application répartie.

Celle-ci s'appuie sur le concept de donnée transmissible, développé au chapitre II. C'est grâce à l'utilisation des données transmissibles que les modules quoique conçus indépendamment les uns des autres peuvent communiquer et se synchroniser.

La gestion des données transmissibles, assurée par le système d'exploitation fait l'objet du chapitre III.

L'utilisation d'une hiérarchie de structures de contrôle permet de décrire l'organisation des modules dans une tâche fonctionnelle (chapitre IV) dont l'exécution met en oeuvre deux composants essentiels du système d'exploitation : l'interpréteur des structures de contrôle de classe II et le gérant des données transmissibles (chapitre V).

Le chapitre VI présente la notion d'événement et les structures de contrôle de classe III qui permettent d'associer une tâche fonctionnelle et l'environnement extérieur dans lequel elle s'exécute.

Au chapitre VII, nous avons montré comment l'introduction, dans le système d'exploitation, d'un système de gestion des entrées/sorties rend les modules indépendants de leur contexte d'exécution et autorise ainsi le déplacement des modules sur d'autres calculateurs permettant ainsi la reconfiguration de l'application.

Enfin, le chapitre VIII présente l'architecture du système d'exploitation que nous avons entrepris de développer sur une maquette constituée d'un Solar 16-40 (SEMS) et d'un microprocesseur Motorola 6800.

CHAPITRE I

CARACTERISTIQUES DE LA DECOMPOSITION D'UNE APPLICATION

CHAPITRE I

CARACTERISTIQUES DE LA DECOMPOSITION D'UNE APPLICATION

I.1. PRESENTATION

Le module est l'élément de base de la description d'une application répartie en SYGARE. Il renferme l'algorithme codant une action élémentaire, qui si l'on excepte la gestion des entrées/sorties traitée de façon indépendante, peut être implantée entièrement sur l'un des calculateur-hôtes du réseau SYGARE et s'y exécuter dans sa totalité. Le module constitue une unité de compilation et d'exécution.

Il peut être assimilé à un automate séquentiel communiquant avec l'extérieur grâce à des "entrées" et à des "sorties" analogues aux entrées et aux sorties d'un dispositif électronique considéré comme une boîte noire (fig.1).

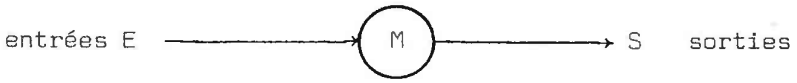


Fig.1. Représentation d'un module sous la forme d'une boîte noire.

Le module possède, comme tout automate séquentiel, des variables d'état ϕ qui permettent d'évaluer les sorties S en fonction des entrées E et de l'algorithme F défini par le texte source et figé par la compilation.

$$(S,\phi) = F(E, \phi)$$

Les variables d'état ne sont pas implantées sous ce nom mais sous forme de variables apparaissant simultanément en entrée et en sortie du module.
Le module peut également effectuer des opérations d'entrée/sortie sur des périphériques classiques ou sur des capteurs et des actionneurs. Ceci lui permet de prendre en compte et d'influer sur le milieu extérieur au dispositif de commande.

La conception et l'écriture du module sont indépendantes de son lieu d'exécution d'une part, de son contexte d'exécution d'autre part.
L'indépendance vis à vis du lieu d'exécution est obtenue par l'emploi d'un unique langage d'Ecriture de Modules indépendant de la machine-cible et grâce à la possibilité d'entreprendre depuis n'importe quel site du réseau des opérations d'entrées/sorties sur n'importe quel périphérique.

Cela se traduit par la possibilité de déplacer une module sur le réseau sans avoir à en modifier le texte-source. L'exécution du module sur le nouveau site doit

se traduire par l'élaboration de nouvelles valeurs de S et ϕ liées aux entrées E et aux valeurs mesurées sur les capteurs par la fonction F.
L'indépendance d'un module vis à vis de son contexte d'exécution est réaliséeen séparant la description de ses liens avec les autres modules de la décomposition, de l'algorithme du module.
On peut ainsi modifier facilement le contexte d'exécution d'un module, supprimer ou insérer des appels à un module sans avoir à en modifier le texte-source.

I.2. LE LANGAGE D'ECRITURE DE MODULES

Le langage d'écriture de modules possède les propriétés suivantes (ACM 78) :

- la programmation de l'algorithme est indépendante de la machine-cible,
- les structures de contrôle employées appartiennent à une même classe homogène,
- les objets manipulés par le module ont des caractéristiques bien définies, leur utilisation est vérifiée à la compilation.

I.2.1. Indépendance du langage de programmation

Le langage d'écriture de modules est un langage unique quelque soit la mchine-cible sur laquelle le module s'exécutera. Il permet la programmation de n'importe quel algorithme séquentiel sans avoir à se préoccuper :

- de la richesse du code d'ordres de la machine-cible,
- de la taille du mot de mémoire,
- des modes d'adressages disponibles,
- de l'initialisation des registres, bases ...

Les modules écrits dans ce langage peuvent être déplacés sans que l'on ait à modifier leur code source pour tenir compte de possibilités offertes par la nouvelle machine-cible. Une simple recompilation permet d'obtenir le code-objet destiné au nouveau calculateur. Tout algorithme s'exécutant de façon correcte sur une machine doit posséder le même effet sur une autre : le module conserve la même fonction de transfert F(& I.1).

I.2.2. Les structures de contrôle de Classe I

Les structures de contrôle utilisées pour la description de l'algorithme du module permettent la description de toute opération séquentielle. Elles forment la classe I des structures de contrôle et autorisent :

- l'enchaînement en séquence,
- les constructions conditionnelles : si ... alors ... fsi
  si ... alors ... sinon ... fsi
- les constructions itératives : tant que ... faire... ftq.

Elles suffisent à programmer tout algorithme (LEDGARD 75).

I.2.3. Les objets manipulés par le module

. Le langage d'écriture de modules permet l'utilisation d'objets simple ou structurés en tableaux et caractérisés par :

- leur type : logique, entier court (octet), entier long (16 bits), réel.
- un attribut défini par l'utilisateur pour préciser la nature de l'objet par exemple : vitesse, courant ...

. Ces objets peuvent être :

- des constantes locales au module et non modifiées par son exécution,
- des variables dites internes utilisées dans l'algorithme du module pour mémoriser des résultats intermédiaires.

Elles ne permettent pas au module de garder des informations entre deux exécutions consécutives et ne sont donc pas des variables d'état, au sens où l'entend l'automaticien.

- les données transmissibles, externes au module peuvent figurer en entrée et/ou en sortie de celui-ci.

Les données transmissibles figurant en entrée du module sont considérées comme des paramètres d'exécution de son algorithme. Les données transmissibles figurant en sortie renferment, en fin d'exécution du module, les valeurs calculées par l'algorithme.

Il peut exister plusieurs versions d'une même donnée transmissible. Pendant l'exécution d'un module, celui-ci n'utilise qu'une seule version d'une donnée. Celle-ci peut être considérée comme locale au module dans le mesure où :

- elle porte un nom local au module.
- la version d'une donnée figurant en entrée d'un module ne peut être modifiée de l'extérieur pendant l'exécution.
- la version d'une donnée figurant en sortie ne peut être consultée par des modules distincts de M tant que celui-ci est susceptible de la modifier (i.e : tant qu'il est actif).

. Les opérations réalisées par le module sur ces objets doivent être compatibles avec les propriétés qui ont été déclarées. Une vérification statique réalisée lors de la compilation permet de s'assurer que :

- les déclarations de type et d'attribut concordent avec les opérations réalisées,
- l'exécution du module ne risque pas de modifier la valeur d'une constante,
- le module ne risque pas d'altérer la valeur des données transmissibles qu'il utilise en entrée seulement,
- le module élabore bien les sorties (la variable figure à gauche d'un signe

d'affectation ou dans un ordre de lecture).

Ces vérifications permettent d'assurer que :

- la valeur des constantes reste la même entre deux exécutions
- plusieurs modules parallèles peuvent partager sans inconvénient le même exemplaire d'une donnée transmissible si celle-ci ne figure qu'en entrée de ces modules.

I.2.4. Structure d'un module compilé

La compilation d'un module doit conserver toutes les informations caractérisant l'emploi des données transmissibles par le module. Ces informations sont regroupées dans une table d'utilisation des données transmissibles. Celui-ci renferme (fig.2) :

- le nom local sous lequel la donnée est désignée,
- le type et l'attribut de la donnée,
- la taille de la donnée : . variable simple : 1,
  . tableau monodimensionnel : longueur,
  . tableau bidimensionnel : longueur totale.
- le mode d'utilisation de la donnée :
  . en entrée seulement,
  . en sortie seulement,
  . en entrée/sortie.
- adresse du relais vers la donnée pour l'exécution.

NOM LOCAL
Type et attribut
Taille de la donnée
Mode d'utilisation : E,S,E/S
Adresse du relais

Fig.2 Descripteur de donnée transmissible.

La structure du module, après l'étape de compilation, est le suivante quelque soit la machine-cible sur laquelle il doit s'exécuter (fig.3).

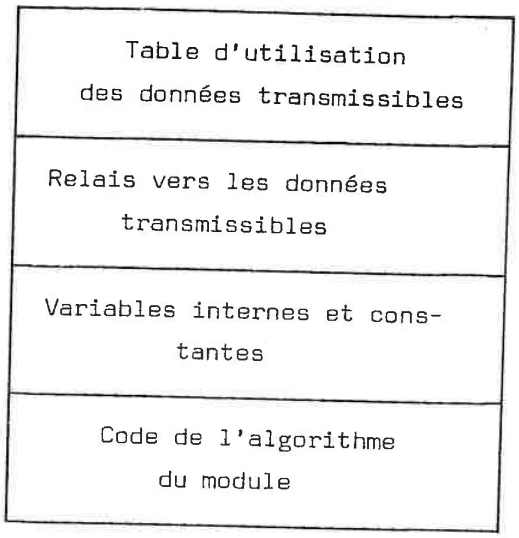


Fig.3 Structure du module compilé.

I.3. L'UTILISATION DES PERIPHERIQUES DU RESEAU

L'indépendance du lieu d'exécution d'un module par rapport aux organes d'entrée/sortie qu'il utilise est obtenue par l'intermédiaire d'un système de gestion des entrées/sorties (&VII.2) permettant l'accès à n'importe quel périphérique du réseau depuis n'importe quel site. Grâce à ce système, les périphériques employés ne constituent plus un obstacle à la répartition des modules sur le réseau, et on peut en modifier la répartition sans difficulté particulière. Le système de gestion des entrées/sorties réalise l'interface entre le module et le périphérique employé comme la plupart des calculateurs (IOCS 76). La différence consiste en l'utilisation par ce système des procédures de transmission du réseau pour atteindre le périphérique souhaité si celui-ci est éloigné. L'influence de la répartition sur le temps de réponse d'un périphérique peut être parfois ressentie comme une gène et on aura intérêt, chaque fois que ce délai est critique, ou que le temps de transfert risque d'être important, à implanter les modules sur les mêmes sites que les périphériques qu'ils utilisent. Cette solution apporte aussi une plus grande sécurité de fonctionnement.

I.4. L'ENVIRONNEMENT DU MODULE

L'environnement d'un module est constitué de l'ensemble de ses liens avec les autres modules composant l'application. Ces liens peuvent être de deux types :

- données échangées par les modules,
- assemblage de modules pour remplir une certaine fonction.

I.4.1. La transmission de données entre modules

Les modules qui ont besoin d'échanger des informations doivent utiliser dans ce but les données transmissibles (& I.2.3).
Grâce aux déclarations de connexion (& IV.2) on peut établir une correspondance entre la sortie S d'un module M1 et l'entrée E d'un module M2 comme le fait apparaître la figure 4.

Fig.4 Connexion entre la sortie S de M1 et l'entrée E de M2.

Au moment de son lancement, le module M2 reçoit la valeur des données transmissibles figurant en entrée. Celles-ci doivent avoir été précédemment produites par un ou plusieurs autres modules où elle figurent en sortie.
Le fait de ne pas préciser dans le texte-source d'un module, l'origine de la donnée figurant en entrée ou la destination de la donnée figurant en sortie permet de modifier les connexions entre modules sans avoir à :

- altérer le texte source des modules,
- recompiler les modules pour tenir compte des nouveaux liens.

I.4.2. L'assemblage des modules en tâches fonctionnelles

Nous avons vu au paragraphe précédent comment les modules pouvaient échanger des informations.
Les connexions entre les modules ne supposent rien quant à leur dépendance.
Les modules peuvent en effet être regroupés au sein d'une même tâche fonctionnelle dont le but est d'accomplir une certaine action. L'assemblage des modules pour former une tâche fonctionnelle est réalisé au moyen de structures de contrôle de classe II. Ces structures forment un sur-ensemble des structures de classe I permettant de coder l'algorithme d'un module (& I.2.2.). On y a adjoint la description du parallélisme entre modules. L'assemblage des modules en tâche fonctionnelle doit rendre compte des possibilités de parallélisme entre modules et des relations de précédences imposées par :

- l'utilisation des données transmissibles : l'exécution de M1 doit précéder celle de M2.
- l'utilisation des périphériques : la fermeture du contact établissant le courant dans un circuit électrique doit intervenir avant la mesure de ce courant ...

La constitution d'une tâche fonctionnelle est réalisée indépendamment du codage

de différentes actions élémentaires qu'elle suppose. On peut ainsi la modifier "en-ligne" sans qu'il soit nécessaire d'arrêter le dispositif de commande :

- insertion de nouveaux modules,
- suppression de certaines séquences devenues inutiles,
- remplacement d'un module par un autre ...

I.4.2.1. Le Parallélisme

Le parallélisme intervient partout où aucune relation de précédence n'existe entre les modules codant les actions à réaliser (HABERMANN 75).

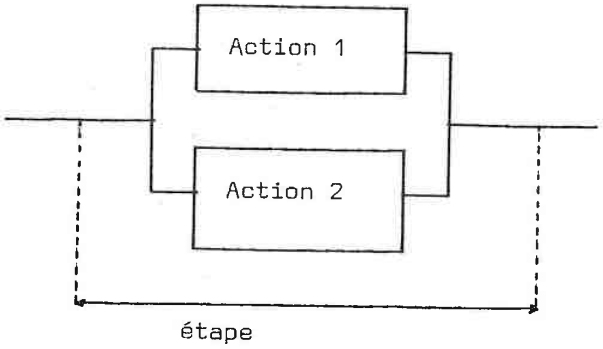


Fig.5 Etape constituée de deux modules parallèles.

Le parallélisme intervient de façon fondamentale dès qu'on se trouve placé sur un réseau de calculateurs, nous allons en présenter ici les caractéristiques :

a) Répartition du traitement

Le parallélisme permet une répartition des opérations à effectuer en fonction des possibilités de chaque calculateur-hôte du réseau. Il est intéressant dès que l'on peut découper une action en actions élémentaires dont le déroulement s'effectue de façon indépendante. Chacune des actions ainsi définies concourt à l'élaboration d'un résultat global que l'on n'obtient définitivement qu'à la fin de l'étape.

La répartition est particulièrement intéressante afin d'améliorer la rapidité d'exécution d'un traitement ou, dans le cas d'un réseau hétérogène comme SYGARE, pour faire traiter par chaque calculateur, le travail pour lequel il a été plus spécialement conçu ou programmé (arithmétique réelle, tracé de courbes, FFT,...).

b) Indépendance des branches

Chacune des branches concurrentes évalue un élément du résultat final ou effectue certaines actions sur le procédé par le jeu des organes d'entrées/sorties :

calcul d'éléments du résultat, mesures sur une partie du dispositif, commandes de certains appareils. L'évolution d'une branche ne doit pas influer celle des autres.

c) Conditions initiales

En ce qui concerne l'utilisation des données transmissibles, les conditions initiales doivent être identiques pour toutes les branches parallèles, ce sont celles qui existent au début de l'étape.
Ce point fondamental entraîne les conséquences suivantes :

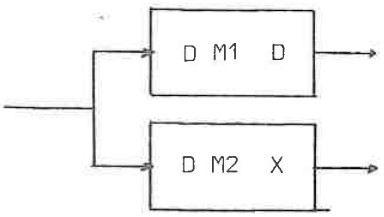
- on ne doit pas lancer une branche si les autres ne peuvent l'être par manque d'une donnée. Cette règle permet de maintenir la cohérence de l'ensemble des données sur lesquelles travaille un ensemble de modules.

- Si une branche n'a pu être lancée par manque de disponibilité des calculateurs sur lequel elle est implantée, elle devra cependant travailler sur les mêmes versions des données transmissibles que celles utilisées par les autres branches. Il faut donc conserver l'état des données au moment du lancement des branches même si une évolution de celle-ci se produit pendant le déroulement de l'étape.

exemple : soient deux modules M1 et M2 définis par :

```
Module : M1                    Module : M2
        entier : D, ...                entiers: D,X
        entrée : D ...                 entrée : D
        sortie : D, ...                sortie : X
           :                              :
        fin                            fin
```

M1 et M2 doivent s'exécuter en parallèle dans une même tâche fonctionnelle suivant le schéma suivant :



Si M1 et M2 sont implantés sur le même calculateur et que l'on ne peut garantir que l'ordre d'exécution sera M2 puis M1, il faut sauvegarder la valeur de la donnée D avant d'exécuter M1 pour transmettre à M2 les mêmes conditions initiales.

d) Regroupement de branches

Dans la mesure du possible, les branches parallèles s'exécutent de façon simultanée sur les différents calculateurs du réseau. Cependant, la charge des

calculateurs, la durée du traitement à réaliser conduisent en général à une période durant laquelle toutes les branches ne sont pas achevées. Il faut donc attendre la terminaison de toutes les branches avant de valider les données produites par chacune d'elles et passer à l'exécution de la séquence suivante.
Les expressions de chemin permettent donc de définir un point de synchronisation dans une tâche fonctionnelle de la même façon qu'une instruction de type SYNCHRO qui intervient au niveau du module dans le langage GAELIC (LE CALVEZ 77).

e) Parallélisation d'actions indépendantes

Les différentes propriétés que nous avons présenté ne sont en aucun cas liées à la répartition des actions sur le réseau. Plusieurs modules parallèles peuvent en particulier être implantés sur le même calculateur sans aucune difficulté. Leur ordre d'activation est laissé à l'initiative de système d'exploitation qui adopte celui qui implique le moins de manipulations sur les données transmissibles.

On peut également trouver un intérêt à déclarer sous forme de deux modules parallèles, deux actions élémentaires non liées par une relation de précédence.

I.4.2.2. Structure conditionnelle

La structure conditionnelle possède la même signification dans une tâche fonctionnelle ou à l'intérieur d'un module, elle permet d'effectuer sous condition des traitements répartis dans plusieurs modules.

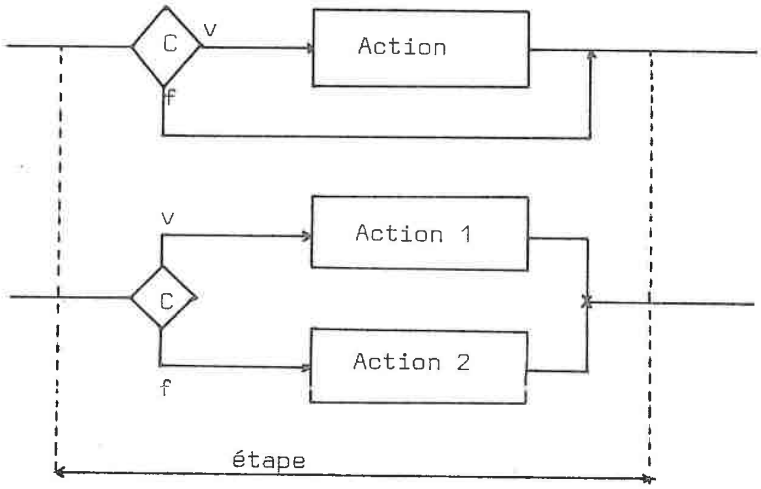


Fig.6 Structures conditionnelles.

La condition évaluée par le système d'exploitation porte sur des valeurs de données transmissibles. Dans un premier temps, nous nous sommes volontairement limités à l'examen de la valeur d'une donnée logique simple. Cependant, une extension pourrait être faite rapidement en développant la partie de

moniteur qui traite l'évaluation des conditions.
La structure ainsi définie a les mêmes propriétés qu'un module unique qui peut la remplacer si les traitements des actions 1 et 2 sont réalisées sur le même calculateur (fig.7).

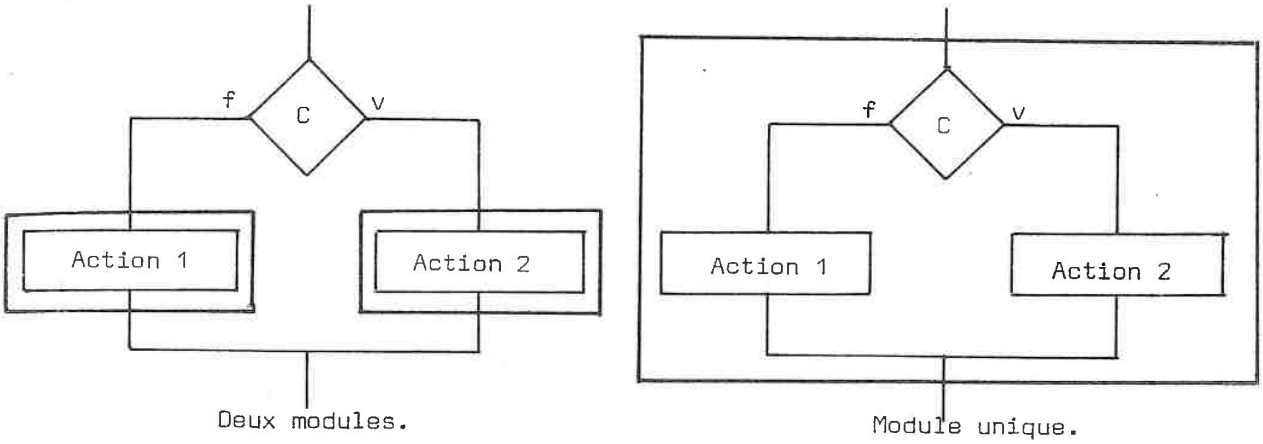


Fig.7 Structures conditionnelles de classe II et de classe I.

I.4.2.3. Structures itératives

Les structures itératives sont analogues à l'intérieur d'une tâche fonctionnelle ou d'un module. Elles permettent de repéter l'exécution d'une action en fonction d'une condition.

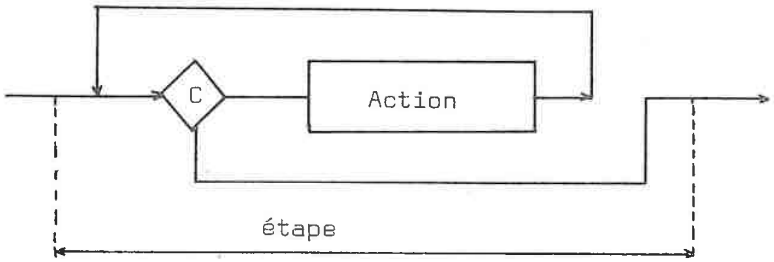


Fig.8 Structure itérative.

La condition utilisée pour l'itération est examinée de la même façon que celle des structures conditionnelles et porte donc, pour l'instant, sur l'état d'une donnée transmissible logique.
Cette structure est indispensable car elle permet de répéter une action décomposée en plusieurs modules indépendants répartis sur des sites divers répartis sur le réseau. Cette étape se présente, vue de l'extérieur, comme l'exécution d'un module unique qui utiliserait les mêmes données transmissibles.

I.5. CONCLUSION

Les propriétés que nous avons attribuées à un module correspondent à un besoin

de structuration de l'application industrielle déjà présenté dans (THOMESSE 77). Elles permettent la séparation de l'algorithme d'une action élémentaire des caractéristiques de son lieu d'implantation et de son contexte d'exécution.

On peut ainsi modifier facilement la répartition des modules sur le réseau sans avoir à modifier le code-source du module déplacé ni le code-objet des autres modules qui sont en liaison avec lui.

On peut également faire évoluer une tâche fonctionnelle en insérant ou en supprimant certains modules et en modifiant les connexions existantes.

C H A P I T R E II

L'UTILISATION DES DONNEES TRANSMISSIBLES

CHAPITRE II

L'UTILISATION DES DONNEES TRANSMISSIBLES

Les déclarations de connexion permettent la description du flux des données transmissibles entre les différents modules de l'application, indépendamment du fait qu'ils appartiennent ou non à une même tâche fonctionnelle.

Nous allons cependant distinguer ces deux cas dans la présentation des propriétés des données transmissibles qui sont liées à leur mode d'utilisation.

II.1. MODULES APPARTENANT A UNE MEME TACHE FONCTIONNELLE

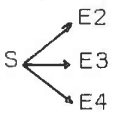
Nous ne nous intéresserons, dans une première phase qu'aux propriétés des données transmissibles utilisées en entrée seulement ou en sortie seulement.

II.1.1. Les données utilisées en entrée ou (exclusif) en sortie

Ce cas est relativement simple ; la sortie S d'un module M1 est connectée aux entrées de n autres modules successeurs de M1 au sens strict (fig.3).

```
Module : M1
        entier : S
        sortie : S
        algorithme
        fin

Module : M2
        entier : E2
        réel : S2
        entrée : E2
        sortie : S2
        algorithme
        fin

Module : M3
        entier : E3
        réel : S3
        entrée : E3
        sortie : S3
        algorithme
        fin

Module : M4
        entier : E4
        réel : E5, E6
        entrées : E4, E5, E6
        algorithme
        fin
```

Les connexions réalisées sont :



S 2 ⟶ E 5

S 3 ⟶ E 6

Les connexions réalisées correspondent au schéma représenté sur la figure 10.

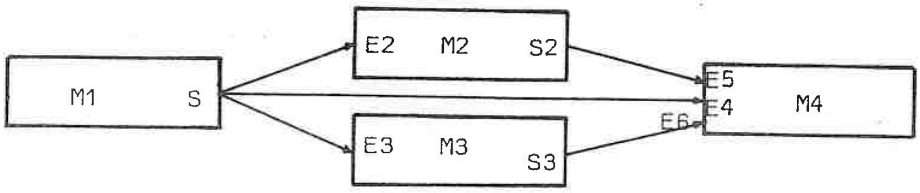


Fig.10 Connexions de données dans une même tâche (entrée ou sortie).

. La donnée S produite par le module M1 est consommée par les modules M2, M3, M4. Comme elle n'est connectée à aucun autre module : elle doit disparaître après terminaison de M4. Lors de l'exécution suivante de la tâche fonctionnelle, le module M1 produira une nouvelle valeur de la sortie S qui sera à nouveau communiquée à M2, M3 et M4.

. L'entrée d'un module ne peut être connectée aux sorties de plusieurs modules que si l'exécution de l'un de ces modules exclut celle de tout autre.
Comme dans la structure si ... alors ... sinon ... fsi.
Dans le cas contraire, il y aurait indétermination sur la valeur de la donnée à transmettre ; cette anomalie peut être détectée lors du traitement des déclarations de connexion, compte tenu de la connaissance des déclarations internes au module.

II.1.2. Données utilisées en entrée/sortie par un module

L'utilisation d'une même donnée en entrée/sortie par un module permet de réaliser une fonction de mémoire.
L'utilisation de ces données est un moyen simple de représenter les variables d'état au sens habituel de l'automatique.
Elle permet de conserver une information entre deux exécutions consécutives d'un même module. Ces exécutions peuvent avoir pour origine l'itération de l'exécution du module dans la tâche fonctionnelle ou plusieurs exécutions de la tâche elle-même.
L'initialisation de la donnée transmissible peut être réalisée de diverses manières :

- par une directive INIT dans les déclarations de connexion,
- par l'exécution d'un module ou d'une tâche fonctionnelle d'initialisation.

La directive INIT permet de définir la valeur initiale d'une donnée transmissible lors de sa première apparition : première exécution de la tâche fonctionnelle si l'on demande que la donnée soit conservée entre deux exécutions - première exécution du module dans la tâche fonctionnelle si l'on désire que la donnée soit réinitialisée à chaque appel de la tâche.

L'utilisation de la directive INIT se traduit par l'exécution d'un pseudo-module d'initialisation. Elle peut être remplacée par un module ou une tâche de l'utilisateur remplissant le même rôle et dont l'exécution doit obligatoirement précéder celle du module utilisant la donnée (fig.11).

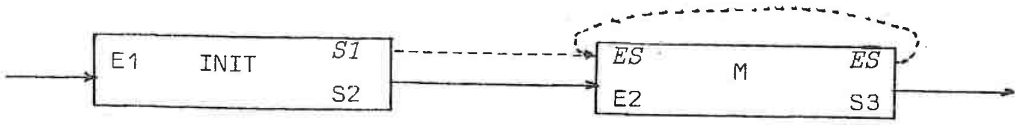


Fig.11 Utilisation de données transmissibles en Entrée/Sortie.

II.1.3. Données utilisées en entrée/sortie par plusieurs modules

Cette extension découle directement du cas présenté aux paragraphes précédents lorsque le module qui utilise la donnée en entrée/sortie doit être découpé en p modules dont l'enchaînement est purement séquentiel.
On retrouve tout naturellement les deux cas présentés au paragraphe précédent :

- utilisation de la donnée en mémoire entre 2 appels d'un même groupe de modules (fig.12).

- utilisation de la donnée en mémoire entre 2 appels consécutifs à la tâche fonctionnelle (fig.13).

L'initialisation des données transmissibles est réalisée de la même façon qu'au paragraphe précédent.

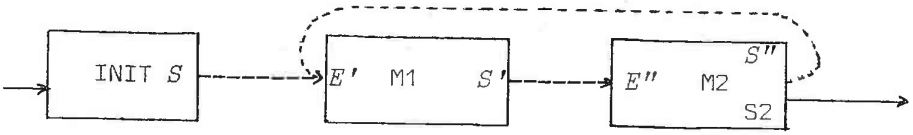


Fig.12 Donnée utilisée en mémoire entre rappels à un ensemble de 2 modules.

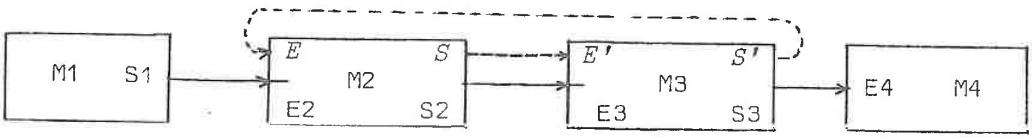


Fig.13 Donnée utilisée en mémoire entre 2 appels à une tâche fonctionnelle.

II.1.4. Remarques

L'utilisation des données transmissibles dans une même tâche fonctionnelle n'entraîne aucune action de synchronisation entre les modules, les données étant toujours produites dans une étape précédent celle où elles sont consultées. Lorsque la tâche fonctionnelle se termine, les données qui ne sont utilisées que

par cette tâche n'ont pas à être conservées, excepté celles qui permettant la mémorisation d'informations entre deux appels à cette tâche fonctionnelle.

II.2. MODULES APPARTENANT A DES TACHES FONCTIONNELLES CONCURRENTES

Dès que l'on s'intéresse à l'utilisation de données transmissibles par des tâches fonctionnelles concurrentes, il devient nécessaire d'introduire la notion de temps et une notion qui lui est étroitement liée, celle de la durée de vie d'un exemplaire de la donnée.
Cette notion n'a pas à intervenir au niveau du module, celui-ci ne faisant que traduire un algorithme. Elle est cependant indispensable lorsqu'on s'intéresse aux connexions entre les modules. Les deux événements essentiels de la vie d'une donnée sont :

- la création d'un nouvel exemplaire,
- la disparition d'un ancien exemplaire.

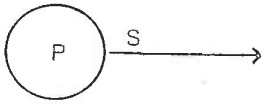
La création d'un nouvel exemplaire de la donnée est réalisé par un module utilisant la donnée en sortie au moins, à la fin de son exécution.
Elle peut être accompagnée suivant les cas :

1°) de sa mise en file d'attente de consommation,
2°) de la péremption de l'exemplaire précédent et de son remplacement par le nouveau.

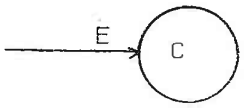
Le premier cas correspond à ce que Holt nomme ressources consommables (HOLT 71). La consommation d'un exemplaire de la donnée est réalisée par un module l'utilisant en entrée. Elle s'accompagne de la disparition de l'exemplaire utilisé. La durée de vie d'un exemplaire d'une donnée consommable est limitée par sa production et par sa consommation.
Le second cas correspond aux ressources rémanentes (HOLT 71). La consultation d'une donnée rémanente est réalisée par un module utilisant la donnée en entrée. Lors de son lancement le module consultant reçoit toujours le dernier exemplaire produit. La durée de vie d'un exemplaire d'une donnée rémanente est donc limitée par sa production d'une part, la production d'un nouvel exemplaire d'autre part.

II.2.1. Utilisation des données consommables

La production d'un exemplaire d'une donnée consommable est réalisée par un module ayant déclaré cette donnée en sortie seulement. Elle n'est effective que lorsque l'exécution du module se termine.



La consommation d'un exemplaire de la donnée est réalisée par un module ayant déclaré cette donnée en entrée seulement. L'exemplaire alloué à un consommateur n'est plus accessible pour un autre module.



L'existence d'une file d'attente permet de conserver lesexemplaires produits non consommés. Le dimensionnement de cette file est laissé au gré de l'utilisateur et permet de limiter le nombre des exemplaires en attente à une valeur raisonnable et d'éviter la saturation du système.
La synchronisation est réalisée entre les modules producteurs et les consommateurs et conduit au :

- réveil d'un producteur lorsque la file n'est plus pleine
- réveil d'un consommateur lorsque la file n'est plus vide.

Il serait également possible d'envisager d'autres méthodes de gestion du tampon. La gestion sous forme de pile permetant par exemple l'implantation de modules récursifs.

II.2.1.1. Un producteur, un consommateur

Ce cas, extrêmement trivial rend compte du classique schéma comportant un producteur, un consommateur (figure 14).
La synchronisation du consommateur se fait sur la production d'une donnée. Celle du producteur sur la consommation d'une donnée. Lorsqu'un des modules est bloqué en attente, toute la tâche à laquelle il appartient reste bloquée, et ceci est général.

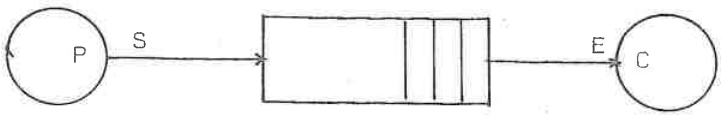


Fig. 14 Modèle un producteur, un consommateur

II.2.1.2. p producteurs, c consommateurs

Lorsque p producteurs sont susceptibles d'intervenir pour produire différents exemplaires de la donnée, ils doivent tous être considérés comme identiques vis à vis de celle-ci. Les règles de synchronisation présentées au paragraphe précédent restent valables.
Cela signifie qu'un producteur ne peut être privilégié par rapport à un autre et que rien ne permet de distinguer le producteur d'un exemplaire de la donnée pris au hasard (fig.15).

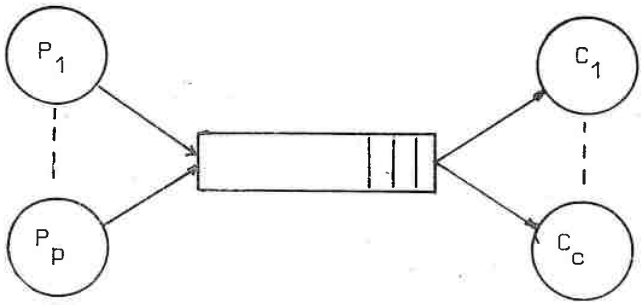


Fig.15 p producteurs, c consommateurs.

De même lorsque c consommateurs sont susceptibles de consommer différents exemplaires d'une même donnée, ces propriétés sont applicables.
Lorsque l'utilisateur prend en charge lui-même la gestion des données transmissibles, il lui est possible de décrire des tâches fonctionnelles comportant en parallèle plusieurs producteurs ou plusieurs consommateurs identiques. Lorsqu'une séquence comportant deux producteurs en parallèle se termine, deux exemplaires distincts de la donnée sont alors produits et déposés dans le tampon. Rien ne permet alors d'identifier leur producteur. Il en est de même pour la consommation de données.

II.2.1.3. Remarques

L'utilisation des données consommables entraîne une synchronisation entre les différentes tâches qui ont accès aux données afin d'interdire la production lorsque le tampon est saturé ou la consommation lorsqu'il n'y a pas de données présentes.
Les tâches fonctionnelles dans lesquelles une module est bloqué en attente de libération d'un accès (producteur ou consommateur) à la donnée ne peuvent pas évoluer du tout tant que le module est bloqué.

II.2.1.4. Exemple

L'utilisation des données consommables peut être une solution simple au problème du traitement d'informations en différé. C'est par exemple le cas de certaines phases du système à piloter qui nécessitent la prise de mesures à une fréquence élevée. Les valeurs relevées peuvent être traitées ultérieurement lorsque le pilotage de l'application nécessite moins d'interventions du dispositif de commande.

La programmation du module effectuant la prise de mesure ou de celui qui réalise le traitement ne fait pas intervenir la gestion de la file d'attente, ce qui permet une lisibilité plus grande. On peut d'autre part modifier la taille du tampon de façon indépendante.

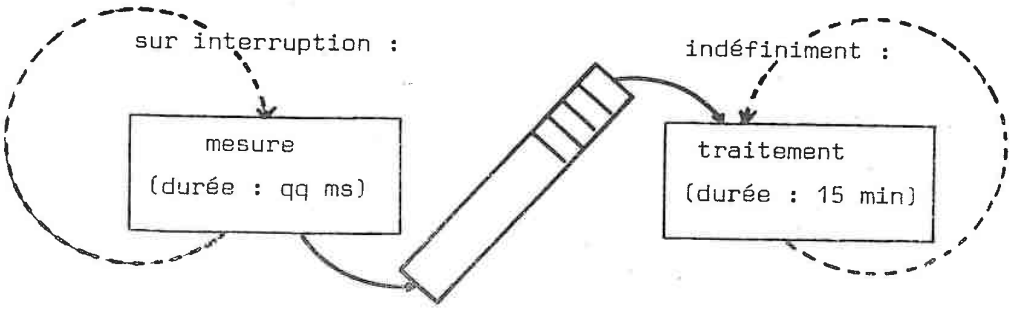


Fig.16 Exemple de production-consommation.

La figure 17 montre l'évolution des différents exemplaires d'une donnée consommable.

A l'étape 1 un module M1 est lancé qui réalise la production d'un exemplaire de la donnée. Lorsqu'il se termine (2), cet exemplaire est chaîné dans la file d'attente de consommation. A l'étape 3, un module M2 qui consomme la donnée est lancé, il faut supprimer l'exemplaire qui lui est alloué de la file d'attente de consommation. Aux étapes 4 et 5, de nouveaux modules qui produisent des exemplaires de la donnée sont lancés. A l'étape 6, l'un des modules produisant un exemplaire se termine, ce dernier est chaîné en file d'attente.

Lorsque le module M2 se termine (7) l'exemplaire qu'il a consommé disparaît définitivement ...

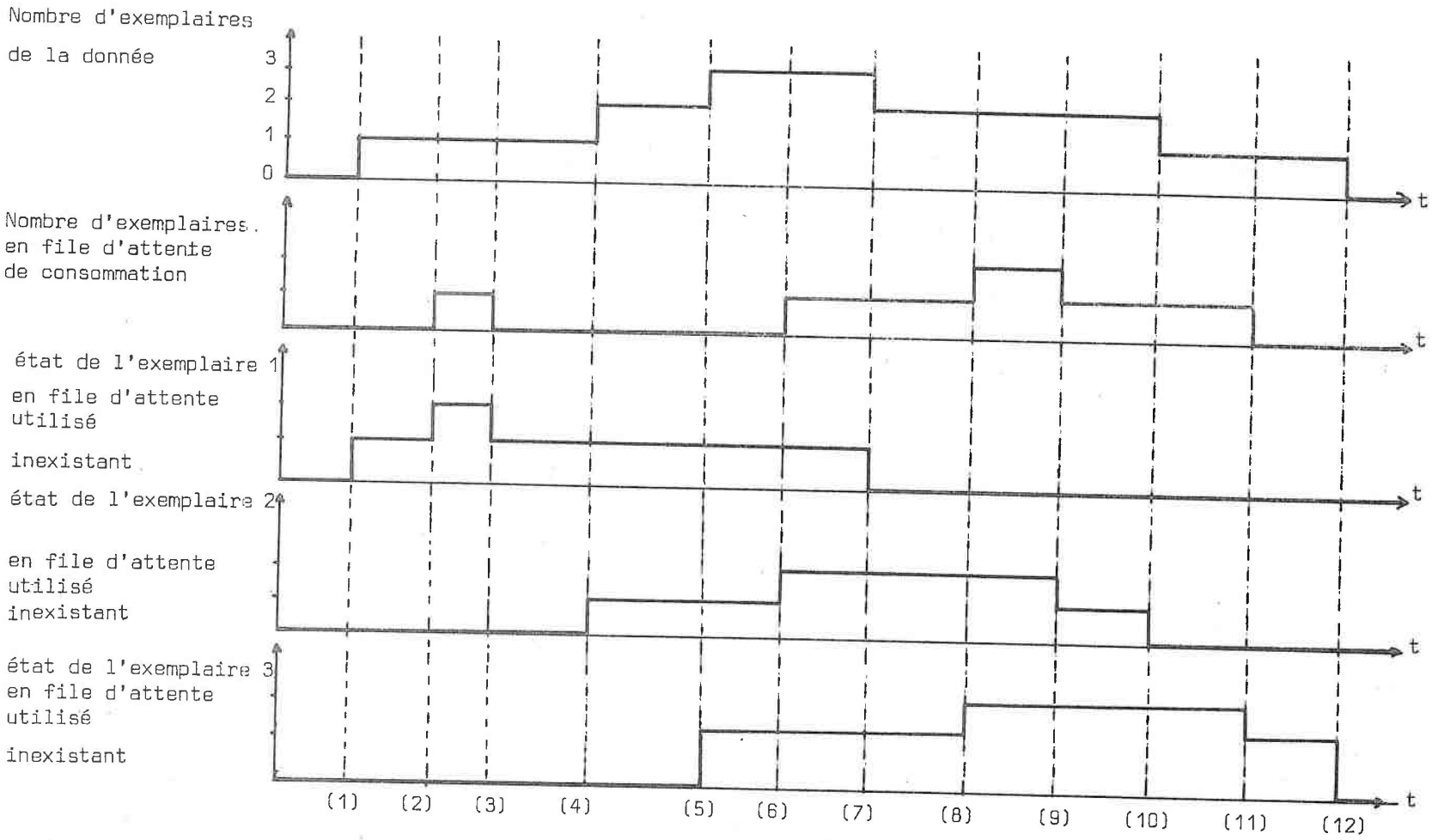
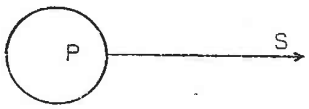


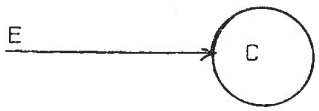
Fig.17 Durée de vie des différents exemplaires d'une donnée consommable.

II.2.2. Utilisation des données rémanentes : production consommation

La production d'un exemplaire d'une donnée rémanente est réalisée par un module ayant déclaré cette donnée en sortie seulement. Elle n'est effective que lorsque l'exécution du module est terminée. L'exemplaire produit remplace le précédent. Il constitue alors la version courante de la donnée.



La consultation de la version courante d'une donnée rémanente est effectuée par un module ayant déclaré cette donnée en entrée seulement. Elle n'entraîne pas de disparition de l'exemplaire consulté.



Les tâches fonctionnelles auxquelles appartiennent les modules producteurs ou consultants peuvent s'exécuter avec des périodes quelconques. Pendant l'exécution d'un producteur la version courante reste disponible. Nous allons maintenant préciser l'utilisation des données rémanentes en examinant divers cas.

II.2.2.1. Un producteur. Un consultant

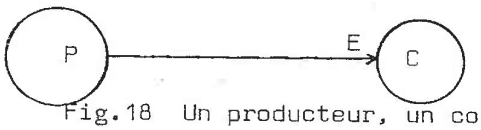


Fig.18 Un producteur, un consultant.

La figure 18 traduit le cas le plus simple d'utilisations de données rémanentes. Soient T_p et T_e les périodes du producteur et du consommateur. Nous distinguerons trois cas :

1°) $T_p < T_e$: Le producteur produit plus de valeurs que n'en peut consulter le module C. Les exemplaires non consultés au moment d'une nouvelle production sont définitivement perdus.

2°) $T_p = T_e$: Le producteur fonctionne au même rythme que le consultant. Chaque exemplaire est consulté une fois et une seule.

3°) $T_p > T_e$: Le producteur produit moins de valeurs que n'en consulte le module C. Celui-ci reçoit plusieurs fois le même exemplaire.

Cette étude nous a conduit à envisager trois cas, dans la réalité les périodes d'exécution des tâches productrice et consultante sont loin d'être fixes l'une par rapport à l'autre. Cela conduit tantôt à perdre certains exemplaires, tantôt à consulter plusieurs fois le même. L'important est que le producteur ou le consultant ne soit jamais bloqué (sauf dans le cas ou l'on veut consulter une donnée non

encore produite).

II.2.2.2. Un producteur, C consultants

Le cas où un unique producteur est relié à C consultants se déduit immédiatement du précédent (fig.19). Les propriétés du producteur et des consommateurs sont analogues à celles présentées au paragraphe II.2.2.1. Plusieurs consultants peuvent se partager le même exemplaire de la donnée.

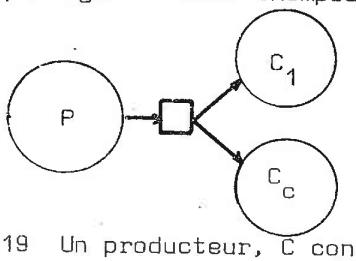


Fig.19 Un producteur, C consultants.

II.2.2.3. P producteurs, C consultants

Enfin, le cas où p producteurs produisent concurremment les différents exemplaires de la donnée est une généralisation de ce type d'utilisation (fig.20). A tout moment, la version courante de la donnée est le dernier exemplaire produit, quelque soit son origine, elle peut être consultée par n'importe quel module.

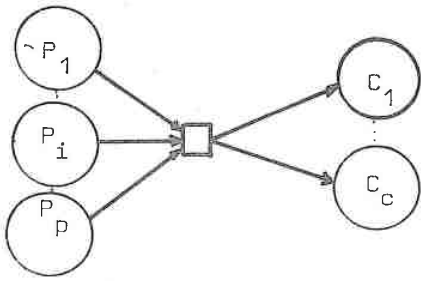


Fig.20 P producteurs, C consultants.

II.2.2.4. Remarques

La seule synchronisation existant avec ce mode d'utilisation des données rémanentes consiste en un blocage des modules voulant consulter une donnéee non produite.
Dans tous les autres cas, le lancement des modules ne peut être différé en raison d'un tel accès à une donnée rémanente.

II.2.2.5. Exemple

L'utilisation de données rémanentes permet l'évaluation d'une donnée par un module et sa prise en compte par un ensemble d'autre.
Un module MESURE effectue périodiquement une mesure de la vitesse d'un moteur et sa transformation en unités du système international.

Trois modules utilisent la donnée V produite par MESURE en consultation.
MOYEN accumule les différentes valeurs de V pour en calculer la moyenne, il est activé avec la même période que MESURE ;
AFFICHE réalise l'affichage de la valeur V échantillonnée à une fréquence relativement basse (durée d'affichage) ;
REGUL calcule les commandes à envoyer au dispositif de commande de ce moteur en fonction des points de consignes affichés. Il est activé périodiquement en fonction des constantes de temps du moteur.
Les périodes de chaque module sont exclusivement fonction des opérations réalisées (fig.21).

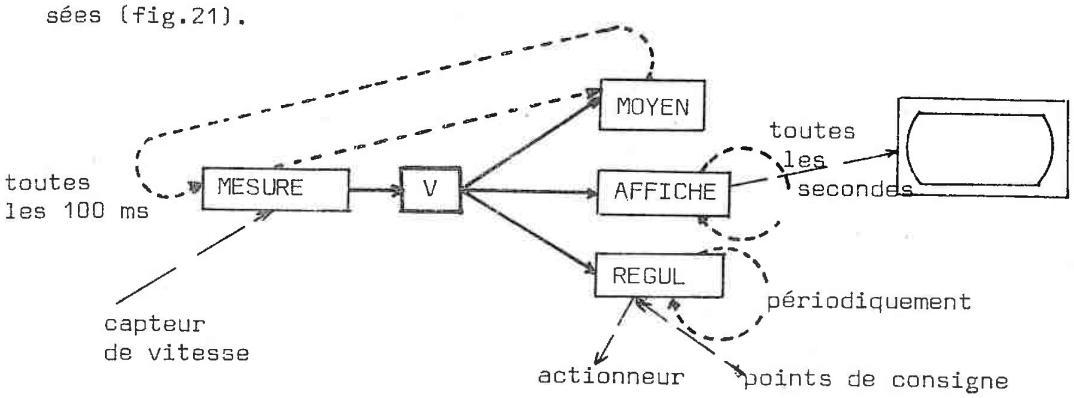


Fig.21 Exemple d'utilisation de donnée rémanente.

La figure 22 montre l'évolution des différents exemplaires d'une donnée rémanente en fonction de l'utilisation qu'en font différents modules.
L'étape 1 correspond au lancement d'un module producteur M1. Un exemplaire 1 de la donnée est créé mais on ne peut le consulter avant la terminaison de M1 (2). Cet exemplaire devient alors la version de la donnée qui est allouée à tout module qui veut la consulter.
Aux étapes 3 et 4 sont lancés deux modules qui consultent l'exemplaire 1. Lorsqu'à l'étape 7 un module produisant un nouvel exemplaire 2 de la donnée se termine, l'exemplaire 1 est périmé et ne pourra plus être alloué. Il est cependant conservé car un module l'utilise.
Ce n'est pas le cas de l'exemplaire 2 qui disparaît à l'étape 12 lorsqu'un module produisant l'exemplaire 3 se termine.

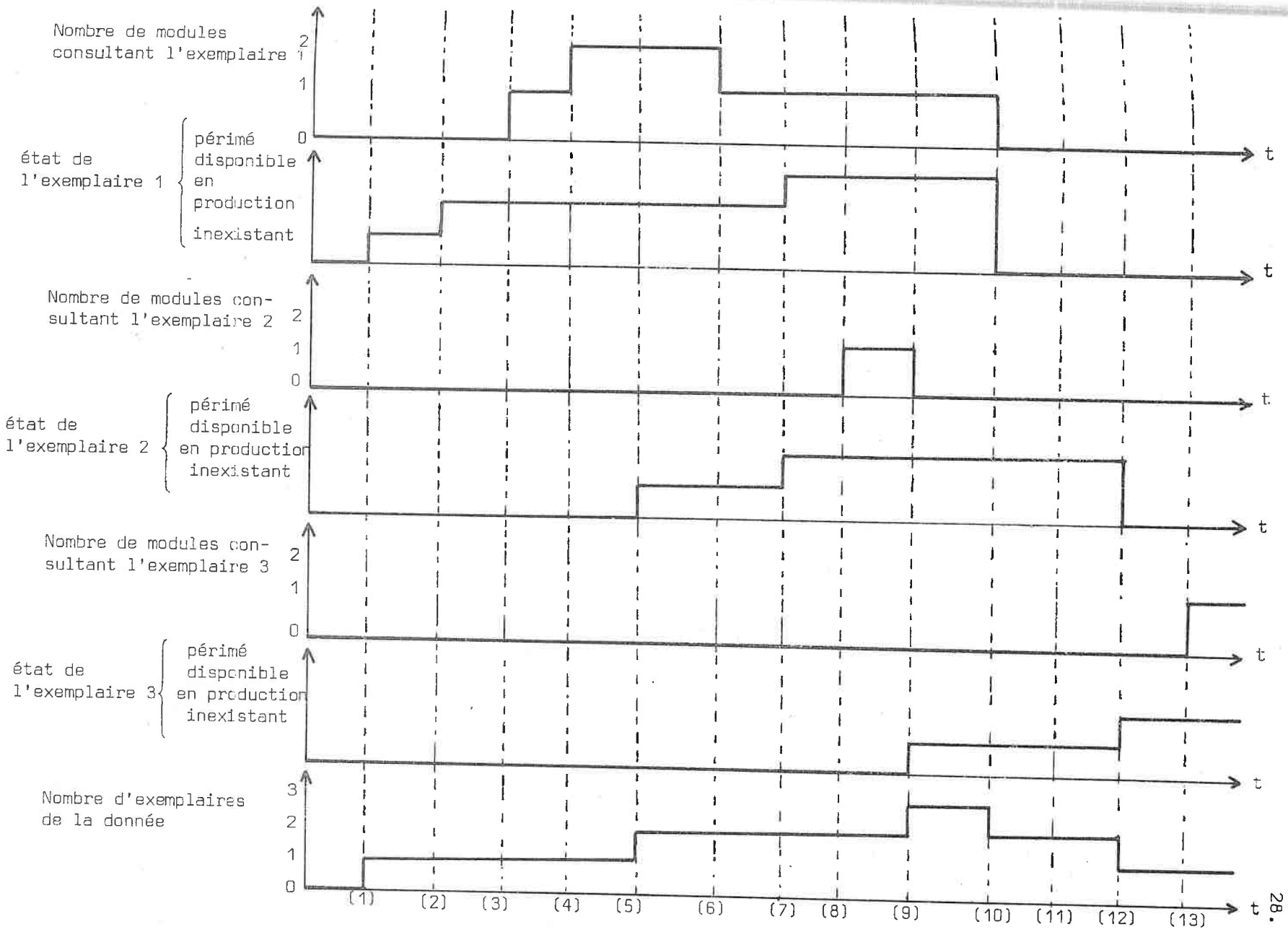
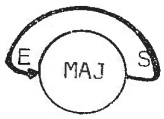


Fig.22 Durée de vie d'un exemplaire d'une donnée rémanente.

II.2.3. Utilisation des données rémanentes : Consultation et Mise à Jour

L'opération de mise à jour d'une donnée rémanente est réalisée par un module ayant déclaré cette donnée en entrée/sortie.



Le module effectuant la mise à jour travaille sur un exemplaire de la donnée initialisé avec sa version courante. Tout se passe ensuite comme s'il produisait une nouvelle version de la donnée : l'exemplaire sur lequel il travaille remplace la version courante lorsqu'il se termine.
Les modules effectuant la consultation de la donnée ont même définition et mêmes propriétés qu'au paragraphe II.2.2.
L'initialisation de la donnée peut être réalisée :

- par affectation d'une valeur de départ définie au moment du traitement de déclarations de connexion.

- par un unique module d'initialisation agissant comme un producteur. Son rôle se limite exclusivement à la production du premier exemplaire.

II.2.3.1. Un module en mise à jour, C consultants

Dans le cas où il n'y a qu'un seul module effectuant la mise à jour, tout se passe comme au paragraphe II. 2.2.2.

Lorsque le module effectuant la mise à jour MAJ doit être lancé, il reçoit immédiatement un exemplaire de la donnée renfermant la version courante. Il peut alors effectuer le traitement demandé (figure 23).

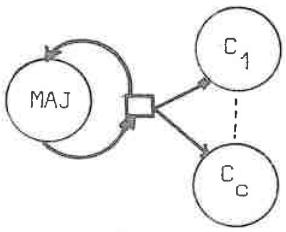


Fig.23 Un module en mise à jour, C consultants.

II.2.3.2. M modules en mise à jour, C en consultation

Lorsque m modules appartenant à des tâches concurrentes sont susceptibles de mettre à jour la donnée, il faut s'efforcer de conserver l'intégrité de la donnée en n'effectuant pas les mises à jour de façon anarchique. Cette opération doit être réalisée en exclusion mutuelle, le corps du module constituant une section critique pour la mise à jour d'une donnée.

La version courante reste cependant toujours disponible pour les modules qui veulent la consulter.

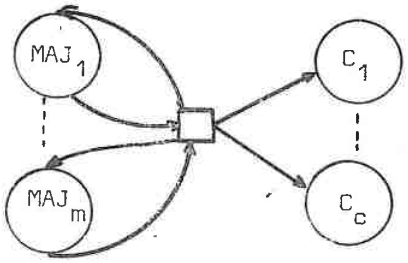


Fig.24 m modules en mise à jour, c en consultation.

II.2.3.3. Remarques

Dans ce mode d'utilisation des données rémanentes, la synchronisation des modules effectuant la mise à jour doit être assurée afin d'éviter de mauvaises utilisations de la donnée et la perte de sa signification.

Une tâche fonctionnelle reste bloquée tant qu'un de ses modules demandant l'accès en mise à jour à une telle donnée ne peut être lancé.

II.3. AUTRES UTILISATIONS DES DONNEES TRANSMISSIBLES

II.3.1. Autres modes de connexion

Les connexions de données transmissibles pourraient ne pas se limiter au schéma présenté précédemment où chaque élément transmis en sortie correspond à un élément figurant en entrée.

On pourrait ainsi utiliser les connexions entre modules pour modifier la nature des objets transmis :

- connexion de n sorties élémentaires sur un tableau de n éléments (fig.25)
- connexion d'un tableau de n éléments en sortie sur n entrées (fig.26).

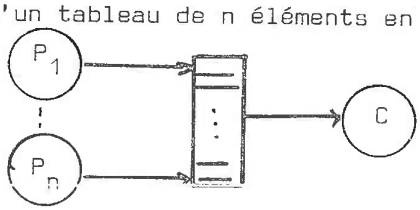


Fig.25 Connexion de n sorties sur une entrée de n éléments.

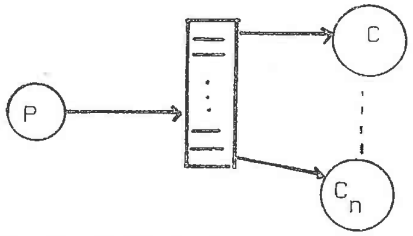


Fig.26 Connexion d'une sortie de n éléments sur n entrées.

De telles connexions peuvent intervenir à l'intérieur d'une même tâche fonctionnelle ou entre tâches fonctionnelles concurrentes.
La production ou la consommation de n éléments peut être effectuée par n modules distincts ou n activations d'un module producteur ou consommateur.
Lorsqu'il s'agit de tâches fonctionnelles concurrentes, les données peuvent être gérées au choix de l'utilisateur sous forme de données consommables (avec synchronisation des modules producteurs et consommateurs) ou sous forme de données rémanentes avec les mêmes propriétés que celles qui ont été présentées précédemment.

II.3.2. Autres modes de gestion de données transmissibles

D'autres modes de gestion des données pourraient être introduits pour permettre la prise en compte d'autres concepts :

- la limitation de la durée de vie d'une donnée à un laps de temps Δt défini : si un exemplaire de la donnée et produit à l'instant t et n'est pas consommé dans l'intervalle $(t, t+\Delta t)$ alors il doit disparaître.

- la limitation de la durée de vie d'une donnée à un nombre n fixé de consommations.

- la possibilité de laisser vieillir une donnée en introduisant un retard entre le moment de se production et celui de sa consommation.

- enfin, la combinaison des différents modes de gestion des données pour rendre compte des différentes actions que l'on peut être amené à entreprendre.

II.3.3. Conclusion

Ces utilisations des données transmissibles n'ont pas été introduites dans notre maquette de SYGARE car elles nécessitent une gestion importante qu'il n'était pas possible d'implanter sur un microprocesseur si l'on voulait qu'il puisse également exécuter des modules pour le compte d'une application.

Les problèmes d'encombrement-mémoire et de rapidité d'exécution se seraient posés avec acuité.

CHAPITRE III

LA GESTION DES DONNEES TRANSMISSIBLES

CHAPITRE III

LA GESTION DES DONNEES TRANSMISSIBLES

III.1. LA GESTION DE DONNEES CONSOMMABLES

Une donnée consommable associée à une file d'attente de taille n répond aux spécifications suivantes :

- un producteur de la donnée doit pouvoir être lancé s'il ne risque pas de faire déborder la file d'attente : le nombre d'exemplaires en file d'attente ou en cours de production doit être inférieur à n.

- Un producteur doit rester bloqué si la file est pleine ou risque de déborder.

- Un consommateur doit rester bloqué si aucun exemplaire n'est disponible en file d'attente.

- Un consommateur peut être lancé s'il existe au moins un exemplaire de la donnée.

En tenant compte de ces spécifications, il est possible de définir les états d'une donnée consommable.

III.1.1. Les états d'une donnée consommable

1° état : Aucun exemplaire de la donnée n'est disponible, des producteurs peuvent être lancés. Lesconsommateurs doivent rester bloqués. C'est dans cet état que se trouve la donnée lors de l'initialisation du système.

2° état : Aucun exemplaire de la donnée n'est disponible, mais n producteurs sont déjà actifs. On ne peut en lancer d'autres. Les consommateurs doivent rester bloqués.

3° état : Il existe d exemplaires de la donnée disponibles en file d'attente. p producteurs sont actifs et $d+p < n$ $(d>0,\ p\geq 0)$. D'autres producteurs peuvent être lancés. Les consommateurs peuvent recueillir un exemplaire de la donnée.

4° état : d exemplaires sont disponibles en file d'attente et p producteurs sont actifs : d+p = n (d>0, p≥0). Les nouveaux producteurs doivent rester bloqués. d consommateurs peuvent être lancés et recueillir un exemplaire de la donnée.

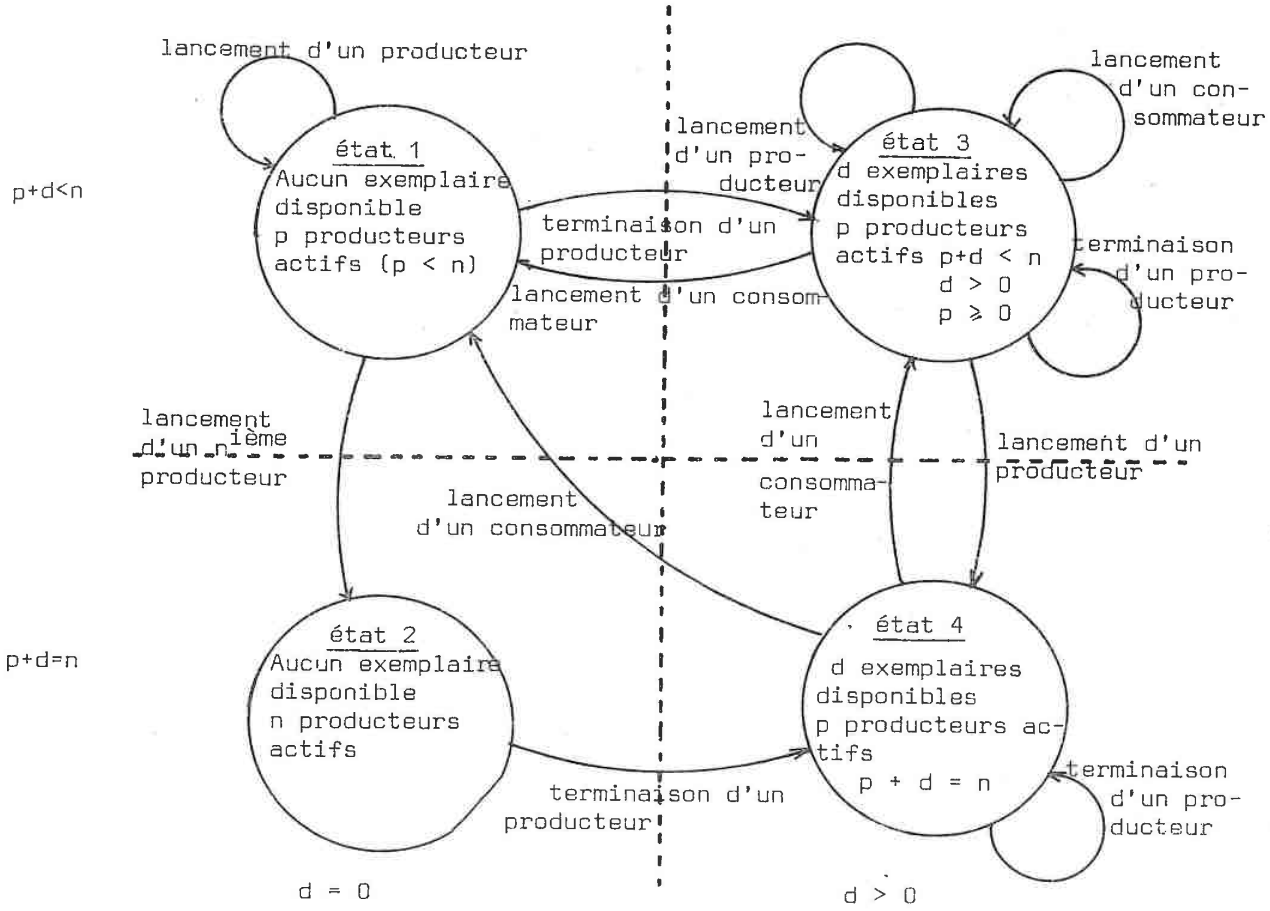


Fig.27 Représentation des graphes des transitions entre les états d'une donnée consommable.

La figure 27 présente le graphe des transitions entre les différents états dans lesquels peut se trouver une donnée consommable.

III.1.2. Allocation des données consommables

La gestion d'une donnée consommable doit tenir compte du nombre d'exemplaires de la donnée disponibles et de la taille maximale n de la file d'attente de consommation.

Deux variables entières simples : NAP et NAC, testés et modifiés par le système d'exploitation en section critique, permettent de connaître à tout moment l'état de la donnée. Ils indiquent le Nombre d'Accès à la donnée pour des

modules de type Producteur et le Nombre d'Accès à la donnée pour des modules de type Consommateur.

Le test et la modification de ces variables sont effectués au moment de lancer un module et lorsque celui-ci se termine.

. NAP est initialisé avec la taille maximale n du tampon afin d'indiquer que n productions sont autorisées avant le blocage des producteurs.

. NAC est initialisé à zéro afin d'indiquer que le tampon est initialement vide et que les consommateurs doivent rester bloqués.

. Le lancement d'un module producteur n'est possible que si la variable NAP est strictement positive, il s'accompagne d'une décrémentation de NAP et du prélèvement d'une zone de mémoires libres de la taille de la donnée destinée à recueillir l'exemplaire produit.

. La terminaison d'un module producteur s'accompagne d'un chaînage de l'exemplaire produit à la queue de la file d'attente et de l'incrémentation de NAC.

. Le lancement d'un module consommateur n'est possible que si NAC est strictement positif, il s'accompagne de la décrémentation de NAC du prélèvement de l'exemplaire situé en tête de la file d'attente et de l'incrémentation de NAP.

C'est sur l'exemplaire de la donnée ainsi récupéré que travaillera le module consommateur.
La terminaison du module consommateur s'accompagne du chaînage de l'exemplaire consommé en file des mémoires libres.

Le nombre d'exemplaires de la donnée produits et en attente de consommation est donné par la valeur de la variable NAC.
Le nombre de modules producteurs actifs simultanément est donné par l'expression :

$$n - (NAP + NAC)$$

Si garde est le sémaphore de protection utilisé par le système d'exploitation lors des opérations de lancement et de fin d'exécution d'un module, celles-ci peuvent être décrites de la façon suivante :

Lancement d'un module producteur :

```
        P(garde);
        si NAP = 0 alors aller à échec ;
                   sinon NAP:=NAP-1 ;
                         EXEMPLAIRE:= mémoire libre ;
                   fsi ;
        V(garde) ; on peut lancer le module
        fin ;
échec : V(garde) ; on ne peut lancer le module
```

Terminaison d'un module producteur :

```
P(garde) ;
NAC:=NAC + 1 ;
chaîner EXEMPLAIRE en tête de la file d'attente ;
V(garde) ;
fin ;
```

Lancement d'un module consommateur :

```
P(garde);
si NAC=0 alors  aller à échec ;
         sinon  NAC:=NAC-1 ;
                NAP:=NAP+1 ;
                EXEMPLAIRE:=tête de la file d'attente ;
         fsi ;
V(garde) ;
;on peut lancer le module ;
fin ;
échec : V(garde) ; on ne peut lancer le module
fin ;
```

Terminaison d'un module consommateur :

```
P(garde) ;
Chaîner EXEMPLAIRE en zone de mémoire libre ;
V(garde) ;
fin ;
```

III.1.3. Le descripteur d'une donnée consommable

Ce descripteur est une table dans laquelle sont rassemblés tous les éléments utiles à la gestion d'une donnée consommable. Elle contient donc (fig.28) :

- les indicateurs précisant que la donnée est de type consommable
- les variables entières NAP et NAC
- le lien de chaînage vers l'exemplaire de la donnée situé en tête de la file d'attente de consommation
- le lien de chaînage vers le dernier exemplaire de la donnée produit.

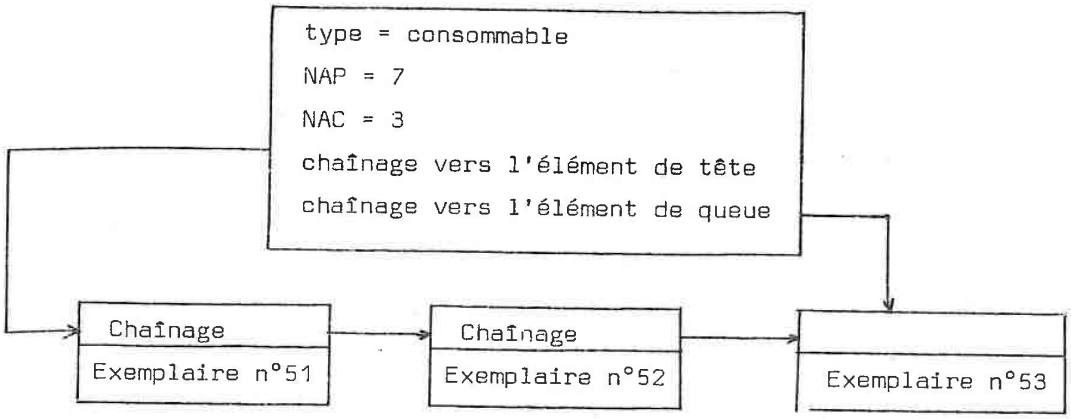


Fig.28 Descripteur d'une donnée consommable (taille de la file d'attente n=10).

III.2. LA GESTION DE DONNEES REMANENTES

Une donnée rémanente peut être utilisée par trois types de modules effectuant :

- la production d'une nouvelle version courante de la donnée ;
- la mise à jour de la donnée ;
- la consultation de la version courante de la donnée.

Rappelons les caractéristiques de chaque opération :

. La production d'un nouvel exemplaire est toujours possible : lorsque le module se termine, l'exemplaire produit devient version courante.

. La mise à jour de la donnée s'effectue en exclusion mutuelle.
Elle n'est possible que si une version courante de la donnée existe. Lorsque le module se termine, l'exemplaire produit devient version courante.

. La consultation de la donnée n'est possible que s'il en existe une version courante. Celle-ci n'est pas modifiée par l'exécution du module.

Ces spécifications permettent de définir les trois états d'une donnée rémanente.

III.2.1. Les états d'une donnée rémanente

1° état : La donnée n'a pas encore été produite ou n'est pas initialisée, seuls les modules producteurs peuvent être lancés.

2° état : La donnée a été produite, il en existe une version courante que l'on peut consulter. Tout module peut être lancé.

3° état : La donnée a été produite, il en existe une version courante. Un module

effectue la mise à jour d'un exemplaire. Seuls les modules effectuant la consultation peuvent être lancés.

La figure 29 donne une représentation du graphe des états dans lesquels une donnée rémanente est susceptible de se trouver.

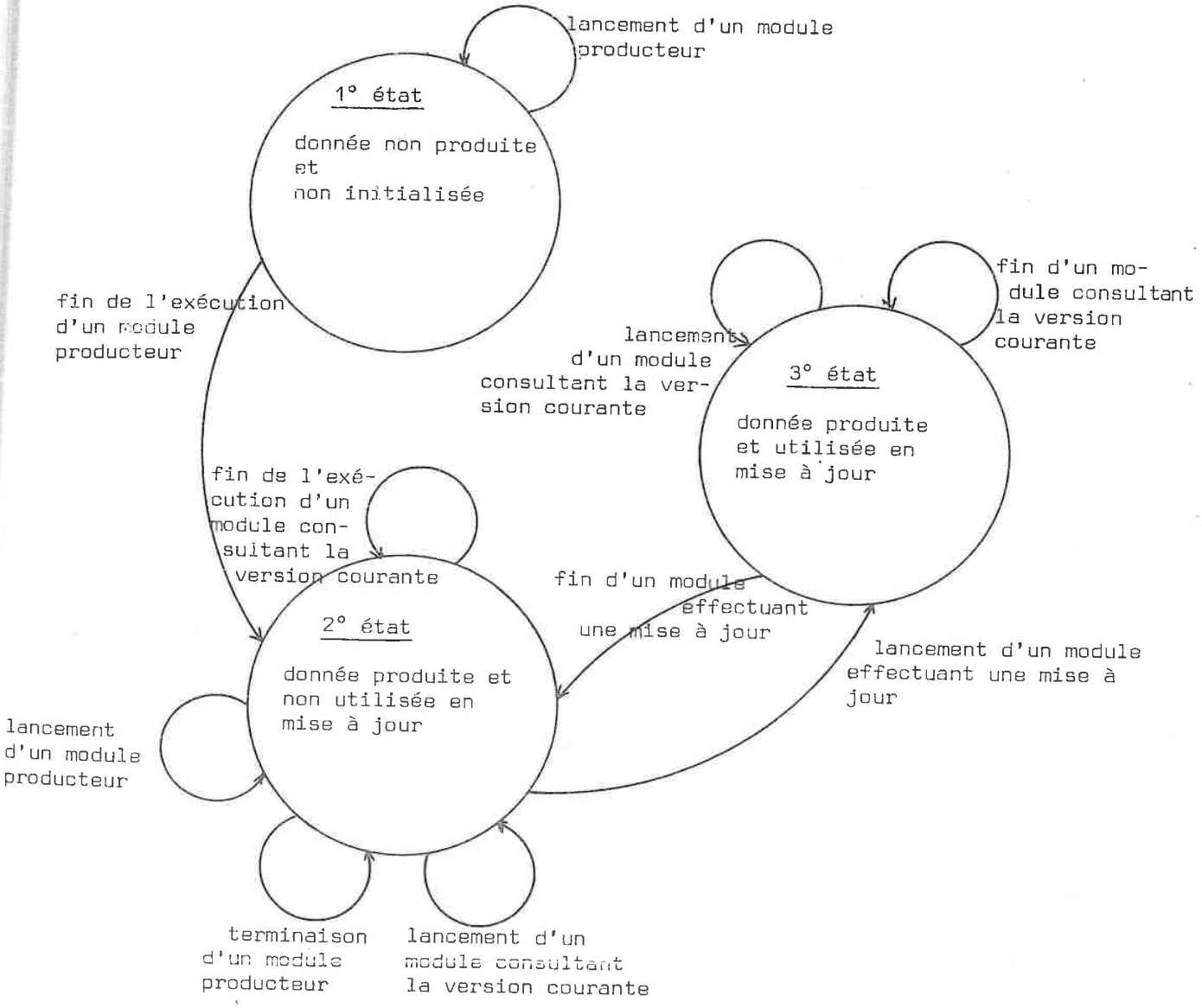


Fig.29 Représentation du graphe des transitions entre les états d'une donnée rémanente.

III.2.2. Les états d'un exemplaire d'une donnée rémanente

Les différents exemplaires d'une donnée rémanente susceptibles de coexister sont :

- des exemplaires utilisés en production : ils renferment une version future de la donnée.
- un exemplaire utilisé ou non en consultation : il renferme la version courante de la donnée.
- des exemplaires périmés utilisés en consultation : ils renferment des versions passées de la donnée.
- un exemplaire utilisé en mise à jour par un module : il renferme une version en mise à jour.

Les transitions entre les différentes versions d'une même donnée sont les suivantes (fig.30) :

- Le lancement d'un module producteur crée une nouvelle version future.
- La terminaison d'un module producteur transforme la version courante en version passée si elle est utilisée, la fait disparaître sinon. L'exemplaire utilisé devient version courante.
- Le lancement d'un module effectuant la mise à jour crée une version en mise à jour initialisée avec la version courante.
- La fin d'un module effectuant une mise à jour a les mêmes conséquences que la fin d'un module producteur.
- La fin d'un module consultant la donnée fait disparaître l'exemplaire utilisé s'il correspond à une version passée qui n'est plus utilisée.

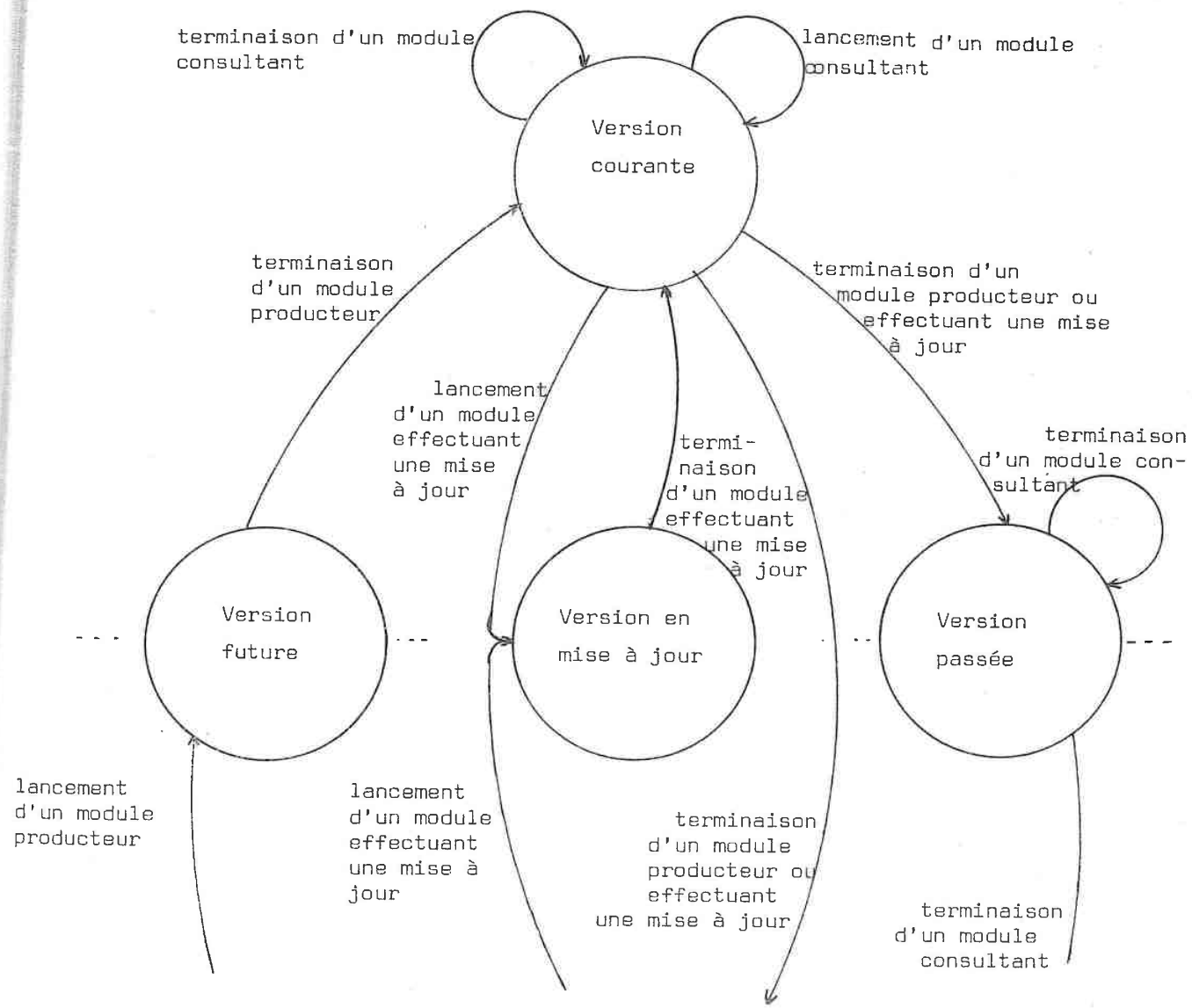


Fig.30 Représentation de transitions entre les différents états d'un exemplaire d'une donnée rémanente.

III.2.3. Gestion des données rémanentes

La gestion des données rémanentes doit tenir compte de l'état global de la donnée et de l'état de l'exemplaire employé.

. Deux variables logiques simples permettent de coder l'état global de la donnée :

- PROD indique si la donnée a déjà été produite ou si elle a été initialisée, si PROD possède la valeur .VRAI. alors il existe une version courante de la donnée.

- MAJ indique avec PROD si on peut lancer un module effectuant une mise à jour de la donnée s'il possède la valeur .VRAI.. La valeur initiale de MAJ est .VRAI..

.Une variable entière simple NU et une variable logique ETAT permettent de connaître le nombre d'utilisateurs d'un exemplaire et si celui-ci renferme une version courante (ETAT = .VRAI) ou passée. Seul le système d'exploitation peut tester et modifier ces variables en section critique uniquement.

- Le lancement d'un module producteur s'accompagne de la création d'une version future. Il n'entraîne aucune modification des variables.

- La terminaison d'un module producteur s'accompagne des modifications suivantes : l'exemplaire renfermant la précédente version actuelle est chaînée en zone de mémoire libre si NU vaut zéro, sinon la variable ETAT correspondant à cet exemplaire passe à l'état FAUX.. L'exemplaire utilisé devient version courante : NU est initialisé à zéro et ETAT à .VRAI..

- Le lancement d'un module consultant la version courante n'est possible que si PROD possède la valeur .VRAI., il s'accompagne de l'incrémentation de la variable NU correspondant à la version courante.

- La terminaison d'un module consultant la donnée s'accompagne des opérations suivantes : la variable NU de l'exemplaire utilisé est décrémentée, si la valeur obtenue est nulle et si ETAT possède la valeur .FAUX., l'exemplaire est chaîné en zone de mémoire libre.

- Le lançement d'un module effectuant une mise à jour de la donnée n'est possible que si PROD et MAJ possèdent la valeur .VRAI. il y a alors création d'une version en mise à jour et on affecte la valeur .FAUX. à MAJ.

- Les opérations réalisées lors de la terminaison d'un module effectuant la mise à jour de la donnée sont similaires à celles accompagnant la terminaison d'un module producteur, MAJ reçoit en outre la valeur .VRAI.

Si garde est le sémaphore de protection utilisé par le système lors du lancement ou de la terminaison ou d'un module, les opérations effectuées correspondent à l'algorithme suivant :

Lancement d'un module producteur :

```
P(garde) ;
EXEMPLAIRE:= mémoire libre ;
V(garde) ;
fin ;
```

Terminaison d'un module producteur :

```
P(garde) ;
si PROD alors
  si NU.VERSION COURANTE=0 alors chaîner VERSION COURANTE en zone de
                                 mémoires libres ;
                           sinon ETAT.VERSION COURANTE:=.FAUX.;
                           fsi ;
  fin ;
VERSION COURANTE:=EXEMPLAIRE ;
ETAT.VERSION COURANTE:=.VRAI.;
NU.VERSION COURANTE:=0 ;
PROD:=.VRAI.;
V(Garde) ;
fin ;
```

Lancement d'un module consultant la version courante :

```
      P(garde) ;
      si PROD alors aller à échec ;
             sinon EXEMPLAIRE:=VERSION COURANTE;
                   NU.VERSION COURANTE:=NU.VERSION COURANTE+1 ;
             fsi ;
      V(garde) ;
      fin ;   on peut lancer le module
échec: V(garde) ;
      fin ;
```

Terminaison d'un module consultant la donnée :

```
        P(garde) ;
        NU.EXEMPLAIRE:=NU.EXEMPLAIRE-1 ;
        si (NU.EXEMPLAIRE=0)& ETAT.EXEMPLAIRE alors
                        chaîner EXEMPLAIRE en zone de mémoire libre ;
                                                fsi ;
        V(garde) ;
        fin ;
```

Lancement d'un module effectuant la mise à jour :

```
        P(garde) ;
        si PROD + MAJ alors aller à échec ;
                    sinon EXEMPLAIRE:=mémoire libre ;
                          (EXEMPLAIRE):=(VERSION COURANTE) ;
                          MAJ:=.FAUX.;
                    fsi ;
        V(garde) ; on peut lancer le module
        fin ;
échec:  V(garde) ; on ne peut lancer le module
        fin ;
```

Terminaison d'un module effectuant la mise à jour :

```
        P(garde) ;
        si NU.VERSION COURANTE =0 alors
                    chaîner VERSION COURANTE en zone de mémoire libre ;
                                sinon ETAT.VERSION COURANTE:=.FAUX.;
                                fsi ;
        VERSION COURANTE:=EXEMPLAIRE ;
        ETAT.VERSION COURANTE:=.VRAI.;
        NU.VERSION COURANTE:=0 ;
        MAJ:=.VRAI.
        V(garde) ;
        fin ;
```

III.2.4. Le descripteur de donnée rémanente

Le descripteur d'une donnée rémanente est la table dans laquelle sont rassemblés tous les éléments utiles à la gestion de la donnée. Elle contient donc (fig.31) :

- un indicateur précisant que la donnée est de type rémanent,
- les variables logiques PROD et MAJ,
- un relais vers la version courante de la donnée.

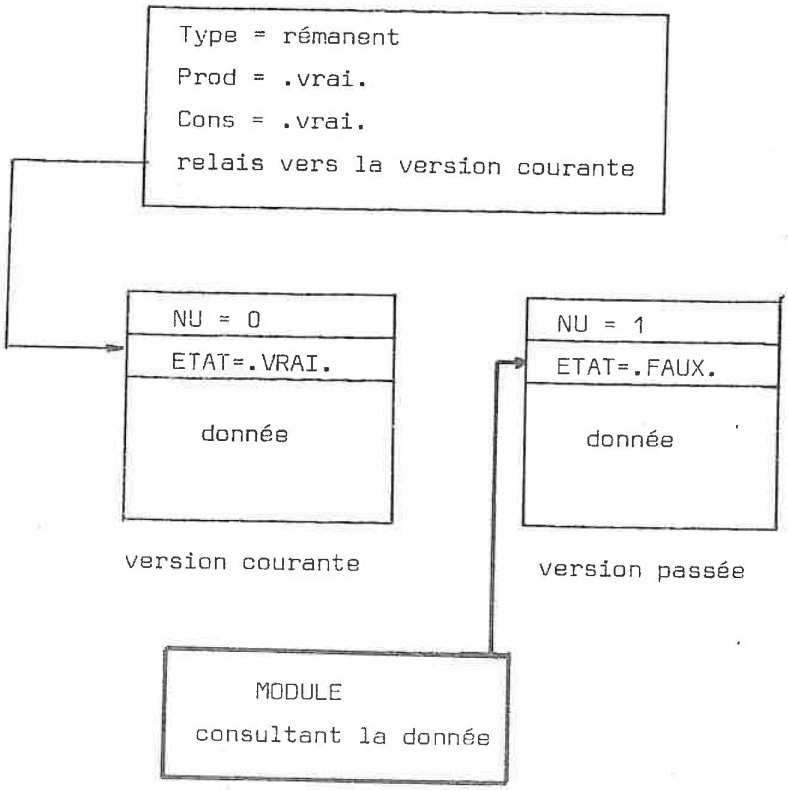


Fig.31 Description de donnée rémanente : une version courante, une version passée.

III.3. CONCLUSION

Dans ce chapitre, nous avons présenté la protection qui est réalisée autour des données transmissibles pour les différents modes d'utilisation proposés.

Nous avons décrit le mécanisme d'allocation d'une donnée à un module dans le cas simple où le module n'utilise qu'une seule donnée. Celui-ci doit cependant être étendu pour recouvrir le traitement de modules utilisant plusieurs données, ce qui représente le cas général.

Le chapitre suivant présente le traitement des déclarations de connexion dans la description d'une tâche fonctionnelle puis l'allocation d'un ensemble de données à un module ou à un ensemble de modules parallèles.

Le chapitre V propose un schéma d'exécution d'une tâche fonctionnelle. Celui-ci est étudié pour traiter le cas des modules utilisant tous les types de données.

CHAPITRE IV

DESCRIPTION D'UNE TACHE FONCTIONNELLE

CHAPITRE IV

DESCRIPTION D'UNE TACHE FONCTIONNELLE

La description d'une tâche fonctionnelle est réalisée en deux étapes distinctes dont le rôle est complémentaire.

. Description des structures de contrôle de classe II qui lient les modules (paragraphe I.4.2).

. Description des connexions de données transmissibles entre les différents éléments de la décomposition.

IV.1. DESCRIPTION DES STRUCTURES DE CONTROLE DE CLASSE II

La description des structures de contrôle de classe II, qui interviennent dans une tâche fonctionnelle, permet la construction du graphe qui représente son exécution. Celle-ci se traduit par un cheminement dans le graphe. L'exécution d'un module est représentée par un arc, un point de synchronisation, par un noeud.

IV.1.1. Formalisation de la description

La constitution d'une tâche fonctionnelle peut être décrite dans un langage tenant compte des structures de contrôle de classe II dont la grammaire est la suivante :

```
<TACHE FONCTIONNELLE>::=<ACTION1>
        <ACTION1>::=<ACTION2>-<ACTION1>|<ACTION2>
        <ACTION2>::=<ACTION3>//<ACTION2>|<ACTION3>
        <ACTION3>::=(<ACTION1>)|
                    <MODULE>|
                    <CONDITIONNELLE>|
                    <ITERATIVE>
<CONTIONNELLE>::= si <CONDITION> alors <ACTION1> fsi|
                  si <CONDITION> alors <ACTION1>
                                 sinon <ACTION1> fsi
   <ITERATIVE>::= tant que <CONDITION> faire <ACTION1> ftq
   <CONDITION>::= nom de donnée transmissible de type logique
      <MODULE>::= suite de déclarations et d'instructions.
```

La grammaire que nous avons adoptée est telle que, en l'absence de tout parenthèsage explicite, l'expression :

A - B // C - D

soit interprétée comme si l'on avait écrit : A - (B // C) - D

Les structures conditionnelles ou itératives sont entièrement parenthésées par les mots clefs : si et fsi ou tant que et ftq, ceci permet de les faire porter sur une expression complexe, et de limiter explicitement leur portée.

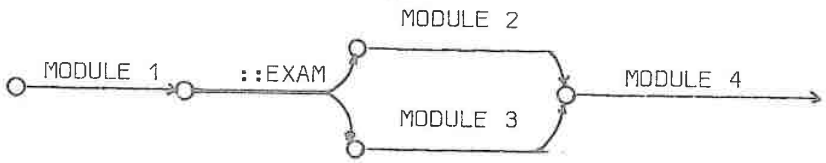
IV.1.2. Traduction de la description

La description d'une tâche fonctionnelle se traduit par la création d'une table renfermant les noms des modules qui interviennent et les chaînages qui les relient. Cette table traduit le graphe représentant l'exécution de la tâche fonctionnelle.

La traduction d'une structure conditionnelle ou itérative nous conduit à introduire un "pseudo-module" nommé ::EXAM qui réalise le choix entre deux groupes de successeurs possibles en fonction d'une condition. Ce "pseudo-module" fait partie intégrante du système d'exploitation. En l'état actuel du projet, la condition est définie par la valeur d'une donnée transmissible rémanente de type logique. La structure conditionnelle

MODULE 1 - si BOOLE alors MODULE 2
sinon MODULE 3 fsi - MODULE 4.

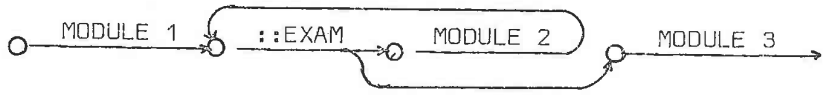
se traduit par le graphe :



La structure itérative :

MODULE 1 - tant que BOOLE faire MODULE 2 ftq - MODULE 3

se traduit par le graphe :



En utilisant le pseudo module ::EXAM à d'autres endroits du graphe, on pourrait introduire d'autres structures de contrôle (repeat, exit_if ...). On pourrait également, en le développant autoriser des structures plus évoluées du type case.

L'introduction de telles structures de contrôle ne nous a cependant pas paru un point primordial dans le cadre du développement de la maquette de SYGARE.

IV.2. DESCRIPTION DES CONNEXIONS DE DONNEES TRANSMISSIBLES

La description des connexions permet de préciser pour chaque appel de module, quelles sont les données transmissibles qu'il utilise indépendamment du site où elles sont implantées.

Un module peut en effet être appelé plusieurs fois dans une même tâche fonctionnelle et s'exécuter avec des contextes différents.
Le contexte d'exécution d'un module est défini lorsqu'on connaît la liste de ses prédécesseurs et de ses successeurs et que l'on a établi les liens entre les noms locaux désignant les données transmissibles et les données elles-mêmes.

IV.2.1. Description des connexions

Les connexions sont décrites par des flèches liant les noms locaux des données transmises.

IV.2.1.1. Données produites par des modules appartenant à la tâche fonctionnelle

Dans le cas de modules appartenant à la même tâche fonctionnelle, la transmission de donnée s'effectue dans un sens seulement. Les flèches sont mono-directionnelles. Pour chaque module, on précise l'origine des données figurant en entrée.

Exemple : B.MODULE2 <- A.MODULE1

Une même donnée peut être connectée aux sorties de deux modules s'excluant mutuellement dans le cas d'une structure conditionnelle :

Exemple : D.MODULE4 <- B.MODULE2 + C.MODULE3

Si MODULE2 a été exécuté la connexion réalisée est :

D.MODULE4 <- B.MODULE2

dans le cas contraire, la connexion est : D.MODULE4 <- C.MODULE3.

IV.2.1.2. Données utilisées par d'autres tâches fonctionnelles

Lorsque les données utilisées en entrée d'un module sont de type consommable, il faut en préciser les modules producteurs et la taille du tampon désirée.

Exemples : A.MODULE1 <- D.PROD.TACHE2 (TAILLE=5)

A.MODULE1 <- D.PROD1.TACHE2+E.PROD2.TACHE3 (TAILLE=2)

Lorsqu'il s'agit de consultation d'une donnée rémanente, il suffit de préciser quels en sont les différents producteurs possibles.

Exemples : A.MODULE1 <- K.PROD.TACHES

A.MODULE1 <- L.PROD1.TACHE6 + M.PROD2. TACHE7.

Lorsqu'il s'agit de mise à jour d'une donnée rémanente, il faut indiquer par une double flèche quels sont les autres modules qui partagent la donnée, le nom du module créant la version initiale de la donnée ou sa valeur, s'il s'agit d'une donnée initialisée au départ :

Exemples : A.MODULE1 <-> B.MODULE2.TACHE2
(INIT=C.MODULE3.TACHINIT)

A.MODULE1 <-> B.MODULE2.TACHE2
(INIT=5)

Ces déclarations permettent de créer la table de connexion correspondant à la tâche fonctionnelle que l'on veut décrire.

IV.2.2. Mémorisation des connexions

Les connexions de données transmissibles utilisées par une tâche fonctionnelle sont mémorisées dans une table de connexions. Cette table contient autant de postes qu'il y a de modules intervenant dans la description de la tâche fonctionnelle. Chaque poste permet de définir avec précision le contexte d'exécution du module concerné. Il renferme autant d'éléments que de données transmissibles utilisées par le module. Cela permet de mémoriser le nom de la donnée qui sera effectivement produite, consommée ou consultée lors du p ième appel au module dans cette tâche fonctionnelle.

IV.3. EXEMPLE DE CONSTRUCTION D'UNE TACHE FONCTIONNELLE

Nous allons montrer sur un exemple les différentes opérations nécessaires à la construction d'une tâche fonctionnelle.

- la description des modules et des données transmissibles qu'ils utilisent.
- la description des structures de contrôle de classe II qui lient les modules entre eux.
- la description du flux de donnée entre les différents modules.

IV.3.1. Ecriture des modules

Supposons que la tâche fonctionnelle à exécuter puisse être découpée en quatre actions séparées. Celles-ci sont décrites par quatre modules écrits de

de façon indépendante.

. Le module MODULE 1 utilisé pour coder l'action numéro 1 permet d'évaluer trois données S1, S2 et S3 en fonction d'un algorithme. Les données transmissibles S1, S2 et S3 sont donc des sorties de MODULE 1. Il peut être écrit de la façon suivante :

Module : MODULE1
entiers : S1, S2, S3
sorties : S1, S2, S3
algorithme de l'action 1.
fin

L'aspect du module compilé est le suivant :

MODULE 1 :

S1 : entier ; sortie ; relais 1
S2 : entier ; sortie ; relais 2
S3 : entier ; sortie ; relais 3
relais 1
relais 2
relais 3
variables internes, constantes
algorithme de l'action 1

. L'action codée par MODULE 2 utilise trois données transmissibles dont l'une apparaît en entrée seulement, la seconde en sortie seulement et la troisième en entrée/sortie. Ce module peut être écrit de la façon suivante :

Module : MODULE2
entiers : E, S, ES
entrées : E, ES
sortie : S, ES
algorithme de l'action 2
fin

Son aspect après compilation est le suivant :

MODULE 2 :

E : entier ; entrée ; relais 1
ES : entier ; entrée/sortie ; relais 2
S : entier ; sortie ; relais 3
relais 1
relais 2
relais 3
variables internes, constantes
algorithme de l'action 2

. La troisième action, codée par MODULE 3, utilise trois données transmissibles pour être exécutées : E1 et E2 en entrées, S en sortie. Le module peut être écrit ainsi :

Module : MODULE3
entiers : E1, E2, S
entrées : E1, E2
sortie : S
algorithme de l'action 3
fin

Son aspect après compilation est alors :

E1 : entier ; entrée ; relais 1
E2 : entier ; entrée ; relais 2
E3 : entier ; sortie ; relais 3
relais 1
relais 2
relais 3
variables internes, constantes
algorithme de l'action 3

. Enfin, l'action numéro 4 peut être codée par MODULE 4 et utilise deux données transmissibles en entrée : E1 et E2. Le module s'écrit alors :

Module : MODULE4
entiers : E1, E2
entrées : E1, E2
algorithme de l'action 4
fin

Après compilation, il devient :

MODULE 4

E1 : entier ; entrée ; relais 1
E2 : entier ; entrée ; relais 2
relais 1
relais 2
variables internes, constantes
algorithme de l'action 4

IV.3.2. Description des structures de contrôle

La tâche que nous voulons construire peut être décrite de façon schématique par l'organigramme suivant (fig.32).

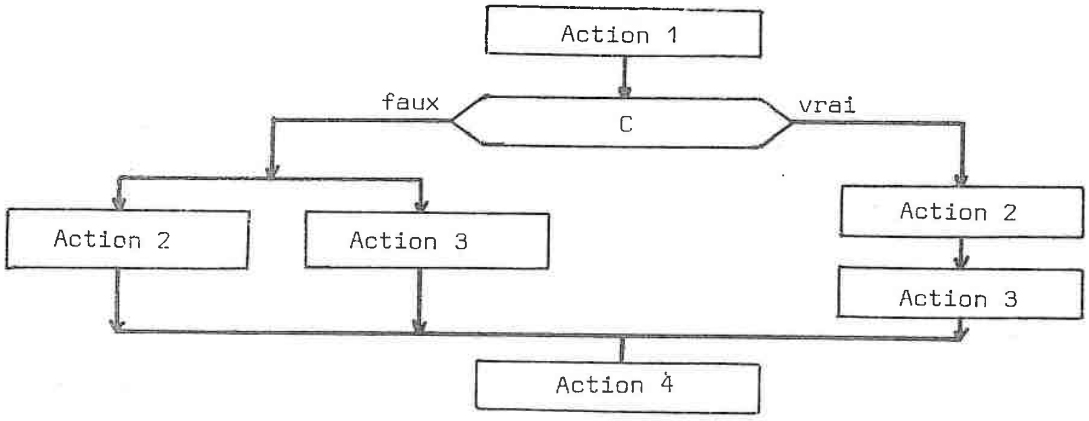


Fig.32 Représentation de la tâche fonctionnelle : TACHE.

Les structures de contrôle définissant cette tâche peuvent être décrites par l'enchaînement suivant :

TACHE : MODULE1 - si BOOLE alors MODULE2 - MODULE3
sinon MODULE2 // MODULE3 fsi
- MODULE4.

Le traitement d'une telle déclaration d'enchaînement permet de construire la table d'enchaînement des modules présentée à la figure 33. Le numéro d'ordre des modules est concaténé à leurs noms pour permettre de distinguer leurs différentes occurences dans la tâche fonctionnelle.

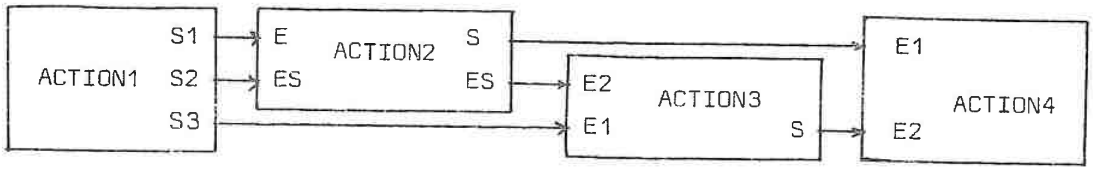
```
1MODULE1 :  NB PREDECESSEURS = 0
            NB SUCCESSEURS   = 1
            SUCCESSEURS :      2 :: EXAM

2::EXAM  :  NB PREDECESSEURS = 1
            NB SUCCESSEURS   = 1 ; 2   < Deux groupes
            PREDECESSEURS :    1MODULE1
            SUCCESSEURS :      3MODULE2
                               5MODULE2 ; 6MODULE3

3MODULE2 :  NB PREDECESSEURS = 1
            NB SUCCESSEURS   = 1
            PREDECESSEURS :    2::EXAM
            SUCCESSEURS :      4MODULE3

4MODULE3 :  NB PREDECESSEURS = 1
            NB SUCCESSEURS   = 1
            PREDECESSEURS :    3MODULE2
            SUCCESSEURS :      7MODULE4

5MODULE2 :  NB PREDECESSEURS = 1
            NB SUCCESSEURS   = 1
            PREDECESSEURS :    2::EXAM
            SUCCESSEURS :      7MODULE4

6MODULE3 :  NB PREDECESSEURS = 1
            NB SUCCESSEURS   = 1
            PREDECESSEURS :    2::EXAM
            SUCCESSEURS :      7MODULE4

7MODULE4    NB PREDECESSEURS = 1 ; 2  < Deux groupes
            NB SUCCESSEURS   = 0
            PREDECESSEURS :    4MODULE3
                               5 MODULE2 ; 6 MODULE3
```

Fig.33 Table d'enchaînement des modules de TACHE.

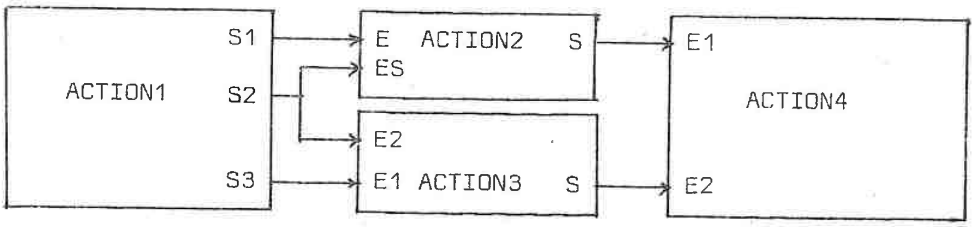
IV.3.3. Description du flux de données

En fonction de la valeur de la donnée transmissible BOOLE examinée dans le test, le flux de données dans la tâche fonctionnelle correspond à l'un des schémas ci-après :

1° Cas : condition vraie



2° Cas : condition fausse



La donnée transmissible BOOLE utilisée dans la structure conditionnelle est produite par MODULES appartenant à TACHE3 sous le nom LOGIC. Les déclarations de connexion sont alors les suivantes :

```
        BOOLE ← LOGIC.MODULES.TACHE3
E.MODULE 2 ← S1.MODULE 1
ES.MODULE 2 ← S2.MODULE 2
E1.MODULE 3 ← S3.MODULE 1
E2.MODULE 3 ← ES.MODULE 2 + S2.MODULE 1
E1.MODULE 4 ← S.MODULE 2
E2.MODULE 4 ← S.MODULE 3
```

Lorsqu'on ne précise pas la tâche à laquelle appartient le module, celui-ci appartient implicitement à la tâche que l'on construit.

La table de connexion correspondant à cette tâche fonctionnelle aura donc l'aspect suivant (fig.34).

```
1MODULE1 :  DONNEE1                 < S1 ; SORTIE
            DONNEE2                 < S2 ; SORTIE
            DONNEE3                 < S3 ; SORTIE

2:: EXAM :  LOGIC.MODULES.TACHE3    < CONDITION

3MODULE2 :  DONNEE1                 < E ; ENTREE
            DONNEE2                 < ES ; ENTREE/SORTIE
            DONNEE4                 < S ; SORTIE

4MODULE3 :  DONNEE3                 < E1 ; ENTREE
            DONNEE4                 < E2 ; ENTREE
            DONNEE5                 < S  ; SORTIE

5MODULE2 :  DONNEE1                 < E  ; ENTREE
            DONNEE2                 < ES ; ENTREE/SORTIE
            DONNEE4                 < S  ; SORTIE

6MODULE3 :  DONNEE3                 < E  ; ENTREE
            DONNEE2                 < E2 ; ENTREE
            DONNEE5                 < S  ; SORTIE

7MODULE4 :  DONNEE4                 < E  ; ENTREE
            DONNEE5                 < E2 ; ENTREE
```

Fig.34 Table de connexion de TACHE.

IV.4. CONCLUSIONS

Nous avons pu remarquer dans ce chapitre qu'une tâche est construite de façon à pouvoir être modifiée de façon très simple :

- modification d'un module,
- modification des structures de contrôle de classe II,
- modification du flux de données.

Cette construction qui est réalisée indépendamment de l'implémentation des modules de la tâche fonctionnelle n'a pas à être reprise lorsqu'on désire modifier

la répartition de ses constituants sur le réseau.

- déplacement d'un module appartenant à la tâche fonctionnelle
- déplacement d'un module produisant ou consommant une donnée utilisée par la tâche.

Les programmes permettant la construction d'une tâche fonctionnelle ont été réalisés sur le SOLAR 16-40 (SEMS) et fonctionnent en mode traitement par lot. Ils permettent de créer les fichiers qui seront ensuite utilisés lors de l'implantation de la tâche fonctionnelle sur le réseau puis lors de son exécution.

CHAPITRE V

L'EXECUTION D'UNE TACHE FONCTIONNELLE

CHAPITRE V

L'EXECUTION D'UNE TACHE FONCTIONNELLE

L'exécution d'une tâche fonctionnelle sur le réseau SYGARE fait intervenir deux composants essentiels du système d'exploitation :

- l'interpréteur des structures de contrôle de classe II,
- le gérant des données transmissibles

La première partie de ce chapitre est consacrée à la présentation de ces éléments, ils permettent d'assurer le cheminement dans le graphe décrivant la succession des modules dans la tâche fonctionnelle, et de contrôler le flux des données dans l'application.

La seconde partie présente l'exécution d'une tâche fonctionnelle proprement dite.

V.1. INTERPRETEUR DES STRUCTURES DE CONTROLE DE CLASSE II

L'interpréteur des structures de contrôle de classe II permet l'exécution d'une tâche fonctionnelle décrite par la table d'enchaînement des modules telle qu'elle a été présentée au chapitreIV. Il n'y a en effet aucune édition de liens statique entre les différents composants d'une tâche et les modules rendant le contrôle au système d'exploitation lorsqu'ils se terminent. A lui de déterminer quel est alors le prochain module à exécuter et d'envoyer éventuellement un message au calculateur-hôte du réseau sur lequel ce module est implanté, afin d'en demander l'exécution. Cette détermination est réalisée à l'aide de la table d'enchaînement des modules : connaissant les noms des successeurs d'un module, on peut alors vérifier que tous les modules appartenant à la même étape que lui ont bien été exécutés.

Si c'est effectivement le cas, on peut passer à l'étape suivante. Sinon, il faut attendre que l'étape en cours se termine, et au besoin lancer les modules qui étaient en attente d'un calculateur.

V.2. GERANTS DE DONNEES TRANSMISSIBLES

L'accès aux données transmissibles n'est pas réalisé directement par les

modules mais par l'intermédiaire de gérants. Ceux-ci sont répartis sur les différents sites du réseau afin d'être plus proches des modules et de permettre une gestion plus efficace des données. Les avantages apportés par cette méthode sont les suivants :

- N'ayant pas accès lui-même aux données, un module ne peut effectuer d'opérations frauduleuses sur elles.

- Les conflits et les blocages résultant d'une mauvaise utilisation des données par suite de mauvais fonctionnement ou d'erreur de conception sont réglés par le gérant des données.

- La synchronisation des modules est réalisée indépendamment de l'action qu'ils représentent.

- La gestion des différents exemplaires des données et leur transmission aux modules qui les utilisent est codée de façon standard.

La conséquence de ces deux derniers points est fondamentale pour le concepteur d'une application modulaire : il est déchargé de tout ce qui concerne les procédures de communication, de synchronisation et de gestion de données. Il en découle un gain de temps appréciable lors de la conception et des tests de l'application. Les gérants peuvent fonctionner dans un mode mise au point qui permet de garder une trace des opérations réalisées. On peut ainsi procéder à une mise au point plus rapide de l'application. Un sous-produit de l'utilisation des gérants de données est le gain de place en mémoire qui en résulte puisque les diverses procédures sont codées une fois pour toutes.
L'introduction des gérants de données transmissibles résoud les problèmes de synchronisation et de blocage des modules. Les problèmes de conflits d'accès aux données se posent cependant au niveau des gérants et leur résolution peut être faite grâce à l'adoption d'une procédure d'allocation des données qui sera utilisée systématiquement à chaque fois qu'un accès sera demandé.

V.3. LES PROCEDURES MISES EN OEUVRE PAR L'EXECUTION D'UNE TACHE FONCTIONNELLE

Nous présentons, dans cette section, les procédures mises en oeuvre par les gérants de données transmissibles et par l'interpréteur des structures de contrôle de classe II lors du lancement ou de la terminaison d'un module.

V.3.1. Le lancement d'un module

Nous nous intéresserons tout d'abord au cas simple ou l'étape de la tâche fonctionnelle à lancer ne comprend qu'un unique Module M n'utilisant que des données implantées sur le même site que lui.

V.3.1.1. Module implanté sur le même site que les données

Connaissant les données connectées au module, la table d'utilisations permet de préciser comment elles interviennent dans ce module et le descripteur de données, l'état dans lequel celles-ci se trouvent à ce moment.

Deux cas peuvent alors se présenter :

a) . Toutes les données employées par le module M n'apparaissent en fait que dans cette seule tâche fonctionnelle.

. Celles qui figurent en entrée sont donc déjà produites et il suffit avant de lancer le module, d'initialiser les relais vers ces données.

. Celles qui figurent en sortie sont créées par le module, il faut leur allouer une zone de mémoires libres dans lesquelles le module rangera leurs valeurs.

. Celles qui figurent en entrée/sortie ne sont utilisées par aucun autre module, M peut les modifier sans difficulté ; il suffit alors d'initialiser le relais vers les exemplaires existants.

Lorsque ces opérations ont été réalisées, le module peut être lancé.

b) . Parmi les données utilisées par le module, certains peuvent être utilisés par d'autres tâches fonctionnelles. Il faut assurer la synchronisation entre ces tâches afin de tenir compte de l'utilisation des données en cours. L'allocation de l'ensemble des données transmissibles à un module doit être effectuée de façon indivisible par le gérant des données afin d'éviter les situations conflictuelles conduisant à un blocage de l'application.

Si en effet, M a besoin pour s'exécuter de deux données D1 et D2 telles que D1 soit disponible et D2 non produite, l'allocation de D1 à M pourrait empêcher l'exécution de modules produisant D2.

Un autre cas de blocage peut apparaître si, au cours de l'allocation des données à M, le gérant des données transmissibles s'interrompt pour traiter une requête provenant d'un autre site.

Ces deux problèmes peuvent être résolus par :

- l'entrée en section critique du gérant des données transmissibles lors du traitement d'un module.

- l'abandon du traitement d'un module sans avoir modifié l'état des données transmissibles lorsque certaines d'entre elles ne sont pas disponibles pour le

module que l'on veut lancer.

L'opération doit donc être réalisée en deux étapes :

Après être entré en section critique, le gérant de données doit effectuer un test de compatibilité portant sur l'ensemble des données utilisées par le module.

Si la conclusion est négative, il abandonne provisoirement le traitement de ce module et quitte la section critique.

Si au contraire, la conclusion du test est positive, on peut passer à la seconde étape qui se déroule toujours dans la même section critique.

Le gérant procède à l'allocation de toutes les données utilisées par le module et modifie éventuellement leur descripteur en tenant compte du mode d'utilisation des données. Il initialise le relais du module vers les exemplaires des données utilisées.

Le gérant des données transmissibles quitte alors sa section critique et le module peut être lancé. L'algorithme du traitement réalisé par le gérant est le suivant :

Algorithme utilisé pour le lancement d'un module lorsque toutes les données sont implantées sur le lieu d'exécution du module.

```
P(garde) ;
pour chaque donnée transmissible utilisée répéter
   si mode d'utilisation = entrée alors
      si type = consommable alors
         si NAC = 0 alors   aller à échec ; il n'y a pas d'exemplaire en file d'attente
                    fsi ;
                          fsi ;
      si type = rémanent    alors
         si PROD    alors aller à échec ; il n'y a pas d'exemplaire produit
                    fsi ;
                          fsi ;
                                fsi ; on pourra consommer ou consulter la donnée
   si mode d'utilisation = sortie alors
      si type = consommable alors
         si NAP = 0 alors   aller à échec ; on ne peut pas produire de nouvel exem-
                    fsi ;                   plaire
                          fsi ;
                                fsi ; on pourra produire la donnée
   si mode d'utilisation = entrée/sortie alors
      si type = rémanent    alors
         si PROD + MAJ alors aller à échec ;
                       fsi ;
                                fsi ; on pourra effectuer la mise à jour
                             fin ; l'état de chaque donnée est compatible avec les
                                   opérations que réalise le module.
```

```
Pour chaque donnée transmissible utilisée répéter
   si mode d'utilisation = entrée alors
      si type = consommable alors  NAC:=NAC-1 ;
                                   NAP:=NAP+1 ;
                                   EXEMPLAIRE:=tête de file d'attente ;
                           sinon
      si type = rémanent   alors NU.VERSION COURANTE:=NU.VERSION COURANTE +1 ;
                                 EXEMPLAIRE:=VERSION COURANTE ;
                           sinon
                EXEMPLAIRE:= donnée produite par la même tâche ;
                           fsi ; fsi ;
   si mode d'utilisation=sortie alors
      si type = consommable alors NAP:=NAP-1 ;
                           fsi ;
      EXEMPLAIRE:=mémoire libre :
                           fsi ;
   si mode d'utilisation = entrée/sortie alors
      si type = rémanent   alors MAJ:=.FAUX. ;
                           EXEMPLAIRE:=mémoire libre ;
                           (EXEMPLAIRE):=(VERSION COURANTE) ;
                           sinon EXEMPLAIRE:=mémoire libre ;
                                 (EXEMPLAIRE):=(donnée produite par la même tâche) ;
                           fsi ;
                                    fsi ;
                                fin ;
V(garde) ;  l'allocation des données au module est terminée, on peut lancer le module
fin ;
   échec : V(garde) ; on aboutit ici quand une donnée n'est pas disponible
                    ; il faut libérer le sémaphore garde pour autoriser d'autres
                      traitements
                    ; on reprendra l'examen plus tard.
fin ;
```

V.3.1.2. Données réparties sur différents sites

Lorsque les données transmissibles sont réparties sur le réseau, le lancement d'un module fait intervenir les gérants des différents sites possédant une donnée.

Ceux-ci doivent à leur tour entrer en section critique afin d'effectuer le traitement nécessaire, sans être interrompu par un traitement local ou par une autre requête.

Exemple :

Soit un module M1 implanté sur le site S1 et défini par :

Module : M1
entiers : A, B
entrées : A, B
...
fin

Supposons que les connexions de ce module fassent intervenir la donnée D1 implantée sur S1 et D2 sur S2. (A ←D1 ; B←D2).
Le gérant de S1 rentre en section critique et examine la table de connexion de M1. Il s'aperçoit alors qu'il a besoin d'une donnée présente sur S2 et en conséquence émet un message à destination du gérant de ce site afin de lui demander la donnée. Il doit préciser l'usage qu'il veut en faire de façon à ce que le gérant de S2 puisse le comparer à l'état de la donnée au moment où il reçoit la requête et que sa réponse en tienne compte. Deux cas sont alors à envisager :

a) Le gérant de S2 est déjà entrain de traiter un autre module, il se trouve dans une section critique. Il doit signaler cet état au gérant de S1 et ne pas tenir compte du message reçu. C'est le gérant de S1 qui doit garder l'initiative du traitement. A la réception du message venant de S2, il doit alors abandonner temporairement le traitement de M1 et quitter sa section critique.

b) Le gérant de S2 peut traiter la requête qui lui a été adressée : il rentre alors en section critique et examine la compatibilité entre l'état de D2 et son utilisation en entrée par M1. Il transmet le résultat de ce test au gérant de S1.
Sur réception du message émis par le gérant de S2 et compte tenu de l'état de la donnée D1, le gérant de S1 peut alors entreprendre deux types d'actions :

a) Envoyer un message au gérant de S2 pour lui signaler que l'état des données étant incompatible, il doit quitter sa section critique et abandonner le traitement en cours. Le gérant de S1 quitte alors également sa section critique.

b) Envoyer un message au gérant de S2 pour lui demander la transmission des données. Celui-ci dpit alors émettre vers le gérant de S1 la valeur des données et modifier leurs descripteurs en conséquence. Lorsque cette opération est réalisée, il peut quitter sa section critique. Le gérant de S1 reçoit alors les données, il doit initialiser les relais de M1 vers celles-ci avant de quitter sa section critique. Le module M1 peut alors être lancé.

L'algorithme suivant présente le traitement simplifié de lancement d'un module dans le cas où les données sont réparties sur le réseau.

entrée en section critique du gérant de S1
examen de la table de connexion du module
si les données sont réparties sur d'autres sites alors
élaboration et émission des messages vers les gérants de ces sites fsi

Sur S1 :

examen de la compatibilité entre l'état des données implantées sur S1 et la nature des opérations réalisées par le module.

sur chaque site différent de S1 concerné :

si le gérant est déjà en section critique alors
élaboration et transmission d'un message signalant cet état au gérant de S1

sinon
examen de la compatibilité entre l'état des données et les opérations réalisées par le module
élaboration et transmission d'un message de compte rendu au gérant de S1

fsi

examen de tous les compte rendus, y compris le résultat du test effectué par le gérant de S1
si incompatibilité alors
si les données sont réparties sur d'autres sites alors
élaboration d'un message aux gérants des autres sites fsi

Sur S1 :
sortie de la section critique sans modification de l'état des données

sur chaque site concerné différent de S1 :
sortie de la section critique sans avoir modifié l'état des données

fin du traitement.

sinon

si les données sont réparties sur le réseau alors

élaboration d'un message destiné aux gérants des autres sites
pour leur demander les données qui intéressent le module fsi

sur chaque site concerné différent de S1 :

émission vers S1 d'un message contenant la valeur de la version
courante des données utilisées
mise à jour des descripteurs de donnée

fin de section critique.

Sur S1 : réception des diverses données destinées au module
allocation de ces données au module
mise à jour des descripteurs de donnée
fin de section critique
lancement du module

fsi

V.3.1.3. Lancement d'une étape comportant plusieurs modules en parallèle

Le lancement d'une étape comportant plusieurs branches en parallèles doit être effectué de manière similaire au lancement d'un module unique. Toutes les données nécessaires aux différents modules doivent en effet être disponibles dès le lancement de l'étape. Si certains modules ne peuvent être lancés immédiatement parce que le calculateur qu'ils emploient n'est pas disponible, elles doivent être éventuellement conservées jusqu'à ce que leur exécution soit possible.

Exemple :

Considérons les modules M1 et M2 définis par :

```
Module : M1                    Module : M2
         entiers A, B                   entiers C, D
         entrées A, B                   entrée  C
         ....                           sortie  D
         fin                            fin
```

Ces modules appartiennent à une même étape de la tâche T dont l'enchaînement est :

T : M-(M1 // M2) - ...

L'opération de lancement de l'étape (M1 // M2) doit comporter un test unique de l'état des quatre données connectées à A, B, C et D.
Elle ne doit être poursuivie que si l'ensemble de ces quatre données est compatible avec les opérations réalisées par les modules M1 et M2. Supposons que A, B, C et D soient des données consommables, il faut que A, B et C aient été produites et qu'il existe une place disponible dans le tampon associé à la donnée D.
L'ordre effectif dans lequel sont ensuite lancés les modules M1 et M2 peut être quelconque ; il ne dépend que de la charge des calculateurs au moment du lancement de l'étape. Les exemplaires des données alloués ne peuvent en effet être modifiés par d'autres modules tant que M1 et M2 ne sont pas terminés.

V.3.2. Terminaison d'un module

Les opérations à réaliser lors de la fin d'exécution d'un module dépendent du fait qu'ils appartiennent à une étape comprenant plusieurs modules parallèles ou non. Si le module est le seul de son étape, il n'y a pas de problème, toutes les données produites par ce module doivent être validées, celles qui ont été consommées doivent impérativement disparaître, celles qui ont été consultées disparaissent éventuellement si elles sont périmées. Si au contraire, le module n'est pas le seul de son étape, il faut savoir s'il est le dernier module à se terminer ou s'il y en a encore d'autres en attente d'exécution ou actifs.

S'il est le dernier, il faut valider l'ensemble des données produites par les modules parallèles et faire disparaître les données périmées ou consommées. Cette opération se déroule de la même façon que dans le cas d'un module unique.

Si le module n'est pas le dernier, il n'y a rien de spécial à réaliser, il faut attendre que tous les modules se soient achevés.
C'est en interprétant les structures de contrôle de classe II que l'on peut savoir si un module est ou non le dernier de son étape (& V.1.).

V.3.2.1. Unique module d'une étape

Lors de la terminaison de l'unique module d'une étape, il faut valider du bloc toutes les données produites ou modifiées par ce module. Il faut en effet conserver l'intégrité des données produites et éviter de créer une situation où un module reçoit simultanément l'ancienne valeur d'une donnée D1 et la nouvelle valeur d'une donnée D2.

Dans ce but, la terminaison d'un module doit être traitée en section critique par le gérant des données transmissibles. De cette façon, on évite les intéractions entre les opérations de lancement et terminaison d'un module.

L'algorithme déroulé par le traitement de la terminaison d'un module est le suivant :

Algorithme utilisé lors de la terminaison d'un module.

```
P(garde) ;
pour chaque donnée transmissible utilisée répéter
    si mode d'utilisation = entrée alors
       si type = consommable alors chaîner EXEMPLAIRE en zone de mémoire libre ;
                              fsi ;
       si type = rémanent     alors NU.EXEMPLAIRE:=NU.EXEMPLAIRE-1 ;
          si (NU.EXEMPLAIRE=0) & ETAT.EXEMPLAIRE alors
                                    chaîner EXEMPLAIRE en zone de mémoire libre ;
                              fsi ;
                                    fsi ;
    si mode d'utilisation = sortie alors
       si type = consommable alors chaîner EXEMPLAIRE en queue de file d'attente ;
                                    NAC:=NAC + 1 ;
                              fsi ;
       si type = rémanent alors
          si PROD alors
            si NU.VERSION COURANTE=0 alors
                                       chaîner VERSION COURANTE en zone de mémoire
                                       libre ;
                                       sinon   ETAT.VERSION COURANTE:=.FAUX. ;
                                       fsi ;
```

```
                fsi ;
        VERSION COURANTE:=EXEMPLAIRE ;
        ETAT.VERSION COURANTE:=.VRAI. ;
        NU.VERSION COURANTE:=0 ;
        PROD:=.VRAI.;
                    fsi ;
                        fsi ;
    si mode d'utilisation=entrée/sortie alors
      si type=rémanent alors
        si NU.VERSION COURANTE=0 alors chaîner VERSION COURANTE en zone de mémoire
                                        libre ;
                                 sinon ETAT.VERSION COURANTE:=.FAUX.;
                                 fsi ;
        VERSION COURANTE:=EXEMPLAIRE ;
        ETAT.VERSION COURANTE:=.VRAI. ;
        NU.VERSION COURANTE:=0 ;
                    sinon  chaîner l'ancienne version de la donnée en zone de
                           mémoire libre ;
                    fsi ;
                        fin ;
V(garde) ;
            fin ;
```

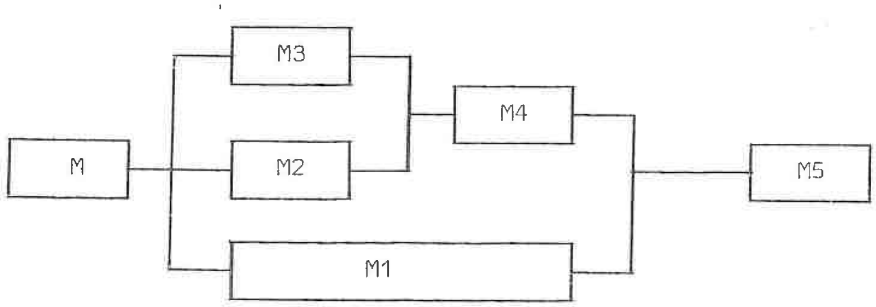
V.3.2.2. Etape comprenant plusieurs modules en parallèle

Seule, la terminaison du dernier module d'une étape en comportant plusieurs en parallèle nécessite un traitement. Celui-ci est analogue au traitement réalisé à la fin d'un module unique. Il porte sur l'ensemble des données utilisées par les modules de l'étape.

Exemple : Soit la tâche fonctionnelle T définie par :

tâche T M-(M1 //((M2 // M3)-M4))-M5

Elle correspond au schéma :



Les traitements de fin d'étape doivent être réalisés lorsque :

M se termine
M2 et M3 sont tous deux terminés
M1 et M4 sont tous deux terminés
M5 se termine.

Ils sont indépendants de l'ordre dans lequel les modules se terminent dans une étape (M2 peut se terminer avant ou après M3, de même pour M1 et M4).

V.4. LES CONFLITS D'ACCES AUX DONNEES TRANSMISSIBLES

Au paragraphe précédent, nous avons fait apparaître que l'allocation des données transmissibles à un module peut se solder par un échec. Il y a deux causes possibles à cet échec :

a) Le module que l'on veut lancer emploie des données qui sont effectivement dans un état tel que celui-ci ne peut s'exécuter : donnée non produite, déjà utilisée, tampon saturé. Il faudra donc réexaminer le lancement de ce module chaque fois que l'état des données sera modifié : fin d'exécution d'un module producteur, lancement d'un module consommateur, ...

b) Les données demandées par le module sont disponibles et auraient pu être allouées si les gérants avaient été en état de le faire. Ceux-ci étaient cependant occupés au lancement d'autres modules et le traitement a du être abandonné. Dès que cette condition cessera, il faudra réexaminer le lancement du module.

Cependant, un cas particulièrement désagréable est susceptible de se produire : celui où l'interprétation des structures de contrôle de deux tâches distinctes conduit au même moment à lancer sur deux sites différents S1 et S2, deux modules M1 et M2. Aucune difficulté ne se présente si l'un des modules au moins n'utilise aucune donnée résidant sur l'autre site. Dans le cas contraire, chaque gérant va vouloir allouer les données au module implanté sur son site. Il va donc entrer en section critique, élaborer un message destiné au gérant de l'autre site et traiter l'allocation des données résidant sur son site. A la réception du message provenant de l'autre site, il enverra une réponse négative puisqu'il est déjà en section critique.
A la réception d'une réponse négative, en provenance de l'autre gérant, chacun abandonne le traitement en cours et quitte sa section critique.
Si les différents gérants possèdent strictement les mêmes propriétés, le problème se reposera de façon identique lors de la tentative suivante. La solution à ce problème consiste à privilégier l'un des gérants et à introduire ainsi une dissymétrie.

Plusieurs méthodes sont possibles :

a) Affecter à chaque gérant une temporisation, entre deux traitements du même module, différente. Lors de la tentative suivante, le gérant qui possède la temporisation la plus courte a de fortes chances de voir sa requête acceptée et traitée.

b) Introduire des priorités différentes aux traitements réalisés par les différents gérants. Celles-ci peuvent être fixes ou fonction de la tâche à traiter. Dans cette hypothèse, un gérant en train d'effectuer un traitement de priorité P1 devrait le suspendre au profit d'une requête pour un traitement de priorité P2, si P1 est inférieur à P2. Il n'y a aucune difficulté si les données utilisées sont distinctes, le traitement interrompu pourra reprendre lorsque la requête aura été traitée.

Si les données utilisées sont identiques et que l'on est en train de vérifier la compatibilité entre le traitement réalisé par le module et l'étude des données, ce dernier n'a pas encore été modifié et le traitement en cours peut être abandonné sans aucune gène. Si par contre, on en est à la phase d'allocation des données et que par conséquent l'état d'un certain nombre d'entre elles a été modifié, le traitement doit être poursuivi jusqu'à son terme. La requête, bien que concernant un traitement plus prioritaire, est alors abandonnée sans avoir été traitée.

V.5. CONCLUSIONS

Les concepts présentés dans ce chapitre n'ont malheureusement pas pu être mis en pratique à ce jour, car l'état d'avancement du projet SYGARE ne permet pas encore l'exécution d'une tâche fonctionnelle répartie utilisant des données implantées sur le réseau. Nous pensons cependant qu'aucune difficulté insurmontable ne devrait survenir dans leur mise en pratique. L'application de ces principes reste cependant la seule possibilité permettant de vérifier leur bien-fondé.

CHAPITRE VI

SYNCHRONISATION ENTRE LE DISPOSITIF DE COMMANDE ET LE PROCEDE

CHAPITRE VI

SYNCHRONISATION ENTRE LE DISPOSITIF DE COMMANDE ET LE PROCEDE

La description des synchronisations entre les différentes tâches fonctionnelles qui permettent de commander l'application ou entre les tâches fonctionnelles et le procédé à piloter est réalisée de façon extérieure aux modules ou aux tâches. Cette méthode offre des avantages identiques à ceux présentés au chapitre IV pour les structures de contrôle de classe II. Celles-ci sont utilisées pour décrire les synchronisations entre les différents modules appartenant à une même tâche fonctionnelle et elles interviennent indépendamment du contenu des modules.

De façon similaire, les structures de contrôle de classe II permettant de décrire les synchronisations entre les tâches fonctionnelles et d'associer les actions qu'elles représentent à leurs conditions de lancement, d'arrêt, de reprise ... Celles-ci peuvent dépendre :

- soit de l'état du procédé à piloter :
  - . fin d'échange sur un périphérique,
  - . prise de mesure,
  - . arrivée d'une interruption externe ...

- soit de l'état du dispositif de commande :
  - . terminaison d'une tâche fonctionnelle,
  - . anomalie dans le déroulement d'une tâche,
  - . panne d'un calculateur, du réseau ...

L'apparition de l'une ou de plusieurs de ces conditions peut être considérée comme un événement requérant un certain traitement, soit parce que cet événement est normal et attendu, soit parce qu'il traduit une anomalie qu'il faut corriger. Il faut donc permettre d'une part, la description des conditions d'apparition de tels événements et d'autre part leur associer les structures de contrôle de classe III employées pour la synchronisation des tâches fonctionnelles.

VI.1. LE CONCEPT D'EVENEMENT

L'évolution du procédé à commander ou celle de son dispositif de commande

transformation en délais réévalués à chaque période de l'horloge permet le positionnement d'un événement lorsque le délai expire. Les structures de contrôle de classe III qui l'utilisent peuvent alors être exécutées.

*Les interruptions matérielles peuvent être provoquées par un périphérique et sont en général traitées par le système de gestion des entrées/sorties. Elles peuvent également être considérées comme un appel externe sur lequel une tâche fonctionnelle est synchronisée. L'association événement - interruption matérielle est réalisée par l'utilisation des structures de contrôle de classe III.

*La gestion des entrées/sorties peut également être la source d'événements traduisant l'évolution du procédé. Ils peuvent être causés par le traitement associé à un échange ou par l'apparition d'un défaut sur le périphérique. En fonction du mode d'échange demandé, ils peuvent se produire : (chapitre VII)

- lorsque l'échange se termine,
- lorsque la mesure réalisée s'écarte d'une zone de tolérance (celle-ci peut être fixe ou dépendre de la mesure précédente),
- lorsque le périphérique est en défaut.

L'association de structures de contrôle de classe III, à de tels événements, permet de synchroniser une tâche fonctionnelle sur la gestion de périphérique de type classique ou particuliers à la commande de procédé.

VI.1.1.2. Les événements liés au logiciel de commande

Les événements liés au logiciel de commande permettent de prendre en compte son évolution. Le lancement et la terminaison d'une tâche fonctionnelle peuvent être considérés comme des points de synchronisation caractéristiques définissant l'état du système. L'association de tels événements et de délais permet de s'assurer du bon fonctionnement du logiciel ou d'entreprendre des actions correctives lorsqu'une anomalie est constatée.

Il est en effet possible qu'à la suite d'erreurs de conception, certaines tâches restent bloquées indéfiniment en attente de données transmissibles produites par une tâche qui ne sera jamais activée, bouclent indéfiniment, ou ne puissent pas être exécutées à temps car le calculateur sur lequel elles sont implantées est déjà utilisé.
L'action prévue doit alors être abandonnée ou remplacée par une autre, ce qui traduit le concept d'échéance ("Deadline") primordial dans les systèmes temps réel (HOLT 71), (RIVEST 76).

peut être perçue comme un "événement" sur lequel on désire synchroniser certaines actions.

Le concept d'événement a déjà été largement utilisé par divers systèmes d'exploitation et par des langages de programmation de haut niveau. Ceux-ci effectuent cependant une distinction entre les événements logiciels engendrés et traités par programme et les interruptions externes, associées à un signal matériel. Dans le langage HALS (FYLSTRA 75) développé par la NASA, la notion d'événement est utilisée comme un outil de synchronisation exclusivement logiciel. Il en est de même dans le langage LTR (PARAYRE 75) ou dans GAELIC (LE CALVEZ 77) avec l'instruction SYNCHRO-PROCOL T2000 (STERIA) permet quand à lui l'association statique d'une interruption matérielle à un événement logiciel. Ce lien peut être créé au moyen d'une description adéquate lors de la génération du système mais ne peut être modifié dynamiquement.

Le système d'exploitation préconisé par P. Ladet distingue la description des événements de leur utilisation et permet la création dynamique de liens événements-action. Lorsqu'aucun lien n'existe, les différentes occurences de l'événement mémorisées peuvent être utilisées par la suite dans une structure de contrôle faisant référence aux événements passés (LADET 77). Dans SYGARE, nous avons voulu supprimer cette différence réalisée entre les événement associés à un support matériel et les événements positionnés par le logiciel que l'on peut qualifier de "libre". Les événements peuvent alors être utilisés sans distinction pour caractériser un état de l'ensemble dispositif de commande - procédé.

VI.1.1. Les causes d'événements

Les causes des événements traités dans SYGARE peuvent être liées à l'état du procédé ou à celui du dispositif de commande. Nous distinguerons donc les deux cas.

VI.1.1.1. Les événements causés par le procédé

Les événements causés par l'évolution du procédé peuvent avoir trois origines :

- l'évolution du temps,
- l'arrivée d'une interruption matérielle,
- la gestion d'un périphérique.

*La prise en compte de l'évolution du temps par le système d'exploitation permet de gérer les relations entre les tâches fonctionnelles de l'application et le temps. Celle-ci peuvent s'exprimer sous forme d'heure absolue ou de délai. Leur

Il faut également introduire dans les événements liés au logiciel de commande la défaillance et la remise en état des moyens matériels qu'il utilise : calculateur ou lien du réseau. Ceux-ci sont en effet des éléments fondamentaux dont il faut tenir compte lors de la conception d'une application.

VI.1.2. Description d'un événement

La description d'un événement doit permettre de prendre en compte toutes les sources possibles d'évolution du dispositif de commande et du procédé. Elle doit pouvoir être introduite lors de la définition de l'application et modifiée en ligne par l'intermédiaire du dialogue avec l'ingénieur responsable de l'application.

Le langage de description utilisé emploie un certain nombre de mots clefs qui permettent d'introduire les caractéristiques de chaque événements que l'on désire définir.

Il utilise un codage symbolique des noms de périphériques et de sites du réseau et permet de faire référence à d'autres événements déjà définis afin de réaliser un codage plus fin, tel que l'arrivée d'un événement evt 1 avant celle de evt 2 ...

Le mot clef "vide" permet de supprimer la description préalablement définie d'un événement sans pour autant supprimer les liens qui y faisaient référence. On peut ainsi figer temporairement l'exécution de certaines structures de contrôle ou redéfinir de façon différente, un événement tout en conservant les structures de contrôle qui lui étaient associées.

Une extension de la description des événements reste toujours possible. Celle-ci devrait permettre une souplesse encore plus grande. Nous pensons même qu'elle pourrait être laissée au gré de l'utilisateur si on lui fournit les éléments de base permettant la description d'autres types d'événements.

Le schéma suivant (fig.35) précise la manière dont s'effectue la description d'un événement :

- les expressions soulignées sont des mots clef
- les expressions entre crochet sont optionnelles
- les expressions entre accolades représentent différentes options.

événement <evt> : =

{

vide

[non][<n1>] occurence(s) de <evt 1> | {avant / après} { <evt 2> / <heure> / <délai> } |

message opérateur : <message> [périphérique <p1>][site <s1>]

<heure>

interruption [niveau <n1>] [S.niveau <n2>] site <S1>

défaut { site <S1> / périphérique <p1> [site <s1>][mod d'état = <défaut>] / lien entre <s1> et <s2> }

fin d'échange sur périphérique <p1> [compte rendu <cr>]

[non] { lancement / terminaison } de <t1> {avant / après} { <evt> / <heure> / <délai> }

}

<n> ::= nombre entier positif

<heure>::= heures/minutes/secondes

<délai>::= heures/minutes/secondes |top de base| autre unité

<t1> ::= nom de tâche fonctionnelle

<s1> ::= nom de site

<p1> ::= nom de périphérique

<défaut>::= code du défaut de périphérique

<cr> ::= compte rendu d'échange

Fig.35 Description d'un événement.

VI.2. LES STRUCTURES DE CONTROLE DE CLASSE III

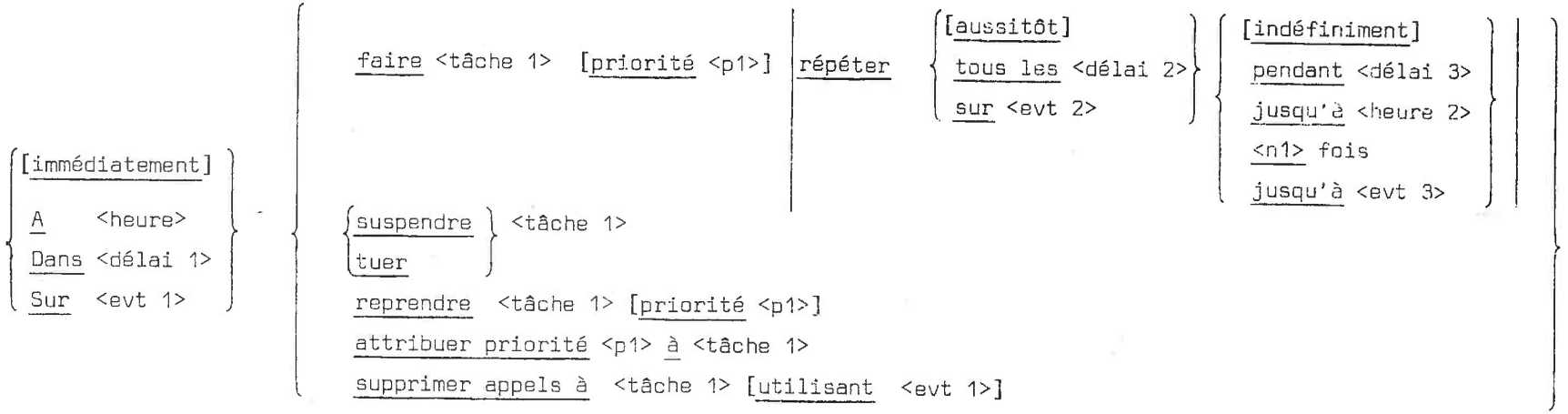
Les structures de contrôle de classe III que nous avons développé dans le projet SYGARE présentent les mêmes fonctions que les requêtes ou les commandes de type temps réel des systèmes d'exploitation actuels (MPES 77). Leur originalité réside dans le fait qu'elles n'interviennent pas directement dans le langage d'écriture des modules mais à l'extérieur des tâches fonctionnelles, ce qui permet de les modifier en ligne sans qu'il soit nécessaire de recompiler les modules ou de reconstruire les tâches fonctionnelles.

Elles peuvent être introduites lors de la description de l'application ou en ligne par le dialogue avec l'ingénieur responsable de l'application. Elles peuvent utiliser les événements, tels que nous les avons présentés au paragraphe précédent, afin de synchroniser les tâches fonctionnelles entre elles ou avec l'application. Dans le système HALS (FYLSTRA 75), seuls les événements logiciels peuvent être utilisés dans ce but et le lancement de la tâche PROCESS 1 sur l'événement EVENT 1 avec la priorité N1 s'écrit :

SCHEDULE PROCESS1 ON EVENT1 PRIORITY N1.

Le schéma suivant (fig.36) précise la syntaxe des structures de contrôle de classe III. Les conventions adoptées sont :

- les expressions soulignées sont des mots clefs
- les expressions entre crochets sont optionnelles
- les accolades indiquent différentes options eéquivalentes.



<heure> ::= heure exprimée en heures/minutes/secondes

<délai 1> ::= heure/minutes/secondes |top de base| autre unité

<evt 1> ::= événement

<tâche 1> ::= nom de tâche fonctionnelle

<p1 > ::= numéro de priorité

<n1> ::= nombre de répétitions.

Fig.36 Syntaxe des structures de contrôle de classe III.

VI.3. L'INTERPRETATION DES STRUCTURES DE CONTROLE DE CLASSE III DANS LE SYSTEME D'EXPLOITATION

Le système de gestion des événements d'une part, l'interpréteur des structures de contrôle de classe III d'autre part, sont deux sous-ensembles importants du système d'exploitation. Ils sont intimement liés au moyen de gestion des communications sur le réseau et au système de séquencement des modules.

La définition d'un événement implique la communication de ses caractéristiques aux calculateurs concernés par son évaluation. Ceux-ci doivent se communiquer l'arrivée d'une condition faisant évoluer l'état de l'événement afin de permettre l'étude de la conjonction de plusieurs événements.

L'occurence d'un événement relié à une structure de contrôle de classe III doit être signalée à l'interpréteur qui doit ensuite entreprendre les actions nécessaires (activation, suspension, reprise, modification de la priorité d'une tâche fonctionnelle). Cette opération doit être réalisée au moyen de noyau de communication du réseau, elle intéresse directement le système d'ordonnancement des modules.

VI.4. CONCLUSION

La description à un niveau élevé des synchronisations de tâches fonctionnelles permet de séparer nettement la description des opérations à réaliser des conditions sur lesquelles on désire les entreprendre. Cette possibilité est à rapprocher de l'ensemble des facultés de conception modulaire de l'application que nous avons voulu développer avec le projet SYGARE.

Bien qu'elles puissent encore être développées afin de couvrir un plus grand éventail de conditions de synchronisation, nous pensons qu'elles assurent un compromis acceptable avec les performances des calculateurs qui supportent la maquette de SYGARE. Leur développement reste cependant possible et peut être envisagé dans le cadre d'une extension du réseau SYGARE en utilisant du matériel plus puissant : systèmestemps réel utilisés en gestion ...

CHAPITRE VII

GESTION DES ENTREES-SORTIES

CHAPITRE VII

GESTION DES ENTREES-SORTIES

Les principaux points que nous avons voulu développer lorsque nous avons défini les caractéristiques essentielles de SYGARE ont trait aux problèmes posés par la répartition de l'application sur le réseau de calculateurs et par la décomposition modulaire de cette dernière.

Pour que la modularité apparaisse au concepteur d'une application comme une facilité supplémentaire mise à sa disposition et non comme une contrainte dûe aux limitations du matériel ou du logiciel d'exploitation, il faut qu'elle s'appuie sur des critères concrets tels que l'unité d'une action à entreprendre.

Les chapitres précédents nous ont permis de montrer qu'un module renfermait le code d'une action, à la condition que celle-ci puisse être réalisée entièrement sur l'un des calculateurs du réseau. Cependant, cette restriction ne doit pas porter sur l'utilisation des organes d'entrée/sortie. Il est en effet indispensable que leur répartition sur le réseau de calculateurs n'ait aucune influence sur la décomposition d'une application ou sur sa répartition. L'inverse doit également être vérifié. Ceci permet d'adopter une répartition des périphériques judicieuse sans qu'il soit nécessaire d'éclater une action en autant de modules que de périphériques employés et de concevoir ainsi la programmation d'une action indépendamment des organes d'entrée/sortie qu'elle met en oeuvre. On accède ainsi à l'indépendance totale des modules vis à vis de leur contexte d'exécution.

L'introduction de la notion de données transmissibles (chapitre II) et d'un gérant de ces données (chapitre III) permet la conception d'un module indépendamment des données qu'il utilise, de la même façon, l'introduction d'une hiérarchie de structures de contrôle permet de séparer la description d'une tâche fonctionnelle (chapitre IV) de ses conditions d'exécution (chapitre V).

Dans ce chapitre, consacré au système de gestion des entrées/sorties, nous nous proposons de montrer comment son introduction permet d'atteindre l'indépendance totale des modules vis à vis de leur lien d'implantation, de leur

contexte d'exécution et des actions réalisées sur les périphériques.

VII.1. GENERALITES

La programmation des entrées-sorties est considérée comme une opération délicate, elle est en général réservée au système d'exploitation.
Cela est dû principalement au fait qu'elle nécessite une bonne connaissance de la machine utilisée :

- unité centrale,
- mécanisme d'interruption,
- interface avec le périphérique,
- processeur de gestion des entrées/sorties éventuel ...

La moindre erreur de programmation peut conduire à une dégradation des performances de la machine, voire à son blocage complet (appel permanent d'un périphérique ...). Ceci n'est pas admissible dans le cadre d'applications industrielles nécessitant un degré de fiabilité élevé ou lorsque plusieurs processus différents se partagent une même machine soit directement, soit par l'intermédiaire d'un réseau.

VII.1.1. Les protections contre les erreurs

Sur les microprocesseurs, les coupleurs de périphériques sont branchés sur le même bus que la mémoire. Ceci permet de les employer exactement comme un mot-mémoire, que l'on lit ou que l'on écrit suivant le sens de l'échange désiré. Les microprocesseurs sont rarement multiprogrammés ou utilisés pour la mise au point de programmes. Ceci explique qu'aucune protection ne soit assurée sur l'utilisation des organes d'entrée-sortie : si un programme comporte une erreur, il se bloque, mais il est le seul à en subir les conséquences.

Les calculateurs actuels, au contraire, sont souvent multiprogrammés. Ceci explique que l'on ait été conduit à développer un mécanisme de protection plus sophistiqué. Pour que les programmes en cours de mise au point, ou réputés "non fiables" n'aillent pas altérer les données ou d'autres programmes, ils sont exécutés dans un mode "esclave". Dans ce mode, ils peuvent être plus finement surveillés et certaines instructions leurs sont interdites : instructions d'entrée/sortie, de manipulation de sémaphores ... En cas d'anomalie détectée, lors de l'exécution d'un programme en mode esclave, celui-ci est interrompu, une tâche de traitement d'alarme est activée.

Les programmes d'intérêt général et le système d'exploitation, s'exécutant en mode "maître", ont le droit d'effectuer ces instructions privilégiées. Aucun contrôle n'est réalisé par la machine car ces programmes sont considérés comme "fiables".
Une instruction d'appel au moniteur permet aux programmes s'exécutant en mode "esclave" de demander à ce dernier d'effectuer pour eux certains traitements qui leur sont normalement interdits. Le système d'exploitation doit avoir été conçu pour recevoir et traiter de telles requêtes. Un compte-rendu d'exécution est fourni au programme "esclave" en même temps que le moniteur lui redonne le contrôle.
Rien ne s'oppose, en principe, à ce que tous les programmes implantés sur la même machine s'exécutent en mode maître et aient tous les mêmes droits et les mêmes possibilités. Une protection plus importante est cependant souhaitable afin d'accroître la fiabilité du système. Il est donc nécessaire d'introduire un système d'exploitation s'exécutant en mode maître, et dans ce système, une partie doit être plus spécialement consacrée à la gestion des entrées/sorties.
Les différents programmes de l'application qui auront une opération d'entrée/sortie à effectuer pourront s'adresser à lui par l'intermédiaire de requêtes au superviseur.

VII.1.2. L'utilisation des organes d'entrée/sortie

Les périphériques utilisables par une application conçue pour SYGARE, peuvent être répartis sur l'ensemble du réseau géré par SYGARE. L'introduction d'un système de gestion des organes d'entrée/sortie réparti, permet de ne pas être géné dans la conception de l'application par la distribution des modules. Son rôle principal est donc de permettre à n'importe quel module, quel que soit son lieu d'implantation, d'accéder à n'importe quel périphérique de SYGARE. Ceux-ci sont donc en quelque sorte banalisés. L'affectation d'un périphérique à un module est réalisée de façon externe au module qui travaille sur un périphérique symbolique.

Les avantages que présente un tel système sont les suivants :

- le découpage de l'application en modules est indépendante de la distribution des périphériques sur le réseau,

- un même module peut à n'importe quel moment utiliser simultanément ou successivement, des périphériques implantés sur des sites distincts. On évite de cette façon, la prolifération des structures de contrôle de classe II que rend nécessaire un découpage en modules très petits et on gagne en lisibilité de l'application,

- la répartition des modules étant indépendante de celle des périphériques on peut déplacer un module sans qu'il soit nécessaire de tenir compte des

périphériques dont il a besoin pour s'exécuter. Ceci se passe de la même façon que pour l'utilisation de données transmissibles par les modules,

- l'interface entre un module et le système de gestion des entrées/sorties peut être standardisé quelque soit le type de périphérique employé et son implantation. De cette façon, on peut modifier l'allocation des périphériques aux modules sans qu'il soit nécessaire de modifier le texte source du module. Cette propriété est intéressante car elle offre la possibilité de reconfigurer l'application en cas de surcharge ou de défaillance d'un périphérique ou d'un réseau, lorsqu'il s'agit de périphériques distants. Elle permet en outre de tester les modules en phase de mise au point en utilisant des périphériques de types différents de ceux qui seront employés en exploitation. Nous pensons même que l'utilisation de fichiers-disque devrait être permise pour simuler les périphériques à cadence rapide. Cette possibilité reste cependant dans le domaine des extensions de SYGARE.

- La gestion des périphériques étant réalisée en standard par le système d'exploitation, on est en droit d'espérer une diminution du nombre d'erreurs de programmation ou d'utilisation. L'ensemble de l'application gagne donc en fiabilité.

Face aux avantages, les inconvénients d'un tel système peuvent sembler mineurs. Il ne faut cependant pas les négliger :

. Le système de gestion des entrées/sorties doit connaître tous les périphériques du réseau et doit permettre un certain nombre d'opérations standard sur ces périphériques : il risque donc d'occuper un volume important.

. L'accès direct aux périphériques doit être interdit aux modules même s'ils sont implantés sur le site pour être réservé au système d'exploitation : le temps de traitement des opérations d'entrée/sortie risque donc d'être fortement allongé. Il nécessite en effet, dans tous les cas, une requête au moniteur et le traitement de cette requête en plus des opérations normalement requises par l'utilisation du periphérique.

VII.2. LES FONCTIONS D'UN SYSTEME DE GESTION DES ENTREES/SORTIES

Comme nous l'avons montré au paragraphe précédent, le système de gestion des entrées/sorties ne constitue en fait que l'un des éléments du système d'exploitation. Si l'on veut qu'il soit particulièrement intéressant, utile et réellement utilisé, il faut qu'il possède un certain nombre de propriétés, qu'il assume

un certain nombre de fonctions complémentaires. Celles-ci, que nous allons développer dans ce paragraphe ne tiennent aucun compte du fait que SYGARE est en réalité implanté sur un réseau de calculateurs hétérogènes. Elles peuvent être regroupées en quatre grandes catégories :

. Le traitement des appels-moniteur et le dialogue avec les programmes utilisant les services de ce système,

. La gestion des différents périphériques,

. La possibilité offerte à l'utilisateur de définir certains traitements spéciaux qui devront être effectués sur le chemin des données entre le module et le périphérique,

. La gestion du parc de périphériques et la possibilité de modifier dynamiquement l'allocation des périphériques aux différents programmes.

VII.2.1. Le dialogue avec le système de gestion des entrées/sorties

Les programmes peuvent dialoguer avec le système de gestion des entrées/sorties par l'intermédiaire de requêtes adressées au système d'exploitation (fig. 37). Celles-ci recueillies par un interface adéquat doivent permettre :

- de demander à connaître l'état d'un périphérique symbolique
- d'initialiser un échange,
- de tester le déroulement d'un échange,
- d'attendre la fin d'un échange,
- de forcer la fin d'un échange,
- d'exécuter certaines fonctions spéciales sur le périphérique.

Le test de l'état du périphérique permet de savoir si un périphérique physique a été associé au nom symbolique utilisé par le module pour le désigner, si celui-ci est libre ou s'il est également partagé par d'autres modules.

. L'initialisation d'un échange permet de préciser les caractéristiques de l'échange souhaité :

- le sens de l'échange
- le type de retour demandé (immédiat après initialisation, après un premier transfert, en fin d'échange, ...)
- les procédures de reprises à utiliser en cas de défaut pendant le transfert et le nombre de reprises maximum toléré (reprise de l'échange complet,

reprise en amont, abandon de l'échange),

- l'adresse du (ou des) tampon (s) contenant (ou qui recevront les informations échangées

- la taille des informations à échanger ou l'adresse d'une table contenant les codes d'arrêt.

- le nom de la procédure à lancer lorsque l'un des tampons est rempli ou vidé (cas de plusieurs tampons utilisés en bascule pour une lecture ou une écriture s'exécutant en continu).

. Lorsqu'un échange a été initialisé avec retour immédiat, le programme ayant demandé l'accomplissement de ce transfert doit pouvoir s'informer de l'état d'avancement de cet échange par une requête au moniteur.

. L'attente de fin d'échange peut également demandée si le programme ne peut effectuer aucun traitement tant que celui-ci n'est pas terminé ; on évite ainsi une attente active bouclant sur le test de bon déroulement de l'échange.

. L'abandon d'un échange peut être demandé par le programme appelant lorsque celui-ci devient sans objet ou tarde trop à se terminer.

. Certaines fonctions spéciales sont également nécessaires. Elles permettent de traiter les périphériques particulières :

- mise en ou hors tension des périphériques,

- exécution d'un saut de page sur imprimante, ou effacement d'un écran,

- changement de code d'un périphérique, passage en mode graphique,

- programmation du gain, de la résolution et de la période d'échantillonnage d'un organe analogique,

- programmation de la fréquence d'une horloge,

- remise à zéro d'un compteur, prépositionnement à une certaine valeur (utilisation en décompteur),

- armement d'une temporisation (chien de garde), ...

Le système de gestion des entrées/sorties doit également fournir un compte-rendu de l'exécution de chaque requête qui lui était adressée. Celui-ci est transmis à l'appelant au moment où il reprend le contrôle de l'exécution. Il peut alors être testé et permet de savoir comment s'est déroulé le traitement de la requête.

Lorsque la réquête est mal formulée, le compte-rendu permet de connaître le paramètre incorrect. S'il s'agit d'une erreur grave ou d'une tâche d'importance capitale, le compte-rendu d'erreurs pourra également être imprimé sur le périphérique de service afin de prevenir l'opérateur de l'anomalie constatée.

Lorsque la requête est formulée correctement, elle peut cependant avoir fait l'objet d'un refus, soit parce que l'accès au périphérique est interdit, soit parce que le périphérique est inconnu : (nom symbolique inconnu ou non associé à un périphérique physique).
Une requête formulée correctement portant sur l'initialisation, le test ou l'attente de la fin d'un échange peut entraîner l'un des compte-rendus suivants :

- échange en cours (initialisation ou test)
- échange terminé normalement (initialisation, test, attente) plus taille de l'information échangée
- échange terminé anormalement (initialisation, test, attente) plus cause d'abandon.

L'exécution d'une fonction spéciale conduit à un compte-rendu spécifiant si celle-ci s'est déroulée correctement ou non. Celui-ci peut en outre être plus détaillé dans le cas de périphériques non standards, dans ce cas, il dépend essentiellement du type de la requête qui a été adressée.

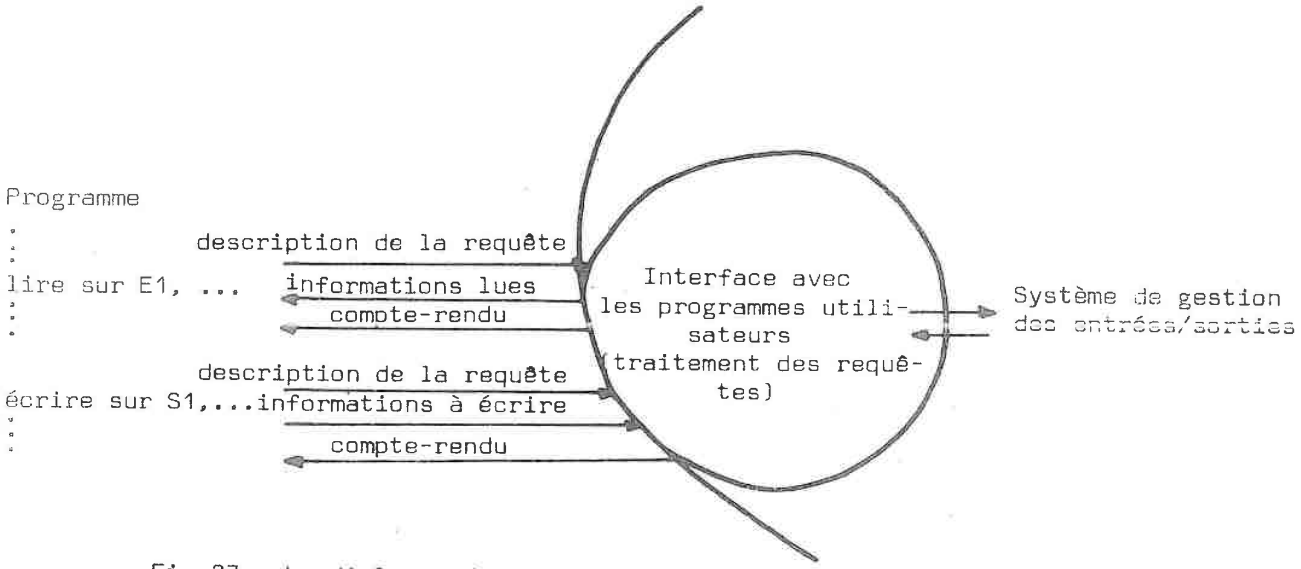


Fig.37. Le dialogue des programmes avec le système de gestion des entrées/sorties.

VII.2.2. La gestion des différents périphériques

Les différents périphériques sont essentiellement connus du système d'exploitation par des programmes de gestion spécifiques à chaque type de matériel parfois appelés Handler ou Driver.
Ces programmes doivent s'exécuter sur la même machine que celle sur laquelle le périphérique est connecté. Ils doivent permettre de réaliser un certain nombre d'opérations dites "standard" sur les périphériques. Suivant leur type, ces opérations seront effectuées soit sur la demende d'un programme-utilisateur, soit pour le système de gestion des entrées/sorties. L'intérêt de leur standardisation est évident : le système d'exploitation n'a pas à connaître les particularités de chaque périphérique et peut cependant faire réaliser n'importe quelle opération par le programme de gestion du périphérique (fig.38) :

- initialisation du périphérique,
- test de l'état du périphérique physique,
- initialisation d'un échange (sur demande d'un programme),
- entretien de l'échange en cours (échange se déroulant en mode programmé par test d'état ou sur interruption),
- traitement de la fin d'un échange :
  . mise au repos du périphérique,
  . élaboration d'un compte-rendu destiné au programme ayant demandé l'initialisation de l'échange
- traitement des défauts survenant en cours ou lors d'échange
- abandon d'échange et réinitialisation du périphérique (sur demande d'un programme).

D'autres fonctions sont nécessaires pour traiter des périphériques non standards. Le système d'exploitation n'a pas à les utiliser, ni même à les connaître, il ne fait que transmettre les demandes d'exécution provenant des programmes utilisant les périphériques. Leur développement dépend des types de matériels employés et de la façon dont ils sont utilisés et doit être laissé au gré du concepteur de l'application. Lui seul est en effet à même de définir les besoins qu'il ressent au niveau des programmes-utilisateurs.

L'adjonction de périphériques se traduit par l'intégration de nouveaux programmes de gestion d'entrées/sorties et par la mise à jour des tables du système d'entrées/sorties. Cette "édition de liens" est en général réalisée lors de la génération du système mais peut également être dynamique. Ceci permet une utilisation plus rationnelle de l'espace mémoire de la machine et

reconfiguration dynamique lors des défaillances de périphériques :

- alimentation en mémoire des programmes de gestion de périphériques effectivement utilisés

- élimination des programmes concernant des périphériques absents, en panne, ou momentanément inutilisés.

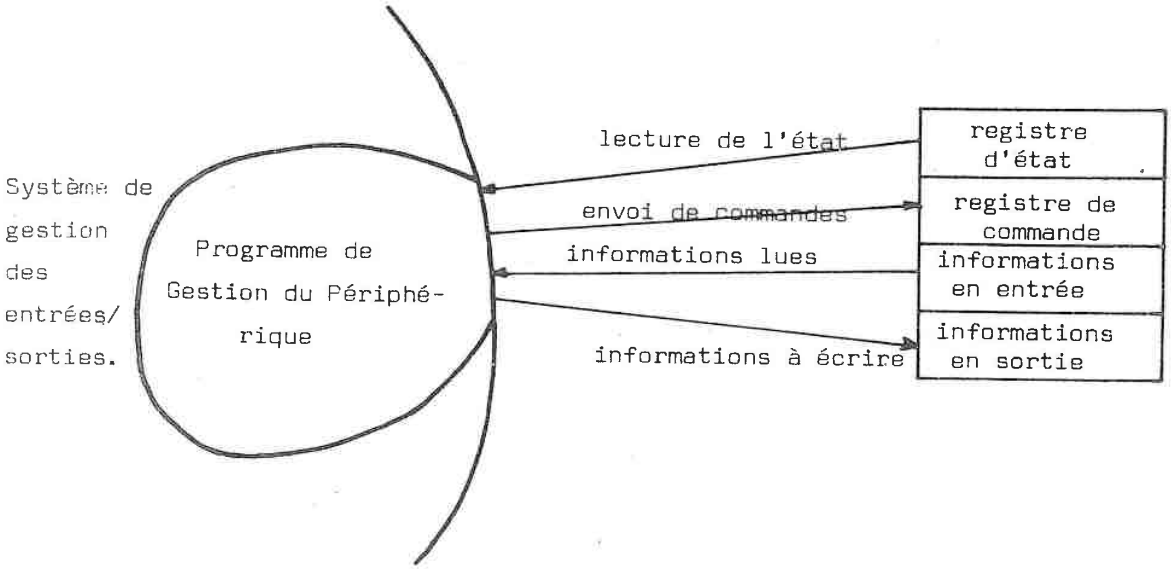


Fig. 38. Les échanges entre le périphérique et son gérant.

VII.2.3. Les autres fonctions d'un moniteur d'entrées/sorties

Dans la majeure partie des systèmes d'exploitation actuels, le moniteur d'entrées/sorties peut être décomposé en trois parties distinctes. Aux deux que nous venons de présenter dans les paragraphes précédents :

- interface avec les programmes utilisateurs,
- programmes de gestion des entrées/sorties,

il convient d'ajouter un noyau, qui outre un certain nombre de fonctions standards, comprend une procédure autorisant l'utilisation de noms symboliques pour désigner les périphériques dans les programmes utilisateurs.
L'affectation d'un nom symbolique qui a un périphérique physique, est réalisée à l'extérieur des programmes et doit pouvoir être modifiée en ligne par le

dialogue-opérateur.

Ceci permet d'aiguiller le flot des informations émises par les modules sur les différents périphériques disponibles, et de procéder à une reconfiguration dynamique en cas de défaillance ou de surcharge d'un périphérique.

Cette fonction est particulièrement intéressante de ce point de vue, et elle peut être réalisée à peu de frais car il s'agit en fait de construire, de modifier et de consulter une table de correspondance entre les noms symboliques et les périphériques physiques (fig.39).

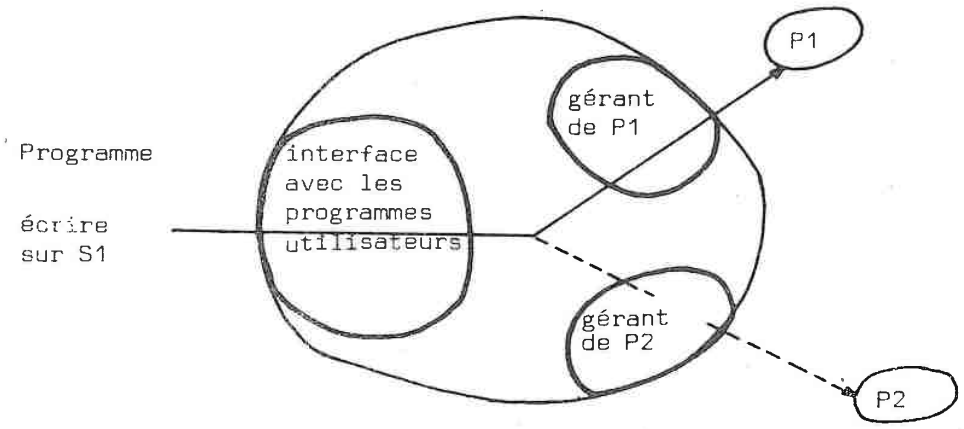


Fig. 39. Association nom symbolique, périphérique physique

_____ (S1, P1)

----- (S1, P2).

Ces opérations n'altèrent pas la nature des informations qui sont échangées entre les modules et les périphériques physiques.

Il peut cependant s'avérer nécessaire de permettre une modification de leur structure en faisant intervenir un transcodage ou un ensemble de programmes écrits dans ce but. C'est le rôle des procédures de formattage qui interviennent entre les programmes décrits dans un langage de haut niveau (FORTRAN, ALGOL, COBOL ...) et les périphériques de type lecteur de cartes ou imprimante.

PROCOL-T2000 permet également l'utilisation de spécifications de format destinés aux échanges utilisant des périphériques industriels. Elles permettent de décrire le nom des procédures-utilisateur requises pour effectuer des opérations aussi diverses que :

- la conversion des grandeurs mesurées en grandeurs physiques ayant une signification pour le programme ainsi que la réciproque :

. transformation d'un comptage en fréquence ou en vitesse
. transformation d'une tension en température

- le filtrage des informations mesurées :

. détection de cohérence
. filtrage numérique

- la détection de seuils, d'extrêma, de passage à zéro.

- le calcul de valeurs moyennes.

Dans SYGARE, ces opérations sont également possibles : on peut utiliser des modules qui soient spécialisés dans la réalisation de telles opérations. Ces modules peuvent accéder aux périphériques via le système de gestion des entrées/sorties, transforme les informations lues et les communique à d'autres modules sous forme de données transmissibles.
Ce sont des modules classiques activés selon les structures de contrôle de classe II et III habituelles (fig. 40).

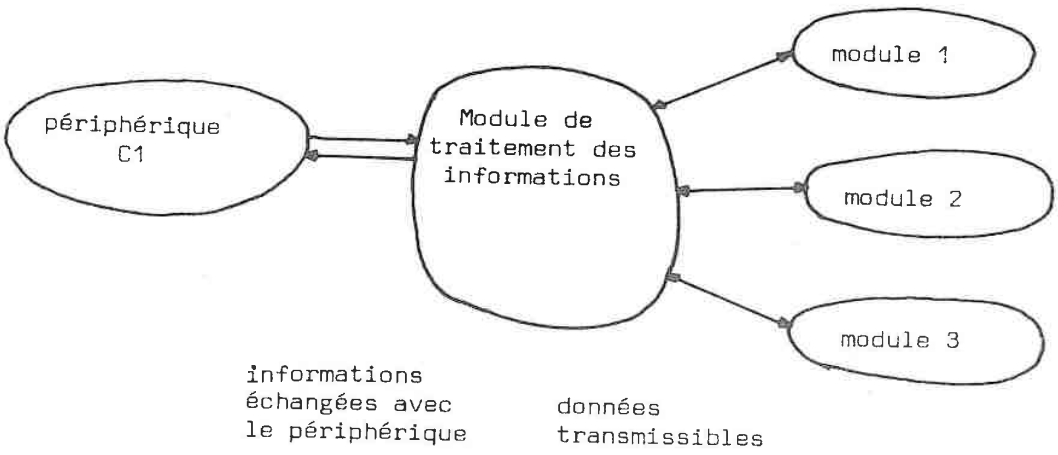


Fig.40. Utilisation d'un module de transformation des informations.

La définition de conditions d'alarme devrait également être possible en utilisant les structures de contrôle de classe III.
Exemple :
supposons que la mesure relevée sur le périphérique C1 soit une vitesse, et que l'on veuille définir l'événement SURVITESSE comme la lecture d'une valeur supérieure à VMAX sur C1, ceci s'écrirait de la façon suivante :

événement SURVITESSE : = mesure relevée sur C1 > VMAX.

Le gérant des événements doit donc intercepter toutes les mesures transmises par le gérant du capteur C1 pour les comparer à VMAX et éventuellement positionner l'événement SURVITESSE.
Cependant, il faut aussi permettre la détection d'un tel événement même lorqu'aucun module n'effectue de mesure sur C1, soit parce qu'ils n'en ont pas besoin, soit parce qu'ils sont momentanément bloqués. Il faut donc définir une période de mesure par défaut qui sera utilisée par le gérant des événements si aucun module ne lit la valeur de C1.
On devra alors écrire :

événement SURVITESSE : = mesure relevée sur C1 > VMAX,
période par défaut : TMAX.

Le gérant des événements se place donc en dérivation, à la sortie du gérant du capteur C1 (fig.41). Il intercepte les valeurs produites et peut éventuellement solliciter le gérant pour qu'il lui en fournisse d'autres.

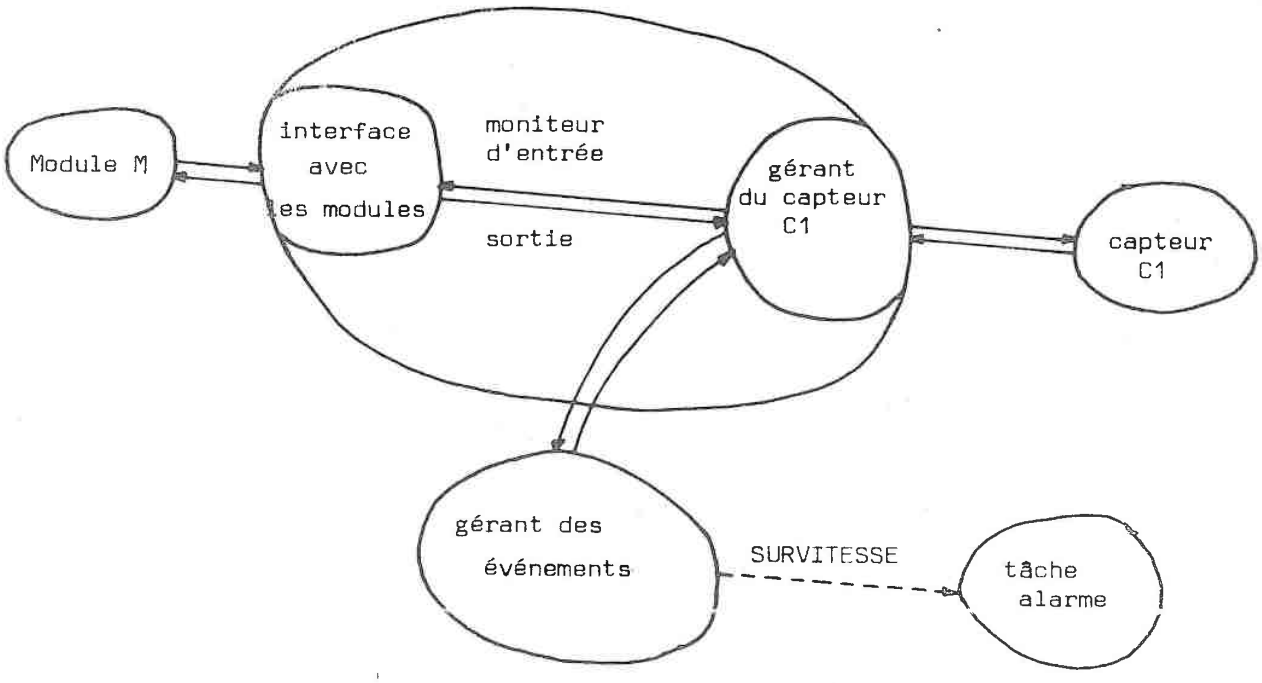


Fig.41. Connexions entre les événements et les entrées/sorties.

VII.3. LE CAS PARTICULIER DE SYGARE

La répartition des modules et des organes d'entrée/sortie sur un réseau impose, dans le cas particulier de SYGARE de s'orienter vers un moniteur d'entrées/sorties réparti.

On peut considérer que celui-ci est en fait constitué d'une collection de moniteurs d'entrée/sortie en étroite relation entre eux. Chacun doit posséder les propriétés qui ont été présentées dans les paragraphes précédents.

VII.3.1. L'organisation du moniteur d'entrée/sortie

Le moniteur d'entrée/sortie de chaque site doit être organisé autour d'un noyau central offrant un certain nombre de programmes de service et en particulier la propriété de dialoguer avec les moniteurs implantés sur les autres sites du réseau (fig. 42). Il peut en outre comporter :

- les différents programmes de gestion des périphériques implantés sur le site lorsqu'il y en a.

- l'interface avec les modules, ou le système d'exploitation, dans le cas où ceux-ci disposent de la possibilité d'effectuer des opérations d'entrée/sortie. On pourrait en effet concevoir des sites consacrés exclusivement à la gestion de terminaux ou des machines orientées vers le calcul scientifique et ne recevant que des modules n'effectuant aucune opération d'entrée/sortie. Dans ce cas l'interface avec les modules est bien entendu inutile et peut être supprimé.

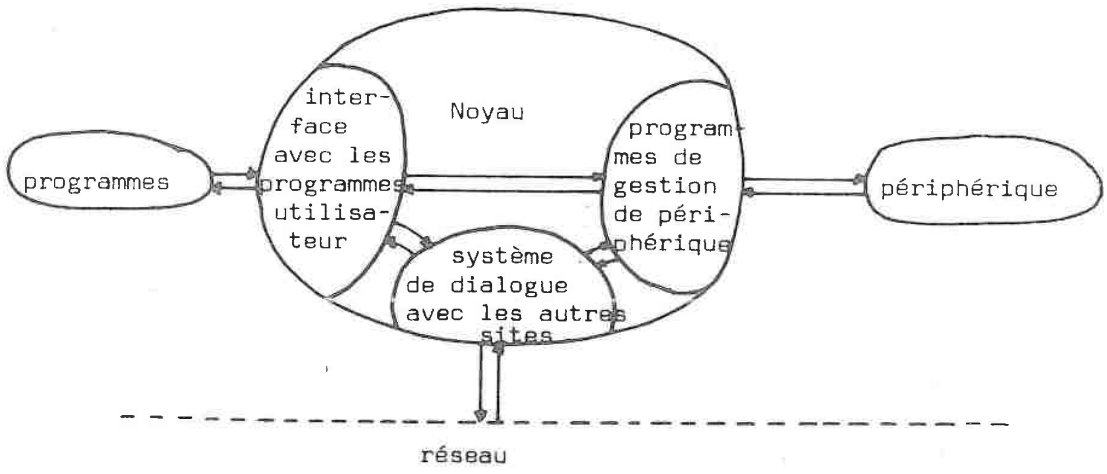


Fig.42. L'organisation du moniteur d'entrées/sorties d'un site du réseau SYGARE.

VII.3.2. L'utilisation du moniteur d'entrées/sorties de SYGARE

L'utilisation du moniteur d'entrées/sorties de SYGARE est analogue à celle d'un moniteur d'entrées/sorties implanté sur un site unique (fig. 43). Les modules de chaque site ne s'adressent qu'à l'interface implanté sur le même calculateur qu'eux. C'est lui seul qui aura la charge du dialogue :

- avec le module d'une part, pour l'analyse de la requête, la transmission du compte-rendu et l'éventuel échange d'informations,

- avec le programme de gestion des périphériques employés en utilisant éventuellement les services du système de dialogue dans le cas d'un périphérique éloigné.

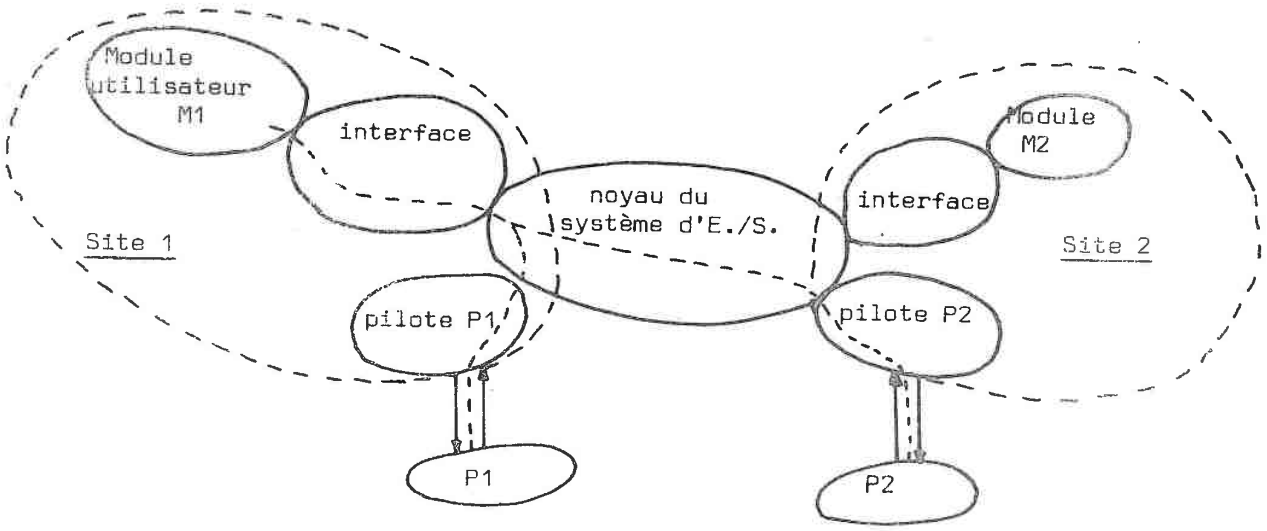


Fig.43. Utilisation du moniteur d'entrées/sorties du réseau SYGARE.

Grâce à une telle organisation, l'écriture des modules peut être réalisée de façon totalement indépendante de l'implantation effective et de celle des périphériques employés. Le module ne possède qu'un seul interlocuteur standard qui assure l'interface avec les périphériques du même site ou les périphériques éloignés sur le réseau.

VII.4. CONCLUSION

Le système de gestion des périphériques de SYGARE n'est actuellement qu'à l'état d'ébauches la présentation qui en a été faite dans ce chapitre montre combien il est important de ne pas négliger la réalisation d'un système de gestion

des entrées/sorties, en particulier dans le cadre d'un système implanté sur un réseau. L'influence de son organisation sur le découpage modulaire de l'application et sur la répartition des modules est importante et l'on ne doit pas la sous-estimer.

La validation de ce que nous venons de présenter passe cependant par le développement d'un système d'entrées/sorties complet et par son utilisation effective dans le cadre d'une application réelle.

CHAPITRE VIII

L'ARCHITECTURE DU SYSTEME D'EXPLOITATION

CHAPITRE VIII

L'ARCHITECTURE DU SYSTEME D'EXPLOITATION

L'architecture d'un système d'exploitation dépend fortement du matériel sur lequel il est implanté et du rôle qu'il aura à jouer dans l'utilisation de ce dernier. SYGARE a été développé pour répondre à un ensemble de spécifications qu'il nous semble nécessaire de rappeler ici, tant leur influence sur l'organisation du système d'exploitation est importante.

VIII.1. LES CARACTERISTIQUES DE SYGARE

VIII.1.1. Les caractéristiques matérielles

Du point de vue matériel, SYGARE est implanté sur un réseau hétérogène, formé de calculateurs reliés par différents modes de connexions :

- liaisons parallèles,
- liaisons séries,
- partage de mémoires communes à plusieurs processeurs ...

Dans ce dernier cas, les processeurs se partageant la mémoire commune peuvent être considérés indifféremment comme deux sites distincts du réseau ou comme une machine unique, le choix dépend en fait du but que l'on cherche à atteindre.

Les systèmes développés pour permettre l'exploitation des réseaux de calculateurs sont nombreux et variés. Chacun présente un certain nombre d'avantages et possède ses défauts. Cependant, le fait d'avoir été développé pour un réseau reste une caractéristique fondamentale.
Le rôle du système d'exploitation de SYGARE est de faire oublier que l'on est en présence d'un réseau et de donner l'illusion que l'on dispose d'une machine unique possédant la somme des proriétés des différentes machines qui forment le réseau.

VIII.1.2. Le rôle du système d'exploitation

Si le réseau est composé de deux machines dont l'une permet la calcul scientifique (arithmétique flottante câblée) et l'autre la gestion d'entrées/sorties graphiques (logiciel de gestion d'écran ou de table traçante), une application scientifique aboutissant à un tracé de courbe pourra être programmée comme si elle devait en fait s'exécuter sur la machine possédant les deux propriétés intéressantes. Un même langage permet de décrire chacune des facettes de l'application. Celle-ci sera cependant composée de deux ensembles de modules dont les fonctions seront très nettement distinctes :

- modules réalisant les calculs arithmétiques
- modules effectuant la sortie des résultats calculés.

Les modules s'exécuteront sur les machines disposant des ressources nécessaires à leur déroulement sans qu'il soit nécessaire d'en tenir compte au niveau de la conception des modules.

Ainsi les instructions de synchronisation ou d'échange d'informations entre modules n'ont elles pas été introduites dans le langage d'écriture des modules.

Celles-ci font l'objet d'une description à un niveau supérieur et sont directement utilisées par le système d'exploitation pour piloter l'application :

- structures de contrôle de classes II et III
- déclarations de connexions entre modules.

Nous avons vu que ces fonctions nécessitaient l'introduction d'un gérant de données transmissibles dont le rôle était de recueillir, conserver et transmettre les données aux différents modules qui se les partagent. A partir de là, les synchronisations entre modules d'une même tâche, ou de tâches concurrentes peuvent être réalisées. Enfin, les relations avec le milieu extérieur, telles que le temps, les interruptions, la prise d'une mesure doivent être exploitées pour lancer, tuer ou suspendre les tâches fonctionnelles, et ceci indépendamment de la répartition des modules qu'elles mettent en oeuvre.

VIII.1.3. Les fonctions du système d'exploitation

. La fonction essentielle du système implanté sur le réseau sera de gérer le réseau et les différentes machines dont il est constitué.

. Un autre point essentiel est constitué de la gestion des entrées/sorties qui a fait l'objet du chapitre précédent.

. L'interprétation des structures de contrôle de classe III permet, en fonction des événements extérieurs ou internes à l'application, d'entreprendre diverses actions sur les tâches fonctionnelles.

. Celles-ci sont constituées de modules liés par des structures de classe II que l'on examinera à chaque fois qu'une action sur une tâche est demandée.

Exemple : l'interprétation de la structure de contrôle :

A 17 H 30 lancer TACHE T

conduira à examiner les structures de contrôle de classe II décrivant l'enchaînement des modules qui constituent TACHE T. L'aboutissement de cela sera le lancement du ou des premier (s) module (s) de la tâche.

. La dernière fonction du système d'exploitation sera d'assurer la gestion des données transmissibles. Celles-ci constituant l'unique méthode de communication d'informations entre les modules. Elle est indispensable dans le cas d'applications réparties lorsqu'on veut n'avoir pas à tenir compte de la répartition lors de sa conception, ou lorsque la répartition est susceptible d'évoluer (reconfiguration dynamique). Il faut en effet que l'un des éléments reste en liaison constante avec l'ensemble des autres, c'est le rôle du système d'exploitation de gérer les liaisons entre les modules.

VIII.2. CENTRALISATION OU REPARTITION

Le système d'exploitation d'un réseau peut être conçu de deux façons différentes et complémentaires :

- sous forme d'un système centralisé,
- sous forme d'un système à intelligence répartie.

Chacune de ces conceptions présente ses avantages et ses inconvénients.

VIII.2.1. Système centralisé

Un système centralisé est particulièrement indiqué dans le cadre d'un réseau étoilé, où le calculateur central est doté d'une puissance nettement supérieure à celle des calculateurs distants, et reste disponible en permanence.

Le système d'exploitation implanté sur le site principal, doit alors comporter l'ensemble des fonctions nécessaires à l'utilisation complète du réseau. Dans le contexte de SYGARE, il devrait gérer les différents autres sites et permettre d'exécuter les différentes fonctions qui ont été décrites au paragraphe précédent.

Dans ce but, chacun des sites éloignés devrait être doté des éléments nécessaires permettant au système central, aux modules et aux organes d'entrée/sortie de dialoguer. Les systèmes d'exploitations des sites distants se comportent alors simplement comme des courroies de transmission" qui permettent d'exécuter les ordres émis par le système central, de lui transmettre les informations recueillies sur le site même ou les comptes rendus d'exécution des ordres reçus :

- télé-chargement d'un module sur le calculateur,
- lancement, terminaison, suspension d'un module,
- transmission, récupération des données transmissibles
- dialogue avec les modules pour les opérations d'entrée/sortie
- dialogue avec les gérants de périphériques.

L'un des avantages de concevoir le système d'exploitation sous forme centralisée sur une machine puissante consiste en la facilité avec laquelle il peut être développé. On sait bien, à l'heure actuelle, écrire un système d'exploitation destiné à une machine unique et les problèmes de synchronisation, de conservation des informations, de redémarrage ont reçu de nombreux types de solutions entre lesquels il suffit de choisir celle qui correspond le mieux au type de problème que l'on veut résoudre.
L'interface entre la machine centrale et les calculateurs éloignés est assez simple à réaliser. Il faut une procédure de transmission capable de supporter des messages de taille variable et de teneur quelconque. Les systèmes d'exploitation des sites éloignés peuvent être réalisés à peu de frais et ne nécessitent pas un encombrement mémoire trop important puisqu'ils ont en fait une liberté très limitée.
En outre, l'adjonction d'un calculateur d'un modèle différent à ceux qui existent déjà peut être réalisée assez facilement, le logiciel d'exploitation dont il sera doté étant d'une réalisation très rapide puisqu'il suffit de traduire dans un langage supporté par la machine, un algorithme déjà au point.
C'est à l'utilisation que l'on s'aperçoit combien un tel système centralisé risque d'être pénalisant.
Une défaillance du site central entraîne l'impossibilité de fonctionner, même dans un mode dégradé. Il faut alors arrêter complètement l'exploitation de l'application ou passer à une exploitation manuelle qui n'est pas toujours possible.
Le volume des messages émis et reçus par le site central peut dans certains cas

(apparition d'un défaut, phase critique, phénomène d'avalanche) être tel qu'il entraîne une dégradation des performances du système complet : augmentation des temps de réponse, délai de traitement ...
En outre, il est possible que le réseau atteigne une saturation, lorsque des modules très courts à l'exécution et nécessitant de nombreuses données transmissibles sont répétés de nombreuses fois.

Enfin, le remplacement du calculateur central par un modèle de type différent nécessite la reprise complète du logiciel d'exploitation de ce site. Celui-ci, supportant les différentes fonctions de SYGARE est nécessairement assez important. Sa traduction dans un langage d'une autre machine peut s'avérer coûteuse et nécessite une période d'essais pendant laquelle l'application ne pourra être exploitée avec la même sûreté qu'auparavant.

VIII.2.2. Système décentralisé

Un système décentralisé présente un certain nombre d'avantages dans le cadre d'un réseau maillé dont les différents constituants ont des puissances à peu près équivalentes.

La décentralisation peut apparaître comme la répartition des différentes fonctions sur le réseau : la gestion des données transmissibles étant réalisée sur un site, l'interprétation des structures de contrôle sur un autre.
Une décentralisation de ce type présente bien peu d'avantages et possède cependant les inconvénients d'un système centralisé :

. Le réseau reste fortement employé à chaque activation et terminaison de module

. La défaillance ou l'arrêt momentané de l'un des sites entraîne le non-fonctionnement de la totalité de l'application.

Une décentralisation bien plus efficace à notre avis consiste en une distribution du potentiel de décision que possède seul le calculateur central d'un système centralisé.

Chacune des fonctions utilisées est implantée dans ce cas sur les différents sites du réseau, ce qui confère une plus grande autonomie aux différents calculateurs.
Ainsi, l'appel au système de gestion des entrées/sorties peut-il être traité plus rapidement lorsque le module appelant et le périphérique concerné sont implantés sur le même site.
De même l'enchaînement de deux modules d'une même tâche implantés sur la même machine peut être effectué sans que l'on ait besoin d'attendre d'ordre extérieur si les données transmissibles nécessaires à l'exécution du second module sont

disponibles sur le site concerné, ce qui est vrai dans la plupart des cas : il n'est pas rare en effet qu'un module utilise en majorité des données produites par l'un de ses prédécesseurs.

Les inconvénients dûs à un tel système d'exploitation sont essentiellement ressentis lors de son développement, de sa maintenance et de son extension.
Il faut en effet développer au départ les différents systèmes d'exploitations destinés aux diverses machines en présence. L'intégration de nouvelles machines au réseau pose les mêmes problèmes par la suite. Ceux-ci peuvent cependant recevoir quelques éléments de solution :

- Lorsque l'analyse des différentes fonctions a été réalisée dans le cas le plus général, la phase de développement des différents systèmes est simplifiée et devrait pouvoir se limiter à la programmation du système dans un langage accepté par la machine, au moyen de compilateurs croisés par exemple.

- Cette génération de systèmes exécutables pourra sans doute être automatisée en utilisant pour la description d'un modèle de système d'exploitation d'un site, un langage d'écriture de système de haut niveau tel que LIS (ICHBIAH 77), LEST (BET 77)
La production du code destiné aux différentes machines se résumera alors à la description des caractéristiques de la machine elle-même : taille du mot mémoire, description des instructions ...
Un autre inconvénient d'une telle organisation du système d'exploitation est qu'il nécessite une capacité de mémoire importante sur chacun des sites du réseau. Bien que le prix de la mémoire diminue sans cesse, il constitue cependant une part non négligeable du coût global d'un microprocesseur.
Enfin, la faible capacité de traitement d'un microprocesseur peut conduire à des cas où le système d'exploitation passe un temps non négligeable à se gérer lui-même. Le nombre de modules qui pourra alors être confié à un site risque d'être diminué si l'on désire un temps de réponse satisfaisant. Les deux dernières objections à un système réparti sont sans nul doute dues au fait que le microprocesseur n'est pas adapté aux systèmes complexes et susceptibles d'évoluer. Il est par contre intéressant dans le cas de petits automatismes figés qui peuvent être produits en grande série, ce qui n'est malheureusement pas le cas de SYGARE.

Les avantages d'un système à intelligence distribuée concernent essentiellement les performances du système complet au moment de l'exécution :

. L'utilisation du réseau pour la gestion par le système d'exploitation est réduite au minimum, celui-ci peut alors être pleinement utilisé pour la transmission des informations nécessaires à la bonne exécution des tâches fonctionnelles qui lui

sont confiées. Celles-ci peuvent alors se dérouler suivant un rythme meilleur.

. Le temps de réponse aux sollicitations extérieures est amélioré lorsqu'une réaction, même partielle, peut être entreprise par le calculateur qui a perçu l'événement. Un traitement plus complet nécessitant l'intervention d'autres machines pourra ensuite intervenir pour traiter plus finement l'événement.

. L'intervention du système d'exploitation entre l'exécution de modules implantés sur un même site étant réduite à son minimum, le concepteur d'une application n'hésitera pas à éclater en plusieurs modules une action complexe dont la mise en oeuvre est délicate.

. La défaillance de l'une quelconque des machines du réseau ne provoquera pas nécessairement l'arrêt complet de l'exploitation et un mode de fonctionnement dégradé sera possible si les modules indispensables peuvent être réimplantés sur les autres sites du réseau. Ceux-ci doivent posséder les ressources nécessaires à l'exécution des modules : capacité mémoire, code d'ordres machine, fonctions programmées ...

VIII.3. LE SYSTEME D'EXPLOITATION REALISE

Le système d'exploitation de SYGARE a été développé et implanté sur :

- d'une part, un mini-ordinateur de milieu de gamme : le SOLAR 16-40 de SEMS
- d'autre part, un microprocesseur MOTOROLA 6800.

La connexion réalisée entre chacun des matériels en présence est du type parallèle sur 16 bits soit la longueur d'un mot sur SOLAR. La nature de la liaison n'a cependant eu qu'une influence mineure sur le système d'exploitation, une liaison de type série mode synchrone ou asynchrone aurait pu convenir également. Le choix d'une liaison de type parallèle nous a été dicté par des impératifs liés à la disponibilité du matériel. En outre, il permet une rapidité de transfert maximum, celle-ci n'étant limitée que par la rapidité d'exécution du programme de gestion de la liaison sur la machine la plus lente : le MOTOROLA 6800.
Celle-ci était réalisée en mode programmé sur interruption sur chacune des deux machines, par un programme spécifique intégré au moniteur sur M6800 (PGPI 77), par un Driver intégré au moniteur de gestion des entrées/sorties IOCS (IOCS 76) sur Solar.
La structure du système développé correspond au système réparti que nous avons présenté au paragraphe précédent.

VIII.3.1. Le système developpé sur SOLAR

Le système d'exploitation développé sur SOLAR 16-40 a été conçu autour du système de multiprogrammation MPES de la SEMS. Ce produit offre

simultanément un système de traitement de travaux par lots (l'exécutif batch) utilisé pour la production de programmes :

- système d'exploitation du Solar 16-40 ou du M 6800
- tâches fonctionnelles de l'application

et un ensemble de fonctions "temps réel" qu'il aurait été aussi inutile que coûteux de développer à notre tour.
Le moniteur d'exécution de SYGARE sur Solar est conçu comme un ensemble de tâches programmées de manière réentrante de l'exécutif temps-réel de MPES. Celles-ci sont issues d'une même tâche mère qui assure la communication avec les matériels connectés, avec les fonctions de système MPES (système de gestion des entrées/sorties, système de gestion de fichiers) et permet la synchronisation et la communication entre ses tâches filles.
Le système de protection inter-tâches du Solar interdit la communication directe entre les différentes tâches filles. Celles-ci doivent donc impérativement utiliser l'interface qui a été définie avec la tâche mère lorsqu'elles ont une requête à adresser à une autre tâche fille, à la tâche mère ou à un système éloigné.
Le rôle attribué à chaque tâche fille correspond aux différentes fonctions du système d'exploitation de SYGARE.

- gestion du dialogue avec l'opérateur sur la télétype de service,
- gestion des événements et interprétation des structures de contrôle de classe III qui leur sont associées.
- enchaînement des séquences des modules formant une tâche, tâche fonctionnelle suivant le schéma défini par les structures de contrôle de classe II.
- gestion et conservation des données transmissibles.

La séparation des différentes fonctions permet leur intégration progressive dans le système d'exploitation, au fur et à mesure de l'évolution de leur développement et des essais.
Pour l'instant, le système d'exploitation est constitué de :

- la tâche mère qui dialogue avec le Motorola 6800 et permet le chargement initial du système sur ce dernier,
- les tâches filles de dialoguer avec l'opérateur local ou sur M 6800, et de séquencement des modules.
- un système de gestion des entrées/sorties restreint permet aux modules d'accéder aux périphériques implantés sur Solar.

L'ensemble a été réalisé en utilisant le langage d'assemblage SOLAR, le seul des langages disponibles sur cette machine avec PL 16, qui nous permette d'accéder aux requêtes-système utilisées dans SYGARE. Le résultat n'est bien entendu pas portable mais, comme il nécessite pour s'exécuter un système d'exploitation de type RTES-D ou MPES, il aurait été difficile qu'il en soit autrement.

Le but recherché n'était pas d'écrire un système portable mais de valider les options qui avaient été retenues lors de la définitions puis du développement de SYGARE. Le système n'étant pas complet, il nous est difficile de conclure à l'heure actuelle sur la validité des choix que nous avons fait.

VIII.3.2. Le système d'exploitation du Motorola 6800

Le microprocesseur Motorola 6800 avec lequel nous avons travaillé était construit autour du kit d'évaluation fourni par le constructeur. Celui-ci comprend un système de base fourni sur une mémoire PROM, permettant l'entrée de programmes et le lancement de leur exécution à partir d'un clavier hexadécimal. Il disposait en outre d'un interface-cassette permettant l'archivage des programmes sur support magnétique, (MOTOROLA).

Nous y avons ajouté 4K-octets de mémoire vive, deux interfaces parallèles (PIA) pour le dialogue avec le Solar 16-40 et un interface série asynchrone permettant la connexion d'un organe de dialogue : télétype ou console de visualisation.

Le logiciel que nous avons développé n'utilise le système fourni par le constructeur que pour le chargement initial de la cassette et le lancement d'un système de base. Celui-ci permet ensuite le chargement du système d'exploitation complet depuis le SOLAR et lui laisse le contrôle lorsqu'il est complètement implanté en mémoire.

Ceci nous a permis de modifier le logiciel d'exploitation du microprocesseur, pour tester et intégrer de nouvelles fonctions sans qu'il soit nécessaire de le rentrer à chaque fois au clavier avec les risques d'erreurs que cette méthode comporte.

Le logiciel d'exploitation du M6800 était produit par un assembleur croisé implanté sur SOLAR sous forme d'un fichier-disque contenant l'image-mémoire du moniteur. Le téléchargement de ce fichier est la première tâche effectuée par le système d'exploitation implanté sur SOLAR, lors de son lancement, ce qui permet d'être sûr du système que l'on a en mémoire du microprocesseur lors d'un démarrage de l'exploitation.

Le moniteur du Motorola 6800 a été conçu pour en permettre l'exploitation en multiprogrammation. Un séquenceur programmé gérant 16 tâches de priorités fixes hiérarchisées lui sert de noyau. Ce séquenceur prend le contrôle après le traitement de

chaque interruption et lorsqu'une tâche lui adresse une requête. Les requêtes que nous avons introduites permettent le lancement, la suspension et la synchronisation des différentes tâches et la demande d'exécution des opérations d'entrée/sortie sur le site :

- utilisation de la console opérateur
- gestion de la ligne de communication avec le SOLAR.

Parmi les tâches implantées, on retrouve sur M 6800 l'équivalent de la tâche mère sur SOLAR, qui intègre, lance et permet aux autres tâches du moniteur SYGARE de dialoguer.
C'est elle qui gère le dialogue avec le moniteur SOLAR pour l'ensemble du système d'exploitation. Elle sert de relais aux trois tâches privilégiées qui peuvent dialoguer avec leurs homologues sur SOLAR :

- l'interpréteur des structures contrôle de classe II,
- le gérant des données transmissibles,
- le système de gestion des entrées/sorties du réseau.

Seule, la première de ces tâches est actuellement opérationnelle, les autres sont prévues dans le système d'exploitation mais n'ont pas été complètement développées. De ce fait, il ne nous a pas été possible d'essayer une tâche fonctionnelle complète sur le réseau.
L'inadaptation d'un microprocesseur tel que le Motorola 6800 que nous avons utilisé, à régler les problèmes liés à la synchronisation de processus concurrents nous a conduit à développer un séquenceur programmé permettant la multi-programmation de ce calculateur.

Une approche complémentaire a été suivie par P. Vernel et M.L. Lagier-Besson lors du développement de NARCIS (LAGIER 76)(VERNEL 77).
Une fédération de microprocesseurs organisés autour d'une mémoire commune et pilotée par un régisseur et un secrétaire est utilisée pour la conduite d'applications multiprogrammées. Chaque calculateur de la fédération exécute la tâche qui lui est confiée par le régisseur et lui adresse un compte-rendu.
Une telle forme de multiprogrammation permet de s'affranchir des problèmes posés par la vitesse d'exécution du microprocesseur et qui limitent l'emploi du Motorola 6800 dans notre application.

VIII.3.3. La procédure de communication

La procédure de communication entre le SOLAR et le Motorola 6800 est de type half duplex. Il ne nous était en effet pas possible d'utiliser le coupleur parallèle du SOLAR en full duplex pour des raisons dues aux microprogrammes gérant le canal.

Comme nous ne disposons d'aucune autre temporisation que les requêtes du système RTES SOLAR, nous avons été conduits à utiliser la liaison de façon dissymétrique, le Solar étant maître et le M 6800 esclave. C'est donc toujours le Solar qui a l'initiative des échanges.
Lorsque, sur un site l'une des tâches désire dialoguer avec son homologue sur l'autre site, un indicateur est positionné indiquant le numéro de la tâche en attente d'émission ou de réception.
Si celle-ci est implantée sur Solar, et que la ligne est libre à ce moment, la tâche mère émettra aussitôt vers le M 6800 une séquence de polling ou de selecting comportant dans un en-tête un champ indiquant la ou les tâches en attente.
La réception d'une telle séquence de supervision provoque sur M 6800 la comparaison de l'information qu'elle renferme avec l'indicateur des tâches en attente et, s'il y a une intersection commune, les tâches homologues pourront alors dialoguer après que la tâche mère ait émis un accusé de réception positif indiquant la tâche concernée par le dialogue. Celui-ci peut ensuite se dérouler seul.
S'il n'y a pas d'intersection commune, un accusé de réception négatif est émis et à sa réception sur Solar, une temporisation est assurée à l'issue de laquelle la demande est renouvelée.
Toute la difficulté consiste à trouver quelle est la durée de la temporisation à choisir pour que l'on ne surcharge pas trop les deux sites et que le temps de réponse ne soit pas trop mauvais. Dans tous les cas, la durée maximum que devra attendre une tâche implantée sur M 6800 alors que son homologue est prête à dialoguer vaut la durée de la temporisation si la ligne est libre à ce moment là.

Une telle procédure de transmission permet le multiplexage de la ligne entre différentes tâches utilisatrices et évite qu'il y ait un nombre trop important de messages en attente. Elle permet en outre la synchronisation des tâches homologues sur deux sites distincts.

VIII.4. CONCLUSION

Le comportement satisfaisant des systèmes d'exploitations que nous avons développé pour les calculateurs formant la maquette du réseau SYGARE permet de valider, au moins partiellement, les choix que nous avons été conduit à faire.

Il est à noter que le logiciel équipant les matériels mis à notre disposition ayant été programmés de façon réentrante, il serait possible, moyennant la réécriture des tables des systèmes, d'étendre le réseau.

Cette extension serait très simple à réaliser dans le cas de la connexion d'une machine d'un type déjà utilisé car nous disposons du logiciel nécessaire à leur fonctionnement. Dans le cas de machines d'un type différent, il serait nécessaire de développer un système d'exploitation respectant les interfaces définis,

et utilisant une procédure de transmission identique. Bien que cela nécessite de bien connaître la structure de la nouvelle machine, cela serait néanmoins réalisable dans des délais relativement courts puisque l'étude du système d'exploitation a été réalisée.

Cette extension permettrait de mieux cerner les problèmes de reconfiguration et de portabilité des modules constituant une tâche fonctionnelle implantée sur le réseau.

CONCLUSION

Cet ouvrage ne présente que l'un des nombreux aspects du projet SYGARE. Notre attention s'est en effet concentrée sur lesproblèmes liés à la conception d'une application répartie et à leurs conséquences sur le système d'exploitation du réseau.

Alain Cochet-Muchy a développé le système de production des modules destinés aux calculateurs-listes de notre réseau expérimental en se réservant la possibilité de l'étendre à d'autres machines. La conjonction de nos efforts avait pour but d'alléger la tâche des concepteurs d'applications réparties en temps réel.

Le projet SYGARE a en effet pour but de mettre à leur disposition un ensemble cohérent d'outils permettant de concevoir simplement les différentes actions à réaliser, sans avoir à se préoccuper de leur lieu d'implantation ni de leur relation.

L'étape suivante consiste à assembler ces actions élémentaires pour former des tâches fonctionnelles en utilisant des structures de contrôle décrivant les enchaînements à réaliser et des déclarations de connexion explicitant le flux des données transmissibles entre les différents composants de l'application.

Enfin, les structures de contrôle de classe III faisant référence à des événements, dont la notion a été étendue de façon à couvrir tout ce qui peut caractériser l'état de l'application, permettent de lier le déroulement des différents programmes de l'application à l'évolution du milieu externe.
Cette hiérarchie a pour but d'accroître la lisibilité et la maintenabilité du logiciel de l'application, tout en introduisant des contrôles statiques et dynamiques de la validité de la description fournie.
Elle nous permet en outre d'introduire de façon automatique des mécanismes de protection et de synchronisation, qui ont pour effet de simplifier le travail de conception et de minimiser les causes d'erreurs ou de mauvais fonctionnement.

L'application des principes retenus dans SYGARE à la description d'un certain nombre d'applications réparties ou non, mettant en jeu des mécanismes de synchronisation et de communication plus ou moins compliqués nous a permis de nous assurer de leur validité.

Il est à noter en effet, que quoiqu'il ait été développé pour un réseau hétérogène de calculateurs, SYGARE est parfaitement adapté au traitement de tous les processus qui requièrent des échanges d'information et des synchronisations même s'ils doivent se dérouler en multiprogrammation sur une seule machine.

Nous souhaitons que les travaux que nous avons entrepris soient poursuivis par une extension du réseau. Un passage à l'échelle industrielle, et l'utilisation de SYGARE pour piloter une application en vraie grandeur seraient le meilleur moyen de valider les principes de base que nous avons retenus et permettraient d'effectuer des mesures indispensables à une conclusion impartiale.

Nous pensons également que l'organisation de SYGARE est bien adaptée au traitement des problèmes liés à la reconfiguration dynamique d'une application répartie et nous souhaiterions également que notre étude serve de point de départ à une recherche dans cette direction.

BIBLIOGRAPHIE

BANATRE 77 — M. Banâtre, A. Couvert, D. Herman, M. Raynal
Conservation d'objets typés. Journées Bigre, IRIA-France, 24-25 novembre 1977.

BETOURNE 77 — C. Bétourné, L. Feraud, J. Joulier, J.M. Rigaud
LEST : langage d'écriture de systèmes
Journées Bigre, IRIA-France, 24-25 novembre 1977.

BEZIVIN 77 — J. Bezivin, Y. Jegou, J.L. Nebut, R. Rannou
Production de systèmes temps-réel fiables et efficaces
Journées Bigre, IRIA-France, 24-25 novembre 1977.

BRINCH-HANSEN 73 — P. Brinch-Hansen
Operating system principles. Prentice Hall, 1973.

COCHET-MUCHY 78 — A. Cochet-Muchy
La compilation dans le projet SYGARE.
Thèse de Docteur-Ingénieur, INPL Nancy, 1978.

COCHET-MUCHY 77 — A. Cochet-Muchy, P. Nonn
Le projet SYGARE. Journée AFCET Temps Réel, Nancy, 9.10.1977.

CROCUS 75 — Collectif
Système d'exploitation des ordinateurs. Dunod, 1975.

DIJKSTRA 68 — E.W. Dijkstra
The structure of the multiprogramming system
CACM vol. 11, n° 5, mai 1968.

DIJKSTRA 71 — E.W. Dijkstra
Hierarchical ordering of sequential processes
Acta Informatica, vol. 1, n° 2, 1971.

DIJKSTRA 75 — E.W. DIJKSTRA
Guarded commands, non determinacy and formal derivation of programs
CACM, vol. 18, n° 8, août 1975.

FYLSTRA 75 — D.H. Fylstra
HAL/S programming language
Journées AFCET Langages Temps Réel, Paris 1975.

HABERMANN 76 — A. Habermann
Path expression technical report
Carnegie Mellon University, 1976.

HOARE 74 — C.A.R. Hoare
Monitors : an operating system structuring concept
CACM, vol 17, n° 10, octobre 1974.

HOLT 71 — R.C. Holt
Some deadlock properties of computer systems
III ACM Symposium on Operating Systems Principles, Stanford, octobre 1971.

ICHBIAH 77 — J.O. Ichbiah, G. Ferran
Separate definition and compilation in LIS and its implementation
M.O.L. Bulletin n° 6, IRIA ed., 1977, p 17-26.

IOCS 76 — SEMS
IOCS : Manuel de référence 1.164.001/00 36 03, 1976.

LADET 77 — P. Ladet
Outils de structuration temps réel dans la commande des procédés industriels. Thèse de 3ème cycle, INP Grenoble, 1977.

LADET 77 — P. Ladet, P. Deschizeaux
Programmation en temps réel de synchronisation par gestion événement-processus.
Journées AFCET Temps Réel, Paris, novembre 1977.

LAGIER 76 — M.L. Lagier
Conception et réalisation d'un moniteur temps réel pour un système multimicrocalculateurs.
Thèse de Docteur-Ingénieur, Université de Nancy I, 1976.

LAUESEN 77 — S. Lauesen
BOSS-2 : a large semaphore based operating system
CACM, vol. 18 n° 7, juillet 1977.

LE CALVEZ 77 — F. Le Calvez
Gaelic, un langage de description globale des synchronisations de processus.
Journées AFCET Temps Réel, Paris 3-4 novembre 1977.

LEDGARD 75 — H.F. Ledgard, M. Marcotty
A genealogy of control structures
CACM, vol. 18 n° 11, novembre 1975.

MOSSIERE 77 — J. Mossière, M. Tchuente, J.P. Verjus
Sur l'exclusion mutuelle dans les réseaux informatiques
IRISA. Publication interne n° 75, octobre 1977.

MPES 77 — SEMS
MPES : Manuel de référence 1 164 /00 36.

MOTOROLA 76 — Motorola Company
M 6800 Microprocessor programming manual. Notice technique, 1976.

NEBUT 74 — J.L. Nebut
Conception d'un système de langage de programmation
Thèse de Docteur-Ingénieur, Université de Paris IV, novembre 1974.

NEGARET 76 — R. Negaret
Etude de l'allocation des ressources dans les systèmes informatiques répartis. Thèse de 3ème cycle, Rennes, 1976.

NONN 78 — P. Nonn, J.P. Thomesse
Le système d'exploitation dans le projet SYGARE
Congrès AFCET, paris, 15 novembre 1978.

PARAYRE 76 — P. Parayre
Real time programming language LTR
Colloque IRIA, Séminaire international sur la programmation en temps réel. IFAC-IFIP, juin 1976.

PATIL 71 — S.S. Patil
Limitations and capabilities of Dijkstra's semaphore primitives for coordination among processes. Computer Structures Group Memo 57. Projet MAC-MIT 1971.

PGPI 77 — Utilisation des entrées/sorties numériques CGPI 24
Notice technique SOLAR, ENSEM 1977 (publication interne).

RITOUT 72 — M. Ritout, P. Bonnard, P. Hugot
Procol : a programming language for process control
5° Congrès IFAC, Paris 1972.

RIVEST 76 — R.L. Rivest, R.P. Vaughan
The mutual exclusion problem for unreliable processes
7th Annual Symposium on foundations of computing science.
Houston Texas, 25-27 octobre 1976.

ROBERT 77 — P. Robert, J.P. Verjus
Toward autonomous description of synchronization modules
Unuversité de Montréal/ENSIMAG
Proceeding of IFIP Congres Toronto, août 1977.

SERAIN 76 — D. Serain
Mécanismes de description et d'évolution d'une structure de réseau de processus s'exécutant sur une machine multiprocesseur.
Thèse de Docteur-Ingénieur, INP Grenoble, 1976.

STERIA — Procol T 2000
Notice technique. 1 161 220/00 39 00.

THOMESSE 77 — J.P. Thomesse
A new set of software tools for conceiving and realizing distributed system in process control. IFAC-IFIP workshop on real time programming, Eindhoven, juin 1977.

THOMESSE 77 — J.P. Thomesse, A. Cochet-Muchy, P. Nonn
Conception et réalisation de systèmesrépartis en conduite de processus industriels - Présentation du projet SYGARE.
Journées sur la production de systèmes industriels. IRIA France, novembre 1977.

THOMESSE 78 — J.P. Thomesse, A. Cochet-Muchy, P. Nonn
Data flow analysis for the data flow and management of mutual exclusion and synchronization in real time applications
IFAC-IFIP Workshop on real time programming, Mariehamn, Finland juin 1978.

VERNEL 77 P. Vernel
NARCIS : Nouvelle architecture d'un calculateur industriel spécialisé.
Thèse de Docteur ès Science, INPL Nancy, avril 1977.

VOJNOVIC 77 D. Vojnovic, M. Lacrouts
Structuration d'une application industrielle autour d'un noyau
Journée Bigre, IRIA, novembre 1977.

WALDEN 72 D. Walden
A system for interprocess communications in a resource sharing computer network.
CACM, vol 15, n° 4, avril 1972.

WULF 74 Wulf, Cohen, Corwin, Jones, Lewin, Pierson, Pollack
Hydra : the kernel of multiprocessor operating system
CACM, vol. 17, n° 6, juin 1974.