Bureau d'Orientation
de la Normalisation
en Informatique

PROJET SCEPTRE

PROPOSITION de STANDARD de NOYAU d'EXECUTIF TEMPS REEL

mai 1980

CONVENTION N° 79.2.36.0055

Domaine de Voluceau
Rocquencourt
B.P 105 78150 - Le Chesnay
France
[redacted] 90 20

PROJET SCEPTRE

PROPOSITION de STANDARD de NOYAU d'EXECUTIF TEMPS REEL

mai 1980

CONVENTION N° 79.2.36.0055

PROJET SCEPTRE

PROPOSITION DE STANDARD

DE

NOYAU D'EXECUTIF TEMPS REEL

Melle	H. FALLOUR	(CAP SOGETI LOGICIEL)
MM.	C. FAULLE	(CIMSA)
	J. FEBVRE	(SEMS)
	M. KRONENTAL	(BNI)
	G. MEMMI, F. MINEL	(ECA AUTOMATION)
	JJ. SIMON	(CRCAM)
	D. VOJNOVIC	(PHILIPS DATA SYSTEMS)

1979-1980

MINISTERE DE L'INDUSTRIE

DIRECTION DES INDUSTRIES ELECTRONIQUES

ET DE L'INFORMATIQUE

BUREAU D'ORIENTATION DE LA NORMALISATION

EN INFORMATIQUE

CONVENTION N° 79.2.36.0055

IMPUTATION BUDGETAIRE 66.05.30 90

SOMMAIRE GENERAL

PREAMBULE

Le projet SCEPTRE(*), développé au BNI depuis 1977 en phases successives :

. étude de faisabilité : rapport PEANUTS, Novembre 1977,

. phase 1 : rapport WALNUTS, Septembre 1978,

. phase 2 : proposition de standard de noyau d'exécutif Temps Réel,

fait la synthèse de l'expérience industrielle française en matière de construction des Exécutifs Temps Réel.

Cette synthèse correspond à un compromis qui tient compte des besoins des applications et des architectures des machines qui seront utilisés dans le début des années 80. Elle s'exprime sous la forme de la présente proposition de standard dont l'ambition est multiple :

. montrer comment structurer les mécanismes de coopération entre activités parallèles dans un Exécutif Temps Réel;

. fixer des concepts et un vocabulaire communs mettant fin aux quiproquo actuels, causés par l'emploi d'un "franglais" ne correspondant à aucun modèle rigoureux du parallélisme;

. spécifier un ensemble d'opérations appelé noyau SCEPTRE permettant de concevoir et de construire un Exécutif Temps Réel portable et performant.

. expliciter la mise en oeuvre de ce noyau.

Cette proposition devra subir l'épreuve des réalisations industrielles avant de faire l'objet d'actions de normalisation. Ceci sera l'objet de la phase 3 du projet SCEPTRE. C'est pourquoi il convient de considérer ce document comme un guide standard pour les ingénieurs du Temps Réel.

Je tiens à remercier tous les experts qui ont accepté de coopérer avec le BNI dans ce travail difficile et en premier lieu J.J. SIMON qui a montré la voie à suivre dans les études préliminaires, les membres du groupe opérationnel, Melle H. FALLOUR, MM C. FAULLE, J. FEBVRE, G. MEMMI, F. MINEL, D. VOJNOVIC, qui ont, pendant plus d'une année, forgé la meilleure synthèse de leur expérience, et les membres du groupe observateur, MM BONNARD, BUISSON, CHEVANCE, DERRINNIC, HANNE, HELEIN, HO, PONCET, MASSON, MAESTRACCI, SAVOYSKY, REMER, DOUSSY, OMNES, DESCLAUD qui ont par leurs judicieuses critiques permis une convergence plus rapide des discussions.
Je tiens également à remercier MMes M.C. MARAIS et F. TOURTE pour leur précieuse collaboration dans l'édition et la rédaction de cette proposition.

Mai 1980,

M. KRONENTAL, responsable du projet SCEPTRE

(*) Standardisation du Coeur des Exécutifs des Produits Temps Réel Européens.

INTRODUCTION : DESCRIPTION ET UTILISATION DU DOCUMENT

SOMMAIRE

1 PRESENTATION

Le présent document est une proposition de standard de noyau d'Exécutif Temps Réel.
L'essence de cette proposition est la spécification d'un ensemble de types et d'opérations élémentaires permettant de construire de façon structurée performante et portable les mécanismes de coopération entre les activités parallèles d'un Exécutif Temps Réel.
Cet ensemble est le noyau SCEPTRE.
Il est spécifié dans la première partie. Sa mise en oeuvre est décrite dans la seconde partie.

Pour comprendre ce document il est important d'avoir les idées directrices suivantes présentes à l'esprit :

- le domaine auquel nous nous intéressons est strictement limité au problème de la coopération (synchronisation, communication) entre activités parallèles;

- nous proposons une approche visant à construire de façon procédurale la plupart des mécanismes de coopération proposés dans les langages Temps Réel à partir d'un petit nombre de types d'objets et d'opérations élémentaires : le noyau SCEPTRE;

- le noyau SCEPTRE n'est pas lui-même portable : il doit être programmé en fonction des caractéristiques des machines-cible pour les utiliser au mieux;

- en revanche les programmes qui utilisent le noyau SCEPTRE ont, en ce qui concerne les parties relatives à la coopération entre activités parallèles, un taux de portabilité maximum.

- le noyau SCEPTRE sépare nettement trois classes de fonctions : la signalisation, l'exclusion mutuelle (appelées usuellement fonctions de synchronisation) et la fonction de communication.

La philosophie de ce document est exposée dans les paragraphes "préambule" et "exposé du problème".
Le paragraphe "concepts de base" fixe le vocabulaire et les concepts indispensables à la compréhension du document.
Les "spécifications fonctionnelles" décrivent les opérations du noyau SCEPTRE.
Les "spécifications logiques" en précisent la sémantique et les relations mutuelles.
Les algorithmes réalisant les mécanismes de coopération connus (sémaphores, monitors, rendez-vous,...) en fonction du noyau SCEPTRE sont explicités dans le paragraphe "bibliothèque de services de haut niveau".
Des exemples d'utilisation et d'implémentation du noyau SCEPTRE figurent dans la seconde partie.
Des outils de spécification de détail permettant de décrire précisément tout ou partie d'une application Temps Réel sont donnés en annexe.

2 CONTENU GLOBAL

Une application Temps Réel est un système de traitement de l'information ayant pour mission de commander un environnement imposé, en respectant les contraintes de temps et de débit (temps de réponse à un stimulus, taux de perte d'information toléré par entrées) qui sont imposées à ses interfaces avec cet environnement.
Une application Temps Réel est composée de programmes s'exécutant sur une machine gérée par un Exécutif.
Chacune de ces exécutions est mise au compte d'agents actifs appelés tâches.
Pour des raisons de stratégie ou de configuration matérielle les Exécutifs varient d'une application à l'autre. Il est donc utile de dégager un ensemble de mécanismes communs à ces Exécutifs permettant de les construire par composition au moyen d'un langage de programmation adéquat.

Le noyau SCEPTRE est l'ensemble de ces mécanismes communs qui facilite le contrôle des tâches, leur coopération et leur communication avec l'environnement de l'application.
La littérature contient de nombreuses propositions de mécanismes de haut niveau permettant la programmation structurée de la coopération entre tâches. La figure qui suit en illustre l'évolution.
L'expérience montre qu'en Temps Réel, les mécanismes de haut niveau sont souvent inadéquats : les automatismes qu'ils fournissent (par exemple mise en file d'attente implicite) s'avèrent ou bien inefficaces ou trop rigides (ordre d'attente imposé, incompatibilité avec l'application). Par ailleurs ces mécanismes imposent une architecture à partage d'information (donc à mémoire commune); avec les rendez-vous (3, 4) il s'agit d'une architecture sans partage d'information et fondée sur la communication entre tâches.

Pour éviter ces inconvénients le noyau SCEPTRE se place à un niveau proche de celui des mécanismes élémentaires fournis par les machines actuellement utilisées en Temps Réel. Ses opérations ne comportent aucun automatisme implicite contraignant. Elles sont nettement différenciées dans leurs fonctionnalités permettant une grande variété de combinaisons et laissant aux concepteurs la possibilité de construire l'Exécutif le mieux adapté aux stratégies de leur application.
Elles présentent un bon compromis généralité/performance et fournissent ainsi une base à la portabilité des Exécutifs.
L'expérience montre que l'emploi de ces opérations de bas niveau dans la programmation du parallèlisme est délicat. Il convient donc de le limiter aux cas où cela est absolument nécessaire.
C'est pourquoi nous accompagnons cette proposition de noyau d'une bibliothèque dans laquelle les algorithmes des mécanismes de haut niveau les plus utilisés sont exprimés au moyen des opérations du noyau SCEPTRE.

EVOLUTION DES PRIMITIVES DE SYNCHRONISATION

NIVEAU	ELEMENT	OPERATIONS	CARACTERISTIQUES	DATE	REFERENCE
MACHINE	INTERRUPTION	MASQUER, DEMASQUER	signalisation, exclusion mutuelle		
	ELEMENT BINAIRE	TEST-AND-SET, RESET	exclusion mutuelle de base		
BAS	VERROU	VERROUILLER, DEVERROUILLER	exclusion mutuelle	avant 1965	(7)
MOYEN	SEMAPHORE	P,V	exclusion mutuelle avec file d'attente implicite	1965	(21)
	SEMAPHORE PRIVE	P,V	signalisation		
	EVENEMENT	ATTENDRE, SIGNALER	signalisation avec file d'attente implicite	1965	(21)
	SECTION CRITIQUE	ENTRER, SORTIR implicites	exclusion mutuelle avec file d'attente implicite	1970	(21)
HAUT	MONITEUR	ENTRER, SORTIR implicites ATTENDRE, SIGNALER	exclusion mutuelle, signalisation, avec files d'attentes implicites	1972	(1)
TRES HAUT	SECTION CRITIQUE CONDITIONNELLE			1970	(21)
	"PATH EXPRESSION"			1974	(26)

3 TRAVAUX VOISINS

Des propositions voisines ont été analysées et comparées en (28). Citons notamment le standard britannique MASCOT (6) et les travaux du TC 8 (7) de l'EWICS (ex-Purdue Europe). Leur approche est essentiellement différente de la nôtre en ce sens qu'ils privilégient un mécanisme de synchronisation particulier :

- la "control queue" de MASCOT,
- le "sémaphore à message" dans le rapport du TC 8,

avec les inconvénients déjà mentionnés.

L'approche SCEPTRE se démarque de ces travaux en ce sens qu'elle fournit une base commune d'opérations pour la construction de tous les mécanismes de coopération de haut niveau sans en privilégier aucun.

Notons pour éviter toute confusion que le projet portabilité de la CEE visant à définir une interface commune de système d'exploitation n'a que très peu de relations avec la présente proposition :

- il ne privilégie pas les problèmes liés au Temps Réel;
- il veut définir une interface située au dessus des systèmes d'exploitation fournis par les constructeurs, alors que le noyau SCEPTRE se veut être une base de construction des Exécutifs Temps Réel.

PREMIERE PARTIE : SPECIFICATION DU NOYAU SCEPTRE

SOMMAIRE

1 - EXPOSE DU PROBLEME

SOMMAIRE

1.1 APPLICATIONS et EXECUTIFS TEMPS REEL

1.1.1 Caractéristiques des applications Temps Réel

Les applications Temps Réel réalisées actuellement présentent les caractéristiques suivantes :

- grande diversité des dispositifs d'Entrée/Sortie

- temps de réponse souvent très courts

- distribution des activités parallèles pour une plus grande disponibilité locale

- contraintes de sécurité de fonctionnement très sévères

- matériels utilisés de plus en plus diversifiés

- contraintes correspondantes liées à la production, à l'adaptation et à la maintenance des programmes.

1.1.2 Caractéristiques des outils de programmation pour le Temps Réel

De nombreux efforts sont actuellement faits pour éviter l'usage d'outils de programmation de bas niveau encore trop répandus.
Les langages Temps Réel, LTR (14), PEARL (16), font l'objet d'une norme nationale. Real-Time FORTRAN (13) fait l'objet d'une norme ISO. Le langage ADA (22) permettra la programmation des applications Temps Réel en conservant les avantages de PASCAL. Sa spécification définitive n'est pas encore fixée à cette date.
Ces langages supposent un Exécutif qui leur est propre, ce qui gêne considérablement :

. la réutilisation de logiciels opérationnels écrits dans des langages différents,

. la gestion de configurations spécifiques (système distribué, périphériques spéciaux,...).

Ils n'apportent pas de solution commode aux problèmes des entrées/sorties Temps Réel et ne sont pas universellement admis quant aux mécanismes de gestion des activités parallèles qu'ils permettent de créer.

1.1.3 Caractéristiques des Exécutifs Temps Réel

Les Exécutifs Temps Réel fournissent des services multiples :

. Cacher les propriétés spécifiques des machines utilisées pour n'en laisser voir que les propriétés logiques afin d'améliorer :

- la compréhension des programmeurs
- l'emploi de langages évolués
- l'adaptabilité et la portabilité des applications

. Fournir une interface unique entre parties d'application écrites dans des langages différents

. Assurer les communications avec l'environnement de l'application (entrées/sortie)

. Gérer les ressources fournies par le matériel pour le compte des activités parallèles (mémoire principale, mémoires secondaires, processeurs)

Cette multiplicité de services se complique encore par :

. la nécessité de gérer des configurations fort diverses (interfaces de communication spéciales, configurations distribuées, configurations à haute sécurité,...)

. la nécessité économique de vendre un Exécutif Temps Réel capable de réaliser des services :

- proches du Temps Réel : informatique transactionnelle, télétraitement,...
- généraux : gestion des fichiers, gestion des textes,...

1.1.4 Problèmes liés au contrôle des activités parallèles

L'une des difficultés de la programmation des applications Temps Réel est encore le contrôle des activités parallèles, de leurs interactions mutuelles et de leur interaction avec l'environnement extérieur.
Dans ce domaine, les outils proposés, soit dans la littérature, soit dans les langages de programmation expérimentaux se succèdent avec une telle rapidité qu'ils sont condamnés à l'obsolescence avant d'avoir été scientifiquement évalués.

L'absence de standards en ce domaine oblige les réalisateurs :

- soit à modifier l'Exécutif fourni par le constructeur

- soit à réaliser complètement l'Exécutif correspondant à leurs besoins

ceci avec d'énormes problèmes de portabilité de l'application construite.

1.2 APPROCHE PRECONISEE

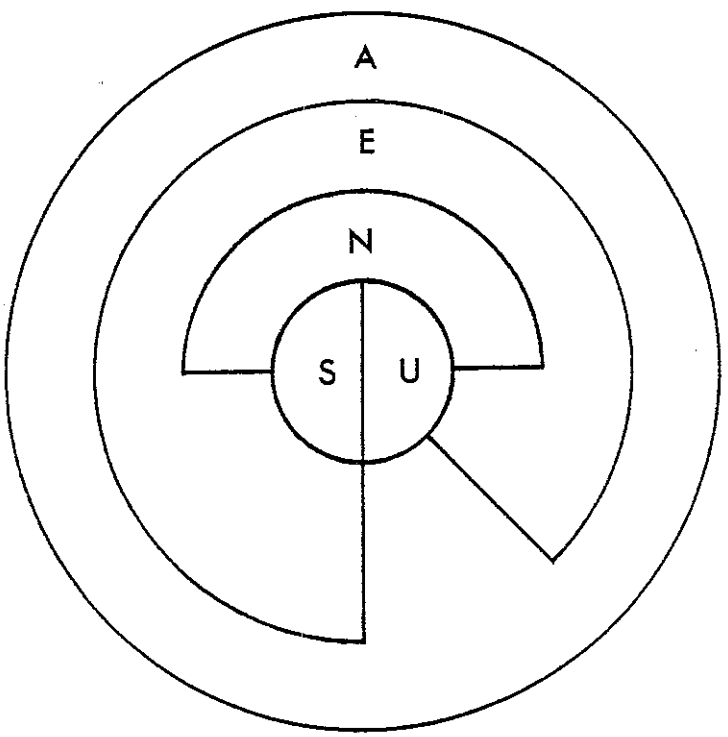
Compte-tenu des difficultés exposées ci-dessus, la réalisation d'un Exécutif Temps Réel performant, sûr, adaptable, peu encombrant et satisfaisant une majorité d'utilisateurs semble une gageure.
A l'opposé, il est peu économique que chaque Application Temps Réel possède son propre Exécutif.
Dans cette situation, un moyen terme possible est d'admettre que les Exécutifs Temps Réel soient construits "à la carte" à partir d'un ensemble de modules paramétrés (cablés ou programmés), que l'on assemble en fonction des besoins.
Cet idéal est impossible à atteindre lorsque l'ensemble des fonctions de l'Exécutif n'est pas définitivement figé. Dans le cas du Temps Réel on peut fixer l'ensemble des services que l'Exécutif doit fournir.
Il s'agit de :

1. la communication
2. la synchronisation
3. la gestion et l'ordonnancement des tâches
4. la gestion de la mémoire
5. la gestion des interruptions et les E/S physiques
6. les E/S logiques et la gestion des périphériques
7. la gestion des fichiers et des supports
8. la gestion des programmes
9. la gestion des travaux et des transactions
10. le traitement des erreurs et des exceptions
11. la gestion du temps.

On sait pouvoir réaliser de telles fonctions à partir des instructions de toute machine et de mécanismes tels que :

- invocation de sous-programme par commutation de contexte,
- masquage et démasquage des interruptions,
- adressage des interfaces d'entrée/sortie,
- déroutements,...

L'emploi direct de ces mécanismes est un obstacle sérieux à la programmation de l'Exécutif en langage de haut niveau. Il fournit un produit difficile à adapter et impossible à porter sur une configuration voisine.
Pour éviter ces inconvénients, il convient de minimiser les parties de l'Exécutif réalisées en langage machine afin de ramener la réalisation de l'Exécutif à la programmation d'une "machine abstraite", plus générale et plus régulière que la machine.
Dans le domaine qui nous intéresse, il n'est pas utile de remplacer la totalité de la machine par une machine abstraite : en effet, la programmation en langage évolué y pourvoit en ce qui concerne la programmation séquentielle.
Il suffit donc de concevoir et de construire la partie de machine abstraite qui n'est pas fournie par le langage.
On appelle noyau cette partie. Le noyau est un ensemble d'opérations et de types d'objets utilisables à partir d'un langage évolué au travers d'une interface procédurale appropriée.
Ceci préconise une méthode de découpage des applications en niveaux logiques.
Au niveau application A, ne sont disponibles que les opérations de l'Exécutif et de la machine en mode utilisateur.
Au niveau Exécutif E, ne sont disponibles que les opérations du noyau, les opérations de la machine en mode superviseur S et en mode utilisateur U.
Au niveau noyau N, ne sont disponibles que les opérations de la machine en mode utilisateur U et en mode superviseur S.
Ces règles de visibilité se traduisent par le schéma :



Tout le problème de ce découpage est de dégager les critères qui déterminent le noyau et l'Exécutif.
Ceci est l'objet des deux paragraphes qui suivent.

1.3 OBJECTIFS VISES

Dans l'ensemble des services fixé ci-dessus, le noyau SCEPTRE doit permettre aux applications et à l'Exécutif d'atteindre les objectifs suivants :

. Adéquation aux besoins des utilisateurs : l'emploi du noyau SCEPTRE doit faciliter la réalisation des fonctions de coopération entre activités parallèles, prévues au cahier des charges d'une application Temps Réel.

. Sécurité : L'application doit être autant que possible exempte de pannes ou d'anomalies de fonctionnement. L'utilisation du noyau doit favoriser la structuration des programmes Temps Réel en fournissant les concepts et les outils qui permettent d'isoler les traitements séquentiels des traitements relatifs à la coopération entre activités parallèles.

. Diminution des coûts et des délais de réalisation et de maintenance : l'utilisation explicite d'un modèle clair et des concepts simples fournis par le noyau, doit permettre une plus grande facilité de conception, de réalisation et de maintenance ou de modification du logiciel, les coûts et durées correspondants s'en trouvant allégés.

. Performance : les temps de réponse à certaines sollicitations extérieures doivent être maintenus inférieurs à certains seuils. L'utilisation du noyau doit réduire les temps d'exécution des services et permettre de diminuer la taille des applications.

. Portabilité : les mécanismes du noyau doivent fournir une base à la portabilité des Exécutifs et des Applications si leur emploi dans un langage portable est assuré par une interface spécifique associée à ce langage.

. Adaptabilité : la modularisation des applications permet le remplacement de certains composants logiciels par des composants matériels moins coûteux ou plus fiables. L'emploi du noyau doit favoriser de telles adaptations. Il doit également favoriser la migration de ces composants vers des sites éloignés dans le cas d'un système distribué.

. Modularité : le noyau doit servir de base à la modularité des systèmes en assurant la coordination entre des fonctions éventuellement réalisées dans différents langages et les composants matériels des machines utilisées. Ceci doit uniformiser les mécanismes de coopération entre les divers composants d'une application et accroît sa modularité et donc son extensibilité à des fonctions nouvelles.

1.4 CRITERES DE CONCEPTION DU NOYAU SCEPTRE

La conception du noyau SCEPTRE répond aux critères suivants :

. Minimalité : le nombre des mécanismes du noyau doit être aussi restreint que possible.

. Généralité : ils doivent s'appliquer uniformément à des configurations matérielles diverses avec la même signification.

. Performance : leur implémentation sur la majorité des machines utilisées en Temps Réel doit fournir le meilleur compromis généralité/performance.

. Simplicité : la simplicité requise doit permettre d'éviter toute ambiguité sémantique.

. Constructibilité : les mécanismes du noyau doivent permettre la construction de mécanismes de plus haut niveau ayant des fonctions analogues, mais présentant une plus grande sécurité d'emploi.

. Indépendance : les mécanismes du noyau doivent être indépendants de tout langage de programmation et de toute machine.

2 - CONCEPTS DE BASE

SOMMAIRE

2.1 INTRODUCTION

Les définitions qui suivent sont indispensables à la compréhension du document. Elles introduisent les entités concernant le noyau SCEPTRE et précisent leurs relations mutuelles, indépendamment des choix de découpage qui constituent l'essence du noyau SCEPTRE.
Ces entités sont la machine, la tâche, l'ordonnanceur et l'agence.

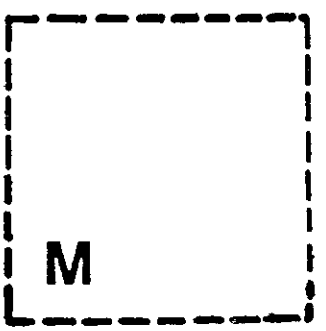
2.2 NOTION DE MACHINE

Une machine est un environnement matériel assurant l'exécution d'une famille de processus cablés, microprogrammés ou programmés, qui partagent en commun une mémoire et/ou des registres.

Une machine possède un ou plusieurs processeurs assurant l'exécution de ces processus.

On représente graphiquement une machine par le schéma :

machine M

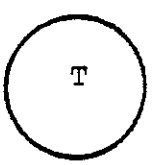


2.3 NOTION DE TACHE

Une tâche est un agent actif responsable de l'exécution par une machine d'un programme composé à partir du répertoire des instructions de cette machine. Une tâche ne peut entreprendre l'exécution d'une instruction de ce programme qu'après avoir terminé l'exécution de l'instruction précédente. En ce sens elle est séquentielle.

On représente graphiquement une tâche par le schéma :

tache T



Une tâche possède un nom et des attributs caractéristiques :

. de la machine sur laquelle elle s'exécute :

- registres, mot d'état, segments mémoire...

. de la façon dont on la gère :

- priorité

- descripteur référençant toutes les informations nécessaires à son exécution (code, constantes, pile, données,...)

On appelle contexte d'exécution d'une tâche l'ensemble des informations strictement nécessaires à un processeur pour :

- en entreprendre l'exécution,

- en reprendre l'exécution après une suspension momentanée.

Sur la plupart des machines, le contexte d'exécution comprend les registres, les bases et le mot d'état.
Suivant les implémentations une tâche peut être programmée, microprogrammée ou même cablée (cas des tâches gérant les unités d'échange ou les coupleurs d'entrée/sortie).
Lorsqu'une tâche est programmée ou microprogrammée, elle possède des données locales qu'elle est seule à manipuler.

2.4 NOTION D'ORDONNANCEUR

On appelle ordonnanceur un module (cablé, microprogrammé ou programmé) chargé de gérer un ensemble de processeurs identiques pour le compte d'une famille de tâches.
Cette gestion consiste à :

. attribuer les processeurs (libres ou libérés) aux tâches prêtes :
- à commencer leur exécution,
- à continuer leur exécution,

. reprendre le processeur d'une tâche :
- à la demande explicite de celle-ci (attente d'une condition),
- inconditionnellement (préemption),

suivant un algorithme de choix spécifique.

Exemples de critères de choix : la priorité des tâches, la proximité d'échéances temporelles.

2.5 NOTION D'AGENCE

On appelle agence un module (microprogrammé ou programmé) fournissant une collection d'opérations spécialisées aux programmes utilisateurs.
Une agence peut exécuter ses opérations au moyen de tâches, de programmes et de données qui lui sont propres.

Exemples :

. Agence de gestion mémoire fournissant les opérations

obtenir-segment
libérer-segment

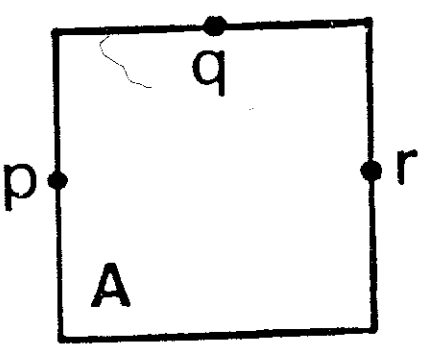
. Agence de gestion des fichiers séquentiels fournissant les opérations

créer
détruire
assigner
ouvrir
fermer
lire
écrire

On représente graphiquement une agence par le schéma:

agence A

(AVEC SES OPERATIONS P, Q, R)



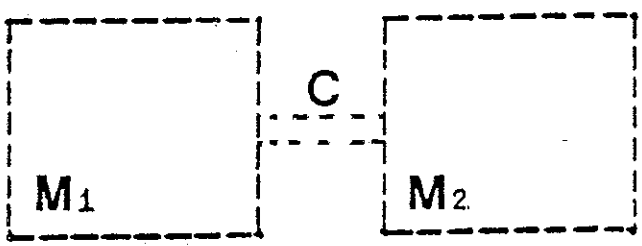
2.6 RELATIONS MUTUELLES

2.6.1 Relation entre machines

Les machines communiquent entre elles au moyen de canaux qui fournissent le support matériel à la transmission d'information entre tâches.

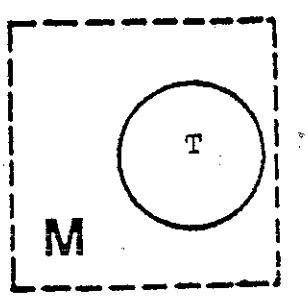
On représente graphiquement un canal par le schéma :

canal C



2.6.2 Relation entre tâche et machine

Une tâche appartient à une machine tout au long de son existence, ce que l'on représente graphiquement par :



Deux tâches appartenant à la même machine sont dites voisines. Dans le cas contraire on dit qu'elles sont distantes.

2.6.3 Relations entre tâches

CREATION
La création d'une tâche consiste à associer à un nouveau nom de tâche tous les objets nécessaires à l'exécution de cette tâche (place mémoire, code, donnée, pile...)
DESTRUCTION
La destruction d'une tâche consiste à effacer l'association précédente.

DEMARRAGE
Le démarrage d'une tâche consiste à en commencer l'exécution.

ARRET
L'arrêt d'une tâche consiste à arrêter définitivement son exécution.

COMMUNICATION
La communication entre deux tâches consiste en un échange d'informations entre ces deux tâches suivant un protocole. Le protocole fixe la forme que doit avoir cet échange pour que les deux tâches concernées se "comprennent" correctement.
L'intermédiaire assurant la communication entre les deux tâches est un système de communication. Le système de communication est responsable du bon acheminement des informations.
La communication entre deux tâches se réalise d'autant plus simplement que ces tâches sont voisines : l'utilisation de la mémoire commune permet de réduire le protocole de communication à des opérations de synchronisation et des manipulations de données partagées contenues dans la mémoire commune.

SYNCHRONISATION
La synchronisation des actions entreprises par une famille de tâches a pour rôle l'ordre d'exécution de ces actions.
Un mécanisme de synchronisation est un moyen par lequel une tâche peut influer sur l'ordre des actions entreprises par les tâches avec lesquelles elle interagit.
Il est commode d'expliciter un protocole de synchronisation en précisant la condition qui doit être remplie pour qu'une tâche puisse entreprendre une action soumise à ce protocole.
Dans le cas où cette condition est satisfaite l'action est entreprise, dans le cas contraire, la tâche doit attendre que cette condition soit satisfaite.
En général, une condition attendue est évaluée par un agent actif (tâche ou environnement de l'application). Lorsque cette évaluation donne un résultat positif, cet agent signale à la tâche en attente que la condition est satisfaite. Celle-ci peut alors entreprendre son action.

SIGNALISATION

Un mécanisme de signalisation est un moyen permettant à une (ou plusieurs) tâche(s) d'attendre qu'une condition associée à ce mécanisme soit satisfaite, et à un agent actif évaluant cette condition de la signaler lorsqu'elle est satisfaite. La signalisation permet d'exprimer des protocoles de synchronisation.

EXCLUSION MUTUELLE

Un mécanisme d'exclusion mutuelle est un moyen d'assurer que les exécutions d'une famille de séquences d'instructions soient disjointes dans le temps quel que soit l'ordre dans lequel ces séquences sont invoquées.

2.6.4 Relation entre tâche et ordonnanceur

Une tâche gérée par un ordonnanceur peut se mettre en attente d'une condition sous la responsabilité de cet ordonnanceur. Celui-ci lui retire son processeur lorsque la condition attendue n'est pas satisfaite. Il lui en attribue un dès que cette condition est satisfaite, compte-tenu de son algorithme de choix.
Pour matérialiser ces relations on introduit les états de tâche suivants :

En cours : état d'une tâche qui s'exécute sur un processeur.

Prêt : état d'une tâche dont toutes les conditions d'exécution sont satisfaites mais qui ne possède pas de processeur pour s'exécuter.

En attente : état d'une tâche en attente d'une condition non satisfaite.

PREEMPTION DU PROCESSEUR D'UNE TACHE PAR UN ORDONNANCEUR

Lorsqu'un ordonnanceur reçoit une demande visant à mettre une tâche dans l'état prêt il a la possibilité (en fonction de son algorithme de choix et des attributs de cette tâche) de prendre le processeur d'une tâche en cours pour l'attribuer à cette tâche.
Cette action s'appelle préemption du processeur.

TACHE IMMEDIATE

Une tâche immédiate est une tâche programmée gérant une interface entre l'application et son environnement.

Exemples :
- interface d'Entrée/Sortie,
- interface avec une horloge externe.

Les tâches immédiates sont conçues pour avoir le temps de réponse le plus court compte tenu de l'état d'exécution de l'application. En conséquence elles ne se mettent jamais en attente d'une condition : elles ne passent donc jamais dans l'état En attente.

Une tâche immédiate est démarrée par un signal apparaissant à l'interface associée. Son achèvement correspond à l'acquittement de ce signal.

TACHE DIFFEREE

Une tâche différée est une tâche qui peut se mettre en attente d'une condition compte tenu de contraintes de temps de réponse moins critiques que pour une tâche immédiate.

ORDONNANCEURS

Les machines actuelles sont généralement gérées par deux types d'ordonnanceurs :
- l'ordonnanceur des tâches immédiates (dit ordonnanceur cablé)
- l'ordonnanceur des tâches différées (dit ordonnanceur programmé)

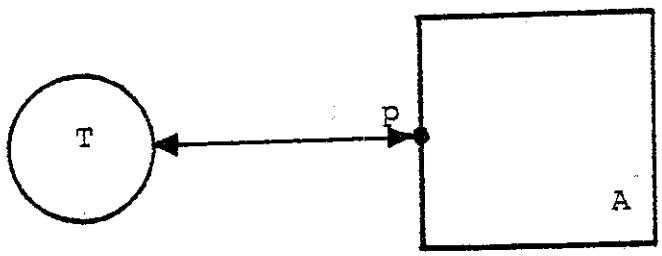
L'ordonnanceur des tâches immédiates correspond au mécanisme d'interruption. L'algorithme de choix de cet ordonnanceur est le plus souvent basé sur les priorités associées aux interruptions.

L'ordonnanceur des tâches différées gère les tâches suivant un algorithme de choix qui est imposé par le type de l'Exécutif à réaliser :

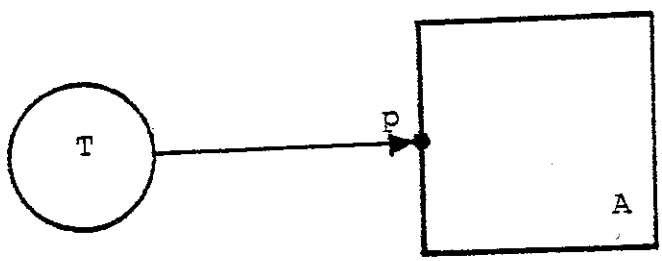
- . préemptif : le passage d'une tâche T dans l'état prêt provoque la préemption du processeur de toute tâche ayant une priorité inférieure à celle de T,
- . non préemptif,
- . à partage de temps.

Les conflits entre ces deux ordonnanceurs sont régis par le mécanisme de préemption du processeur (le plus souvent associé aux interruptions) et par les opérations d'autorisation et d'inhibition de cette préemption.

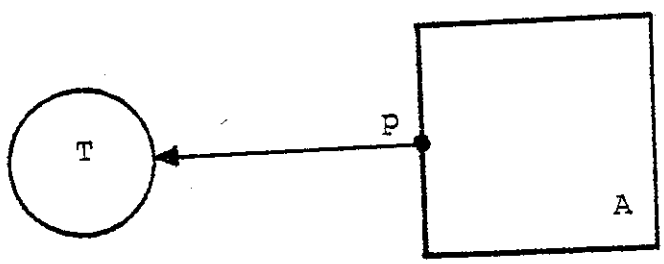
2.6.5 Relation entre tâche et agence



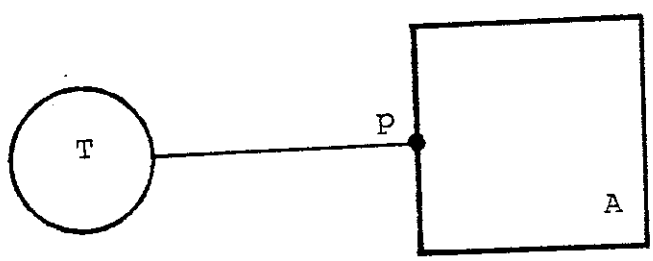
T invoque p en fournissant des arguments donnés et en obtenant des arguments résultats.



T invoque p en fournissant des arguments donnés



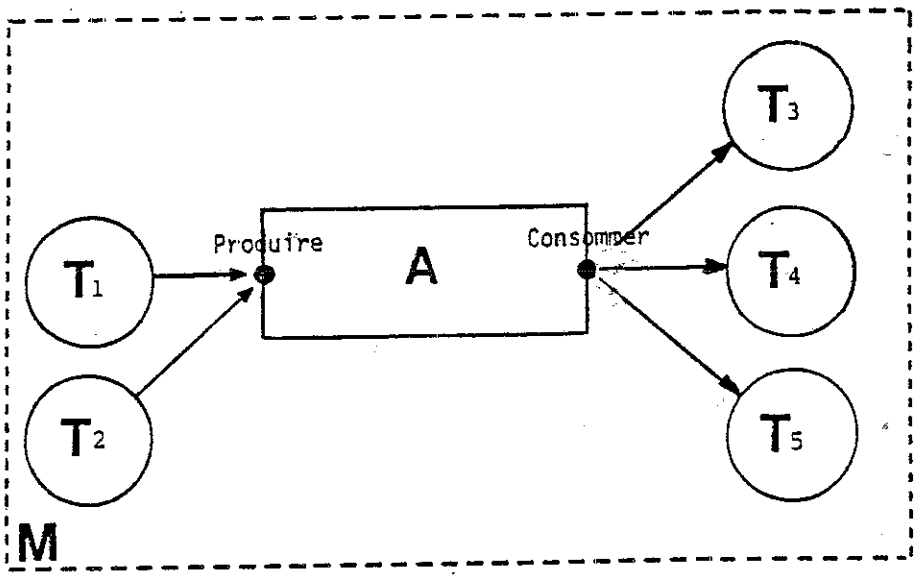
T invoque p en obtenant des arguments résultats



T invoque p sans transmission d'argument

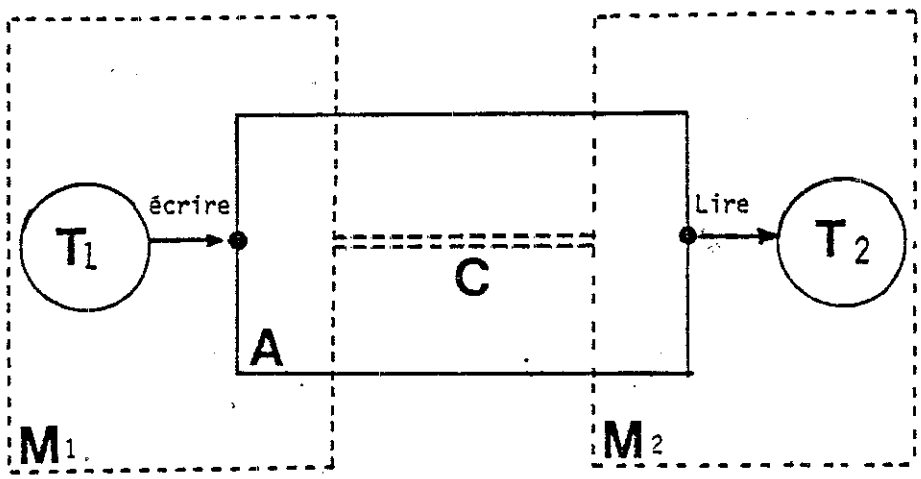
EXEMPLES D'UTILISATION

GESTION D'INFORMATION PARTAGEE



Dans cet exemple les tâches T1 et T2 produisent une information "consommable" et l'envoient à l'agence A en invoquant la procédure "produire".
Les tâches T3, T4, T5 demandent à consommer une information produite en invoquant la procédure "consommer".

ENTREE-SORTIE : AGENCE DISTRIBUEE



La communication entre T1 et T2 s'effectue au travers de l'agence "distribuée" A gérant le canal C lors des invocations des procédures "écrire" et "lire".

3 - SPECIFICATIONS FONCTIONNELLES DU NOYAU SCEPTRE

SOMMAIRE

3.1 CARACTERISATION DU NOYAU

3.1.1 Caractérisation fonctionnelle du noyau SCEPTRE

Dans l'ensemble des services que doit réaliser un Exécutif Temps Réel déjà mentionné ci-dessus :

1. la communication
2. la synchronisation
3. la gestion et l'ordonnancement des tâches
4. la gestion de la mémoire
5. la gestion des interruptions et les E/S physiques
6. les E/S logiques et la gestion des périphériques
7. la gestion des fichiers et des supports
8. la gestion des programmes
9. la gestion des travaux et des transactions
10. le traitement des erreurs et des exceptions
11. la gestion du temps,

le noyau SCEPTRE se limite aux fonctions élémentaires des domaines 1, 2, 3, 5, à l'exclusion des fonctions :

- de désignation et d'adressage des objets,
- de création et de destruction des objets,
- d'accès entre entités appartenant à des machines distinctes.

Ces fonctions relèvent soit de l'outil de programmation séquentielle, soit de la gestion des mémoires. Dans tous les cas, elles peuvent être spécifiées et réalisées en termes d'agences employant le noyau SCEPTRE.
Ce choix est conforme aux critères de minimalité, généralité, simplicité et constructibilité du paragraphe 1. Il suppose que les mécanismes d'accès aux objets sont :

- fournis a priori indépendamment du noyau SCEPTRE,
- ou bien réalisés sans recourir aux mécanismes fournis par le noyau SCEPTRE.

3.1.2 Présentation générale du noyau SCEPTRE

Le noyau SCEPTRE fournit des opérations élémentaires permettant de réaliser efficacement les fonctions suivantes :

- ordonnancement,
- signalisation,
- exclusion mutuelle,
- communication.

A chacune de ces fonctions le noyau SCEPTRE associe un type d'objet :

- TACHE pour identifier les tâches,
- EVENEMENT pour mémoriser le signal marquant qu'un condition est devenue vraie,
- REGION pour caractériser la possession exclusive d'un objet partagé,
- FILE pour ranger les informations provenant de différentes tâches, et les prendre en compte dans leur ordre d'arrivée.

Chaque noyau SCEPTRE se fonde sur un ordonnanceur propre dont l'algorithme de choix est unique (FIFO selon les priorités des tâches) mais qui peut être préemptif ou non selon l'implémentation.

3.2 OBJETS MANIPULES PAR LE NOYAU

Les objets manipulés par les opérations du noyau sont de l'un des types TACHE, EVENEMENT, REGION et FILE.

- Un objet de type TACHE désigne l'une des tâches gérées par le noyau.

- Un objet de type EVENEMENT contient une marque représentant le fait qu'une condition attendue par une tâche est devenue vraie.

- Un objet de type REGION contient une marque représentant le fait qu'un objet partagé est ou non possédé par une tâche.

- Un objet de type FILE est une structure permettant de classer des informations transitoires suivant un ordre qui est celui de leur arrivée dans la FILE. Cet ordre peut être modulé en fonction d'un critère de priorité associé à chacune de ces informations.

3.3 OPERATIONS DU NOYAU

La liste des opérations ci-dessous constitue l'ensemble des opérations élémentaires fournies par le noyau SCEPTRE.

3.3.1. Opérations de gestion des tâches

DEMARRER(TACHE) : lance l'exécution de TACHE

ARRETER(TACHE) : ordre d'arrêt d'exception de TACHE

SE TERMINER : terminaison de la tâche courante

CHANGER-PRIORITE(TACHE, NOUVELLE-PRIORITE)

et les fonctions :

ETAT(TACHE)

PRIORITE(TACHE) : entier

TACHE-COURANTE : TACHE qui exécute l'instruction courante.

3.3.2 Opérations de manipulation des événements

ATTENDRE (liste d'événements appartenant à TACHE-COURANTE)

SIGNALER(EVENEMENT, TACHE)

EFFACER (liste d'événements appartenant à TACHE-COURANTE)

et le prédicat

ARRIVE (liste d'événements appartenant à TACHE-COURANTE)
qui est vrai lorsque tous les événements de la liste ont été signalés.

3.3.3. Opérations de manipulation des régions

ENTRER(REGION)

SORTIR(REGION)

3.3.4. Opérations de manipulation des files

ENVOYER(élément,FILE(,critère))

RETIRER(élément, FILE(,critère))

et les prédicats

VIDE(FILE)
PLEINE(FILE)

3.4 PROPRIETES COMMUNES AUX OPERATIONS DU NOYAU

3.4.1 Propriété d'exclusivité

Les opérations du noyau ayant en paramètre un objet de type identique sont exclusives entre elles, c'est à dire qu'à un moment donné une opération au plus peut être en cours d'exécution.

3.4.2 Propriété d'atomicité

Les opérations du noyau sont atomiques c'est à dire qu'un processeur exécutant une opération du noyau ne peut être préempté au profit d'une autre tâche.

3.4.3 Conséquence

Ce besoin d'une exclusion mutuelle très efficace de bas niveau peut conduire à des restrictions quant aux types de matériel qui peuvent supporter SCEPTRE et constituer un obstacle à la réalisation de l'objectif d'indépendance vis à vis du matériel.

	DEM	ARR	SE-TER	SIG	ATT	EFF	ENT	SOR	ENV	RET
DEMarrer	+	+	+	+	-	-	-	-	-	-
ARRêter	+	+	+	+	-	-	-	-	-	-
SE-TERminer	+	+	+	+	-	-	-	-	-	-
SIGnaler	+	+	+	+	+	+	-	-	-	-
ATTendre	-	-	-	+	+	+	-	-	-	-
EFFacer	-	-	-	+	+	+	-	-	-	-
ENTrer	-	-	-	-	-	-	+	+	-	-
SORtir	-	-	-	-	-	-	+	+	-	-
ENVoyer	-	-	-	-	-	-	-	-	+	+
RETirer	-	-	-	-	-	-	-	-	+	+

TABLEAU RECAPITULATIF DES REGLES D'EXCLUSION

LEGENDE + signifie exclut
- signifie n'exclut pas
les noms des opérations sont abrégés horizontalement en ne conservant que les lettres majuscules données verticalement.

3.5 SECURITE DES OPERATIONS DU NOYAU

Les opérations du noyau peuvent donner lieu à des cas d'erreur. Le mécanisme permettant de notifier ces cas d'erreur à la tâche appelante dépend uniquement de l'implémentation retenue :

- compte-rendu systématique comme dans Real-time FORTRAN

- exceptions comme dans ADA.

Toutes les opérations du noyau SCEPTRE donnent lieu à un cas d'erreur unique dans le cas où l'un des arguments utilisés dans une invocation n'est pas défini (objet inexistant ou objet dans un état incohérent).

3.6 COMPOSITION DES OPERATIONS DU NOYAU

. Les opérations du noyau manipulant explicitement les mêmes objets garantissent l'accès exclusif à ces objets.

. Cette propriété n'est pas assurée pour des séquences d'opérations du noyau. Pour rendre exclusive et atomique une telle séquence (et former ainsi une section critique), il faut la parenthèser par

ENTRER(R)...SORTIR(R)

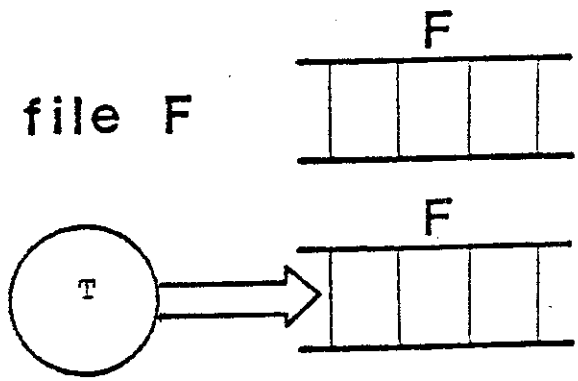
R étant un objet de type REGION à associer avec l'ensemble des objets partagés manipulés par la séquence. Une telle section critique peut inclure les invocations de toute opération du noyau à l'exception de celles qui peuvent entrainer la préemption du processeur courant.

. Dans le cas où l'on recherche une efficacité particulière, il est souhaitable de remplacer une telle section critique par la juxtaposition des corps des opérations du noyau optimisée grâce à la connaissance du contexte utilisant cette section critique.
Un tel sous-programme est appelé sous-programme-noyau.

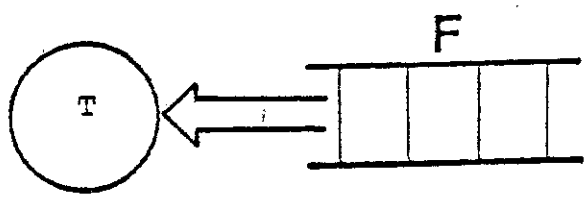
. En dehors de ces règles la composition procédurale fournie par le langage de programmation utilisé convient parfaitement (voir les remarques du paragraphe 1 de la seconde partie).

3.7 UTILISATION DES CONCEPTS ET OBJETS SCEPTRE POUR LES SPECIFICATIONS D'ENSEMBLE

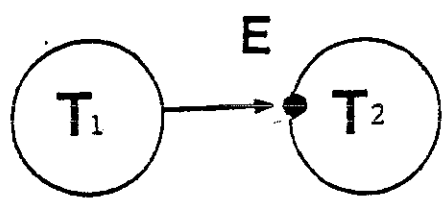
Les concepts introduits par le noyau SCEPTRE sont suffisamment généraux pour spécifier globalement le comportement d'un ensemble de tâches. Ces concepts peuvent avoir des réalisations diverses cablées, microprogrammées ou programmées.
Pour éviter toute ambiguité on précisera le cas échéant à quelle réalisation du noyau on se réfère.
Dans le graphisme permettant de représenter tout ou partie d'une application on ajoute aux symboles déjà définis dans le paragraphe 2, les symboles suivants :



Envoi d'un élément dans la file F par la tâche T



Retrait d'un élément de la file F par la tâche T



Attente de l'événement E appartenant à la tâche T2
et signalisation de E par la tâche T1.

4 - SPECIFICATIONS LOGIQUES DU NOYAU SCEPTRE

SOMMAIRE

4.1 METHODE DE SPECIFICATION

Les opérations du noyau étant exclusives et atomiques, il convient pour les spécifier d'indiquer :

- quelle est la condition éventuelle qui autorise le démarrage de leur exécution,
- quel est l'état initial des objets manipulés par l'opération lors de ce démarrage,
- quel est l'état final de ces mêmes objets à la fin de cette exécution,
- quels sont les cas d'erreur détectés par l'opération.

En utilisant la notion de transition développée en annexe on peut assimiler une telle spécification à celle d'une transition.

On trouvera dans cette annexe des exemples de spécification des opérations du noyau et de la bibliothèque SCEPTRE qui utilisent cette notion.

4.2 GESTION DES TACHES

4.2.1 DOMAINE NOYAU

Le noyau SCEPTRE est le seul domaine où sont accessibles les informations matérialisant l'état des tâches et des processeurs (contexte d'éxécution, descripteurs,...) gérés par le noyau. A l'extérieur de ce domaine, on ne peut influer sur l'éxécution d'une tâche qu'en la désignant par son nom lors de l'invocation d'une opération du noyau.

4.2.2 LE TYPE TACHE

Les objets de type TACHE désignent les taches gérées par le noyau SCEPTRE.

4.2.3 ETATS D'UNE TACHE

Pour faciliter la compréhension des opérations du noyau SCEPTRE on introduit des états qui permettent de caractériser la façon dont les tâches gérées par le noyau s'exécutent.

Les termes suivants définissent soit des états, soit des ensembles imbriqués d'états de tâche.

Inexistant : il n'y a pas de descripteur associé à la tâche.

Existant : la tâche possède un descripteur défini.

Non-exécutable : la tâche possède un descripteur défini, mais elle ne peut

- soit démarrer son exécution,
- soit la continuer.

Exécutable : la tâche possède un descripteur défini, et elle peut s'exécuter.

Hors-service : la tâche est exécutable (au sens précédent) mais on n'a pas encore demandé son démarrage ou bien son exécution est terminée.

En-service : la tâche est exécutable, a commencé son exécution et ne l'a pas terminée.

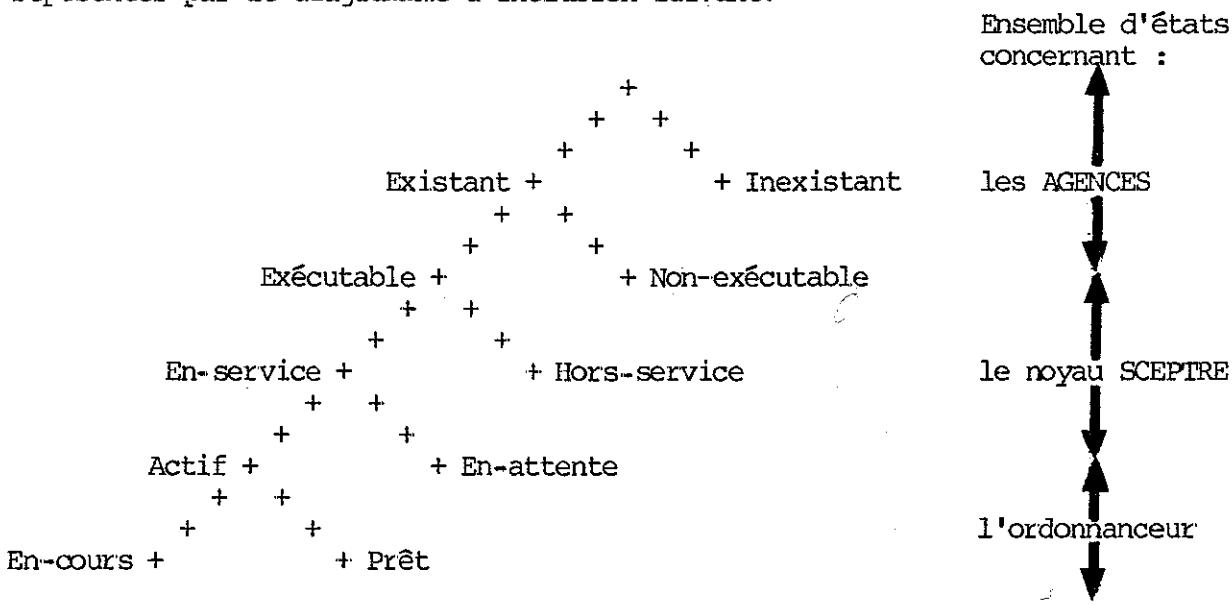
En-attente : la tâche est en service et attend qu'une condition explicite soit satisfaite pour continuer son exécution.

Actif : la tâche est en service, n'attend aucune condition sauf peut être qu'un processeur soit libre pour qu'elle puisse s'exécuter.

Prêt : la tâche est active (au sens précédent) et elle attend qu'un processeur soit libre pour qu'elle puisse s'exécuter.

En-cours : la tâche est active et elle dispose d'un processeur pour s'exécuter.

Compte-tenu des définitions qui précèdent les états ci-dessus peuvent se repésenter par le diagrammme d'inclusion suivant.



Les transitions entre les états Existant, Inexistant, Exécutable, Non exécutable sont soit à la charge d'agences spécifiques, soit effectuées par le noyau, sous forme de d'opérations utilisables pour réaliser ces agences.

Les transitions entre les états En cours et Prêt sont réalisées par l'ordonnanceur.

4.2.4 MODE D'EXECUTION D'UNE TACHE

Quand une tâche est dans l'ensemble d'états En service elle peut être dans l'un des deux modes suivants :

- Non-préembtible : dans ce mode le processeur de la tâche ne peut lui être retiré au profit d'une autre tâche ;
- Préembtible : dans ce mode le processeur de la tâche peut lui être retiré au profit d'une autre tâche.

4.2.5 FONCTION TACHE-COURANTE

La fonction TACHE-COURANTE est une fonction qui permet à un programme de connaître l'identité de la tâche qui exécute ce programme. Elle fournit en résultat un objet de type TACHE.

4.2.6 FONCTION PRIORITE(TACHE)

La fonction PRIORITE fournit en résultat un entier qui est la priorité de la TACHE dont le nom est fourni en argument.

4.2.7 FONCTION ETAT (TACHE)

La fonction ETAT fournit en résultat l'état de la tâche convenablement représenté.

4.2.7 ORDONNANCEUR ET FONCTIONS D'ORDONNANCEMENT

L'ordonnanceur est une ensemble de fonctions allouant les processeurs libres aux tâches prêtes.

4.2.7.1 LES OBJETS CONNUS DE L'ORDONNANCEUR

Les objets suivants sont les seuls connus de l'ordonnanceur, mais partagés avec les opérations du noyau :

- Contextes des tâches actives
- Ensemble des tâches en attente de processeur

4.2.7.2 FONCTION

Le rôle de l'ordonnanceur consiste à désaccoupler le processeur de la tâche courante pour l'allouer à une tâche prête après exécution de l'algorithme de choix.

4.2.7.3 ALGORITHME DE CHOIX

L'algorithme de choix attribue les processeurs aux tâches prêtes en les prenant dans l'ordre FIFO avec priorité.

4.2.7.4 PREEMPTION

L'ordonnancement peut être sur option préemptif ou non préemptif.
Dans le cas où l'ordonnancement est préemptif, toute demande d'activation (par DEMARRER ou SIGNALER exclusivement) implique l'exécution de l'algorithme de choix. L'exécution de cet algorithme peut résulter dans le retrait du processeur de la tâche en cours lorsque celle-ci a une priorité inférieure à celle de la tâche concernée par la demande d'activation.
Dans le cas où l'ordonnancement est non préemptif, l'algorithme de choix n'est invoqué que lorsqu'un processeur est libéré.

4.2.7.5 INVOCATION DE L'ORDONNANCEUR

L'ordonnanceur ne peut être invoqué que par l'opération CHOISIR. Cette opération est interne au noyau. Elle n'est pas fournie aux utilisateurs du noyau.

4.2.7.6 FONCTION D'AJOUT D'UNE TACHE DANS L'ENSEMBLE DES TACHES PRETES

AJOUTER (TACHE)
Insérer la TACHE désirée dans l'ensemble des tâches prêtes suivant sa priorité. Cette opération est également interne au noyau.

4.2.2.7 OPERATIONS INVOQUANT L'ALGORITHME DE CHOIX

ATTENDRE dans le cas non passant,
DEMARRER, SIGNALER dans le cas préemptif,
SE TERMINER dans tous les cas

4.2.8 MECANISMES DE DEMARRAGE ET D'ARRET

INTRODUCTION

La création et la destruction des tâches ne sont pas des opérations du noyau, car elles peuvent être longues et coûteuses dans certains environnements matériels et logiciels. De plus, elles se trouvent, sur le plan logiciel, à la frontière entre les aspects "fonctionnement-exécution" et "production de programmes".
Le noyau SCEPTRE ne comprend donc que les opérations élémentaires DEMARRER une tâche, ARRETER une tâche et SE-TERMINER.

DEMARRER (TACHE)

Etat initial : la TACHE est dans l'état Hors-service.

Etat final : la TACHE est dans l'état Prêt.

Cas d'erreur : si la TACHE n'est pas dans l'état initial correct, un compte-rendu d'erreur est rendu à la tâche appelante.

DEMARRER peut induire une préemption lorsque l'ordonnancement est préemptif.

Remarque: L'opération DEMARRER ne fournit pas de passage de paramètres spécifiques. Ceci doit être effectué de façon explicite, à l'aide d'autres opérations du noyau.

ARRETER (TACHE)

Traitement :

a) Si la TACHE est dans l'état En-attente : le contexte de la TACHE est initialisé pour l'éxécution d'une séquence d'exception.
La TACHE est retirée de l'ensemble des taches en attente, et rangée dans l'ensemble des tâches prêtes.

b) Si la TACHE s'éxécute sur un processeur, celui-ci est arrêté, le contexte de la TACHE est initialisé pour qu'elle éxécute immédiatement une séquence d'exception, le processeur est alors relancé sur ce nouveau contexte.

SE-TERMINER

Traitement :

La tâche courante est désaccouplée de son processeur. Elle est mise dans l'état Hors-service.
Pour toute réactivation ultérieure éventuelle de cette tâche, on doit utiliser l'opération DEMARRER.

4.2.7.8 MODIFICATION DE LA PRIORITE D'UNE TACHE

CHANGER-PRIORITE (TACHE,NOUVELLE-PRIORITE)
La priorité de TACHE devient NOUVELLE-PRIORITE.

Cette opération n'est utilisable que dans les cas suivants :

. Auto-modification de la priorité d'une tache :

CHANGER-PRIORITE (TACHE-COURANTE,NOUVELLE-PRIORITE)

. Modification de la priorité d'une tache En-attente

CHANGER-PRIORITE (TACHE,NOUVELLE-PRIORITE)

4.3 GESTION DES EVENEMENTS

4.3.1 DESCRIPTION

La signalisation entre deux tâches nécessite une opération dans chacune des tâches : attente de signal dans l'une et envoi de signal dans l'autre. L'opération qui suit l'attente, dans la tâche "réceptrice", est éxécutée après l'opération qui précède l'envoi du signal, dans la tâche émettrice.

La signalisation est donc asymétrique par nature. Elle exprime une relation de cause à effet : par exemple, la tâche signalante a mis à la disposition de la tâche attendant une certaine information, que la tâche attendant va donc pouvoir traiter.
Une tâche doit pouvoir attendre successivement, à des instants différents de son éxécution , des événements différents.
Un événement représente chaque cause possible de continuation d'une tâche lors de ses opérations d'attente explicite. Un événement est donc spécifique d'une certaine tâche, susceptible de l'attendre.
C'est la responsabilité de la tâche attendant de vérifier, après que l'événement ait été signalé, que le traitement peut effectivement se poursuivre, c'est-à-dire que les informations ou les ressources nécessaires sont effectivement disponibles.
Dans le noyau SCEPTRE tout événement est associé à une tâche.
Ce choix fondamental est dicté par les raisons suivantes :

- Lorsqu'un même événement peut être attendu par plusieurs tâches, il faut lui associer une file d'attente (voir la réalisation des événements LTR dans la bibliothèque); un tel mécanisme résulte donc de la composition d'opérations de signalisation et de communication; l'association des événements aux tâches ne nécessite aucune file d'attente : elle fournit donc un mécanisme élémentaire permettant de construire les mécanismes de synchronisation de plus haut niveau.

- De plus l'association des événements aux tâches permet de minimiser les commutations de contexte intervenant lors de la signalisation d'un événement (voir recommandation d'implémentation).

Une tâche dispose ainsi d'un nombre fini d'événements, qu'elle peut attendre (ATTENDRE), attendre sélectivement ou purement et simplement effacer (EFFACER).
Par exemple, une tâche déclenchant une demande de délai (time out) immédiatement après une entrée/sortie non blocante doit pourvoir manipuler séparément (attendre et/ou effacer) deux événements : fin d'entrée/sortie et fin du délai.
Mais il est souhaitable aussi qu'elle puisse manipuler globalement un ensemble d'événements :

- Attendre la première occurrence d'événements parmi plusieurs,
- Effacer les événements d'une liste.

Il est ainsi possible de déclencher une entrée/sortie avec time-out et d'attendre le résultat de l'entrée/sortie ou la fin du délai.

4.3.2 SPECIFICATION

Un évènement est un objet qui peut être dans l'un des deux états "arrivé" ou "non arrivé".
Il est géré par les opérations élémentaires :

SIGNALER
EFFACER
ATTENDRE
et le prédicat ARRIVE

Dans la suite on note T la tâche possédant les événements E,E1,E2,...EN.

PREDICAT ARRIVE (E1,E2,...EN) :

Ce prédicat est vrai si les événements E1,E2,...EN sont tous arrivés.

SIGNALER (E,T) :

état final : E est mis dans l'état "arrivé".

Si T est en attente de E, alors T est ajoutée à l'ensemble des tâches prêtes. Si l'ordonnancement est préemptif, on appelle l'algorithme de choix.

EFFACER (E1, E2,...EN)

état final : E1, E2,...EN sont mis dans l'état "non arrivé".

ATTENDRE (E1,E2,...EN) :

Seule la tâche courante T peut exécuter ATTENDRE.

état final : Si l'un au moins des événements E1,E2,...EN est arrivé, l'opération ne suspend pas la tâche T. Si aucun de ces événements n'est arrivé, T est mise en attente sous le contrôle de l'ordonnanceur jusqu'à ce qu'au moins un de ces événements soit signalé.
ATTENDRE ne modifie pas l'état des événements E1, E2,...EN.

4.4 GESTION DES REGIONS

4.4.1 GENERALITES SUR L'EXCLUSION MUTUELLE

Le parallèlisme des tâches induit un phénomène de compétition pour la manipulation des objets partagés.
Pour assurer la cohérence de ces manipulations, il est nécessaire d'utiliser un mécanisme d'exclusion mutuelle.
Un tel mécanisme assure que les exécutions des séquences d'instructions manipulant un même objet partagé soient disjointes dans le temps quel que soit l'ordre dans lequel ces séquences sont invoquées.
On appelle section critique une telle séquence.
Usuellement la réalisation d'un section critique consiste à parenthéser la séquence d'instructions de la manière suivante :

```
commencer (section-critique) ;

séquence d'instructions ;

terminer (section-critique) ;
```

où section-critique est associée à l'objet partagé que l'on souhaite protéger.

Les opérations commencer et terminer sont telles que

- lorsque plusieurs tâches exécutent concurremment l'opération commencer (section-critique), seule l'une d'entre elles peut terminer cette exécution : elle est alors propriétaire de la section-critique.

- les autres doivent attendre (activement ou passivement) que la tâche propriétaire ait exécuté terminer (section-critique) pour être en mesure d'achever leur exécution de commencer (section-critique).

Cette définition très générale ne précise pas

- l'ordre d'attribution d'une section critique aux tâches ayant invoqué commencer (section-critique),

- le genre d'attente subie éventuellement par une tâche ayant invoqué commencer (section-critique) et n'ayant pas pu devenir propriétaire de la section critique,

- ce qui se passe en cas de tentative de préemption du processeur de la tâche propriétaire de la section critique.

Avec ce degré de généralité cette définition peut aussi bien modéliser les sémaphores booléens que l'exclusion mutuelle la plus élémentaire : celle qui consiste à attribuer la possession exclusive de toute la machine à une tâche lors de commencer (section-critique) et relacher cette possession lors de terminer (section-critique), section-critique étant alors un objet unique représentant la ressource machine.

Compte tenu des critères de généralité, de constructibilité et de performance (première partie,1.4) le noyau SCEPTRE se doit de fournir un mécanisme d'exclusion mutuelle qui :

- précise explicitement les aspects non concernés par la définition précédente,
- permette la construction de mécanismes d'exclusion mutuelle de plus haut niveau (sémaphores, monitors, control-queues,...)

et ceci

- sans dégrader les performances obtenues en utilisant directement la machine nue
- en permettant une sémantique uniforme sur monoprocesseur ou multiprocesseur.

4.4.2. PROPRIETES DU MECANISME D'EXCLUSION MUTUELLE RETENU

Le mécanisme d'exclusion mutuelle proposé par le noyau SCEPTRE est fondé sur le concept de REGION avec les caractéristiques suivantes :

- il ne suppose aucune attente passive et par suite aucun passage de la tâche concernée dans l'état En-attente ;
- il possède des propriétés uniformes sur monoprocesseur et sur multiprocesseur.
- il interdit toute préemption du processeur de la tâche courante tant qu'elle exécute une section critique protégée par une REGION .
- il permet de réaliser les constructions suivantes :
  - l'imbrication des sections critiques protégées par des REGIONs ;
  - l'utilisation à l'intérieur d'une telle section de toute opération du noyau SCEPTRE à l'exclusion de celles qui peuvent résulter dans la préemption du processeur courant.

4.4.3.SPECIFICATION

Un objet de type REGION peut prendre deux états : "libre" ou "occupée". Il peut être manipulé par deux opérations :

ENTRER (REGION)
SORTIR (REGION)

ENTRER (REGION)

Condition de démarrage de l'exécution : REGION = libre

Etat initial : état (TACHE-COURANTE) = En-cours
REGION = libre

Etat final : état (TACHE-COURANTE) = En-cours
mode d'exécution : Non préemptible
REGION = occupée

Sur multiprocesseur, tant que la condition de démarrage de l'exécution n'est pas satisfaite, la TACHE-COURANTE est en attente active.

SORTIR REGION)

Etat initial : REGION = occupée
état (TACHE-COURANTE) = En-cours
mode d'exécution : Non préemptible

Etat final : libre
état (TACHE-COURANTE) = En-cours
mode d'exécution = le mode d'exécution de TACHE-COURANTE avant la dernière exécution de l'opération ENTRER

4.4.4 RELATION AVEC LES AUTRES OPERATIONS DU NOYAU

4.4.4.1 IMBRICATION DES SECTIONS CRITIQUES PROTEGEES PAR DES REGIONS

Il est possible d'imbriquer statiquement des sections critiques protégées par des REGIONs :

```
ENTRER (REGION1) ;
        .
        .
        .
ENTRER (REGION2) ;
        .
        .
        .
SORTIR (REGION2) ; — on reste en mode Non préemptible
        .
        .
        .
SORTIR (REGION1) ; — on revient en mode Préemptible
```

Il est également possible d'obtenir une telle imbrication par le jeu des invocations de sous-programmes.

4.4.4.2 UTILISATION DES OPERATIONS DU NOYAU A L'INTERIEUR D'UNE SECTION CRITIQUE PROTEGEE PAR UNE REGION

Il n'est pas possible d'utiliser toutes les opérations du noyau à l'interieur d'une telle section.
Sont interdites les opérations qui peuvent résulter dans une préemption du processeur courant :

- ATTENDRE dans tous les cas,
- SIGNALER et DEMARRER en cas d'ordonnancement préemptif.

4.4.4.3 VERIFICATIONS DE CORRECTION A L'EXECUTION

L'utilisation des opérations interdites dans une section critique SCEPTRE provoque une erreur à l'exécution.
En revanche, la responsabilité d'éviter une étreinte fatale ("deadlock") provoquée par une imbrication erronnée de sections critiques SCEPTRE incombe à l'utilisateur.

4.5 MECANISMES DE COMMUNICATION ENTRE TACHES VOISINES

4.5.1 INTRODUCTION

La communication entre tâches voisines s'effectue le plus souvent par le partage d'informations dans la mémoire commune. Ce partage est régi dans chaque cas particulier par un protocole réglementant l'ordre des accès aux informations partagées.
Les stratégies les plus courantes utilisent la structure de file :

- files d'attente de tâches
- files de messages.

C'est pourquoi le noyau SCEPTRE propose un ensemble d'opérations de gestion des files pour réaliser ces protocoles.

4.5.2 GESTION DES FILES

Une file est un composant passif qui assure le stockage transitoire d'éléments selon un ordre de classement par critères (premier sorti = premier entré dans le groupe de critère demandé ou de poids le plus fort, ce critère étant optionnel).
Une file peut avoir un nombre borné ou non d'éléments.

4.5.3 OPERATIONS

Les opérations de manipulation d'une FILE sont :

ENVOYER (élément, FILE (, critère))

RETIRER (élément, FILE (, critère))

pouvant donner lieu aux cas d'erreur

- FILE VIDE : on essaye de retirer un élément dans une file vide
- FILE PLEINE : on essaye d'envoyer un élément dans une file dont le nombre d'éléments est égal à la borne fixée à la création.

Le critère est un entier positif.

Pour faciliter la gestion des files, on fournit les prédicats :

VIDE (FILE) : prédicat dont la valeur est vraie lorsque la FILE est vide

PLEINE (FILE) : prédicat dont la valeur est vraie lorsque la FILE est pleine (nombre d'éléments borné)

4.5.4 SPECIFICATION DES OPERATIONS

. ENVOYER (E, F(, C))

Le critère C est un entier positif ou nul.

Par défaut le critère C est égal à 0

état initial : F est une suite éventuellement vide de groupes d'éléments chaque groupe étant associé à une valeur du critère, les groupes étant ordonnés suivant les valeurs décroissantes de critère.

traitement :

- On cherche le groupe g correspondant au critère C.
- On range l'élément E à la queue dans le groupe g.

. RETIRER (E, F (, C))

état initial : F est une suite non vide de groupes d'éléments, chaque groupe étant associé à une valeur du critère, les groupes étant ordonnés suivant les valeurs décroissantes de critère.

traitement :

- Cas d'erreur si le groupe g correspondant au critère C est vide
- Lorsque F n'est pas vide on retire le premier élément du groupe g et on le retourne dans E. Par défaut, g est le groupe dont le critère à la valeur la plus grande.

VIDE (F) = vrai équivaut à F ne contient aucun groupe

PLEINE (F) = vrai équivaut à borne (F) = nombre d'éléments de F.

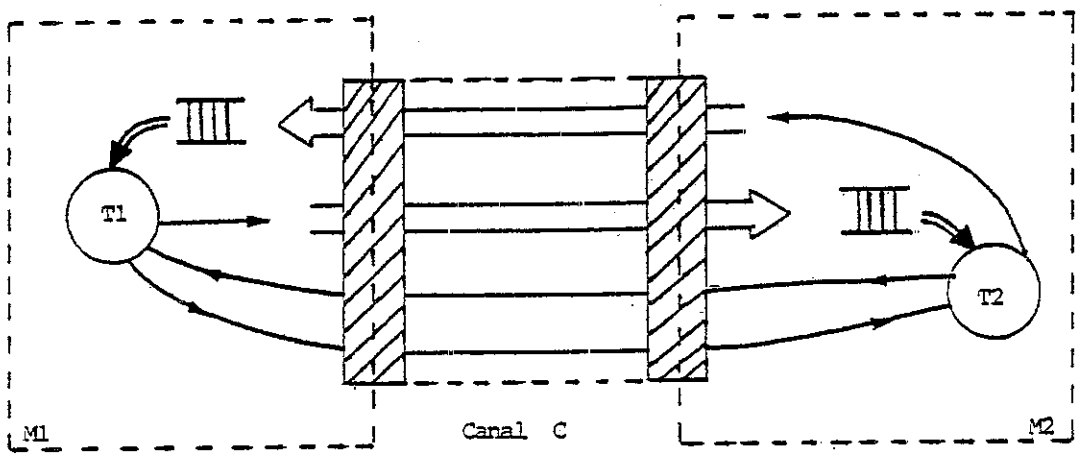
4.6 MECANISME DE COMMUNICATION ENTRE TACHES DISTANTES

4.6.1 SCHEMA GENERAL DE LA COMMUNICATION ENTRE 2 MACHINES

La communication entre 2 machines M1 et M2 est assurée par un canal C qui comprend généralement les éléments suivants :

- des chemins de données allant de M1 vers M2 ou de M2 vers M1.

- des chemins de signaux permettant la signalisation de M1 vers M2 ou de M2 vers M1.

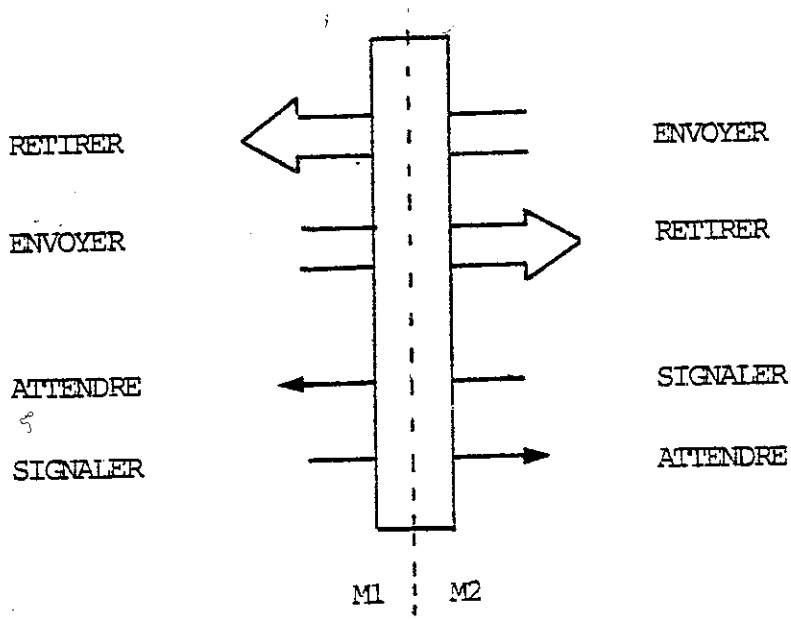


Vu de l'une des machines, le canal apparait par son interface qui est géré par des opérations du type :

- envoyer/retirer pour les données

- signaler/attendre pour la signalisation.

On remarquera que ces opérations sont symétriques selon qu'on les voit de M1 ou de M2.

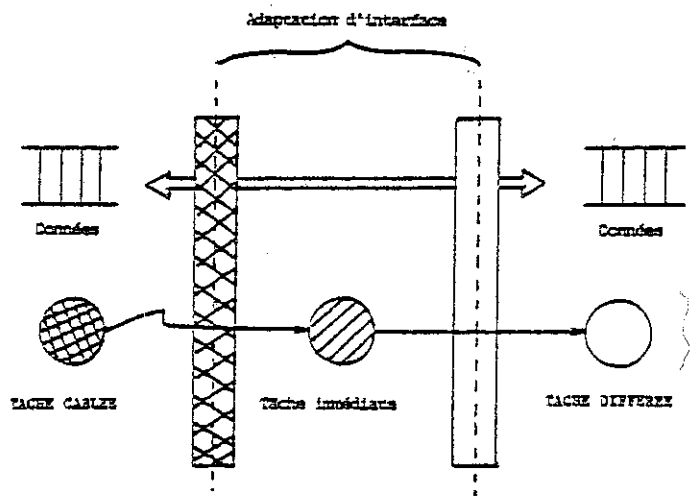


Description d'une interface

4.6.2 ADAPTATION DE L'INTERFACE

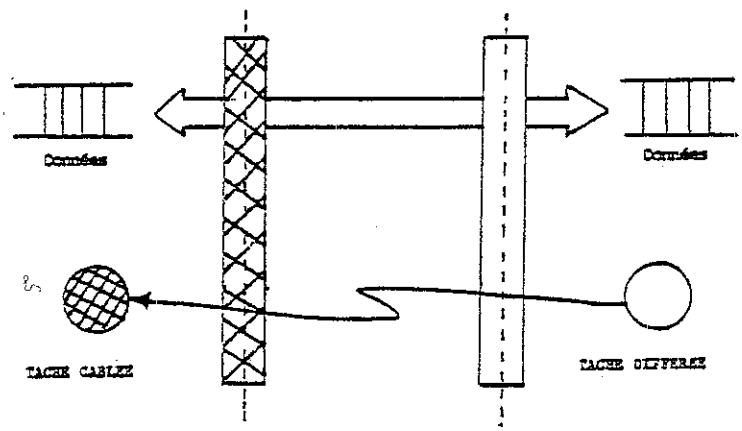
4.6.2.1 INTERFACE D'ENTREE

Les différents mécanismes de signalisation de l'extérieur vers le système sont transformés par le matériel en un mécanisme unique : celui des interrruptions.
La transformation du signal d'interruption en une signalisation SCEPTRE est assurée au moyen d'une tâche dite immédiate qui est associée à un niveau d'interruption.



4.6.2.2 INTERFACE DE SORTIE

La signalisation du système vers l'extérieur est effectuée par des fonctions spécifiques.
Par souci d'homogénéité, on regroupe les fonctions de communication et de signalisation vers l'extérieur dans une agence de communication. Cette agence permet aux tâches d'accéder aux fonctions de communications externes par une interface banalisée.



Remarque : Une tâche immédiate peut signaler un événement à une tâche différée. En revanche elle ne peut en aucun cas se mettre en attente d'un événement.

4.7 GESTION DES INTERFACES

4.7.1 PRESENTATION

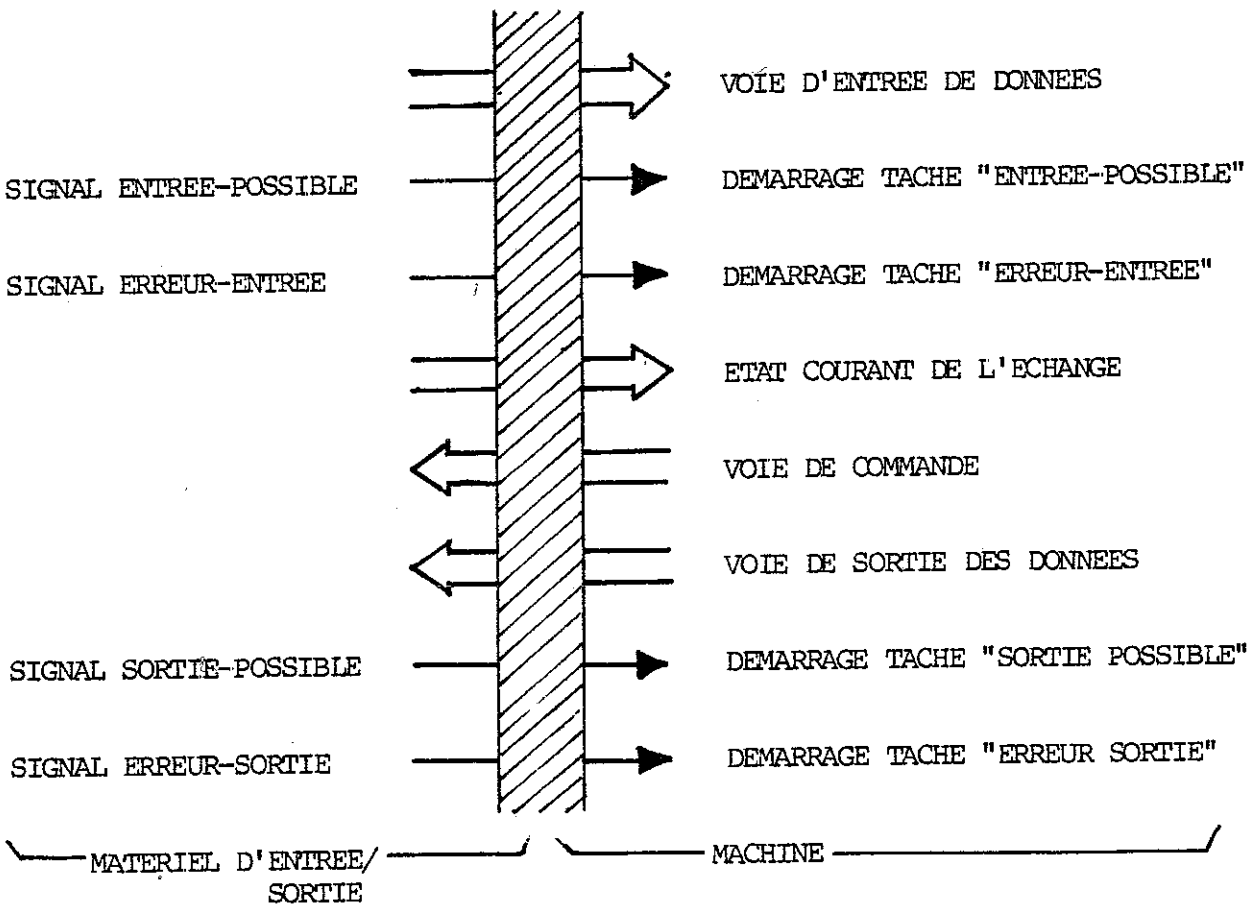
Le problème de la communication entre tâches distantes ou entre tâches et environnement de l'application (entrées/sorties) passe par la gestion des interfaces.

La diversité des interfaces du marché tient à la fois aux techniques de transfert des informations, aux protocoles de communication et à la signification des informations transmises.

Cette diversité nous empêche de proposer un type interface qui jouerait le même rôle que le type TACHE en ce sens qu'il pourrait avoir une implémentation unique pour chaque machine.

Cependant, et de façon analogue à l'introduction du type FILE, nous proposons un type INTERFACE qui modélise bon nombre d'interfaces matérielles actuelles.

4.7.2 MODELE D'INTERFACE



Deux interfaces correspondant à ce modèle ne diffèrent que par les types des objets qui peuvent y transiter : données en entrée, données en sortie, commandes, états.
Une interface apparait ainsi comme une structure de données. En ce sens elle est tout à fait analogue à une FILE.

4.7.3 OPERATIONS SUR LES INTERFACES

COMMANDER (INTERFACE, ORDRE)
TESTER (INTERFACE, ETAT)
LIRE (INTERFACE, RESULTAT)
ECRIRE (INTERFACE, DONNEE)

. Ces quatre opérations sont exclusives et atomiques. Elles n'induisent aucune attente de la tâche qui les utilise.

. L'envoi d'un ORDRE par COMMANDER peut se faire n'importe quand.
Les commandes peuvent être suivant les interfaces :

- . simples : dans ce cas, on peut énumérer les ordres possibles : exemple : arrêt-lecture, arrêt-écriture.

- . composites : dans ce cas l'ORDRE fournit plusieurs informations
  exemple : ordre de positionnement d'une tête de disque.

. Le test de l'état peut également s'exécuter à tout moment.

Les opérations d'entrées/sorties basées sur un mécanisme d'accès direct à la mémoire se font par l'intermédiaire de l'opération COMMANDER (INTERFACE, ORDRE) où ORDRE spécifie la nature du transfert et la localisation des données transférées. Ceci suppose l'existence d'un automate capable d'interpréter l'ORDRE (exemple "canal programme").
Dans les autres cas les transferts sont réalisés par les opérations LIRE et ECRIRE correspondant à la manipulation des registres d'entrée/sortie d'un coupleur.
Les signaux ENTREE-POSSIBLE, SORTIE-POSSIBLE, ERREUR-ENTREE, ERREUR-SORTIE sont traduits par l'interface en démarrage de tâches immédiates associées.

ENTREE-POSSIBLE indique qu'une opération de lecture est possible, soit par COMMANDER, soit par LIRE

SORTIE-POSSIBLE indique qu'une opération d'écriture est possible, soit par COMMANDER, soit par ECRIRE

Les signaux d'erreur indiquent que le transfert en cours ne s'est pas terminé normalement.

4.7.4 INTERFACE MULTIPLEXEE

Une interface multiplexée correspond à un tableau d'interfaces simples pour lequel les signaux d'interface sont regroupés en quatre signaux identiques à ceux de l'interface simple précédante. La forme des opérations pour une telle interface est :

COMMANDER (INTERFACE, (VOIE,) ORDRE
TESTER (INTERFACE, (VOIE,) ETAT)

la mention d'une voie est optionnelle

LIRE (INTERFACE, VOIE, RESULTAT)
ECRIRE (INTERFACE, VOIE, DONNEE)

DEUXIEME PARTIE : MISE EN OEUVRE DU NOYAU SCEPTRE

SOMMAIRE

1 - COMMENT UTILISER LE NOYAU SCEPTRE

SOMMAIRE

1 COMMENT UTILISER LE NOYAU SCEPTRE

1.1 Remarques sur l'implémentation d'un noyau SCEPTRE

L'implémentation d'un noyau SCEPTRE sur un matériel donné a pour base de départ le répertoire d'instructions offert par la machine. Il englobe les instructions du mode superviseur et les instructions du mode utilisateur.
Ce répertoire d'instructions doit offrir au minimum les possibilités suivantes :

1) un moyen d'implémenter l'exclusion mutuelle : si l'on est dans un cas de monoprocesseur, il suffit de disposer d'instructions permettant l'inhibition de la préemption (masquage) et la validation de la préemption (démasquage);
si l'on est dans un cas de multiprocesseur, il faut disposer en plus d'un mécanisme garantissant l'accès exclusif à la mémoire commune (exemple : "test-and-set", "reset").

2) un mécanisme d'invocation des opérations du noyau : ce mécanisme peut être de type procédural ou de type appel superviseur (SVC).
On donne ici, à titre indicatif, l'ordre dans lequel il est souhaitable d'analyser l'implémentation des types ou opérations du noyau SCEPTRE :

1 - Les régions
2 - La représentation des tâches et des événements qui leur sont attachés
3 - La désignation des objets SCEPTRE
4 - L'ordonnanceur
5 - Les files.

On trouvera dans ce document un exemple d'implémentation des opérations ENTRER et SORTIR au paragraphe 4 de cette deuxième partie.

1.2 Remarques sur la construction d'un Exécutif à partir du noyau

1.2.1 Portabilité de l'Exécutif

L'Exécutif assure la gestion des ressources fournies par le matériel. Il est donc illusoire de vouloir assurer une portabilité totale entre des matériels qui ne sont pas homogènes.
Il semble qu'on puisse tout de même garantir une bonne portabilité en ce qui concerne :

- la synchronisation
- la signalisation
- l'exclusion mutuelle
- la communication entre tâches voisines.

L'élément essentiel de cette portabilité est assuré par le choix, pour l'écriture de l'Exécutif, d'un langage portable.

1.2.2 Eléments de méthodologie

1) On recommande de limiter l'utilisation des opérations du noyau à la construction de mécanismes de plus haut niveau. Les opérations sur région, par exemple, ne seront utilisées que pour construire les mécanismes du type monitor, rendez-vous, sémaphore, etc...
Dans le cas où la machine utilisée comporte des mécanismes de ce type (exemple : sémaphore), cela revient, du point de vue de l'Exécutif à considérer qu'ils font partie du noyau.

2) Il vaut mieux éviter de construire des mécanismes de haut niveau à partir d'autres mécanismes de haut niveau. Ces mécanismes doivent à chaque fois être construits à partir des opérations du noyau. On trouve une illustration de ceci dans la littérature avec l'implémentation érronée dans le cas d'un multiprocesseur des monitors à partir des sémaphores. On préfèrera pour une telle implémentation avoir recours aux opérations offertes par le noyau.

3) La manipulation des événements peut se faire sans difficulté à tous les niveaux. Toutefois on devra faire attention au fait que les différentes agences qui composent l'Exécutif utilisent certains événements pour leurs besoins propres. Il y a donc lieu de veiller au partage de ces événements et d'en contrôler l'utilisation.

4) Les objets manipulés par l'Exécutif sont des objets composites obtenus à partir des objets du noyau. De même, les opérations offertes par l'Exécutif sont des opérations composites obtenues à partir des opérations élémentaires du noyau. L'invocation depuis l'Exécutif des opérations du noyau devra se faire à l'aide de sous-programmes ouverts (macro-génération en ligne) plutôt que de manière procédurale afin d'éviter la multiplication des couches et une diminution de la performance.

1.2.3 Interface avec le langage de programmation

Lorsqu'on utilise un langage de programmation Temps Réel (LTR, ADA...) l'Exécutif est caché par le langage. On peut dire que l'interface avec cet Exécutif est assuré par le compilateur.
Si l'on utilise un langage de programmation qui n'est pas spécifiquement Temps Réel, on devra invoquer les fonctions de l'Exécutif de manière procédurale. L'interface est alors assurée par un "Run-Time" (Exemple : FORTRAN Temps Réel).
On risque cependant de rencontrer une certaine difficulté si le langage ne donne pas accès à une instruction du type SVC.

1.3 Noyau d'entrée/sortie associé au noyau SCEPTRE

Le modèle d'interface que nous avons présenté doit permettre pour chaque implémentation du noyau SCEPTRE la réalisation d'un noyau d'entrée/sortie associé qui soit :

- uniforme dans les notations,
- compatible avec l'approche SCEPTRE,
- compatible avec le noyau SCEPTRE.

Ce noyau d'entrée/sortie faciliterait la manipulation de périphériques particuliers, assurant la portabilité des programmes utilisateurs à condition de conserver les propriétés logiques de la configuration d'entrée/sortie ainsi gérée. Dans tous les cas il constituerait une excellente interface avec un langage de programmation évolué pour l'écriture des systèmes.

1.4 Remarques générales

Le noyau SCEPTRE est conçu de façon à assurer la gestion d'une seule ressource, la machine et à ne fournir que des mécanismes élémentaires indépendants de toute application.
La gestion des ressources du système, à l'exception de la machine est confiée à un certain nombre d'agences utilisant elles-mêmes les mécanismes fournis par le noyau (les entrées/sorties, horloge...).
L'Exécutif utilise directement les mécanismes élémentaires du noyau pour la construction de l'interface offerte à l'application. Les caractéristiques de cette interface sont déterminées par les spécifications et les contraintes de l'application.
Les mécanismes du noyau SCEPTRE sont suffisamment élémentaires pour permettre la construction de la plupart des mécanismes de communication.
Les règles à appliquer pour cette construction et son utilisation doivent cependant viser à ne pas diminuer les performances de la machine utilisée.
De manière générale, la construction d'un mécanisme de communication se fonde sur la spécification externe de l'activité de communication : "fonction + interface".
A partir de cette spécification il est possible de construire, sur la base des mécanismes du noyau, les mécanismes recherchés.
Le chapitre "Bibliothèque de service de haut niveau" donne la structure des algorithmes de construction de différents mécanismes de communication ou synchronisation (sémaphore, monitor, rendez-vous...) à partir des fonctions fournies par le noyau SCEPTRE. L'invocation et l'intégration des mécanismes ainsi construits, dans le langage de haut niveau ou dans l'application, sont réalisés selon les besoins et les contraintes du langage de haut niveau ou de l'application. En ce qui concerne les mécanismes de gestion des tâches, leur utilisation doit être soigneusement analysée. Pour des raisons de cohérence du système, il est préférable d'assigner la fonction de gestion de tâches à une agence spécialisée dans ce but. De cette manière l'agence prendra toutes les requêtes en considération et n'exécutera que celles qui sont conformes aux spécifications du fonctionnement du système.

2 - BIBLOTHEQUE DE SERVICES DE HAUT NIVEAU

SOMMAIRE

2.1 METHODE DE DESCRIPTION

Les mécanismes de haut niveau de cette bibliothèque sont décrits de la façon suivante.

Une introduction informelle précise la signification des types et opérations associés au mécanisme.

Un module décrit en utilisant les notations du langage ADA précise la représentation de ces types et opérations en termes du noyau SCEPTRE.
La notation pointée permettant de désigner les composants d'un objet composite ("record") est évitée partout où il n'y a aucune ambiguité sur l'appartenance de ces composants.

Les objets et opérations du noyau SCEPTRE sont notés en lettres majuscules.

Dans la bibliothèque nous utilisons les REGIONs en évitant au maximum les constructions pouvant conduire à des imbrications de régions. Ceci doit permettre aux utilisateurs d'obtenir rapidement les implémentations les plus efficaces.

2.2 SEMAPHORES BOOLEENS

2.2.1 DESCRIPTION INFORMELLE

Le sémaphore booléen est un mécanisme d'exclusion mutuelle destiné à protéger une ressource ne pouvant être détenue que par une tâche à la fois.

On admet qu'un sémaphore booléen est dans l'état initial "libre". Il passe dans l'état "occupé" à chaque fois qu'une tâche en prend possession par l'opération "request". Toute tâche qui essaye d'exécuter cette opération alors que le sémaphore booléen est dans l'état "occupé" est mise en file d'attente.
Lorsque la tâche propriétaire du sémaphore booléen le libère par l'opération "release", le sémaphore reprend l'état "libre" si cette file d'attente est vide. Dans le cas contraire, la première tâche de la file d'attente devient propriétaire du sémaphore booléen et peut ainsi terminer l'opération "request" en cours.

2.22 DESCRIPTION ALGORITHMIQUE

```
type sémaphore-booléen is record
                              libre : booléen := true ;
                              F : FILE(FIFO) of TACHE
                          end record ;

X : REGION ; —on associe une région au type plutôt qu'à chaque objet du
             type
```

```
procedure request (s : in out sémaphore-booléen) is

    begin
        ENTRER (X) ;
        if not libre then ENVOYER (TACHE-COURANTE, F) ;
                          SORTIR (X) ;
                          ATTENDRE (El) ;
                          EFFACER (El) ;
                     else libre := false ;
                          SORTIR (X)
        end if
    end request ;

procedure release (s : in out sémaphore-booléen) is

    begin T : TACHE ;
        ENTRER (X) ;
        if not VIDE (F) then RETIRER (T, F) ;
                             SORTIR (X) ;
                             SIGNALER (El, T) ;
                        else libre := true ;
                             SORTIR (X)
        end if
    end release ;
```

2.3 SEMAPHORES A COMPTE

2.3.1 DESCRIPTION INFORMELLE

Le sémaphore à compte est un mécanisme d'exclusion mutuelle destiné à protéger une ressource dont le nombre de détenteurs potentiels à un instant donné est fixé par un compte associé au sémaphore.
Pour le reste le sémaphore à compte fonctionne comme un sémaphore booléen.

2.3.2 DESCRIPTION ALGORITHMIQUE

```
type sémaphore is record
                      F : FILE (FIFO) of TACHE ;
                      compte : integer -- à initialiser à la déclaration
                  end record ;

X : REGION ;

procedure p (s : in out sémaphore) is

     begin
          ENTRER (X) ;
          compte := compte - 1 ;
          if compte LT 0 then  ENVOYER (TACHE-COURANTE, F) ;
                               SORTIR (X) ;
                               ATTENDRE (El) ;
                               EFFACER (El)
                          else SORTIR (X)
          end if
     end p ;


procedure v (s : in out sémaphore) is

     begin T : TACHE ;
          ENTRER (X) ;
          compte := compte + 1 ;
          if compte LE 0 then  RETIRER (T ,F) ;
                               SORTIR (X) ;
                               SIGNALER (El, T)
                          else SORTIR (X)
          end if
     end v ;
```

2.4 MONITORS

2.4.1 DESCRIPTION INFORMELLE

Le monitor est un mécanisme d'exclusion mutuelle et de signalisation.
En tant que mécanisme d'exclusion mutuelle, il se comporte comme un sémaphore booléen.
Une tâche propriétaire du monitor peut se mettre en attente d'une condition associée à ce monitor. Cette attente a pour effet de libérer le monitor pour les tâches qui veulent en prendre possession.
Chacune des conditions spécifiques associées à un monitor est évaluée par les tâches qui en deviennent tour à tour propriétaires. Lorsque une condition est devenue vraie, la tâche qui l'évalue le signale à la première tâche en attente de cette condition. Celle-ci peut alors reprendre son exécution.
Pour éviter que deux tâches ne se trouvent alors propriétaires du monitor, la tâche qui signale la condition :

. ou bien quitte le monitor en signalant ,

. ou bien se met en attente prioritaire de la ressource monitor.

Pour réaliser ce mécanisme on introduit les types monitor et condition ainsi que les opérations :

enter : demande de prise de possession de la ressource monitor

exit : libération du monitor

wait : attente d'une condition

signal : signalisation d'une condition avec attente prioritaire

signal-exit : signalisation d'une condition avec libération du monitor.

2.4.2 DESCRIPTION ALGORITHMIQUE

```
type monitor is record libre : booléan := true ;
                       F : FILE (à-priorité) of TACHE
                end record ;

type condition is FILE (FIFO) ;

X : REGION ;

procedure enter (m : in out monitor) is

     begin
          ENTRER (X) ;
          if not libre then ENVOYER (TACHE-COURANTE, F, 0) ; -- 0 : priorité
                                                                  minimum
                            SORTIR (X) ;
                            ATTENDRE (El) ;
                            EFFACER (El)
                       else libre := false ;
                            SORTIR (X)
          end if
     end enter ;
```

2.4 MONITORS

2.4.1 DESCRIPTION INFORMELLE

Le monitor est un mécanisme d'exclusion mutuelle et de signalisation.
En tant que mécanisme d'exclusion mutuelle, il se comporte comme un sémaphore booléen.
Une tâche propriétaire du monitor peut se mettre en attente d'une condition asscociée à ce monitor. Cette attente à pour effet de libérer le monitor aux tâches qui veulent en prendre possession.
Chacune des conditions spécifiques associées à un monitor est évaluée par les tâches qui en deviennent tour à tour propriétaires. Lorsque une condition est devenue vraie, la tâche qui l'évalue le signale à la première tâche en attente de cette condition. Celle ci peut alors reprendre son exécution.
Pour éviter que deux tâches ne se trouvent alors propriétaires du monitor, la tâche qui signale la condition :

. ou bien quitte le monitor en signalant ,

. ou bien se met en attente prioritaire de la ressource monitor.

Pour réaliser ce mécanisme on introduit les types monitor et condition ainsi que les opérations :

enter : demande de prendre possession de la ressource monitor

exit : libération du monitor

wait : attente d'une condition

signal : signalisation d'une condition avec attente prioritaire

signal-exit : signalisation d'une condition avec libération du monitor.

2.4.2 DESCRIPTION ALGORITHMIQUE

```
type monitor is record libre : booléan := true ;
                       F : FILE (à-priorité) of TACHE
                end record ;

type condition is FILE (FIFO) ;

X : REGION ;

procedure enter (m : in out monitor) is

     begin
          ENTRER (X) ;
          if not libre then ENVOYER (TACHE-COURANTE, F, 0) ; — 0 : priorité
                                                               minimum
                            SORTIR (X) ;
                            ATTENDRE (El) ;
                            EFFACER (El)
                       else libre := false ;
                            SORTIR (X)
          end if
     end enter ;
```

```
procedure exit (m : in out monitor) is

     begin T : TACHE ;
          ENTRER (X) ;
          if not vide (F) then RETIRER (T,F) ;
                               SORTIR (X) ;
                               SIGNALER (T, El)
                          else libre := true ;
                               SORTIR (X)
          end if
     end exit ;


procedure wait (m : in out monitor ; c : in out condition) is

     begin
          -- la tâche courante est propriétaire du monitor
          ENVOYER (TACHE-COURANTE, c) ;
          exit (m) ;
          ATTENDRE (El) ;
          EFFACER (El)
     end wait ;


procedure signal (m : in out monitor ; c : in out condition) is

     begin T : TACHE ;
          if not VIDE (c) then RETIRER (T, c) ;
                               ENTRER (X) ;
                               ENVOYER (TACHE-COURANTE, F, 1) ;
                               -- la priorité 1 permet à la TACHE-COURANTE
                               -- de reprendre possession du monitor avant
                               -- les tâches qui y entrent par enter
                               SORTIR (X) ;
                               SIGNALER (El, T) ;
                               ATTENDRE (El) ;
                               EFFACER (El)
          end if
     end signal ;
```

Remarque : l'événement El est utilisé dans enter et wait avec deux significations différentes. Cette utilisation est correcte. Elle nuit cependant à la compréhension de l'algorithme.

```
procedure signal-exit (m : in out monitor ; c : in out condition) is

     begin T : TACHE ;
          if not VIDE (c) then RETIRER (T, c) ;
                               SIGNALER (El, T);
                          else exit (m)
          end if
     end signal-exit ;
```

2.5 CONTROL-QUEUES

2.5.1 DESCRIPTION INFORMELLE

La control queue de MASCOT est un objet de synchronisation composé de 4 éléments :

- Une priorité PRIOR modélisant la rareté de l'objet protégé par la control queue

- Une file d'attente F

- Un élément FRONT contenant une tâche ayant exécuté un WAIT

- Une variable STATE ayant les valeurs suivantes :

1 Personne n'utilise la control queue

2 La control queue est utilisée

3 Quelqu'un est en attente dans FRONT

4 Un signal a été émis alors qu'on était dans l'état 2

5 Un signal a été émis alors qu'on était dans l'état 1

Les états 4 et 5 mémorisent l'émission d'un STIM.

Il existe quatre primitives :

. JOIN (control queue) correspond au REQUEST du sémaphore booléen. La tâche prend alors la priorité de la control queue.

. LEAVE (control queue) correspond au RELEASE du sémaphore booléen. La priorité de la tâche est restaurée.

. WAIT (control queue) ne peut être utilisée qu'entre JOIN et LEAVE. La tâche se met en attente d'un stimulus dans la case FRONT. Si le stimulus a été memorisé (état 4 ou 5) la primitive est passante.

. STIM (control queue). Si une tâche est en attente (état 3) elle est réactivée. Autrement le stimulus est mémorisé par le passage dans l'état 4 ou 5. Un seul stimulus est mémorisé à la fois.

2.5.2 DESCRIPTION ALGORITHMIQUE

```
type CONTROL QUEUE is record
                          F : FILE DE TACHE;
                          PRIOR : INTEGER CONSTANT;
                          STATE : (1,2,3,4,5) : = 1;
                          FRONT : TACHE;
                          OLD-PRIOR : INTEGER;
                      end record
```

PRIOR est initialisé pour chaque déclaration de control queue;

X : REGION ;

```
procedure JOIN (CQ : in-out control queue) is

    begin
      ENTRER (X);
      if (STATE LT 5 and STATE GT 1)
            then ENVOYER (TACHE-COURANTE, F);
                 SORTIR (X);
                 ATTENDRE (E); EFFACER (E)
            else STATE := if (STATE EQ 1) then 2 else 4 endif ;
            SORTIR (X)
      end if ; --la tâche-courante est ici propriétaire unique de CQ.
      OLDPRIOR := PRIORITE (TACHE-COURANTE) ;
      CHANGER-PRIORITE (TACHE-COURANTE, PRIOR) ;
 end JOIN;

procedure LEAVE (CQ : in-out control queue) is

    begin
      CHANGER-PRIORITE(TACHE-COURANTE, OLDPRIOR) ;
      ENTRER (X);
      if (STATE NE 2) or (STATE NE 4)
            then SORTIR(X) ; -- cas d'erreur
      elsif not VIDE(F)
            then RETIRER (T,F);
                 SORTIR (X);
                 SIGNALER (E,T);
      else STATE := if (STATE EQ 2)then 1 else 5 endif ;
      SORTIR (X)
      endif
    end LEAVE ;

procedure STIM (CQ in-out control queue) is

    begin T : TACHE
      ENTRER (X);
      if (STATE  EQ  3)
            then T := FRONT;
                 STATE := 2;
                 SORTIR (X);
                 SIGNALER (E,T);
            else STATE := if (STATE  EQ 1) then 5
                          elsif(STATE EQ 2) then 4 end if
            SORTIR (X);
      end if
    end STIM ;

procedure WAIT (CQ in-out control queue) is

    begin
      ENTRER (X);
      if (STATE EQ 2)
            then FRONT := TACHE-COURANTE;
                 STATE := 3 ;
                 SORTIR (X) ;
                 ATTENDRE (E) ; EFFACER (E) ;
      elsif (STATE EQ 4)
            then STATE:= 2 ;
                 SORTIR (X) ;
            else SORTIR (X) ; -- cas d'erreur
      endif
    end WAIT
```

2.6 EVENEMENTS DE LTR

2.6.1 DESCRIPTION INFORMELLE

Dans le langage LTR un événement est un objet qui peut prendre deux états: "arrivé" et "non-arrivé"
Il se distingue des événements du noyau SCEPTRE par les caractéristiques suivantes :

- Une ou plusieurs tâches peuvent se mettre en attente d'un événement

- Il existe deux façons d'exprimer qu'un événement est arrivé :

  - statique : l'événement est mis dans l'état "arrivé" ; toutes les tâches en attente de cet événement sont mises dans l'état prêt ; l'événement reste dans l'état "arrivé" jusqu'à ce qu'on le mette explicitement dans l'état "non-arrivé".

  - impulsionnelle : toutes les tâches en attente de l'événement sont mises dans l'état prêt ; l'événement reste dans l'état "non-arrivé"

La manipulation des événements en LTR se traduit par les cinq opérations suivantes.

Resetev
L'événement est mis dans l'état "non-arrivé". A partir de cet instant, toute tâche se mettant en attente de cet événement est mise dans l'état suspendu.

Setev
L'événement est mis dans l'état "arrivé". Les tâches en attente de l'évenement sont mises dans l'état prêt. Tant que l'événement reste dans cet état, toute tâche se mettant en attente de l'événement reste active.
L'opération n'a pas d'effet si l'événement est déjà arrivé.

Activate
Les tâches en attente de l'événement sont mises dans l'état prêt. L'événement reste dans l'état "non-arrivé". Par suite, toute tâche se mettant ultérieurement en attente de l'événement est mise dans l'état suspendu.

Testev
Retourne un résultat booléen mis à vrai si l'événement est arrivé et à faux dans le cas contraire.
L'état d'un événement impulsionnel acquis par testev est considéré comme "non-arrivé".

Wait
Met la tâche courante dans l'état suspendu si l'événement est dans l'état "non-arrivé".

2.6.2 DESCRIPTION ALGORITHMIQUE

```
type événement is record
                    F : FILE (FIFO) of TACHE ;
                    arrivé : booléen
                  end record ;

X : sémaphore-booléen ;

procedure Resetev (e : in out événement) is

    begin
      request (X) ;
      arrivé := false ;
      release (X)
    end Resetev ;


procedure Setev (e : in out événement) is

    begin T : TACHE ;
      request (X) ;
      while not VIDE (F) loop
         RETIRER (T, F) ;
         SIGNALER (El, T) ;
      end loop ;
      arrivé := true ;
      release (X)
    end Setev ;


procedure Activate (e : in out événement) is

    begin T : TACHE ;
      request (X) ;
      while not VIDE (F) loop
         RETIRER (T, F) ;
         SIGNALER (El, T)
      end loop ;
      release (X)
    end Activate ;


procedure Testev (e : in out événement ; ARRIVE : out booléen) is

    begin request (X) ; ARRIVE := arrivé ; release (X) end Testev ;


procedure wait (e in out événement) is

    begin
      request (X);
      if not arrivé then ENVOYER (TACHE-COURANTE, F) ;
                         release (X) ;
                         ATTENDRE (El) ; EFFACER (El) ;
                    else realease (X)
      end if
    end wait ;
```

2.7 SEMAPHORES MULTIPLES

2.7.1 Description informelle

Dans certaines applications il est nécessaire de pouvoir réserver de façon indivisible plusieurs ressources. Le mécanisme des sémaphores ne permet pas d'y parvenir comme le montre l'exemple académique des "philosophes aux spaghettis".
La réservation multiple de ressource peut se faire en introduisant le mécanisme de sémaphore multiple qui opère de la façon suivante.
On regroupe au sein d'un sémaphore multiple une collection de ressources identifiées par les numéros 1..n.
Toute réservation multiple précise la liste des ressources demandées. Tant que toutes ces ressources ne sont pas simultanément disponibles la tâche demanderesse reste dans l'état suspendu. La satisfaction des demandes se fait dans l'ordre FIFO.
Pour traduire cette manipulation on introduit le type sémaphore-multiple et les opérations :

réserver (sémaphore-multiple, ressources)
libérer (sémaphore-multiple, ressources)

où ressources est une chaîne de bits

ressources (J) = 1 si la ressource J est réservée et 0 dans le cas contraire.

2.7.2 Description algorithmique

```
type ressource is array (1..n) of bit;

type sémaphore multiple is record : libre : ressource : = (1..n =  1);
                                    Q : ensemble de requête;—type à définir
                            end record;

type requête is record d : ressources;
                       t : TACHE
                end record;
X : sémaphore-booléen;

procedure réserver (S : in out sémaphore-multiple; R : ressources) is

        begin request (X)
          if R inclus dans libre -- i.e.  R and  libre = R
                then libre : = libre and not (R);
                     release (X)
                else ranger dans Q la requête (d = R, t = TACHE-COURANTE);
                     release (X);
                     ATTENDRE (El);
                     EFFACER (El)
          end if
        end réserver;
```

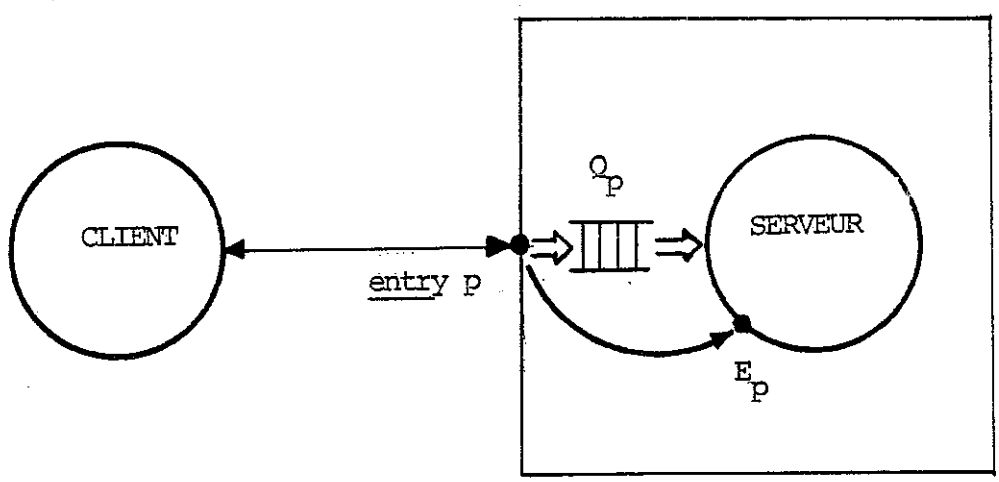
```
procédure libérer (S : in out sémaphore-multiple; R : ressource) is

        begin request (X)
              libre : = libre or R;
              trouver dans Q (en respectant l'ordre FIFO) la première requête A
              telle que A.d inclus dans libre;
          if A trouvé then retirer A de Q;
                           libre : = libre and not (A.d);
                           release (X);
                           signaler (El, A.t);
                      else release (X)
          end if
        end libérer
```

Notons que la complexité de cette représentation réside dans celle de l'ensemble Q, ce qui est un problème purement algorithmique qui ne relève pas des mécanismes du noyau SCEPTRE.

2.8 RENDEZ-VOUS DE ADA

2.8.1 Description informelle

Le langage ADA fournit un mécanisme de rendez-vous permettant à une tâche "client" de faire exécuter un service par une tâche "serveur". Le service a la forme d'une procédure qui en ADA prend le nom d'entry. Le client doit attendre que le serveur soit en mesure de le servir pour exécuter cette procédure. Lorsque c'est le cas, le client exécute la procédure et reprend son cours normal après cette exécution.
Lorsque cette procédure a été exécutée le serveur reprend son activité.



2.8.2 Description algorithmique

Pour réaliser ce rendez-vous il suffit d'associer à toute entry p une FILE Qp et deux événements Ep et Fp du serveur. Le protocole du rendez-vous est alors

```
client : begin
         .
         .
         .
         implémentation de l'invocation de l'entry p :
         ENVOYER (TACHE-COURANTE, Qp);
         SIGNALER (Ep, serveur);
         ATTENDRE (El); EFFACER (El);
         INVOQUER p avec transmission d'arguments;
         .
         .
         .
         end client

serveur : begin
         .
         .
         .
         implémentation de l'acceptation de p :

         ATTENDRE (Ep); EFFACER (Ep);
         RETIRER (T, Qp);
         SIGNALER (El, T);
         ATTENDRE (Fp); EFFACER (Fp);

         code de p :
         .
         .
         .
         p se termine par les instructions

         SIGNALER (Fp, serveur);--permettant au serveur de continuer
         return;--permettant au client de continuer en séquence
         .
         .
         .
         end serveur
```

Cas de l'attente sélective du serveur

Lorsque le serveur exécute une instruction de la forme

```
select
     when C1 =  accept p1
  or when C2 =  accept p2
  end select
```

Celle-ci peut s'implémenter sous la forme

1. calcul du vecteur booléen des événements attendus

```
A : = (1..n =  false);
A (Ep1) : = C1;
A (Ep2) : = C2;
```

2. attente sélective

```
ATTENDRE (A); —on assimile A à une liste d'événements
```

3. lorsque l'on est réveillé

```
if A (Ep1) and ARRIVE (Ep1)
           then EFFACER (Ep1);
                RETIRER (T, Qp1);
                SIGNALER (E1, T);
                ATTENDRE (Fp1);
                EFFACER (Fp1);
elsif A (Ep2) and ARRIVE (Ep2)
              then EFFACER (Ep2);
                   RETIRER (T, Qp2);
                   SIGNALER (Fp2);
                   ATTENDRE (Fp2);
                   EFFACER (Fp2)
end if
```

3 - EXEMPLES D'AGENCES

SOMMAIRE

3 EXEMPLES D'AGENCES

3.1 Agence Horlogerie

L'agence Horlogerie que nous décrivons ici est une collection de tâches et de procédures exécutant des opérations en relation avec le temps.

Trois fonctionnalités sont offertes par cette agence :

- gestion de temps relatif (intervalles de temps), à l'aide de chronomètres, la précision étant celle demandée par l'utilisateur.

- gestion du temps absolu (avec la précision désirée)

- gestion (demande ou annulation) de délais demandés par des tâches extérieures à l'Agence :
  - . signalisation unique (temporisation-délai-time out)
  - . signalisation cyclique

On accède à l'Agence selon des protocoles spécifiques (ou des procédures) définis au moment de la conception du système.

Gestion du temps relatif

L'agence Horlogerie fait évoluer des chronomètres associés à des tâches qui demandent leur gestion.
Les chronomètres évoluent à intervalle de temps, précisé à l'initialisation de la demande.

Gestion du temps absolu

Deux opérations permettent la gestion du temps absolu :

- initialisation du temps

- lecture du temps absolu

. Initialisation du temps : une opération indivisible (garantissant l'exclusion mutuelle) permet cette initialisation

. Lecture du temps absolu : une opération permet la consultation de l'heure à tout instant par n'importe quelle tâche.

Demande et annulation des événements de fin de délais

Toute tâche peut demander à l'agence Horlogerie, par échange de messages, ou appel de procédures, de lui signaler par l'événement associé à la demande l'expiration d'un délai.
Il doit être possible, à la tâche concernée ou à toute autre, d'annuler une demande de signalisation d'un évènement de fin de délai, par l'agence Horlogerie.
L'agence Horlogerie conçue sur les opérations et mécanismes SCEPTRE permet à un tâche :

- de se suspendre sur l'attente d'expiration d'un délai (temporisation)

- de se suspendre en attente d'un ou plusieurs évènements et une temporisation

- de demander la signalisation de l'expiration d'un ou plusieurs délais, sans suspension explicite

Constitution de l'Agence Horlogerie

L'Agence Horlogerie est constituée de diverses tâches (différées et immédiates) et procédures.
La liste des procédures ci-après n'est pas exhaustive :

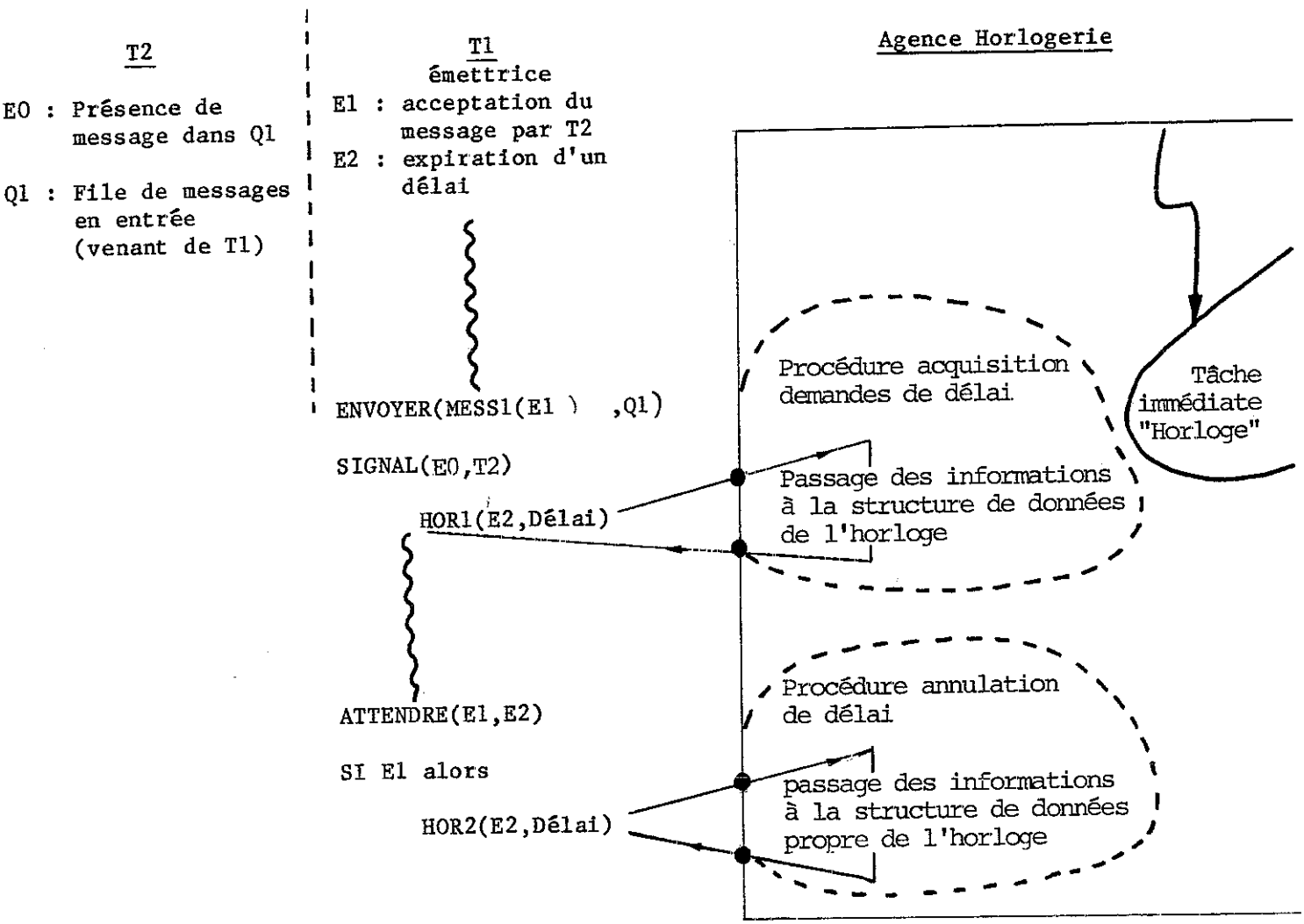
- procédure d'initialisation de demande de délai
- procédure d'annulation de demande de délai
- procédure de lecture du temps
- procédure de demande d'initialisation de temps
- etc...

La tâche centrale de l'Agence Horlogerie est une tâche "immédiate", démarrée par les interruptions de l'horlogerie matérielle (cf. Chapitre : Communication entre tâches distantes). Cette tâche fait évoluer le contenu de la structure de données propres à l'Agence Horlogerie (chronomètres, temps absolu, délais...).
Les procédures précisées ci-dessus peuvent signaler des tâches différées qui communiquent avec la tâche immédiate. Ces tâches auront pour charge la mise à jour de la structure de données de l'Agence.
Il est toutefois possible que ces procédures agissent directement sur la structure de données.

Exemple 1

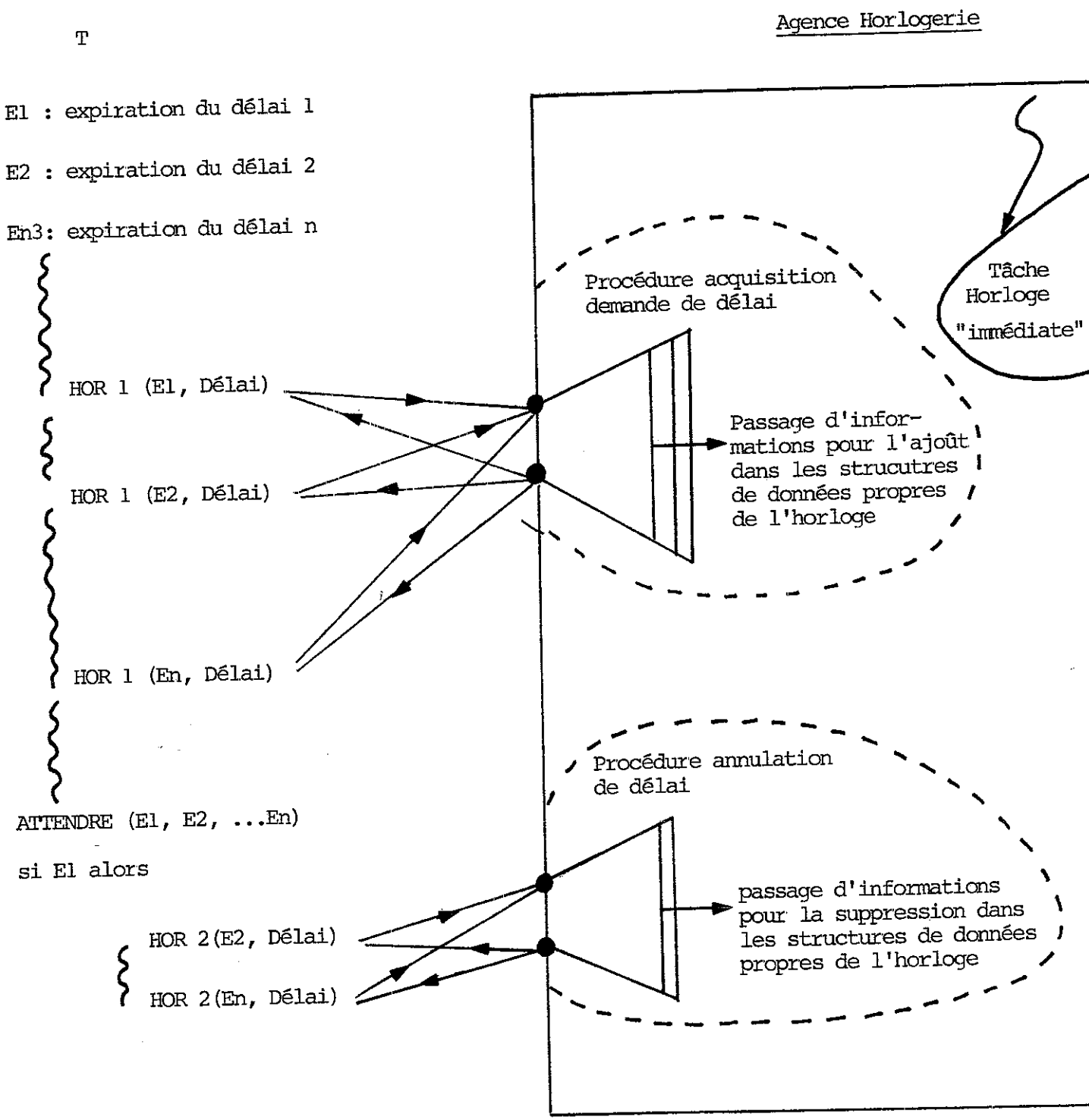
Communication entre 2 tâches, avec temporisation :

La tâche T1 attend une réponse à son message 1 (MESS 1) ou la fin d'un délai.



Exemple 2

Demande de signalisation de l'expiration d'un ou plusieurs délais



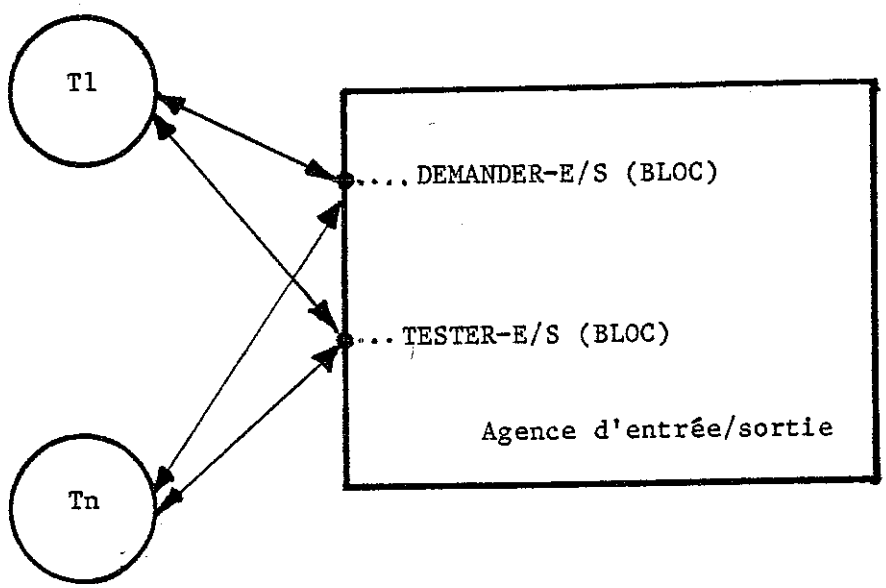
3.2 Agence d'entrée/sortie

La très grande diversité des dispositifs d'entrée/sortie empêche de donner un modèle général de description et de construction d'une agence unique pour les entrées/sorties.
Néanmoins il est possible de dégager les caractéristiques essentielles d'un dispositif d'entrée/sortie indépendamment de la complexité de sa structure.
Dans le fonctionnement d'une agence d'entrée/sortie, on distingue des activités séquentielles et des activités de communication.
Indépendamment du nombre et de la complexité des activités de communication, le processus d'entrée/sortie est toujours basé sur le modèle PRODUCTEUR-CONSOMMATEUR que l'on réalise facilement en utilisant le noyau SCEPTRE.

3.2.3 Spécification fonctionnelle de l'agence d'entrée/sortie

L'agence d'entrée/sortie ici décrite fournit aux tâches utilisatrices deux opérations :

- DEMANDER-E/S
- TESTER-E/S



L'opération DEMANDER-E/S a pour paramètre un bloc d'informations spécifiant notamment :

- l'ordre à exécuter,
- l'adresse des données à transférer (le cas échéant),
- le compte-rendu de l'opération,
- l'événement associé à la fin de l'entrée/sortie : FIN-E/S
- la durée maximum de l'exécution de l'ordre ("time-out").

L'opération TESTER-E/S permet à la tâche utilisatrice d'obtenir des informations concernant la nature des opérations d'entrée/sortie (taille optimale des tampons, type et état du périphérique).

Le protocole d'utilisation de l'agence d'entrée/sortie est de la forme :

```
tâche utilisatrice

  .
  .
  .
TESTER-E/S (BLOC) ;
DEMANDER-E/S (BLOC) ;
  .
  .
  .
ATTENDRE (FIN-E/S) ;
  .
  .
  .
```

3.2.2 Spécification logique de l'agence d'entrée/sortie

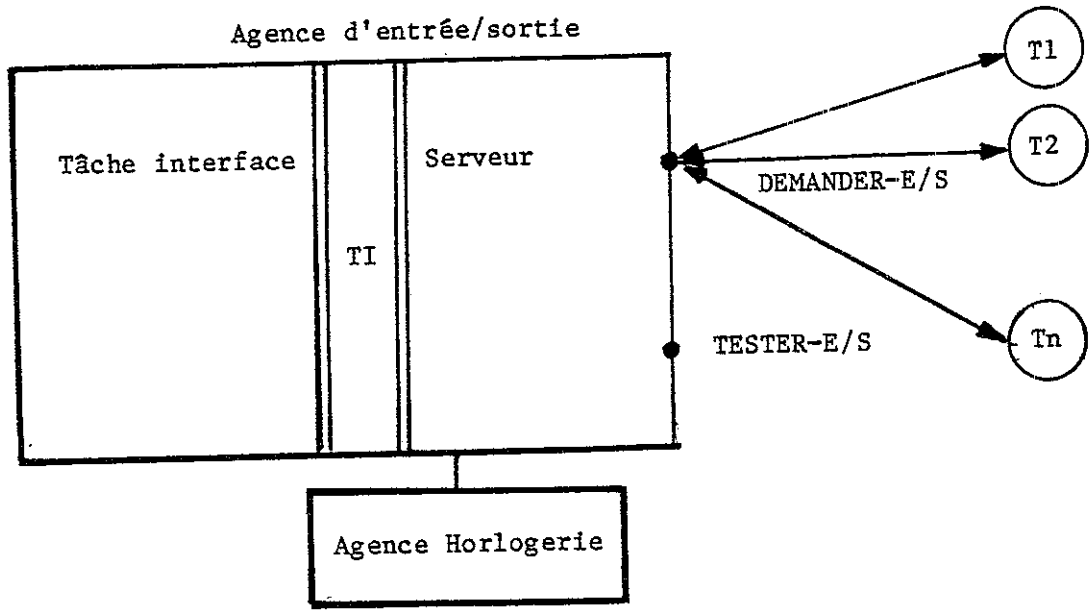
L'agence d'entrée/sortie décrite ici est constituée de trois tâches

- Tâche-Interface : tâche cablée s'exécutant dans l'unité de contrôle d'entrée/sortie,
- TI : tâche immédiate gérant l'interruption provenant de cette unité,
- Serveur : tâche différée assurant l'exécution des demandes de service à l'agence.

Les procédures DEMANDER-E/S et TESTER-E/S assurent l'interface entre l'agence et les tâches utilisatrices. Ces procédures mettent en oeuvre deux fonctions :

- PROLOGUE : complète éventuellement le BLOC pour initialiser une demande d'entrée/sortie.
- EPILOGUE : met àjour le BLOC à la fin (normale ou non) d'une opération d'entée/sortie.

Dans cet exemple l'unité de contrôle est supposée simple, c'est à dire qu'elle gère un seul périphérique. Pour des raisons de sécurité, toute demande d'entrée/sortie est associée à un mécanisme de "time-out" utilisant l'agence horlogerie.



Les interactions entre ces tâches sont décrites dans le schéma qui suit. Les significations de ces interactions sont explicitées dans les commentaires associés aux objets répertoriés suivants.

FILES

DEMANDE-E/S : tout bloc d'entrée/sortie doit être déposé dans cette file.

FIN-INTERFACE : file où la tâche immédiate dépose le message indiquant comment l'opération d'entrée/sortie s'est terminée.

EVENEMENTS

E/S : pour signaler l'arrivée d'une demande d'entrée/sortie

ORDRE : pour signaler l'arrivée d'un ordre pour l'unité de contrôle

FIN-I : pour signaler que l'unité de contrôle a terminé son travail.

FIN-E/S : fin de l'opération d'entrée/sortie.

TIME-OUT : expiration du délai spécifié dans l'opération d'entrée/sortie.

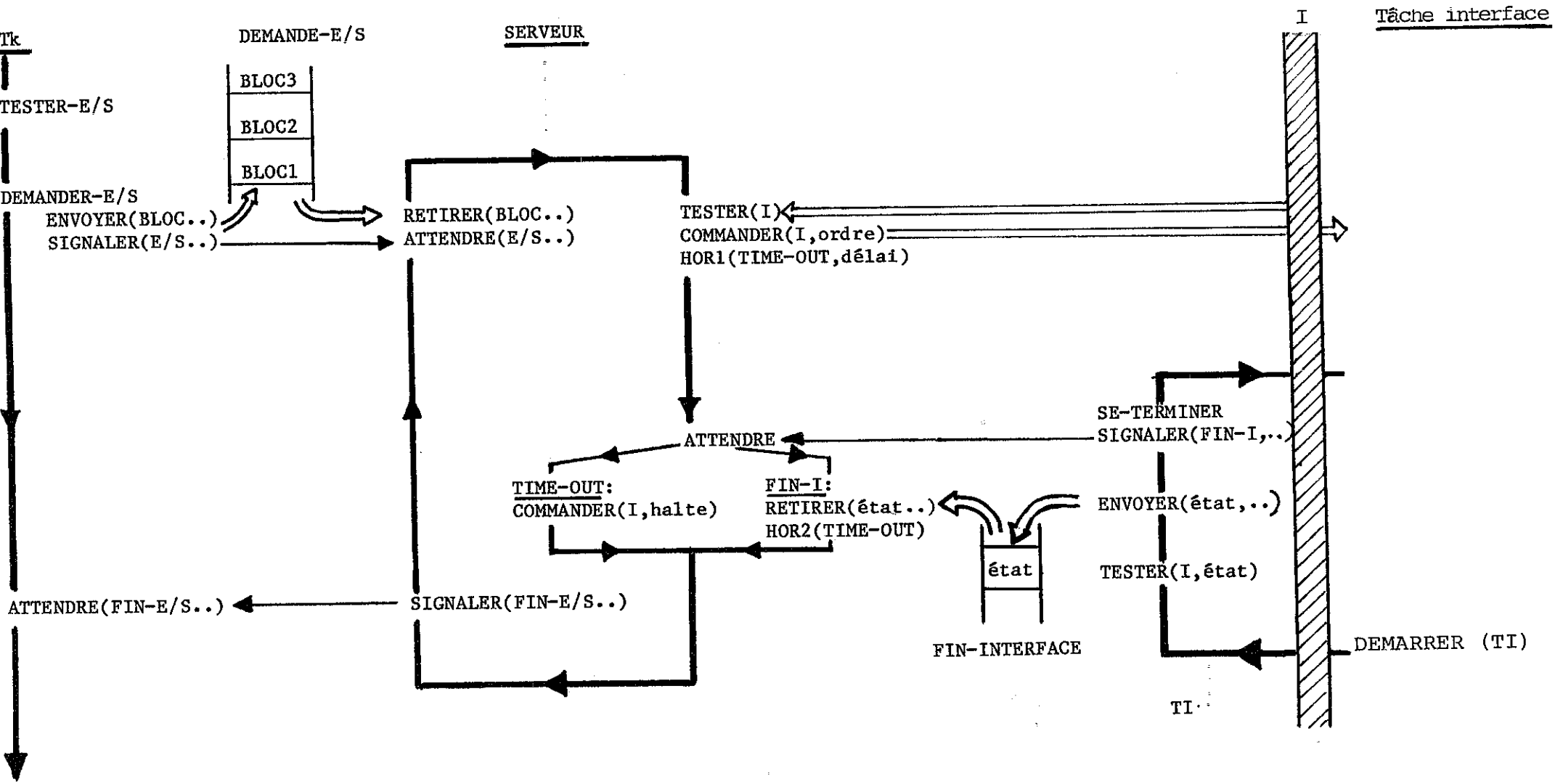
PROCEDURES

DEMANDER-E/S

TESTER-E/S

HOR1 : procédure de l'agence Horlogerie demandant la signalisation d'un délai.

HOR2 : procédure permettant d'annuler la demande précédente.



AGENCE D'ENTREE/SORTIE

4 EXEMPLES D'IMPLEMENTATION

4.1 Implémentation des régions

4.1.1 Contrôle de la préemption

Pour permettre le contrôle de la préemption du processeur à la TACHE-COURANTE on utilise deux opérations INHIBER-PREEMPTION qui interdit toute préemption du processeur sauf éventuellement, et suivant les implémentations, les préemptions telles que celles qui proviennent :

- de l'opérateur,
- de l'horloge,
- d'une panne de l'alimentation,

et AUTORISER-PREEMPTION qui annule l'inhibition précédente.

Sur monoprocesseur :
INHIBER-PREEMPTION équivaut au masquage des interruptions compte-tenu des restrictions précédentes,
AUTORISER-PREEMPTION équivaut au démasquage des interruptions.

Sur multiprocesseur, il faut distinguer les options :
. ordonnancement non préemptif : l'implémentation est identique
. ordonnancement préemptif : il faut en plus obtenir la ressource processeur.

4.1.2 Contrôle du mode d'exécution

On peut utiliser une pile pour mémoriser le parenthèsage des ENTRER et SORTIR.
Plus simplement on peut associer à chaque tâche un entier COMPTE-REGION qui est nul dans le mode Préemptible et non nul dans le mode Non préemptible.

4.1.3 Implémentation des régions sur monoprocesseur

```
type REGION is (libre, occupée) ;

procedure ENTRER (R : in out REGION) is
begin
     INHIBER-PREEMPTION;
     COMPTE-REGION : = COMPTE-REGION + 1 ;
         if R = occupée then erreur : l'attente active est interdite sur
                                      monoprocesseur .
         end if ;
     R := occupée
end   ENTRER ;


procedure SORTIR (R : in out REGION) is
begin
     R := libre ;
     COMPTE-REGION := COMPTE-REGION - 1 ;
     if COMPTE-REGION = 0 then AUTORISER-PREEMPTION end if
end SORTIR ;
```

4.1.4 Implémentation des REGIONs sur multiprocesseur

Cette implémentation suppose l'attente active. Elle nécessite l'existence d'un mécanisme permettant d'attribuer la ressource "mémoire commune" à une tâche. L'attribution de cette ressource empêche l'exécution de toute instruction adressant la mémoire commune pour le compte d'une autre tâche, jusqu'à ce que la ressource soit libérée par son propriétaire.
En général ce mécanisme n'est pas directement accessible. Il est commandé par des instructions telles que "test-and-set".
Dans la description qui suit nous utilisons ce mécanisme sous la forme
SOLO : prendre possession exclusive de la mémoire commune.
TUTTI : libérer cette propriété exclusive

Cette fois l'implémentation des régions est la suivante :

```
type REGION is (libre, occupée) ;

procedure ENTRER (R : in out REGION) is
begin
     INHIBER-PREEMPTION;
     COMPTE-REGION : = COMPTE-REGION + 1 ;
         loop SOLO ;
              if R = libre then R := occupée ;
                                TUTTI ;
                           else AUTORISER-PREEMPTION ;

                                INHIBER-PREEMPTION
              end if
         end loop ;
end    ENTRER ;

procedure SORTIR (R : in out REGION) is
begin
     SOLO  ;
     R := libre ;
     TUTTI ;
     COMPTE-REGION := COMPTE-REGION - 1 ;
     if COMPTE-REGION = 0 then AUTORISER-PREEMPTION end if
end SORTIR ;
```

4.2 Recommandation d'implémentation des événements

Le mécanisme de signalisation proposé par le noyau SCEPTRE doit permettre la minimisation des commutations de contexte lors des opérations de signalisation.
Cette minimisation n'est possible que si l'on regroupe dans une même structure de données :

- le vecteur des événements d'une tâche,
- le vecteur des événements attendus par cette tâche,

et si cette structure de données est directement adressable par le noyau SCEPTRE (sans changement de contexte).

Ce choix suppose que l'on fixe à la génération du noyau SCEPTRE le nombre maximum des événements d'une tâche.

ANNEXE 1 GLOSSAIRE ET INDEX

DEFINITIONS — PAGES

ACTIF — 32, 33

Etat d'une tâche.

AGENCE — 17, 22

On appelle agence un module (microprogrammé ou programmé) fournissant une collection d'opérations spécialisées aux programmes utilisateurs. Une agence peut exécuter ses opérations au moyen de tâches, de programmes et de données qui lui sont propres.

APPLICATION TEMPS REEL — 3

Une application Temps Réel est un système de traitement de l'information ayant pour mission de commander un environnement non programmé, en respectant les contraintes de temps et de débit (temps de réponse à un stimulus, taux de perte d'information toléré par entrées) qui sont imposées à ses interfaces avec cet environnement. Une application Temps Réel est composée de programmes s'exécutant sur une machine gérée par un Exécutif. Chacune de ces exécutions est mise au compte d'agents actifs appelés tâches.

ARRET — 19, 35

L'arrêt d'une tâche consiste à arrêter définitivement son exécution.

CANAL — 18

Un canal est un support matériel de la communication entre deux machines.

CONTEXTE D'EXECUTION — 16

On appelle contexte d'exécution d'une tâche l'ensemble des informations strictement nécessaires à un processeur pour :

- en entreprendre l'exécution,
- en reprendre l'exécution après une suspension momentanée.

Sur la plupart des machines, le contexte d'exécution comprend les registres, les bases et le mot d'état.

DEFINITIONS	PAGES
COMMUNICATION La communication entre deux tâches consiste en un échange d'informations entre ces deux tâches suivant un protocole. Le protocole fixe la forme que doit avoir cet échange pour que les deux tâches concernées se "comprennent" correctement.	19, 41 à 46
CREATION La création d'une tâche consiste à associer à un nouveau nom de tâche tous les objets nécessaires à l'exécution de cette tâche (Code, données, pile...).	19
DEMARRAGE Le démarrage d'une tâche consiste à en commencer l'exécution.	19, 35
DESTRUCTION La destruction d'une tâche consiste à effacer l'association nom de la tâche/ensemble des attributs de la tâche.	19
ETATS D'UNE TACHE Une tâche peut être dans l'un des états suivants : Inexistant Existant Non-exécutable Exécutable Hors-service En-service En-attente Actif Prêt En-cours.	32, 33

DEFINITIONS	PAGES
EN COURS Etat d'une tâche	32, 33
EXCEPTION L'exécution anormale d'un traitement peut être détectée par le matériel ou le logiciel. Les mécanismes assurant cette fonction sont appelés mécanismes d'exception. Ils peuvent utiliser différentes techniques (compte rendu, déroutement,.)	28, 35
EXCLUSION MUTUELLE Un mécanisme d'exclusion mutuelle est un moyen assurant que les exécutions d'une famille de séquences d'instructions soient disjointes dans le temps quel que soit l'ordre dans lequel ces séquences sont invoquées	20, 39
EXECUTABLE Etat d'une tâche.	32, 33
EXECUTIF Une application Temps Réel est composée de programmes s'exécutant sur une configuration matérielle gérée par un Exécutif. Les services rendus par un Exécutif ont la forme d'opérations que l'on invoque en fournissant les arguments correspondants aux paramètres de ces opérations.	8 à 11
HORLOGERIE L'horlogerie est la partie de l'Exécutif responsable de la gestion du temps. Elle peut, à la demande, fournir les services suivants : - initialisation de la date et l'heure - consultation de la date ou de l'heure - demande de signalisation (cyclique ou non) d'un évènement à l'expiration d'un délai donné - annulation d'une telle demande de signalisation.	67 à 70

DEFINITIONS — PAGES

INHIBITION DE LA PREEMPTION — 20, 32, 33, 76, 77

Inhibition de la préemption du processeur.

INTERRUPTION — 21

Mécanisme cablé de démarrage d'une tâche immédiate.

MACHINE — 15

Une machine est un environnement matériel assurant l'exécution d'une famille de processus cablés, microprogrammés ou programmés, qui possèdent en exclusivité l'accès à une mémoire commune et qui sont exécutés par un ou plusieurs processeurs.

NOYAU SCEPTRE — 2, 10 à 13

Ensemble de types et d'opérations élémentaires permettant de construire de façon structurée, performante et portable les mécanismes de coopération entre les activités parallèles d'un Exécutif Temps Réel.

OPERATION

Sous-programme cablé, microprogrammé ou programmé, invoqué en transmettant les arguments nécessaires à son exécution.

ORDONNANCEUR — 16, 35

On appelle ordonnanceur un module (cablé, microprogrammé ou programmé) chargé de gérer un ensemble de processeurs identiques pour le compte d'une famille de tâches.

PREEMPTION — 20, 32, 33, 76, 77

Action consistant à prendre le processeur d'une tâche pour l'attribuer à une tâche prête plus prioritaire ou immédiate.

PRET — 32, 33

Etat d'une tâche.

DEFINITIONS — PAGES

PRIORITE D'UNE TACHE — 16, 25, 26, 33

La priorité d'une tâche est l'attribut de cette tâche qui est pris en compte par l'algorithme de choix lors d'une demande d'activation la concernant.

SIGNALISATION — 19, 37, 38

Un mécanisme de signalisation est un moyen permettant à une (ou plusieurs) tâche (s) d'attendre qu'une condition associée à ce mécanisme soit satisfaite, et à un agent actif évaluant cette condition de la signaler lorsqu'elle est satisfaite. La signalisation permet d'exprimer des protocoles de synchronisation.

SYNCHRONISATION — 19

Un mécanisme de synchronisation est un moyen de contrôler l'ordre d'exécution des instructions entreprises par une famille de tâches.

TACHE — 15, 16

Une tâche est un agent actif responsable de l'exécution par une machine d'un programme composé à partir du répertoire des instructions de cette machine.
Une tâche ne peut entreprendre l'exécution d'une instruction de ce programme qu'après avoir terminé l'exécution de l'instruction précédente. En ce sens elle est séquentielle.

TACHE IMMEDIATE — 21, 44

Une tâche immédiate est une tâche programmée gérant une interface entre l'application et son environnement. Une tâche immédiate ne peut jamais se mettre en attente d'une condition.

TACHE DIFFEREE — 21

Une tâche différée est une tâche qui peut se mettre en attente d'une condition.

ANNEXE 2 OUTILS DE SPECIFICATION DE DETAIL

INTRODUCTION

Les outils de spécification locale servent à représenter l'évolution des tâches d'une application Temps Réel, leurs interactions et leur relation avec les automates (cablés ou autres) existant dans l'environnement d'exécution de l'application. Ils constituent une base suffisamment rigoureuse pour spécifier le noyau SCEPTRE et la bibliothèque associée.

CONCEPTS DE BASE

VARIABLE

Une variable représente l'une des entités passives d'une application Temps Réel.

exemples :
- variable d'un programme
- registre d'entrée/sortie,
- contexte d'exécution d'une tâche.

A chaque instant toute variable possède une valeur, en particulier à l'instant initial.
Pour définir les propriétés d'une variable on adopte la notation :

nom de la variable : type := valeur initiale

où type spécifie l'ensemble des valeurs possibles de la variable.

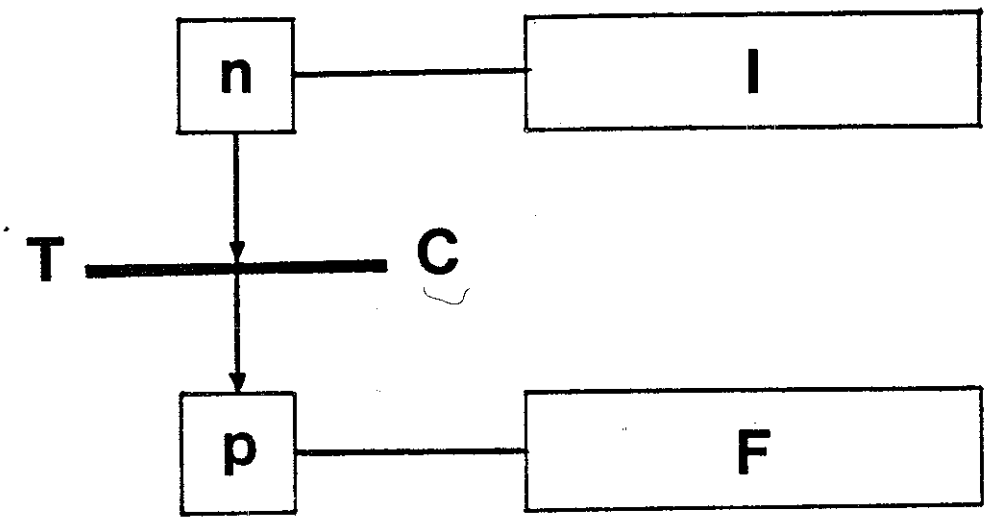
Dans ce modèle on suppose que les variables varient de façon discrète dans le temps.

TRANSITION

Toute modification de la valeur d'une variable résulte du franchissement d'une transition.

Toute transition est caracterisée par :
- un identificateur éventuel,
- l'ensemble des variables qu'elle manipule,
- une condition de franchissement portant sur ces variables et indiquant lorsqu'elle est vraie que la transition peut être franchie.
- l'effet de la transition sur les variables manipulées.

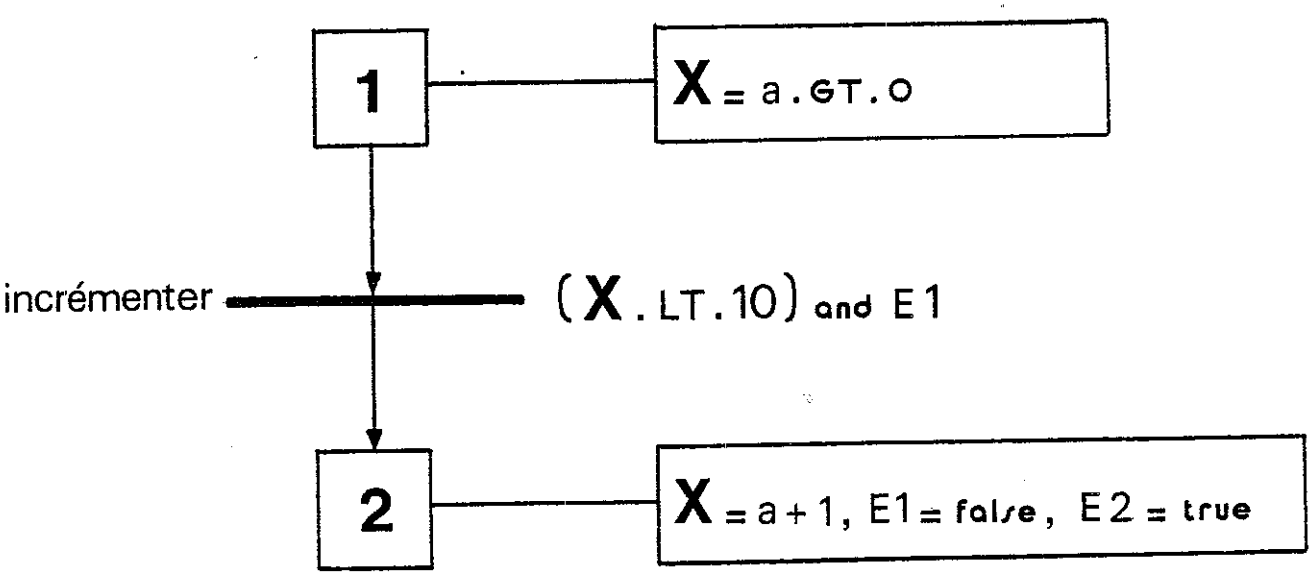
On adopte la représentation graphique proposée par GRAFCET (25)



T : Nom de la transition
C : condition de franchissement portant sur les variables
n : numéro de l'état initial
I : caractérisation de l'état des variables au moment où la transition T a été choisie pour être franchie
p : numéro de l'état final
F : caractérisation de l'état des variables après franchissement de la transition.

Exemple de transition

variables manipulées E1, E2 : boolean := false
X : integer := 0

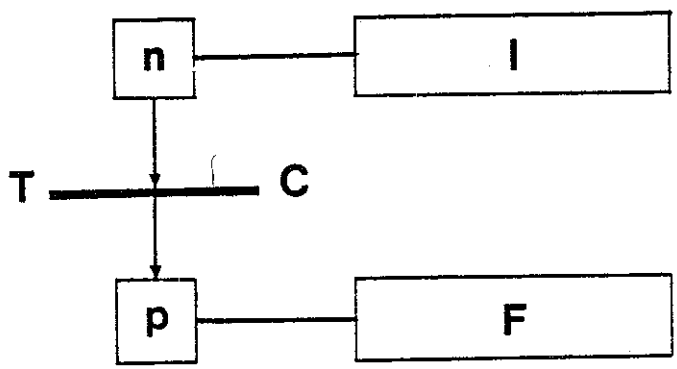


Les variables booléennes E1 et E2 servent au contrôle : E1 = true indique que l'on a demandé le franchissement de la transition; E2 = true indique que la transition "INCREMENTER" a été franchie.
La condition de franchissement (X .LT. 10) and E1 indique que la transition ne peut être franchie que lorsque

. X .LT. 10
. E1 = true : on a demandé le franchissement.

NOTATION IMPORTANTE

On associe à toute transition T



les variables de contrôle booléennes En Ep ayant comme dans l'exemple précédent la signification

En = true : on a demandé le franchissement de la transition T

Ep = true : la transition a été franchie

La condition de franchissement est alors de la forme :

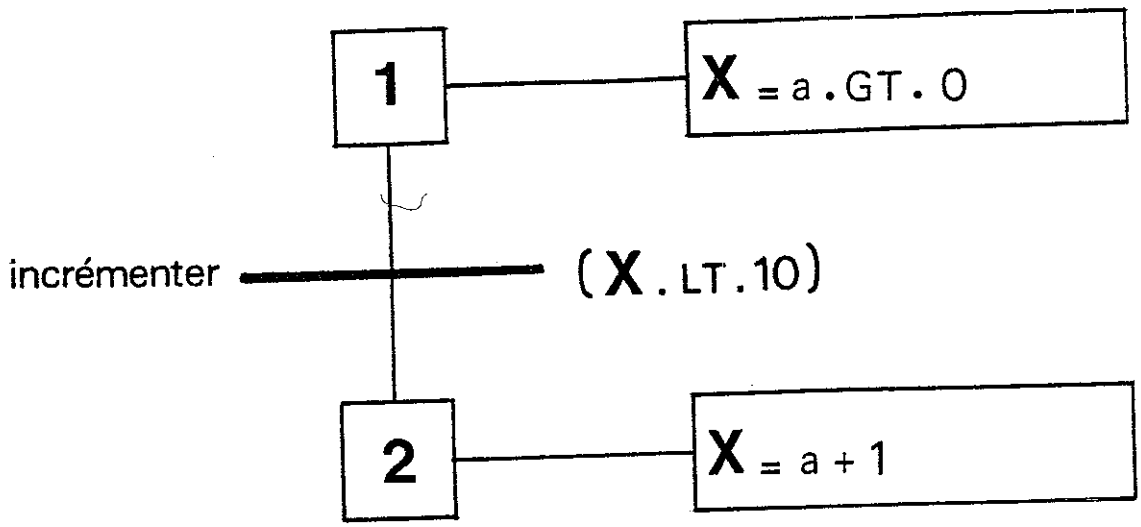
En and ...

L'état final est de la forme

En = false, Ep = true,...

Pour allèger les notations on ne mentionne pas ce qui touche En et Ep lorsqu'il n'y a aucune ambiguité.

Dans l'exemple précédent on écrira :

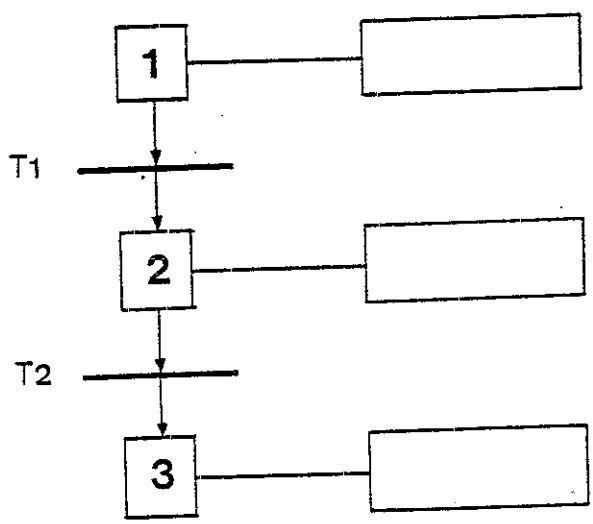


Notons que l'absence de condition de franchissement C signifie que la seule condition de franchissement est implicitement :

En

COMBINAISONS DE TRANSITIONS

SEQUENCE

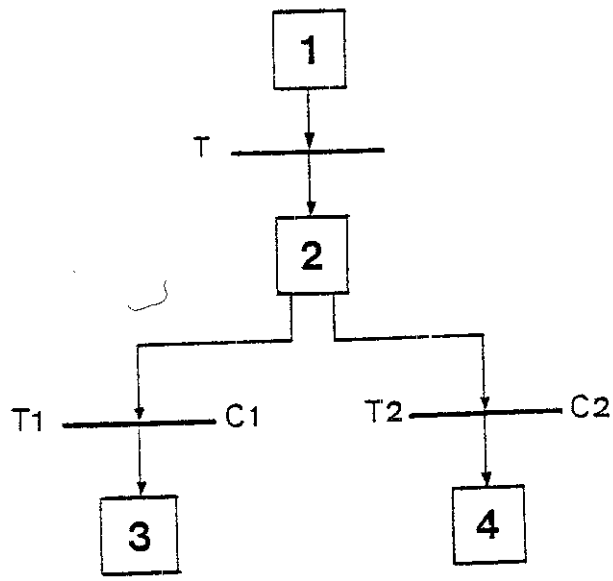


La mise en séquence des transitions T1 et T2 signifie que T2 ne peut être franchie que lorsque T1 l'a été. En termes des variables de contrôle précédentes, cela signifie que la condition de franchissement de T2 est de la forme explicite :

E2 and...

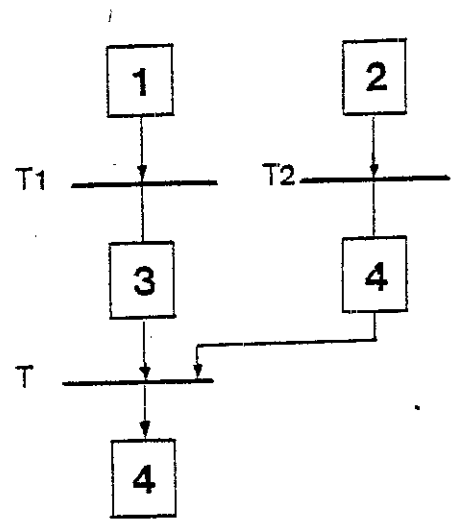
Sachant que E2 n'est mis à la valeur true que par la seule transition T1.

SELECTION



Lorsque T a été franchie l'une (et une seulement) des transitions T1 ou T2 peut être franchie suivant que l'une des conditions exclusives C1 ou C2 est vraie.
La sélection s'étend au cas de plus de deux transitions T1 et T2. Dans cette extension les conditions de franchissement sont mutuellement exclusives.

REGROUPEMENT



La transition T ne peut être franchie que si T1 ou (exclusif) T2 ont été franchies.

PROCESSUS SEQUENTIEL

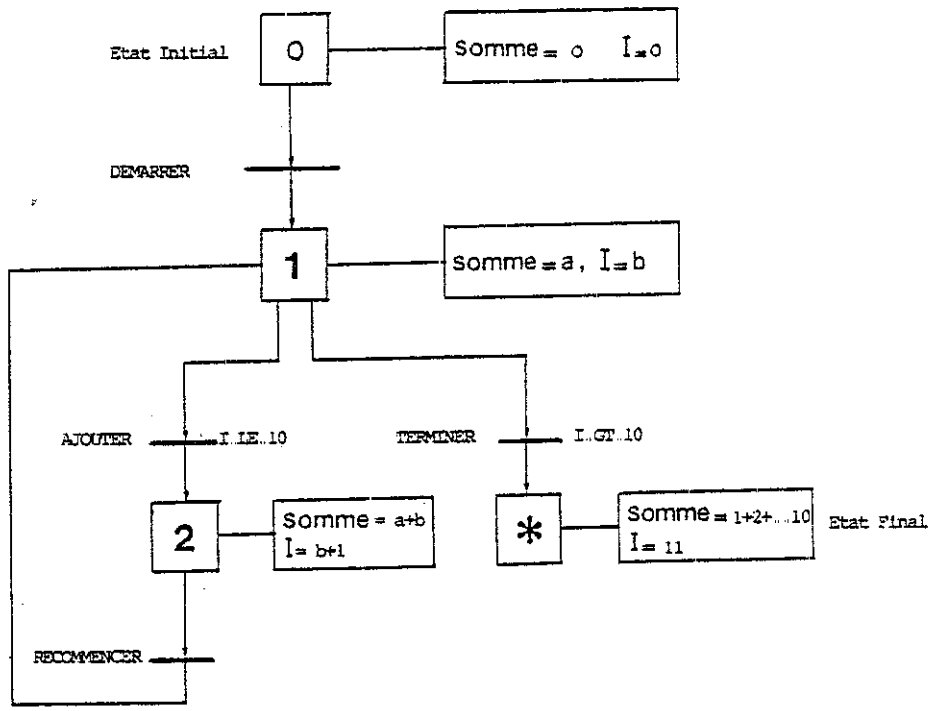
Un processus séquentiel est un ensemble de transitions comportant un état initial numéroté 0 et un état final numéroté * ayant les propriétés suivantes :

. Toutes ces transitions sont liées exclusivement au moyen de SEQUENCES, de SELECTIONS ou de REGROUPEMENTS.

. Toute transition fait partie d'au moins une séquence de transitions menant de l'état 0 à l'état *.

Exemple de processus séquentiel

Calcul de la somme des 10 premiers nombres
variables SOMME,I : integer : = 0;



RESEAU DE TRANSITIONS

Un réseau de transitions est une collection de variables et de processus séquentiels qui les manipulent.

EVOLUTION D'UN RESEAU DE TRANSITIONS

ETAT INITIAL DU RESEAU

Il est défini de la manière suivante :

. Toutes les variables du réseau ont leur valeur initiale

. Tous les processus du réseau sont dans leur état initial (pour chaque processus E0 = true)

ETAT COURANT DU RESEAU

Toutes les variables ont une valeur bien déterminée et chaque processus est dans l'état n caractérisé par En = true.

EVOLUTION ELEMENTAIRE

A partir de l'état courant choisir parmi les seules transitions franchissables UNE TRANSITION UNIQUE et la franchir. Ceci modifie la valeur des variables manipulées par cette transition ainsi que celle-ci le spécifie.

ETAT FINAL DU RESEAU

Le réseau atteint son état final lorsque tous les processus sont dans leur état final.

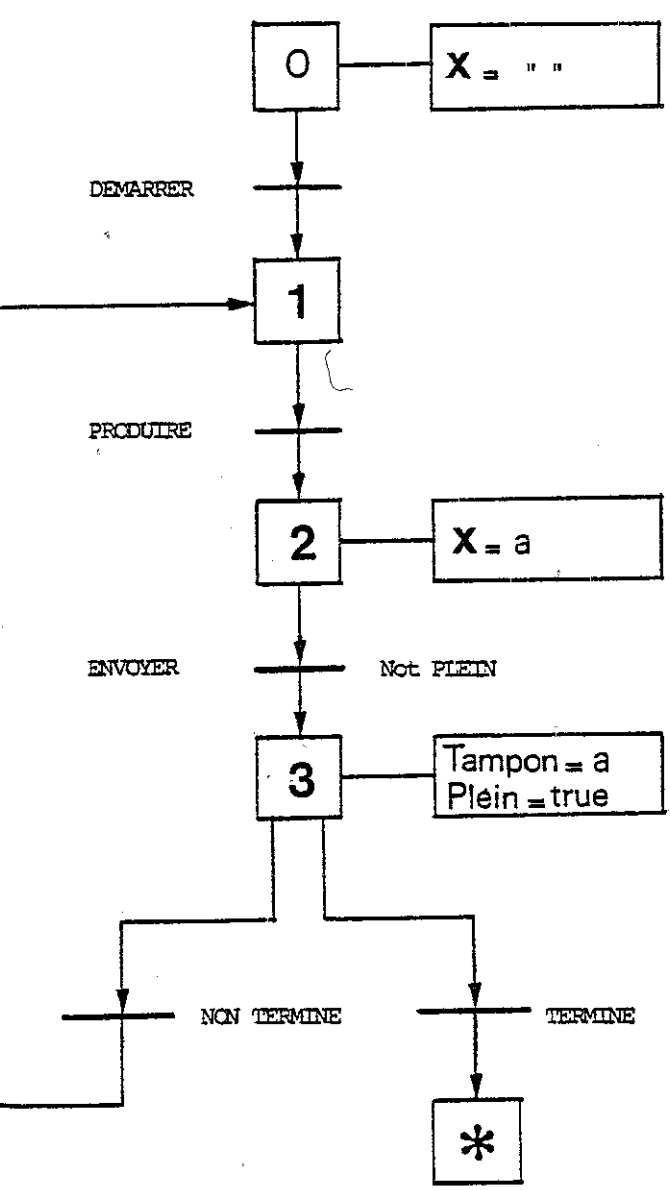
BLOCAGE

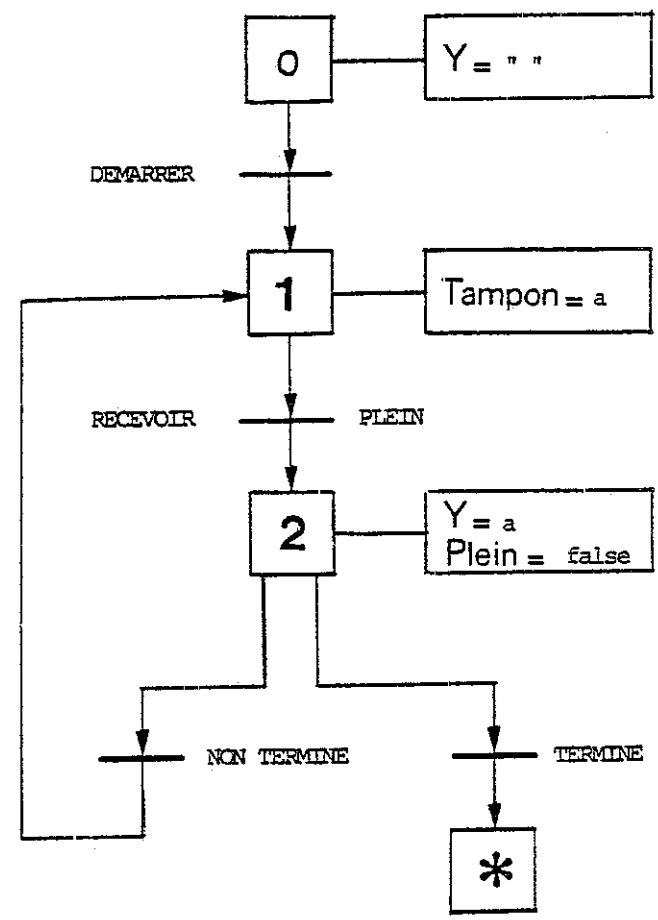
Le réseau est bloqué lorsqu'il n'est pas dans son état final et qu'aucune transition n'est franchissable.

EXEMPLE SIMPLE DE RESEAU

PRODUCTEUR CONSOMMATEUR

```
variables X,Y,TAMPON : character : = " ";
          PLEIN : boolean : = false;
```

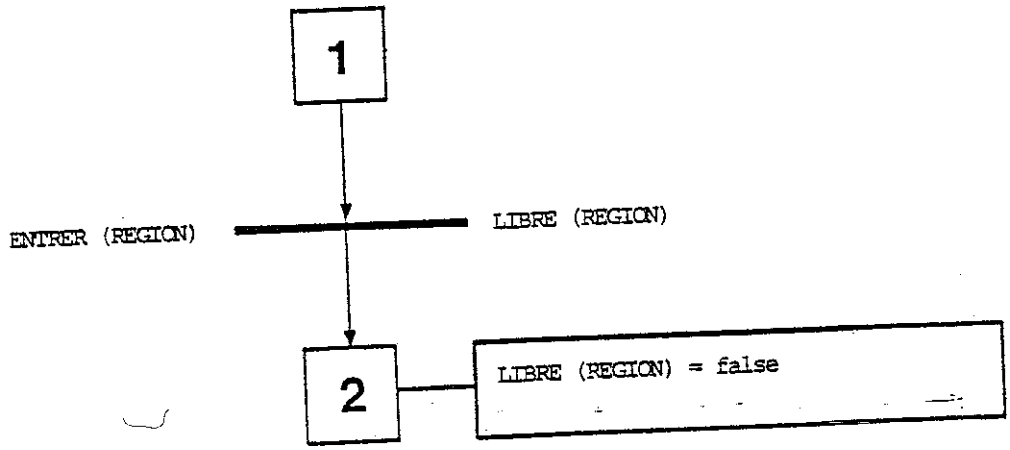


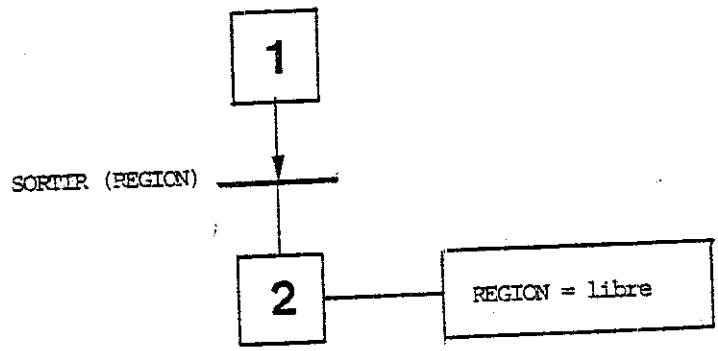


Dans ce modèle on voit que l'exclusion mutuelle des accès au tampon est garantie par la règle d'évolution élémentaire. La signalisation est réalisée très simplement au moyen de transitions conditionnelles portant sur la variable PLEIN.

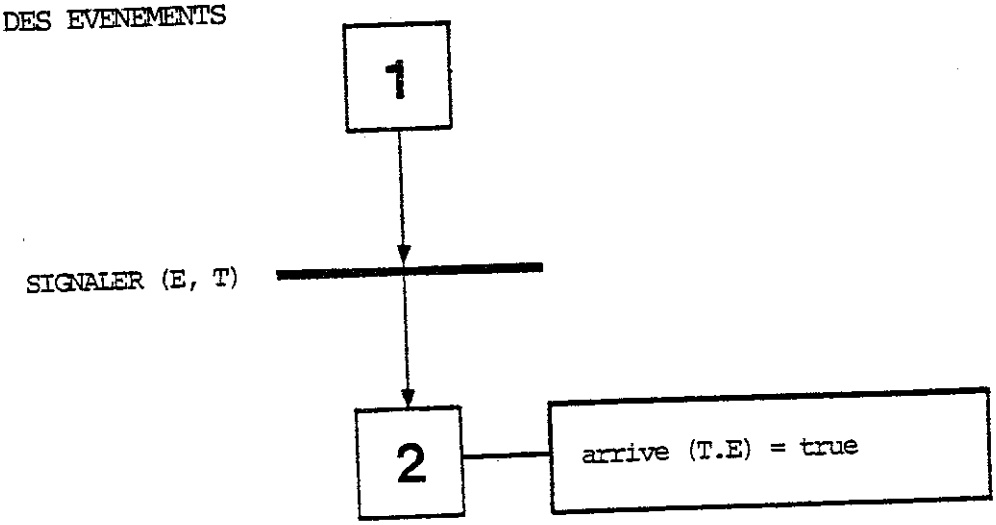
XEMPLES DE SPECIFICATION DES OPERATIONS ELEMENTAIRES DU NOYAU

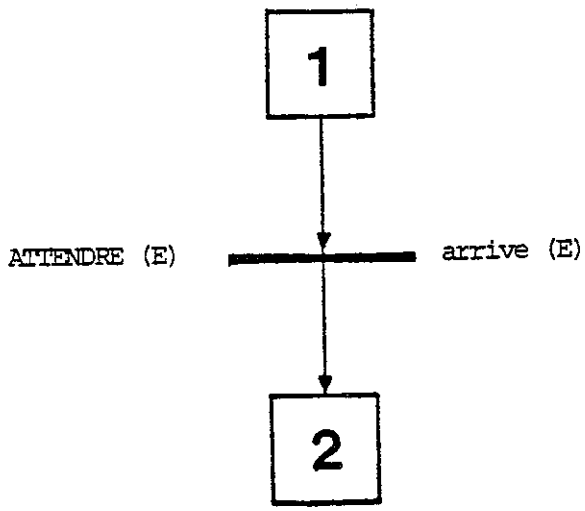
ANIPULATION DES REGIONS

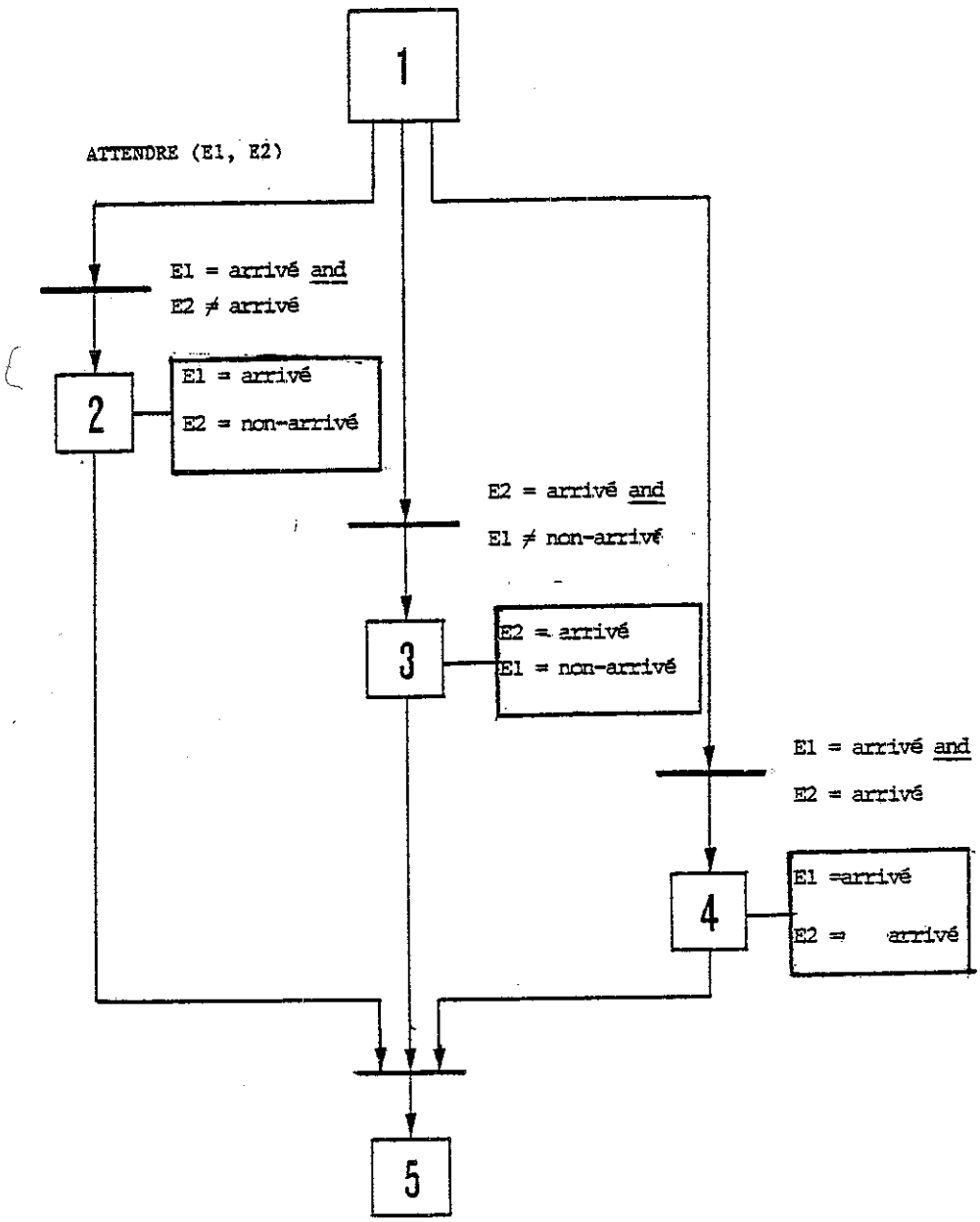




MANIPULATION DES EVENEMENTS




1
ATTENDRE (E)
arrive (E)
2


1
ATTENDRE (E1, E2)
E1 = arrivé and
E2 ≠ arrivé
2
E1 = arrivé
E2 = non-arrivé
E2 = arrivé and
E1 ≠ non-arrivé
3
E2 = arrivé
E1 = non-arrivé
E1 = arrivé and
E2 = arrivé
4
E1 =arrivé
E2 = arrivé
5

ANNEXE 3 : LISTE DES PARTICIPANTS AU PROJET SCEPTRE

GROUPE OPERATIONNEL

Monsieur FEBVRE
SEMS
Rue de Provence
38130 ECHIROLLES

Monsieur FAULLE
CIMSA
10-12 avenue de l'Europe
78140 VELIZY VILLACOUBLAY

Mademoiselle FALLOUR
CAP SOGETI LOGICIEL
5, rue Louis Lejeune
92120 MONTROUGE

Monsieur MEMMI
1bis, rue Morère
75014 PARIS

Monsieur VOJNOVIC
Conseiller en informatique
15 bis rue Marc Sangnier
92290 Chatenay Malabry

Monsieur MINEL
ECA AUTOMATION
315, Bureaux de la Colline
92210 SAINT CLOUD

Monsieur J.J. SIMON
CRCAM de la Mayenne
Route de Nantes
53000 LAVAL

Monsieur KRONENTAL
INRIA
Domaine de Voluceau-Rocquencourt
78150 LE CHESNAY

GROUPE OBSERVATEUR

Monsieur BONNARD
STERIA
26, avenue de l'Europe
78140 VELIZY
VILLACOUBLAY

Monsieur CHEVANCE
CII HB
68, route de Versailles
78430 LOUVECIENNES

Monsieur BUISSON
Centre Technique Informatique
INRIA
Domaine de Voluceau
78150 LE CHESNAY

Monsieur HELEIN
SEMS
36-38, rue de la
Princesse
78430 LOUVECIENNES

Monsieur PONCET
SESA
30, Quai National
92806 PUTEAUX

Monsieur HO
CERCI
56, rue Roger Salengro
94120 FONTENAY SOUS BOIS

Monsieur MASSON
CIMSA
10-12 avenue de l'Europe
78140 VELIZY
VILLACOUBLAY

Monsieur SAVOYSKY
LCPC
58, boulevard Lefèvre
75732 PARIS Cedex 15

Monsieur MAESTRACCI
CIT ALCATEL
10, rue Latécoère
78140 VELIZY VILLACOUBLAY

CORRESPONDANTS

Monsieur REMER
CAP SOGETI LOGICIEL
5, rue Louis Lejeune
92120 MONTROUGE

Monsieur OMNES
Dpt SLC/PGL
BP 40 - CNET
22301 LANNION Cedex

Monsieur DOUSSY
SEMS
Rue de Provence
38130 ECHIROLLES

Monsieur DERRINIC
Dpt STI/SME
BP 40 - CNET
22301 LANNION Cedex

Monsieur DESCLAUD
Dpt STI/SME
BP 40 - CNET
22301 LANNION Cedex

Monsieur HANNE
CNET
38-40, avenue du Général Leclerc
92130 ISSY LES MOULINEAUX

REFERENCES

1 Hoare C.A.R;, "Monitors : an operating system structuring concept", C.ACM, Oct. 1974.

2 Vojnovic D., "Kernel Real-time System IFAC, IFIP Eindhoven, Pergamon Press 1977.

3 Hoare C.A.R., "Communicating sequential processes", C.ACM, Aug.1978.

4 Reference manual for the GREEN programming language", Cii Honeywell Bull, Feb. 1978.

6 The official definition of MASCOT, Infotech special seminar, Oct. 1978.

7 Technical Committee n°8 of International Purdue Workshop on Industrial Computer Systems, "Up to date report", Sep. 1978.

8 Fisher D., "DoD's Common Programming Language Effort", Computer, Mar. 1978.

9 Ford W.S., "Implementation of a generalized Critical Region Construct"", IIE transactions on Software Engineering, Nov. 1978.

10 Petri C.A., "Concept of net theory", Proc. of Symposium and summer school, High Tatras Math. Inst. of the Slovak Academy of Science, Bratislava, Sep. 1973.

11 Dijkstra E.W., "Guarded Commands, non determinacy and formal derivation of programs" C.ACM, Aug. 1975.

13 Industrial computer system FORTRAN procedures for Executive functions, process input/output and bit manipulation, ISO/TC 97/SC 5/WG 1 N86/DP 6705, Dec. 1978.

14 LTR, draft standard AFNOR n° Z65350, Apr. 1979.

15 Official definition of CORAL 66, British Ministery of Defence 1970.

16 Basic PEARL, draft standard DIN 66253 Teil 1 Dec. 1978.

17 Proposal of the EWICS/TC 1 on Industrial Real-Time FORTRAN, Mar. 1979.

18 ISO/TC 97/SC 5/WG 1 N78, Aug. 1978.

19 MODULA 2, ETH of Zurich report, by N. Wirth Dec. 1978.

20 Sten Andler, "Synchronization primitives and the verification of concurrent programs", Second Colloque International sur les systèmes d'exploitation, IRIA, Oct. 1978.

21 Brinch Hansen, "Concurrent programming concepts" Computing Surveys, vol.5, Dec. 1973.

22 Reference manual for Ada, Cii Honeywell Bull Apr. 1979.

23 Reference manual for the RED language, Intermetrics, Apr. 1979.

24 Towards a discipline of Real-time programming by N. Wirth, ETH of Zurich, 1976.

26 Habermann A.N., "Path expressions", Carnegie Mellon University, Jun. 1975.

27 J. Bezivin,..., Un noyau standard pour systèmes industriels, Premier symposium européen sur l'informatique en Temps Réel et le contrôle des processus, Berlin Ouest, Octobre 1979.

28 M. Kronental, Towards the standardization of Real-time operating system kernels, IFAC SOCOCO, June 1979.