É DE NANCY I

CENTRE DE RECHERCHE EN INFORMATIQUE
DE NANCY

THÈSE

présentée

A L'UNIVERSITÉ DE NANCY I

pour obtenir le titre de

DOCTEUR INGÉNIEUR EN INFORMATIQUE

par

Mohamed BENMAÏZA

LE CONCEPT D'ÉVENEMENT DANS LA SPÉCIFICATION ET LA PROGRAMMATION D'APPLICATIONS TEMPS RÉEL

Soutenue publiquement le 12 mars 1984, devant la Commission d'Examen:

J.C.DERNIAME	Président
C.IUNG	Examinateurs
P.LADET	
J.C.MATHIEU	
J.P.THOMESSE	

NIVERSITÉ DE NANCY I

CENTRE DE RECHERCHE EN INFORMATIQUE
DE NANCY

THÈSE

présentée

A L'UNIVERSITÉ DE NANCY I

pour obtenir le titre de

DOCTEUR INGÉNIEUR EN INFORMATIQUE

par

Mohamed BENMAÏZA

E CONCEPT D'ÉVENEMENT DANS LA SPÉCIFICATION ET LA PROGRAMMATION D'APPLICATIONS TEMPS RÉEL

Soutenue publiquement le 12 mars 1984, devant la Commission d'Examen:

J.C.DERNIAME	Président
C.IUNG	Examinateurs
P.LADET	
J.C.MATHIEU	
J.P.THOMESSE	

Au terme de deux ans et demi de travail, je veux remercier tous ceux qui m'ont aidé, de près ou de loin à réaliser cette thèse.

Mes remerciements vont tout d'abord à Monsieur le Professeur J.C. DERNIAME, Directeur du Centre de Recherche en Informatique de Nancy pour toutes les suggestions et remarques constructives qu'il m'a formulées et m'ont permis d'améliorer ce travail et pour le grand honneur qu'il me fait d'accepter de présider ce jury.

Que Monsieur le Professeur J.P. THOMESSE soit remercié pour m'avoir accueilli dans son équipe et accepté de diriger cette recherche et pour toute l'amitié qu'il m'a témoignée. Je le remercie également pour toutes les suggestions qu'il a pu me faire et sans lesquelles ce travail n'aurait pu être mené à terme.

Monsieur C. IUNG, Professeur à l'ENSEM (Nancy) a bien voulu lire ce rapport et apporter un jugement d'automaticien. Qu'il en soit remercié.

Je remercie Monsieur P. LADET, Professeur à l'E.N.S.I.E.G. (Grenoble) et dont les travaux ont été à l'origine d'une partie de cette thèse, pour avoir bien voulu juger ce travail et faire partie de ce jury.

Je tiens également à remercier J.C. MATHIEU, Directeur du Département Automatisation à la SOLLAC d'avoir bien voulu s'intéresser à ce travail et apporter un jugement d'"industriel".

./...

Mes remerciements vont aussi à tous les membres du CRIN (Nancy) (je n'en nommerai aucun pour être sûr de ne pas en oublier !) pour toute leur amitié.

Enfin, je tiens à remercier Mme M.T. DRIQUERT et Melle M. TESOLIN pour leur extrême gentillesse et leur conscience dans le travail et sans l'aide desquelles la réalisation technique de ce document n'aurait pas pu se faire dans des délais aussi brefs. Qu'elles m'excusent aussi de les avoir un peu bousculées.

PLAN

INTRODUCTION

La dénomination "application en temps réel" peut regrouper plusieurs aspects, aussi est-il important d'en préciser le sens dès à présent. Dans le domaine du temps réel, nous nous intéressons plus spécifiquement au domaine du contrôle de procédés industriels et lorsque nous parlerons d'applications, ce sera d'applications en contrôle de procédés industriels que nous noterons ACPI.

Dans un tel contexte, le concept d'événement est un concept largement répandu et est intégré dans beaucoup de langages orientés vers le temps réel. Un événement est alors perçu comme étant un objet purement interne servant essentiellement à la signalisation entre les différentes entités composant une application. Dans certains travaux cependant ([LAD 82] par exemple), on peut noter que les événements peuvent également être utilisés pour prendre en compte, d'une certaine façon, l'évolution du procédé.

Toutefois, il nous semble que le concept d'événement a été utilisé essentiellement dans le domaine de la programmation des ACPI et qu'il n'y a pas, à notre connaissance, de tentative pour utiliser ce concept dans le domaine de la spécification d'ACPI.

Lorsque l'on sait qu'une spécification d'ACPI peut se définir comme étant la description, à partir du cahier des charges, du comportement d'une application en fonction de l'évolution du procédé contrôlé, on comprend mieux l'intérêt que l'on peut porter au concept d'événement comme outil intervenant au cours d'une spécification d'ACPI.

Il s'agit alors de ne plus percevoir un événement comme objet de signalisation seulement, mais comme objet servant à décrire l'évolution du procédé.

Outre, donc, le fait que le concept d'événement est utile dans la programmation des ACPI, il peut également constituer une ouverture intéressante vers le domaine de la spécification d'ACPI.

De ce fait, il est nécessaire de généraliser ce concept, pour une description plus précise de l'évolution du procédé, et d'essayer d'en donner une définition précise.

Cette thèse constitue, alors, une contribution à l'étude du concept d'événement dans la spécification, mais aussi dans la programmation d'ACPI.

En partant des besoins rencontrés dans la spécification et programmation d'ACPI, la justification du concept d'événement fait l'objet du chapitre I. Comme la tendance actuelle, et de nombreux travaux dans ce domaine le montrent bien, est à l'utilisation des concepts de communication et d'entités communicantes, il nous a semblé utile de situer notre démarche par rapport à celles déjà proposées. Une comparaison plus complète sera donnée après les développements des chapitres suivants (cf. IV.9).

Dans le chapitre II, nous commençons par faire une étude générale du concept d'événement et en proposons une définition. Nous tenterons de mettre en évidence les trois aspects d'un événement qui nous semblent importants : la production, la détection et l'utilisation.

Nous terminerons ce chapitre par une étude bibliographique comparative qui nous permettra de voir ce qu'il faut ajouter au concept d'évenement pour en faire un outil apte à la spécification.

Les chapitres III, IV et V précisent l'aspect spécification à travers notamment les définitions précises des opérations que l'on peut faire sur les événements.Les exemples donnés dans le chapitre VI viendront étayer nos propositions.

Si ce travail est plus axé sur la spécification, nous ne perdons pas de vue l'aspect programmation et la réalisation d'un traducteur du langage proposé en FORTRAN temps-réel, présenté au chapitre VII, viendra conforter ce point de vue.

Au chapitre VIII seront abordés quelques problèmes relatifs au concept d'événement lorsqu'il est considéré dans un milieu réparti.

Nous noterons, pour terminer cette introduction, que ce travail, sans présenter d'aspects aussi formels, peut être rapproché de celui de [HALB & al 82]. Si dans [HALB & al 82], le comportement d'une application est modélisé par l'intermédiaire d'un système d'équations et d'inéquations définies dans une algèbre des événements appelant à la recherche analytique d'une solution du système, nous nous orientons plus vers une spécification de ce même comportement sous une forme plus constructive (car permettant de construire une solution). Une telle spécification peut alors faire appel à la simulation comme nous le suggérons dans la conclusion du chapitre VII, pour effectuer certaines vérifications.

CHAPITRE I :

SPÉCIFICATION PAR ÉVÉNEMENT : POURQUOI

I.1.- INTRODUCTION

Dans ce premier chapitre, nous allons essayer de mettre en évidence un certain nombre de concepts qui à notre avis doivent être intégrés dans un langage se voulant à la fois de spécification et de programmation d'Applications en Contrôle de Procédés Industriels (ou ACPI), l'aspect programmation n'étant validé qu'à travers l'existence d'un générateur de code (cf. chapitre 7).

Si l'intérêt de cette démarche est de rapprocher par le biais d'un seul langage, les aspects spécification et programmation d'ACPI, il n'en demeure pas moins que le problème majeur réside dans la définition de concepts communs entre le niveau programmation et le niveau spécification.

Les ACPI étant par nature distribuées et nous situant dans une période de développement et proposition de langages tels que ADA [ICH & al 79, KRU & al 81] centré sur le concept de communication, il nous a paru opportun d'essayer de voir si de tels langages, ne pouvaient pas répondre à nos préoccupations.
Démarrant donc d'une analyse des besoins pour la programmation des ACPI que nous classons en deux catégories : besoins de base en ce qui concerne l'aspect proprement "temps réel", et besoin de structuration en ce qui concerne l'aspect "génie logiciel", il nous a paru intéressant d'établir quelques comparaisons avec le développement des langages de programmation des systèmes opératoires étant donné la similitude des concepts exprimés dans ces deux domaines. Ceci nous permettra en particulier :

- de mieux comprendre l'influence réciproque que les développements de ces deux domaines ont pu avoir l'un sur l'autre (commander un ordinateur n'est-il pas déjà commander un procédé ?)
- et de là, d'expliquer en partie l'aboutissement à la définition de langages uniques permettant à la fois la programmation d'ACPI et de systèmes opératoires comme le but visé dans

la définition du langage ADA par exemple,

- mais aussi de montrer certaines limites de tels langages centrés sur la communication quant à l'expression de concepts purement temps réel.

Afin d'apporter une solution à un certain nombre de problèmes que nous soulevons, nous proposons alors de compléter l'aspect communication par le concept d'événement.

Nous donnons, à la fin de ce chapitre, l'énoncé de deux exemples qui vont servir de base pour étayer nos propositions même si pour certains concepts on fera plutôt appel à certains exemples plus "académiques".

Le choix de ces deux exemples a été motivé par le fait que :

- le premier est l'exemple-test du groupe de travail "temps réel" EWICS-TC 11 [SAVØ 83].
- le deuxième est l'exemple de la pompe de Kramer [KRAM & al 82] qui également sert d'exemple-test dans nombre de travaux notamment dans le domaine de la communication. Ceci a pour avantage d'établir une base de comparaisons.

I.2.- BESOINS POUR LA SPECIFICATION ET PROGRAMMATION D'UNE ACPI

I.2.1.- Caractérisation d'une ACPI

Une ACPI est destinée à contrôler un procédé qui :

- possède sa propre dynamique que nous appelons évolution du procédé en fonction du temps,
- est souvent géographiquement réparti,
- impose des contraintes de temps parfois sévères exprimées sous forme de temps de réponse de l'application à des stimuli émanant du procédé,

- impose une grande sûreté dans le fonctionnement.

Ces caractéristiques permettent à leur tour de mettre en évidence un certain nombre de besoins qu'il est nécessaire d'exprimer dans un langage de spécification et programmation d'ACPI.

Ces besoins peuvent être classés en deux catégories :

- besoins de base qui concernent l'aspect temps réel.
- besoins en structuration touchant plus au domaine du génie du logiciel.

I.2.2.- Besoins de base

a) Contrôle du procédé

Expression, dès la spécification, du contrôle direct du procédé nécessitant la prise en compte de son évolution. Ce contrôle est déduit du cahier des charges qui en principe contient une description des interactions entre le procédé et l'application qui le contrôle.

b) Expression du parallélisme naturel (ou de situation)

Un procédé industriel est, par excellence, le siège de beaucoup d'activités parallèles. Il est bon de pouvoir exprimer, dès la spécification ce parallélisme de situation lié à la nature même du problème posé. L'expression de ce parallélisme complète l'expression du contrôle direct de l'ACPI sur le procédé.

c) Expression des contraintes de temps

A ce niveau nous retenons deux aspects :

- l'aspect expression des contraintes de temps elles-mêmes sous forme de "time-out" par exemple et la spécification des actions à entreprendre en cas de non respect de ces contraintes.

- l'aspect moyens ou encore outils mis à la disposition du spécifieur afin qu'il puisse modeler ou remodeler son application de façon à respecter les contraintes de temps prévues dans le cahier des charges. Là, on retrouve l'aspect répartition d'une ACPI ou plus généralement l'expression de ce que l'on appelle le parallélisme de conception, parallélisme qui n'est pas lié à la nature propre du problème mais qui est introduit pour répondre à des problèmes de conception. C'est là que nous plaçons le découpage de l'application en entités communicantes, découpage, qu'il est important de pouvoir remettre en cause sans toucher aux spécifications des contrôles sur le procédé et à l'expression du parallélisme de situation (qui, eux, font partie de la nature du problème posé).

d) Expression de la sûreté de fonctionnement

Bien connu des concepteurs d'ACPI, l'aspect sûreté pendant le fonctionnement est particulièrement important dans la conduite de procédé industriel.

Là encore, un certain nombre de sécurités sont précisées déjà au niveau du cahier des charges et nous pensons qu'il est important de pouvoir les exprimer dès le stade de la spécification.

Dans ce problème de sûreté dans le fonctionnement, il est important de pouvoir d'une part détecter des anomalies dans le fonctionnement et d'autre part se donner les outils nécessaires pour éventuellement pouvoir y remédier.

- un moyen de détection peut être constitué par la définition de ce que l'on appelle des délais de garde (time-out), déduits à partir du cahier des charges.
- lié à cela, on doit pouvoir définir des traitements urgents à entreprendre en cas d'anomalies, toute autre activité cessante. Ce qui aboutit naturellement à l'expression de la préemption dès le stade de la spécification et plus généralement à un contrôle direct de l'activité des tâches. La préemption conduit à l'expression des priorités relatives de ces traitements et suppose que les suspensions de traitements sont possibles sur la base de ces priorités.

e) E/S Spécialisées

Les ACPI sont caractérisées par le nombre et la variété des E/S dites industrielles, et qu'il faut prendre en compte.

Quoique généralement précisées dans le cahier des charges, nous pensons que l'on peut faire abstraction de cet aspect au cours d'une spécification en considérant les traitements associés à ces entrées/sorties sans s'occuper du détail des procédures d'E/S...

I.2.3.- Besoins en structuration

Ces dernières années ont vu se développer une discipline nouvelle intitulée "génie logiciel". Née d'un constat des difficultés de transportabilité, lisibilité, maintenance ... des logiciels, cette discipline vise principalement la structuration des logiciels pour tenter d'éliminer ces inconvénients.

Il est évident que ces problèmes se posent avec acuité dans le domaine des ACPI. A notre sens, la structuration doit viser, dans ce domaine :

- la facilité de maintenance
- l'évolutivité
- la transportabilité et la réutilisation
- et enfin la fiabilité du logiciel en facilitant notamment les preuves de bonne marche (localisation des points de synchronisation....).

En effet, un procédé est rarement figé : les machines changent, certaines sont ajoutées. De plus, même lorsque le procédé ne change pas, de nouvelles fonctions peuvent être introduites, et enfin, il faut tenir compte du développement du procédé qui se fait presque

toujours progressivement. Lorsqu'on parle de structuration, on pense évidemment à une méthode de structuration. Plusieurs approches basées sur différents critères existent. Sans vouloir rentrer dans les détails d'une méthode de structuration, notons qu'il est difficile d'avoir une méthode précise de découpage du logiciel ; on peut toutefois, toujours, se donner un "guide" de découpage qui est celui de la fiabilité du logiciel avec un effet bénéfique certain sur la sûreté de fonctionnement, aspect important des ACPI comme souligné plus haut.

I.3.- SYSTEMES OPERATOIRES ET ACPI : QUELQUES COMPARAISONS

En énumérant les besoins des ACPI, on peut se rendre compte qu'ils ne sont pas très éloignés des besoins de la programmation des systèmes opératoires : expression du parallélisme, problème de préemption, E/S et enfin structuration. Pour les mêmes besoins, il est normal de retrouver les mêmes concepts à exprimer et il est naturel que les développements de l'un et l'autre des deux domaines se soient faits de manière plus ou moins coordonnées. Il nous a paru intéressant de faire un rapide tour d'horizon de ces développements afin de mieux mettre en évidence l'état de l'art et justifier nos préoccupations actuelles.

A la base des besoins communs nous retrouvons la gestion d'un grand nombre d'activités parallèles que l'on appelle tâches ou processus au sens [CROC 75]. Ces développements se sont faits de la façon suivante :

a) Au départ, les systèmes opératoires aussi bien que les ACPI étaient programmés en assembleur. Le sentiment répandu était qu'il fallait être "très près" de la machine pour mieux contrôler les activités qui s'y déroulent. Cela était dû, à notre sens, à une définition imprécise des véritables besoins de ce type de programmation et aboutissait à un manque de lisibilité, transportabilité ... évidents des applications.

b) Utilisation des langages dits évolués, conséquence d'une meilleure définition des besoins en ce qui concerne notamment la gestion du parallélisme. Si l'introduction de primitives évoluées telles que P et V [DIJK 65] paraît plus en rapport avec le développement des systèmes opératoires (intégration de ces primitives dans le langage universel Algol 68 ...), il n'en demeure pas moins que dans le même temps, le concept de base à exprimer et qui est celui de la gestion des tâches, se retrouve dans nombre de langages dit temps réel qui sont :

- soit des extensions de langages existants comme FORTRAN temps réel [SEMS 1], BASIC-TR [BIAN & al 76].
- soit encore de nouveaux langages intégrant ces concepts : CORAL 66 [WEBB 75], PEARL [ELZ 75], RTL/2 [BARN 75], PRØCØL [RIT & al 75], LTR [PAR & al 75], HAL/S [FLYS 75] et autres langages dont on peut trouver une bonne étude dans [GER & al 75].

Notons que PEARL et PRØCØL utilisent tous deux la notion de sémaphores avec les primitives P et V.

c) Les concepts de bases étant bien définis, et une certaine expérience étant acquise sur ces bases aussi bien dans l'écriture des ACPI que des systèmes opératoires, très vite se sont posés les problèmes d'évolutivité, maintenabilité, facilités de preuves aboutissant à une plus grande fiabilité du logiciel ... et par là-même les problèmes de structuration des logiciels.

On peut alors distinguer deux phases dans l'évolution des problèmes de structuration des logiciels :

Phase 1

On s'était rendu compte qu'une entrave majeure à une bonne structuration des logiciels était liée à la dispersion des primitives de synchronisation dans le texte des programmes. Ceci, bien

évidemment empêchait toute vue globale de l'application et rendait difficile toute modification.

Sur la base de ces remarques, un certain nombre de recherches visant la séparation de ce que l'on appelle la partie contrôle (représentant l'expression des synchronisations) de la partie opérative (représentant l'expression des calculs et actions à entreprendre) ont été entreprises. Parmi ces travaux et sans vouloir être exhaustif, on peut citer : les modules de contrôle de Robert et Verjus [RVER 77], les expressions de chemin de Habermann [CAMP 74], les mots de synchronisation de ROUCAIROL [ROU 78], les moniteurs de Hoare [HOA 74], les commandes gardées [DIJK 75].

Plus spécifiquement, dans le domaine du temps réel où l'idée de séparation des parties contrôle et opérative est dirons-nous "naturellement" admise, notamment par les automaticiens comme l'ont soulignés H.G. MENDELBAUM [MEND 77], F. MADAULE [MAD 77] et C. SOURISSE [SOU 79], se sont développés quasiment en même temps et basés sur les mêmes concepts un certain nombre de travaux : [MEND 76], [LAD 77], [LAD 82], [LECAL 79], [THOM 80] ou encore plus spécifiquement sur le concept de commandes gardées : [GRIE 76], [HAHA 77], [HADE 79] ...

La liste des références bibliographiques, qui n'est certainement pas exhaustive, tend à mettre en évidence la "proximité" des deux domaines étudiés, au moins en ce qui concerne les concepts à exprimer.

Soulignons que cette structuration, en même temps qu'elle résoud des problèmes d'évolutivité, maintenance.., permettait de décomposer la conception d'un logiciel en deux sous-problèmes distincts :

- conception et écriture de toutes les activités purement séquentielles.
- mise en place des synchronisations.

Du coup, ceci permettait une vue globale d'une application avec naturellement :

- une tendance à suivre une démarche de conception descendante en allant de la vue globale vers les actions élémentaires.
- l'introduction du concept de modularité permettant une séparation (et réutilisation possible) des actions élémentaires.

Phase 2

Prise en compte de l'aspect "communications".

Depuis une decennie environ, le développement des réseaux de transmission de données ont poussé les concepteurs à adopter des solutions réparties quand la nature du problème le permettait.

Ce caractère de répartition a évidemment apporté une dimension nouvelle au problème de la conception de gros logiciels : celle de la communication.

Si au départ synchronisation et communication étaient étroitement liées, les problèmes de communication se ramenant souvent à l'utilisation de mémoires communes avec la synchronisation que cela impose, la communication, avec le développement des réseaux, s'est peu à peu dégagée comme concept de base. Elle faisait, certes, appel à la synchronisation en dernier recours mais la synchronisation était considérée comme ne faisant pas partie de la nature du problème posé.

Un certain nombre de recherches se sont faites dans le domaine, recherches ayant abouties à la définition de langages de programmation supportant comme concept de base, celui de communication. Parmi les plus importants, on peut citer CSP [HOA 78], DP [BRIN 78], PLITS [FELD 79] ou encore, plus près du domaine industriel, les travaux de Kramer [KRAM & al 82] ou du CRIN (Nancy) ([ZAK 84]...).

Pour tous ces langages, le problème est pensé et décrit en termes d'entités communicantes à travers des ports [KRAM & al 82], [ZAK 84], des points d'entrée [HOA 78], des appels de procédures [BRIN 78].

Encore une fois, il est intéressant de noter que toutes ces recherches, non spécialement faites pour le temps réel, ont vite trouvé application dans le domaine temps réel (*).

Deux exemples frappants de ces "applications" nous sont donnés par les langages ADA [ICH & al 79] et CHILL [BRAN & al 83].

Si CHILL est plus orienté "problèmes de téléphonie" ADA par contre s'est voulu plus "universel". Dans la nouvelle version de LTR, LTR.V3 [PAR 80], il est également tenu compte des nouveaux concepts apportés par ces recherches.

La caractéristique de ces langages est de permettre un découpage et donc une structuration, relativement aisé d'une solution en entités communicantes.

Généralement doublé d'un mécanisme d'encapsulation tel que développé dans les langages supportant les types abstraits ([LIS & al 81], [MIN 79], ADA [ICH & al 79], LTR.V3 [PAR 80]...),

ces langages permettent un découpage de la solution en entités indépendantes uniquement sur la base de la communication. Ces entités prennent le nom de modules ([KRAM & al 82]....) ou encore de "task" dans ADA.

Cette façon de voir le problème est évidemment très séduisante et on aboutit généralement à un découpage "élégant" des solutions. La question que l'on peut se poser, cependant, concerne l'adéquation de cette vue, son aptitude en quelque sorte à traiter tous les types de problèmes rencontrés, en particulier dans le domaine du temps réel.

C'est cette question que nous allons essayer d'examiner dans le paragraphe suivant.

(*) Ceci s'explique par l'aspect naturellement réparti des procédés industriels, aspect qui a poussé à une utilisation de réseaux locaux dans ce domaine.

I.4.- APPLICATIONS REPARTIES ET ACPI : COMMUNICATION ET EVENEMENT

De l'analyse précédente, il ressort clairement que dans le domaine de la spécification et programmation des ACPI, une voie qui semble se dégager actuellement est celle basée sur les entités communicantes. Cela est dû, en particulier, à la nature de plus en plus distribuée de ce type d'applications (utilisation de réseaux locaux). Un certain nombre de travaux se sont développés dans ce sens parmi lesquels on peut citer [BRIN 78], [MAO & al 79], [KRAM & al 82] ou encore au CRIN (Nancy), [DER & al 83(1)], [DER & al 83(2)], [ZAK 84], [GØF 84].

La remarque qui s'impose alors est que, dans un tel contexte, une application est considérée avant tout dans son aspect distribuée.

Si d'un point de vue structuration d'une application, cette démarche présente un intérêt certain, il nous semble néanmoins légitime de nous poser la question suivante : comment sont pris en compte les besoins spécifiques du temps réel dans de telles approches ? Les éléments de réponse à cette question nous permettront de savoir si effectivement ces approches sont totalement adaptées au temps réel ou de proposer, le cas échéant, une solution.

Si l'on pose cette question, c'est en effet pour mettre en évidence le fait qu'il nous semble qu'une démarche privilégiant l'aspect distribué comme base pour la structuration peut conduire à négliger l'aspect temps réel.

Il est clair que dans un langage se voulant temps réel, il est nécessaire de tenir compte de l'aspect distribué à travers la définition des communications notamment, mais il est clair également que les aspects temps réel doivent y être bien intégrés.

Concernant ces aspects temps réel, ils se rapportent, pour nous, aux besoins de base exprimés dans le § I.2.2.

Rappelons que les points les plus importants concernent :

a) le contrôle du procédé.

b) la sûreté de fonctionnement (qui est liée en fait au point a

c) le respect des contraintes de temps imposées par la cahier des charges.

L'on comprend alors aisément qu'il soit nécessaire de disposer d'outils permettant d'avoir un contrôle fin et explicite sur le déroulement des activités d'une application (c'est habituellement le cas comme l'ont par exemple souligné [PIKE 72] et [MAH 81]), dans un langage se voulant temps réel.

Ce contrôle des activités doit être lié à l'évolution du procédé mais également à des expressions relatives au temps (à 18 h faire Tâche 1, par exemple) et c'est là un aspect typiquement temps réel.

Qu'en est-il alors dans les langages privilégiant l'aspect distribué et quel est notre point de vue ?

I.4.1.- Les langages basés sur la communication

Une des caractéristiques des langages basés sur la communication, qu'ils soient orientés vers le temps réel (DP [BRIN 78]...) ou non (PLITS [FELD 79]) consiste en l'utilisation du concept de communication à deux fins :

a) - pour l'expression de la nature distribuée d'une application en spécifiant les échanges entre ses constituants.

b) - comme outil de structuration permettant une plus grande modularité (et donc indépendance des entités) d'une application.

Cette modularité assurée uniquement par le biais de la communication suppose donc que la coopération entre les différentes entités composant une application est assurée par le biais de messages. C'est ce qui assure l'indépendance des entités mais aussi s'oppose à la notion de contrôle explicite du déroulement des activités. Le contrôle est en effet, dans ce cas, assuré par des messages et tend en fait à devenir un auto-contrôle : une entité reçoit un message d'une certaine nature et peut décider ou non de réaliser une fonction donnée en fonction du message reçu.

Des exceptions à cette règle existent, notamment lors de l'arrêt d'urgence d'une entité (ABORT de ADA).

On retrouve alors, traités sur un même niveau d'égalité, des messages qui n'ont pas forcèment le même caractère d'urgence : un message signalant une alarme grave et un message de demande de compte rendu par exemple.

De plus, tous les messages subissent les synchronisations inhérentes au système de communication, pouvant subir des attentes qui ne sont pas toujours souhaitables (rendez-vous de ADA par exemple).

La viabilité d'une application, au moins en ce qui concerne la prise en compte rapide de certains messages, repose dans ces conditions sur la disponibilité des entités. Cette disponibilité peut être admise par hypothèse : c'est le cas de DP [BRIN 78] ou de CP [MAØ & al 79] où à chaque entité est associé un processeur d'un réseau de micro-ordinateurs avec pratique de l'attente active. Elle peut également être assurée par un mécanisme fourni dans le langage : c'est le cas dans [DER & al 83(2)] où la disponibilité est assurée par l'expression des règles de préemption au niveau de la communication.

Dans tous les cas cependant, la décision revient toujours à l'entité destinatrice et il n'y a de ce fait aucune garantie de réalisation de la fonction souhaitée.

Cela est particulièrement gênant dans le domaine du temps réel, où la garantie du contrôle du procédé dans des délais prévus est précisément fonction de la garantie d'exécution de certaines entités d'une application dans les délais prévus,et peut nuire à la fiabilité

du logiciel produit.

Il en est de même pour les problèmes de sûreté de fonctionnement pour lesquels il est nécessaire de garantir les arrêts, suspensions ... d'entités, dans des délais acceptables.

On se rend compte alors que l'utilisation de la communication en tant que concept de base pour la structuration encourage une expression implicite des contrôles sur le déroulement des activités d'une application.

Si ce contrôle implicite va dans le sens d'une plus grande modularité, il nous semble néanmoins, comme on l'a vu dans ce paragraphe, qu'il s'oppose à une bonne expression de certaines concepts temps réel.

Il est évident que l'expression explicite des contrôles dans le corps des entités et l'indépendance des entités sont deux objectifs contradictoires.

On peut alors envisager une solution qui consiste à exprimer explicitement ces contrôles en dehors du corps des entités comme dans [THOM 80]. La structuration ne se ferait plus uniquement sur la base de la communication mais intègrerait l'aspect contrôle.

Il faudrait dans ce cas, également examiner ce que deviennent les messages qui étaient liés à ce contrôle.

Quelle est alors la nature de ces messages ?

On peut classer ces messages, que l'on peut qualifier de critiques pour une application, en deux catégories :

a) - ceux qui comportent un ordre précis : arrêt par exemple, et qui vont s'intégrer au contrôle.
b) - ceux qui représentent l'évolution du procédé et qui sont donc liés à changements d'états importants intervenus dans le procédé : une alarme par exemple.

Concernant les messages de la catégorie b), on remarque que ce n'est pas la valeur du message qui est exploitée mais beaucoup

plus le fait qu'ils représentent un changement d'état.

Une entité donnée ne n'intéressera pas à la valeur d'un tel message mais au fait qu'il soit arrivé ou non ; un tel message comporte une sémantique implicite.

Ces messages constituent donc un type particulier de messages qui sont proches de ce que l'on appelle habituellement les événements. Il nous semble alors souhaitable de les traiter en tant qu'objets particuliers et à part entière d'un langage orienté vers le temps réel car ils représentent une caractéristique importante d'une application temps réel : l'évolution du procédé .

En effet, si au cours de la conception d'une application temps réel répartie, certains aspects sont facilement et naturellement appréhendés en termes de communications, il nous semble également que d'autres aspects (évolution du procédé, évolution du temps) qui sont plus "temps réel" peuvent être naturellement appréhendés en termes d'événements.

Concernant l'expression du temps en particulier (le temps représente un aspect important dans le domaine du temps réel), il est difficile, ou en tous cas pas naturel de l'exprimer sous forme de messages.

Evénements et communications semblent alors tous deux indispensables et surtout complémentaires dans tout langage se voulant temps réel.

En tenant compte de cette analyse, on peut donc envisager une solution permettant d'exprimer explicitement et indépendam-

ment du système de communication tous les contrôles que l'on souhaite avoir sur le déroulement des activités en fonction de l'évolution du procédé ou de l'évolution du temps.

Les événements seraient traités non plus comme messages quelconques mais comme objets à part entière servant à exprimer l'évolution du procédé, du temps ou des entités de l'application et permettant donc d'exprimer les coordinations nécessaires entre le procédé (au sens large) et l'application.

De cette façon, nous pensons pouvoir tenir compte dans un langage orienté vers le temps réel, à la fois de l'aspect distribué et de l'aspect temps réel.

De plus, comme nous le verrons par la suite, cette démarche permet de tendre vers une spécification d'application.

I.4.2.- Notre système basé sur les événements

Plutôt que de considérer tous les types de messages sur un même niveau, notre démarche consiste à proposer l'introduction de ce que

l'on peut appeler un niveau de contrôle, basé sur le concept d'événement et qui nous permet d'exprimer explicitement tous les contrôles de déroulement des activités souhaités. La communication sera utilisée pour exprimer la coopération entre les différentes entités composant l'application en tenant compte du contexte réparti.

De cette façon, adoptant une analyse différente, on rejoint les points de vue exposés dans [THOM 80] et qui sont résumés dans l'annexe I.

Il faudra néanmoins étudier plus à fond ce concept d'événement et l'aspect expression du contrôle.

Cette démarche permet de privilégier l'aspect temps réel d'une application mais également de montrer la complémentarité des concepts d'événements et de communications.

Quels sont les avantages apportés par une telle démarche ?

a) Séparation d'une application en partie contrôle et partie opérative comme préconisée dans [TR 77].
   Ceci permet d'une part un rapprochement du mode de pensée des automaticiens et d'autre part de prendre en compte le problème important de l'évolutivité d'une application.

b) Le concept d'événement permet d'exprimer clairement les synchronisations entre le procédé et l'application.

c) La partie contrôle permet d'exprimer les contrôles nécessaires sur le déroulement des activités en fonction de l'évolution du procédé représentée en termes d'événements, indépendamment des activités à contrôler.
   Elle peut donc être vue comme une expression statique du comportement d'une application en fonction de l'évolution du procédé. Cette évolution étant elle-même déduite du cahier des charges, on s'aperçoit que de cette façon on se rapproche véritablement de la spécification d'une application.

Les outils de base de cette spécification sont alors :

- les expressions de contrôle
- les événements

et la spécification est de type déclaratif.

d) Le concept d'événement nous semble être un concept relativement naturel dans une phase de spécification d'une application temps réel dans ce sens que notre perception d'un problème temps réel ne se fait pas forcément en termes de messages échangés ; en effet, à la lecture d'un cahier des charges, lorsqu'on rencontre une phrase du type "tel traitement doit être effectué lorsque telle situation survient", le problème est plutôt perçu en termes d'événements.

e) D'un point de vue sûreté de fonctionnement, le concept d'événement autorise un recensement exhaustif des cas à traiter, ce qui ne peut qu'accroître la fiabilité du logiciel. De plus fonctionnement normal et fonctionnement anormal peuvent être pris en compte de la même façon dès la spécification.

f) D'un point de vue programmation, le concept d'événement est un concept qui apparaît dans nombres de langages temps réel, ce qui tend à rapprocher d'une certaine façon la spécification d'une application de sa programmation.

g) Le fait de considérer les événements comme objets à part entière nous permet d'introduire une plus grande puissance d'expression des synchronisations (liées à l'évolution du procédé) que les messages.

Remarque

Notre spécification va consister en l'expression du comportement des composants de l'application en fonction de l'évolution du procédé. Ceci rejoint le point de vue exprimé dans [ZAK 84] qui consiste à décrire pour chaque entité, son comportement vis-à-vis de tous les messages reçus. La différence réside dans le fait que nous nous intéressons seulement au comportement des entités vis-à-vis de l'évolution du procédé et du temps.

I.5.- CONCLUSION

L'étude de l'évolution de la programmation des ACPI nous aura permis, à travers quelques comparaisons avec les systèmes opératoires et les applications réparties d'une manière plus générale, de mettre en évidence la nécessité d'un contrôle explicite des entités d'une application dans un contexte temps réel.

Ce contrôle devant se faire en fonction de l'évolution du procédé, nous avons mis en évidence également la nécessité de disposer d'un type d'objets adéquat pour rendre compte de cette évolution : les événements. Ce faisant, nous nous sommes rendus compte que les événements, associés à une expression statique du contrôle permettaient de tendre vers un outil de spécification, mais aussi de programmation d'applications en temps réel.

Le support de cet outil de spécification sera un langage de manipulation d'événement dont la syntaxe est proche de celle proposée dans [THOM 80] en mettant l'accent sur :

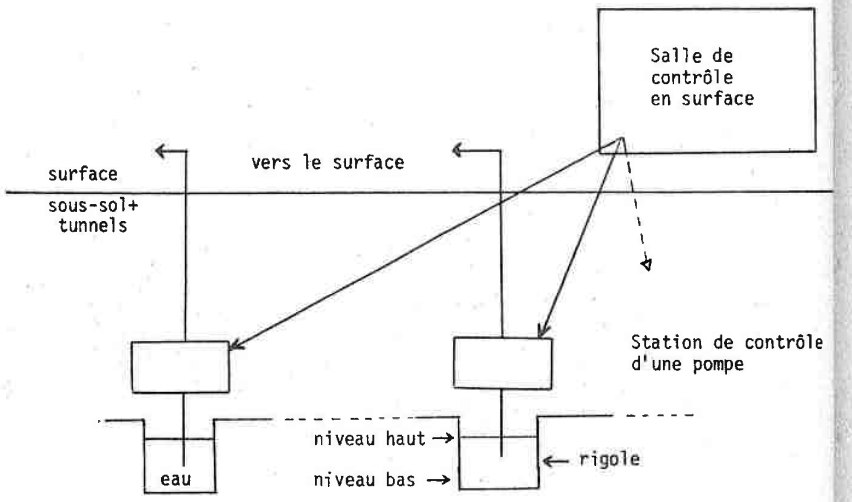
- une étude approfondie du concept d'événement,
- une étude sur l'expression du comportement des entités,

pour tendre plus vers l'aspect spécification.

La terminologie de [THOM 80] (voir également l'annexe I) est conservée.

I.6.- ENONCE DES EXEMPLES

I.6.1.- Pompe de KRAMER dans les mines de charbon



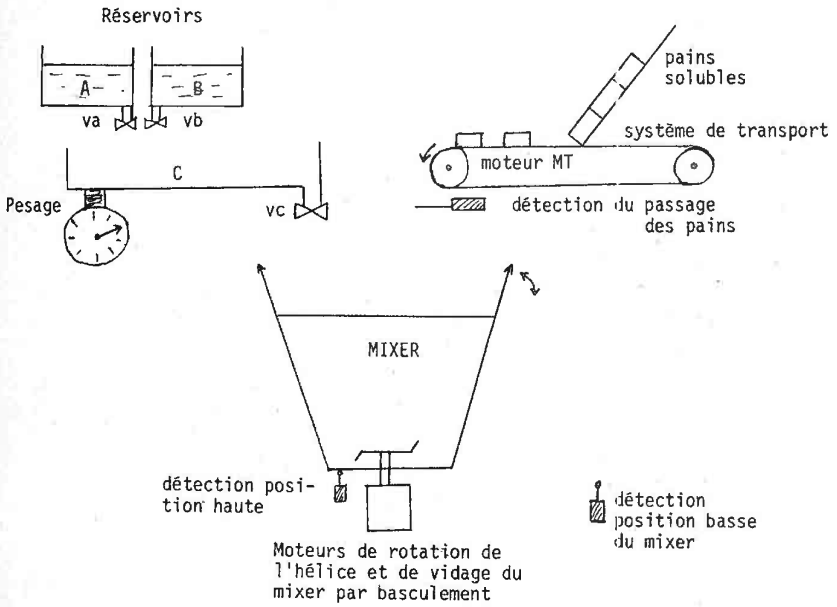
Le problème est le suivant :

Les pompes doivent fonctionner automatiquement, contrôlées par le niveau d'eau dans les rigoles, lorsqu'elles en reçoivent l'ordre de la salle de contrôle.

La détection du niveau haut d'eau dans une rigole, provoque le fonctionnement de la pompe associée jusqu'à la détection d'un niveau bas.

Pour des raisons de sécurité, les pompes ne doivent être mise en marche ou continuer de fonctionner quand le méthane atteint un certain niveau dans les tunnels.

I.6.2.- Exemple-test de l'EWICS-TC 11



Il s'agit de la fabrication d'un produit déterminé P constitué par une quantité déterminée d'un mélange des produits A et B (la pesée dans C détermine cette quantité) et un nombre déterminé de pains solubles (détectés par comptage), mixé pendant un temps déterminé Δt ;

Au bout de ce temps, le mixer est vidé, puis un autre cycle de fabrication recommence.

Les quantités de matières A, B et pains sont supposées infinies.

CHAPITRE II :

LES ÉVÉNEMENTS : DÉFINITIONS GÉNÉRALES - ÉTUDE BIBLIOGRAPHIQUE COMPARATIVE - NOS PROPOSITIONS

II.1.- GENERALITES

La conception d'une application de contrôle et commande d'un système physique tel qu'un procédé industriel passe par une étape nécessaire qui est celle de l'observation du procédé. Le contrôle suppose, en effet, une bonne connaissance de l'évolution du procédé à contrôler. Cette évolution se traduit généralement par des phénomènes physiques observables et quantifiables.

Parmi les phénomènes physique pouvant se produire dans un procédé, seul un sous-ensemble représentatif de l'idée que l'on se fait du procédé par rapport au contrôle visé est l'objet d'observations : C'est ce qui constitue l'environnement de l'ACPI et que nous qualifierons de niveau externe. Le niveau interne, quant à lui, est constitué par l'ACPI elle-même. Ce niveau peut également être l'objet d'observations de phénomènes internes tels que les fin de tâches, début de tâches...

Afin de préciser un peu plus le vocabulaire, plutôt que de parler de phénomènes observables, nous parlerons de grandeurs observables, ce qui donne une idée de quantifications et de mesures. Ces grandeurs correspondent à ce que les automaticiens appellent les variables du procédé. Elles peuvent prendre différentes valeurs et il est intéressant de noter que l'on peut s'intéresser :

- soit aux valeurs de ces grandeurs. Dans ce cas on s'intéresse aux états de ces grandeurs.

- soit aux changements d'états de ces grandeurs.

Dans le contrôle d'un procédé industriel, il est important de prendre en compte les deux aspects : une même grandeur peut aussi bien servir à calculer une consigne à appliquer dans un algorithme de type PID, par exemple, qu'être à l'origine d'une alarme lorsqu'elle dépasse un seuil de sécurité donné.

Lorsque l'on parle de changement d'état d'une grandeur, il est

bien évident qu'il ne s'agit pas de n'importe quel changement d'état : Si la grandeur ne peut prendre que deux valeurs booléennes F et V, ou bien on s'intéresse à la transition F → V, ou bien on s'intéresse à la transition V → F, ou bien enfin aux deux transitions à la fois sachant qu'il s'agit de deux phénomènes distincts.

L'observation correcte d'un procédé industriel suppose alors :

- que l'on se fixe le sous-ensemble de grandeurs à observer, significatives pour le problème traité.
- que l'on précise si on s'intéresse aux états ou aux changements d'états,
- que l'on précise à quelles valeurs aux changements d'états on s'intéresse.

Si le fait de s'intéresser à des états de grandeurs données n'est pas fondamentalement un problème lié au contrôle de procédés industriels, les changements d'état en revanche traduisent plus l'évolution d'un procédé et définissent des instants caractéristiques dans la vie d'un procédé. Par rapport à ces instants caractéristiques on peut prendre certaines décisions et avoir une réaction en temps réel (supposant donc l'existence d'un délai maximum entre l'arrivée d'une occurrence et la réaction de l'ACPI).

Dans les objectifs que l'on s'est fixé, c'est cet aspect "changement d'état" qui nous intéresse en premier lieu et qui va faire l'objet de plus de développements dans la suite.

Mais on s'aperçoit déjà que, parler de changements d'état suppose un moyen de détection. Parler de détection suppose que l'on connaît la source de ces changements d'état, donc leur production. Une fois ces changements d'état détectés, il faut également se poser le problème de ce que l'on veut en faire : il s'agira de préciser leur utilisation. Ceci nous donne les grands points de la suite de ce chapitre en commençant par des définitions.

II.2.- DEFINITIONS

II.2.1.- Notion d'occurrence d'événement

On appelle occurrence tout changement d'état attendu d'une grandeur observée.

II.2.2.- Notion d'événement

Un événement est la suite, ordonnée dans le temps, des mêmes changements d'état d'une grandeur observée.

C'est donc une suite ordonnée d'occurrences.

Notons que dans ces définitions, il n'a pas été fait mention d'un aspect logique ou physique. Elles sous-entendent de plus que la notion d'occurrence suppose une détection (terme "attendu").

Le changement d'état étant attendu, donc parfaitement défini et connu, à l'occurrence sera attaché l'instant du changement d'état.

L'instant du changement d'état est l'instant de détection.

II.3.- SOURCES DES OCCURRENCES : PRODUCTION

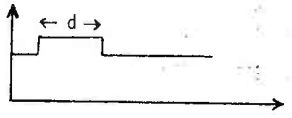
II.3.1.- Sources externes [NANT 70]

II.3.1.1.- Signaux à états

a) Signaux stables

Ce qui caractérise ces signaux, c'est leur stabilité relative par rapport à la vitesse de traitement d'un calculateur. Cette stabilité, définie par une constante de temps élevée par rapport à la vitesse de traitement leur confère le statut de statiques.

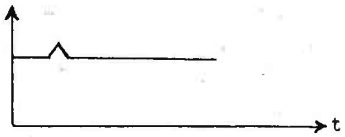
Schématiquement, on peut représenter un tel signal par



d étant non nul

b) Signaux impulsionnels

Contrairement aux signaux stables, cette catégorie de signaux est caractérisée par une durée très brève et voisine de zéro. Leur constante de temps est très faible par rapport à la vitesse de traitement d'un calculateur . Comme exemple de ces signaux on peut donner les interruptions. Schématiquement, on a la représentation :



c) Notion de mémorisation physique

Très souvent, la faiblesse de la durée d'un signal impulsionnel exige un dispositif de mémorisation physique. Cette mémorisation confère au signal une certaine stabilité et permettra de différer légèrement sa prise en compte. C'est ce qui se passe avec les interruptions.

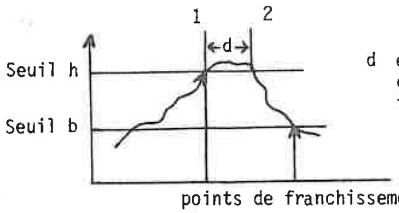
L'importance relative, par contre, de la durée d'un signal qualifié de stable rend inutile toute mémorisation physique autre que celle propre au signal lui-même.

Il faut noter à ce niveau que tout signal comporte de toutes les façons une mémorisation (bascule dans un certain état pendant un certain temps, bascule dans un état jusqu'à effacement...).

II.3.1.2.- Signaux continus ou analogiques

Dans cette catégorie, rentrent tous les signaux à variation continue telle que les tensions électriques.

Si pour les signaux à état le changement d'état est parfaitement défini de par la nature discontinue du signal, l'observation dans le cas de signaux continus porte plus sur le franchissement de certains seuils définis, ce qui en définitive revient au même.



d est définie par les deux points de franchissement 1 et 2

Dans tous les cas, finalement on s'intéresse aux instants de passage d'une grandeur par des valeurs particulières.

Ces signaux sont considérés comme stables dans le mesure où ce temps de passage par une valeur particulière est suffisamment élevé par rapport à la vitesse de traitement pour permettre leur détection.

II.3.2.- Sources internes

On regroupe sous cette rubrique toutes les causes d'occurrences purement internes au système de contrôle. La source d'une occurrence peut être programmée donc explicite ou alors implicitement définie. Dans les deux cas cependant, ces sources sont considérées comme stables dans la mesure où leur production est entièrement contrôlée de manière interne. Ceci n'était pas le cas des sources externes qui dépendaient uniquement du procédé industriel. On retrouve ici la notion d'événement logique tel qu'elle est définie dans les systèmes classiques (SIRIS 8 ...).

II.3.2.1.- Source programmée

Dans ce cas, l'occurrence est produite par l'exécution d'une

primitive du type POST (evt) ou SIGNAL (evt). L'occurrence est considérée comme produite dès l'exécution de la primitive. L'instant de production est alors aussi l'instant de détection.

II.3.2.2.- Source implicite

Il n'y a, dans ce cas, pas de primitive particulière de production d'occurrences. Des instants particuliers dans la "vie" des composants du système de contrôle sont interprétés en tant qu'occurrences d'évènements. Si, par exemple, dans l'ACPI, on dispose de la notion de tâche, on peut s'intéresser aux occurrences "fin de tâche", "début de tâche"...

Remarquons que ces deux cas s'appuient sur les mêmes principes et qu'on peut passer sans difficulté d'une forme d'expression à l'autre.

La différence réside dans la position des points de production d'occurrences ou encore points de synchronisation : sans position précise dans le premier cas mais exprimés explicitement, en des points précis mais non exprimés explicitement dans le deuxième cas.

Cette forme de signalisation interne est utilisée pour la synchronisation des composants de l'ACPI. Elle n'est pas fondamentalement différente d'une signalisation externe.

Précisons pour terminer qu'une expression implicite permet :

- une localisation plus aisée des points de synchronisation,
- une expression de la synchronisation externe aux composants du système de contrôle.

II.4.- DETECTION DES OCCURRENCES

II.4.1.- Notion d'observabilité : durée de vie intrinsèque attachée à une occurrence

Il paraît, en premier lieu, paradoxal de parler d'une durée de vie d'une occurrence alors que celle-ci est définie en terme d'instant précis.

Pourtant cet instant est le fait d'une détection qui est réalisée à partir de la source de l'occurrence. Or cette source, comme déjà vu plus haut, peut prendre différentes formes et peut être en particulier plus ou moins brève : elle a donc une durée. On constate facilement que la détection éventuelle d'une occurrence d'événement ne peut se faire que pendant ce laps de temps, la détection en elle-même se faisant toujours par comparaison entre un état précédent et un état courant.

Définition

On appelle durée de vie intrinsèque attachée à une occurrence, l'intervalle de temps pendant lequel, il est possible de détecter l'occurrence. Cet intervalle de temps dépend étroitement de la source et un examen cas par cas est donc nécessaire.

II.4.2.- Types de sources et durée de vie intrinsèque

II.4.2.1.- Occurrences à source externe

a) Signaux à état

- Signaux impulsionnels

La durée de tels signaux est considérée comme proche de la valeur 0. S'il est difficile de fixer une durée précise définissant un signal impulsionnel, on peut par contre en parler en terme de nécessité de mémorisation physique sous peine de perte du signal, donc de l'occurrence. La non mémorisation de tels signaux oblige quasiment à adopter une détection avec attente active, c'est-à-dire à ne faire que de la détection.

Ceci est le cas des interruptions sur les calculateurs où la

brièveté du signal oblige à une mémorisation physique à l'aide d'une bascule.

Comment définir alors la durée de vie intrinsèque attachée à une occurrence dans ce cas ?

Vis-à-vis d'un observateur direct du signal, cette durée de vie serait simplement la durée du signal, c'est-à-dire proche de 0.

Notre "observateur" quant à nous, se situe après la mémorisation physique bien évidemment.

Le signal étant mémorisé, on peut considérer que la durée de vie attachée à l'occurrence est infinie. Elle est, dans la réalité cependant, limitée par le cycle du détecteur : cycle de scrutation des voies d'interruption d'un calculateur, par exemple, avec aquittement des niveaux d'interruption.

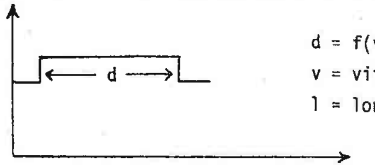
- Signaux stables

Dans ce cas, la durée des signaux est telle qu'une détection est possible sans autre mémorisation physique que celle propre à la source.

L'observation se fait donc directement sur le signal et la durée de vie intrinsèque attachée à une occurrence est en fait égale à la durée du signal.

Cette durée, pour un même phénomène peut être fixe ou variable.

Exemple : Pour la détection d'une pièce passant sur une bande transporteuse d'une chaîne d'usinage on peut avoir le schéma :

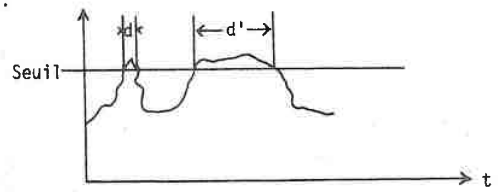


d = f(v,l)
v = vitesse de transport
l = longueur de la pièce

les E/S dites "Tout-ou-Rien" correspondent à ce type de signaux

b) Signaux continus

L'observation pour ces signaux se fait par rapport à un état donné (dépassement de seuil...). Prenons un exemple pour voir ce que l'on appelle durée de vie attachée à une occurrence dans ce cas :



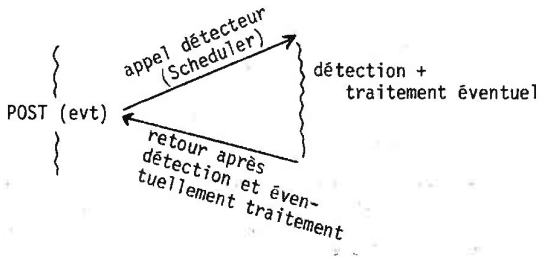
d et d' sont des intervalles de temps pendant lesquels il est possible de détecter une occurrence.
d a été volontairement choisie petite sans pour autant que le signal soit considéré comme impulsionnel.

Ce type de signaux est en fait considéré comme étant, par nature, suffisamment stable pour permettre une observation directe.

Le durée de vie dans ce cas est définie par la durée pendant laquelle le signal est dans un état donné.

II.4.2.2.- Occurrences à source interne

La production des occurrences est dans ce cas entièrement interne et se fait par appel direct au Scheduler qui joue le rôle de détecteur d'occurrences : c'est en effet exactement cette interprétation qui est donnée à une primitive du type POST (evt) des systèmes SIRIS 7/8 de la CII ou encore du SEVENT [SEMS II].



L'instant de production est l'instant d'appel.

Le retour ne se fait qu'après détection et éventuellement traitement de l'occurrence. Ceci revient exactement à une mémorisation physique jusqu'à la détection de l'occurrence. De ce fait, ces "signaux" ne peuvent être que stables. La primitive POST(evt) n'est pas "perdue" et attend son traitement. La durée de vie est considérée alors comme étant infinie.

Il faut noter ici que l'on n'a pas parlé de mémorisation logique qui intervient éventuellement (elle n'existe pas dans tous les cas) après détection, mais des problèmes de détection d'occurrence. Le problème est le même pour les sources implicites dont l'implantation fait appel à l'intégration contrôlée de primitives du type POST en des points précis.

II.4.3.- La détection

D'une manière générale, la détection se fait par comparaison d'états (état antérieur, état actuel) ou alors en cas de mémorisation physique sur la présence ou l'absence du signal, c'est-à-dire sur un seul état. Une scrutation est toujours nécessaire, dans tous les cas, qu'elle soit matérielle ou logicielle.

II.4.3.1.- Signaux à état

a) mémorisé physiquement

L'algorithme de détection peut se résumer à :

Cycle
- scruter voie physique à travers la mémorisation associée
- si mémorisation = état donné alors une occurrence est détectée
- remise de la mémorisation à état qualifié d'initial.

fin cycle

b) non mémorisé physiquement

Algorithme de détection :

- fixer état initial de la voie

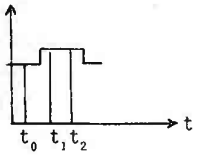
Cycle
- scruter voie physique -- représente l'$état_{i+1}$ de la voie
- si $état_i$ $\begin{cases}>\\<\end{cases}$ $état_{i+1}$ alors occurrence détectée
- état i+1 = état i

fin cycle

Exemple : voie booléenne - transition 0 → 1

t_0, t_1, t_2 sont les instants de scrutation

état initial = 0



Cycle

si $état_i$ < $état_{i+1}$ alors occurrence détectée

$Etat_{i+1}$ = $Etat_i$

fin cycle

Ceci permet une détection en t_1 mais pas en t_2.

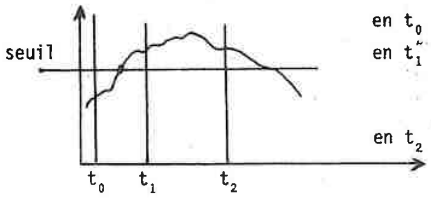
Note : Dans ce cas, l'état est défini par la mémorisation propre au signal et obtenu simplement par lecture de la voie.

II.4.3.2.- Signaux continus

Dans la mesure où, en ce qui concerne les signaux continus, la détection se fait par rapport à un état donné (dépassement de seuil par exemple), état qui par nature varie, il est indispensable de tenir compte de cette variation au cours de la détection :

- scruter la voie physique
- si $état_{i+1}$ > (resp. <) $état_i$ et
  $état_i$ ≤ (resp. ≥) seuil alors occurrence détectée

Les états représentent les valeurs de la voie physique scrutée.



en t_0 : pas d'occurrence
en t_1 : occurrence car
$état_i = état_{t_0} <$ seuil
$état_{i+1} = état_{t_1} >$ seuil
en t_2 : pas d'occurrence car
$état_{i+1} = état_{t_2} >$ seuil
$état_i = état_{t_1} >$ seuil

II.4.3.3.- Sources internes

Concernant les sources internes, on remarque que production d'occurrence et détection sont deux opérations simultanées.

La primitive POST(evt), par exemple, constitue à la fois la production de l'occurrence (primitive en elle-même) mais également le moyen de détection (appel au détecteur). Ce n'est pas le détecteur qui va "chercher" l'occurrence mais c'est l'occurrence qui vient au détecteur. L'algorithme de détection est par conséquent implicitement défini par la primitive elle-même.

II.4.4.- Problème de la perte d'occurrences

II.4.4.1.- Echantillonnage

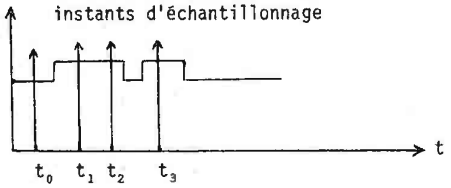
A travers les différents types de détection examinés précédemment, on remarque que la détection des occurrences se fait toujours par examen périodique (période fixe ou variable) des voies physiques sauf pour les sources internes. On se contente en fait d'une approximation dans l'examen du signal et ceci pour la simple raison qu'une observation continue n'est pas possible (échantillonnage obligatoire).

De cette approximation résultent ou peuvent résulter des erreurs d'observations desquelles peut découler une perte d'occurrences. Il devient important alors, pour un signal donné, de fixer soigneusement la période d'échantillonnage.

II.4.4.2.- Perte d'occurrences

Comme déjà vu, la perte d'occurrences est liée à la fréquence d'échantillonnage.

Prenons quelques exemples pour illustrer ces propos.



On s'aperçoit que la fréquence d'échantillonnage est telle que entre t_2 et t_3 il y a un changement d'état qui passe inaperçu.

Il en résulte une perte d'occurrences due à un défaut de détection. L'exemple pris concerne plus un signal sans mémorisation physique. Il en est exactement de même dans le cas de mémorisation physique : dans ce cas, si le cycle de scrutation est trop élevé, il

peut se passer qu'une occurrence soit perdue par mémorisation d'une seconde occurrence arrivée entre temps.

Le problème est différent pour ce qui est des sources internes car dans ce cas il y a impossibilité de produire une occurrence tant que l'occurrence précédente n'a pas été détectée. Il en résulte qu'il ne peut y avoir perte d'occurrences dans ce cas (au niveau de la détection, cela s'entend).

II.4.5.- Différents types de détection

La détection peut être entièrement prise en charge et programmée au niveau de l'application : on dira qu'elle est programmée. Cela oblige le concepteur à considérer que la détection fait partie de la nature du problème traité et peut donc nuire à la clarté de la démarche. Une détection système, au contraire, suppose une prise en compte totale de la détection au niveau du système et surtout une séparation entre les problèmes de détection et les problèmes liés à la solution envisagée.

Il s'agira alors, pour un concepteur, de préciser au moment voulu comment il désire faire cette détection.

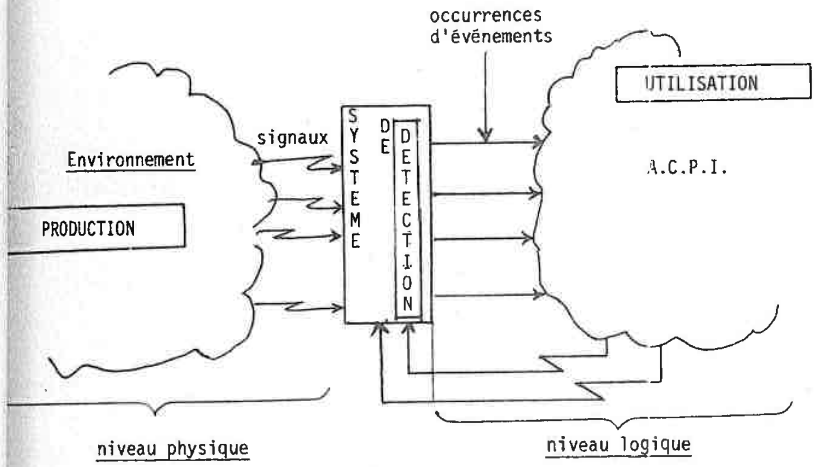
II.5.- UTILISATION DES OCCURRENCES

II.5.1.- Niveau physique - Niveau logique

Dans toute l'analyse précédente, seul l'aspect physique des phénomènes pouvant survenir dans un procédé industriel donné a été étudié. Cet aspect physique constitue bien évidemment une partie importante du procédé industriel puisqu'il est à la base de tous les traitements appliqués.

D'un autre côté, nous avons l'ACPI elle-même pour laquelle il faut également définir une façon de voir le procédé, façon de voir

qui doit être la plus indépendante possible de l'aspect physique. Autrement dit, l'ACPI doit être doté d'un mécanisme d'abstraction permettant une représentation du procédé industriel sans faire référence à une quelconque réalisation. Nous appelons niveau logique le niveau constitué par l'ensemble ACPI et son mécanisme d'abstraction. Le niveau physique sera constitué pour tout ce qui est source d'occurrences internes ou externes. Entre ces deux niveaux se trouve le système de détection.



Le mécanisme d'abstraction est constitué par ce que l'on appelle les événements.

Nous pouvons dès à présent souligner plusieurs points :

- un événement est une entité purement logique constituant une abstraction d'une partie de l'environnement (les événements internes de l'ACPI peuvent également être considérés comme faisant partie de l'environnement pour certains composants)

et permettant une synchronisation entre le système physique et les composants d'une ACPI et entre composants de l'ACPI.

- il est souhaitable de pouvoir composer ces événements pour réaliser les synchronisations plus élaborées.
- il est nécessaire de bien définir les liens entre le niveau logique et le niveau physique.
- Le système de détection constitue en quelque sorte une implantation des événements.

II.5.2.- Différents types d'événements

II.5.2.1.- Evénements simples

Un événement simple, ou événement élémentaire représente, par définition l'abstraction d'un seul phénomène physique donné. Il est nommé à travers un identificateur.

Exemple : On peut décider que l'événement d'identificateur e_1 représente la fermeture de la porte i d'un four de cuisson.

Toute occurrence de e_1 sera considérée comme représentant une fermeture de porte i et traitée en tant que telle.

II.5.2.2.- Evénements dérivés

Un événement dérivé est le résultat de l'application d'un opérateur monadique sur un événement simple.

Les opérateurs monadiques peuvent être du type :

- <nombre d'occurrences> de <événement>
- <délai> après <événement>.

.....

Un événement dérivé comporte également un identificateur et une déclaration.

Un tel événement peut être considéré comme ayant une source interne.

II.5.2.3.- Evénements composés

Un événement composé est défini par une expression d'événements construite à partir :

- d'évenements simples
- d'événements dérivés
- et d'opérateurs diadiques (ET, OU, AVANT, APRES ...).

Un événement composé comporte une déclaration et un identificateur. Si pour les événements simples, la sémantique est relativement claire, les événements dérivés et composés demandent à être précisés un peu plus. Il se pose notamment un problème de définition d'une sémantique précise dans l'interprétation des expressions d'événements, occurrences concernées, résultat d'évaluation d'expressions.

Notons également que les événements composés comportent un aspect entièrement interne.

Les événements composés et dérivés reviennent en fait à la définition de nouvelles sources internes d'événements simples. Cela revient à dire que vis-à-vis du traitement, l'on considère toujours des événements simples.

II.5.3.- Prise en compte des occurrences au niveau du traitement

Tout comme pour les problèmes de détection d'occurrences d'événements, à nouveau vont se poser ici, les deux problèmes de :

- mémorisation, mais cette fois-ci logique, des occurrences.
- perte d'occurrences pouvant résulter d'une mauvaise prise en compte de celles-ci au niveau du traitement ou plus simplement être volontaire.

II.5.3.1.- Notion de mémorisation logique

Précisons tout de suite que lorsqu'une occurrence est détectée, il est toujours possible de la mémoriser. Le problème est alors de savoir si d'un point de vue traitement, il est souhaitable ou utile de faire une mémorisation ou non.

Cette mémorisation logique peut être liée à la nature même de l'événement (exemple [FLYS 75]) ou au contraire dépendante du traitement.

Plusieurs degrés de mémorisation peuvent alors être définis.

a) Prise en compte fugitive

Vis-à-vis d'un traitement donné, aucune mémorisation d'occurrence n'est définie. Toute occurrence détectée est alors soit immédiatement prise en compte si le traitement associé est disponible soit perdue pour le traitement. Dans ce cas, on accepte donc la perte d'occurrence et l'on estime qu'il est sans gravité pour le traitement.

b) Mémorisation d'une seule occurrence

Ce cas rejoint le cas de la mémorisation physique (système d'interruption). C'est l'occurrence la plus récente qui est prise en compte dans le traitement - Ici également on admet l'idée de perte d'occurrences.

c) Mémorisation totale de toutes les occurrences

Aucune perte d'occurrence n'est tolérée dans ce cas. Elles sont toutes mémorisées en vue d'un traitement différé.

II.5.3.2.- Notion de partage d'événements

Il nous semble normal d'admettre que plusieurs traitements puissent être liés à un même événement. L'événement est alors dit partagé.

Il est alors nécessaire de définir des règles de partage précises afin de garantir une bonne utilisation des événements. Le partage se fait au niveau des occurrences.

a) Consultation d'occurrences

Un traitement "consulte" une occurrence signifie que celle-ci déclenche le traitement mais qu'elle reste présente et peut éventuellement déclencher d'autres traitements.

b) Consommation d'occurrences

Dans certains cas, inscrits dans la logique du traitement, il est nécessaire de s'"accaparer" une occurrence pour en garantir un traitement correct. Dans ce cas, l'occurrence n'est plus disponible pour d'autres traitements.

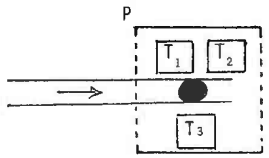
c) Duplication d'occurrences

Si dans certains cas un événement est utilisé à la fois en consommation et en consultation, il devient nécessaire de dupliquer les occurrences pour assurer un traitement correct.

Une occurrence ne peut être consommée qu'une seule fois.

Exemple 5.3.2.

Soit un poste d'usinage représenté par le schéma suivant :



Les pièces arrivent à un poste d'usinage P au niveau duquel trois traitements sont possibles :

- T_2 : préparation d'un état de sortie
- T_1 et T_3 traitements équivalents servant à doubler les capacités du poste P.

Soit e l'événement représentant l'arrivée d'une pièce.

Il est évident que T_1 et T_3 ne peuvent utiliser les mêmes occurrences de e. De plus elles devront consommer les occurrences.

Pour T_2 il s'agira uniquement de consulter les occurrences.

La consommation exige donc une duplication pour T_2.

II.5.3.3.- Expression de la prise en compte des occurrences d'événements

Nous nous situons ici au niveau utilisation des occurrences d'événements. Il s'agit par conséquent de la prise en compte des occurrences d'événements au niveau des composants d'une ACPI (tâches ...).

Comme suggéré dans le chapitre I, il existe deux possibilités d'expression de cette prise en compte :

a) Statique

Dans ce cas, la structure de prise en compte est nécessairement en dehors de tout texte de programmes car elle doit être indépendante de toute idée d'exécution. Ce type d'expression va dans le sens d'une séparation entre les parties opérative et contrôle et favorise donc l'évolutivité, lisibilité ...

Etant donné nos préoccupations, c'est vers ce type d'expression que nous penchons.

Exemple de Sygare [THOM 80] :

sur ev_k faire $tâche_j$

Notons que ce type d'expression permet de véritablement spécifier le comportement d'une ACPI sans être obligé d'examiner dans le détail comment sont effectués les traitements.

De plus, ne dépendant que de la nature du problème et étant complètement indépendante de toute exécution, la spécification présente l'avantage de rester valide quelle que soit l'implantation adoptée.

b) Dynamique

La structure de prise en compte se situe dans le texte d'entités destinées à être exécutées (programmes, processus de LADET [LAD 77]).

On peut donner comme exemple des primitives du type WEVENT [SEMS III, WAIT (SIRIS 7/8) ... mais également des structures plus élaborées du type Pour evt... faire tâche [LAD 77, LAD 82] situées à l'intérieur de ce que LADET appelle un processus exécutable.

Dans ce cas, la sémantique que l'on peut donner à une telle expression (occurrence concernée notamment) est forcément dépendante de l'instant d'exécution de cette expression. C'est cet aspect dynamique qui peut être, rend de telles expression plus proches de la programmation que de la spécification de comportements.

c) Notion de lien et d'instant de lien

Liée à cet aspect expression de la prise en compte des occurrences, nous trouvons une notion importante soulignée par LADET [LAD 77] : la notion de lien.

Un lien est défini comme la structure qui permet de lier un événement et le traitement associé. Associé à ce lien est défini un instant de lien qui est simplement l'instant d'évaluation du lien. La sémantique d'un lien est alors entièrement précisée par rapport à cet instant : occurrences passées, futures ...

Dans le cas d'une structure dynamique, cet instant dépend bien sûr de l'exécution de l'entité renfermant le lien. Que devient alors cet instant de lien dans le cas d'une structure statique ?

On peut considérer qu'il est défini une fois pour toutes par un instant de référence considéré comme absolu et qui est l'instant de lancement de l'ACPI. La sémantique d'une telle structure peut ainsi être précisée, de manière statique, par l'intermédiaire d'expressions d'événements.

II.5.3.4.- L'état du système : les conditions

Dans la prise en compte des occurrences d'un événement, il est nécessaire de préciser non seulement quelles occurrences on désire traiter mais également dans quelles conditions elles doivent être traitées. L'état du système global (aussi bien informatique que le procédé lui-même) détermine les traitements à appliquer et les occurrences d'un même événement peuvent recevoir des traitements différents selon l'état du système = le fait d'appuyer sur un bouton d'appel d'un ascenseur peut provoquer une montée, une descente ou une ouverture de porte selon l'état dans lequel il se trouve.

L'expression de ces conditions peut prendre différentes formes :

- expression statique dans SYGARE [THOM 80] :

  sur evt [si cond] faire tâche

- directement dans le corps d'une tâche, par programmation donc, dans le cas de structure dynamique.

II.5.3.5.- Priorité de prise en compte

En parlant de priorité, il nous semble important de distinguer deux types de priorités :

a) Priorité de détection

Celle-ci intervient lors de la détection des occurrences d'événement et est de ce fait plus "physique". Elle permet de tenir compte de la nature du signal traité : impulsionnel, stable ... et intervient dans la phase de définition des liens entre le niveau physique et le niveau logique. Le choix de connexion d'un capteur à un niveau d'interruption détermine, par exemple, la priorité de détection. Elle peut être fonction de la durée de vie intrinsèque des occurrences.

b) Priorité de traitement

Cette priorité définit l'urgence relative avec laquelle doit être traitée l'occurrence d'un événement. Si la priorité de détection dépend plus de choix d'implantations, la priorité de traitement au contraire nous semble plus liée à la nature même du problème.

Certains auteurs attachent cette priorité aux événements eux-mêmes. Nous préférons, pour notre part, la lier aux traitements (tâches) pour la raison suivante : en cas de partage, un événement n'est pas nécessairement "vu" de la même façon par tous les traitements ; une occurrence qu'il est urgent de traiter pour une tâche i peut très bien ne pas avoir ce même caractère d'urgence pour une tâche j. Une occurrence d'un événement alarme peut déclencher deux traitements : disjonction de circuits électriques par exemple et avertir l'opérateur. On ne peut pas faire ces deux traitements avec le même degré d'urgence.

Cela veut dire en particulier que si un problème de choix se posait entre ces deux traitements, c'est le premier traitement qui est privilégié.

II.5.3.6.- Délai de prise en compte des occurrences : les exceptions

Si l'on admet que pour des occurrences fugitives, le délai de

prise en compte est nul par définition (ou bien les occurrences sont immédiatement traitées ou bien elles deviennent, aussitôt produites, caduques), il est en revanche intéressant de pouvoir fixer un délai de prise en compte pour les occurrences mémorisées. Ce délai exprimera un intervalle de temps maximum au bout duquel le traitement d'une occurrence devient inutile. Ce délai revient alors à la définition d'une échéance de traitement et peut être utilisé pour augmenter la priorité d'une tâche à l'approche de l'échéance.

Ce délai peut être attaché à un événement mais également, et c'est là notre point de vue, à un traitement. En cas de partage, en effet, chaque tâche aura ainsi sa propre vue de l'événement. Le délai définissant une certaine urgence de traitement, les tâches n'ont pas forcément la même notion d'urgence.

Associées à ces délais, nous trouvons les notions d'exceptions provoquées par le non-respect de ces délais. A ces exceptions correspondent des traitements dits traitements d'exception.

Plusieurs expressions, selon que la structure de prise en compte est statique ou dynamique sont possibles.

II.6.- RELATION ENTRE LE NIVEAU PHYSIQUE ET LE NIVEAU LOGIQUE : ASSIGNATION

II.6.1.- Indépendance physique - logique

Dans la conception de systèmes de contrôle de procédés industriels, il est essentiel, pour des raisons d'évolutivité et de reprise sur panne, de rester le plus indépendant possible du matériel utilisé.

Le système de contrôle doit donc pouvoir travailler sur un "bon" modèle du système physique, modèle duquel sont exclues toutes références au système physique. Ceci permet d'obtenir un ni-

veau d'abstraction, défini dans notre cas par l'ensemble ces événements représentatifs du système physique à contrôler. Nos événements se situent à un niveau logique sans une quelconque référence à une implantation possible.

Corollairement à cette indépendance, il faut définir une opération permettant de réaliser la liaison entre le niveau physique et le niveau logique. Pour les raisons déjà évoquées plus haut, cette liaison doit être aisément modifiable.

II.6.2.- Opération d'assignation

Nous appelons assignation la liaison entre un événement et sa source (aspect logique ⇔ aspect physique).

L'assignation peut être fixe et définie une fois pour toutes pour toute la durée de vue du système de contrôle ou au contraire être dynamique et répondre aux impératifs d'évolutivité et de tolérance aux pannes.

Dans le cas d'une assignation fixe, toute évolutivité est compromise dans la mesure où toute modification du matériel se traduit par une recompilation d'une partie du système de contrôle supposant une localisation des endroits à modifier.

Une assignation dynamique, au contraire, rend le système de contrôle indépendant du matériel et intervient dans la phase ultime d'un système de contrôle : l'implantation.

C'est au niveau de l'assignation que sera précisée la nature physique d'un événement et le moyen de détection.

Une assignation dynamique peut s'exprimer par :

assigner <événement> à <source >
détecté par <moyen de détection>

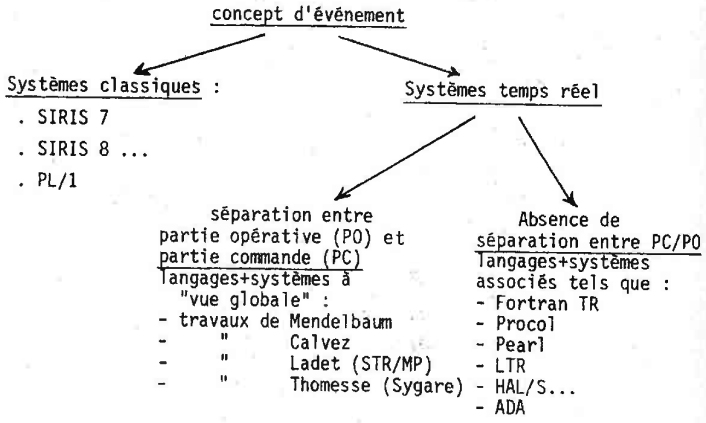
Notons pour terminer qu'un moyen de détection peut être considéré comme faisant partie et pris en charge par une application ou alors comme pris en charge par le système sous lequel s'exécute

l'application. L'assignation dynamique est un moyen de dissocier les moyens de détection, de l'application.

II.7.- LES EVENEMENTS DANS LES AUTRES SYSTEMES : ETUDE COMPARATIVE

Ce paragraphe constitue une étude de la notion d'événement et de l'utilisation qui en est faite dans quelques systèmes existants en fonction des définitions et des concepts énoncés plus haut.

Avant d'aborder cette étude, nous donnons une classification générale des types de systèmes utilisant le concept d'événement et pouvant servir de guide pour la suite.



II.7.1.- Critères de comparaison

Les critères de comparaison sont orientés selon les trois grands points soulignés : production (sources) - détection - utilisation.

Un certain nombre de systèmes et langages ont été choisis pour cette étude sur l'aspect événement.

Nous avons néanmoins essayé de représenter les "grandes" familles de systèmes ou langages utilisant le concept d'événement : systèmes classiques non-temps réel, systèmes temps réel, avec l'aspect présence ou absence de la séparation en partie opérative - partie contrôle.

a) Production - détection

Dans la plupart des systèmes étudiés il existe des événements purement internes (ou encore logiciels) explicitement définis et utilisés ou implicitement utilisés (comme les fin de tâches par exemple).

D'un point de vue externe cependant, à une exception près ([LAD 82]), les seules sources admises sont : les interruptions et les tops d'horloge. Les autres sources sont simplement traitées comme Entrées/Sorties obligeant par là le concepteur à assurer lui-même la détection et signalisation d'un certain nombre d'événements. Nous estimons pour notre part que les problèmes de détection qui sont des problèmes d'implantation ne doivent pas être traités au niveau de l'application. Il faut donc, véritablement arriver à considérer les événements comme une abstraction de l'évolution du procédé et se concentrer sur la façon de prendre en compte cette évolution. Pour cela il faut, bien sûr, élargir la définition des sources possibles d'évenements.

b) Utilisation

A ce niveau, que nous avons qualifié de logique, il s'agit de préciser :

- les types d'événement (simple, composé...)
- types de mémorisation : fugitif, mémorisation totale.
- occurrences utilisées : non précisées, toutes,...
- partage : consommation, consultation, duplication.
- prise en compte des exceptions.

- priorité : de détection, de traitement.
- instant de lien
- prise en compte de l'état du système
- type d'assignation.

Dans la suite nous développerons une étude que nous avons voulu la plus synthétique possible sur l'aspect événement dans les systèmes non-temps réel suivie d'un tableau comparatif entre différents systèmes ou langages dits temps réel.

L'aspect production-détection ayant été résumé dans le sous-paragraphe a), nous nous intéresserons plus à l'aspect utilisation.

II.7.2.- Les systèmes classiques

Par systèmes classiques, nous entendons systèmes non temps réel du type SIRIS 8. Nous ne traitons pas le cas d'un système spécifique mais essayons de rester le plus général possible.

a) Production

Source : généralement interne et programmée à l'aide de primitive du type POST (evt), SIGNAL (evt) ...

b) Détection

Elle est mise en place pour des sources internes et assurée par appel au système (primitive POST).

c) Utilisation

- Prise en compte des occurrences : il n'y a aucune vue globale de l'application. Cette prise en compte se fait par primitives (WAIT (evt)...) diffuses dans le texte des programmes de l'application.

- types d'événements :
  - . simples
  - . composés : *1 parmi n (OU)
    *n (ET)

- type de mémorisation : imposée par le système et généralement de type mémorisation d'une seule occurrence.
  (Les sémaphores peuvent toutefois être considérés comme événements à mémorisation totale (toutes les occurrences sont mémorisées) lorsqu'ils sont utilisés pour la signalisation).
- La mémorisation est attachée à la définition de l'événement : les tâches ne peuvent pas avoir des vues différentes d'un même événement.
- partage : généralement admis avec une sémantique mal définie. Il n'y a pas de notion de consultation d'une manière générale et la consommation est implicite. La duplication n'est pas assurée : il faut la programmer explicitement (deux POST (evt) par exemple).
- exceptions : il n'y a pas de délai de non prise en compte des occurrences d'un événement. Ce contrôle doit être explicitement programmé.
- la notion de priorité de détection n'existe pas (source interne). La priorité de traitement est attachée à la tâche.
- l'instant de lien est l'instant d'exécution d'une primitive de type WAIT(evt).
  Les occurrences traitées peuvent être passées (si elles sont déjà produites) ou futures (cas de l'attente) sans aucune précision.
- Etat du système : pris en compte par programmation explicite.

d) Assignation

La sémantique des événements est implicite et il n'y a pas de notion d'assignation. Cela revient en fait à une assignation fixe.

II.7.3.- Les systèmes temps réel

II.7.3.1.- Langage temps réel sans vue globale

Il existe de nombreux langages pouvant entrer dans cette catégorie et une étude exhaustive de tous les langages serait longue.

Nous avons choisi un certain nombre de langages que nous avons estimé représentatifs de cette classe de langages.

Les comparaisons sont consignées dans le tableau II.7.

II.7.3.2.- Systèmes à "vue globale"

Là également, nous avons établi une comparaison entre quelques uns parmi eux.

Cette comparaison se trouve dans le tableau II.7.

Concernant les travaux de CALVEZ J.P [CALV 82], nous pensons qu'ils procèdent d'une démarche différente. Dans cette démarche, il s'agit de démarrer d'un cahier des charges sans aucune supposition sur le matériel et d'aboutir à une solution comprenant à la fois les aspects matériels et logiciels. Les démarches des systèmes exposés présupposent au contraire un matériel, et sont axés sur la conception du logiciel. SYGARE (THOM 80) fait partie de cette classe de systèmes. La démarche de CALVEZ permet de démarrer d'une structure fonctionnelle (Modèle fonctionnel) décrivant le comportement du système à implanter et d'aller progressivement vers une structure d'exécution puis un modèle d'intégration permettant le choix du matériel le plus apte à répondre au modèle fonctionnel y compris dans l'aspect contraintes de temps. Le choix du matériel paraît donc déterminant et c'est le matériel qui doit se "plier"

au modèle fonctionnel.

Si dans les systèmes exposés, le choix des structures d'expression du contrôle paraît avoir été guidé par un souci de rendre compte d'un certain nombre de situations possibles (mémorisation d'occurrences ...), la démarche de CALVEZ impose a priori des structures de contrôle dont on garantit le bon fonctionnement par le bon choix du matériel.

Ceci a été possible grâce à l'intégration de toutes les phases d'un projet d'automatisation : de la conception à la réalisation.

La nature des problèmes n'étant pas la même, ces types de systèmes seraient difficilement comparables entre eux étant donnés les critères que l'on s'est fixés.

critères / systèmes	Type d'événement	Type mémorisation	occurrences prises en compte	partage	Traitement exception
proc◊l [rit pal 75]	- simple - composé : ou sur les interruptions software	- mémorisation une occurrence	non précisées (la plus récente implicitement)	- partage non précisé - consommation implicite - Absence de duplication	- time-out prévu dans les événements AVEC delai
HAL/S [Fly S 75]	- simple, - composé : ET, ou, non sans précision sur les occurrences	- mémorisation une occurrence - Fugitif (UNBatched)	non précisées (la plus récente implicitement)	- partage Admis - consommation par Reset - consultation - Absence de duplication	- time-out A programmer explicitement
PEARL [Elz 75]	- simple	- mémorisation d'une occurrence (mal precisé)	non precisées (la plus récente implicitement)	- partage non précisé	- A programmer explicitement
LTR.V3 [par 80]	- simple	- mémorisation d'une occurrence - fugitif	non precisées (la plus récente implicitement)	- partage Admis - consommation implicite sur wait - pas de duplication	- A programmer explicitement
GAELIC * [mend 76] [lecal 79]	- simple - composé : ET, OU, ⇒ OUex sans précision sur les occurrences	- mémorisation une occurence (chaque) - non précisé pour d'autres structures	non precisées (la plus récente implicitement)	- partage non precisé - pas de duplication	- A programmer explicitement
STR/MP * [lad 82] [lad 77]	- simple - composé : ET "parallèle" et "sequentiel"	- mémorisation totale - mémorisation une occurrence	par rapport à l'instant de lien : - occurrences passées - occurrences futures - toutes ou la plus recente	- Absence de partage - consommation implicite - pas de duplication	- A programmer explicitement

* : systèmes à vue globale

priorité de detection	priorité de traitement	instant de liens	etat du système	ASSIGNATION
- priorité IT pour les evts externes	priorité de l'interruption (tâche immediate)	- instant d'execution de primitive WAIT - prise en compte de l'occurrence la plus récente ou la prochaine par rapport à cet instant	- non explicitement pris en compte	- fixée pour les IT à la generation du système - difficilement modifiable
- Absence de priorité pour les pseudo - IT	priorité tache (n° entier)			
- Aspect externe non precisé		- instant d'execution de primitive WAIT FOR EVT - même remarque que precedemment	- idem	Aspect externe non précisé
- Absence priorité pour les EVTS logiciel	priorité tâche			
priorité IT pour l'aspect EXTERNE	priorité IT	- instant d'execution de primitive WHEN ... - même remarque	- idem	- ASSURÉE dans la SYSTEM Division - modifiable par recompilation
utilisation de SEMAPHORE de signalisation				
priorité IT pour l'aspect externe	priorité IT pour les tâches immediates	- instant d'execution de primitive WAIT - même remarque	- idem	- ASSURÉE dans "l'article" system DATA - modifiable par recompilation
- absence de priorité pour les événements internes	priorité DE la tâche			
priorité IT pour les EVTS EXTERNES	priorité IT	instant d'execution de structure la prise en compte d'occurrence (quand, si...) - même remarque	- prise en compte par variables dites d'etats attachés aux EVENEMENTS EXTERNES	- ASSURÉE dans la partie environnement - modifiable par recompilation
priorité IT pour les EVTS induits	priorité de l'evenement induit (≠ de la priorité IT)			
priorité IT pour les interruptions	priorité EVENEMENT	instant d'exc de structure de prise en compte (pour EVT...) distinction entre occurrences passées et futures / cet instant	- Explicitement pris en compte au niveau global	- ASSURÉE dans la partie déclaration d'evenements - modifiable par recompilation
non precisée pour les autres evenements	priorité d'EVENEMENTS			

TABLEAU II 7

De l'analyse précédente (chapitres I et II), il ressort que :

- le concept d'événement associé à une prise en compte statique des occurrences permet une expression claire et explicite du contrôle des activités d'une application.
- et qu'en même temps, pour permettre une spécification correcte d'une ACPI, ce concept doit être enrichi par :
  * l'introduction, avec la définition d'une sémantique précise, d'expressions d'événements qui permettent d'augmenter "l'expressibilité" de la spécification. Par ce biais on pourra tenir compte par exemple des occurrences "passées" ou "futures" telles qu'introduites par LADET [LAD 77] et ce de manière statique.
  * la définition précise du partage d'événement qui nous semble une notion importante mais mal définie.
  * la prise en compte des exceptions de manière plus "naturelle" dès la spécification.
  * la prise en compte de l'état du système dès l'étape de spécification également.

II.8.- NOS PROPOSITIONS

Au cours de ce chapitre de définitions générales mais également, de mise en évidence des caractéristiques se rapportant au concept d'événement, trois phases liées à l'existence d'un événement ont été reconnues : la production des occurrences, la détection des occurrences et enfin l'utilisation des occurrences.

Nous essayerons de traduire, dans nos propositions, ce souci

constant de séparation entre ces trois parties, ce qui, nous l'espérons, contribuera à la clarification des concepts attachés à la notion d'événement. Cela se traduira par des propositions pour les phases de :

i) Production

Donner la possibilité d'exprimer de manière indépendante les sources des événements prises en compte : c'est l'assignation.

ii) Détection

Préciser dans cette assignation éventuellement des modules de détection mais également des priorités de détection.

iii) Utilisation

- expression claire du partage
- définition de priorités de traitement
- occurrences utilisées
- état du système
- prise en compte des exceptions

avec une définition la plus large possible des synchronisations (événements composés) et en gardant beaucoup de propositions faites dans [THOM 80] portant sur :

- la séparation partie contrôle - partie opérative
- l'aspect communications.

Un rappel sur SYGARE [THOM 80] est donné dans l'annexe I.

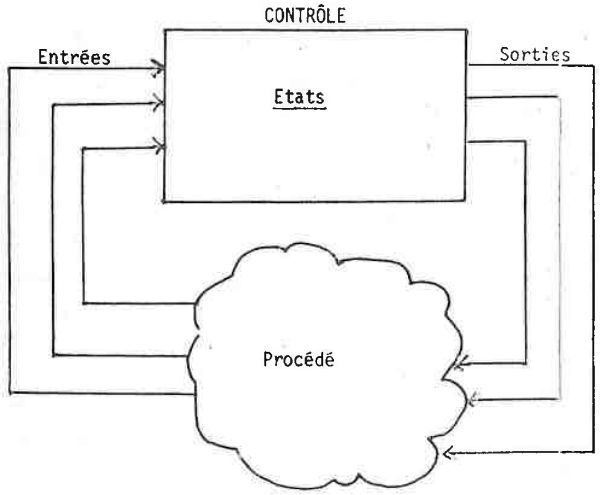
Nous espérons obtenir ainsi, une spécification claire d'une ACPI.

CHAPITRE III :

ÉVÉNEMENTS SIMPLES
ÉVÉNEMENTS COMPOSÉS
EXPRESSION D'ÉVÉNEMENTS

III.1.- INTRODUCTION

Si on reprend la représentation classique de l'ensemble procédé + application qui le contrôle sous la forme d'un automate :



modélisable en termes de :

- élaboration des états du système de contrôle en fonction des états précédents et des entrées

$$état_n = f_1(état_{n-1}, entrées_n)$$

- calcul des sorties en fonction de l'état du système de contrôle et des entrées

$$sortie_n = f_2(état_n, entrées_n)$$

(n représentant l'instant de calcul),

on peut remarquer qu'il existe deux systèmes dynamiques (évoluant donc en fonction du temps) en présence : le procédé lui-même

qui représente l'aspect externe et le système de contrôle (ou ACPI) qui représente l'aspect interne.

Ceci se traduit, au niveau de la spécification par la prise en compte de l'évolution de ces deux systèmes :

- évolution du procédé prise en compte par l'intermédiaire de la variation des entrées et qui donne ce que l'on a appelé des sources externes.
- évolution du système de contrôle lui-même pris en compte par l'intermédiaire des changements d'état et qui donne les sources internes.

D'un point de vue spécification, ces deux types d'évolution seront pris en compte par l'intermédiaire du concept d'événement sans essayer de faire de distinctions.

On s'aperçoit que, dès que l'on veut spécifier des systèmes dynamiques (en construire un modèle donc), on est obligé, et c'est bien normal, de tenir compte de la variable temps donc des événements qui, en fait, peuvent être considérés comme une abstraction du temps.

De cette façon, on peut justifier l'importance que l'on accorde aux événements, précisément lorsque, dès l'étape de spécification, on veut prendre en compte l'évolution de l'ensemble du système.

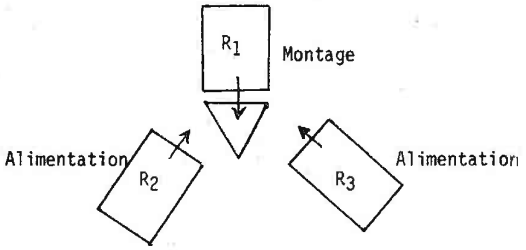
De plus, lorsqu'on parle de temps, on parle forcément de synchronisation. Le problème essentiel est, en effet, l'expression des synchronisations qui peuvent exister entre ces différents éléments du système global.

Dans un tel contexte, il est aisé de comprendre l'introduction d'événements simples mais aussi composés et dérivés (la dérivation étant une forme de composition).

Il faut alors préciser le choix des opérateurs syntaxiquement et sémantiquement.

L'introduction d'opérateurs traduit la volonté de prendre en compte, dès l'étape de spécification , une certaine réalité des faits consignée dans un cahier des charges.

Si on prend, en effet l'exemple de trois robots, l'un de montage et les deux autres d'alimentation en pièces :



il est évident que pour spécifier le fonctionnement de R_1, on doit préciser qu'il ne fonctionne que sur occurrences de pièces arrivant de R_2 et R_3. En d'autres termes, si on veut représenter cette synchronisation par le biais d'un événement, cet événement sera certainement composé à partir de deux événements simples et d'un opérateur ET. De plus, il ne s'agira pas de prendre en compte n'importe comment les pièces. Il faut alors, dans la composition, préciser qu'il s'agit de composer une à une toutes les occurrences provenant de l'alimenteur R_2 et toutes les occurrences provenant de l'alimenteur R_3. D'autres opérateurs sont nécessaires pour exprimer d'autres synchronisations.

Il est également important, comme l'a souligné LADET [LAD 77 et LAD 82] de préciser quelles sont les occurrences prises en compte par rapport à l'instant de liens : passées, futures ... Nous pensons pouvoir exprimer cela également par le biais d'un événement composé en introduisant les opérateurs "temporels" du type AVANT, APRES. L'introduction d'événements composés nous permet de traiter ce problème non pas en tant qu'utilisation des occurrences par le traite-

ment mais en tant que spécification des règles de synchronisation. Le choix des opérateurs est bien sûr important et devra tenir compte des caractéristiques des synchronisations à exprimer. Ainsi, par exemple, la prise en compte des occurrences d'un événement survenues après une certaine date (depuis d, tous les Δt faire ...) pourra se définir par l'intermédiaire de l'opérateur APRES.

Il nous a semblé alors intéressant, pour mieux justifier les choix que nous seront amenés à faire, de revenir sur la notion d'événement défini en tant que suite d'occurrences.

Cette notion de suite temporelle (voisine de celle utilisée par J. JARAY et J.P. FINANCE du CRIN (Nancy)) permet de mettre en évidence les grandes classes d'opérations que nous allons introduire sur les événements.

III.2.- LES EVENEMENTS EN TANT QUE SUITES

Rappelons qu'un événement est défini en tant que suite ordonnée d'occurrences connues par leurs instants de détection.

Liés à cette notion de suite, on peut mettre en évidence deux aspects :

- un aspect "fabrication" de la suite ou encore production de ses éléments constitutifs,
- un aspect utilisation de la suite : consultation d'éléments, prélèvement d'éléments.

Or, conceptuellement, un événement peut, précisément, être vu en ces termes. C'est ce qui nous permet, alors, de ne plus considérer l'événement comme concept véritablement nouveau mais de le rapprocher du concept de suite et d'essayer de déduire les opérations que l'on peut faire sur un événement à partir des opérations sur les suites.

Soulignons toutefois l'aspect dynamique des suites d'occurrences, aspect dû à leur côté temporel. C'est ce qui les différencie

essentiellement des suites classiques.

Dans ce cas, en effet, dire que l'on veut "composer" une suite S_1 et une suite S_2 pour produire une suite résultat S suppose que cette opération "composer", dès lors qu'elle est "lancée" ne s'arrête plus tant qu'il y a des occurrences de S_1 et S_2 (et rien ne dit qu'il n'y en aura plus).

Ce qui veut dire que toute opération ne prend pas une suite figée S_1 et une suite figée S_2 pour produire une suite figée S mais au contraire se poursuit dans le temps au fur et à mesure que les occurrences arrivent. Il faut bien évidemment rendre compte de cet état de fait dans la sémantique des opérations et c'est une des raisons qui nous ont poussé à choisir les chronogrammes pour rendre compte de cette sémantique.

III.2.1.- Fabrication d'une suite : production des éléments

La construction d'une suite peut se faire de deux façons :

- les éléments proviennent directement d'un producteur qui peut être une source externe par exemple. Dans ce cas, on parlera de suite simple par analogie aux événements simples.
- les éléments proviennent de compositions de suites simples, et dans ce cas on a une suite composée.

En parlant de termes de types abstraits, les profils des opérations de production pourraient être

- soit de la forme élément×suite → suite
- soit de la forme suite×suite → suite

Ces opérations de production joueront le rôle de constructeurs de type [DER & al 79*, GUT 78, GUT & al 78, GOG & al 78].

Concernant le choix des opérations de composition de suites, on s'intéressera plus à celles qui sont adaptées à l'aspect temporel :

- production d'une occurrence sur présence simultanée de deux occurrences particulières : synchronisation ET.

- production d'une occurrence sur présence d'une occurrence au moins parmi deux occurrences : synchronisation OU

- production "ordonnée" d'occurrences : synchronisations AVANT, APRES;

Une définition d'un type suite-temporelle (notée suitemp), pourrait d'un point de vue constructeurs être :

```
type suitemp =
    opération
        const : → suitemp
    produire élem : élément×suite → suite
ET,OU,AVANT,APRES : suite×suite → suite
fin type suitemp
```

Il est certain que le choix des opérateurs, quoique assez naturel travaillant sur des suites temporelles, est certainement guidé par le souci d'exprimer un certain nombre de synchronisation au niveau de la spécification. Nous pensons que, complété avec des opérateurs de "dérivation" définissant un cas particulier de suites que l'on appelle les suites dérivées, ce choix d'opérateurs permet une expression assez large des synchronisations dans le domaine qui nous intéresse.

On peut vouloir s'intéresser :

- à un délai écoulé depuis l'apparition d'une occurrence, ce qui donne encore une occurrence ; opération de profil

```
délai après : délai×suite → suite
```

- aux occurrences produites sur un nombre déterminé d'occurrences :


```
nbr-occurrences : entier×suite → suite
```

Pour terminer cet aspect construction d'une suite, notons que nous n'avons rien dit à propos de la sémantique des opérations, celle-ci devant être précisée par des chronogrammes ; mais qu'il faudra de toute façon reporter une part de cette sémantique au niveau des expressions d'événements en précisant les occurrences concernées par les compositions.

Dire simplement que l'on est en train de composer deux suites pour obtenir une troisième suite ne donne aucune information sur la façon dont doit se dérouler la composition.

C'est cet aspect que l'on retrouvera dans ce qu'on appellera les expressions d'événements.

III.2.2.- Utilisation d'une suite

Le second aspect d'une suite, à examiner, c'est son utilisation, c'est-à-dire la prise en compte des occurrences.

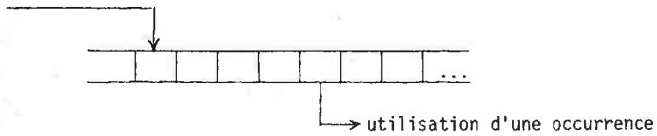
Un certain nombre d'opérations qui,somme toute, sont assez classiques peuvent être retenues :

- consultation d'une occurrence

- prélèvement d'une occurrence que l'on appelle consommation.

On peut percevoir à travers ces opérations l'aspect utilisation des événements en tant que ressources.

Schématiquement, on a pour la production-utilisation :



L'aspect dynamique de la suite apparaît également dans les opérations de consommation - consultation.

Que se passe-t-il, en effet, si on veut utiliser une occurrence non encore produite et qui n'existe par conséquent pas dans la suite ?

Dans l'interprétation classique, ce cas serait certainement traité comme élément inexistant, réponse, si l'on peut dire, "statique".

Dans le cas qui nous intéresse, il faut l'interpréter en tant qu'attente de la production future, d'une occurrence.

Ce qui veut dire que dans la sémantique de ces opérations il faudra prévoir cette attente. Sans trop vouloir rentrer dans les détails, nous pouvons considérer que cette attente fait partie de l'implantation de ces opérations.

Là encore, il faut préciser quelles sont les occurrences auxquelles on s'intéresse :

- $j^{ième}$
- toutes
- les plus récentes
- ...

D'un point de vue type abstrait on peut qualifier ces opérations de simplificateurs avec leur profil :

consulter : suite × θ → élément

consommer : suite × θ → [suite × élément]

où θ désigne l'occurrence concernée.

Nous voudrions insister, avant de terminer ce paragraphe, sur le fait que les opérations définies sur les suites temporelles sont, somme toute, assez naturelles et permettent de bien appréhender les synchronisations en termes de :

- coïncidence entre plusieurs instants : compositions de suites

- disponibilité ou non d'éléments de suites : utilisation de suites.

Cette vue des événements en termes de suites va se traduire concrètement par la définition de deux types d'événements simples et composés.

III.3.- CARACTERISATION ET DECLARATION DES EVENEMENTS

III.3.1.- Caractérisation des événements

Dans l'étude sur le concept d'événement que nous avons menée jusqu'à présent, nous avons essayé de faire apparaître trois centres d'intérêt : la production, la détection et l'utilisation des occurrences.

A travers le concept de suite, nous avons mis en évidence un certain nombre d'opérations liées à chacun de ces centres d'intérêt.

Un événement est alors caractérisé :

- par la suite de ses occurrences,
- les opérations définies sur cette suite.

Ce qui fait apparaître le concept d'événement comme un type, au sens type de donnée à inclure dans le langage.

Comme on l'a vu au chapitre II, beaucoup de ces opérations sont habituellement implicites. De par l'aspect spécification que nous voulons aborder par l'intermédiaire de ce concept, nous pensons qu'il est nécessaire de les préciser complètement, chacune par rapport au centre d'intérêt auquel elle est rattachée.

Quand on définit un événement, on devra donc s'intéresser :

- à la façon dont il est produit,
- à la façon dont il est détecté,

- à la façon dont il est utilisé.

Ces préoccupations ne sont pas forcément ainsi ordonnées. La spécification ne sera toutefois complète que lorsque toutes les caractéristiques liées à chacun de ces centres d'intérêt auront été définies.

Dans une spécification donnée, un événement est complètement défini lorsque ses aspects production, détection et utilisation auront été complètement précisés.

Il reste bien sûr à définir les opérations que nous introduisons pour la manipulation des objets de type événement.

Dans la suite de ce chapitre, nous nous intéresserons essentiellement aux opérations liées à l'utilisation des occurrences d'événements dans les expressions d'événements.

Le chapitre IV sera consacré à l'utilisation des événements par les traitements.

Dans les deux cas, cependant, nous serons amenés à manipuler des objets de type événement pour lesquels il est donc nécessaire de préciser la déclaration.

III.3.2.- Déclaration d'événements

La déclaration d'un objet de type événement a pour but d'associer un identificateur unique à la suite d'occurrences définissant l'événement. Toutes les opérations que l'on définira sur le type événement pourront donc avoir comme argument cet identificateur.

D'un point de vue type abstrait et en faisant le parallèle avec le paragraphe précédent, la déclaration d'un objet du type est représentée par l'opération const qui permet de construire un objet du type.

En remplaçant la définition du type suitemp par cette équivalente du type événement, on pourrait, en empruntant la syntaxe

de PASCAL, déclarer des objets événement de la façon suivante :

e_1, e_2 : événement

où e_1, e_2 sont des identificateurs d'événement, donc des suites temporelles d'occurrences.

Pratiquement, selon la terminologie du chapitre 2, nous introduisons deux types d'événements, simples et composés, pour lesquelles, et pour des commodités purement syntaxiques, nous serons amenés à distinguer les déclarations.

III.3.2.1.- Déclaration d'événement simple

D'un point de vue syntaxique nous avons :

<déclaration d'événement simple>::=
événement <ident>:élémentaire,(<partage>)
[,(dup, id-événement>)]
<partage>::= PART|NOPART

Ce paramètre indique simplement si un événement est déclaré partageable ou non. Il sert uniquement à limiter l'usage de l'événement. Le partage en lui-même est un problème d'utilisation et sera détaillé dans le chapitre IV.

Le paramètre dup indique si l'événement est dupliqué ou non et sera examiné plus en détail au § IV.3.1.

III.3.2.2.- Déclaration d'événement composé

D'un point de vue syntaxique nous avons :

<déclaration d'événement composé>::=
événement<ident>: composé (<expression d'événements>),(<partage>)
<expression d'événement>::=<facteur><opérateur diadique><facteur>
<facteur>opérateur monadique><facteur>

facteur représentant un événement simple.

La différence avec la déclaration d'un événement simple réside dans l'introduction de l'expression d'événement. On remarque qu'en fait, l'expression sert à indiquer comment est fabriquée la suite composée. Elle indique, de ce fait, la source de l'événement composé et est donc liée à l'aspect détection.

On devrait retrouver normalement dans les expressions d'événement les opérations liées à la fabrication des suites.

Comme déjà stipulé dans l'introduction de ce chapitre, il est donc nécessaire, pour les expressions, d'événements de préciser :

- les occurrences concernées par l'évaluation. Cela sera fait par l'intermédiaire de qualificatifs d'occurrences.
- les opérateurs utilisés dans les expressions, découlant des opérations sur les suites et dont la sémantique sera donnée sous la forme de chronogrammes.

Les qualificatifs d'occurrences tiennent également de l'aspect "suite" des événements et découlent en fait des opérations d'accès aux éléments de suites.

La syntaxe de facteur sera alors :

<facteur>::=[<qualificatif d'occurrence>]<événement simple>

Pour compléter la définition des événements composés, une étude des expressions d'événements est nécessaire.

Dans la suite de ce chapitre on trouvera :

- les choix concernant les qualificatifs,
- les choix concernant les opérateurs,
- une étude de la sémantique des opérateurs.

A chaque fois, nous essayerons de donner :

- la syntaxe complète,
- la syntaxe simplifiée quand la sémantique le permet.

Dans l'étude des expressions et uniquement pour des raisons de clarté, on considèrera des expressions avec des événements simples (cf. syntaxe de facteur). Un parenthesage explicite revenant à une définition séparée de chacun des événements contribue à lever toutes les ambigüités et évite le choix de priorités entre les opérateurs.

Liée à la définition des expressions d'événements, se trouve également posée la question de l'évaluation de ces expressions.

Les événements composés supposent en effet :

- d'une part la définition de ce que l'on a appelé les expressions d'événement,
- mais également l'existence d'un évaluateur d'expressions d'événement pour lequel nous faisons deux hypothèses :

a) l'évaluateur est suffisamment performant pour autoriser une prise en compte de toutes les occurrences, sans perte, au cours de l'évaluation, ce qui nous semble fondamental.

b) L'utilisation des occurrences au niveau des traitements (en particulier en cas de consommation) ne doit pas influer sur l'évaluation d'une expression d'événement, car cette dernière doit avant tout, dans son évaluation, rendre compte d'une situation qui a été vraie à un moment donné = les synchronisations représentées par les expressions d'événements concernent des instants purs résultant de la concordance d'autres instants, alors qu'au cours de l'utilisation, on s'intéresse beaucoup plus à ce que représente une occurrence d'événement : arrivée d'une pièce sur une chaîne d'usinage...

Notons que la détection des occurrences d'un événement composé est faite par l'évaluateur des expressions.

Remarques

1) Au chapitre II, dans l'étude générale du concept d'évenement, nous avons fait apparaître un certain nombre de caractéristiques liées à ce concept : priorités de détection et de traitement, mémorisation, délai de prise en compte.

Notons qu'aucune de ces caractéristiques n'apparaît dans la déclaration d'un événement si ce n'est son type simple ou composé.

Cela est dû en particulier au fait que ces caractéristiques ne font partie de la nature propre d'un événement mais sont plus liées soit à la détection (priorité de détection), soit à l'utilisation (mémorisation, délai de garde, priorité de traitement) et apparaîtront au cours de l'étude de ces aspects-là.

2) Nous nous intéressons à la spécification des synchronisations en termes d'événements indépendamment de toute implantation. Bien sûr, il faudra lors de cette dernière étape respecter les spécifications précédemment données.

Pour terminer ces définitions générales sur ce que l'on appelle des événements composés, nous donnons un exemple tiré de l'exemple-test du groupe EWICS-TC 11 énoncé au chapitre I.

Exemple :

Dans cet exemple, une phase du problème consistait à assurer le mixage de deux produits A et B pour assurer la fabrication d'un produit C.

Ce fonctionnement peut être spécifié en termes de :

- un événement A pour la présence du produit A dans le mixer
- un événement B pour la présence du produit B dans le mixer
- un événement C pour indiquer la présence à la fois de A et de B et qui assure la spécification des synchronisations pour le fonctionnement du mixer C : A et B.

En même temps, on remarque qu'une telle expression présente une ambigüité dans la mesure où, dans le temps, il y a plusieurs occurrences successives de A et plusieurs occurrences successives de B. Il faut donc préciser comment on interprête l'expression A et B en spécifiant les occurrences concernées.

III.4.- LES EXPRESSIONS D'EVENEMENTS

III.4.1.- Choix des opérateurs

Plusieurs facteurs ont contribués à ce choix :

- l'étude des systèmes existant proposant déjà cette notion d'expression d'événements,
- le rapprochement des événements du concept de suite,
- quelques suggestions à propos d'expression de la synchronisation par le biais d'expressions de ce type que l'on peut trouver dans [MID 77],
- les objectifs que l'on s'est fixé : expression globale et la plus large possible des synchronisations,

Nous donnons ci-après les opérateurs retenus. Nous pensons qu'ils permettent d'exprimer un grand nombre de synchronisations sans prétendre que nous sommes exhaustifs. A ce sujet il ne serait peut-être pas inintéressant d'introduire un mécanisme de définition de nouveaux opérateurs avec la sémantique associée.

Ces opérateurs se subdivisent en deux classes :

- les opérateurs monadiques :
  - <nombre> d'occurrences de <evt>
  - <délai> après <evt>

- les opérateurs diadiques :
  - "logiques" : ET, OU
  - "temporels" : AVANT, APRES

Dans la sémantique que nous prenons, l'application de ces opérateurs à des événements a pour effet de produire d'autres événements.

A ce sujet et contrairement à certains auteurs, notons l'absence de l'opérateur NON et sur lequel nous reviendrons par la suite (III.5.5.).

De plus, sans vouloir démontrer la minimalité de l'ensemble des opérateurs, il est possible de définir d'autres opérateurs à partir de cet ensemble : ENTRE à partir du AVANT et APRES par exemple.

III.4.2.- Evaluation d'expressions d'événements - Qualificatifs d'occurrences

III.4.2.1.- Evaluation .

Rappelons que nous condérons l'expression attachée à un événement composé comme définissant sa source et que l'évaluation de cette expression revient simplement à la détection des occurrences de l'événement composé.

Tout comme pour la source d'un événement simple, il se pose le problème de savoir à quel moment est faite l'évaluation : périodiquement ou non ? Si elle est faite périodiquement qui définit cette période et sur quelle base ?

Pour notre part, nous avons préféré opter pour une évaluation maximale. Par évaluation maximale, nous entendons une évaluation sur production de chacune des occurrences des événements simples de l'expression (on les appellera événements composants).

Cela suppose que l'on doit disposer d'un évaluateur suffisamment performant pour prendre en compte toutes les occurrences et cela a pour principal avantage d'éviter la perte d'occurrences au

moment de l'évaluation.

La raison de la perte d'occurrences nous paraît suffisamment forte pour justifier ce choix mais on peut toujours prévoir dans un système donné plusieurs types d'évaluation et reporter ce choix au niveau du concepteur d'une application.

De plus, l'évaluation maximale est imposée pour ne pas perdre a priori d'occurrence au cours de l'évaluation. On pourra accepter, cependant d'en "oublier", sur spécification explicite de l'application. C'est le rôle des qualificatifs d'occurrences qui permettent de préciser quelles sont les occurrences concernées par l'évaluation.

Pour terminer, on considère un instant initial t_o, qu'on appelle instant de démarrage de l'application et défini par le fait qu'il faut un arrêt complet de l'application pour pouvoir redéfinir un instant initial ultérieur et qui représente la référence de temps absolue pour toute évaluation. Autrement dit et pour reprendre la terminologie de LADET [LAD 82], toutes les occurrences considérées sont futures par rapport à cet instant. Si le procédé est partitionné en "sous-procédés", il peut y avoir plusieurs instants initiaux relatifs à ces partitions.

III.4.2.2.- Qualificatifs d'occurrences : choix et simplifications

De la même façon que pour le choix des opérateurs pour les expressions d'événements, le choix des qualificatifs d'occurrences s'inspire directement de la notion de suite : cela revient en effet, à préciser des fonctions d'accès à des éléments de suites :

- spécifier que l'on désire faire participer toutes les occurrences à une évaluation : qualificatif chaque.
- spécifier que l'on s'intéresse seulement à des éléments particuliers :

- <$j^{ième}$> occurrence
- récente occurrence par rapport à l'instant d'évaluation (ce qui suppose l'oubli volontaire d'occurrences au cours de l'évaluation : qualificatif récent.

Nous nous situons à une étape de spécification. Cela veut dire que l'on ne sait pas a priori comment sera exactement implantée une telle expression (ni par conséquent la structure exacte de l'évaluateur : une tâche, plusieurs tâches, réparti ...). On sait par contre que toute implantation doit respecter la spécification d'un point de vue sémantique des expressions. En principe, donc, chaque expression doit exprimer des synchronisations liées à la nature du problème uniquement en termes des occurrences mises en cause. Il n'empêche qu'un qualificatif comme chaque fait immédiatement penser à une mémorisation.

Cependant, tout comme vis-à-vis de l'utilisation des occurrences, et l'évaluation en est une, il nous a semblé normal de considérer que le problème de mémorisation est un problème d'implantation.

Dans ce contexte, l'évaluation qui est une "tâche" utilisatrice des occurrences d'événements au même titre que les autres tâches, possède également sa propre vue de l'événement. Cela se voit également dans le fait qu'un événement composé peut faire partie de plusieurs expressions.

On peut, toutefois remarquer qu'étant donné le processus d'évaluation "continue" choisi (évaluation sur toutes les occurrences) :

- les occurrences d'un des deux événements composant une expression seront forcément traitées en tant que récentes occurrences, l'instant d'évaluation correspondant à l'instant de détection.

- Selon l'opérateur traité et les occurrences des événements composant une expression en présence, l'évaluation peut être positive ou négative. On dira qu'elle est positive si elle permet de conclure - c'est-à-dire si, étant donné la sémantique de l'opérateur considéré, on peut à l'instant de l'évaluation, dire s'il y a ou non occurrence de l'événement composé - et évaluation négative dans le cas contraire.

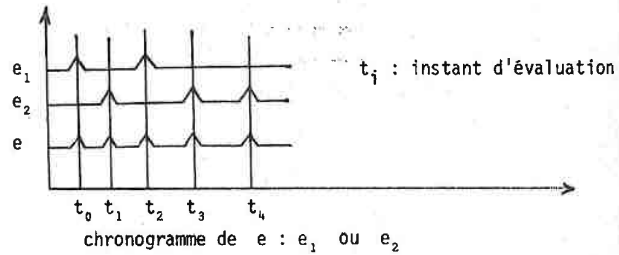
L'idée du qualificatif chaque étant de faire participer toutes les occurrences d'un événement à une évaluation donnée, il sera nécessaire de conserver les occurrences d'un tel événement lors d'évaluations négatives. Ceci veut dire aussi que si on est sûr que l'évaluation ne peut être que positive, alors cette conservation est inutile.

Ceci nous donne les simplifications suivantes :

Cas 1 : Equivalence des qualificatifs chaque et récent

Si, pour une expression donnée, l'évaluation est toujours positive pour toutes les occurrences d'un des événements composants, alors il n'y a plus lieu de préciser s'il s'agit de chaque occurrence ou seulement de la récente. Cela ne nécessite aucune conservation d'occurrences et il est clair que chaque est implicitement considéré. D'un point de vue syntaxique il est inutile alors de préciser le qualificatif et d'un point de vue implantation il est inutile de prévoir une mémorisation par exemple.

Ceci est typiquement le cas de l'opérateur OU :



chronogramme de e : e_1 ou e_2

Cas 2 : Distinction entre qualificatifs "chaque" et "récent"

Si, pour une expression donnée, son évaluation peut être négative, alors l'accumulation d'occurrences est possible.
(C'est le cas typique du ET comme nous le verrons).

Les qualificatifs chaque et récent ne peuvent être confondus dans ce cas et doivent être clairement explicités.

Le qualificatif chaque spécifie que toutes les occurrences d'événement accumulées doivent être prises en compte à l'évaluation.

Le qualificatif récent spécifie que seule une occurrence (considérée comme la dernière) doit être prise en compte : tout se passe dans ce cas comme si une seule date, celle de la dernière occurrence était conservée.

III.4.2.3.- Consommation - Consultation d'occurrences à l'évaluation

Si les qualificatifs d'occurrences servent à préciser les occurrences participant à l'évaluation, il reste encore à préciser ce que l'on fait des occurrences après évaluation et en particulier si elles peuvent ou non être réutilisées au cours de l'évaluation.

Cela contribue à renforcer le sens de l'expression en permettant d'établir des relations du type 1-1, 1-n, n-1 entre les occurrences d'événements composants une expression.

Cette notion est introduite par l'intermédiaire d'un attribut de consommation, noté :

- avec cons pour exprimer le fait qu'une occurrence est perdue, après évaluation positive, pour l'évaluation.
- sans cons, qu'on appelle encore consultation, pour exprimer le fait qu'une occurrence peut être réutilisée au cours de l'évaluation.

Il en résulte que, lorsque l'évaluation ne peut être que positive, la consommation est implicite.

La consultation par ailleurs peut engendrer, à des moments déterminés, des évaluations toujours positives, ce qui peut être une source de simplifications d'implantation.

Avant d'aborder une étude par cas des expressions proposées, notons que :

- la consommation dont il s'agit ici est une consommation à l'évaluation qui n'a aucune incidence sur l'utilisation des occurrences des événements composants, pas plus d'ailleurs que la consommation au niveau utilisation n'a de répercussions sur l'évaluation car exprimant des règles de partage d'événements entre les tâches , (tout se passe comme si les occurrences étaient systématiquement dupliquées pour l'évaluation).

Ceci va dans le sens de la séparation entre l'évaluation (qui tient plus de la détection) et l'utilisation des occurrences.

III.5.- EXPRESSIONS D'EVENEMENTS : ETUDE PAR CAS

Pour chacune des expressions, nous donnerons une expression

complète suivi des simplifications possibles. L'expression complète sert en quelque sorte de référence alors que les simplifications détermineront la syntaxe définitive.

La sémantique de quelques cas qui nous semblent intéressants sera expliquée à l'aide de chronogrammes. L'étude des cas restants est reportée dans l'annexe II.

III.5.1.- Cas du AVANT

a) Expression complète

{chaque / récent} <evt 1> {avec / sans} cons {AVANT-A / AVANT-S} {chaque / récent} <evt 2> {avec / sans} cons

Nous appelons evt 1, événement comparé (noté EC) et

evt 2, borne (noté B).

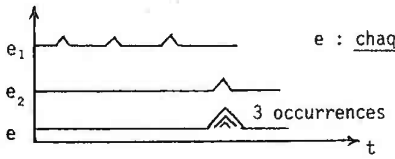
Cet opérateur permet de comparer les instants de détection des occurrences. Il y a production d'une occurrence de l'évenement composé à chaque fois que la condition occurrence i avant, occurrence j est vraie, si occurrence i et occurrence j sont des occurrences déterminées de EC et B.

Il y a alors deux façons de vérifier cette condition :

1.- ou bien attendre que les occurrences de EC et B soient en présence pour décider : c'est le cas dit sans anticipation représenté par AVANT-S.

2.- ou bien anticiper sur la production des occurrences et c'est le cas avec anticipation représenté par AVANT-A.

Le cas 1 peut s'exprimer par



e : chaque e_1 avant chaque e_2

et autorise donc des productions simultanées d'occurrences.

Dans le cas de prise en compte fugitive d'un tel événement, il est évident que seule une occurrence sera traitée. Dans le cas de prise en compte de toutes les occurrences, les 3 occurrences seront traitées.

Une telle expression permet, en outre, d'exprimer facilement le "passé" tel que défini par LADET [LAD 82]. :

L'expression E suivante dans STR/MP [LAD 82]:

```
        Processus P
              { }
              { } P1
(t)⇝  Pour chaque evt 1 passé faire T

        fin
```

où t représente l'instant d'exécution de la structure Pour, peut être traduite par une expression du type :

e : chaque evt 1 AVANT-S chaque b
avec cons avec cons

où b représente les occurrences d'un événement lié par exemple à la fin d'exécution d'une tâche représentant la partie P1.

De cette façon aucune exécution n'est autorisée avant l'arrivée d'une occurrence de b et c'est bien ce qui se passe dans l'expression e.

Dans le cas 2, le problème lié à la prise en compte fugitive

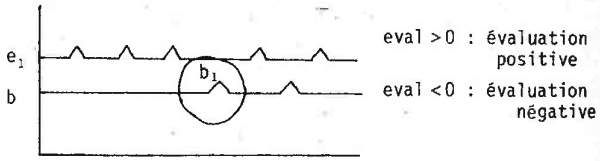
disparaît mais peut imposer à l'implantation (- si les événements composants une expression peuvent être répartis), le choix d'algorithmes complexes et coûteux du type de celui proposé par LAMPORT [LAMP 78] pour garantir la sémantique de l'expression. On peut également, à l'implementation prendre des hypothèses simplificatrices qui consistent à considérer que les délais de transmissions sont négligeables, ce qui équivaut à une implantation centralisée (ceci est particulièrement vrai dans le domaine du contrôle de procédé industriel où prédominent les réseaux locaux).

Nous ne faisons pas de choix entre l'une ou l'autre des interprétations : les deux nous semblent intéressantes à garder et surtout complémentaires.

b) Simplifications : cas du AVANT-A

- Consultation de la borne

L'absence de consommation de la borne revient à comparer les occurrences d'un événement par rapport à une borne fixe.



b_1 n'étant pas consommée, les occurrences situées avant elle donneront des éval > 0 et celle après elle donnent des éval < 0 (borne toujours présente). De plus seule la 1ère occurrence de la borne compte.

Cela revient à une expression :

{chaque / récent} <evt> {avec / sans} cons AVANT-A<j ième><borne>

L'évaluation se faisant par anticipation et s'arrêtant logiquement à b_1, elle ne peut être que positive ce qui réduit l'expression à :

E1. <evt>AVANT-A j ième<borne>

Ce qui définit une sous-suite de evt.

- Consommation de la borne

L'évaluation se faisant par anticipation, la production d'occurrences de la borne ne risque pas de donner des évaluations positives.

L'accumulation des occurrences de la borne est donc possible.

Seules les occurrences de l'événement composé peuvent donner des évaluations positives.

Deux cas se présentent :

- ou bien les occurrences de l'événement composé sont avant une occurrence de la borne, auquel cas l'évaluation est forcément positive et il n'y a pas d'accumulation possible d'occurrences,
- ou bien les occurrences de l'événement composé sont après une ou plusieurs occurrences de la borne et auquel cas elles seront considérées au moment de leur détection et il n'y a pas d'accumulation possible.

Tout se passe comme si l'évaluation est toujours positive pour l'événement comparé mais pas pour la borne.

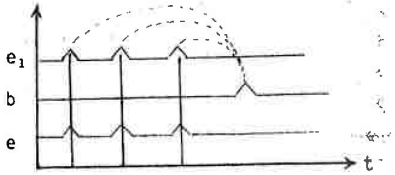
Il en résulte la simplification suivante :

E2. <evt>AVANT-A {chaque / récent}<borne> — La consommation est donc implicite

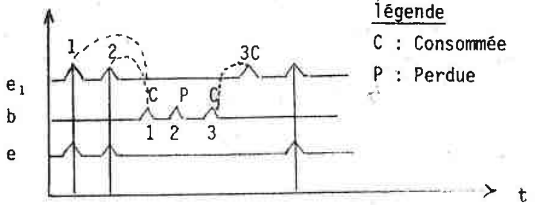
Ce qui nous donne donc deux formes d'expression du AVANT-A avec borne fixe et borne mobile.

Nous avons les chronogrammes suivants :

pour E1 e : e_1 AVANT-A 1er b



pour E2 e : e_1 AVANT-A récent b



L'occurrence b_1 est consommée aussitôt produite (à cause de l'anticipation).

L'occurrence e_{13} est consommée en même temps que b_3 dans la mesure où une conclusion (e_{13} après b_3) a pu être faite.

Cette évaluation est considérée comme positive.

L'occurrence b_2 est perdue de par le qualificatif récent.

Le cas chaque b n'est pas fondamentalement différent. b_2 ne serait pas perdue dans ce cas et pourra donc être comparée.

Remarque

L'opérateur AVANT-A est pris dans son sens strict et une occurrence de e_1 arrivée en même temps qu'une occurrence de b par

exemple est considérée comme située après cette occurrence de b.

c) Simplifications : cas du AVANT-S

- Consultation de borne

L'absence de consommation de la borne revient à comparer les occurrences d'un événement par rapport à une borne fixe.

De même que précédemment on aura une expression du type

{chaque / récent} <evt> {avec / sans} cons AVANT-S <j^ième><borne>

La différence vient du fait que l'évaluation peut être négative et l'accumulation d'occurrences, par conséquent, possible.

Il est nécessaire de pouvoir préciser explicitement les occurrences.

On aura alors deux types d'expressions :

E3	chaque <evt> AVANT-S <j^ième><borne>
E4	récent <evt> AVANT-S <j^ième><borne>

La consommation est implicite.

L'expression E4 permet la production d'une seule occurrence et n'a peut être pas beaucoup d'intérêt.

- Consommation de la borne

L'absence d'anticipation autorise des évaluations négatives.

La production d'une occurrence de la borne provoque l'évaluation de toutes les occurrences accumulées et la consommation de la borne après évaluation.

L'accumulation des occurrences de la borne est également possible.

C'est donc la syntaxe complète qui prévaut :

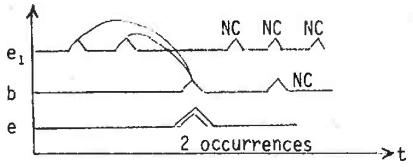
E5 | {chaque / récent} <evt> AVANT-S {chaque / récent} <borne> |

avec consommation implicite de EC et B.

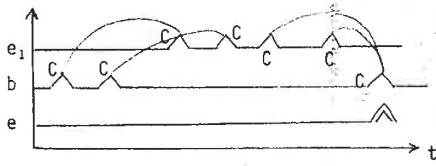
Dans ce cas également nous avons deux formes d'expression : avec borne fixe et avec borne mobile.

Quelques cas

- e : chaque e_1 avant-s 1er b

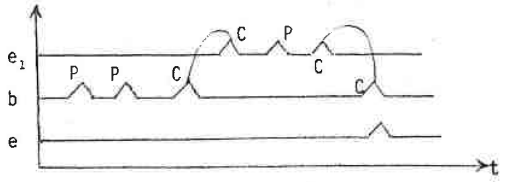


- e : chaque e_1 avant-s chaque b



Légende :
P : perdue
C : consommée
NC : non considérée

- e : récent e_1 avant-s récent b



III.5.2.- Cas du APRES

Hormi le problème de l'anticipation à l'évaluation qui n'existe pas ici, cet opérateur est complémentaire du AVANT et procède des mêmes principes : comparaisons des instants de production des occurrences pour produire les occurrences de l'événement composé. Il est normal alors que l'on retrouve les mêmes problèmes, qui seront traités de façon analogue. On pourrait le rapprocher plus du AVANT-A.

a) Expression complète

{chaque / recent} <evt1> {avec / sans} cons APRES {chaque / recent} <evt2> {avec / sans} cons

Relativement à la terminologie de LADET, cet opérateur permet de rendre compte des occurrences futures par rapport à une occurrence ou un groupe donné d'occurrences.

Il pose les mêmes problèmes que ceux cités à propos du cas 2 au § III.5.1. et exige les mêmes précautions.

b) Simplifications

A nouveau deux cas vont se présenter : consommation ou consultation de la borne.

- Consultation de la borne

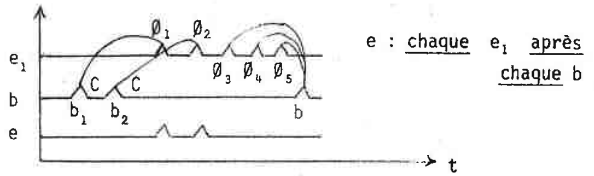
Cela revient au cas de la borne fixe et s'exprime donc par :

E6 : | <evt> après $j^{ième}$ <borne> |

- Consommation de la borne

Dans ce cas, la borne étant consommée, cela veut dire qu'on

peut avoir des situations où des occurrences de EC se trouvent effectivement après une ou plusieurs occurrences de la borne, et des situations, après consommation de la borne sur eval > 0 où des occurrences de EC sont avant une prochaine occurrence de B :



e : chaque e_1 après chaque b

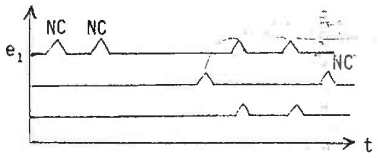
ainsi $\emptyset_1$ et $\emptyset_2$ se trouvent bien après b_1 et b_2.
mais b_1 et b_2 étant consommées, $\emptyset_3$, $\emptyset_4$ et $\emptyset_5$ se trouvent du coup avant la prochaine occurrence de B, c'est-à-dire b_3.

C'est un peu une situation symétrique du AVANT-A et les mêmes simplifications s'imposent :

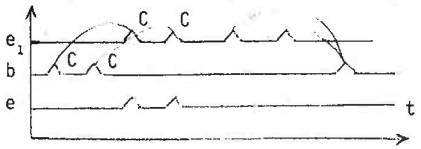
E6 : <evt> après {chaque / recent} <borne>

Quelques cas

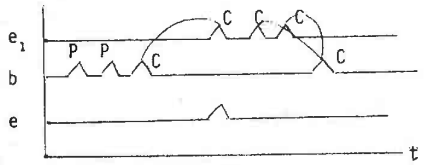
e : e_1 après 1er b



e : e_1 après chaque b



e : e_1 après recent b



III.5.3.- Cas du ET

a) Expression complète

{chaque / recent} <evt1> {avec / sans} cons ET {chaque / recent} <evt2> {avec / sans} cons

Intuitivement, cet opérateur signifie qu'une occurrence de l'événement composé est produite à chaque fois qu'il y a présence simultanée à la fois d'une occurrence de evt1 et de evt2.

Il est bien évident qu'il est nécessaire de préciser exactement ce que l'on entend par "présence simultanée" pour compléter la sémantique intuitive de cet opérateur.

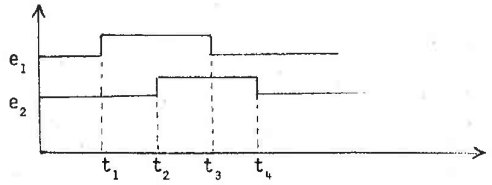
Deux cas se présentent :

i) - ou bien on s'intéresse aux occurrences non pas, par leur instant de détection mais simplement par leur présence : on dira d'une occurrence est présente pour l'évaluation, si elle a été produite et si elle n'a pas fait l'objet d'une consommation à l'évaluation depuis. Cela revient à constater la coïncidence de faits sans se préoccuper de la date des faits. C'est ce qui se passe dans les synchronisations classiques : si une tâche T doit être exécutée uniquement lorsque une tâche T1 et une tâche T2 ont toutes deux terminé leurs exécutions, il est évident que l'on ne se préoccupera pas des instants particuliers de fin d'exécution de T1 et

T2 mais du simple fait qu'elles se sont toutes deux terminées.

Cela ne va pas sans une conservation du fait que certaines occurrences se sont bien produites et fait immédiatement penser à une prise en compte mémorisée : cela se traduira par l'utilisation des qualificatifs chaque et recent.

ii) - ou bien on s'intéresse aux dates des occurrences, auquel cas cette notion de simultanéité devient beaucoup plus forte. On définira la simultanéité dans ce cas par le fait que les durées de vie intrinsèques des deux occurrences se chevauchent dans le temps :



Il faut que les deux occurrences soient donc telles que :

- ou bien $t_1 \leqslant t_2 \leqslant t_3$
- ou bien $t_2 \leqslant t_1 \leqslant t_4$

Cette interprétation est déjà peut-être un peu "temps réel".

Cela peut donner l'impression que l'on se rapproche de l'aspect physique. Il n'en demeure pas moins qu'il est essentiel de pouvoir exprimer de telles situations dès la spécification de la solution s'agissant du domaine de la conduite de procédé industriel. On introduit alors un opérateur SIM, abréviation de simultané, pour exprimer cet état de fait dès la spécification. Il va de soi qu'il faudra tenir compte de cette contrainte dans l'implantation de cet opérateur.

b) Simplifications

Il va de soi également que toutes les occurrences des événements composant une expression avec l'opérateur SIM participent à l'évaluation d'une telle expression ; dans ce cas on est toujours capable de conclure lors de l'évaluation (évaluations toujours positives donc) ce qui implique le choix implicite de l'opérateur chaque. Dans ce cas une expression se réduit à :

<evt1> SIM <evt2>

Concernant le cas i), il est clair qu'il y a possibilité de cumul pour les deux événements composant une expression.

De ce fait, aucune simplification syntaxique n'est possible : c'est l'expression complète qui prévaut.

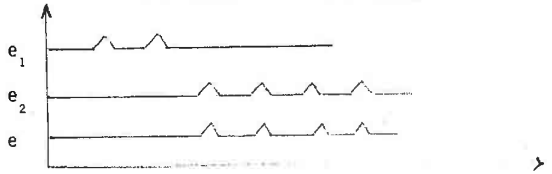
Tous les cas étudiés ne sont pas forcément intéressants pour l'expression de la synchronisation au niveau de la spécification particulièrement en l'absence de consommation (réutilisation des occurrences).

Nous n'avons, cependant pas voulu être limitatif et laissé l'expression la plus large possible même, si parfois, elle aboutit à des redondances.

Cela dit, il est intéressant, concernant l'implantation, de tenir compte de l'absence de consommation, pour la simplifier.

Ainsi, pour une expression du type

e : chaque (sans cons) e_1 et chaque (sans cons) e_2

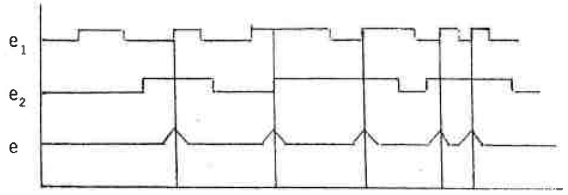


on est sûr que dès lors qu'une seule occurrence de e_1 ou e_2 est produite, l'événement e est équivalent à e_1 ou e_2.

A l'implantation, on peut alors cesser l'évaluation dès la production de cette première occurrence pour tenir compte de cette équivalence.

Quelques cas : ET "simultané "

e : e_1 sim e_2



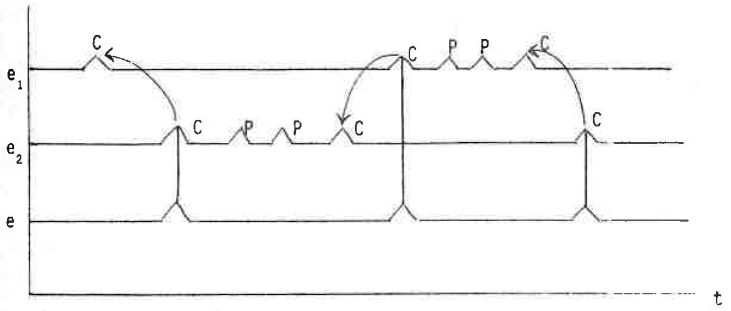
Dans ce cas, la durée de vie des occurrences intervient de manière fondamentale.

ET "classique"

Rappelons que les notations que nous utiliserons dans les chronogrammes sont les suivantes :

- $\emptyset_{ij}$ désigne une occurrence de rang j pour l'événement e_i
- la lettre P, une occurrence, perdue pour l'évaluation
- la lettre C, une occurrence consommée par l'évaluation.

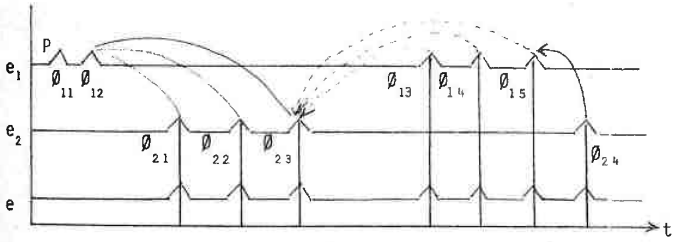
a) C : récent e_1 et récent e_2
avec cons avec sons



Les occurrences perdues sont le fait du qualificatif "recent".

L'instant de production d'une occurrence de e correspond à l'instant d'une évaluation positive de l'expression.

b) e : récent e_1 et récent e_2
sans cons sans cons

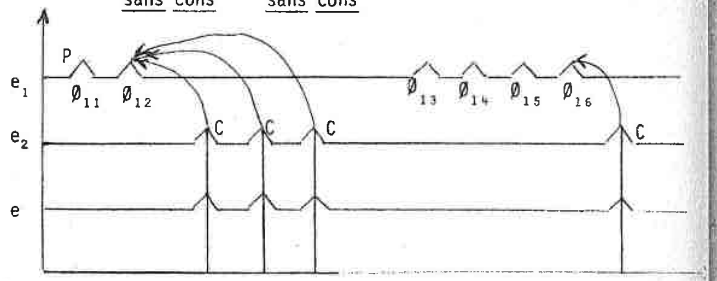


L'absence de consommation permet de faire participer une même occurrence à plusieurs évaluations : exemple de $\emptyset_{12}$.

Le qualificatif "récent" suppose l'idée de péremption des occurrences au niveau de l'évaluation.

Pour $\emptyset_{13}$, $\emptyset_{14}$ et $\emptyset_{15}$, l'occurrence présente ne peut être que $\emptyset_{23}$, pour $\emptyset_{24}$, l'occurrence présente ne peut être que $\emptyset_{15}$.

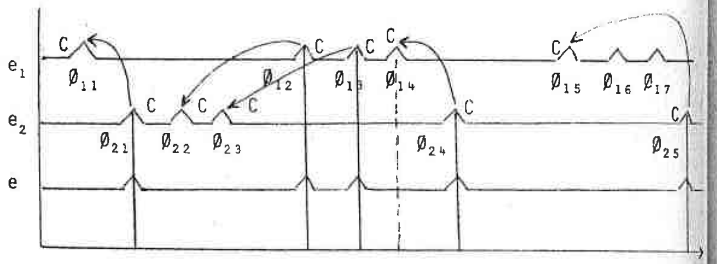
c) e : <u>récent</u> e_1 et <u>récent</u> e_2
<u>sans cons</u> <u>sans cons</u>



<u>Remarque</u>

L' attribut "sans cons" exprime le fait qu'il y a eu au moins une arrivée d'occurrence. Dans le cas c) par exemple, nous aurions eu exactement la même évaluation s'il n'y avait pas $\emptyset_{13},\ldots,\emptyset_{16}$, l'évaluation se serait faite uniquement par rapport à $\emptyset_{12}$.

d) e : <u>chaque</u> e_1 <u>et</u> <u>chaque</u> e_2
<u>avec cons</u> <u>avec cons</u>



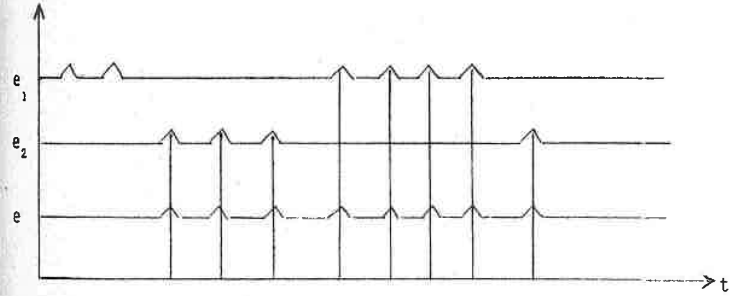
L'idée ici, est de faire participer toutes les occurrences à l'évaluation. Mais en même temps, on exprime le fait que chaque occurrence ne participe qu'à une seule évaluation.

Il faut noter que pour l'évaluation correspondant à $\emptyset_{12}$, par exemple, ce qui est requis, c'est uniquement la présence d'une occurrence au moins de e_1.

Par ailleurs, au niveau de $\emptyset_{14}$ par exemple, il y a eu évaluation dont le résultat était simplement négatif. Dans ce cas, enfin, on remarque que ce sont les occurrences de même rang qui sont mises en relation les unes avec les autres.

Cela est dû à la présence de consommation à l'évaluation.

e) e : <u>chaque</u> e_1 <u>et</u> <u>chaque</u> e_2
<u>sans cons</u> <u>sans cons</u>
avec le chronogramme suivant



Avec la sémantique adoptée, on remarque que ce cas revient exactement au même que le cas b). Dans ce cas en fait, une seule occurrence de e (la première donc) suffit pour toutes les évaluations suivantes. Les notions de "récent" et "chaque" s'estompent au profit de la présence ou non d'une occurrence, donc au profit de l'attribut de consommation au niveau de l'évaluation.

Remarque

En règle générale, seule la consommation, que ce soit au niveau de l'évaluation ou au niveau des tâches fait jouer leur plein rôle aux qualificatifs d'occurrences "récent" et "chaque". Ce qui explique, qu'en l'absence totale de consommation, certains cas ont tendance à se confondre.

f) Autres cas

Nous pensons avoir soulevé les principaux problèmes posés par l'évaluation du ET à travers les cas discutés plus haut. Il existe bien évidemment d'autres cas : mélange de "récent" et "chaque", de consommation et d'absence de consommation. Les règles d'évaluation étant définies, leur étude ne nous apporterait pas plus d'informations. Nous donnons simplement un exemple de ce que peut être une expression avec "mélange" :

$$e : \underset{\text{avec cons}}{\text{récent } e_1} \text{ et } \underset{\text{sans cons}}{\text{chaque } e_2}$$

en reportant le lecteur à l'annexe II pour plus de détails.

III.5.4.- Cas du OU

a) Expression complète

$$\{\text{chaque} \mid \text{récent}\}\langle evt_1\rangle\{\text{avec} \mid \text{sans}\}\text{cons OU}\{\text{chaque} \mid \text{récent}\}\langle evt_2\rangle\{\text{avec} \mid \text{sans}\}\text{cons}$$

Intuitivement, cet opérateur signifie que l'on produira une occurrence composée à chaque fois qu'il y a présence d'une occurrence de evt_1 ou de evt_2.

Nous remarquons que nous ne parlons pas du cas de présence

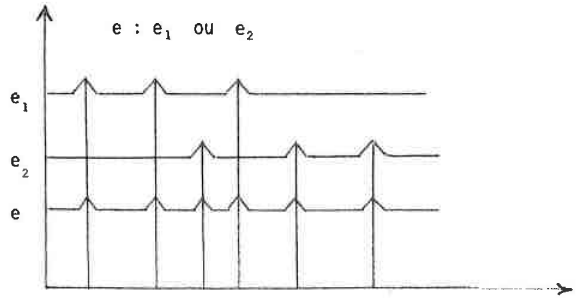
simultanée de deux occurrences. L'évaluation se faisant sur détection de chacune des occurrences des événements composant une expression, dans ce cas également il y a anticipation à la production des occurrences de l'événement composé.

b) Simplifications

Dans ce cas, étant donné la sémantique de l'opérateur OU, il ne peut y avoir que des éval > 0 pour les deux événements. L'expression se réduit à :

$$\boxed{e_1 \text{ ou } e_2}$$

L'événement composé sera défini par la suite de toutes les occurrences des événements composants ordonnées selon l'instant de leur détection.



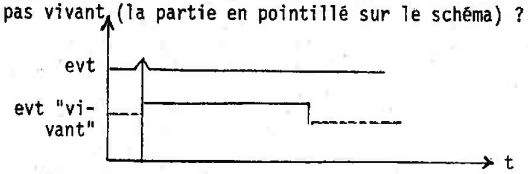
III.5.5.- Cas du NON : Evénement ou Condition ?

Tous les opérateurs que nous avons vu jusqu'à présent, lorsqu'ils sont employés dans une expression, donnent lieu à une génération d'occurrences dites composées. En fait c'est l'évaluation de l'expression qui donne lieu à cette génération d'occurrences.

Il reste cependant, un opérateur nécessaire à introduire et dont le cas n'est pas simple : le NON. La question qui se pose est la suivante : doit-on considérer une expression du type non <evt> comme un événement ou une condition ?

Si non <evt> est considéré comme un événement, il est nécessaire, alors, de se poser la question de la production d'une telle occurrence.

Rappelons que la détection d'une occurrence a toujours lieu sur un changement d'état. Cela a pour conséquence immédiate de se demander sur quel changement d'état va se baser l'évaluateur pour détecter une telle occurrence puisque, ce que l'on demande, c'est précisément la "détection" des non-changements d'états : doit-il continuellement ou périodiquement (qui définit cette période ?) produire des occurrences de non evt à chaque fois que evt n'est pas vivant (la partie en pointillé sur le schéma) ?



Pour les raisons évoquées plus haut, il nous a semblé plus logique de considérer plutôt le non <evt> comme condition. Il arrive, en effet, plus fréquemment d'attribuer la sémantique "occurrence non présente" ou "occurrence non arrivée" à une expression non evt, ce qui revient en fait à un test d'état, donc à une condition.

III.5.6.- Cas des événements dérivés

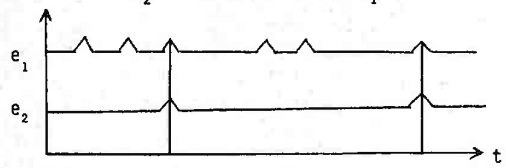
III.5.6.1.- <nombre> occurrences de <evt>

Dans ce cas, il n'y a pas lieu de prévoir les qualificatifs d'occurrence, l'évaluation étant toujours positive.

Son sens est : tous les "nombre" occurrences de <evt> dès l'instant initial.

Exemple :

e_2 : 3 occurrences de e_1



L'évaluation se faisant sur chaque occurrence, la production de e_2 se fait exactement à l'instant de toutes les 3 occurrences de e_1.

L'évaluation se ramenant à un comptage d'occurrences, il n'y a pas lieu non plus d'indiquer des attributs de consommation. Il y a en fait une espèce de consommation implicite dans la mesure où les occurrences ayant participées à un comptage sont naturellement oubliées au niveau de l'évaluation qui est forcément positive.

L'événement e est un événement dérivé pouvant à son tour faire partie d'une expression ou être utilisé au niveau d'un traitement.

III.5.6.2.- <Délai> après

On remarque que dans ce cas également l'évaluation ne peut être que positive. Il n'y a par conséquent pas lieu de préciser de qualificatif d'occurrence ni de consommation toutes les occurrences étant concernées.

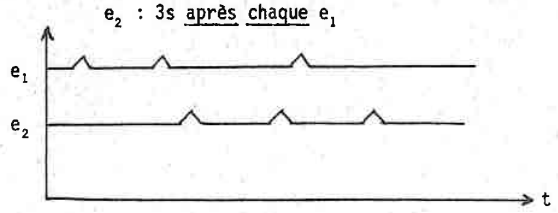
On peut avoir les expressions suivantes :

E1 :	<délai> après <evt>
E2 :	<délai> après j^{ième} <evt>

Le délai s'exprime dans une unité de temps admise.

E1 a pour sémantique la production d'un événement "décalé" dans le temps.

Exemple :



E2 autorise la production d'une seule occurrence.

Remarquons que par son aspect temporel, cet opérateur est peut-être plus orienté vers le temps réel.

III.6.- CONCLUSION

Dans ce chapitre, nous avons explicité ce que nous entendons par événement, concept à la base de toute notre démarche.

Nous situant à une étape de spécification d'une solution, nous avons souligné :

- l'aspect "vue globale" que devaient conserver les événements dans la spécification pour précisément permettre une expression des synchronisations à la fois indépendante des tâches et statique.

- l'aspect "naturel" des événements qui restent à la fois proches du cahier des charges et de l'expression d'une solution.

- et enfin, toujours dans le sens d'une vue globale et d'une expression statique, l'indépendance entre les problèmes de détection-évaluation d'événements et les problèmes d'utilisation des événements.

Le concept de suite nous a permis de mettre en évidence un certain nombre d'opérations sur les événements.

Dans le cadre de l'étude des expressions d'événements, nous

avons été amené à définir et justifier les opérations suivantes classées en deux groupes :

- opérations de composition : ET, OU, AVANT-A, AVANT-S, APRES, SIM, <délai> APRES, <nbre> OCCURRENCES DE que nous avons appelé opérateurs dans leur aspect statique de définition des synchronisations.

- opérations d'accès : CHAQUE, RECENT, <j^{ième}> que nous avons appelé qualificatifs d'occurrences.

De plus, l'étude des compositions d'évenements (voir Annexe II, également), a fait ressortir des cas de similitude qu'il se-sait intéressant d'exploiter pour tendre vers une minimalité de l'ensemble des opérateurs nécessaires et donc une implantation plus simple.

Il reste une opération liée à la détection des événements et qui sera examinée au chapitre V : l'assignation, qui permet de préciser la source d'un événement.

CHAPITRE IV :

LIENS ENTRE ÉVÉNEMENTS ET TÂCHES : UTILISATION DES ÉVÉNEMENTS

IV.1.- GENERALITES

De la même façon que, dès l'étape de spécification d'une solution à un problème posé, il est nécessaire de préciser comment sont perçues les synchronisations à travers la définition des événements, il est également nécessaire de préciser, dès l'étape de spécification, comment sont utilisés ces événements.

Rappelons que la spécification proposée, issue de SYGARE [THOM 80] se fonde sur trois concepts fondamentaux :

- les événements
- les tâches
- les liens entre événements et tâches

et qu'il s'agit dans ce chapitre de préciser justement ces liens entre événements et tâches et que nous appellerons structures d'utilisation.

Il faut bien voir, alors, que ce que nous voulons exprimer, c'est une utilisation liée à la nature même du problème posé et indépendante de toute implantation : dans l'exemple de la pompe de KRAMER, on trouve, dans l'énoncé du "cahier des charges", des phrases du type "arrêt de la pompe à chaque fois que le niveau d'eau est bas" ; on voudrait pouvoir, dès la spécification, appréhender le phénomène "niveau d'eau bas" (ce qui donne un événement) et exprimer le fait que sur chaque occurrence de cet événement, sans oubli, il faut arrêter la pompe. Il s'agit bien là, d'exprimer une utilisation des occurrences d'un événement représentant un certain phénomène, conforme au cahier des charges.

Notons que l'expression des liens entre événements et tâches constitue le coeur de la spécification et que l'on a tendance à confondre, la spécification d'une application avec la spécification de ces liens.

Ceci est un peu normal car ce n'est qu'à travers la spécification de ces liens que l'on arrive à appréhender le fonctionnement

du système global et nous faisons également cette confusion.

On demande alors à cette spécification, et c'est là à notre avis une qualité essentielle, d'être statique, c'est-à-dire qu'à l'examen d'une telle spécification, on soit capable de comprendre le fonctionnement du système sans être obligé d'imaginer en plus une structure d'exécution (ce qui rendrait la spécification dynamique).

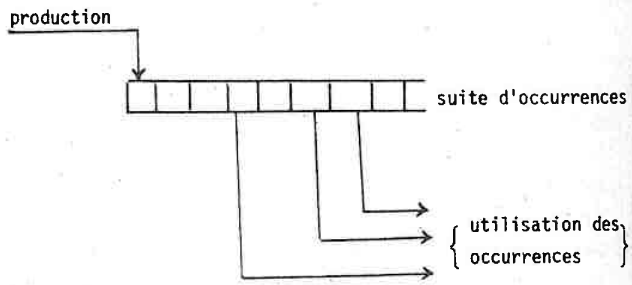
Cet aspect est très important car il confère à la spécification une grande puissance d'expression ou "expressibilité". Nous pensons que le concept d'événement apporte beaucoup à cet aspect.

De plus, la spécification se trouve liée uniquement à la nature du problème et indépendante de toute implantation : toute modification à l'implantation ne doit avoir aucune incidence sur cette spécification.

On peut alors revenir sans difficulté sur la perception des événements en termes de suites dont les éléments sont des ressources utilisables par les tâches et qui nécessite des synchronisations.

Le problème se trouve transposé donc, et il s'agit à présent de l'exprimer en termes d'utilisation d'une certaine ressource.

En rappelant le schéma :



Nous posons la question suivante : quels concepts doit-on exprimer à ce niveau, pour être sûr de bien rendre compte du fonctionnement du système ?

Là, et c'est loin d'être étonnant, on retrouve des concepts classiques :

a) - préciser à quelles occurrences on s'intéresse : accès aux éléments de la suite.

b) - préciser les règles de partage : aspect ressource.

c) - préciser ce qui se passe lorsque la prise en compte d'une occurrence est impossible pour une raison ou pour une autre : les exceptions.

d) - préciser les conditions de prise en compte des occurrences : l'état du système.

e) - préciser l'urgence relative du traitement : priorité.

On remarque que le point c) se rattache au problème de sûreté de fonctionnement que l'on exprime dès la spécification : cet aspect fait également partie de la nature du problème et en constitue même une part très importante.

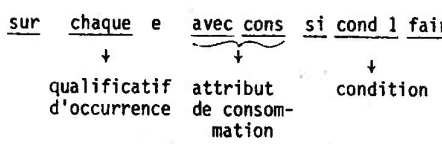
Lié à ce problème de sécurité, on retrouve également la notion de préemption, c'est-à-dire la possibilité de suspendre un traitement au profit d'un autre traitement. Si la spécification est censée "donner une idée" au fonctionnement général du système sans référence aucune à une quelconque implantation, alors il est aisé de comprendre que cet aspect doit être intégré dès la spécification = il fait aussi partie de la nature du problème.

Ce qui se passeen réalité, c'est que l'on met au même niveau

le fonctionnement que l'on peut appeler "normal" et le fonctionnement exceptionnel et c'est ainsi que nous espérons obtenir, dès l'étape de spécification, une vue complète du fonctionnement global du système. La démarche, quoique naturelle, peut paraître plus complexe car il s'agit de traiter deux problèmes à la fois. On peut néanmoins objecter que :

- elle évite les "retouches" sur la spécification du fonctionnement normal que nécessiterait une démarche séparée.
- la spécification du fonctionnement exceptionnel ne paraît plus comme étant une partie "en plus" de "la" spécification.
- le concept d'événement peut aider à une spécification complète dans la mesure om il oblige à faire un recensement exhaustif de tous les types de fonctionnement dès la spécification.

Dans le but de rendre plus claires les explications qui vont suivre nous donnons, pour terminer, l'aspect général d'une structure d'utilisation, qui est du type :



IV.2.- OCCURRENCES CONCERNEES : QUALIFICATIFS D'OCCURRENCES

Un certain nombre de synchronisations étant exprimées à travers les événements composés, il s'agit ici, en fonction du problème posé, de préciser comment sont prises en compte les occurrences.

Le problème revient alors tout comme pour l'évaluation, à donner les moyens permettant d'accéder aux éléments de la suite d'occurrences.

La question que l'on se pose est la suivante : pour un traitement donné, de quelles occurrences d'un événement donné a-t-on besoin : toutes, seulement quelques unes (et lesquelles)...?

Que peut-on choisir alors comme quelificatifs d'occurrences ?

On se rappelle ici que l'on traite des suites temporelles et on utilise l'analyse faite à propos des expressions d'événements (l'évaluation étant une utilisation particulière des occurrences d'événements).

On peut vouloir spécifier que l'on s'intéresse :

- à toutes les occurrences : qualificatif chaque
- seulement à la plus récente occurrence par rapport à la fin du traitement : récent. Dans ce cas il y a oubli volotaire d'occurrences.
- seulement à la prochaine occurrence par rapport à la fin du traitement : abscence de qualificatif.

Notons que ces qualificatifs

- s'expriment en termes d'opérations d'accès aux éléments d'une suite. A ce titre, le fait de vouloir considérer une occurrence "prochaine", et qui n'existe par conséquent encore pas entraîne une attente d'occurrences au niveau de l'implantation d'une telle opération.
- qu'ils se rapprochent de la vue classique que l'on a des événements :
  - chaque exprime l'idée de mémorisation de toutes les occurrences d'un événement pour ne pas en perdre.
  - récent exprime la mémorisation d'une seule occurrence.
  - prochain exprime l'idée d'événement fugitif [CROC 75] en insistant cependant sur le fait que cette vue en termes de mémorisation d'occurrences

n'est qu'une implantation possible d'une telle spécification et qu'en tout état de cause on ne doit pas s'en soucier au niveau de la spécification.

IV.3.- PARTAGE D'EVENEMENTS

Le partage d'un événement, pour essayer d'en donner une définition s'exprime par le fait que ses occurrences peuvent être "utilisées" par plusieurs traitements à la fois. Il ne peut s'appliquer qu'à un événement déclaré partageable.

La question qui se pose alors est de savoir ce que l'on veut exprimer dès la spécification, par le biais de ce partage.

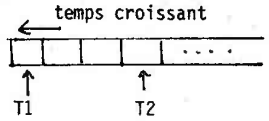
Il nous a semblé intéressant de recenser un certain nombre de situations de partage et d'examiner l'utilisation des occurrences d'un événement dans chacun des cas.

Trois cas ont retenu notre attention :

IV.3.1.- Nécessité de prendre en compte toutes les occurrences : Consultation

Dans ce cas, la nature des traitements est telle que chacune des tâches participant au partage doit prendre en compte une fois et une seule toutes les occurrences qu'elle souhaite prendre en compte.

Chacune des tâches s'exécutent à son propre rythme, elles ne prennent pas en compte nécessairement les mêmes occurrences aux mêmes moments :



Dans ce cas, il est nécessaire que les occurrences restent disponibles après utilisation par une tâche ; elles peuvent de cette façon être réutilisées par les autres tâches.

Il y a là une notion d'occurrence vue en tant que ressource non périssable.

A cette ressource qu'est l'occurrence, on adjoint une opération dite de consultation.

Consulter une occurrence, signifie pour une tâche donnée la prendre en compte pour un traitement et en même temps la laisser disponible pour les autres tâches.

Il est nécessaire d'exprimer une telle opération explicitement car dans beaucoup de cas (cf. chapitre II), l'occurrence "disparaît" implicitement dès sa prise en compte.

Vis-à-vis de chaque tâche participant au partage, il y a une notion de péremption des occurrences : une tâche ne peut traiter plusieurs fois la même occurrence.

Cette situation exprime en fait un certain parallélisme dans le déroulement des tâches : rien n'empêche les tâches de se dérouler en "même temps" pour des occurrences différentes, mais également pour une même occurrence.

Syntaxiquement, elle s'exprime par l'attribut de consommation (au niveau d'une structure d'utilisation) sans cons (sans consommation).

IV.3.2.- Nécessité de traiter une fois et une seule une occurrence : consommation

Dans ce cas la nature des traitements est telle qu'une occurrence ne doit être prise en compte qu'une fois et une seule par l'une au plus des tâches participant au partage.

Ce cas peut se retrouver par exemple lorsqu'une occurrence doit être prise en compte par un traitement parmi un ensemble de traitements équivalents.

On peut exprimer cette situation en disant que l'occurrence est consommée par une tâche.

On peut alors assimiler une occurrence à une occurrence périssable.

La consommation d'une occurrence par une tâche provoque sa non-disponibilité pour les autres tâches. De cette façon, on peut en garantir un traitement unique.

Ce cas exprime une exclusivité dans le traitement.

Le parallélisme n'est pas possible.

Syntaxique, la consommation s'exprime par l'attribut de consommation : avec cons (avec consommation).

Dans les deux cas précédents, il s'agissait, à chaque fois, pour une tâche donnée, d'exprimer ce que devenait une occurrence, après prise en compte, pour les autres tâches.

Parmi l'ensemble des tâches utilisatrices d'un événement nous avons recensé deux classes :

- une classe des tâches pour lesquelles il est nécessaire de prendre en compte toutes les occurrences souhaitées : tâches consultantes.
- une classe nécessitant un traitement exclusif : tâches consommatrices.

Il existe cependant une situation intermédiaire dans laquelle pour certaines tâches une prise en compte de toutes les occurrences est nécessaire alors que cette même prise en compte doit être exclusive pour d'autres tâches. C'est le troisième cas étudié.

IV.3.3.- Mélange des cas 3.1. et 3.2. : Duplication

Schématiquement on a :

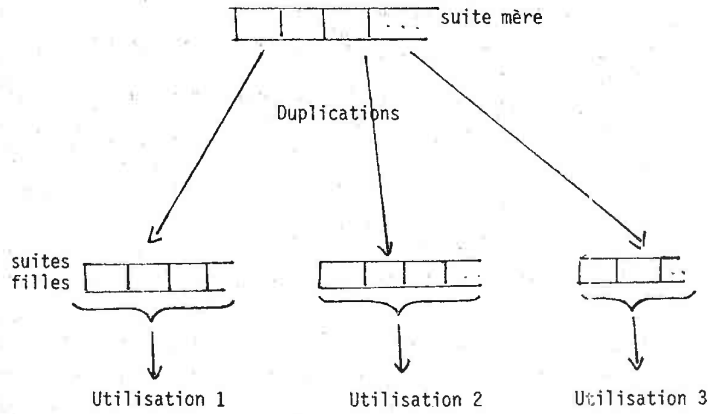
Le problème vient alors de la consommation : une occurrence consommée ne peut pas être réutilisée, ce qui est en contradiction avec la consultation.

Ce problème peut être réglé par le biais de l'opération de duplication des occurrences qui consiste en la production de suites identiques d'occurrences.

A l'utilisation, tout se passe alors comme si on avait des événements différents.

Cet aspect est intéressant car il nous permet d'exprimer les situations de parallélisme.

Tout se passe comme si, à partir d'une "suite-mère", on définit des "suites-filles" permettant des "vues parallèles" d'un même phénomène :



Ceci s'exprime simplement dans le langage par le paramètre dup intervenant dans la déclaration d'un événement élémentaire (III.3.1.).

Ce paramètre est suivi du nom de l'événement "père". Les événements dupliqués sont absolument équivalents.

Ce n'est certainement pas la seule façon d'exprimer la duplication. Elle présente l'avantage de permettre de se rendre compte, en examinant la spécification, qu'un certain nombre d'événements expriment en fait le même phénomène et conserve également l'idée de vue indépendante que chacune des tâches a des événements, base du parallélisme entre toutes les tâches de l'application.

Précisons que cette duplication est purement logique et permet uniquement d'exprimer une situation de parallélisme : à l'implantation, il n'y a pas forcément duplication physique des occurrences. Les traitements seront perçus comme étant des traitements pouvant se dérouler en parallèle (parallélisme vrai ou quasi-parallélisme selon les implantations et les contraintes imposées).

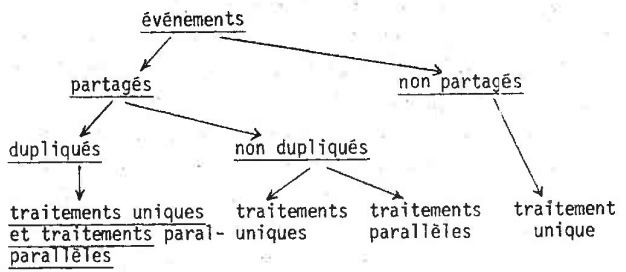
Remarques

1) Dans toutes les situations considérées, nous n'avons pas parlé d'exclusion mutuelle entre les traitements.

Celle-ci est en fait supposée traitée au niveau des modules de SYGARE (voir Annexe I) par les mécanismes propres de SYGARE.

2) La notion de partage exprimée dans ce paragraphe est basée sur une notion d'indépendance des tâches chacune se déroulant à son propre rythme. Cependant, lorsqu'il est nécessaire d'effectuer un certain nombre de traitements dans les mêmes conditions et pour les mêmes occurrences, on peut exprimer cette situation au niveau d'une seule tâche composée des modules (au sens SYGARE) représentant les traitements. De plus, de cette façon, il est possible d'exprimer des enchaînements de traitements.

IV.3.4.- Schéma récapitulatif du partage et syntaxe générale



Syntaxe générale

Sur [{chaque / récent}] <id evt> {avec / sans} cons ... faire <id tâche>

qualificatif d'occurrence — attribut de consommation

L'absence de qualificatif d'occurrence signifie que la prise en compte est fugitive.

L'attribut de consommation est obligatoire quoique inutile dans les cas où la consommation est implicite (événement non partagé).

Par abus de langage et certainement par analogie avec les notions classiques d'événement :

- aux structures d'utilisation avec chaque et récent on fera correspondre la notion de prise en compte mémorisée.
- aux structures d'utilisation sans qualificatif d'occurrence on fera correspondre la notion de prise en compte fugitive.

IV.4.- PRISE EN COMPTE DE L'ETAT DU SYSTEME : LES CONDITIONS

Lorsque dans un cahier des charges, on rencontre une phrase du type "faire tel traitement toutes les 10 mn seulement si telle vanne est fermée", cela montre clairement que non seulement il est nécessaire de tenir compte de certaines conditions dans le traitement des occurrences d'événement mais aussi que cela fait partie de la nature du problème posé et doit par conséquent être explicité dès la spécification.

C'est ce que l'on appelle prise en compte de l'état du système.

Qu'entend-on alors, au juste par système ?

C'est en fait l'ensemble procédé+système de contrôle et commande (qui est une ACPI).

L'état du système se subdivise donc en deux ensembles d'états : état du procédé, état du système de contrôle et commande.

IV.4.1.- Etat du procédé

Vis-à-vis d'une ACPI, un procédé est connu par ses entrées. C'est donc par ces entrées que l'on peut connaître l'état du système. Il faut préciser que dans notre démarche, on suppose que le procédé existe. On peut alors admettre que ses entrées sont connues de manière symbolique et non ambiguë.

On peut également faire remarquer que l'on tient déjà compte d'une partie de l'état du procédé à travers les événements composés : que signifie, par exemple evt 1 sim evt 2 si ce n'est :

sur evt 1 si evt 2 "vivant"
ou sur evt 2 si evt 1 "vivant"

evt "vivant" signifiant que la dernière occurrence de evt (la plus récente donc) est toujours présente.

D'une manière générale, l'occurrence d'un événement signifie

qu'une certaine condition s'est réalisée. Il faut bien voir aussi que les conditions qui ont déclenché une occurrence d'événement peuvent ne plus être vraies au moment du traitement. Il peut se produire donc un certain décalage temporel dans la vision de la réalité des faits pour un traitement donné.

Il nous semble cependant important de pouvoir tenir compte des états au moment même des traitements. C'est le rôle de ce que l'on appelle des conditions.

La structure d'utilisation est de la forme :

sur evt [si<condition>] faire tâche

Les conditions sont de la forme

entrée i{>,<,>,<,=} valeur (pouvant être booléenne)

On peut avoir également des expressions booléennes de conditions. Les entrées sont du type signaux stables (il nous semble difficile de considérer des entrées impulsionnelles).

De telles conditions complètent l'utilisation des événements : alors que pour les événements seuls les changements de valeur de ces entrées nous intéressaient, ici ce sont les valeurs elles-mêmes, de ces entrées qui sont importantes. Nous utilisons en fait ici le principe de dualité entre événements et conditions définis dans [THOM 80, VIT 81].

IV.4.2.- Etat du système de contrôle

Cet état est représenté par l'ensemble des éléments constitutifs d'une ACPI [cf. Annexe I] :

- les tâches
- les modules
- les données transmissibles

(comme déjà vu au paragraphe précédent, les événements représentent plus l'état du procédé).

L'état des modules étant englobé dans l'état des tâches, c'est plutôt l'état de ces dernières qui sera considéré ici.

IV.4.2.1.- Etats des tâches

Au niveau d'une spécification, ce qui peut nous intéresser en ce qui concerne les états d'une tâche, ce sont les états globaux du type :

- tâche active pour exprimer que la tâche est, ou bien en exécution, ou bien que rien se s'oppose à son exécution si ce n'est l'attente d'un processeur. Ce qui signifie, d'un point de vue spécifications, que le traitement est en cours.

- tâche inactive qui signifie que certaines conditions de scheduling de la tâche ne sont pas encore réalisées ou que la tâche a été suspendue. D'un point de vue spécification, on considère que le traitement n'est pas fait.

Cela traduit donc des phrases que l'on peut trouver dans un cahier des charges du type "ne lancer tel traitement que si le traitement j n'est pas en cours".

Syntaxiquement, l'expression est la même que précédemment avec les conditions supplémentaires

<id-tâche>{active / inactive}

IV.4.2.2.- Etats des données transmissibles

Bien plus que des conditions sur les données transmissibles elles-mêmes du type "donnée pleine" ou "donnée vide" qui sont prises en compte dans les synchronisations entre les modules, il nous a semblé important d'introduire la notion de variables d'état "consignant" l'évolution du système de contrôle.

Une variable d'état est simplement une donnée transmissible dont seul le plus récent exemplaire est considéré [cf. Annexe I].

Le récent exemplaire détermine bien la valeur la "plus" à jour de cette variable.

En tant que telle, elle est manipulable par les tâches (entrées et sorties de module [cf. Annexe I]). Les problèmes d'exclusion mutuelle sont réglés par les mécanismes de synchronisation inter-modules.

Une variable d'état est une variable interne au système de contrôle.

Une telle variable pouvant être de n'importe quel type, on peut avoir comme conditions

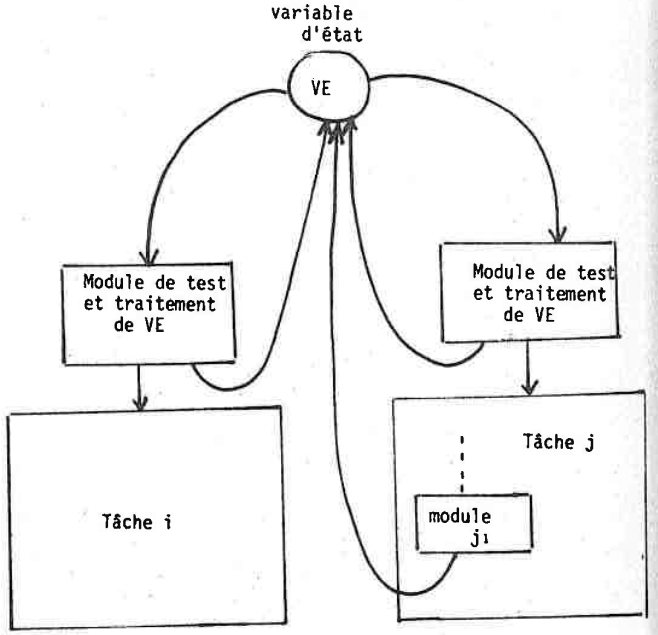
<id-donnée> { = / ≠ / < / ≤ / > / ≥ } valeur

Une telle variable contrôle l'exécution d'un certain nombre de tâches. Il nous a paru intéressant alors, de la même façon que [PUL 78], d'introduire la possibilité de modifier la valeur d'une telle donnée à l'entrée des traitements, c'est-à-dire au niveau de la structure d'utilisation :

sur<evt> si <condition> [avec <traitement>] faire tâche i

Ceci permet de garantir l'exclusivité d'un traitement sur une condition donnée à condition que l'exclusion nutuelle soit appliquée au traitement permettant de modifier la condition.

Cela peut se faire par adjonction d'un module de test et de traitement (module au sens SYGARE) à l'entrée de chaque tâche concernée par cette variable d'état :



L'exclusion mutuelle est garantie par les mécanismes de SYGARE.

Que représente alors le traitement ?

Il ne peut aller que dans le sens d'une modification par affectation de la valeur de la variable d'état.

Exemple :

Dans l'exemple de la pompe de Kramer on peut avoir :

sur chaque meth-high si état pompe = "en marche"
avec état pompe = "arrêtée" faire arrêt-pompe

meth-high est un événement représentant la détection des seuils hauts de méthane.

Remarques :

- Ce type de traitement est applicable aux variables d'états, variables, internes donc, et n'a pas de sens pour les autres conditions.
- Ce traitement revient en fait à une espèce de consommation de la condition au sens où elle n'est plus "vraie" après le traitement.

IV.4.3.- Aspects syntaxiques

D'un point de vue syntaxique, on distingue les conditions simples et les conditions composées :

- une condition simple porte sur un objet tâche, variable d'état ou entrées telles qu'elles ont été précisées dans ce chapitre.
- une condition composée est définie comme étant une expression booléenne construite à partir de conditions simples et d'opérateurs logiques ET, OU, NON.

La syntaxe de la structure d'utilisation s'exprime par :

```
sur <qualificatif><evt><attribut de >
      d'occurrence          consommation

si <condition>[avec<traitement>] faire tâche
```

IV.5.- ORDONNANCEMENT DE TÂCHES ET PRISE EN COMPTE DES OCCURRENCES

A présent que la structure d'utilisation est à peu près complète, il nous faut examiner comment peut se passer l'activation d'une tâche en fonction des occurrences et des conditions : que se passe-t-il, par exemple, lorsqu'une occurrence est disponible sans que la condition ne soit vraie ?

L'occurrence est-elle perdue ? Doit-on attendre la réalisation de la condition ?

Il va de soi qu'une occurrence ne peut être prise en compte par une tâche que si la condition est réalisée.

Pour bien comprendre le problème prenons le cas simple d'une tâche utilisant un événement non partageable de la façon suivante :

sur chaque e si cond faire T

Cela exprime :

- d'une part que la tâche T doit prendre en compte toutes les occurrences de e.
- que cette prise en compte ne peut se faire que lorsque la condition est vraie.

Il peut se produire alors que pendant une exécution de T, plusieurs occurrences de e surviennent. On a pu choisir de les mémoriser pour ne pas les perdre.

La prise en compte ne peut survenir qu'à la fin d'exécution de T.

C'est uniquement à ce moment-là que le scheduler peut essayer de relancer la tâche et c'est à ce moment-là que la condition est examinée :

- si la condition est vraie, l'occurrence est prise en compte
- si la condition est fausse :
  a) ou bien on accepte de perdre l'occurrence. Cela peut poser un problème pour la sémantique du chaque mais peut être compensé en prévoyant un traitement supplémentaire :

sur e ..... si cond faire T

sinon faire T'

Un problème subsiste, cependant : dans le cas où le traitement complémentaire n'est pas prévu, que faut-il considérer comme fin de traitement ?

En fait, dans ce cas, tout se passe comme si on avait un traitement considéré comme nul.

b) Ou bien on accepte d'attendre que la condition soit réalisée pour traiter l'occurrence.

Cette attente sera de toute façon limitée par le délai de garde et on pourra également prévoir un traitement complémentaire. Nous retrouvons dans ce cas également la dualité entre les événements et les conditions et l'implantation d'une structure avec une telle sémantique impose la définition pour chaque condition, de l'événement dual : passage-à-vrai (cond) et qui impose une détection supplémentaire.

Nous choisissons la sémantique définie en a) car cela permet d'assurer au moins une prise en compte fugitive.

De plus, complétée de la façon suivante,

sur <evt> si <cond> faire T1 [sinon faire T2]

cette structure permet de compenser cet effet de perte et en tout état de cause de le contrôler.

Nous pensons enfin que ce cas d'"attente" de condition peut s'exprimer à l'aide de l'opérateur SIM, car il s'agit bien de vérifier la coïncidence de deux événements.

IV.6.- TRAITEMENT DES EXCEPTIONS : EXPRESSION

Fonctionnement normal et fonctionnement anormal d'un procédé doivent être considérés comme faisant partie intégrante d'une spécification d'ACPI et c'est l'ensemble de ces deux fonctionnements qui représente "le" fonctionnement du système.

Partant de là, aucune différence ne doit être faite au cours de la spécification entre ces deux "parties" et les mêmes concepts peuvent et doivent être utilisés dans les deux cas.

Le concept d'événement pouvant représenter aussi bien un "phénomène" normal et attendu, qu'une panne, attentue, tend à notre avis à unifier ces spécifications de ces deux types de fonctionnement.

Comment peut-on intégrer le traitement des exceptions dès la spécification :

- en définissant des événements représentant directement des cas de panne : moteur qui s'arrête, chute de tension ...

- en spécifiant des délais de garde (ou time-out) maximisant le temps de non-prise en compte d'une occurrence.L'écoulement d'un tel délai définit lui-même un événement appelé événement d'exception auquel peut être rattaché un traitement.

Syntaxiquement on peut le préciser par

sur <evt> ..... faire<tâche>[(TØ,<evt-exception>,<délai>)]

Cette notion de délai de garde est intéressante à plus d'un titre :

- dans le passage d'une spécification à une implantation, il peut constituer un paramètre de choix décisif des structures d'exécutions. Dans ce cas, c'est un paramètre de réglage d'une application.
- au cours du déroulement d'une application, c'est un paramètre contribuant à la sûreté de fonctionnement.
- constituant une échéance de traitement, il peut intervenir dans le schéduling pour "accélérer" l'activation d'une tâche à l'approche de l'échéance.

Le TØ introduit une définition supplémentaire d'un événement considéré comme élémentaire, non partageable. La déclaration de l'événement exception est faite implicitement.

Par définition d'une prise en compte fugitive, ce délai n'intervient pas dans ce cas-là.

Lié à un couple (événement, tâche), le délai exprime en fait, comme déjà précisé, une urgence de traitement. Ainsi, par exemple, si une occurrence doit être traitée par une tâche aussitôt détectée (passage d'une pièce sur une chaîne d'usinage), cette même occurrence peut "attendre" pour un traitement moins urgent ; cela peut se traduire, par exemple, par une mémorisation pendant un temps donné maximum, de l'occurrence.

Ce délai n'a rien à voir avec la durée de vie intrinsèque d'une occurrence mais est lié aux traitements.

La prise en compte fugitive va tout-à-fait dans ce sens.

Cette façon de voir le problème explique également pourquoi on a parlé de paramètre de réglage : par simulation, on peut en effet approcher une structure d'exécution capable de supporter ces contraintes de temps, car ce délai exprime bien des contraintes de temps à respecter.

Remarque

Conceptuellement, un délai de garde est associé à chaque occurrence produite lorsque la demande en est faite.

Pratiquement, il est difficile d'attacher un "réveil" à chaque occurrence produite.

Cette remarque fait apparaître le rôle des time-out peut-être plus comme paramètre de réglage d'une application en cours de mise au point et met en évidence la nécessité d'un outil de simulation pour la vérification du respect des spécifications avant une implantation définitive.

On peut alors espérer que les situations d'exception de-

viennent moins fréquentes mais qu'il est néanmoins nécessaire de prévoir avec moins de contraintes d'implantation (un seul réveil pourrait alors suffire).

IV.7.- LA PRIORITE DE TRAITEMENT

Tout comme pour les délais de garde, la priorité est une notion liée à un couple (événement, tâche).

Au cours d'une spécification, elle sert à indiquer l'importance relative des traitements.

Il est bien évident que l'on peut "saisir" cet aspect urgence de traitement dès la spécification.

Elle rejoint les notions classiques de priorités avec l'idée que, pour une implantation donnée, à chaque fois que les moyens d'exécution ne sont pas suffisants, c'est le traitement le plus prioritaire (le plus urgent donc) qui passe.

Elle est définie par un entier qui exprime une priorité croissante, lorsqu'il évolue dans le sens décroissant (0 représente la plus forte priorité).

D'un point de vue syntaxique elle s'exprime par :

sur <evt>.... faire<tâche> avec priorité = <entier>

IV.8.- AUTRES STRUCTURES DE CONTRÔLE

Au chapitre I, nous avons montré la nécessité d'un contrôle explicite du déroulement des activités (tâches) qui :

- d'un point de vue programmation, permet de disposer de primitive de contrôle de l'activité des tâches (scheduling).
- d'un point spécification permet d'énoncer les règles de contrôle que l'on désire appliquer.

Nous avons examiné jusqu'à présent les structures d'utilisation concernant le déclenchement des traitements.

Dans ce même ordre d'idée, nous avons également souligné l'importance d'une expression explicite de la préemption.

Cela peut s'exprimer explicitement par l'intermédiaire des priorités à condition d'attacher à ces priorités une sémantique qui suppose la préemption. Dans ce cas, l'exécutif sous-tendant le langage doit comporter également un mécanisme de pré-emption.

Pour compléter cet aspect, il est nécessaire de pouvoir exprimer les possibilités de suspension et reprise et arrêt de traitements selon des conditions fixées à l'avance.

Pour des raisons de sécurité, nous devons en effet disposer d'un moyen puissant de suspension des traitements.

Cela peut s'exprimer simplement en introduisant des structures, dites de suspension :

sur <qualificatif><evt>{tuer / suspendre}<tâche>

Notons l'absence d'attributs de consommation car il ne s'agit pas dans ce cas de "traiter" une occurrence mais de réagir à une situation exprimée par un événement.

De plus elle est inconditionnelle étant donné le caractère d'urgence qu'elle représente.

Elle agit de la façon suivante :

- l'état "tâche inactive" est précisé par découpage en deux sous-états : "suspendue" et "en attente" (d'occurrences).
- si la tâche était "en attente" ou "active", elle est suspendue immédiatement et passe dans l'état "suspendue" (suspendre) ou "inexistante" (tuer)

- si la tâche était déjà suspendue, il ne passe rien (suspendre) ou la tâche passe dans l'état "inexistant" (tuer).

L'utilisation conjointe d'une structure permettant d'activer une tâche et d'une structure de suspension permet d'exprimer une forme de pré-emption lorsque les mêmes occurrences des mêmes événements sont utilisées :

e_1 → sur chaque e_1 .... faire T1
e_1 → sur chaque e_1 .... suspendre T2

L'exécution d'une tâche tuée ne peut plus être reprise étant donné l'aspect statique des structures d'utilisation.

Ce cas s'applique donc aux situations jugées graves.

Il est nécessaire, pour terminer, de compléter la structure de suspension par une structure de reprise :

sur <qualificatif><evt> reprendre <tâche>

La reprise exprime le passage de la tâche de l'état "suspendue" à l'état "en attente" sauf si elle y était déjà ou qu'elle était active. Les occurrences arrivant entre l'instant de suspension et l'instant de reprise ne sont pas perdues mais peuvent provoquer des exceptions si un délai de garde est spécifié.

IV.9.- EVENEMENTS ET COMMUNICATIONS : QUELQUES COMPARAISONS

Au cours du chapitre I, nous avons essayé de mettre en évidence l'aspect complémentarité des événements et des communications dans une spécification d'ACPI. L'étude des événements aux chapitres III et IV sous l'aspect production et utilisation montre également que l'on peut trouver des aspects similaires entre les événements et les communications (occurrences concernées,

duplication ...).

C'est la raison pour laquelle, il nous a semblé intéressant de revenir sur cet aspect et voir en quoi événements et communications sont semblables et en quoi ils sont différents.

Une tentative de classification des différents types de communication a été faite dans [DPT 83] et un certain nombre d'opérateurs de composition de messages ont été introduits dans [DER & al 83 (2)].

Les comparaisons qui vont suivre seront donc axées sur ces études, qui à notre avis, sont représentatives de l'état de l'art.

IV.9.1.- Classification des communications [DER & al 83(2)], [DPT 83]

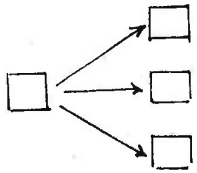
Parmi les critères pouvant caractériser une communication, introduits dans [DPT 83] nous trouvons :

- le protocole d'échange qui définit les synchronisations : échange synchrone ou asynchrone avec ou sans réponse.
- la sémantique de l'échange qui définit le comportement des entités à partir des données échangées :
  - échange déterministe : il existe une relation connue entre les données produites et les données utilisées.
  - échange indéterministe : il n'existe aucune relation connue et définie entre les données produites et les données utilisées.
- le type de liaisons : bipoint ou multipoint.
- le type de désignation des partenaires.

Dans [DER & al 83(2)], nous avons surtout noté, conjointement à l'utilisation de certains types de liaisons, la définition d'un certain nombre d'opérateurs :

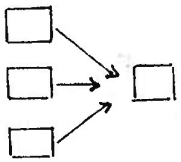
- opérateur de diffusion : & qui permet la diffusion d'un

message à partir d'une entité vers plusieurs entités



(structure divergente)

- opérateur de sélection : | (ou exclusif) qui permet la sélection d'un message et un seul par une entité qui en reçoit plusieurs.



(structure convergente)

- L'opérateur d'entrelacement : , qui est un ou inclusif.

IV.9.2.- Nos événements

Un événement étant défini comme une suite temporelle d'occurrences, nous avons introduit, pour des besoins de spécification :

- un certain nombre de relations :
  1) Entre les occurrences d'événements : relations logiques (ET, OU, SIM) ou temporelles (AVANT, APRES). C'est de que nous avons appelé les expressions d'événements.
  2) Entre les entités qui produisent les occurrences d'événement et les entités qui utilisent ces occurrences :
     - partage ou non
     - diffusion
     - consultation, consommation

- flots d'occurrences utilisées : chaque, récent, fugitif, $j^{ième}$.

L'absence ou présence de partage introduisent des possibilités de liaisons bipoint ou multipoint entre les entités productrices (procédé) et les entités utilisatrices.

IV.9.3.- Comparaisons : similitudes et différences

a) Similitudes

Les similitudes que l'on peut relever entre les événements et la communication interviennent dès que l'on introduit une distinction entre les occurrences d'un événement qui rappellent donc les occurrences d'un message. Les mêmes concepts interviennent naturellement dans les deux cas particulièrement en ce qui concerne l'utilisation des occurrences :

- les occurrences sont-elles partagées ou non, dupliquées ou non : liaison bi-point, multipoint.
- y-a-t-il attente de producteur ou non : échange synchrone ou asynchrone. Dans notre cas il n'y a pas attente du producteur : l'échange est donc asynchrone.
- les occurrences sont-elles utilisées toutes, dans un ordre précis, quelques-unes seulement : échange déterministe ou non déterministe.

b) Différences

Si à l'utilisation, on peut relever des similitudes entre le concept d'événement et le concept de communication, les différences apparaissent en revanche lorsqu'on examine la production des occurrences. Dans ce cas en effet, l'on est amené à préciser la nature des occurrences produites.

On constate alors que l'introduction de certains opérateurs de composition d'occurrences s'appliquent aux événements tels que nous les définissons et sont difficilement généralisables à tous les types de messages : nous pensons notamment aux opérateurs SIM, AVANT, APRES. Peut-être ces opérateurs pourraient-ils être définis sur des messages particuliers.

Cela veut dire en particulier que nous nous trouvons en présence d'objets qui sont de nature différente.

Ceci est encore plus vrai lorsqu'on considère la prise en compte du temps : un instant particulier est rarement perçu comme message à transmettre mais plus comme entité complète en soi et que l'on doit utiliser en tant que telle. Il peut en être de même pour tous les événements qui ne sont que des formes de représentation du temps.

Enfin, un événement comporte toujours une sémantique implicite. Ce qui n'est pas le cas d'un message qui doit être souvent interprété.

En conclusion, nous pouvons dire que les événements, même si par certains côtés se rapprochent du concept de communication, représentent quand même des objets à part entière, dans un contexte temps réel, avec lesquels on peut exprimer, comme on l'a montré avec les expressions d'événement, des synchronisations qu'il n'est pas toujours facile de réaliser avec des messages. Il est certain également que la communication est nécessaire dans un contexte temps réel et, comme on l'a déjà souligné au chapitre I, événements et communications apparaissent comme complémentaires pour le domaine concerné.

IV.10.- CONCLUSION

Dans ce chapitre, nous nous sommes efforcés de préciser l'aspect utilisation des événements. Ceci a donné lieu à la définition de ce que l'on a appelé structure d'utilisation et dont la syntaxe générale est la suivante :

```
sur <qualificatif occurrence> <id-evt> <attribut consommation> si <condition> [avec <traitement>]
                    faire <id-tâche> avec priorité=<entier>
                                        [(TØ,<evt-exception>,<délai>)]
                                    [sinon faire <id-tâche>].
```

L'absence de TØ est équivalente à (TØ,∞) (délai infini).

De même, pour l'expression de la sécurité, des structures de suspension et reprise ont été définies :

```
sur <qualificatif occ> <id-evt> {tuer | suspendre} <id-tâche>

sur <qualificatif occ> <id-evt> reprendre <id-tâche>
```

Notons enfin qu'à travers ces structures, on retrouve beaucoup de concepts mis en évidence dans [PIKE 72], pour une proposition de standardidation des langages temps réel.

Démarrant du fait que beaucoup de ces concepts permettent de rendre compte d'une situation "temps réel", nous avons essayé de les intégrer à quelques éléments de ce que pourrait être un langage de spécification d'applications en temps réel.

CHAPITRE V :

LIENS PHYSIQUE-LOGIQUE : ASSIGNATION

V.1.- OPERATION D'ASSIGNATION

Dans ce chapitre, nous abordons la partie dans laquelle il est nécessaire de prendre en compte l'aspect matériel, donc physique, d'une application. Dans notre démarche, en effet, nous avons préconisé une séparation des deux aspects physique et logique, laissant le concepteur raisonner uniquement à un niveau logique. L'avantage d'un tel découplage est double : cacher le plus longtemps possible les détails d'implantation pour ne laisser transparaître que l'aspect logique d'un procédé d'une part, et permettre, d'autre part, des modifications des liaisons physique-logique très utiles dans la prise en compte des problèmes de marche dégradée et reprise sur panne notamment.

Cela suppose, bien entendu, que l'on dispose d'une opération permettant d'établir une telle liaison. L'étude qui suit va porter sur une telle opération dite d'assignation. Contentons-nous, pour l'instant, d'en donner une syntaxe approchée que nous compléterons par la suite :

assigner < id_ evt_simple > à < source >
sur < adresse site >

L'adresse site permet de préciser, dans un milieu réparti, le site sur lequel est implanté l'évènement simple. Avant d'aborder des problèmes inhérents à l'assignation, commençons par préciser la nature des sources.

V.2.- DIFFERENTS TYPES DE SOURCES

Comme sources nous pouvons avoir :

- des interruptions désignées par leurs adresses
- des transitions vrai - faux, faux - vrai d'entrées booléennes désignées par des adresses

- des passages au-dessus ou en-dessous d'un seuil donnè d'entrées analogiques ou numériques désignées par leurs adresses.
- le temps (heure fixe par exemple)
- l'arrivée de messages-opérateur
- la réalisation de certaines conditions sur des données transmissibles : donnée pleine, vide.
- les fins de tâches et de modules
- les débuts de tâches et de modules.

Remarquons que l'on aurait pu parfaitement intégrer les événements composés à ce niveau-là : une expression d'événements représente tout simplement un aspect physique d'un événement simple traité par une tâche. Les expressions concernant le temps (heure fixe, délai,...) sont exprimées directement dans la définition des événements composés.

D'un point de vue syntaxique, nous avons :

< source > ::=
| transition vf E B < adresse_entrée_booléenne >
| transition fv E B < adresse_entrée_booléenne >
| IT < adresse_d'interruption >
| messop = < message_opérateur >
| DT < id_donné_transmissible > vide
| DT < id_donné_transmissible > pleine
| fin de < id_tâche >
| fin de < id_module >
| début de < id_tâche >
| début de < id_module >

| {transition fv / transition vf} de EN < adresse_d'entrée_numérique >< op_de_relation ><const

| {transition fv / transition vf} de EA < adresse_d'entrée_analogique><op_de_relation ><const

< adresse d'interruption > ::= < n° d'it >
< adresse_entrée_booléenne > ::= < n° entrée >
< adresse_entrée_analogique < ::= < n° entrée >
< adresse_entrée_numérique < ::= < n° entrée >
< opérateur de relation > ::= >|<| = |¬ = |> = |< =

Le numéro d'entrée peut lui-même se décomposer, si nécessaire, en n° de module, n° de voie par exemple.

L'adresse site est importante à noter. C'est, évidemment, une adresse qui permet de désigner de manière unique un site.

Le message opérateur est constitué par le texte du message.

La désignation des tâches, modules et données transmissible est supposée être unique.

V.3.- TRAITEMENT DE L'ASSIGNATION

L'idée de l'assignation est d'indiquer à un module système chargé de la détection des occurrences d'événements où, et éventuellement comment détecter les occurrences d'événements. L'on imagine aisément qu'un tel module détecteur dispose d'une table de description de tous les événements déclarés dans le système. Cette table est complétée soit à la déclaration d'événements composés, soit à la rencontre d'opérations d'assignation. Une opération d'assignation s'adresserait donc directement au module détecteur qui est lié au module évaluateur. Cet évaluateur est alors perçu comme étant le module implantant les événements.

. L'évaluateur est forcément en relation avec le régulateur des tâches ou scheduler , pour le contrôle du déroulement des tâches.

Il faut uniquement retenir, pour le moment, que l'opération d'assignation agit sur la table des événements. Son profil peut être défini par :

assigner : table_d'evt x désignation de source → table_d'evt

Cette opération a pour effet :

- de vérifier que l'événement concerné existe et a été déclaré,
- de vérifier que la source de l'évènement (donnée transmissible, interruption....) existe bien sur le site indiqué. Ceci ne s'applique toutefois pas aux événements issus de messages opérateur qui sont supposés toujours exister.
- de compléter, une fois toutes ces vérifications faites, la table des évènements réalisant ainsi la connection logique-physique.

Une deuxième assignation avec le même événement annule la précédente.

Notons que si, jusqu'à présent, l'on pressent le rôle d'éléments centraux du système de l'évaluateur et de la table des évènements, à aucun moment il n'a été question de les considérer comme centralisés. Nous retenons, en effet, la possibilité d'une solution décentralisée comme nous le verrons dans la chapitre 8.

Si l'opération d'assignation est simple dans sa définition, il n'en demeure pas moins qu'il reste un certain nombre de questions à éclaircir, parmi lesquelles :

- l'adéquation aspect logique-aspect physique, ou encore : un évènement peut-il être implanté par n'importe quelle source (par exemple, indifféremment interruption ou entrée booléenne) ?
- Pré-existe-t-il des modules de détection par scrutation dans le cas d'entrées numériques, analogiques dans le système ?
- Pré-existe-t-il dans le système des programmes de prises en compte des interruptions ?

V.3.1.- Adéquation logique-physique.

Quoique d'apparence sans fondements, étant donnée l'indépendance physique-logique préconisée, cette question d'adéquation prend toute son importance lorsque l'on tient compte des différences importantes pouvant exister entre les différentes sources d'événements. Le choix entre une interruption, par exemple, dont la scrutation est entièrement gérée par le calculateur lui-même et une entrée booléenne dont la scrutation est programmée, n'est très certainement pas fait au hasard. Les constantes de temps du procédé ainsi que la puissance de traitement peuvent y jouer un grand rôle.

Dans notre cas, nous occupant plus spécialement de spécification de logiciel de commande, nous considérons que ce problème de choix est traité à un autre niveau. Les problèmes des choix des matériels, y compris le matériel de détection, sa liaison avec le calculateur sont considérés comme déjà résolus et servant donc, pour nous, de base de travail. Ceci suppose, notamment, que le concepteur d'une application a une bonne idée du matériel sur lequel il travaille - ou alors qu'il en fait abstraction.

V.3.2.- Préexistance de modules de scrutation et d'interruption.

L'idée qui prévaut ici est que, en général, et pour un matériel donné, toutes les scrutations et les prises en compte d'interruptions se font de la même façon. Il est, dès lors, intéressant de disposer de modules standards de scrutation et de prise en compte d'interruption. Ces modules seraient parties intégrantes du système et plus exactement de l'évaluateur. Il est non moins intéressant de laisser également à un utilisateur la possibilité de fournir lui-même ses modules de détection, donnant ainsi plus de souplesse d'adaptation. Il est alors nécessaire de compléter la syntaxe de l'opération d'assignation :

assigner < id_d'evt_logique > à < source >
sur < adresse_site >

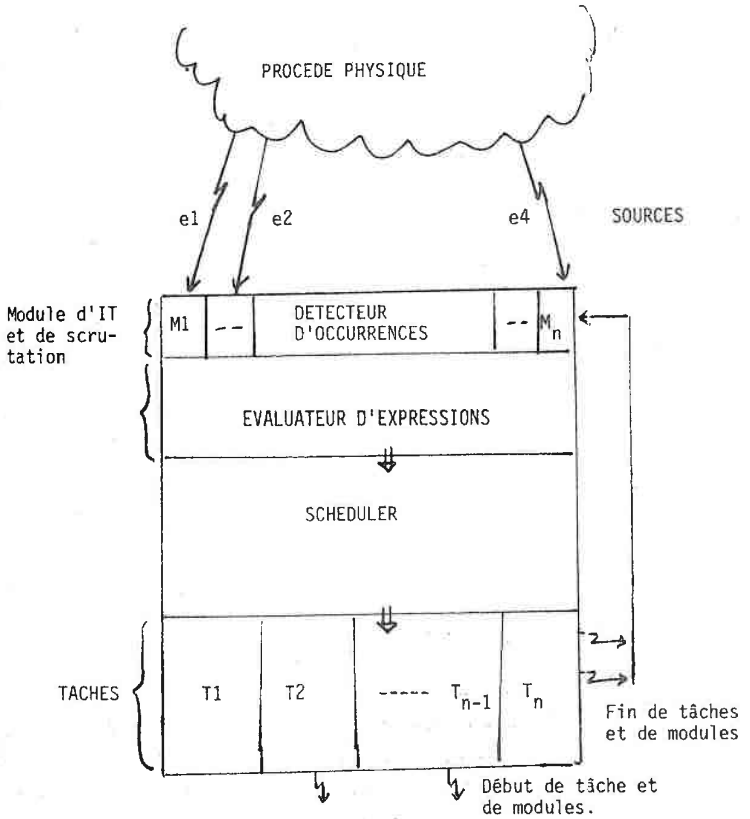
détecté par < nom_de_module >

[avec fréquence = < fréquence d'échantillonnage >]

La fréquence de scrutation n'est utile que pour les entrées numériques (booléennes ou non) et analogiques. Il faut noter qu'il s'agit là de véritable module d'Entrées-Sorties industrielles et pour lesquels il faut tenir compte des problèmes de filtrage et linéarisation (entrées analogiques). Pour les modules dits standards également, il faut préciser ces paramètres tout comme pour un module d'E/S. Concernant les interruptions, on constate qu'une tâche d'une application n'est jamais directement un programme d'interruption. Il existe en fait un interface soit fourni par le concepteur soit préexistant qui correspond exactement à la notion de tâche immédiate telle que définie dans [SCEP 82] . L'ensemble de ces tâches immédiates font partie du détecteur d'occurrences et donc du système. Cette façon de procéder a pour principal inconvénient de retarder la prise en compte d'une interruption mais permet par contre le partage d'un événement (et donc sa duplication) et sa redéfinition d'une manière aisée. De cette façon, également, on concrétise un véritable découplage entre le procédé physique et l'application.

L'assignation permet donc, soit le choix, soit la mise en place d'un module de détection des évènements.

Un tel système est synthétisé sur le schéma suivant :



Remarques.

1) La priorité de détection est définie :
   - directement par la spécification d'une interruption
   - par la fréquence de scrutation qui est une façon de préciser cette priorité. Celle-ci est en effet fonction de la vitesse d'évolution d'un phénomène et plus cette vitesse est élevée plus vite doit se faire la scrutation ; ceci traduit une certaine urgence de détection.
   - pour les messages opérateur, cette priorité est liée à la priorité de la tâche système assurant le dialogue opérateur.

2) Dans LTR.V3 [PAR 80] on retrouve également une notion d'assignation concernant ce qui est défini sous la terminologie d'interruption logique.

Une interruption logique peut être connectée à un niveau d'interruption physique et attachéeà une tâche particulière, par l'intermédiaire d'opérations définies dans le langage. Cet aspect ne concerne cependant que les interruptions.

CHAPITRE VI :

EXEMPLES

VI.1.- INTRODUCTION

Dans ce chapitre, nous allons traiter trois exemples pour étayer nos propos.

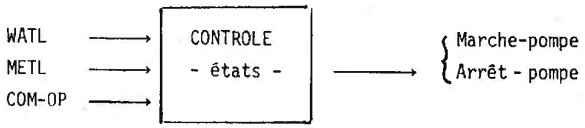
Aucune méthodologie particulière n'est suivie si ce n'est le recensement exhaustif des évènements et actions, permettant d'appréhender le fonctionnement du système. Les deux premiers exemples ont été surtout choisis par souci de possibilité de comparaison avec d'autres méthodes de spécifications - s'ils ne sont pas probants d'un point de vue partage d'évènements, ils permettent au moins de percevoir l'analyse de ce type de problèmes en termes d'automates.

Le troisième exemple est plus orienté "industrie manufacturière". Ce qui le différencie avec les problèmes précédents c'est essentiellement l'idée de transformation progressive d'un produit à usiner : ce peut être une piéce mécanique par exemple. On représente alors la pièce par une occurrence au niveau de l'application, occurrence à laquelle on peut appliquer naturellement des règles d'utilisation : partage.... C'est peut être la raison pour laquelle, nous arrivons à mieux percevoir dans ce cas les idées de concordance d'occurrences d'événements et partage d'occurrence.

VI.2.- EXEMPLE DE LA POMPE DE KRAMER.

Pour nous, on peut voir le système de contrôle et commande à spécifier, dans un premier temps, sous la forme d'une "boîte noire" caractérisée par un ensemble d'entrées et de sorties. La "boîte noire" réagit comme un automate élaborant les sorties vers le procédé à partir des entrées venant du procédé et de l'état du système global. Les entrées peuvent être booléennes, numériques.... On peut au cours de l'analyse s'intéresser aussi bien aux valeurs des entrées qu'à leurs variations (événements).

Les sorties déterminent les actions à entreprendre sur le procédé. On peut, alors, avoir le schéma.



WATL : niveau d'eau
METL : niveau de méthane
COM-OP : autorisation opérateur

A partir de ces entrées, il faut recenser les événements qui peuvent traduire l'évolution du procédé et sont donc susceptibles de déclencher certaines actions :

Evénements

- passage par la valeur symbolique "haut" de WATL : WH
- " " " " " "bas" de WATL : WL
- " " " " " "haut" de METL : MH
- " " " " " "bas" de METL : ML
- " " " " " "GØ" de COM-OP : GØ
- " " " " " "HALT" de COM-OP : HALT

Les actions à entreprendre sont dans ce cas très imples :
- mise en marche de la pompe : MARCHE
- arrêt de la pompe : ARRET

Examinons les conditions de déclenchement de ces actions à partir des événements et de l'état du système :

MARCHE :

La pompe ne peut être mise en marche que :

- sur une occurrence de WH à condition que le niveau de méthane soit bas et l'autorisation de l'opérateur effective.
- ou bien sur une occurrence de ML si le niveau d'eau est haut et l'autorisation de l'opérateur effective.
- ou bien sur une occurrence de GØ si le niveau d'eau est haut et le niveau de méthane bas.

En d'autres termes, on ne peut mettre en marche la pompe que si trois occurrences de ML, GØ et WH sont vivantes en même temps.

Ce qui correspond à l'opérateur SIM.

L'événement composé sera alors : GPUMP : WH SIM ML SIM GØ

(Nous utilisons une expression d'événements à trois éléments car celle-ci ne présente aucune ambiguité).

ARRET :

La pompe doit être arrêtée lorsque :
- le niveau de méthane devient haut.
- ou bien le niveau d'eau devient bas.
- ou bien l'opérateur a décidé d'arrêter la pompe, inconditionnellement.

Autrement dit chacune des occurrences de MH, WL et HALT doit provoquer l'arrêt de la pompe.

Il s'agit là d'une utilisation type de l'opérateur OU.

L'événement composé sera : HPUMP : MH OU WL OU HALT.

On remarque que cette démarche s'appuie sur les états dans lesquels peut être une pompe ("en marche" ou "à l'arrêt") caractérisés par des valeurs précises des entrées:

	"à l'arrêt"	"en marche"
ETATS	ou WATL = bas ou METL = haut COM-OP = halt	et WATL = haut et COM-OP = go METL = bas

Toute variation d'état doit entraîner l'une ou l'autre des actions MARCHE ou ARRET.

En fait il existe des variations d'entrées qui laissent inchangé l'état de la pompe : la pompe est arrêtée et le niveau de méthane devient haut.

Dans ce sens, l'expression WH SIM ML SIM GØ reflète bien toutes les conditions de marche.

Concernant l'arrêt, on est obligé, si on ne veut pas arrêter plusieurs fois de suite la pompe, d'introduire une variable d'état liée à la pompe : STATP.

Cette variable d'état devra être mise à jour par les tâches ARRET et MARCHE.

Liens entre événements et tâches

Que ce soit pour l'arrêt ou la marche, il s'agit de faire une action sur chacune des occurrences des événements associés.

Les liens seront du type :

```
sur  chaque GPUMP faire MARCHE
     avec cons        avec priorité = p1
                      (TØ, EV-TØ1, d1)

sur  chaque HPUMP  si  STATP = "marche" faire ARRET
     avec cons                          avec priorité = p2
                                        (TØ, EV-TØ2, d2)
```


avec perte d'occurrences si les conditions ne sont pas vérifiées. Le TØ pour la tâche MARCHE par exemple, a pour signification : si dès l'instant où les conditions de marche sont réalisées, la mise en marche est impossible dans les délais prévus (d1) alors l'occurrence peut être considérée comme caduque et un événement d'exception est déclenché (EV-TØ1). Ceci peut avoir pour signification qu'étant donné la vitesse de variation du procédé, la situation autorisant la marche de la pompe peut ne plus être vraie au bout de ce délai.

Dans la spécification proposée, on remarque que le niveau de communication entre le procédé et l'application n'apparaît pas explicitement. Le procédé est perçu en termes d'événements ; la façon de détecter et communiquer ces événements ne nous intéresse pas à ce niveau. L'existence de modules de détection est donc supposée, sans plus.

D'un point de vue évolutivité on peut par exemple :

- rajouter une condition d'arrêt/marche supplémentaire : le niveau de dioxyde de carbone dans le tunnel. Ce qui doit être retouché à ce moment-là, c'est uniquement la définition des événements et des moyens de détection. La spécification du fonctionnement reste valable.
- rajouter une nouvelle tâche d'affichage de l'état de la pompe, et des niveaux d'eau et de méthane à la demande de l'opérateur avant de prendre la décision d'arrêter la pompe.

On peut alors imaginer une tâche qui, selon la demande de l'opérateur fournit :

* soit l'état de la pompe
* soit le niveau d'eau et de méthane
* soit les deux.

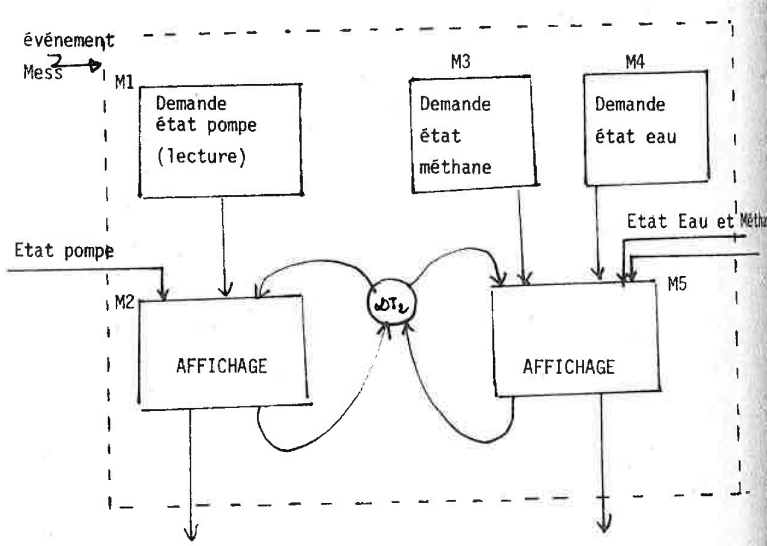
On pourrait avoir l'événement

mess : message-op = 'état pompe' ou message_op = "état eau met"
ou message-op = "état global".
et la structure d'utilisation

sur chaque Mess avec cons faire AFFICHAGE

avec AFFICHAGE ayant la structure de niveau 2 suivante (cf Annexe I)

si $\mathcal{D}$T1 = 'état pompe' alors (M1 ; M2)
sinon si $\mathcal{D}$T1 = 'état eau met' alors (M3 // M4) ; M5
sinon (M1 ; M2) // ((M3 // M4) ; M5)



$\mathcal{D}$T2 : donnée transmissible représentant une console opérateur par exemple et sur laquelle il y a exclusion mutuelle.

L'intégration de cette tâche a un but double :

- montrer que le reste de la spécification reste inchangé
- donner un exemple de tâche un peu moins "élémentaire" que MARCHE et ARRET.

Comme définitions d'événements nous avons :

evenement ML : elementaire, NOPART
evenement MH : elementaire, NOPART
evenement WH : elementaire, NOPART
evenement WL : elementaire, NOPART
" GØ : " , "
" HALT: " , "
evenement GPUMP : composé (ML SIM WH SIM GØ), NOPART
" HPUMP : " (MH OU WL OU HALT), NOPART

avec les assignations :

assign ML to transition vf (EB n°1) détecté par MOD1 avec fréquence = f1
" MH " " " (EB n°2) " " MOD2 " " = f1
" WH " transition vf (EB n°3) " " MOD3 " " = f2
" WL " transition fv (EB n°3) " " MOD3 " " = f2

Pour les messages opérateurs on aura

assign GØ to COM-OP = "GO" détecté par MESSOP
assign HALT to COM-OP = "HALT" " " "

Le module MESSOP est associé au dialogue opérateur.

Remarque .

La nature du problème traité se prête bien à une description sous forme d'automate élaborant des sorties correspondant à des actions simples.

La spécification reflète naturellement cet état de chose.

Cependant, les composants d'un système de contrôle ne se prêtent pas tous à une telle description et l'on trouve fréquemment des actions que l'on peut appréhender plutôt sous la forme d'un système combinatoire : élaboration "continue" de sorties en fonction d'entrées en tenant compte uniquement de la valeur des entrées. Dans cette catégorie on trouve les algorithmes de régulation du type PID. On peut en tenir compte au cours de la spécification de la façon suivante :

sur chaque evt faire PID
avec evt défini par : tous les Δt .

La valeur de Δt détermine la précision de la régulation.

Dans ce cas, on voit que ce ne sont pas les variations des entrées qui déterminent l'action à entreprendre.

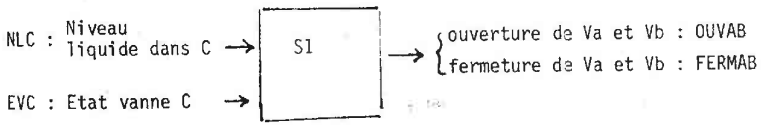
VI.3.- EXEMPLE DE L'EWICS TC11 .

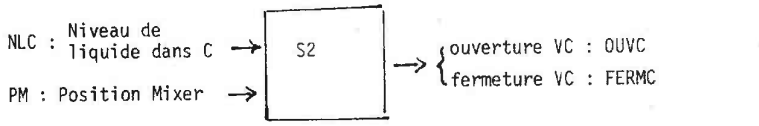
En poursuivant notre vue du système à concevoir sous la forme de "boîte noire", on peut pour cet exemple, décomposer le système à concevoir en sous-systèmes relativement indépendants dans ce sens que pour chacun de ces sous-systèmes on peut définir un fonctionnement autonome.

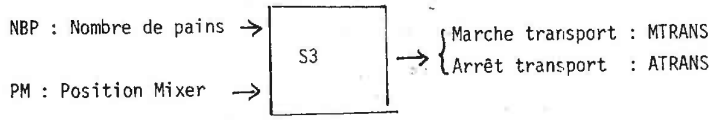
Intuitivement on peut avoir :

- un sous-système pour le contrôle des cuves A et B : S1
- " " " " " " de la cuve C : S2
- " " " " " " du transport : S3
- " " " " " " du mixer : S4

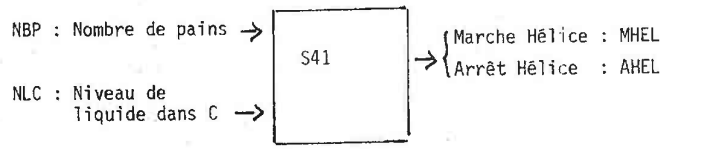
Pour chacun de ces sous-systèmes on aura :

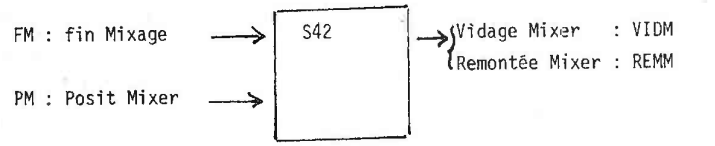






S4 se décompose lui-même en deux sous-systèmes : Mixage à proprement parlé S41 et Vidage du Mixer S42





Pour l'ensemble des sous-systèmes on aura comme événements :

vrai (NLC = haut) : CFULL (cuve C pleine)
vrai (NLC = bas) : CEMPTY (" " vide)
vrai (NBP = n) : NPAIN (nbre de pains = n)
vrai (PM = haut) : MH (position haute du mixer)
vrai (PM = bas) : ML (" basse " ")
vrai (EVC = fermée) : FC (vanne C fermée)
FM = vrai : FM (fin du mixage)

En plus de cela nous avons la détection du passage d'un pain:
UNPAIN (passage d'un pain).

Les actions sont celles recensées pour chacune des sorties des sous-systèmes.

Dans la suite, on suppose l'existence d'une tâche d'initialisation qui permet un démarrage correct du système : vérification des conditions initiales, déclenchement des actions.

Conditions de déclenchement des actions :

	Action	Evénement	Condition
S1	OUVAB	FC	vraie
	FERMAB	CFULL	vraie
S2	OUVC (1)	ØC : CFULL SIM MH	vraie
	FERMC	CEMPTY	vraie
S3	MTRANS	MH	vraie
	ATRANS	NPAIN	vraie
S41	MHEL (2)	MIX	PM = "haute"
	AHEL (3)	AMIX	VE = "marche"
S42	VIDM (3)	AMIX	VE = "arrêt"
	REMM	ML	vraie

(1) Les conditions d'ouverture de la vanne C sont exprimées par :

- sur occurrence de CFULL si le mixer est haut

ou - sur occurrence de MH si la cuve C est pleine.

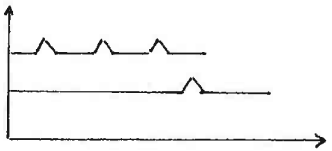
(ce qui traduit le fonctionnement parallèle des systèmes S1 et S4). On a alors un opérateur SIM.

(2) A ce niveau, il s'agit d'exprimer les synchronisations nécessaires indiquant la présence des produits permettant la fabrication du produit fini. Ce processus de fabrication consiste à mélanger ces produits deux-à-deux pendant un temps fini. On aura un événement du type

MIX : chaque CEMPTY et chaque NPAIN
avec cons avec cons

car le seul moyen que l'on ait de vérifier la présence des produits en question dans le mixer est de comptabiliser les occurrences de CEMPTY et NPAIN.

Il faut alors s'assurer que l'on ne risque pas de tomber dans la situation où plusieurs occurrence de CEMPTY (ou NPAIN) sont produites avant une occurrence de NPAIN (CEMPTY) :



ce qui ne peut pas être le cas étant donnée la relation de précédence imposée par le cycle de fonctionnement du mixer sur le fonctionnement du transport et le vidage de C et traduit par l'événement MH.

Ainsi, même si elles ne sont pas visibles, on voit apparaître les relations de précédence imposées par le fonctionnement et liées donc à la nature du problème. La condition est nécessaire car rien ne permet d'affirmer le contraire.

(3) Dans ce cas, on fait intervenir le temps du mixage, seul élément pour déterminer la fin du mixage.

On aura un événement du type :

AMIX : Δt2 <u>après</u> MIX

La condition est nécessaire car, tout ce qu'indique AMIX c'est simplement qu'un délai s'est écoulé depuis que les produits ont été placés dans le mixer. Rien ne dit que la mise en marche du mixer a été réussie.

VE est une variable d'état mise à jour par les tâches AHEL et MHEL

Comme fonctionnement anormal, on peut prévoir, par exemple, le cas où le vidage ne se fait pas (panne du moteur correspondant). On peut avoir alors un événement panne :

PANNE1 : Δt3 <u>après</u> MIX et

avoir une structure d'utilisation du type :

<u>sur</u> PANNE1 <u>si</u> PM = "haute" <u>alors</u> Tâche Exception

Le délai Δt3 est calculé de telle sorte qu'en fonctionnement normal la position du Mixer soit différente de haute.

<u>Structures d'utilisation</u> :

<u>sur chaque</u> FC <u>faire</u> OUVAB
<u>avec cons</u>

<u>sur chaque</u> CFULL <u>faire</u> FERMAB
<u>avec cons</u>

<u>sur chaque</u> ØC <u>faire</u> OUVC
<u>avec cons</u>

<u>sur chaque</u> CEMPTY <u>faire</u> FERMC
<u>avec cons</u>

<u>sur chaque</u> MH <u>faire</u> MTRANS
<u>avec cons</u>

<u>sur chaque</u> NPAIN <u>faire</u> ATRANS
<u>avec cons</u>

<u>sur chaque</u> MIX <u>si</u> PM = "haute" <u>faire</u> MHEL
<u>avec cons</u> <u>sinon faire</u> Anomalie 1

* sur chaque AMIX si VE = "marche" faire AHEL
  sans cons sinon faire Anomalie 2

* sur chaque AMIX si VE = "arrêt" faire VIDM
  sans cons sinon faire Anomalie 3

* AMIX est partagé

sur chaque ML faire REMM
avec cons

avec les déclarations d'événements :

événement	CEMPTY	:	élémentaire
"	CFULL	:	"
"	MH	:	"
"	ML	:	"
"	UNPAIN	:	"
"	FC	:	composé (Δt1 après CEMPTY)
"	ØC	:	" (CFULL SIM MH)
"	NPAIN	:	" (n UNPAIN)
"	MIX	:	" (chaque CEMPTY et chaque NPAIN) avec cons avec cons
"	AMIX	:	" (Δt2 après MIX), PART

AMIX est un événement partagé entre VIDM et AHEL ; la consultation est dans ce cas nécessaire.

Ceci permet de traduire la situation de "parallélisme" (qui est plus de l'indépendance) entre les actions d'arrêt de mixage (AHEL) et de vidage du mixer (VIDM) : celles-ci peuvent en effet démarrer sur la même occurrence de Δt2 après MIX.

- Comme assignations on peut avoir :

assign MH to transition fv (EN n°1 > seuil 1) détecté par MOD1
avec fréquence = f1

assign ML to transition fv (EN n°2 > seuil 2) détecté par MOD2
avec fréquence = f2

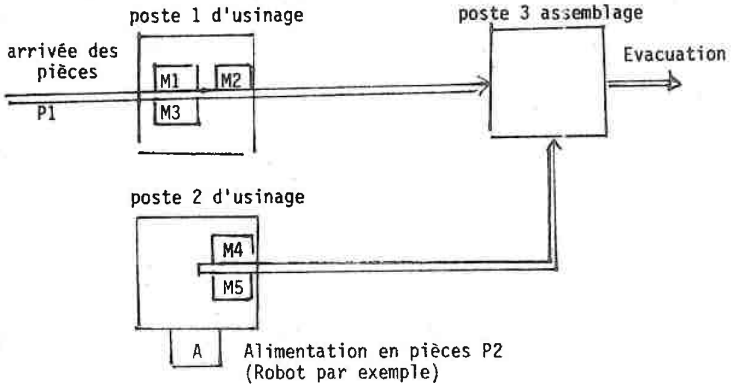
assign CFULL to transition fv (EB n°5) détectée par MOD3
avec fréquence = f3

assign CEMPTY to transition fv (EB n°6) détectée par MOD4
avec fréquence = f4

assign UNPAIN to IT n°10

VI.4.- EXEMPLE D'UN ATELIER D'USINAGE.

On dispose, dans un atelier d'usinage, de deux postes d'usinage et d'un poste d'assemblage représenté de la façon suivante :



M1, M2...M5 représentent les machines-outils.

Le problème est le suivant : disposant de 2 types de pièces P1 et P2, il s'agit, pour chacune des pièces :

- de faire subir un usinage 1 au niveau du poste 1 pour P1
- de faire subir un usinage 2 au niveau du poste 2 pour P2
- de faire un assemblage de ces deux pièces au niveau du poste 3 pour réaliser le produit fini.

Comme contraintes nous avons :

- M1 et M2 peuvent travailler en même temps sur une pièce.
- M4 et M5 également.
- Pour des raisons de cohérence au niveau de l'assemblage, le poste 2 doit usiner autant de pièce P2 que de pièce P1
- On n'impose pas au poste 1 et poste 2 de travailler au même rythme.

Dans la suite, nous utiliserons les notations de SYGARE (Annexe I)

1) Le système à concevoir dispose de trois entrées : celle qui permet de détecter le passage d'une pièce P1 avant le poste 1, et celles qui permettent de détecter l'arrivées des pièces P1 et P2 au niveau du poste d'assemblage.

2) Comme événements nous avons :

- arrivée d'une pièce P1 : AP
- événements indiquant l'arrivée d'une pièce P1 et d'une pièce P2 au poste 3 : AP1 et AP2

3) Comme actions, nous avons :

- usinage M1
- " M2
- " M3
- " M4
- " M5

- Alimentation AL
- Assemblage : ASS
- Evacuation : EVC

les actions peuvent être regroupées en tâches fonctionnelles, c'est-à-dire un ensemble d'action logiquement liée :

T1 : (M1 // M3) ; M2
T2 : AL ; (M4 // M5)
T3 : ASS ; EVC

4) Liens entre événements et tâches

- T1 devra être exécutée pour chacune des occurrences de AP
- T2 " " " " " " " " AP

On remarque que l'événement AP est partagé par T1 et T2.

De plus chacune des tâches devra utiliser toutes les occurrences d'après l'énoncé du problème.

- T3 devra être éxécutée pour chacune des occurrences d'un événement à définir et qui indique les synchronisations nécessaires.
- Aucune condition n'est imposée pour la prise en compte des occurrences.

Définition des événements :

Evénement AP : élémentaire , PART
Evénement AP1 : élémentaire , NØPART
" AP2 : " , NØPART

Les synchronisations nécessaires pour l'éxécution de T3 sont du type :

AP1 et AP2.

Il est évident que chaque occurrence de AP1 et AP2 devra être considérée une fois et une seule dans la composition - ce qui nous donne :

Evénement SYNC : composé, NOPART
(chaque AP1 ET chaque AP2)
avec cons avec cons

On aura alors les structures d'utilisation suivantes :

sur chaque AP sans cons faire T1
sur chaque AP sans cons faire T2

ce qui exprime bien le fait que chacune des tâches utilise bien toutes les occurrences souhaitées
et

sur chaque SYNC faire T3
avec cons

Cet exemple nous a servi surtout à illustrer l'aspect partagé d'événement et c'est la raison pour laquelle nous n'avons pas voulu y faire figurer les priorités, assignations.... qui sont semblables à celles des exemples précédents.

VI.5.- CONCLUSION ET REMARQUES.

La spécification proposée ne tient pas compte des sous-systèmes associés à des objets comme par exemple un module gestion de la pompe dans l'exemple 1 ou un module gestion du mixer dans l'exemple 2.

Toutefois, on peut considérer que cette spécification fait apparaître ces sous-systèmes.

En effet, après énumération des différents événements et actions associées, on peut regrouper ces actions traduites en tâches à l'intérieur d'un module du type de ceux de Zakari [ZAK 84].

Les événements sont alors tous associés à ce module sous la forme de message et à un événement peut être attaché une tâche.

Il est également possible de faire des fusions de tâches (comme par exemple la tâche OUVAB dans l'exemple 2) pour tendre vers une représentation plus synthétique des sous-systèmes.

On remarque alors que la spécification proposée présente l'avantage de permettre une énumération exhaustive des événements et actions à entreprendre dans une première étape et en même temps laisse la possibilité dans d'autres étapes de regrouper les différentes actions selon un découpage souhaité (sous-systèmes...).

De plus, avec une telle spécification :

- l'on connaît exactement les conditions de déroulement des traitements,
- toute modification dans le déroulement d'une action est facile à réaliser et ne remet pas en cause l'action elle-même (changement du délai $\Delta t2$ par exemple)
- il est extrêmement simple de rajouter une nouvelle action, modifier une action.... permettant ainsi une bonne évolutivité,
- le recensement exhaustif des événements permet de prévoir tous les cas et va dans le sens d'une plus grande fiabilité du logiciel.

CHAPITRE VII :

RÉALISATION

VII.1.- PRESENTATION GENERALE

D'une manière générale, cette réalisation vise à montrer la possibilité d'une traduction de la spécification proposée (concernant les aspects déclarations d'événements, structures d'utilisation et assignation) en une application multitâches dans un langage donné intégrant des primitives temps réel FØRTRAN-TR sous RTES-D sur l'ordinateur SOLAR 16-40 de la SEMS. Nous ne nous intéressons pas à la compilation des tâches elles-mêmes qui est supposée faite par ailleurs. Ce premier objectif de faisabilité atteint, le second objectif est de montrer la complexité (en nombre de tâches obtenues au moins) de l'application multitâches ainsi obtenue, complexité due à l'inéxistence dans le système visé de mécanismes adéquats d'expression des concepts utilisés dans la spécification. Remarquons que cette complexité peut être réduite en adjoingnant au système visé une couche supplémentaire implantant un certain nombre de concepts proposés.

De quoi démarrons-nous pour assurer cette traduction ?

Sans vouloir proposer une syntaxe précise, nous supposons que l'on démarre d'une spécification écrite dans un langage de spécification intégrant les concepts développés ; une spécification pourrait être de la forme :

```
SYGAP
        DECEVT
         --   déclaration des événements
        FINEVT
        DECCOND
         --   déclaration des conditions
        FINCOND
--      Déclaration des  Modules
--          "        "   Données transmissibles
--          "        "   Connexions
--          "        "   Tâches
        DECNIV3
          Déclaration des structures d'utilisation
        FIN NIV3
PAGYS
```

Nous nous intéressons alors à la compilation des paragraphes DECEVT et DECNIV3.

C'est en fait cela qui permet de déterminer la structure de l'application multitâches visée.

Quelle est la structure de cette application multitâches ?

A ce niveau, nous retrouverons, et c'est normal, deux types de tâches :

- les tâches dites de service
- les tâches utilisateur telles que précisées dans la spécification.

Parmi les tâches de service, nous définissons plusieurs niveaux de tâches :

- les tâches immédiates liées aux niveaux d'interruptions, entrées numériques.... implantant les opérations d'assignation. Ce sont les tâches de détection.

- les tâches implantant les expressions d'événements et qui constituent l'évaluateur et enfin.
- les tâches assurant le partage des événements appelées tâches de niveau 3

Pour chacun des événements déclarés il existe :

- une tâche d'évaluation
- une tâche de niveau 3.

On trouvera ci-après un schéma général montrant cette décomposition en niveaux.

Au cours de cette réalisation nous ne nous sommes pas occupés de l'analyse syntaxique. Toutes les structures de données ont néanmoins été définies et chaque fois que cela a été nécessaire, nous indiquons ce que la phase d'analyse syntaxique fournie.

Nos préoccupations se sont portées plutôt vers la génération de l'ensemble des tâches avec les liens d'activations nécessaires pour faire marcher l'application.

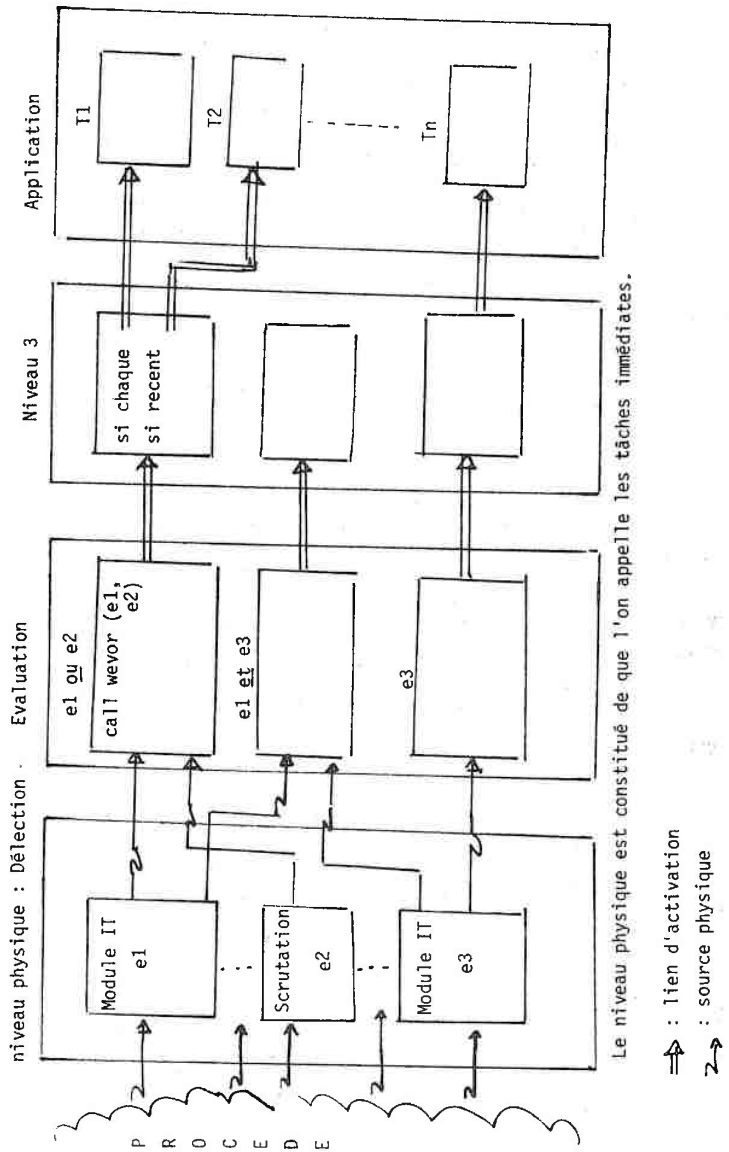
Le système RTES-D étant monosite, l'application ainsi obtenue est forcément monosite.

Les priorités des tâches seront affectées globalement de la façon suivante :

- tâches de détection
- tâches d'évaluation
- tâches de niveau 3
- tâches utilisateur

par ordre décroissant.

Un principe a été retenu pour cette décomposition de l'application: pas de pertes d'occurrences involontaires à tous les niveaux.
Ce qui explique le nombre de tâches parallèles et les priorités globales.



SCHEMA GENERAL

Les priorités définies dans la spécification ne correspondant pas aux priorités définies sur le système hôte, un classement des tâches intervient entre l'analyse syntaxique et la génération des tâches pour faciliter l'affectation des priorités par le compilateur.

Les tables référencées se trouvent dans l'annexe III.

Les primitives FORTRAN-TR utilisées seront introduites au fur et à mesure des besoins.

Les descriptions de réalisation que l'on va trouver dans la suite de ce chapitre seront plus axées sur la génération des tâches des différents niveaux.

Un dernier point avant de terminer cette introduction concerne la définition nécessaire d'une tâche initiale, INIT, éxécutée au lancement de l'application et chargée de :

- lancer les tâches de détection
- lancer les tâches liées au temps : démarrage à heure fixe...
- lancer les tâches cycliques d'évaluation des expressions

et d'autres fonctions qui seront précisées par la suite.

D'une manière générale, la génération de cette tâche INIT se fait en parallèle de la génération des autres tâches.

Les modules de détection sont supposés pré-existants.

L'algorithme de principe est le suivant

A) Génération des tâches d'évaluation

-- il y a en a une par événement simple ou composé
-- tables utilisées : TABEXP, TEVN3, TABASS
TABSEQ.

tant que TABEXP non vide faire

-- 1)-- pour les tâches d'évaluation
- Rechercher la séquence FØRTRAN à compléter à partir de TABSEQ.
- La compléter à partir des autres tables.

-- 2) pour la tâche INIT

Les tâches d'évaluation étant cycliques, générer la primitive d'activation adéquate.

fait

B) Génération des tâches de niveau 3

Tables utilisées : TABEXP, TEVN3, TABTK3, TABNIV3
TABSEQ.

Tant que TEVN3 non vide faire :

- Rechercher la séquence correspondante à partir de TABSEQ;
- Compléter la séquence à partir des autres tables :
appels des tâches de l'utilisateur notamment.

C) Complétion des tâches de l'utilisateur

Dans ce cas le travail consiste à rajouter une "enveloppe" à chacune des tâches utilisateur pour assurer une prise en compte correcte des occurrences ; notamment en ce qui concerne le partage.

Tables utilisées : TEVN3, TABTK3, TABNIV3.

Le parcours se fait sur la table TABTK3.

D) Génération des tâches de gestion des time-out

Tables utilisées : TABNIV3, TABTØ, TABTK3, TABSEQ.

Il s'agit ici de générer une tâche qui prenne en compte le délai de garde. Cette tâche est capable de produire une occurrence de l'événement d'exception quand le délai de garde est écoulé.

Plus de détails seront donnés dans la suite concernant toutes ces tâches.

Un exemple de traduction est donné en annexe IV.

VII.2.- SCHEMAS DE TRADUCTION DES STRUCTURES D'UTILISATION AU NIVEAU DES TACHES UTILISATEURS

Cela dépend de la structure d'utilisation et en particulier :

i) - de la prise en compte fugitive : absence de qualificatif
- de la prise en compte des plus récentes occurrences :
qualificatif Recent
- de la prise en compte de toutes les occurrences :
qualificatif chaque

ii) - de la consommation : attribut avec cons
- de la consultation : attribut sans cons

Parmi les requêtes programmées au système RTES-D nous pouvons déjà retenir

- CALL RUN (n° tâche...) qui permet une demande d'activation sans délai. Ce type de requête est automatiquement comptabilisé par le système : les demandes d'activation successives sont mémorisées. Le nombre d'appels cumulables peut être limité à la déclaration d'une tâche (paramètre IAPPMAX)

- CALL SEVENT (EVT) : qui permet de positionner un événement à "arrivé".
Il y a mémorisation d'une seule demande de positionnement à la fois.

- CALL WEVENT (EVT) : qui permet d'attendre le positionnement d'un événement à l'état "arrivé"

- CALL REVENT (EVT) : positionnement d'un événement à l'état "non arrivé" et qui correspond à ce que l'on a appelé consommation.

- CALL EXIT : Fin d'éxécution d'une tâche

Les occurrences seront traitées comme des ressources :

suite d'occurrences.

avec deux variables particulières

i : nombre d'occurrences consommées
ve : nombre d'occurrences produites.

Ces deux variables nous permettent d'assurer la gestion de la consommation des occurrences.

Note.

Il existe d'autres requêtes concernant la gestion du temps et auxquelles nous nous intéresserons par la suite.

Traduction

Pour traduire le recent, chaque et fugitif, on utilise simplement le paramètre IAPPMAX

IAPPMAX = 0 : prise en compte fugitive, les appels n'étant pas mémorisés
IAPPMAX = 1 : recent
IAPPMAX = n : chaque (en pratique on prend n = 10)

Les tâches utilisateurs sont appelées à partir des tâches de niveau 3 après vérification des conditions par CALL RUN (n° tâche...).

La consommation/consultation est implantée au niveau de chaque tâche utilisateur. Chaque événement est représenté par son couple de variables (ve_k, i_k).

Des sémaphores d'exclusion mutuelle sont nécessaires pour la gestion de ces variables.

Sous RTES-D on peut déclarer une ressource (= sémaphore général)

déclarer une ressource : CALL RESDEF (N°, Nb-accès)
= 1 si exclusion mutuelle
demander un accès : CALL RRQST (N°)
demander une libération : CALL RRLSE (N°)

a) Traduction de sans cons

```
deb : P(mutex_j)
      si ve_k = i_k alors (V(mutex_j)
                           allerà fin)
      V(mutex_j)
        :
        :      corps de la tâche
        :
fin : call exit
```

b) Traduction de avec cons

```
deb : P(mutex_j)
      si ve_k = i_k alors (V(mutex_j)
                          allerà fin)
      i_k := i_k+1  -- consommation
      V(mutex_j)
      -
      -
      -
fin : call exit
```

c) Traduction de la prise en compte fugitive

```
deb = -
      -
      -   corps de la tâche
      -
      call exit
```

La traduction des qualificatifs chaque, recent ou fugitf est assurée, rappelons-le par le paramètre IAPPMAX d'une part et par l'appel de la tâche par call RUN dans une tâche de niveau 3 d'autre part.

VII.3.- SCHEMA DE TRADUCTION D'UNE TACHE DE NIVEAU 3

L'activation des tâches est liée :

- d'une part à l'arrivée des occurrences
- et d'autre part aux conditions traduisant l'état du système.

Nous avons donc besoin d'une tâche supplémentaire, jouant le rôle d'une tâche principale contrôlant l'activité des tâches de l'application dépendant d'un événement donné en fonction des occurrences de l'événement et de l'état du système.

Afin d'assurer le parallèlisme entre les tâches, il existe une tâche de ce type par structure d'utilisation. Il s'agit en quelque sorte de l'implantation des structures d'utilisation.

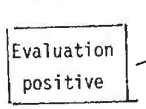
A chacune des entrées de TEVN3 correspondra une tâche de ce type. Cette tâches est elle-même réveillée à chaque évaluation positive de l'événement attaché.

Rappelons que cet événement peut provenir d'une expression d'événements ou encore d'un événement simple.

A ce niveau, on ne peut autoriser de perte d'occurrences. L'activation d'une telle tâche se fait obligatoirement par un appel du type

```
Call run (n° tâche...)
```

Nous avons alors :




```
Tâches de Niveau 3
P(mutex_j)
   ve := ve+1 ; -- l'occurrence est enregist
V(mutex_j)
-- Recherche, à partir de TEVN3 et TSTN3 des tâches
 attachées à cet événement --
   Pour chacune de ces tâches faire
    -- vérifier condition
       si cond alors  call run (n° tâche--)
       sinon si ∃  des parties sinon si (n°Branche > 0)
```

Pour chaque branche faire
si indic-sinon alors call run (n° tâche)
sinon si cond alors call run(n° tâche--)
fait
-- si aucune condition n'est vérifiée et ≠ de partie sinon alors il n'y a pas d'activation --
fait

Remarques.

1) Toutes les tâches dont les conditions sont vraies sont activées. Le "tri" se fait au niveau de chacune de ces tâches car la gestion de la consommation est assurée à ce niveau, en exclusion mutuelle ; toutes les tâches ne se dérouleront pas, par conséquent, effectivement.

2) La séparation de l'évaluation et du traitement de niveau 3 est motivée principalement par le souci de prise en compte rapide (séquences courtes et prioritaires) des occurrences simples et composées. Une fois l'évalutation assurée, un traitement plus "lourd" (recherche de tâches, activations...) est ensuite réalisé.

VII.4.- IMPLANTATION DES MODULES DE DETECTION ET LIENS AVEC L'EVALUATION

Cette implantation est directement issue :

- des descriptions faites dans TABASS
- des opérations d'assignation.

Si on suppose les modules de détections fournis, on peut avoir deux cas :

i)- il s'agit d'une interruption

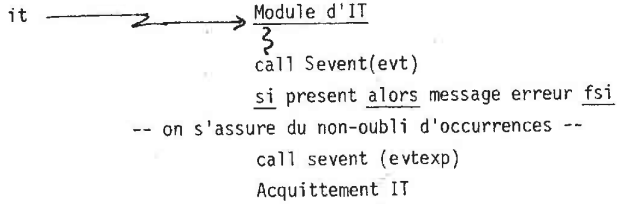
. On vérifie l'existence du niveau d'interruption.
. On "accroche" le module au niveau d'IT en le complétant par une opération signalant l'arrivée de l'occurrence pour toutes les expressions utilisant cet événement.

Dans la mesure où un tel module doit être très court on pense naturellement à la requête

CALL SEVENT(evt)
↳ provenant de TABEXP et servant de lien avec les tâches de traitement des expressions comme nous le verrons plus loin.

Cette requête évite de connaître l'identité de tâches de traitement des expressions (ce qui n'est pas le cas de CALL RUN(n° T)) mais présente le principal inconvénient de ne pas empêcher complètement les pertes d'occurrences.

On peut compliquer légèrement ce module de détection en s'assurant par exemple, avant de répercuter une occurrence au niveau des tâches de traitement des expressions, de la "disponibilité" de ces tâches :



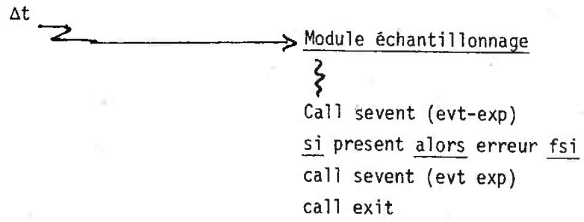
ii) il s'agit d'un module d'échantillonnage

Les problèmes de traitement du signal (filtrage, linéarisation,...) sont traités dans le module

. il faut compléter le module de la même façon que le module d'IT.
. lui adjoindre une structure d'appel périodique fonction de la fréquence d'échantillonnage (table TABASS)

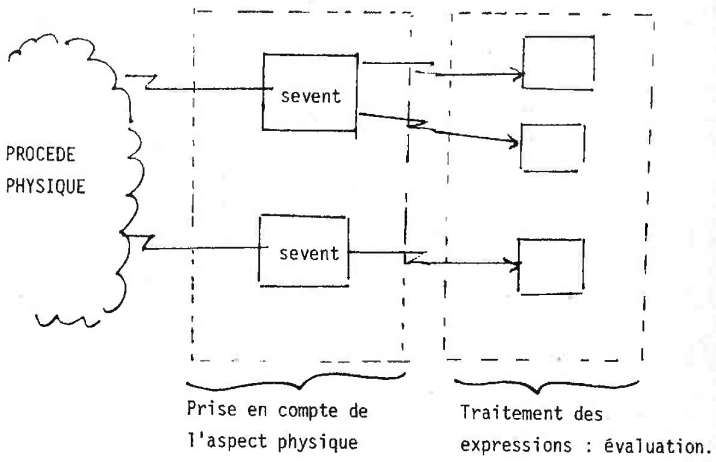
CALL START (n° tâche, 0, 0, ier, ipar, ip, iup)
fréquence d'appel.

Le module est donc traité en tant que tâche périodique qu'il faut démarrer impérativement au démarrage de l'application :



Ce niveau physique constitue le premier niveau d'implantation en laison directe avec le procédé

On peut avoir le schéma :



VII.5.- SCHEMAS DE TRADUCTION DES TACHES D'EVALUATION.

Il s'agit ici d'étudier, pour chacune des expressions proposées. Les possibilités d'implantation.

Comme expressions, nous avons

- Avant avec et sans anticipation
  - e avant j^{e} < borne >
  - e avant {chaque / recent} < borne >
- ET

  {chaque / recent} e1 {avec / sans} cons ET {chaque / recent} e2 {avec / sans} cons
- OU

  e1 OU e2
- < Nombre > occurrences de < evt >
- < delai > après < evt >
- Après

  e après j^{e} < borne >

  e après {chaque / recent} < borne >

D'une manière générale et comme esquissé dans l'introduction, l'implantation des expressions se fait sous la forme de tâches parallèles, à raison d'une tâche par expression.

La raison essentielle de ce choix est d'avoir les traitements les plus courts possibles de façon à ne pas perdre d'occurrences.

Le découpage n'est pas strict et des regroupements de plusieurs tâches doivent être possibles sous certaines conditions afin d'éviter une trop grande prolifération des tâches. Sous certaines conditions, il est également possible de regrouper le niveau production et le niveau détection.

Les requêtes intéressantes que l'on pourra utiliser sont CALL WEVØR(-) (OU entre deux événements) et CALL WEVAND (ET entre deux événements).

Etude de l'implantation des expressions

1) evt avant-a j^{ieme} borne

```
-- Il est nécessaire d'introduire un compteur
                          cpt : entier -- variable locale
      (beg : call wevor (--, evt, borne)
   -- attente de evt ou borne
      call revent (evt) ; call revent (borne)
         si borne alors (cpt+:=1 ;
                         si cpt = j aller à fin
                         sinon aller à beg
        -- sinon il s'agit de evt
          call run (n° tâche ---)
   -- le n° de tâche est retrouvé à partir du nom de l'evt
      traité dans  TEVN3  = il s'agit d'une tâche de niveau 3
           aller à beg ;
      fin : call exit)
   fin tâche
```

2) evt avant-s $j^{ième}$ borne

```
      Tâche
      vevt, cpt : entier := 0
           (beg : call wevor (-, evt, b) ;
                  call revent (evt) ; call revent (b) ;
             si  borne alors (cpt +:= 1 ;
                              si cpt = j aller à fin
                              sinon aller à beg)
             vevt +:= 1 ; -- compte le  nombre d'occurrences de evt
```



```
           aller à beg ;
      fin : tant que  vevt ≠ 0  faire
            call run (n° tâche ---)
            vevt -:=1 ;
                                  fait
            call exit)
fin tâche
```

3) evt avant-a chaque < borne >

La consommation de la borne est implicite

```
      Tâche
           vborne : entier := 0 ; aig : booleen = faux
           (beg : call wevor (-, -, evt, borne) -
                  call revent (evt)
                  call revent (borne)
                  si borne alors (si ¬aig ou vborne >0
                              alors vborne +:= 1 fsi
                                 aller à beg)
                 -- le test sur aig sert à assurer la
                    consommation de la borne --
                  aig := vrai -- il y a eu au moins une
                  occurrence de evt
                  si vborne > 0 alors (--borne avant evt) vborne -:=1 ;
                                 sinon  call run (n° tâche---) fsi
                                 -- appel tâche de niveau 3 --
           aller à beg)
```

4) evt avant-s chaque < borne>

```
Tâche
      vborne, vevt : entier := 0
      (beg : call wevor (--, evt, borne)
             call revent (evt) ; call revent (borne)
             si borne alors (si vevt > 0  alors
                                (tant que vevt > 0 faire
                                 vevt -:= 1 ;
                                 call run (n° tâche --)
                                                fait
                                )
                                sinon vborne +:= 1 ;
                             aller à beg)
        -- sinon il s'agit de evt
           vevt +:= 1 ;
           aller à beg)
      )
fin tâche
```

Les cas portant sur le récent exemplaire de la borne se déduisent aisément des cas précédents en déclarant vborne : booléen et en remplaçant les incrémentations par vborne := vrai , les décrémentations par vborne := faux. Les tests sont modifiés en conséquence.

Remarque :

La simultanéité des occurrences de la borne et de l'événement est considérée au temps de cycle de la tâche de détection près. En cas de simultanéité, l'occurrence de l'événement comparé est exclue.

5) Cas du ET

D'une manière générale

- chaque est implanté à l'aide d'un compteur d'occurrences
- recent " " " " d'une variable booléenne
- la consommation : - en décrementant le compteur ou
  - en rendant fausse la variable booléenne

Nous traitons trois cas pour montrer comment on peut implanter de telles expressions, les autres cas s'en déduisant facilement

cas 1

* chaque e1 et chaque e2
avec cons avec cons

```
Tâche
      ve1, ve2 : entier := 0 -- pour traduire chaque
      (beg : call wevor (--, e1, e2)
             si e1 alors (ve1 +:= 1 ; call revent (e1)) ;
             si e2 alors (ve2 +:= 1 ; call revent (e2)) ;
             si ve1 > 0  et  ve2 > 0  alors
                            (ve1 -:= 1 ; -- la       --
                             ve2 -:= 1 ; -- consommation --
                             call run (n° tâche ---)
             aller à beg)
fin tâche
```

cas 2

* chaque e1 et recent e2
sans cons avec cons

```
Tâche
      ve1 : entier := 0 ; ve2 : booleen := faux ;
```

```
        (beg : call wevor (--, e1, e2)
          si e1 alors (ve1 +:= 1 ; call revent (e1)) ;
          si e2 alors (ve2 := vrai ; call revent (e2)) ;
          si ve1 > 0  et  ve2  alors
                          (ve2 := faux -- consommation
                           call run (n° tâche ---)
                           )
    aller à beg)
  fin tâche
```

Remarque :

Dans la mesure où la consommation n'a pas d'incidence sur l'évaluation, on aurait pu implanter chaque e1 par une variable booléenne.

cas 3

* recent e1 et recent e2
avec cons avec cons

Tâche

```
      (beg) : call wevand (-, -, e1, e2)
              call revent (e1)
              call revent (e2)
              call run (n° tâche --)
              aller à beg)
```

fin tâche

6) Cas du OU

e1 ou e2

Tâche

```
      (beg : call wevor (-, -, e1, e2)
             call revent (e1)
             call revent (e2)
             call run (n° tâche ---)
             aller à beg
      )
```

fin tâche

7) Cas du APRES

Ce cas n'introduit aucune difficulté majeure et utilise exactement les mêmes mécanismes que le AVANT et le ET

8) Cas du < délai > après < evt >

Δt après e

Dans ce cas plutôt que d'utiliser CALL WAIT (délai) qui provoquerait la suspension de la tâche de détection pendant un délai pouvant entraîner des pertes d'occurrences, il faut plutôt avoir recours à l'activation différée de la tâche de niveau 3 correspondante

CALL START (n° tâche, < délai > ---)

On aura donc

Tâche

```
      (beg : call wevent (e, --) ;
             call revent (e) ;
             call start (n° tâche, Δt,-----)
                         niveau 3
             aller à beg)
```

fin tâche

Les activations par CALL START sont comptabilisées par RTES-D.

9) Cas du < nombre > occurrences de < evt >

N occurrences de e

```
Tâche
        ve : entier := 0 ;
        (beg : call wevent (e, ---)
               call revent (e ---)
               ve +:= 1 ;
               si ve = N alors ( call run (n° tâche ---)   ;
                                 ve := 0) ;
               aller à beg
fin tâche
```

VII.6.- PRISE EN COMPTE DU TEMPS

Nos expressions concernant l'utilisation du temps portent sur les possibilités de :

- spécifier une heure précise :
  dans la déclaration d'un événement
  événement <evt> : composé (heure = < heure >)
- spécifier des événements périodiques
  . à partir d'une heure précise
  dans la déclaration
  événement <evt> : composé (chaque Δt après <heure>)
  . avant une heure précise
  événement <evt> : composé (chaque Δt avant <heure>)

La prise en compte de tels événements est nécessairement fugitive (ce qui est normal étant donné l'aspect temporel).
La plupart des systèmes temps réel étant dotés de mécanismes gestion du temps, il nous a paru judicieux, plutôt que d'alourdir

inutilement la gestion des tâches, d'essayer de faire une traduction plus directe utilisant les mécanismes du système hôte.
Le compilateur peut, à partir de la déclaration des événements et la spécification des structures d'utilisation engendrer directement la structure d'activation ad-hoc sous le système hôte.
La traduction s'en trouve d'autant plus simplifiée.

Dans le cas qui nous intéresse, on peut utiliser directement les appels au système RTES-D (sur SOLAR 16) intégrés à FORTRAN-TR :

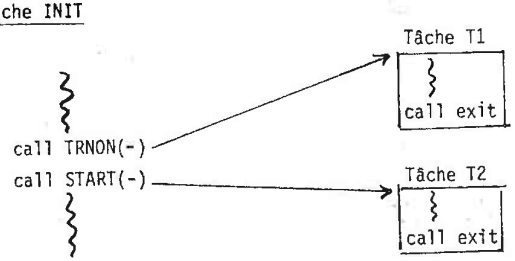
- CALL TRNON (N° tâche, < heure >, p..) pour l'activation à une heure donnée d'une tâche - et éventuellement périodiquement après -
- CALL START (N° tâche, --- [< périodes >]) pour l'activation périodique d'une tâche.

Le schéma de traduction est le suivant :

créer pour chaque structure d'utilisation un appel TRNON ou START selon le cas, au niveau de la tâche INIT.

Les tâches elles-mêmes ont une structure simple, le compilateur se contentant de rajouter un CALL EXIT à la fin de chaque tâche

Tâche INIT



Remarques :

1) Nous avons donné l'architecture générale de toute l'application une fois toutes les structures traduites.

Cette architecture peut paraître lourde en égard au nombre de niveaux imposés. Il faut noter cependant que :

- dans la plupart des cas (expressions d'événements, partage d'événements) ce schéma est quasiment imposé.
- il peut faciliter, de par le découpage, certaines modifications :

supposons qu'il ne s'agit plus de l'expression e1 ou e2 qu'il faut implanter, mais e1 ou e3. La seule modification portera sur la tâche de détection et ne touchera ni le niveau physique, ni le niveau 3.

De plus, dans certain cas précis (événements non partagés) , l'implantation peut être simplifiée en reliant par exemple une tâche directement à un niveau d'IT.

Afin d'assurer une gestion correcte des occurrences d'événements, les tâches immédiates, de détection et de niveau 3 doivent être plus prioritaires que les tâches de l'application.

2) Dans cette implantation nous avons supposé la possibilité d'une partie sinon si dans une structure d'utilisation.

VII.7.- PRISE EN COMPTE DU TIME-OUT

Primitives utilisées :

- CALL SEVDEL (-) : signalisation différée d'une occurrence
- CALL ØFF (-) : suppression des appels, déjà enregistrés, à une tâche
- CALL INHIB (-) : suspension de l'éxécution d'une tâche. Seul l'opérateur peut décider sa reprise.

Le principe est alors de :

- compléter les tâches de niveau 3 en distinguant les cas fugitif et absence de Time-out et les cas Recent et chaque avec Time-out avant l'activation des tâches.
- compléter les tâches utilisateur par une signalisation de leur fin d'éxécution (événement interne affecté par le compilateur)
- générer une tâche de gestion du time-out qui "active" le délai et se met en attente de la fin du délai ou de la fin de la tâche utilisateur.

On aura :

1°) Tâche Niveau 3

avant d'activer une tâche :

- test si fugitif ou absence de Time-Out :

dans ce cas la tâche utilisateur est lancée sans plus

- si présence de time-out :

. récent : les activations précédentes de la tâches de Time-out sont suspendues. Le décompte se fait pour la nouvelle occurrence.

activation de la tâche time-out

activation de la tâche utilisateur

. chaque : activation de la tâche utilisateur

2°) Au niveau des tâches de l'application, il faut rajouter un événement de signalisation de fin de tâche. On aura

Tâche Ti

beg :

...

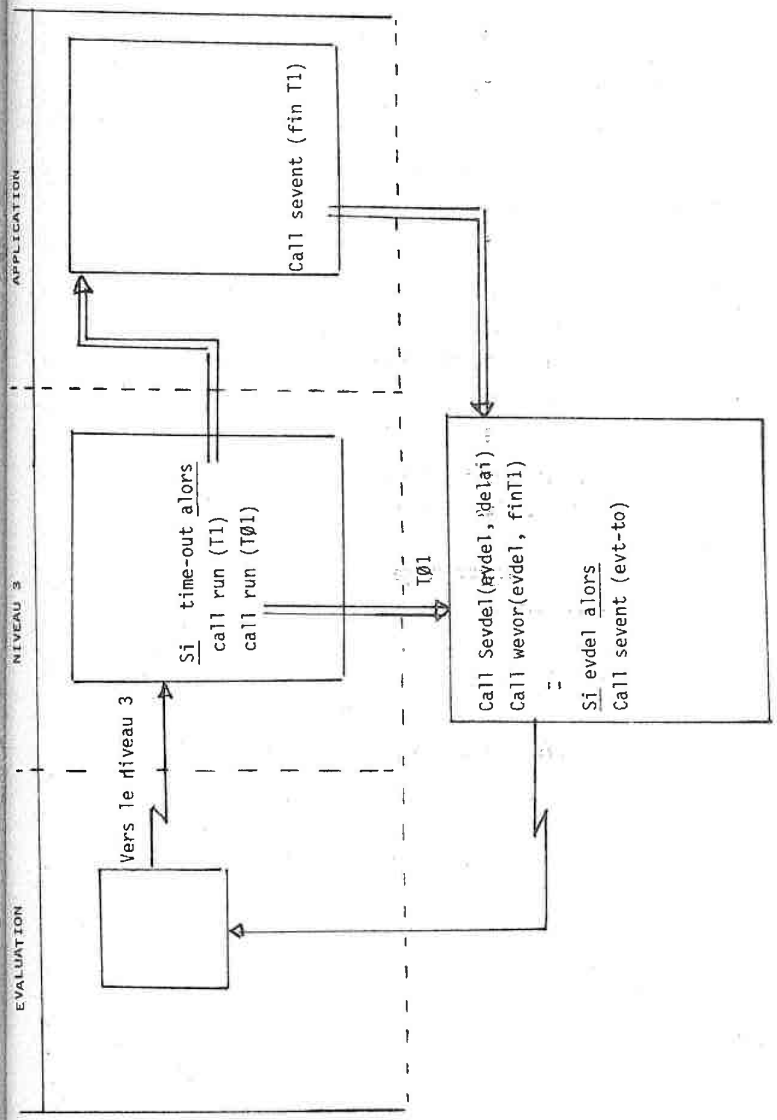
call sevent (evt fin-de-tâche)

pour une tâche traitant le récent exemplaire par exemple.

3°) Il faut enfin ajouter une tâche de prise en compte du time-out, de structure assez simple

```
Tâche    -- Time-out --
  (beg : call sevdel (evt-to, délai)
         call wevor (evt-to, evt-fin-de-tâche)
         call revent (evt-to)
         call revent (evt-fin-de-tâche)
         si evt-fin-de-tâche alors aller à fin
         -- sinon c'est evt-to--
--suspension tâche- call  INHIB (n° tâche positive---)
                   call  SEVENT (evt-time-out)
-- pour permettre aux tâches de reprise de s'éxécuter normalement --
-- cet appel s'adresse à une tâche d'évaluation --
       fin :  CALL exit
fin de tâche
```

On aura le schéma global de traitement du time-out suivant :



VII.8.- STRUCTURE DE LA TACHE INIT

Dans cette tâche on retrouvera :

- les activations de toutes les tâches périodiques liées au temps
- les activations de toutes les tâches cycliques liées à l'évaluation,

mais également :

- la mise en place de certaines structures de données (notamment la table TABTK3) nécessaires à l'éxécution des tâches de service et issue de la compilation.
- la réservation de la zone mémoire destinée à contenir les variables ve et i nécessaires à la gestion du partage des événements.

VII.9.- CONCLUSION

Cette réalisation peut à notre avis être vue de deux façons :

a) Comme compilation simple d'un langage de programmation en un langage intermédiaire donné. Cet aspect permet de préciser la sémantique du langage défini et considère le côté programmation du langage.

b) Comme traduction d'une spécification dans un langage donné en vue d'une simulation sous un système donné. Ceci permet, en quelque sorte, de rendre "éxécutable" une spécification et autorise une vérification du respect des spécifications pour une implantation particulière (choix des tâches, structures des tâches...). Cet aspect nous semble très important car il apporte une aide appréciable à la conception d'ACPI.

CHAPITRE VIII :

PROBLÈMES DE LA RÉPARTITION

VIII.1.- INTRODUCTION

Si la sémantique de tous les opérateurs définis (ET, OU ...) doit être indépendante de toute implantation, il n'en demeure pas moins qu'en relation avec la classe de problèmes traités, on peut être amené à faire certaines hypothèses facilitant l'implantation.

Il en va ainsi des opérateurs AVANT, APRES ou SIM pour lesquels le délai de transmission dans un contexte réparti peut être déterminant.

Pour respecter, donc, la sémantique de ces opérateurs dans une implantation donnée, ou peut être amené à implanter des algorithmes d'ordonnancement d'événements comme celui proposé dans [LAMP 78] par exemple. Cette solution peut dans certains cas être nécessaire ; elle peut également être très pénalisante.

On peut alors adopter une solution intermédiaire dans laquelle on tient compte de la classe de problèmes traités.

Dans notre cas par exemple, il s'agit typiquement d'applications implantées sur réseaux locaux et pour lesquelles les délais de transmission peuvent être considérés comme nuls.

Dans ce chapitre donc, partant des problèmes d'évaluation d'expressions d'événements liés à la répartition, nous essayerons de fixer des hypothèses d'implantation qui nous paraissent "raisonnables" et de proposer une solution au problème de l'évaluation d'expressions "réparties" dans le cadre SYGARE ([THOM 80], rappels dans l'annexe I).

VIII.2.- PROBLEMES LIES A LA REPARTITION

Le problème le plus important, à notre sens, lié à la répartition est celui de l'indépendance des références de temps sur les différents sites constitutifs d'un réseau. Comme souligné dans [LAMP 78] et en présence de délais de transmission non

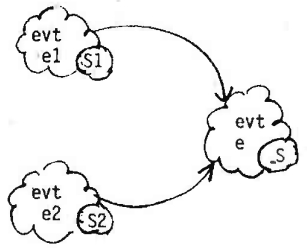
négligeables, l'ordre des occurrences peut être modifié. L'ordre des occurrences sur un réseau est donc un ordre partiel qu'il faut parfois transformer en ordre total en introduisant des horloges logiques comme le fait Lamport par exemple.

Pour nous, se pose un double problème :

- pour certains opérateurs (avant ...) il est impératif de conserver un ordre total sur les occurrences d'événements.
- pour d'autres opérateurs (SIM...) une expression risque d'être fausse uniquement à cause des délais de transmission, alors que dans l'"absolu" elle est vraie.

Dans les deux cas cependant, le point essentiel consiste en la détection des occurrences produites et ce d'une manière absolue .

Dans le cas qui sera plus vraisemblablement le nôtre (travail sur réseaux locaux généralement admis en milieu industriel) où il existe des horloges synchronisées (même heure à la dérive près) par recalage périodique, le problème des délais de transmission peut demeurer avec la question importante de savoir à quel niveau est faite l'évaluation des expressions d'événements. Si on prend l'exemple suivant :



trois sites S1, S2 et S sur lesquels sont définis trois événements e1, e2 et e respectivement avec e : <qual-1>e1 avant <qual-2>e2
(e est donc composé)

plusieurs questions liées à la répartition surviennent :

- comment sont implantées les expressions d'événements réparties : chaque événement sur son site d'implantation ? ou autrement ...
- dans quel site se fait l'évaluation d'une expression répartie ? Ce qui pose le problème d'un évaluateur réparti.
- A quel niveau est décidé l'activation des tâches qui sont elles-mêmes réparties ? A quel site appartient une tâche ?

et enfin :

- quelles sont les hypothèses de travail considérées ?

VIII.3.- HYPOTHESES DE TRAVAIL

On se place dans le cadre d'un réseau local sur lequel il n'y a pas nécessairement une horloge unique.

Chacun des sites peut avoir sa propre horloge.

Une synchronisation des horloges est nécessaire périodiquement à cause de la dérive des différentes horloges.

Nous ne nous intéressons pas à la façon dont cette synchronisation est faite. On la suppose réalisée à un autre niveau et l'on considère un réseau synchronisé.

On suppose qu'une communication par message par exemple existe avec les protocoles adéquats entre les différents sites.

Ces messages peuvent être le support d'occurrences d'événements transitant entre les différents sites.

Selon le site où est faite l'évaluation, le délai de transmission peut jouer un rôle dans la détermination du résultat. Ce délai est toutefois considéré en règle générale comme étant nul. (contexte de réseau local).

VIII.4.- EXPRESSIONS REPARTIES D'EVENEMENTS : EVALUATION

D'une manière générale, l'idée qui prévaut ici, est d'essayer de prendre en compte les occurrences d'un événement le plus près possible de sa source, c'est-à-dire de son lieu d'implantation.

La prise en compte d'un événement consiste en son acceptation par l'évaluateur.

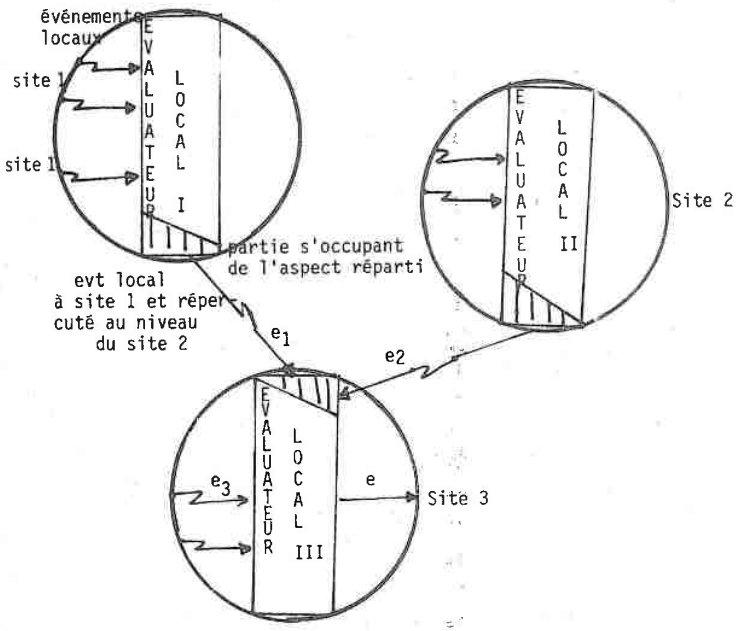
Si cette idée est simple à comprendre s'agissant d'un événement simple, il en va autrement lorsque l'on considère les événements composés. Si l'événement composé est entièrement local à un site, il est pris en compte comme un événement simple du point de vue répartition.

Si un événement composé est réparti, c'est-à-dire appartient à plusieurs sites à la fois, il se pose alors le problème du site d'évaluation : que veut dire dans ce cas "le plus près possible de sa source" ?

Il nous a paru judicieux alors de responsabiliser un seul site dans l'évaluation d'une expression. Le choix du site responsable n'a pas a priori d'importance, mais on peut retenir le site le moins "chargé" en nombre d'expressions à évaluer par exemple.

Il se posera certainement un problème de sécurité à l'évaluation et il faudra peut-être prévoir un site d'évaluation de secours. Tout se passe alors comme si chacun des sites disposait d'un évaluateur local chargé avant tout de la gestion des événements locaux mais qui peut être amené à dialoguer avec les autres sites pour évaluer un événement composé.

On a le schéma suivant :



L'événement e est défini par

e : e1 et e2 et e3 (par exemple).

La décision d'attribuer l'évaluation de e au site 3 s'est donc faite sur examen de sa "charge" et des assignations concernant les événements élémentaires composant l'expression.

Un changement d'assignation peut évidemment remettre en cause cette distribution. Cela peut se faire sans trop de dommage si

l'on considère que le nombre de changement d'assignations n'est pas très élevé, ce qui en principe est le cas.

En résumé nous avons :

- les événements élémentaires dont la localisation est déterminée par les opérations d'assignation.
- les événements composés dont la localisation dépend d'une décision du système selon des critères à fixer. La décision aurait pu également revenir au concepteur qui fixe le site d'évaluation.
- il existe un évaluateur local par site.
- le fonctionnement général est le suivant : sur arrivée d'une occurrence d'un événement au niveau d'un évaluateur donné, l'évaluateur local examine s'il s'agit d'une évaluation entièrement locale. Dans ce cas il l'effectue. Si l'événement est destiné à un ou plusieurs autres évaluateurs (un événement peut faire partie de plusieurs expressions à la fois) l'évaluateur se charge de le transmettre aux évaluateurs concernés selon une table de répartition des expressions.

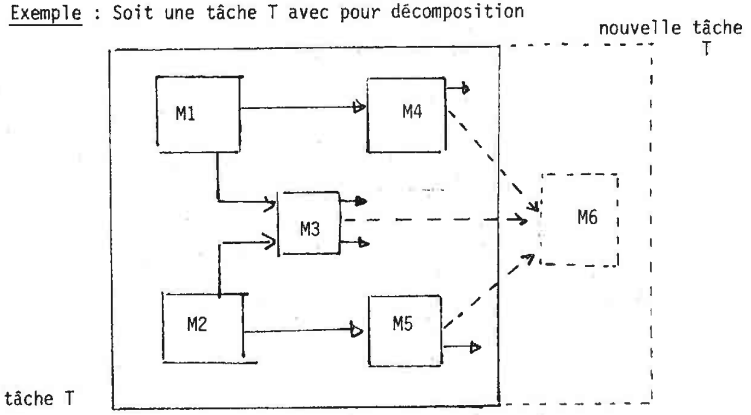
Pour les évenements "fin de tâche" et "fin de module", nous faisons l'hypothèse que chaque tâche peut être décomposée en unités plus petites qu'on appelle modules. Les modules constituant une tâche peuvent être répartis. Leur enchaînement est défini dans des structures de contrôle dites de niveau 2 comme dans le système SYGARE (cf. ANNEXE I).

La localisation des événements "fin de tâche" et "fin de module" se fait de la façon suivante : pour les événements "fin de module", aucun problème ne se pose étant donné qu'un module est entièrement localisé sur un site : l'événement appartient donc à ce site. Pour un événement "fin de tâche" le problème est

un peu plus délicat car la tâche est elle-même répartie par le biais de ses modules et elle ne se termine pas toujours par le même module (structure de niveau 2 conditionnelle, modules parallèles ..).

Dans ce cas on peut rajouter à chaque tâche un module dit module de localisation qui n'a pour d'autre rôle que de terminer toujours de la même façon et au même endroit une tâche. Ce module peut être rajouté à la compilation et localisé selon le principe de l'évaluation le moins chargé.

Exemple : Soit une tâche T avec pour décomposition

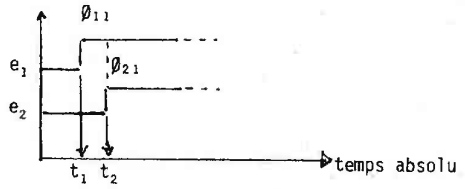


La tâche T se termine par deux modules parallèles M4 et M5.

L'adjonction d'un module de localisation M6 permet de résoudre le problème de la localisation.

Remarques

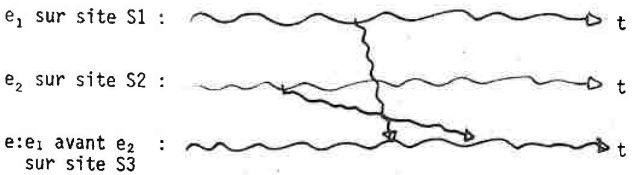
1) Dans le cas du SIM, l'évaluation d'une expression est positive lorsqu'il y a chevauchement des durées de vie d'occurrences composantes :



Avec le problème de la répartition et de la localisation de l'évaluation il est possible qu'entre le moment où $\emptyset_{11}$ est produite (t_1) et le moment où elle est reçue sur le site d'évaluation possédant e_2 par exemple, $\emptyset_{21}$ n'est plus vivante. Le résultat de l'évaluation sera évidemment négatif alors qu'en temps absolu il doit être positif. Pour cette raison, il est important de considérer que les délais de transmissions sont faibles par rapport aux constantes de temps du procédé.

2) Dans le cas du AVANT, c'est le site de production d'une occurrence qui est responsable de la datation de l'occurrence. Le site d'évaluation, ainsi, dispose de dates réelles indépendantes du délai de transmission.

Si les délais de transmission ne sont pas considérés comme négligeables, nous pouvons avoir le phénomène suivant :



et une évaluation par anticipation d'une telle expression donnerait des résultats aberrants. Dans de telles hypothèses, il faut attendre l'arrivée des deux occurrences concernées avant d'évaluer l'expression.

L'implantation d'horloges logiques [LAMP 78] peut être nécessaire dans ce cas pour disposer d'une même base de temps (simulation d'un temps absolu).

Toutes ces opérations sur les événements supposent que l'on est capable de les ordonner de manière absolue.

Il faut être prudent dans la manière d'ordonner. En effet, un mécanisme du type horloge logique établit bien un ordre total mais il s'agit d'un ordre arbitraire. Donc, il permet l'évaluation de toutes les expressions qui reposent sur un tel ordre. Le résultat de l'évaluation peut toutefois être contraire à la réalité.

On retrouve exactement le même phénomène que celui connu sur toute machine : à savoir que deux interruptions qui se sont produites entre deux instants d'observation sont considérées comme simultanées et que leur ordre de prise en compte (priorité) n'a rien à voir avec leur ordre d'arrivée.

Tout mécanisme de datation supposant une discrétisation du temps oblige à considérer une telle possibilité de simultanéité.

Selon les cas, le concepteur doit décider s'il permet ou non un ordonnancement arbitraire. S'il ne le permet pas, on devrait définir une exception indiquant qu'une évaluation d'expression d'événements est indécidable.

Cette exception, à condition qu'elle soit définie de façon analogue à celle que nous avons déjà proposée, pourrait être prise en compte et traitée.

On peut donner l'exemple d'un procédé dans lequel e_1 et e_2 représentent des arrivées de pièces par exemple. L'ordre d'arrivée des occurrences de e_1 et e_2 est important pour la fabrication et selon qu'une occurrence de e_1 arrive avant une occurrence de e_2 ou inversement, des opérations différentes sont accomplies :

e_1 avant e_2 → opération 1
e_2 avant e_1 → réordonner les pièces puis opération 1

Dans ce cas, s'il y a "simultanéité" d'arrivée d'occurrence, il peut être dangereux d'appliquer un ordre arbitraire et il est préférable de détecter une exception.

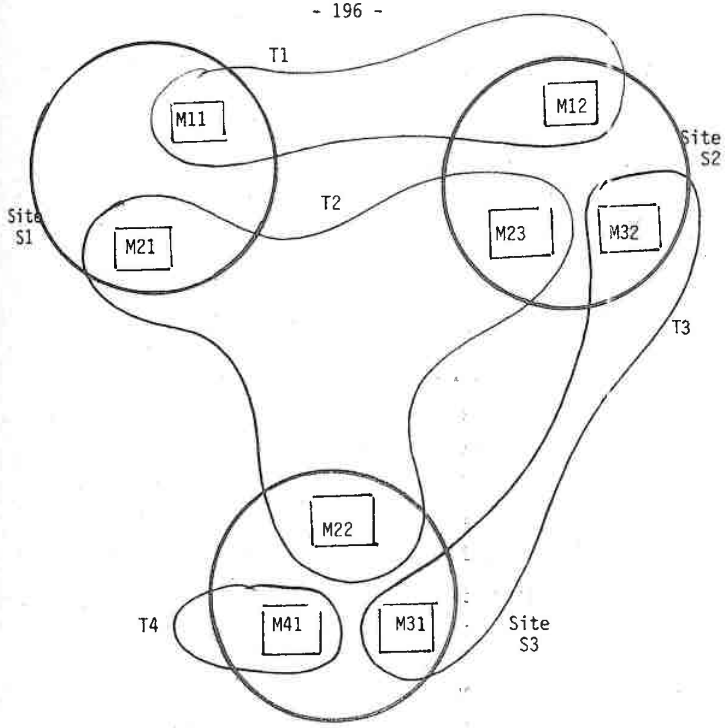
VIII.5.- LOCALISATION DES TÂCHES - ACTIVATION DES TÂCHES

Dans ce paragraphe, l'aspect important à mettre en évidence est, sans doute, constitué par toutes les relations pouvant exister entre l'évaluation répartie (ensemble des évaluateurs locaux) et le scheduler.

Que se passe-t-il au juste quand une occurrence d'un événement est produite ? Comment réagit le scheduler ? Quelles décisions peut-il prendre et comment ? Quelles sont les structures de donnée dont il dispose pour la prise de décision ? et enfin, où le scheduler est-il situé ?

VIII.5.1.- Localisation des tâches

Considérons le schéma suivant, représentant un exemple de répartition de tâches via les modules constitutifs :



avec les structures de niveau 2 suivantes :

T1 : M_{11}-M_{12}
T2 : (M_{21} // M_{22})-M_{23}
T3 : M_{31}-M_{32}
T4 : M_{41}

L'exécution de ces tâches requiert la participation de plusieurs sites et une table d'enchaînement [NONN 78] est nécessaire pour en assurer une exécution correcte. La table d'enchaînement implante donc les structures de niveau 2. C'est donc cette table qui indique par"quel bout il faut prendre une tâche" et permet en

quelque sorte sa localisation. A cette table d'enchaînement est associé un interprète des structures de niveau 2, chargé de réaliser les enchaînements entre les différents modules constitutifs d'une tâche.

L'activation d'une tâche, quant à elle, indépendamment des enchaînements, est le fait du Scheduler.

D'un point de vue répartition, nous décidons de "domicilier" une tâche sur le même site que l'événement auquel elle est attachée.

La domiciliation d'une tâche consiste à responsabiliser un site pour l'activation et la prise en compte de la fin d'exécution d'une tâche selon un schéma de répartition que nous allons examiner.

VIII.5.2.- Schéma de répartition et activation

Au niveau de chaque site nous trouvons :

- une copie de la table des enchaînements,
- un évaluateur local tel que déjà défini,
- un scheduler local chargé de la régulation de toutes les tâches domiciliées sur ce site,
- un interprète des structures de niveau 2 local utilisé par le scheduler pour l'exécution des tâches,
- une copie de la table de répartition des expressions d'événements.

De plus, le choix du site d'évaluation d'une expression détermine la domiciliation d'une tâche.

Une tâche étant liée à un événement et un seul, elle est domiciliée sur un site et un seul.

L'événement "fin de tâche" est alors naturellement attaché à ce site également.

Le fonctionnement général au niveau d'un site peut alors être décrit de la façon suivante :

- L'évaluateur local, après évaluation d'une expression, produit une occurrence d'un événement logique et avertit le scheduler local. On remarque à ce niveau là, qu'étant donné le choix de répartition fait (domiciliation des tâches sur le même site que l'événement attaché), tous les événements logiques implantés par un évaluateur local sont traités par le scheduler local. Dans le cas où l'événement n'est pas domicilié sur ce site, il est représenté par l'évaluateur au niveau des sites concernés sans que le scheduler ne le sache.

- Le scheduler local peut alors, selon les conditions associées à la tâche décider de son activation ou non.

Le rôle du scheduler local consiste à :

- activer les tâches quand les conditions sont réunies.
- suspendre les tâches lorsque la demande en est faite.
- dialoguer avec l'interprète des structures de niveau 2 pour la prise en compte de l'enchaînement des modules.
- traiter l'événement "fin de tâche" pour la relance éventuelle de la tâche (chaque...).

Le scheduler apparaît alors comme le module implantant les structures d'utilisation.

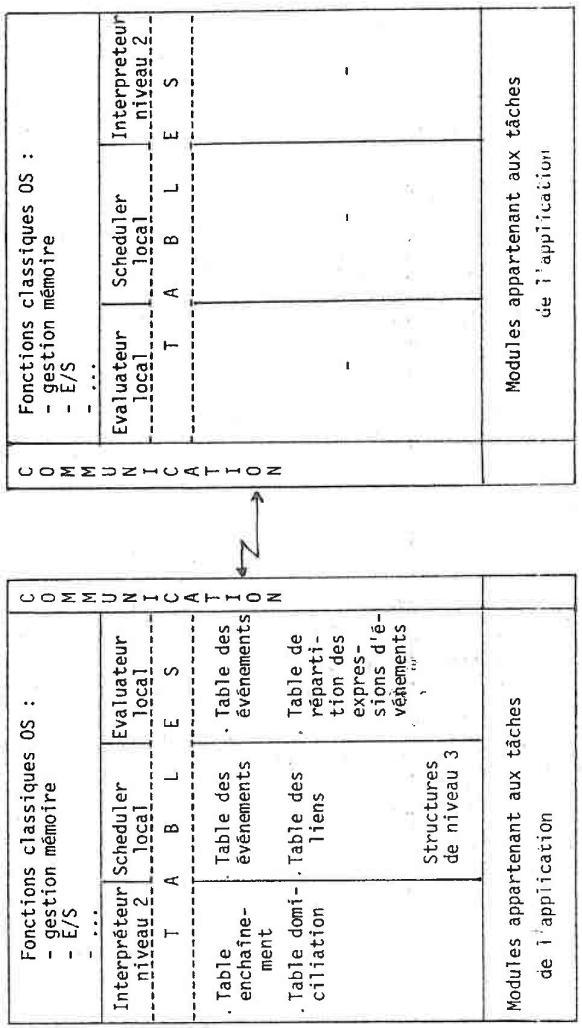
L'enchaînement des modules d'une tâche peut être assuré par chacun des sites du réseau étant donné que chacun des sites dispose de la table d'enchaînement. Celle-ci a alors un rôle similaire à une table de routage. C'est uniquement la fin de tâche, en fait, qui est importante. C'est à ce moment-là en effet, qu'il faut retourner au site domiciliateur. Ce retour peut facilement se faire grâce à une table de domiciliation qui indique pour chacune des tâches, le site domiciliateur. Chacun des sites dispose d'une copie de cette table.

Remarque

Plusieurs tables étant dupliquées sur chacun des sites, nous serons confrontés fatalement au problème de mise-à-jour de copies multiples sur un réseau. Cela étant, nous retenons toutefois le fait important que les modifications d'une ACPI sont faites à des instants bien déterminés et sont relativement peu fréquents. On peut de ce fait espérer que le nombre de mise-à-jour des copies de table n'est pas très important et peut se faire off-line.

En conclusion de ce chapitre, nous donnons un schéma global du système réparti avec les structures de données que nous avons jugées importantes.

Notons que l'on essaye par ce schéma de répartition de prendre un maximum de décisions à un niveau local. C'est ce qui explique la domiciliation des tâches, les copies de tables et l'existence d'organes de décisions locaux : sheduler, évaluateur et interprète niveau 2.



SCHEMA GLOBAL

CONCLUSION

Ce travail est une contribution à la définition d'un langage de spécification d'application en contrôle de procédés industriels basé sur le concept d'événement qui peut également être vu comme un langage de programmation. Cette dualité a été possible grâce à une expression statique des synchronisations, expression du type :

sur événement faire tâche.

Une telle expression peut, en effet, être vue comme une spécification d'un comportement souhaité de l'application ou alors comme phrase de synchronisation d'un langage de programmation.

La spécification comme la programmation s'appuient sur :

- les événements qui représentent l'évolution du procédé.
- les tâches qui représentent les actions à entreprendre.
- les liens entre les événements et les tâches qui expriment le comportement de l'application (ou les synchronisations entre le procédé et l'application).

Du point de vue de la spécification, nous avons souligné le rôle central tenu par les événements : ils permettent d'exprimer l'évolution du procédé et l'expression de comportement de l'application se fait par rapport à eux.

Nous avons été, de ce fait, conduit à préciser un certain nombre de notions relatives au concept d'événement notamment en ce qui concerne les aspects production, détection et utilisation avec la définition d'opérations précises attachées à chacun de ces centres d'intérêt.

Ceci contribue largement à tendre vers une spécification non ambiguë et notre langage apparaît comme un langage de manipulation d'événements orientés vers la spécification d'ACPI.

Un certain nombre d'avantages concernant l'utilisation du concept d'événement dans une spécification d'ACPI ont été recensés :

- Le concept d'événement est un concept naturel : il nous semble que le mode de raisonnement adopté face à un problème temps réel est fondamentalement basé sur le concept de changement d'état, donc d'événement.

- Cet aspect naturel tend donc à rapprocher l'expression d'un cahier des charges de l'expression d'une spécification. Ceci facilite grandement la spécification et permet une clarification certaine du cahier des charges. Dans ce sens, la spécification constitue un document précieux car elle représente une expression claire et non ambiguë du cahier des charges.

- Ce concept d'événement permet, comme nous l'avons montré à travers les expressions d'événements, de prendre en compte toutes les évolutions du procédé.

- Il encourage, au moment de la spécification, un recensement exhaustif des cas à traiter, ce qui ne peut qu'accroître la fiabilité du logiciel produit.

- De plus, l'utilisation du concept d'événement comme entité synchronisante mais aussi véritablement comme une ressource pour laquelle nous avons défini un certain nombre de règles de partages et des opérations précises de manipulation confère à notre langage une grande puissance d'expression.

La définition de ces règles et de ces opérations traduit le désir d'exprimer le plus fidèlement possible et dès le stade de spécification, le comportement de l'application à partir du cahier des charges.

- La spécification est progressive dans ce sens que, partant d'une première spécification d'une solution, on peut, par regroupements successifs d'actions, aboutir à un niveau de découpage souhaité.

- Le concept d'événement permet une prise en compte aisée

et surtout uniforme, dès la spécification, aussi bien du fonctionnement normal que du fonctionnement anormal d'une application : le même concept permet de spécifier aussi bien l'occurrence d'une panne que la fermeture normale d'une vanne par exemple.

D'un point de vue programmation, notons également que le concept d'événement est un concept largement répandu dans les langages et systèmes temps réel.

Ceci tend à notre avis à rapprocher, d'une certaine façon, le niveau spécification et le niveau programmation d'une application. Pour valider cet aspect nous avons réalisé un traducteur qui permet de générer une application multi-tâches en FORTRAN-TR à partir des spécifications proposées.

Ce faisant, nous nous sommes rendus compte que l'application générée pouvait parfaitement servir de base à une simulation pour la vérification du respect des spécifications plutôt que d'avoir à écrire un simulateur. Ceci nous donne alors l'avantage de disposer d'une spécification "exécutable" dont l'intérêt a été déjà souligné dans [ZAVE 82].

Avant d'aborder le traditionnel regard vers l'avenir, après avoir passé en revue le travail déjà effectué, soulignons qu'un point important de notre démarche a été également de montrer la complémentarité du concept d'événement et du concept de communication dans la spécification d'applications en temps réel, au moment où une certaine tendance à une spécification en termes d'entités communicantes voit le jour.

Le travail futur peut être envisagé sous plusieurs aspects :

1) Aspects méthodologiques

a) Guider plus la recherche des événements à prendre en compte

et les actions à entreprendre en s'appuyant sur un modèle théorique (automate d'états finis, logique temporelle...) à l'instar de [VIT 81].

b) La spécification proposée peut comporter plusieurs étapes : une première étape où les événements et actions sont complètement énumérés et une deuxième étape permettant d'aboutir par regroupements successifs des événements et actions à une représentation plus synthétique de l'application (par exemple sous forme d'entités communicantes).

Dans notre travail seule la première étape est définie. Il serait intéressant d'examiner les possibilités de systématiser ce regroupement. Ceci peut être fait conjointement avec le point a).

c) Développer l'aspect type abstrait des événements déjà ébauché dans ce travail.

d) Tendre vers une spécification plus formelle.

2) Aspects aide à la conception

a) Il y a un premier aspect déjà évoqué et concernant la simulation. Il y a là certainement encore beaucoup de travail à réaliser pour aboutir à un outil efficace d'aide à la conception d'application en temps réel, ce qui permet également de prendre en compte les problèmes de dimensionnement d'une application.

b) Lorsque l'aspect programmation de notre langage est considéré et étant donné que les points de synchronisation et communication sont définis séparément du texte des programmes,

on peut également s'orienter ver des méthodes de preuve de programmes basées sur la logique temporelle [BERN & al 81].

3) Aspects implantation

Une réalisation centralisée a été présentée.

Il reste bien entendu, à valider complètement les idées proposées par une réalisation répartie.

ANNEXE I

Rappels sur SYGARE [THOM 80]

• Philosophie :

Conçu comme un SYstème de conception et de Gestions d'Applications en temps réel et REparties, SYGARE admet comme prinpales caractéristiques :

- Une expression des synchronisations situées en dehors du texte de programmes.

- Une expression statique des synchronisations dans ce sens que cette expression est complètement indépendante de toute exécution.
De cette façon, le langage d'expression des synchronisations est plutôt de type déclaratif.

Ces deux premiers points permettent de prendre en compte facilement les problèmes d'évolutivité.

- La communication entre les différents composants d'une application est vue sous la forme de la spécification d'un flux de données, explicitement décrit par le biais de connexions, avec des synchronisations implicites à ce niveau. Les entités communicantes de SYGARE sont synchronisées sur arrivée de toutes les données dont elles ont besoin. En ce sens, ces entités sont pilotées par les données un peu comme dans [DEN 75].

- L'expression des synchronisations est asynchrone dans ce sens qu'aucune idée d'attente explicite n'est spécifiée. Le flot de contrôle est exprimé sous la forme d'un flux d'occurrences d'événements traitées de manière asynchrone.

- Les problèmes d'implantation sont repoussés le plus loin possible (définition d'un langage d'implantation) de façon à avoir une description d'une application la plus indépendante possible

d'une réalisation particulière.

- La répartition enfin est prise en compte dès le départ de par la structuration-même proposée.

• Eléments constitutifs d'une application SYGARE

Une application est vue comme un ensemble de tâches parallèles coopérant pour la réalisation d'un certain objectif.

Pour des raisons de respect de contraintes de temps par exploitation du parallélisme vrai, les tâches sont de composées en entités communicantes appelées modules. Un module exprime un algorithme de traitement (régulation par exemple) et est perçu comme un automate séquentiel élaborant des sorties à partir d'entrées définies.

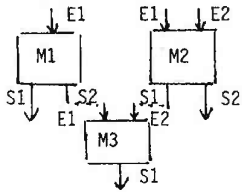
Le module constitue également l'unité de répartition.

La communication intermodule se fait par l'intermédiaire de canaux virtuels reliant une ou plusieurs entrées à une ou plusieurs sorties appelées données transmissibles.

Rien n'est dit, au moment de la description, sur l'implantation de ces données transmissibles : ce peut être une véritable voie physique de communication comme une simple mémoire partagée.

Les protocoles d'utilisation de ces données sont définies au niveau d'une structure dite de connexion, véritable schéma connecteur qui rend les connexions inter-modules indépendantes des modules eux-mêmes. Il apparaît finalement qu'un module n'est connu que par ses entrées et ses sorties.

Exemple de connexion :



Les structures de connexion s'expriment par :

S2.M1 → $\mathcal{D}_1$

$\mathcal{D}_1$ → E1.M3 récent exemplaire sans consommation

S1.M2 → $\mathcal{D}_2$

$\mathcal{D}_2$ → E2.M3 ancien exemplaire avec consommation

.....

$\mathcal{D}_1$, $\mathcal{D}_2$ représentent les données transmissibles.

On voit apparaître les protocoles d'utilisation sous la forme :

- de l'exemplaire de la donnée concerné : tous (ancien) ou le plus récent.
- de la consommation (prélèvement physique) ou non (consultation) de l'exemplaire.

En cas de partage d'une donnée transmissible et si des conflits peuvent survenir, une opération de duplication des exemplaires de la donnée est définie.

Le prélèvement des exemplaires se fait en FIFO.

Les synchronisations entre modules, relatives aux données transmissibles sont du type producteur-consommateur. Une description détaillée de l'aspect synchronisation peut être trouvée dans [DER & al 79]. Cette vue est également à rapprocher de celle exposée dans [COH 80].

Les constituants d'une application étant définis, il reste à préciser les enchaînements des modules et les liens entre les événements et les tâches.

Ce sont les différents niveaux de SYGARE.

Différents niveaux de SYGARE

- Le niveau I est le niveau d'expression algorithmique des modules : il constitue en fait le niveau de programmation d'un module. Nous rappelons qu'un module est strictement séquentiel.

- Le niveau II constitue le niveau d'expression des enchaînements entre modules d'une même tâche.

Dans les enchaînements on peut exprimer le parallélisme et la séquentialité.

Exemple

Tâche T1 : (M1 // M2) ; M3
pour l'exemple précédent.

- Le niveau III représente l'expression statique du parallélisme définie en termes de relations entre les événements et les tâches.

L'expression est du type

sur<evt>faire<tâche>

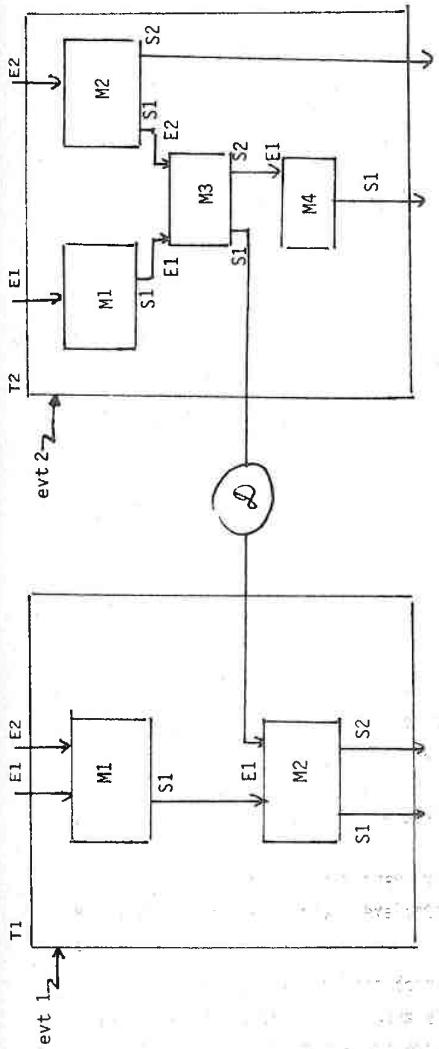
Exemple

sur evt avec consommation faire T1
récent explaire

où evt représente le nom d'un événement défini par ailleurs.

On remarquera la similitude existant entre cette expression et une expression de connexion. Un des points forts de SYGARE, en effet, est d'unifier les vues que l'on peut avoir d'un événement et d'une donnée. Les deux problèmes concernent en fait l'utilisation précise qui est faite soit des exemplaires d'une donnée soit des occurrences d'un événement et sont donc de même nature.

Un exemple de découpage peut être trouvé dans la page suivante.



Niveau 1 : Déclaration de M1.T1, M2.T1, M1.T2, M2.T2, M3.T2, M4.T2

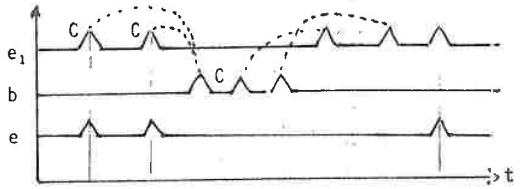
Niveau 2 : Tâche T1 : M1 ; M2
Tâche T2 : (M1 // M2) ; M3 ; M4

Niveau 3 : sur evt 1 avec cons faire T1
rct explaire
sur evt 2 sans cons faire T2
ancien explaire

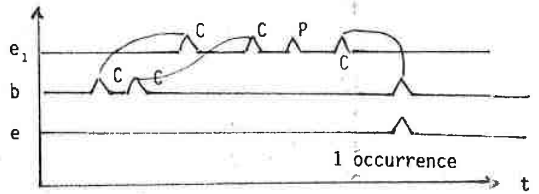
Connexions : voir le schéma

ANNEXE II

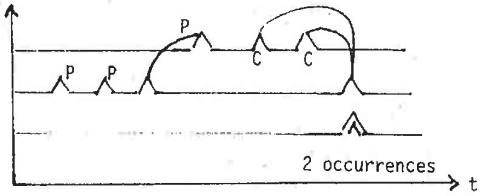
1) e : e_1 avant-a chaque b



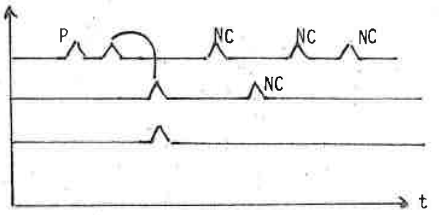
2) e : récent e_1 avant-s chaque b



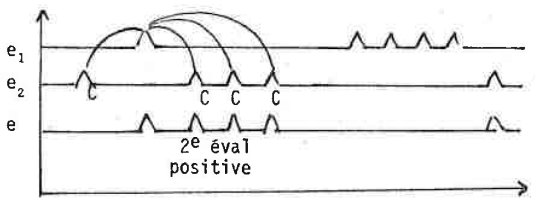
3) e : chaque e_1 avant-s récent b



4) e : récent e avant-s j^e b



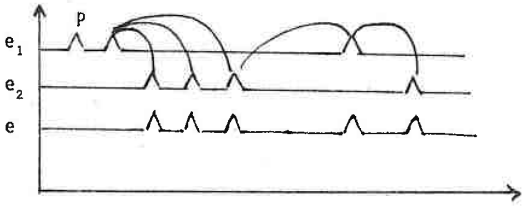
5) e : chaque sans cons e_1 et chaque avec cons e_2



L'absence de consommation de e_1 rend e équivalent à e_2 dès la 2ème évaluation positive.

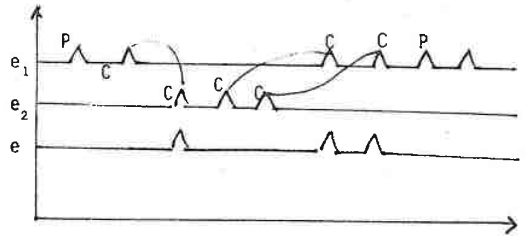
On peut l'exprimer par (e_2 après 1^{er} e_1) ou 1^{er} e_1.

6) e : récent sans cons e_1 et chaque sans cons e_2

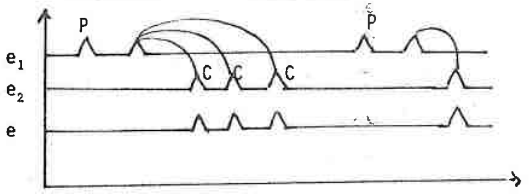


équivalent à (e_1 ou e_2) après (1^{er} e_1 ou 1^{er} e_2)

7) e : récent avec cons e_1 et chaque avec cons e_2

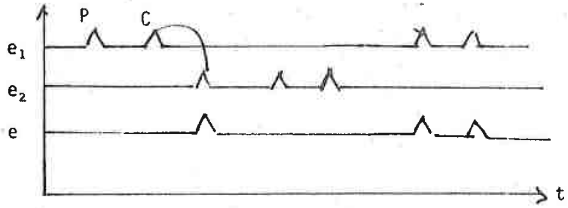


8) e : récent sans cons e_1 et chaque avec cons e_2



équivalent au cas 5). Cela est dûà l'absence de consommation de e_1.

9) e : récent avec cons e_1 et chaque sans cons e_2



e est équivalent à e_1 dès la 2ème évaluation positive.

≡ (e_1 après 1^{er} e_2) ou (1^{er} e_2).

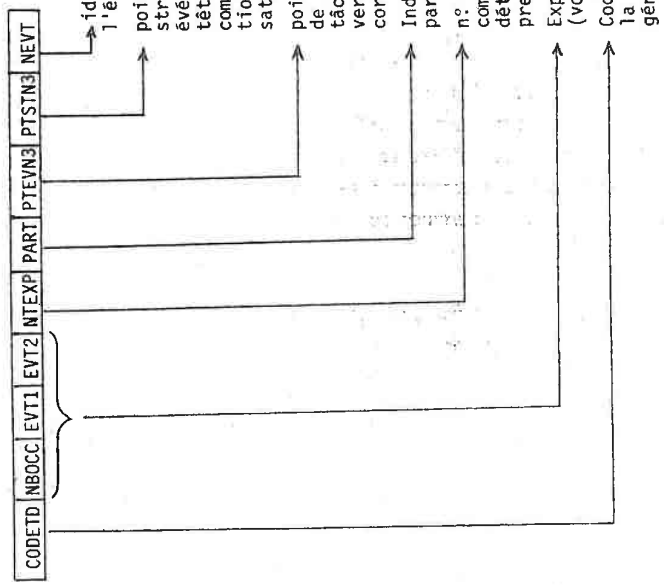
ANNEXE III

1) Table des expressions d'événements : TAB EXP

- 1 poste par événement déclaré quel que soit son type.
- complétée lors de la compilation de la partie déclaration d'évenements.

Cette table sert à générer les tâches de détection des occurrences.

Structure



Représentation des expressions

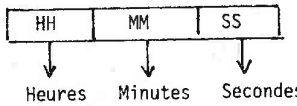
EVT1 et EVT2 représentent selon le cas des pointeurs vers
- TAB ASS : table des assignations
- TABTIME : table des expressions liées au temps

ou encore des valeurs lorsque l'expression le permet.

Code TD	NBOCC	EVT1	EVT2	Expression représentée
0	/	/	/	Evénement simple
1	j^ième	E1	B	E1 avant-a j^ième B
2	j^ième	E1	B	E1 avant-S j^ième B
3	/	E1	B	E1 avant-S chaque B
4	/	E1	B	" avant-A "
5	/	E1	B	E1 avant-S récent B
6	/	E1	B	E1 avant-A "
7	jième	E1	B	E1 après jième B
8	/	E1	B	" " chaque B
9	/	E1	B	" " récent B
10	/	E1	E2	chaque E1 avec cons ET chaque E2 avec cons
11	/	E1	E2	récent " " récent "
12	/	E1	E2	chaque E1 sans cons ET récent E2 avec cons
*13	---	---	---	----------
14	/	E1	E2	E1 OU E2
15	délai	E1	unité	<délai> après E1
16	Nbre	E1	/	<nbre> occurrences de E1
17	HH	MM	SS	<heure>
18	vers TABTIME	Δt	unité	chaque Δt après<heure>
19	vers TABTIME	Δt	unité	chaque Δt avant<heure>
20	Δt	E1	unité	Evénement simple de time-out

* tous les cas du ET ne sont pas représentés
E1 et B sont des pointeurs vers TBASS
HH, MM et SS sont des valeurs.

La table TABTIME se présente sous la forme



Un poste de type 20 est créé à chaque rencontre, lors de la compilation d'une structure de niveau 3, d'une définition de délai de prise en compte.

2) Tables des structures d'utilisation

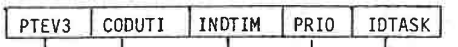
Un ensemble de trois tables parallèles sont créées par le compilateur lors de l'analyse syntaxique d'une structure d'utilisation :

TABTK3 : table créée à la compilation mais nécessaire à l'exécution d'une tâche de service dite de niveau 3, tâche assurant la liaison entre les tâches d'évaluation des expressions d'événements et les tâches de l'application. Les tâches de niveau 3 assurent en particulier le partage.

TABNIV3 : Permet de générer l'"enveloppe" nécessaire à une tâche de l'application pour assurer une liaison correcte avec les tâches dites de niveau 3 et surtout de garantir un partage correct des événements utilisés.

TABTØ : table assurant la génération des tâches de service nécessaires à la gestion des time-out .

TAB NIV 3



identificateur de la tâche utilisateur
(∈ à l'application)

priorité de la tâche utilisateur

≠ 0 pointeur vers TABTIME
Indique que l'on utilise une expression du type chaque Evt avant<heure>; cette expression demande une génération de code différente des autres expressions.

= 24 structure avec avec cons

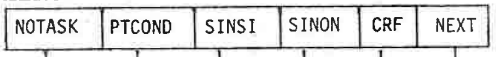
= 25 structure avec sans cons

= 26 prise en compte fugitive

permet la génération de séquence de partage d'événements

pointeur vers la table TEVN3. Pointeur inverse de PTTASK de TEVN3.

TABTK3



pointeur vers le prochain poste utilisant le même événement (poste issu de la compilation d'une autre structure d'utilisation). Ce poste est utile pour une tâche de niveau 3 en cas de partage d'événement. Il sert à la recherche de toutes les tâches à activer.

= 3 ⇒ structure d'utilisation avec chaque
= 2 ⇒ " " récent
= 1 ⇒ " " fugitive
Ce poste est utile lors de l'activation des tâches car celle-ci ne se fait pas de la même façon dans les trois cas étant donné l'implantation.

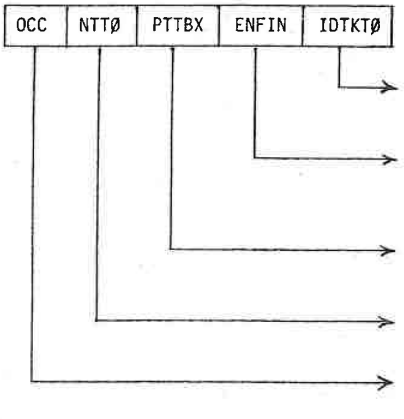
= 1 ⇒ il s'agit d'une partie SINON

= 1 ⇒ il s'agit d'une partie SI ou SINON SI

vers une table des conditions.

n° interne de la tâche de l'application concernée.
Affecté par le compilateur.

TABTØ



identificateur de la tâche de service assurant la gestion du time-out considéré. Cet identificateur est donné par le compilateur.

n° interne d'événement affecté par le compilateur pour compléter la tâche de l'application par la signalisation nécessaire de la fin d'exécution.

pointeur vers la table des expressions TABEXP vers le poste contenant la déclaration de l'événement d'exception associé.

n° interne de tâche affecté par le compilateur à la tâche de gestion du time-out.

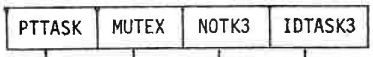
= 0 ⇒ poste de TABTØ inoccupé
= 1 ⇒ " " occupé

3) Table assurant la génération des tâches de service assurant la gestion du partage des événements (tâche de niveau 3) : TEVN3.

Cette table est créée lors de la composition des structures d'utilisation.

Structure

TEVN3



identificateur tâche de niveau 3 affecté par le compilateur.

n° interne de tâche de niveau 3 affecté par le compilateur.

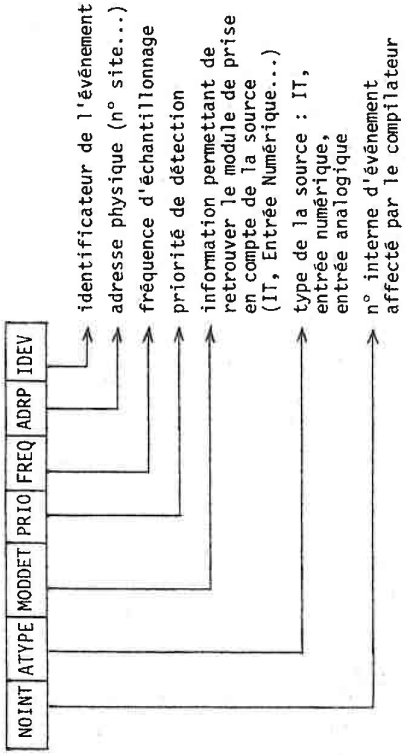
n° de sémaphore d'exclusion mutuelle affecté par le compilateur et permettant une gestion correcte des occurrences d'événements (notamment entre la production et l'utilisation).

Tête de chaîne vers les structures d'utilisation utilisant cet événement (vers TABTK3). Ce pointeur affecté à une tâche de niveau 3 pour le parcours de TABTK3 lors de la prise en compte d'une occurrence. Comme une tâche de niveau 3 est attachée à une tâche de détection, ce pointeur concerne bien toute la liste des tâches partageant cet événement.

4) Table d'assignations : TABASS

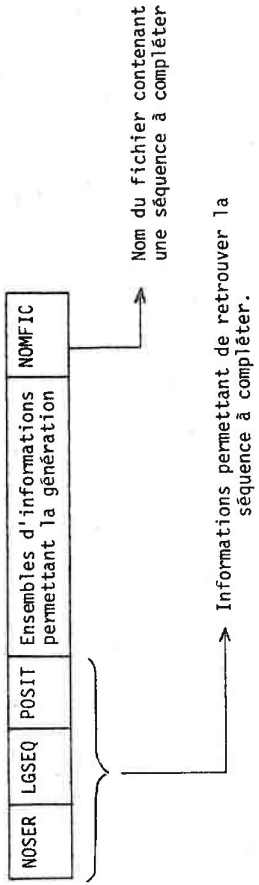
Cette table est créée lors de la compilation des déclarations d'événements élémentaires et complétée lors du traitement des opérations d'assignation.

Structure



5) Table annexe : TABSEQ

Elle permet, lors de la génération de code, de retrouver les séquences, tâches ... à compléter.



ANNEXE IV

Exemple traité

```
    DEC EVT
    événement E1   : élémentaire, NOPART
        "     E2   :      "         "
        "     E3   :      "         "
        "     E4   :      "         "
        "     B    :      "         "
    évènement EXP1 : composé (E3 avant-a 1ère B), NØPART
        "     EXP2 :      "  (E1 OU E2), PART
        "     EXP3 :      "  (récent E1 ET récent E2),NØPART
                              avec cons    avec cons
    événement TIM1 : composé (12H53mn06 sec)
        "     TIM2 :      "  (chaque 20 mn après 18H7mn6sec)
        "     TIM3 :      "  (chaque 100 sec avant 18H)
    FIN EVT
    DEC NIV 3
sur chaque EXP1 avec cons si COND1 faire T1
                            sinon faire T2
sur récent EXP2 avec cons si COND2 faire T3,(TØ,EVTØ,3sec)
sur chaque EXP2 sans cons si COND3 faire T4
sur EXP3 faire T5
sur TIM1 faire T6
sur TIM2 faire T7
sur TIM3 faire T8
sur EVTØ faire T9 ... pour le traitement du TØ.
```

On a supposé, sans les fixer, que les priorités étaient croissantes de T1 à T9.

```
*IFOR TS= 21,XNT= 31 ——> priorité et n° de tâche.
      DIMENSION ICR(2)
      INT= 80 ——> n° tache de niveau 3 associé
      IE1= 01 ——> E3 (n° interne 1)
      IE2= 02 ——> E4 (n° interne 2)
      JJJ= 01 ——> 1ère
      ICPT=0
   10 ICR=-1
      CALL WEVOR(IERR,ICR,IE1,IE2)
      CALL REVENT(IE1,IERR)
      CALL REVENT(IE2,IERR)
      IF(ICR(1).EQ.-1)GOTO 30
      ICPT=ICPT+1
      IF(ICPT.EQ.JJJ)GOTO 50
      GOTO 10
   30 CALL RUN(INT,IERR) ——> activation tache niveau 3.
      GOTO 10
   50 CALL EXIT
      END
```

Tâche d'évaluation correspondant à EXP1 : E3 avant-q 1re E4

```
*IFOR TS= 22,XNT= 32
      DIMENSION ICR(2)
      INT= 50
      IE1= 03
      IE2= 04
   10 CALL WEVOR(IERR,ICR,IE1,IE2)
      CALL REVENT(IE1)
      CALL REVENT(IE2)
      CALL RUN(INT,IERR)
      GOTO 10
      END
```

EXP2 : E1 ou E2

```
*IFOR TS= 23,XNT= 33
      DIMENSION ICR(2)
      LOGICAL E1,E2
      INT= 41
      IE1= 03
      IE2= 05
   10 ICR=-1
      CALL WEVOR(IERR,ICR,IE1,IE2)
      CALL REVENT(IE1)
      CALL REVENT(IE2)
      IF (ICR(1).EQ.-1) E2=.TRUE.
      IF (ICR(2).EQ.-1) E1=.TRUE.
      IF ((.NOT.E1).OR.(.NOT.E2))GOTO 10
      CALL RUN(INT,IERR)
      GOTO 10
      END
```

EXP3 : 2ct E1 et 2ct E4 Ac Ac

```
*IFOR TS= 24,XNT= 34
      INT= 70
      IE1= 08 ——> n° d'evenement declenché par la tâche Time_out
      CALL WEVENT(IE1,IERR)
      CALL REVENT(IE1)
      CALL RUN(INT,IERR)
      CALL EXIT
      END
```

Evt simple : EVT4


```
*IFOR TS= 84,XNT=107
      DIMENSION IDATE(7),ITIM(3)
      ITIM(1)= 07
      ITIM(2)= 15
      ITIM(3)= 00
      INT=107
      CALL TIME(IDATE,IERR)
      IF (IDATE(4).GT.ITIM(1)) GOTO 50
      IF (IDATE(4).LT.ITIM(1)) GOTO 60
      IF (IDATE(5).GT.ITIM(2)) GOTO 50
      IF (IDATE(5).LT.ITIM(2)) GOTO 60
      IF (IDATE(6).GT.ITIM(3)) GOTO 50
   60 CALL OFF(INT,IERR)
      CALL EXIT
   50 CONTINUE
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C INCLUSION A CE NIVEAU DE L'APPEL DU RESTE DE LA TACHE UTI
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL EXIT
      END
```

Tâche T8 : Tâche utilisateur correspondant à l'appel chaque 20 mn après 15 H. Activée dans INIT par call TRAIN...

```
*IFOR TS= 25,XNT= 80 ——> n° de tâche
      DIMENSION ITAB(6),KTAB(2)
      LOGICAL COND
      IPTTSK= 01
      IMUT= 02
      IDEB=0
      IDEP=IMUT*2+IDEB
      IADN3=100
      IADCOND=300
      IN=7
      IMUC=1
      IC=1
      IPTT=IPTTSK
      CALL RRQST(IMUT,IERR);
      CALL REDGET(2,IERR,IDEP,KTAB)
      KTAB(1)=KTAB(1)+1;
      CALL REDPUT(2,IERR,IDEP,KTAB)
      CALL RRLSE(IMUT,IERR)
   10 IF (IPTT.EQ.999) GOTO 1000
      IP=IN*IPTT + IADN3
      CALL REDGET(IN,IERR,IP,ITAB)
      IQ=IC*ITAB(2)+IADCOND
      CALL RRQST(IMUC,IERR)
      CALL REDGET(IC,IERR,IQ,COND)
      CALL RRLSE(IMUC,IERR)
      IF (COND) GOTO 200 ——> vérification de la condition
      K=ITAB(2)
   20 IF (K.EQ.0) GOTO 100
      IP=IP+1
      CALL REDGET(IN,IERR,IP,ITAB)
      IF (ITAB(4).EQ.1) GOTO 200
      IQ=IC*ITAB(2)+IADCOND
      CALL RRQST(IMUC,IERR)
      CALL REDGET(IC,IERR,IQ,COND)
      CALL RRLSE(IMUC,IERR)
      IF (COND) GO TO 200
      K=K-1
```

Tâche de niveau 3 correspondant à EXP1
- ITAB correspond à TAB TK3
- ITAB(9) = n° tâche time-out si elle existe

mise-à-jour du nombre d'occurrences

```
      GO TO 20
  100 IPTT=ITAB(6)
      GOTO 10
  200 IF (ITAB(5).LE.1) GOTO 300
      IF (ITAB(5).EQ.3) GOTO 210
      CALL OFF(ITAB(7),IERR)
  210 CALL RUN(ITAB(7),IERR)
  300 CALL RUN(ITAB(1),IERR)
 1000 CALL EXIT
      END
C
C
C
*IFOR TS=26,XNT=60
      DIMENSION ITAB(6),KTAB(2)
      LOGICAL COND
      IPTTSK= 03
      IMUT= 02
      IDEB=0
      IDEP=IMUT*2+IDEB
      IADNO=100
      IADCOND=300
      IN=7
      IMUC=1
      IC=1
      IPTT=IPTTSK
      CALL RRQST(IMUT,IERR);
      CALL REDGET(2,IERR,IDEP,KTAB)
      KTAB(1)=KTAB(1)+1;
      CALL REDPUT(2,IERR,IDEP,KTAB)
      CALL RRLSE(IMUT,IERR)
   10 IF (IPTT.EQ.999) GOTO 1000
      IP=IN*IPTT + IADN3
      CALL REDGET(IN,IERR,IP,ITAB)
      IQ=IC*ITAB(2)+IADCOND
      CALL RRQST(IMUC,IERR)
      CALL REDGET(IC,IERR,IQ,COND)
      CALL RRLSE(IMUC,IERR)
      IF (COND) GOTO 200
      K=ITAB(2)
   20 IF (K.EQ.0) GOTO 100
      IP=IP+1
      CALL REDGET(IN,IERR,IP,ITAB)
      IF (ITAB(4).EQ.1) GOTO 200
      IQ=IC*ITAB(2)+IADCOND
      CALL RRQST(IMUC,IERR)
      CALL REDGET(IC,IERR,IQ,COND)
      CALL RRLSE(IMUC,IERR)
      IF (COND) GO TO 200
      K=K-1
      GO TO 20
  100 IPTT=ITAB(6)
      GOTO 10
  200 IF (ITAB(5).LE.1) GOTO 300
      IF (ITAB(5).EQ.3) GOTO 210
      CALL OFF(ITAB(7),IERR)
```

Vérification existence Time.out et activation de la tâche

Tâche de Niveau 3 correspondant à Exp2.


```
  210 CALL RUN(ITAB(7),IERR)
  300 CALL RUN(ITAB(1),IERR)
 1000 CALL EXIT
      END
C
C
C
*IFOR TS= 27,XNT= 41
      DIMENSION ITAB(6),KTAB(2)
      LOGICAL COND
      IPTTSK= 05
      IMUT= 04
      IDEB=0
      IDEP=IMUT*2+IDEB
      IADNO=100
      IADCOND=300
      IN=7
      IMUC=1
      IC=1
      IPTT=IPTTSK
      CALL RRQST(IMUT,IERR);
      CALL REDGET(2,IERR,IDEP,KTAB)
      KTAB(1)=KTAB(1)+1;
      CALL REDPUT(2,IERR,IDEP,KTAB)
      CALL RRLSE(IMUT,IERR)
   10 IF (IPTT.EQ.999) GOTO 1000
      IP=IN*IPTT + IADN3
      CALL REDGET(IN,IERR,IP,ITAB)
      IQ=IC*ITAB(2)+IADCOND
      CALL RRQST(IMUC,IERR)
      CALL REDGET(IC,IERR,IQ,COND)
      CALL RRLSE(IMUC,IERR)
      IF (COND) GOTO 200
      K=ITAB(2)
   20 IF (K.EQ.0) GOTO 100
      IP=IP+1
      CALL REDGET(IN,IERR,IP,ITAB)
      IF (ITAB(4).EQ.1) GOTO 200
      IQ=IC*ITAB(2)+IADCOND
      CALL RRQST(IMUC,IERR)
      CALL REDGET(IC,IERR,IQ,COND)
      CALL RRLSE(IMUC,IERR)
      IF (COND) GO TO 200
      K=K-1
      GO TO 20
  100 IPTT=ITAB(6)
      GOTO 10
  200 IF (ITAB(5).LE.1) GOTO 300
      IF (ITAB(5).EQ.3) GOTO 210
      CALL OFF(ITAB(7),IERR)
  210 CALL RUN(ITAB(7),IERR)
  300 CALL RUN(ITAB(1),IERR)
 1000 CALL EXIT
      END
C
C
C
```

Tâche de Niveau 3 correspondant à exp3

```
*IFOR TS= 28,XNT= 70
      DIMENSION ITAB(6),KTAB(2)
      LOGICAL COND
      IPTTSK= 09
      IMUT= 05
      IDEB=0
      IDEP=IMUT*2+IDEB
      IADN3=100
      IADCOND=300
      IN=7
      IMUC=1
      IC=1
      IPTT=IPTTSK
      CALL RRQST(IMUT,IERR);
      CALL REDGET(2,IERR,IDEP,KTAB)
      KTAB(1)=KTAB(1)+1;
      CALL REDPUT(2,IERR,IDEP,KTAB)
      CALL RRLSE(IMUT,IERR)
   10 IF (IPTT.EQ.999) GOTO 1000
      IP=IN*IPTT + IADN3
      CALL REDGET(IN,IERR,IP,ITAB)
      IQ=IC*ITAB(2)+IADCOND
      CALL RRQST(IMUC,IERR)
      CALL REDGET(IC,IERR,IQ,COND)
      CALL RRLSE(IMUC,IERR)
      IF (COND) GOTO 200
      K=ITAB(2)
   20 IF (K.EQ.0) GOTO 100
      IP=IP+1
      CALL REDGET(IN,IERR,IP,ITAB)
      IF (ITAB(4).EQ.1) GOTO 200
      IQ=IC*ITAB(2)+IADCOND
      CALL RRQST(IMUC,IERR)
      CALL REDGET(IC,IERR,IQ,COND)
      CALL RRLSE(IMUC,IERR)
      IF (COND) GO TO 200
      K=K-1
      GO TO 20
  100 IPTT=ITAB(6)
      GOTO 10
  200 IF (ITAB(5).LE.1) GOTO 300
      IF (ITAB(5).EQ.3) GOTO 210
      CALL OFF(ITAB(7),IERR)
  210 CALL RUN(ITAB(7),IERR)
  300 CALL RUN(ITAB(1),IERR)
 1000 CALL EXIT
      END
C
C
C
```

Tâche de Niveau 3 correspondant à l'événement simple EVT0

```
*IFOR TS= 29,XNT=108
      DIMENSION ICR(2)
      INTO=108
      INT=102
      IDEL= 11
      IFIN= 07
      IEVTO= 08
      IUD= 01
```

TACHE TIME_OUT

n° tâche utilisateur utilisant ce time-out
n° evt délai
n° evt fin de tâche
n° evt de liaison avec la tâche d'évaluation


```
      IUD= 01
      ICR=-1
      CALL SEVDEL(IDEL,IERR,ID,IUD)
      CALL WEVOR(IERR,ICR,IDEL,IFIN)
      CALL REVENT(IDEL)
      CALL REVENT(IFIN)
      IF (ICR(1).EQ.-1)GOTO 100
      CALL INHIB(INT,IERR)
      CALL SEVENT(IEVTO,IERR)
      CALL INHIB(INTO,IERR)
  100 CALL EXIT
      END
```

→ suspension tâche utilisateur.
→ la tâche va se détruire

```
*IFOR TS= 29,XNT=100,IAPPMAX= 10
      DIMENSION ITAB(2)
      IEVT= 10
      IMUT= 02
      IDEB=0
      IDEP=IMUT*2+IDEB
      CALL RRQST(IMUT,IERR)
      CALL REDGET(2,IERR,IDEP,ITAB)
      IF (ITAB(1).EQ.ITAB(2)) GOTO 20
      ITAB(2)=ITAB(2)+1
      CALL REDPUT(2,IERR,IDEP,ITAB)
      GOTO 30
   20 CALL RRLSE(IMUT,IERR)
      CALL EXIT
   30 CALL RRLSE(IMUT,IERR)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C ICI APPEL TACHE UTILISATEUR : T1                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL SEVENT(IEVT,IERR)
      CALL EXIT
      END
```

TÂCHE UTILISATEUR T1

→ chaque

Exclusion mutuelle

→ consommation

```
*IFOR TS= 29,XNT=101,IAPPMAX= 10
      DIMENSION ITAB(2)
      IEVT= 11
      IMUT= 02
      IDEB=0
      IDEP=IMUT*2+IDEB
      CALL RRQST(IMUT,IERR)
      CALL REDGET(2,IERR,IDEP,ITAB)
      IF (ITAB(1).EQ.ITAB(2)) GOTO 20
      ITAB(2)=ITAB(2)+1
      CALL REDPUT(2,IERR,IDEP,ITAB)
      GOTO 30
   20 CALL RRLSE(IMUT,IERR)
      CALL EXIT
   30 CALL RRLSE(IMUT,IERR)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C ICI APPEL TACHE UTILISATEUR : T2                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL SEVENT(IEVT,IERR)
      CALL EXIT
      END
```

T2

```
*IFOR TS= 29,XNT=102,IAPPMAX= 01 --> recent
      DIMENSION ITAB(2)
      IEVT= 07
      IMUT= 03
      IDEB=0
      IDEP=IMUT*2+IDEB                                    T3
      CALL RRQST(IMUT,IERR)
      CALL REDGET(2,IERR,IDEP,ITAB)
      IF (ITAB(1).EQ.ITAB(2)) GOTO 20
      ITAB(2)=ITAB(2)+1
      CALL REDPUT(2,IERR,IDEP,ITAB)
      GOTO 30
   20 CALL RRLSE(IMUT,IERR)
      CALL EXIT
   30 CALL RRLSE(IMUT,IERR)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C ICI APPEL TACHE UTILISATEUR : T3
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL SEVENT(IEVT,IERR)
      CALL EXIT
      END
*IFOR TS= 29,XNT=103,IAPPMAX= 00
      DIMENSION ITAB(2)
      IEVT= 12
      IMUT= 03
      IDEB=0                                             T4
      IDEP=IMUT*2+IDEB
      CALL RRQST(IMUT,IERR)
      CALL REDGET(2,IERR,IDEP,ITAB)
      IF (ITAB(1).GT.ITAB(2)) GOTO 20
      CALL RRLSE(IMUT,IERR)
      CALL EXIT
   20 CALL RRLSE(IMUT,IERR)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C ICI APPEL DE LA TACHE UTILISATEUR : T4              C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL SEVENT(IEVT,IERR)
      CALL EXIT
      END
```



```
*IFOR TS= 29,XNT=104,IAPPMAX= 00 --> fugitif          T5
      IEVT= 13
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C ICI APPEL DE LA TACHE UTILISATEUR : T5                C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL EXIT
      END
*IFOR TS= 29,XNT=105,IAPPMAX= 00                      T6
      IEVT= 14
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C ICI APPEL DE LA TACHE UTILISATEUR : T6                C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL EXIT
      END
*IFOR TS= 29,XNT=106,IAPPMAX= 00                      T7
      IEVT= 15
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C ICI APPEL DE LA TACHE UTILISATEUR : T7                C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL EXIT
      END
*IFOR TS= 29,XNT=108,IAPPMAX= 00                      T9
      IEVT= 17
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C ICI APPEL DE LA TACHE UTILISATEUR : T9                C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL EXIT
      END
*IFOR TS= 10,XNT= 30                          [tâche INIT]
      DIMENSION IVE(2)
      DIMENSION ITIM0(3)
      DIMENSION ITIM1(3)
      IVE=0
C*** TRANSFERT DE VE ET I DANS LA ZDR **
      DO 10 I=1,50
      CALL REDPUT(2,IERR,0,IVE)
   10 CONTINUE
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C LECTURE ET TRANSFERT DE TABTK3(FICHIER SOUT3) DANS LA ZDR C
C           ICI DOIVENT SE TROUVER LES INSTRUCTIONS          C
C           FORTRAN DE LECTURE ET DE TRANSFERT               C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL RUN( 31,IERR) }
      CALL RUN( 32,IERR) }  activation tâches cycliques
      CALL RUN( 33,IERR) }  d'évaluation (n° 31, 32, 33 et 34)
      CALL RUN( 34,IERR) }
      ITIM0(1)= 12 }
      ITIM0(2)= 53 }  12h 53' 06"
      ITIM0(3)= 06 }
      CALL TRNON(105,ITIM0,IERR) --> activation de T6
      ITIM1(1)= 18 }
      ITIM1(2)= 07 }  18h 07' 00"
      ITIM1(3)= 00 }
      CALL TRNON(106,ITIM1,IERR,0, 20,1)  --> activation de T7
      CALL START(107,0,0,IERR,0,100,0)    --> activation de T8
      END                                     (périodique)
```

ANNEXE V

- <spécification d'application>::=
  SYGAP<partie déclaration d'événements><partie utilisations>[<partie assignations>]PAGYS

- <partie déclaration d'événements>::=

DECEVT <suite de déclaration d'événements>FINEVT

- <partie utilisations>::=

DECNIV3 <suite d'utilisations>FINNIV3

- <partie assignations>::= DECASS<suite d'assignations>FINASS

- <suite de déclarations d'événements>::=
  <déclaration d'événement>;<suite de déclaration d'événements>
  |<déclaration d'événement>

- <suite d'utilisations>::=
  <utilisation>;<suite d'utilisation>
  |<utilisation>

- <suite d'assignations>::=<assignation>;
  <suite d'assignation>|<assignation>

- <déclaration d'événement>::=
  <déclaration d'événement simple>|<déclaration d'événement composé>

- <déclaration d'événement simple>::=
  EVENEMENT<ident-événement>:ELEMENTAIRE,<partage>
  [,(DUP,<ident-événement>)]

<partage>::= PART | NOPART

- <déclaration d'événement composé>::=
  EVENEMENT <ident-événement>: COMPOSE (<expression d'événements>),<partage>

- <expression d'événement>::=
  <facteur><opérateur diadique><facteur>|
  <opérateur monadique><facteur>

- <facteur>::=[<qualificatif d'occurrence>]<ident-événement>

- <opérateur diadique>::= ET|OU|AVANT|APRES|SIM

- <opérateur monadique>::=<entier>OCCURRENCE DE
  |<délai>après
  <entier> est un nombre entier
  <délai> est un nombre d'unités de temps

- <qualificatif d'occurrence>::= CHAQUE|RECENT

- <utilisation>::=<structure d'utilisation>|
  <structure de suspension>|<structure de reprise>

- <structure d'utilisation>::=
  SUR <événement qualifié><attribut de consommation>
  <partie si>[<partie sinon>]

- <événement qualifié>::=[<qualificatif d'occurrence>]<ident-événement>

- <attribut de consommation>::= AVEC CONS|SANS CONS

- <partie si>::= SI<condition>[AVEC<traitement-cond>]
  <partie traitement$_1$>

- <partie sinon>::= SINON<partie traitement$_2$>

- <partie traitement$_1$>::= FAIRE<ident-tâche>
  AVEC PRIORITE = <entier>
  [,(TØ,<ident-d'événement>,<délai>)]

- <partie traitement$_2$>::= FAIRE<ident-tâche>

- <condition>::=<condition simple>|<condition composée>

- <condition simple>::=<état du procédé>|<état tâches>
  |<état donnée transmissible>

- <condition composée>::=<condition simple>
  <opérateur booléen><condition simple>

- <opérateur booléen>::= ET|OU

- <état du procédé>::=<ident d'entrée><opérateur de relations>
  <valeur>

- <opérateur de relation>::=>|<|≥|≤|=|≠
  <ident-entrée> représente un identificateur symbolique d'une variable du procédé. La valeur est ure valeur conforme au type d'entrée considéré.

- <état tâche>::=<ident-tâche>ACTIVE|<ident-tâche>INACTIVE

- <état donné transmissible>::=<ident donnée>
  <opérateur-de-relation><valeur>

- <traitement-cond>::=<ident-donnée>:=<valeur>

- <structure de suspension>::=
  SUR<qualificatif occurrence><ident-d'événement>
  { SUSPENDRE / TUER } <id-tâche>

- <structure de reprise>::=
  SUR <qualificatif occurrence><ident d'événement>
  REPRENDRE <id-tâche>

- <assignation>::= ASSIGNER<ident evt simple>A<source>
  SUR <adresse-site >
  AVEC FREQUENCE =<fréquence d'échantillonnage>
  DETECTE PAR <ident module de détection>

- SOURCE ::= transition vf EB <adresse-entrée-booléenne>
  | transition fv EB < " " >
  | IT <adresse d'interruption >
  | messop = <message-opérateur>
  | DT <ident-donnée-transmissible>vide
  | " < " >pleine
  | finde <id-tâche>
  | finde <id-module>
  | transition fv de EN<adresse entrée numérique>
  <op relation><constante>
  | transition vf de " "
  | transition fv de EA<adresse entrée analogique>
  <op relation><constante>
  | transition vf de EA " "
- <op relation>::= >|<|=|≠|≥|≤

BIBLIOGRAPHIE

AFCET 80 — Sureté de fonctionnement des systèmes informatiques
Monographie de l'AFCET 1980.

BARN 75 — J.G.P. BARNES
The design and use of RTL/2
Journées AFCET "Temps Réel" Paris 1975.

BEN & al 83 — M. BENMAIZA - J.P. THOMESSE
A specification of Real Time Applications by events.
Real Time Digital Control Applications
vol. 1 IFAC Symp. GUADALAJARA, janv. 1983.

BERN & al 81 — A. BERNSTEIN, P.K. HARTER JR
Proving Real Time Properties of Programs with Temporal Logic
Operating System Review Vol. 15 n° 5 Déc. 1981.

BIAN & al 76 — G. BIANCHI - A. LEWIS
A pragmatic Approach to a high level Real-Time language.
Proceedings of the 1976 IFAC-IFIP
Workshop on "Real time Programming" Rocquencourt
2-4 juin 1976.

BOS & al 81 — J. Van Den BOS - R. PLASMEIJER - J. STROET
Process communication Based on Input Specifications
ACM TOPLS Vol. 3 n° 3 - Juillet 1981.

BRIN 78 — Distributed Processes : A concurrent Programming concept
CACM 21,11 1978.

BRAN & al 82 P. BRANQUART - G. LOUIS - P. WODON
Aspects de CHILL, le langage du CCITT
TSI vol. 1 n° 1 1982

CALV 82 J.P. CALVEZ
Une méthodologie de conception des systèmes multimicroordinateurs pour les applications de commande en temps réel.
Thèse de Docteur-es-Sciences
Université de Nantes nov. 1982

CALV & al 82 J.P. CALVEZ - T. GUILMIN
Un système d'aide à la conception pour les applications de commande en temps réel
BIGRE n° 31 oct. 1982

COH 80 D. COHEN
Flow control for real time communications
ACM Computer communication Review
janv/avril 1980 - vol. 10 n° 1 et 2.

CAMP 74 R.H. CAMPBELL – HABERMANN
The Specification of Process Synchronisation by Path Expressions
Colloque sur les aspects théoriques et pratiques des systèmes d'exploitation
IRIA Paris 1974

CROC 75 CROCUS
Systèmes d'exploitation des ordinateurs
Ed DUNOD 1975.

DEN 75 J.B. DENNIS
First version of a data flow procedure language
Mac Technical Memorandum 61
M.I.T. mai 1975.

DER & al 79 J.C. DERNIAME - J.P. THOMESSE
Flux de données et synchronisation
IRIA/IGØD Workshop
Aix-en-Provence - mai 1979

DER & al 79 * J.C. DERNIAME - J.P. FINANCE
Types abstraits de données : "Spécification, Utilisation et Réalisation"
Ecole d'été de l'AFCET Monastir 1979.

DER & al 83 (1) J.C. DERNIAME - A. ZAKARI
Spécification d'application de conduite d'un atelier flexible
Convention Informatique Latine
Barcelone 1983

DER & al 83 (2) J.C. DERNIAME - A. ZAKARI
Langage de conception pour des applications réparties de conduite de procédés.
Journée BIGRE 83 Le Cap d'Agde
17-19 oct. 1983.

DPT 83 J.C. DERNIAME - G.R. PERRIN - J.P. THOMESSE
Communication based design of Distributed System.
Convention Informatique latine
Barcelone 1983

DIJK 65 E.W. DIJKSTRA
Solution of a Problem in concurrent Programming
CACM 8,9 1965.

DIJK 75 E.W. DIJKSTRA
Guarded command, Nondeterminacy and Formal Derivation of Programs
CACM 18,8 1975.

DUN & al 80 A.G. DUNCAN - J.S. HUTCHINSON
Using ADA for Industrial Embedded Microprocessor.
Sig plan Symposium on the ADA Prog. Language.
Boston Massachussets - Déc. 9-11, 1980.

ELZ 75 Pr. ELZER
PEARL
Journées AFCET "Temps Réel"
Paris 1975.

FLYS 75 D. FLYSTRA
HAL/S Programming Language
Journées AFCET "Temps Réel"
Paris 1975.

FELD 79 J.A. FELDMAN
High Level Programming for Distributed Computing
CACM 22,6 - juin 1979.

GER & al 75 J. GERTLER - J. SEDLAK
Survey Paper
Software for Process Control
Automatica vol. 11, pp. 613-625
Pergamon Press 1975.

GØF 84 Thèse Docteur-Ingénieur INPL (Nancy)
A paraître.

GØG & al 78 J.A. GOGUEN - J.W. THATCHER - O.G. WAGNER
An Initial Algebra Approach to the specification
Correctness and Implementation of Abstract Data Type
in Current trends of Programming Languages 1978

GRIE 76 P.D. GRIEM
Approaching an easy-to-learn Methode of Programming
Real-Time Parallel Processes
Proc. of the 1976 IFAC/IFIP Workshop on "Real-time Programming"
Rocquencourt 2-4 juin 1976.

GUT 78 J.V. GUTTAG
"Notes on types abstraction"
Program Construction LNCS n° 69 - 1978.

GUT & al 78 J.V. GUTTAG - E. HOROWITZ - D.R. MUSSER
"The Design of Data Type Specification"
in current Trends of Programming Languages - 1978.

HADE 79 V.H. HAASE - W.M. DEHNERT
High Level Language structure for Distributed Real-Time Programming
2nd SOCOCO - 1979 - Prague

HAHA 77 V.H. HAASE - H. HALLING
Description of R.T. Applications using the Guarded command Concept
TR 77

HALB & al 82 N. HALBWACHS - P. CASPI
Algebra of Events : A Model for Parallel and Real Time Systems
RR. n° 285 - IMAG. Grenoble.

HOA 74 C.A.R. HOARE
Monitors : An Operating System Structuring Concept
CACM 17,10 - 1974.

HOA 78 C.A.R. HOARE
Communicating Sequential Processes
CACM 21,8 1978.

ICH & al 79 J.D. ICHBIACH - J.C. HELIARD - O. ROUBINE
J.G.P. BARNES - B. KRIEG BRÜCHNER - B.A. WICHMAN
Rationale for the Design of the ADA Programming Language
Sig plan notice vol. 14 n° 6 - juillet 1979

KAIS & al 78 C. KAISER - M. KRONENTAL - J. LANCET - S. NATKIN
S. PALASSIN
Système informatique réparti pour la conduite d'un atelier mécanique.
Congrès AFCET Théorie et Technique Informatique
Paris 1978.

KRAM & al 72 KRAMER - MACEE - SLOMAN
A software architecture for distributed computer control system.
IFAC Symp on Theorie and Applications of Digital Control - 1982 India.

KRON 79 M. KRONENTAL
Towards the standardization of real time operating system Kernels.
SOCOCO 1979 Prague.

KRU & al 81 A.&.P. KRUCHTEN
Manuel de référence du langage de programmation ADA
Editions Eyrolles 1981.

LAD 77 P. LADET
Outils de structuration Temps Réel dans la commande des procédés industriels
Thèse 3ème cycle - INPG 1977.

LAD 82 P. LADET
Contribution à l'étude des systèmes informatiques répartis pour la commande des Procédés Industriels.
Thèse de Docteur-es-Sciences 1982 - INPG.

LAMP 78 L. LAMPORT
Time, Clocks and the Ordering of Events in a Distributed System
CACM 21,7 1978.

LECAL 79 F. LECALVEZ-LISH
Définition d'un langage de description globale des applications temps réel.
Thèse de 3ème cycle. Université Pierre et Marie Curie (Pairs VI) 1979.

LIS & al 74 B. LISKOV - S. ZILLES
Programming with Abstract Data Type
Sig plan Notices vol. 9-4 1974.

LIS & al 81 B. LISKOV - R. ATKINSON - T. BLOOM - E. MOSS - J.C. SCHAFFERT - R. SCHEIFLER - A. SNYDER
CLU Reference manual
LNCS n° 114.

MAD 77 F. MADAULE
L'évolution des langages temps réel
I.P.P.
in TR77.

MAH 81 A. MAHJOUB
Some concerts on ADA as a Real Time Programming Language
ACM SIGPLAN Notices
Vol. 16 n° 2 février 1981.

MAO & al 80 T.W. MAO - R.T. YEH
Communication port :
A language concept for concurrent Programming
IEEE TOSE - vol. SE-6, N° 2 , mars 1980.

MEND 76 H.G. MENDELBAUM
"Structuration dans la conception et réalisation des systèmes Temps Réel"
Thèse de docteur-ès-Sciences IPP 1976.

MID 77 T. MIDDLETON
Specifying Program Structure Through Sequence Relation Ships
Sigplan Notices oct. 1977.

MIN 79 R. MINOT
"ATM : un système de fabrication de programmes basés sur les concepts de modularité et de types abstraits"
Thèse 3ème cycle Nancy I 1979.

NANT 70 J.P. NANTET
Ordinateurs en Temps Réel. Applications Industrielles.
Ed Masson & Cie 1970.

NONN 78 P. NONN
Le système d'exploitation dans SYGARE
Thèse de Docteur Ingénieur INPL Nancy 1978.

PAR & al 75 P. PARAYRE - M. TROCELLO
LTR, un système de réalisation pour l'informatique Temps Réel
Journées AFCET "Temps Réel" Paris 1975

PAR 80 P. PARAYRE
LTR. V3
Projet de spécification technique
Publication CELAR (BRUZ). 1980.

PIKE 72 — H.E. PIKE
Procedural language development in the Purdue workshop on the standardization of industrial computer languages
Congrès IFAC - Paris 1972.

PUL 78 — PULOU J.
A Tool for the specification of synchronisation in High Level Languages
RAIRO Computer Sciences n° 4 1978.

RIT & al 75 — M. RITOUT - P. HUGOT
PROCOL. Un système de programmation Temps Réel
Journées AFCET "Temps Réel" Paris 1975.

ROU 78 — G.P. ROUCAIROL
Mots de synchronisation
RAIRO Informatique vol. 12 n° 4 1978.

RVER 77 — J.P. VERJUS - P. ROBERT
Expression autonome de la synchronisation de processus concurrents.
dans TR 77.

SAVØ 82 — S. SAVOYSKY
Proposition de méthodologie pour la description de systèmes automatiques
Thèse de docteur-ès-Sciences - Paris 6 - 1982.

SEMS 1 — Fortran Temps Réel
Manuel de référence 1.164.002 00/30 04
Manuel d'utilisation 1.164.002 00/36 04

SEMS II — RTES-D
Manuel de référence 1.164.309 4/--36

SCEP 82 — Projet SCEPTRE : Proposition de Standard de Noyau d'exécutif Temps Réel -
Rapport BNI 1982.

SCHU 79 — On the Design of a Language for Programming Real-Time Concurrent Processes
H.A. SCHUTZ
IEEE TOSE Vol. SE5, n° 3 , mai 1979.

SOU 79 — C. SOURISSE
L'évolution des automatismes séquentiels et son influence sur les choix technologiques
Le Nouvel Automatisme - nov. 1979.

TR 77 — Congrès AFCET sur
la programmation globale des synchronisations dans les applications Temps Réel
Paris 3-4 nov. 1977.

THOM 80 — J.P. THOMESSE
SYGARE : Une structuration pour la conception d'application en temps réel et réparties.
Thèse de Docteur-ès-Sciences INPL 1980.

VIT 81 M. VITINS
Requirements specifications for Industrial Real Time Automation system
Rapport n° KLR -81-59C Avril 1981.

WEBB 75 J.T. WEBB
CORAL 66
Journées AFCET "Temps Réel" Paris 1975.

WIRT 77 N. WIRTH
Toward a discipline of Real Time Programming
CACM 20,8 1977.

ZAK 84 A. ZAKARI
Thèse de 3ème cycle - à paraître - Université de Nancy I

ZAVE 82 P. ZAVE
A operational approach to requirements specification for embedded systems
IEEE TOSE Vol. SE-8 , n° 3, May 1982.

NOM DE L'ETUDIANT : BENMAIZA Mohamed

NATURE DE LA THESE : Doctorat Ingénieur en Informatique

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le 24 [illegible] 454

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

R. MAINARD

Résumé :

Le concept d'événement est étudié sous les aspects spécification et programmation d'applications en conduite de procédés industriels qui sont vues en termes d'automates.

Un outil de spécification d'applications de ce type consistant en un langage de manipulation d'événement est défini.

Une des caractéristiques de ce langage est de permettre une séparation totale entre les problèmes de détection, de définition et d'utilisation des événements autorisant par là une spécification séparée de ces trois aspects et donc une remise en cause aisée de l'un ou l'autre de ces aspects.

Mots clés : temps réel, événement, spécification, programmation, application en conduite de procédés, évolutivité, sûreté de fonctionnement, répartition, communication, synchronisation.