Méthode de conception descendante de systèmes temps réel

Daniel Pilaud

THESE

présentée à

l'Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR DE 3ème CYCLE
«Génie Informatique»

par

Daniel PILAUD

METHODE DE CONCEPTION DESCENDANTE DE SYSTEMES TEMPS REEL.

Thèse soutenue le 23 novembre 1982 devant la commission d'examen.

	G. VEILLON	Président
Messieurs	J.P. BANATRE	
	P. CASPI	
	P. CLAUDIN	
	R. DAVID	Examinateurs
	J. GOLDBERG	
	J.P. PERRIN	
Madame	G. SAUCIER	

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1979-1980

Président : M. Philippe TRAYNARD

Vice-Présidents : M. Georges LESPINARD
M. René PAUTHENET

PROFESSEURS DES UNIVERSITES

MM.	ANCEAU François	Informatique fondamentale et appliquée
	BENOIT Jean	Radioélectricité
	BESSON Jean	Chimie Minérale
	BLIMAN Samuel	Electronique
	BLOCH Daniel	Physique du Solide - Cristallographie
	BOIS Philippe	Mécanique
	BONNETAIN Lucien	Génie Chimique
	BONNIER Etienne	Métallurgie
	BOUVARD Maurice	Génie Mécanique
	BRISSONNEAU Pierre	Physique des Matériaux
	BUYLE-BODIN Maurice	Electronique
	CHARTIER Germain	Electronique
	CHERADAME Hervé	Chimie Physique Macromoléculaires
Mme	CHERUY Arlette	Automatique
MM.	CHIAVERINA Jean	Biologie, Biochimie, Agronomie
	COHEN Joseph	Electronique
	COUMES André	Electronique
	DURAND Francis	Métallurgie
	DURAND Jean-Louis	Physique Nucléaire et Corpusculaire
	FELICI Noël	Electrotechnique
	FOULARD Claude	Automatique
	GUYOT Pierre	Métallurgie Physique
	IVANES Marcel	Electrotechnique
	JOUBERT Jean-Claude	Physique du Solide - Cristallographie
	LACOUME Jean-Louis	Géographie - Traitement du Signal
	LANCIA Roland	Electronique - Automatique
	LESIEUR Marcel	Mécanique
	LESPINARD Georges	Mécanique
	LONGEQUEUE Jean-Pierre	Physique Nucléaire Corpusculaire
	MOREAU René	Mécanique
	MORET Roger	Physique Nucléaire Corpusculaire
	PARIAUD Jean-Charles	Chimie - Physique
	PAUTHENET René	Physique du Solide - Cristallographie
	PERRET René	Automatique

.../...

MM.	PERRET Robert	Electrotechnique
	PIAU Jean-Michel	Mécanique
	PIERRARD Jean-Marie	Mécanique
	POLOUJADOFF Michel	Electrotechnique
	POUPOT Christian	Electronique - Automatique
	RAMEAU Jean-Jacques	Chimie
	ROBERT André	Chimie Appliquée et des matériaux
	ROBERT François	Analyse numérique
	SABONNADIERE Jean-Claude	Electrotechnique
Mme	SAUCIER Gabrielle	Informatique fondamentale et appliquée
M.	SOHM Jean-Claude	Chimie - Physique
Mme	SCHLENKER Claire	Physique du Solide - Cristallographie
MM.	TRAYNARD Philippe	Chimie - Physique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

CHERCHEURS DU C.N.R.S. (Directeur et Maître de Recherche)

M.	FRUCHART Robert	Directeur de Recherche
MM.	ANSARA Ibrahïm	Maître de Recherche
	BRONOEL Guy	Maître de Recherche
	CARRE René	Maître de Recherche
	DAVID René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	KAMARINOS Georges	Maître de Recherche
	KLEITZ Michel	Maître de Recherche
	LANDAU Ioan-Doré	Maître de Recherche
	MERMET Jean	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Personnalités habilitées à diriger des travaux de recherche (décision du Conseil Scientifique)

E.N.S.E.E.G.

MM.	ALLIBERT Michel	
	BERNARD Claude	
	CAILLET Marcel	
Mme	CHATILLON Catherine	
MM.	COULON Michel	
	HAMMOU Abdelkader	
	JOUD Jean-Charles	
	RAVAINE Denis	
	SAINFORT	C.E.N.G.

.../...

MM. SARRAZIN Pierre
SOUQUET Jean-Louis
TOUZAIN Philippe
URBAIN Georges — Laboratoire des Ultra-Réfractaires ODEILLO

E.N.S.M.E.E.

MM. BISCONDI Michel
BOOS Jean-Yves
GUILHOT Bernard
KOBILANSKI André
LALAUZE René
LANCELOT Françis
LE COZE Jean
LESBATS Pierre
SOUSTELLE Michel
THEVENOT François
THOMAS Gérard
TRAN MINH Canh
DRIVER Julian
RIEU Jean

E.N.S.E.R.G.

MM. BOREL Joseph
CHEHIKIAN Alain
VIKTOROVITCH Pierre

E.N.S.I.E.G.

MM. BORNARD Guy
DESCHIZEAUX Pierre
GLANGEAUD François
JAUSSAUD Pierre
Mme JOURDAIN Geneviève
MM. LEJEUNE Gérard
PERARD Jacques

E.N.S.H.G.

M. DELHAYE Jean-Marc

E.N.S.I.M.A.G.

MM. COURTIN Jacques
LATOMBE Jean-Claude
LUCAS Michel
VERDILLON André

Je tiens à exprimer toute ma reconnaissance à Madame Gabrièle SAUCIER, Professeur à l'ENSIMAG, d'avoir bien voulu m'accueillir dans son équipe de recherche et d'avoir, à tout instant, encadré ce travail.

Je tiens à remercier :

Monsieur G. VEILLON, Professeur à l'ENSIMAG, de m'avoir fait l'honneur de présider le jury de cette thèse,

Monsieur R. DAVID, Maître de recherche au laboratoire d'automatique de Grenoble, d'avoir accepté d'être rapporteur de cette thèse et d'avoir grandement contribué à son amélioration par ses remarques constructives.

Monsieur P. CASPI, Chargé de recherche à l'IMAG, dont la collaboration, les critiques et suggestions m'ont été d'une aide précieuse.

Je voudrais également remercier :

Monsieur J.P. BANATRE, Ingénieur de recherche à l'IRISA,

Monsieur P. CLAUDIN, Chef du service automatique de Merlin Gerin,

Monsieur J. GOLDBERG, Directeur du laboratoire d'informatique du Standford Research Institute,

Mosnieur J.P. PERRIN, Chef du service des études à la RATP,

qui ont accepté de participer au jury de cette thèse.

Que soient aussi remerciés :

- tous mes collègues et amis de l'équipe "Conception et Sécurité des Systèmes", tout particulièrement N. HALBWACHS dont j'ai pu apprécier toutes les qualités au cours de notre collaboration,

- les membres de l'atelier de microinformatique dirigé par Monsieur R. BOUTTAZ ainsi que S. LAFFORGUE, J.F. MILLIONI et H. GUENNOUNI pour leur contribution à ce travail,

- toutes les personnes qui ont assuré la réalisation matérielle de cette thèse : Mesdames G. BOULESTEIX, G. DUFFOURD, S. ROCHE pour la frappe, ainsi que Monsieur D. IGLESIAS et l'équipe de reprographie de l'IMAG pour le tirage.

P L A N

CHAPITRE I

-.-

INTRODUCTION

I - INTRODUCTION

I - 1. GÉNÉRALITÉS SUR LA CONCEPTION DESCENDANTE DES SYSTÈMES TEMPS RÉELS

L'avènement des microprocesseurs tend à accélérer l'informatisation de systèmes de contrôle (avionique, transport, contrôle de processus). Ces domaines d'application ont pour caractéristiques l'exigence d'une haute sécurité, et la présence de contraintes temporelles sur le fonctionnement des systèmes réalisés. En conséquence, la nécessité de disposer de méthodes de conception sûre pour de tels systèmes, apparaît cruciale.

Dans le domaine du logiciel classique, l'approche couramment proposée (voir [2], [3], [4]) consiste, à partir d'une spécification formelle du problème -généralement exprimée en termes de pré et post conditions (voir [33])- à construire progressivement la solution par décompositions successives du problème en sous problèmes de complexité moindre. Les choix de réalisation effectués au cours de ces décompositions sont ainsi facilités, et d'autre part la validité de chaque transformation peut être certifiée formellement.

Dans un contexte plus général, d'autres travaux (voir [34], [35]) proposent une approche analogue de conception par transformations progressives des spécifications, partant d'un niveau de spécification initiale plus élevé, et concernant des réalisations non spécifiquement logicielles.

Cependant, l'application d'une telle approche à la conception de systèmes temps réel n'a été que peu étudiée à ce jour. L'objectif de cet ouvrage est d'ébaucher une telle étude.

I - 2. APPROCHES EXISTANTES

Deux approches se font jour actuellement pour la conception des systèmes temps réel.

. La première est une approche pragmatique, qui connaît un certain succès dans l'industrie. Elle s'est développée, en France, à partir du modèle GRAFCET [9] . Celui-ci permet de décrire le fonctionnement du système à un niveau d'abstraction permettant d'une part une description relativement abstraite et intuitive, et d'autre part une réalisation quasi automatique, notamment à l'aide d'automates programmables. Cette approche nous paraît criticable sous divers aspects. D'une part le niveau d'abstraction choisi semble trop bas pour que la démarche décrite plus haut constitue à elle seule une véritable démarche de conception. En effet, on ne décrit en GRAFCET qu'un fonctionnement du système. Ceci implique que les principaux choix de réalisation ont déjà été faits -sans qu'aucune aide à ces choix ne soit proposée- et que les erreurs les plus coûteuses à corriger ont pû être précédemment introduites -sans qu'aucun moyen pour éviter ou détecter ces erreurs ne soit fourni-. D'autre part, nous verrons que le passage automatique de la description GRAFCET à la réalisation pose des problèmes, en particulier en ce qui concerne les systèmes soumis à des contraintes temporelles "dures", puisque le GRAFCET n'est pas interprétable strictement, du point de vue temporel.

. Une deuxième approche, plus théorique, voit le jour à partir des travaux sur la modélisation des systèmes parallèles. Dans ce domaine, le choix des langages de description et de programmation est souvent subordonné à la préoccupation de fournir des méthodes de preuve. Ainsi plusieurs modèles (voir [29], [30]) ont été proposés qui permettent, à partir d'un nombre réduit d'opérateurs complètement axiomatisés de définir des systèmes et de les composer entre eux, l'axiomatique permettant de déduire le comportement d'un système à partir de celui des sous-systèmes qui le composent, et d'effectuer des preuves d'équivalence entre comportements. Cette approche a été étendue à la modélisation de systèmes synchrones (voir [31]), applicable dans le contexte qui nous intéresse.
Cependant, à notre avis, les modèles de ce type sont encore de trop bas niveau pour être utilisés dans les premières phases de la conception, et un autre désavantage de cette approche est d'être ascendante, au sens où elle procède par assemblage de sous-systèmes et non par décomposition.

Au vu de cette discussion rapide il apparaît que les points fondamentaux qui nous permettent de nous situer par rapport aux deux approches citées ci-dessus tiennent au niveau des spécifications initiales, et au mode de passage à la réalisation.

I - 3. LES DIFFÉRENTS NIVEAUX DE SPÉCIFICATION

Une des thèses fondamentales de ce travail, encore très discutée dans les milieux industriels, est la nécessité de spécifier rigoureusement le système à réaliser avant d'initialiser sa conception. L'intérêt d'une telle spécification initiale est multiple :

- Dans le contexte qui nous intéresse, il est fréquent que le "client" qui commande le système, et le concepteur, qui le réalise, soient des individus ou des entreprises différents.
  La spécification initiale constitue alors une base de dialogue entre les deux parties : il est essentiel que l'accord se réalise très tôt sur la définition du système, afin que sa conception puisse être initialisée sur des bases solides.

- Une spécification rigoureuse permet de garantir le respect des niveaux de décision. On évite ainsi que le concepteur prenne des décisions de réalisation qui modifient la fonction souhaitée.

- Enfin, la spécification initiale sert de référence à toute validation de la réalisation, que cette validation soit effectuée au moyen de preuves formelles ou de tests.

Pour atteindre ces objectifs il faut que l'outil et la méthode de spécification possèdent certaines qualités parfois contradictoires. La principale de ces qualités est l'aptitude à définir le plus précisément possible le comportement attendu d'un système tout en laissant au concepteur la plus grande latitude dans les choix de réalisation. Deux écueils sont en effet à éviter :

- La sous-spécification qui conduit le concepteur à prendre des décisions qui incomberaient normalement au spécificateur;
- la surspécification qui empiète sur les choix de conception et dont l'effet peut être de fermer la porte à des possibilités de réalisation qui sont techniquement ou économiquement plus avantageuses.

Les outils en usage ou en cours d'étude peuvent être classés selon les trois types de description d'un système proposés par Barbacci dans [1], à savoir : les types structurels, algorithmiques ou comportementaux.

I - 3.1. APPROCHE STRUCTURELLE

Il s'agit d'une approche par analogie : on spécifie le comportement désiré d'un système en décrivant dans une certaine technologie la structure d'une machine présentant ce comportement. C'est le cas de l'approche schéma à relais ou ampli-opérationnels utilisée autrefois pour rester proche de la technologie d'implantation, et qui continue d'être utilisée alors que la technologie a évolué (automates programmables, microprocesseurs) afin de tirer le meilleur parti du savoir-faire acquis. On peut aussi classer dans ce type d'approche la spécification à l'aide d'un langage de programmation, dans la mesure où le programme, lié à des choix matériels sous-jacents, est censé représenter une réalisation possible du comportement désiré.

Le fait que les primitives de description soient inspirées d'une technologie particulière limite déjà inévitablement les champs d'application de ces outils à des secteurs cloisonnés.

De plus, dans ce type d'approche, le rôle du spécificateur et celui du concepteur ont tendance à se confondre :

- s'agissant d'une approche qui vise, par essence, à ce que la spécification soit quasiment réalisable telle quelle, le spécificateur est amené à proposer une solution possible (en référence à la technologie en question) plutôt que de se restreindre à spécifier une classe de comportements admissibles;
- à partir d'un certain degré de complexité, l'extraction du comportement décrit par une telle spécification devient difficile. Les possibilités de validation se réduisent alors à une vérification structurelle (correction de syntaxe ou de construction) et le concepteur informaticien aura tendance à simuler cette spécification plutôt que de concevoir une réalisation optimisée.

I - 3.2. APPROCHE ALGORITHMIQUE

C'est l'approche qui consiste à spécifier le comportement attendu du système en explicitant les algorithmes qui doivent régir son fonctionnement interne. La spécification définit alors une sorte de "machine abstraite" éventuellement indéterministe que le concepteur interprète pour en comprendre le ou les fonctionnement(s) et le recréer dans une réalisation concrète. Le GRAFCET et les modèles axiomatiques cités plus haut participent de cette approche. En ce qui concerne la spécification initiale, cette approche nous paraît criticable dans la mesure où un grand nombre de choix algorithmiques échappent au concepteur.

I - 3.3. APPROCHE COMPORTEMENTALE

Cette approche consiste à voir le système à décrire comme une "boîte noire" qui réalise une transformation entrées-sorties. La spécification se limite à décrire les relations (fonctionnelles et temporelles) qui lient les entrées et sorties, sans préjuger d'aucune façon de la manière dont cette transformation doit être conçue. C'est l'approche empruntée implicitement quand on décrit une spécification en langue naturelle en la complétant de formules mathématiques, de chronogrammes, de tables de vérité, etc. Ces descriptions pouvant être imprécises, des méthodes ont été proposées pour leur donner un cadre plus précis, à l'aide de formulaires de description de variables et de fonctions [32] . Enfin, l'accent a été mis par [34] et [35] sur les qualités du langage mathématique -précision, compréhension universelle, généralité, aptitude à la transformation et à la démonstration- pour la spécification des programmes.

Au vu de cette typologie des approches de spécification, nous proposerons que les spécifications initiales soient de type comportemental, dans la mesure où ce type de spécification permet de minimiser les risques de sur-spécification, en laissant donc au concepteur une liberté de choix maximale, et où, dans de nombreux cas, cette approche est la plus naturelle et permet donc de minimiser les risques de distorsions entre l'intention informelle du "client" et la première expression formelle de cette intention.

Au chapitre II, nous présenterons brièvement un modèle comportemental de description des systèmes temps réel.

Il reste qu'au cours d'une conception descendante, le concepteur passera progressivement du niveau comportemental au niveau algorithmique, puis du niveau algorithmique au niveau structurel, et que les autres approches seront utiles pour décrire ces étapes intermédiaires de la conception. Le niveau algorithmique fera l'objet du chapitre III. Le problème de la validation formelle du passage entre les niveaux comportemental et algorithmique, très peu étudié à ce jour en ce qui concerne les systèmes temps réel, dépasse largement le cadre de ce travail, et constitue certainement un sujet fondamental pour une étude ultérieure. En l'absence de résultats rigoureux dans ce domaine, nous avons adopté une attitude pragmatique, correspondant d'ailleurs aux préoccupations de nos interlocuteurs industriels, consistant à approcher cette validation par simulation de la description algorithmique, exprimée en GRAFCET.

En ce qui concerne le passage du niveau algorithmique au niveau structurel, c'est-à-dire à la réalisation, deux approches seront proposées (chapitres IV et V) selon la rigueur des contraintes temporelles imposées au système.

I - 4. PROBLÈMES D'IMPLANTATION

L'avantage de passer par un niveau de description algorithmique est évident dans la mesure où, jusqu'à ce niveau, le concepteur peut effectuer un certain nombre de choix fondamentaux sans tenir compte des spécificités du matériel d'implantation. Dans ce domaine des systèmes logiciels non temps réel, la conception s'arrête à ce niveau, la réalisation effective incombant alors au compilateur du langage utilisé. Dans la même optique, de nombreux langages de haut niveau ont été proposés pour la programmation temps réel, qui comme tous les langages évolués, sont fondés sur des mécanismes d'abstraction permettant de cacher au programmeur des détails d'implantation liés à l'architecture. On peut penser qu'il est illusoire de vouloir compiler automatiquement des systèmes temps réel -c'est-à-dire des systèmes dont la correction dépend du temps d'exécution des tâches qui les composent- programmés à l'aide de tels langages -dont la sémantique soit indépendante du matériel sur lequel ils sont implantés-. En fait de nombreux langages

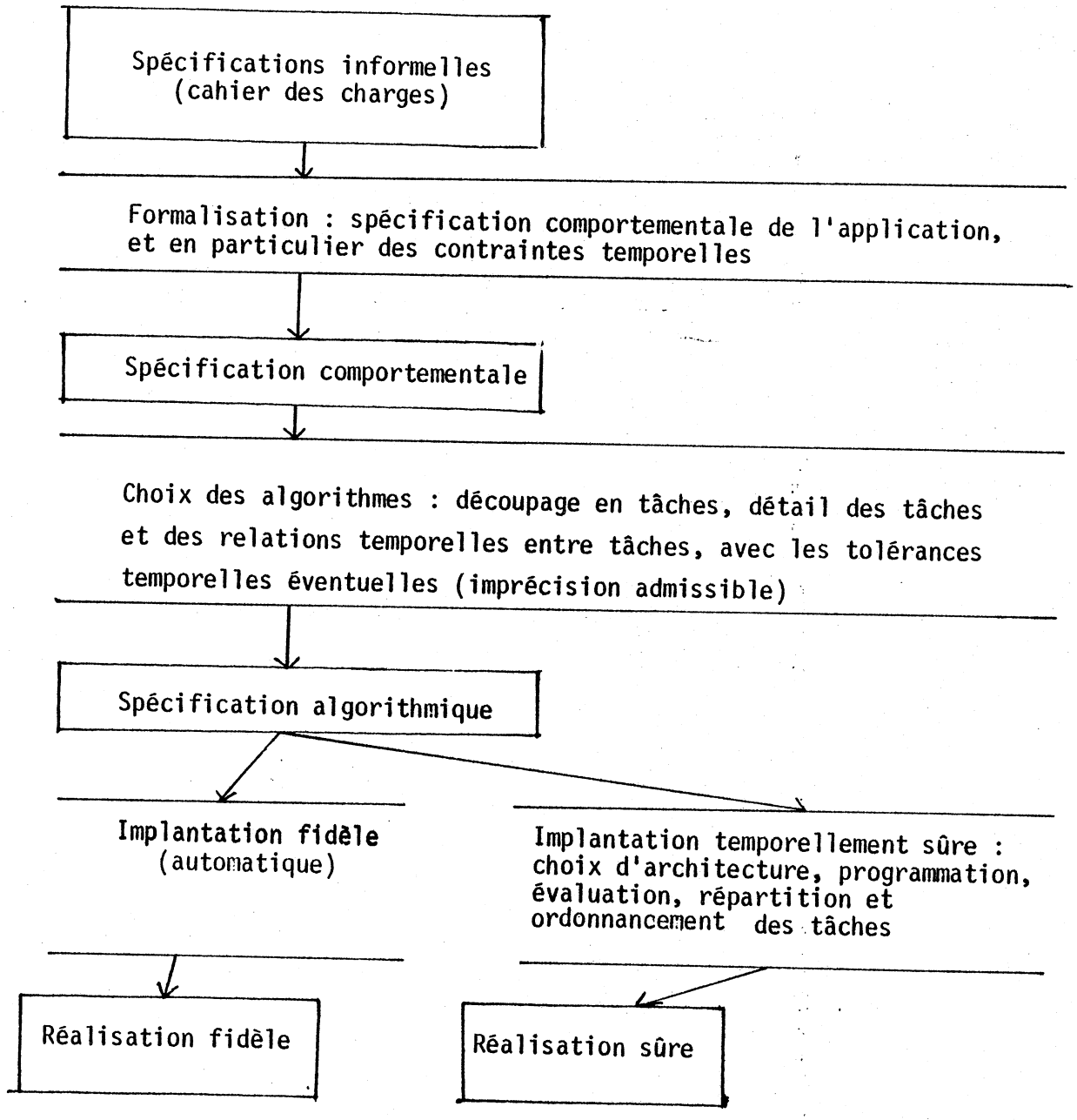
possèdent une sémantique purement abstraite et sont fondés sur l'hypothèse que les matériels informatiques sont suffisamment rapides pour que cette sémantique puisse être réalisée sans qu'apparaissent de distorsions temporelles significatives. D'autres langages tentent de minimiser ces distorsions en laissant au programmeur une part importante des choix de réalisations (ordonnancement) au moyen de primitives de bas niveau.

Nous aborderons les problèmes d'implantation en distinguant deux types de systèmes :

. Les systèmes où le respect des contraintes temporelles est impératif en toutes circonstances. Ces systèmes apparaissent dans des domaines à fortes contraintes de fiabilité, et ne peuvent qu'être implantés "à la main". Cette implantation, que nous appellerons "temporellement sûre", implique la connaissance détaillée du matériel et de ses performances, et une étude minutieuse et coûteuse de l'ordonnancement des tâches. Ce type d'implantation est étudié au chapitre V, et nous montrons sur quelques exemples simples de stratégies d'ordonnancement comment le respect des contraintes temporelles peut être prouvé.

. Les systèmes où la violation exceptionnelle des contraintes temporelles peut être corrigée par un relai humain ou automatique. L'important est alors moins de minimiser ces violations que de les détecter. Dans ce contexte, nous proposerons (Chapitre IV), de réaliser automatiquement une implantation "fidèle" à partir d'un langage évolué, c'est-à-dire une implantation se comportant comme la machine abstraite spécifiée par le programme, moyennant une imprécision temporelle, explicitement délimitée, vis-à-vis de l'environnement, ou détectant tout risque de distorsion par rapport à ce comportement. Un prototype d'interpréteur fidèle de GRAFCET a été réalisé, dont nous pensons que le noyau peut être facilement généralisé à d'autres langages sources.

Les étapes successives de la conception d'un système temps réel, selon la démarche que nous proposons, sont schématisées par le tableau suivant :



Nous allons illustrer sur un exemple très simple la démarche que nous proposons.

Exemple

Cet exemple est tiré de la réalisation d'un système de conduite de tir sur microprocesseurs.

Ce système doit en particulier calculer l'intersection d'une droite $y = c'$ et d'une parabole $y = ax^2 + bx + c$ (la droite $y = c'$ représente une droite définissant un côté du domaine de tir et la parabole $y = ax^2 + bx + c$ la prédiction de la trajectoire de la cible).

A) *La spécification comportementale de cette partie de la conduite de tir est :*

a) Définition des entrées-sorties

2 entrées (a,b,c) et c'

3 sorties x', x" et "perte de contact"

perte de contact = 1 si l'équation $y = ax^2 + bx + c - c'$ n'a pas de solution

= 0 sinon

x' et x" sont les solutions de l'équation $y = ax^2 + bx + c - c'$ lorsqu'elles existent.

b) Définition des relations avec le temps

Le système impose une fréquence de saisie f_1 pour (a,b,c) et de f_2 pour c'.

"Perte de contact" doit être émise pour chaque nouvelle valeur de c' et de (a,b,c) et ce avant l'écoulement d'un temps Δ_1 pour(abc) Δ_2 pour c'.

x' et x" doivent être émises pour chaque nouvelle valeur de c' ou de (a,b,c) lorsque x' et x" sont définies et ce avant l'écoulement d'un temps Δ_1 pour (abc) et Δ_2 pour c'.

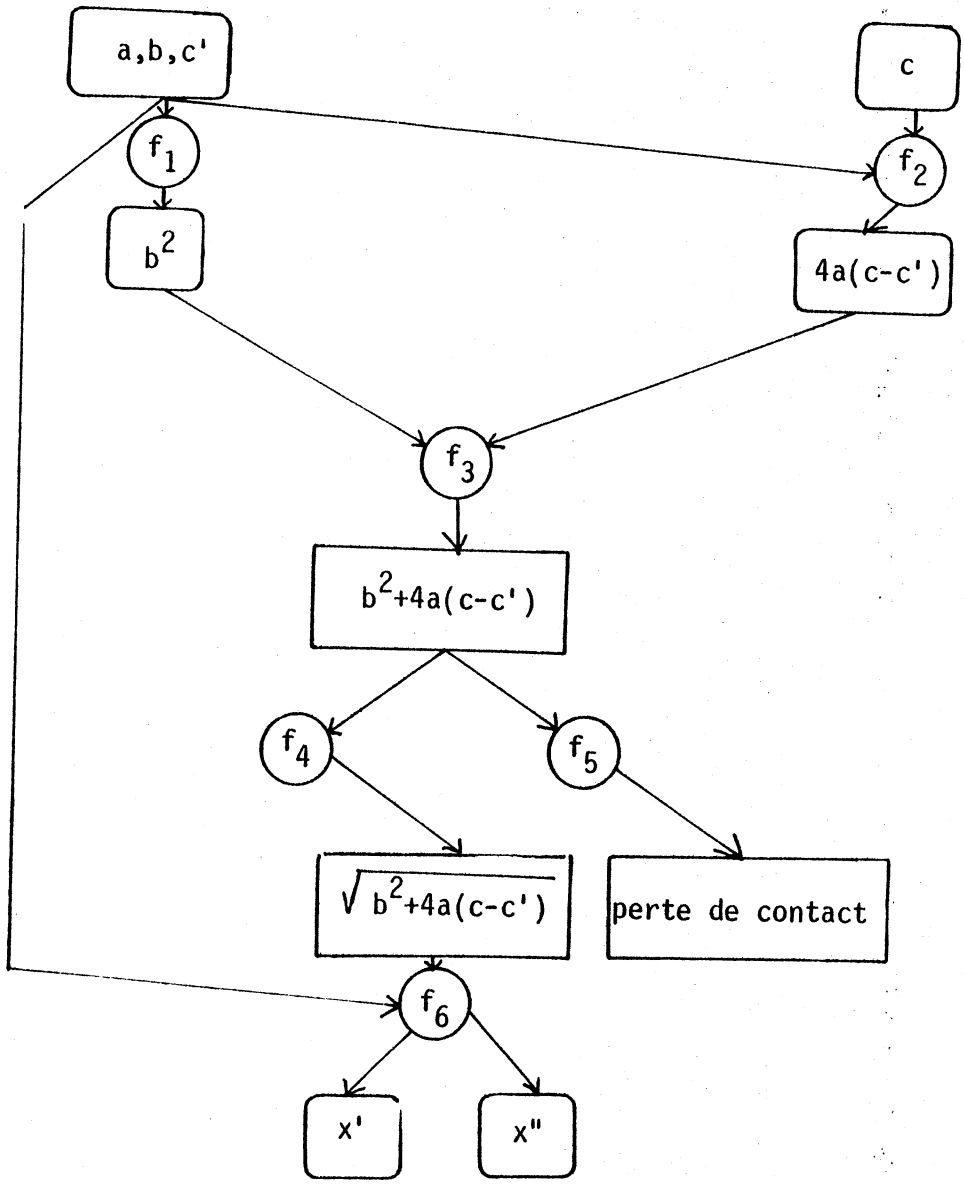
Il est à remarquer que la spécification comportementale n'induit pas de solution particulière (par exemple le concepteur garde le choix d'une solution calculée ou tabulée).

B) *La spécification algorithmique*

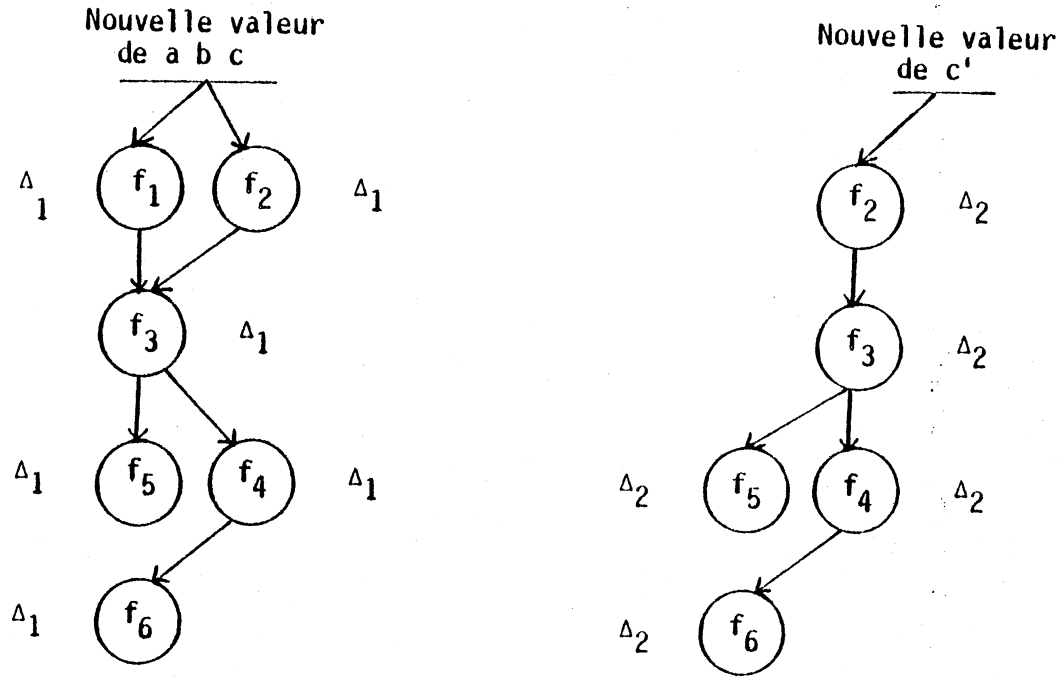
Le premier travail du concepteur est de décomposer le problème. Cela lui permettra d'exprimer la solution de son système sous forme de plusieurs sous fonctions explicitées à l'aide d'un algorithme, ces fonctions (ou tâches) se partageant un certain nombre de données et ayant certaines contraintes d'ordonnancement. Ceci peut par exemple se donner sous la forme d'un graphe de donnée et d'un graphe d'ordonnancement.

Dans notre exemple nous avons choisi comme algorithme le calcul des racines de l'équation (plutôt que la recherche des racines dans une table). Cet algorithme peut se décomposer en sous fonctions. Les relations entre ces sous fonctions peuvent être exprimées par les graphes suivants :

Graphe de donnée



Graphe d'ordonnancement



Ce graphe permet d'exprimer les relations d'ordonnancement entre les fonctions ainsi que les délais de réponse.

C) *L'implantation*

Le concepteur doit écrire son logiciel et choisir son matériel. Dans l'exemple nous avons par exemple choisi une implantation monoprocesseur avec les ordonnancements figés suivants :

Ordonnancement 1

Nouvelle valeur
de a b c

Ordonnancement 2

Nouvelle valeur
de c'

Supposons que nous voulons une implantation temporellement sûre, il faut vérifier que le système satisfait aux contraintes temporelles. Pour cela il faut :

a) Déterminer les temps maximum d'exécution des deux ordonnancements. Ceci nécessite d'étudier le temps d'exécution de chaque fonction. On peut alors déterminer le temps c_1 mis pour exécuter l'ordonnancement 1 et le temps c_2 mis pour exécuter l'ordonnancement 2.

b) Déterminer si les temps d'exécution permettent le respect des délais de réponse.
Sur notre exemple, une condition suffisante pour que ces délais soient respectés est que :

$$c_1 + c_2 \leq \text{Min}(\Delta_1, \Delta_2, T_1, T_2)$$

L'ensemble des idées présentées dans cette thèse a été appliqué à un exemple concret, de taille industrielle, qui est la spécification et la conception d'un poste d'aiguillage SNCF informatisé.
Cette étude nous a permis d'expérimenter le modèle de spécifications comportementales utilisé et de nous convaincre du bien fondé de l'idée d'implantation fidèle. Cet exemple sera présenté au chapitre VI.

CHAPITRE II

LES SPECIFICATIONS INITIALES

On appellera spécifications initiales, la représentation du système que le concepteur déduira du cahier des charges. Dans le cahier des charges, le concepteur rencontre :

Les spécifications opérationnelles

Exemples

- coût et contrainte sur le choix matériel
  - . compatibilité avec un matériel donné,
  - . choix à priori d'une technologie,
  - etc...
- environnement
  - . température
  - . consommation
  - . encombrement
  - etc...
- sûreté de fonctionnement
  - . fiabilité, sécurité, disponibilité
  - . tolérance aux erreurs transitoires
  - . maintenabilité (stocks, temps d'intervention, etc...)

Les spécifications fonctionnelles

Il s'agit de la donnée des

- entrées du système
  - . nature
  - . précision
  - . fréquence durée intervalle.

- Sorties du système
  . nature
  . précision
  . fréquence temps de réponse.

- Les fonctions à effectuer par le système, ce sont les relations entrées-sorties du système.

Ces spécifications fonctionnelles sont souvent données dans le même cahier des charges sous formes très diverses.

- Sous forme structurelle, en faisant référence à une réalisation dans une autre technologie (analogique ou câblée).
- Sous forme algorithmique à l'aide de modèles type GRAFCET ou R de P.
- Sous forme comportementale.
  Expression des relations entrées-sorties sous forme de relations mathématiques.

Le premier travail du concepteur sera d'unifier la forme de ces spécifications fonctionnelles. Dans [9] les auteurs ont souligné les différents avantages et inconvénients des trois types de spécifications. Il est évident (absence de surspécification, universalité...), qu'il vaut mieux adopter, pour les spécifications initiales, une spécification de type comportemental . Pour cela, dans [7,8], un outil est proposé. Nous allons en rappeler les principales notions.

II - 1. MODÈLE MATHÉMATIQUE PERMETTANT LA SPÉCIFICATION COMPORTEMENTALE [7,8]

Ce modèle est basé sur la notion métrique du temps propre aux systèmes temps réel et sur une formalisation, à l'aide de suites du concept d'événement.

II - 1.1. TEMPS

Ce modèle fait référence à un temps absolu correspondant à celui d'un observateur externe au système étudié. Le domaine de définition T des temps est représenté soit par $\mathbb{R}$ soit par $\mathbb{Z}$. La terminologie employée fait appel indifféremment à des instants, moment ou date.

II - 1.2. EVENEMENTS

Un événement est défini comme une transition survenant entre deux états d'un système. L'occurrence d'un événement est considérée de durée nulle.

- Définition d'un événement e

Un événement e est une suite croissante d'instants, finie ou infinie où e(n) représente la date de la $n^{ième}$ occurrence.

- Remarque importante :

Dans la suite du travail, nous considèrerons que tout événement e ayant un nombre infini d'occurrences a pour propriété :

$$e(n) \to +\infty \quad \text{quand} \quad n \to +\infty.$$

(Ceci est toujours vérifié quant T est associé à Z).

II - 1.3. VARIABLE

Une variable X est un couple de suites $(x,\tilde{x})$

où

- x est la suite des valeurs prises au cours du temps.(x_0, x_1, x_n) indique que la variable a pris pour première valeur x_1 puis a changé de valeur et a pris x_2, ... puis x_n.

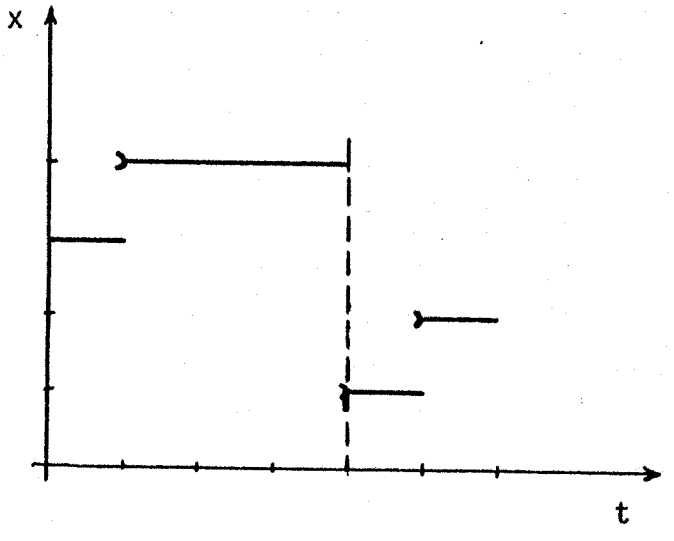
- $\hat{x}$ est la suite des dates associées au changement de valeur.

ex $X = \{x, \hat{x}\}$

avec $x = \{3, 4, 1, 2\}$

$\hat{x} = \{1, 4, 5, 6\}$

est représenté par



Remarque :

- une variable est une fonction de T dans $\mathbb{D}$ constante par morceaux,
- Toute fonction x(t) ne peut bien entendu pas se mettre sous la forme $(x(n)\ \hat{x}(n))$. La notion de variable définie plus haut est plus "informatique".

A l'aide de ces notions de base, du langage mathématique et de la logique des prédicats, il est possible de construire certaines primitives d'ordre général facilitant la description d'un système temps réel.

II - 1.4. OPERATEURS

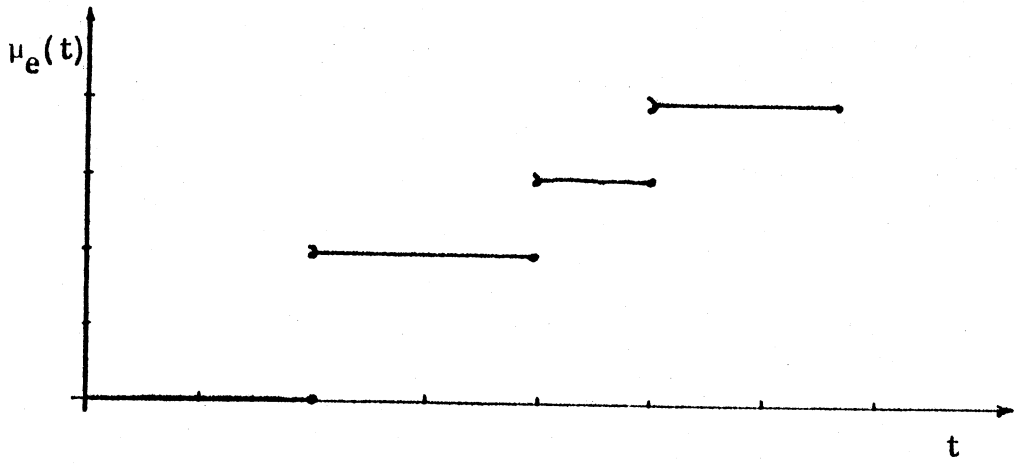
II - 1.4.1. Compteurs

A un événement e est associé un compteur μ_e qui est une application de T vers $\mathbb{N}$. $\mu_e(t)$ représente le nombre d'occurrences de e qui se sont produites strictement

avant la date t. μ_e est une fonction en escalier, à valeur entière, croissante et continue à gauche.

$\forall\ t \in T,\ \mu_e(t) = \max \{n \in \mathbb{N},\ e(n) < t\}$

Ex. Soit e = (2, 4, 5, 7)



De même, est défini $\mu_e^+(t)$

$\forall\ t \in T\ \ \mu_e^+(t) = \max \{n \in \mathbb{N},\ e(n) \leq t\}$.

qui est la fonction continue à droite.

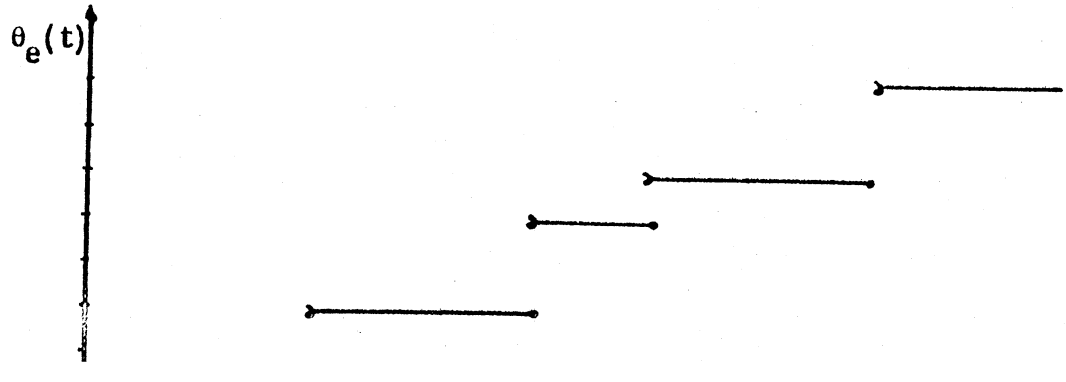
On remarquera qu'un événement e se produit à l'instant t si et seulement si $\mu_e(t) < \mu_e^+(t)$.

II - 1.4.2. Fonctions de dernière occurrence

$\theta_e(t)$(resp $\theta_e^+(t)$) associe la date de la dernière occurrence de l'événement e, qui précède l'instant t au sens strict (resp large). On note ces fonctions

$\theta_e = e \circ \mu_e$(resp θ_e^+ eo μ_e^+)

Ex. Soit e = (2, 4, 5, 7)



II - 1.4.3. Evènement conditionnel

Une condition c est une variable booléenne temporisée

$\forall t \quad c(t) \in \{vrai, faux\}$

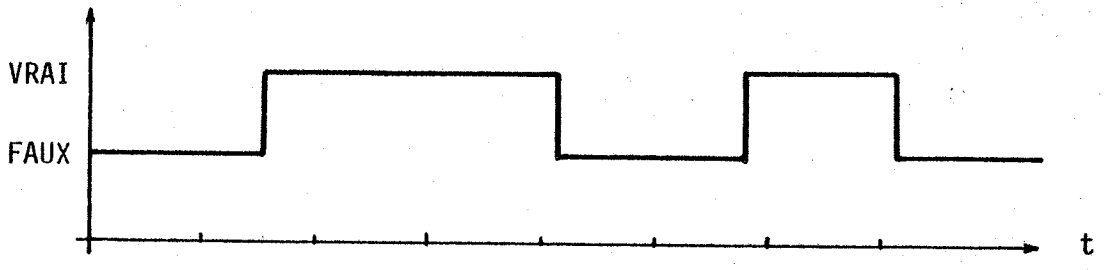
Un événement conditionné par c noté e/c se produit chaque fois que c est vraie et que e a une occurence.

$$\forall t \in T \; \mu_{e/c}(t) = card \{n \in \mathbb{N}^* / e(n) < t \text{ et } c(e(n)) = vrai\}$$

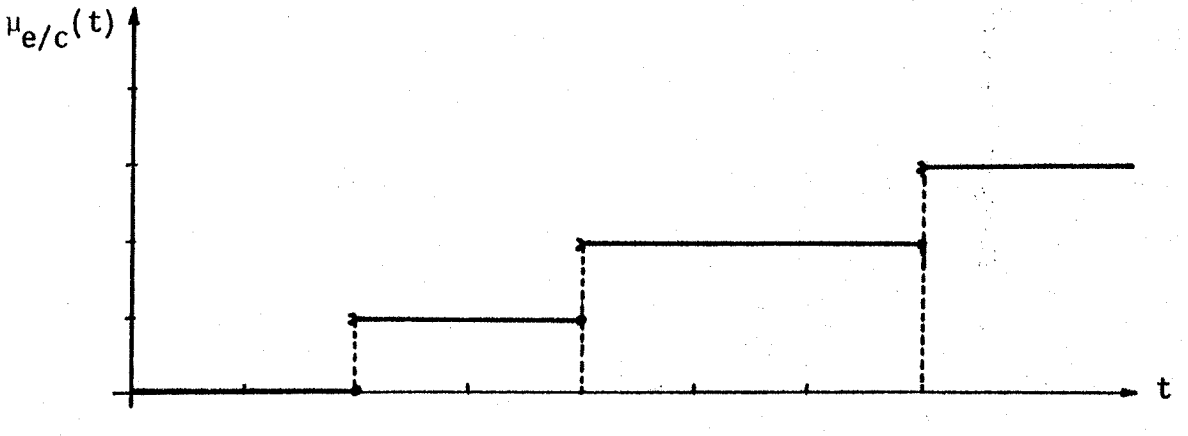
Exemple

soit e = (2, 4, 5, 7)

soit c défini par



e/c = (2,4,7) car c(5) = faux



II - 1.4.4. Somme de deux événements

La somme de deux événements e et f est définie comme l'événement qui se produit chaque fois que e ou f surviennent :

$$\mu_{e+f}(t) = \mu_e(t) + \mu_f(t)$$

II - 2. UTILISATION DU MODÈLE POUR LA SPÉCIFICATION DES SYSTÈMES TEMPS RÉEL

Spécifier un système temps réel, c'est spécifier les relations qui doivent exister entre les entrées et les sorties du système.

Chaque entrée I d'un système est vue soit comme une fonction i(t), soit comme une variable $(i,\hat{i})$ où l'événement associé $\hat{i}$ est l'événement "i a changé de valeur".

Chaque sortie S sera vue comme une variable $(s,\hat{s})$. Il s'agira donc d'établir les relations qui doivent exister entre les sorties et les entrées.

II - 2.1. EXEMPLE DE SPECIFICATION

Reprenons l'exemple de la conduite de tir (chap. I) et spécifions le problème à l'aide du modèle :

1) Soit $A = (a, \hat{a})$, $B = (b, \hat{b})$, $c = (c, \hat{c})$

Les entrées a b c changeant en même temps, on a $\hat{a} = \hat{b} = \hat{c}$.

Comme ces entrées changent de façon périodique de période T_1, on a $\hat{a}(n) = \hat{b}(n) = \hat{c}(n) = (n-1)T_1 + \psi_1$.

Soit $c' = (c',\hat{c}')$ la variable d'entrée c'. Le changement de valeur est périodique de période T_2, on a $c'(n) = nT_2 + \psi_2$.

2) Soit l'événement $\hat{ch}$, l'événement "une entrée a changé"

$\hat{ch} = \hat{c} + \hat{c}' = \hat{a} + \hat{c}' = \hat{b} + \hat{c}'$ car $\hat{a} = \hat{b} = \hat{c}$

Posons $a_1(n) = a(\mu_{\hat{a}}(\hat{ch}(n))$, $a_1(n)$ représente la valeur de a à l'instant $\hat{ch}(n)$.

De même, posons $b_1(n) = b(\mu_{\hat{b}}(\hat{ch}(n))$

$c_1(n) = c(\mu_{\hat{c}}(\hat{ch}(n))$

$c'_1(n) = c'(\mu_{\hat{c}'}(\hat{ch}(n))$

3) Spécification de la perte de contact

La sortie perte de contact doit être faite avant un délai de réponse Δ, on a donc $\forall n \quad \hat{ch}(n) + \Delta_1 \geq \hat{S}(n) \geq \hat{ch}(n)$

$S(n) = 1$ si $a_1(n)\, x^2 + b_1(n)\, x + c_1(n) - c'_1(n)$ n'a pas de racine

$S(n) = 0$ sinon.

Spécification des sorties x' x"

x'(n) doit être sortie avant un délai de réponse Δ_2 de même pour x"(n).

$\forall n \quad \hat{ch}(n) + \Delta_2 \geq \hat{x}'(n) \geq \hat{ch}(n)$

$\hat{ch}(n) + \Delta_2 \geq x"(n) \geq \hat{ch}(n)$

$x'(n) = x'(n-1)$ si $a_1(n)\, x^2 + b_1(n)\, x + c_1(n) - c'_1(n) = 0$

$x"(n) = x"(n-1)$ n'a pas de racines.

x'(n) et x"(n) sont les racines de l'équation sinon.

Remarque

Lors de la spécification, l'utilisation du modèle permet de découvrir par rapport au chapitre 1 :

- une imprécision : quelle est la valeur des sorties x' et x" lorsque les entrées n'ont pas de racines? |levée par $x'(n) = x'(n-1)$ $x"(n) = x"(n-1)$
- une éventuelle incomplétude : les sorties x' et x" sont décorellées temporellement à l'intérieur de Δ_2, la spécification permet la sortie de x' à l'instant $\hat{ch}(n)$ et la sortie de x" à l'instant $\hat{ch}(n) + \Delta_2$, ce qui veut dire qu'il est permis que pendant Δ_2, x' soit une racine de la nouvelle équation et x" une racine de l'ancienne. Cela peut traduire une éventuelle incomplétude de la spécification.

II - 2.2. SPECIFICATION DE SYSTEME COMPLEXE A L'AIDE DE CE MODELE

L'utilisation de ce modèle pour les spécifications initiales de systèmes de taille "industrielle" (voir par exemple le chapitre IV, Spécifications de l'interpréteur, chapitre VI Spécifications du PAI), nous a permis de nous rendre compte que :

- on peut éviter les surspécifications et les ambiguités,
- la nécessité de décrire le problème de cette façon permet une meilleure compréhension.

En revanche, elle est peu lisible pour un système complexe car la nécessité d'introduire des variables intermédiaires rend la spécification touffue.

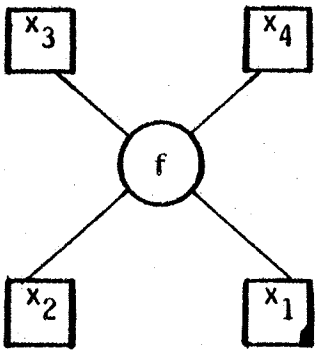
Il nous a semblé nécessaire de respecter, pour les systèmes complexes, un certain nombre de principes.

1) Donner une vue d'ensemble des relations entrées, variables intermédiaires, sorties. On peut le faire sous forme d'un graphe des variables type graphe de donnée. On peut par exemple adopter le formalisme suivant : le graphe est un graphe biparti.

On associe un noeud case à chaque variable du système et un noeud rond à chaque relation liant les variables du système.

Ainsi, la relation fonctionnelle

$(x_1, x_2) = f(x_3, x_4)$ se représente sur le graphe par

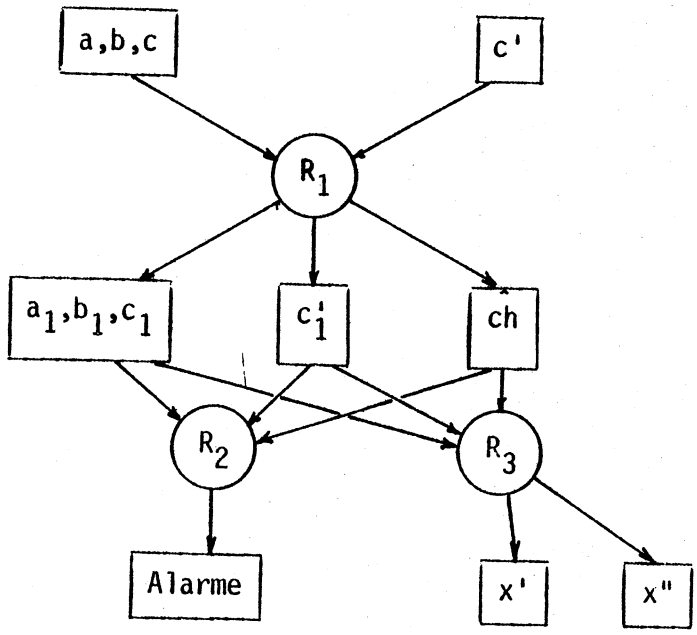


Une variable d'entrée se représente par une case avec une flèche entrante et une variable de sortie par une case avec une flèche sortante.

2) Décrire les relations entre les variables (noeud rond). Cela doit se faire systématiquement en deux temps :

- relation temporelle, ce sont les liens existant entre $\hat{x}$ et $\hat{y}$.
- relation fonctionnelle, ce sont les liens existant entre x(n) et y(n).

Aussi, l'exemple II.2.1. peut se décrire par :



Puis la description de R_1, R_2, R_3 telle qu'elle est faite en II.2.1.

Un cas particulier rencontré fréquemment lorsque les spécifications initiales sont tirées d'un cahier des charges (où les relations fonctionnelles sont données sous forme structurelle ou algorithmique), est la description de relations de type fonction entre les variables intermédiaires

$$(y_1, y_2, y_n) = f(x_1, x_2, x_p)$$

Décrire f peut se décrire systématiquement par :

- un événement d'activation de f notée $\hat{C}_f$

  c'est un événement défini de façon générale comme une somme d'événements associés aux variables d'entrées.

Cet événement C_f permet de décrire simplement la liaison temporelle entre entrée et sortie dans le sens où "la date de la $n^{ième}$ occurrence de l'affectation de la sortie par la fonction f correspond à la date de la $n^{ième}$ activation de f"

$$\forall i \quad \forall p \in \mathbb{N} \quad \hat{y}_i = \hat{c}_f$$

- la description de la fonction $y_i(p) = f(x_1', x_2', x_n')$ où x_1', x_2', x_n' sont les valeurs de x_1, x_2, x_n à la date $\hat{c}_f(p)$.

Ce mode de description a été largement utilisé pour la spécification initiale du PIA (chapitre VI).

Il est à noter qu'une telle démarche (graphe de variable, description des relations), est compatible avec une méthode descendante. Une relation trop difficile à décrire peut elle-même se mettre sous la forme d'un graphe de variables (où seront introduites les variables intermédiaires nécessaires), complété par la description des relations "élémentaires". Il est bien évident que le procédé peut être répété. Cependant, plus les variables intermédiaires sont nombreuses, plus la spécification sera difficile à relire.

CHAPITRE III

LES SPECIFICATIONS ALGORITHMIQUES

Les concepteurs de systèmes temps réel complexes utilisent un certain nombre d'outils pour passer des spécifications initiales à la réalisation finale (matériel et logiciel) de leur système. Certains de ces outils (outils graphiques tels que réseaux de Petri ou Grafcet), sont plus proches des spécifications initiales. D'autres outils tels que les langages temps réel, sont plus proches de l'implémentation.

Ces outils ont généralement comme point commun l'expression de la solution sous forme d'un ensemble de tâches (P-O) et de l'expression de la synchronisation de ces tâches, soit entre elles, soit entre les tâches et le monde extérieur (P-C). L'expression de cette solution à l'aide de ces outils peut être considérée comme la spécification algorithmique du système temps réel. Nous allons voir dans quelle mesure les outils utilisés permettent :

- la vérification de la conformité aux spécifications initiales et en particulier du respect des contraintes temporelles,
- la vérification de l'implémentation future par rapport aux spécifications algorithmiques exprimées à l'aide de ces outils.

III - 1. LES MODÈLES EXISTANTS

On peut distinguer deux types de modèles.

III - 1.1. LES MODELES SYNCHRONES (Réseaux de Petri interprétés, Grafcet) [9, 10, 11, 18, 28].

Dans ces modèles, tout se passe comme si l'on disposait d'une infinité de processeurs infiniment rapides. La règle est donc l'instantanéité des réactions des tâches à l'environnement. Lorsque les tâches ne sont pas de durée infiniment petite, elles sont de durée précisée. Aussi, à l'aide de ces modèles, le concepteur décrit une machine abstraite synchrone.

III - 1.2. EXEMPLE DE MODELE SYNCHRONE : LE GRAFCET [9 - 18]

Nous nous proposons d'utiliser le GRAFCET comme modèle de description des synchronisations entre les tâches et le monde extérieur. Ces tâches elles-mêmes seront décrites dans un langage de type procéduriel permettant aussi la description des actions de sorties.

Un grafcet est un graphe sur lequel est superposée une interprétation.

a) Le graphe est un graphe fini biparti

Les deux types de noeuds sont :

- les étapes (représentées par des ronds graphiquement),
- les transitions (représentées par des traits graphiquement).

On note E l'ensemble des étapes et e_i une étape

$$E = \{e_1, e_2, \ldots e_n\}$$

On note T l'ensemble des transitions et t_i une transition

$$T = \{t_1, t_2, \ldots t_n\}$$

. Etapes actives

- l'état du système est représenté par un vecteur

$X = \{x_1, x_2, \ldots x_n\}$, où x_i peut prendre comme valeur 1 ou 0.

Si $x_i = 1$, on dit que l'étape e_i est active (ceci est représenté graphiquement par une marque à l'intérieur du cercle).

. Transitions validées

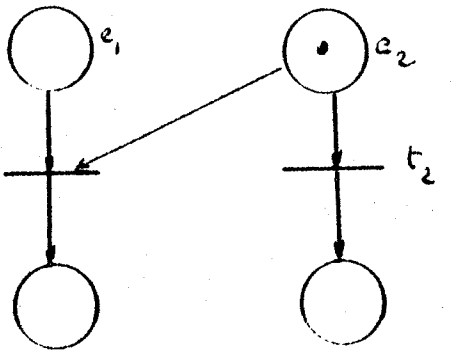
- soit t_i^- l'ensemble des étapes amonts de la transition T_i
- soit $X_i = (x_{i1}, \ldots x_{in})$ le vecteur tel que $x_{iJ} = 0$ si $e_J \notin t_i^-$

$x_{iJ} = 1$ si $e_J \in t_i^-$

On dit que t_i est validée par X si $X_i \leq X$.

(t_i est validée si toutes ces étapes amonts sont actives).

Exemple



e_2 est active. t_2 est validée.

b) L'interprétation

. A toute place est associée une tâche (éventuellement vide).

A toute transition est associée une réceptivité composée d'un événement (éventuellement l'événement toujours occurrent noté e) et d'une condition (éventuellement toujours vraie).

- l'événement traduit le passage de la valeur d'une variable d'un domaine de valeur à l'autre.
- la condition peut porter sur les valeurs des variables du système et éventuellement sur l'état d'activité ou d'inactivité des étapes.

. Transitions franchissables

- une transition est franchissable si :
  - . elle est validée,
  - . son événement survient,
  - . sa condition est vérifiée.

. Mise à feu d'un ensemble de transitions franchissables

Soit T un ensemble de transitions franchissables, la mise à feu de ces transitions consiste à désactiver les étapes amonts de ces transitions et à activer les étapes avales.

Le Grafcet est ici vu comme une machine permettant l'activation des tâches associées aux étapes, en fonction de l'arrivée des événements et de la modification des conditions. Le Grafcet a le comportement suivant (dérivé de [9] et [18]).

Soit X_0 l'état de départ appelé état initial.

Point 1 : exécuter toutes les tâches associées aux étapes actives.
aller au point 4.

Point 2 : à l'arrivée d'un événement externe, déterminer l'ensemble des transitions franchissables T. Si $T = \emptyset$ aller au point 2 sinon aller en 3.

Point 3 : mise à feu des transitions franchissables T ou T^2.
Exécution des tâches associées aux étapes qui étaient inactives avant la mise à feu et qui sont actives ensuite.

Point 4 : déterminer l'ensemble des transitions franchissables T^2 sur occurrence de e si $T^2_e = \emptyset$ aller au point 2 sinon aller en 3.

Remarque 1

Ainsi définies, les tâches sont équivalentes à la notion d'actions impulsionnelles (éventuellement complexes). La notion d'action à niveau n'est pas utilisée. Cette notion peut être explicitée au niveau du corps de la tâche.

Remarque 2

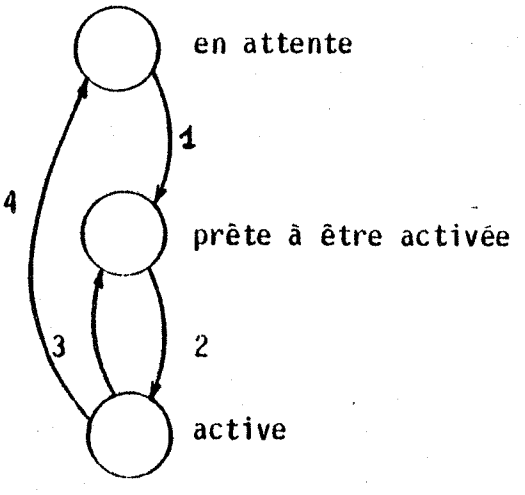
L'expression de temporisations peut se faire très simplement, en associant un temps aux transitions comme il est proposé dans [9]. Toute fin de temporisation sera considérée comme une variable externe.

III - 1.2. LES MODELES ASYNCHRONES [12 - 13 - 14 - 15 - 16 - 6]

Ce sont généralement des modèles dérivés des langages purement parallèles. Dans cette approche, les tâches sont considérées de durée non nulle et d'activation différable. De plus, on spécifiera implicitement ou explicitement une date limite d'activation des tâches. Généralement, ces modèles sont des langages de programmation de haut niveau, possédant des primitives de synchronisation (Sémaphores, Rendez-vous), inspirées des langages parallèles auxquels il a été ajouté la notion d'événement,d'attente sur événement, d'émission d'événement.

Dans un modèle asynchrone, une tâche peut être dans trois principaux états :

- en attente d'un événement ou d'une synchronisation,
- prête à être activée,
- active.



Seules les relations 1 et 4 sont exprimées dans les langages temps réel que l'on pourrait qualifier d'abstraits [12 - 13].

Dans les langages dits concrets [14, 15, 16], à l'aide en particulier du concept de priorité, le programmeur peut influer sur 2, 3. Mais le contrôle de [2, 3] n'est jamais complet et varie selon l'architecture cible visée. [Mono, Multi-processeurs, etc...]. De ce fait, l'utilisation des langages concrets au niveau

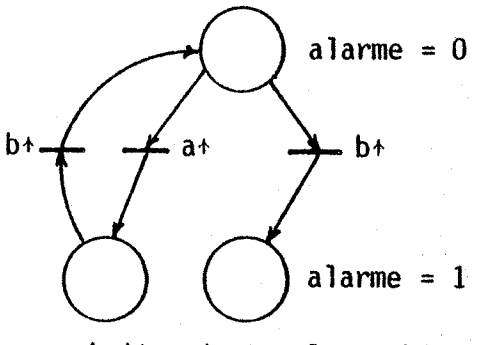
de l'étape de spécifications algorithmiques, dans notre démarche progressive est impossible (en effet,nous ne connaissons pas au niveau de la spécification algorithmique, l'architecture able). Nous ne parlerons donc à ce niveau que des langages synchrones et des langages asynchrones abstraits.

III - 2. UTILISATION DE CES MODÈLES DANS UNE DÉMARCHE PAR SPÉCIFICATION PROGRESSIVE

Quel que soit le modèle utilisé (Synchrone ou Asynchrone), le concepteur définit une machine abstraite. Cette machine abstraite a un comportement idéal. Il est bien-entendu impossible de réaliser une machine ayant exactement le comportement idéal décrit. Ceci pose évidemment un gros problème dans une démarche de conception, car il est impossible de vérifier l'implémentation (c'est-à-dire les étapes suivantes), en se référant aux spécifications algorithmiques. Voici deux exemples illustrant les difficultés rencontrées.

Exemple 1

Nous avons la spécification algorithmique suivante

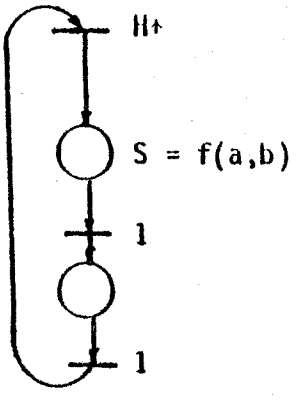


a et b étant deux signaux indépendants, la machine abstraite délivre une alarme chaque fois qu'il n'existe pas un signal a↑ entre deux signaux b↑ (ou entre l'instant 0 et b↑) et ce, quel que soit l'intervalle de temps existant entre les deux signaux. Il est bien évident qu'il est impossible à un concepteur de réaliser pratiquement un tel système.

En effet, si a et b arrivent très rapprochés la réalisation ne pourra classer l'arrivée de a et l'arrivée de b.

Exemple 2

Un système a trois entrées H (valeur 1 ou 0), a, b. Il délivre une sortie S. La spécification algorithmique est la suivante :



Le comportement de la machine abstraite est très claire. On a :

$$S[\hat{H}(n)] = f[a(\hat{H}(n)), b(\hat{H}(n))]$$

où $\hat{H}(n)$ est la date de la $n^{ième}$ occurrence de H↑.

Lors de la réalisation, le concepteur se posera trois questions :

1) Peut-on saisir tous les fronts montants de H↑ ?

2) Est-il important que les valeurs a b soient celles existant à l'instant H↑ ? Si non, quel est l'écart admissible ?

3) La nouvelle valeur de S doit-elle exactement sortir à l'instant H↑ ? Si non, quel est l'écart admissible ?

Pour répondre à ces questions, le concepteur devra se replonger dans les spécifications initiales (la spécification algorithmique ne lui suffisant pas).

Il apparaît donc très clairement que pour un système temps réel, la description algorithmique sous forme de modèle abstrait ne suffit pas, il faut aussi préciser les limites temporelles maximales qu'il ne faut pas dépasser à l'implémentation. Pour décrire ces limites, il suffit de décrire le flou temporel admissible sur les entrées (définition de l'imprécision admissible de connaissance des dates d'entrées), et le flou temporel admissible sur les sorties (Retard maximal

admissible sur les sorties). Toute machine concrète ayant le même comportement que la machine abstraite à ce flou temporel près, sera considérée comme conforme.

III - 2.1. FORMALISATION DE DERIVEE TEMPORELLE

III.2.1.1. Histoires

Soit $A = \{A_1, A_2, \ldots, A_n\}$ un ensemble de variables ($A_i = (a_i, \hat{a}_i)$) au sens du chapitre 2.

L'histoire de A est le n-uplé.

$(\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_n)$ où

$\tilde{a}_i$ est une fonction associant à tout instant t la valeur de la variable A_i à cet instant.

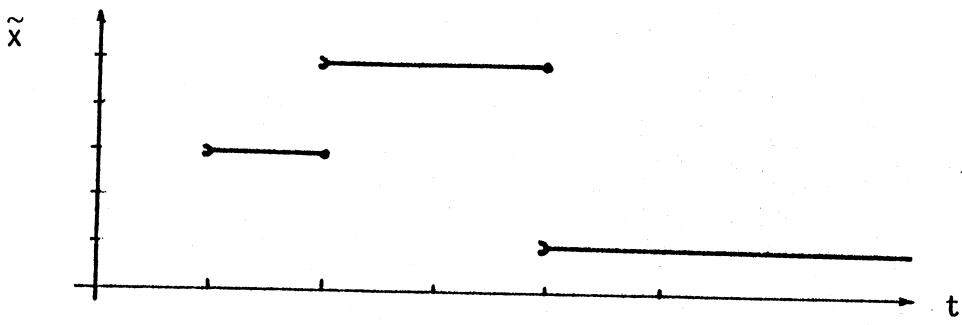
$$\tilde{a}_i = a \; 0 \; \mu_{\hat{a}}(t)$$

Ex. Soit $X = (x, \hat{x})$

où $x = (3, 5, 1)$

$\hat{x} = (1, 2, 4)$

$\tilde{x}(t)$ est représenté par



$\tilde{x}$ est la représentation de la variable sous la forme d'une fonction du temps et non pas de deux suites.

III-2.1.2. Machine abstraite

Soit S un système temps réel, soit H(E) l'ensemble des histoires possibles de ses entrées et H(O) l'ensemble des histoires possibles de ses sorties. Une machine abstraite réalisant S est une fonction f_S de H(E) dans l'ensemble des parties de H(O). En effet, la machine abstraite fait correspondre à une histoire de ces variables d'entrées, une (cas de la machine abstraite déterministe), ou un ensemble (cas d'une machine abstraite indéterministe) d'histoires des variables de sorties.

Remarques

. L'environnement d'entrée n'est décrit que par le domaine de définition des variables d'entrées. Son élaboration ne fait pas partie de la définition de la machine abstraite.

. Décrire un GRAFCET est bien définir une fonction f_S. Le GRAFCET [9,18] est d'ailleurs déterministe.

A une histoire des entrées, il existe une et une seule histoire des sorties.

. Le R de P Interprété est indéterministe ; à une histoire des entrées, le R de P Interprété associe une ou plusieurs histoires des sorties.

III-2.1.3. Image approchée d'une histoire

Soit $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n)$ un n-uple de réels positifs

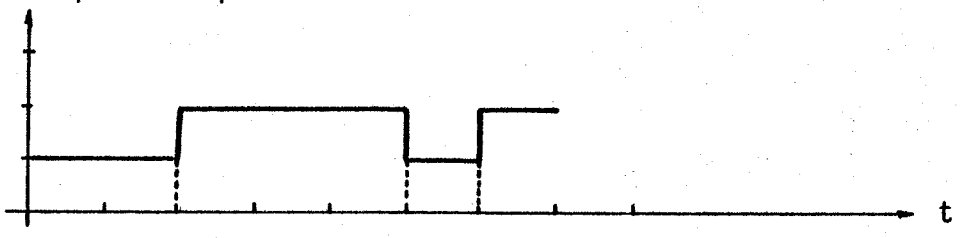
et $h = (\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_n)$ une histoire.

L'ensemble I_ε des images approchées de h à ε près, est défini par :

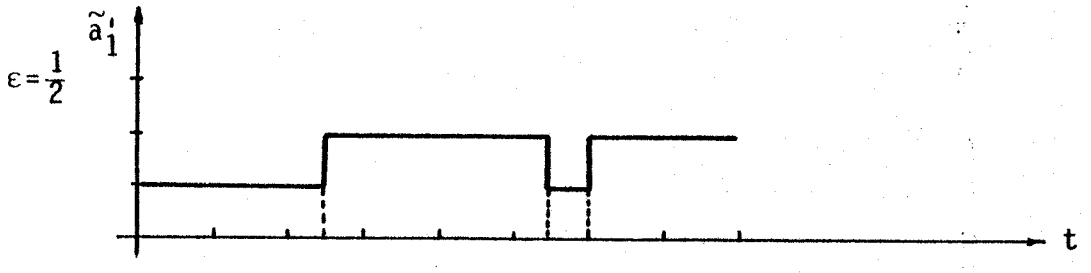
$$I_\varepsilon(h) = \{(\tilde{a}'_1, \tilde{a}'_2, \ldots, \tilde{a}'_n) \mid \forall\, t \in T \text{ et } \forall\, i \in \{1,\ldots,n\}$$
$$\exists\, t_j \in]t - \varepsilon_i, t] \text{ tel que } \tilde{a}'_i(t) = \tilde{a}_i(t_j)\}$$

Exemple

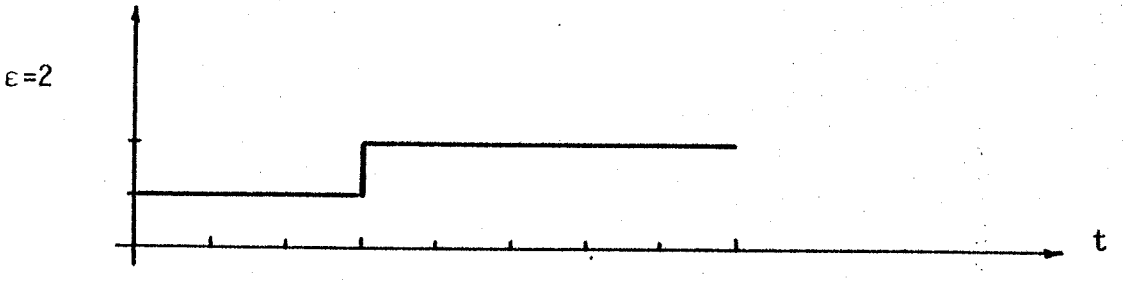
Soit $\tilde{a}$ représenté par :



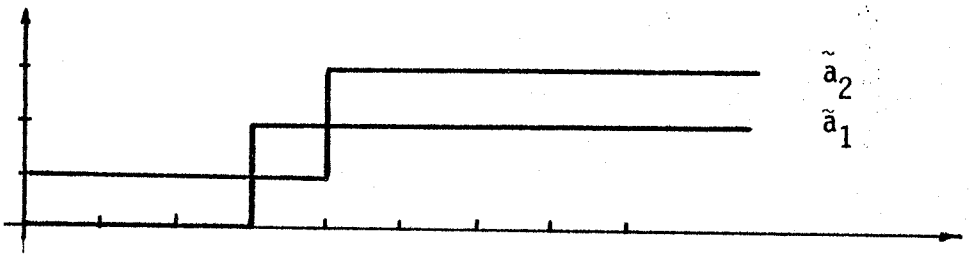
une image approchée à $\frac{1}{2}$ près peut être



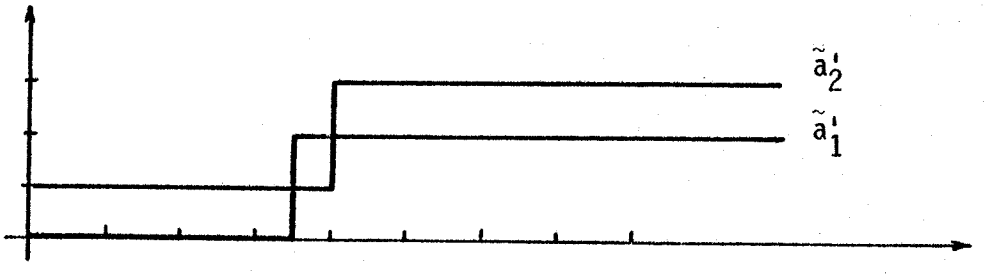
une image approchée à 2 près peut être



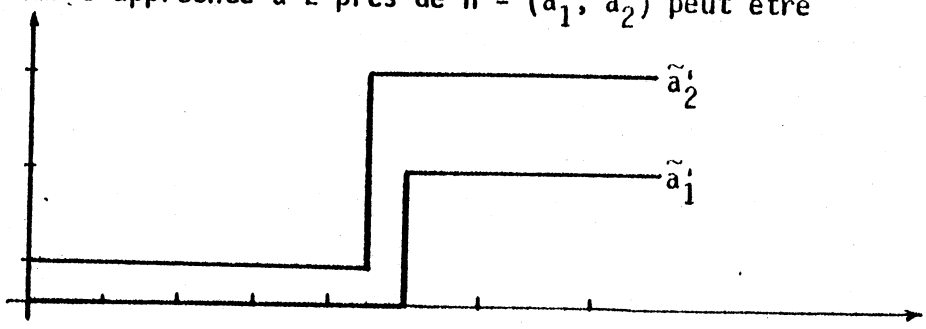
Exemple à deux variables



une image approchée à $\frac{1}{2}$ près de $h = (\tilde{a}_1, \tilde{a}_2)$ peut être



Une image approchée à 2 près de $h = (\tilde{a}_1, \tilde{a}_2)$ peut être



L'image approchée à 2 près modifie aussi l'ordre d'arrivée des événements, ce qui n'est pas vrai pour $\varepsilon = \frac{1}{2}$.

III-2.1.4. Comportement approché à (ε, Δ) près d'une machine abstraite

Soit f_S une machine abstraite, E l'ensemble de ses entrées, O l'ensemble de ses sorties.

Une fonction f'_S est dite avoir un comportement approché à (ε, Δ) près, si elle a la propriété suivante :

$$\forall\, h \in H(E)\ \exists\, h' \in I_\varepsilon(t)\ \exists\, k \in f_S(h')$$

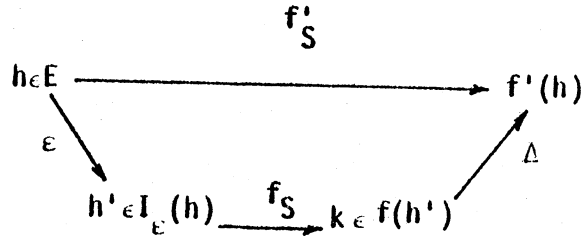
tel que $\quad f'_S(h) \in I_\Delta(k)$

ceci traduit le fait que f'_S est dit avoir un comportement approché à (ε, Δ) près si f'_S est la composition de trois fonctions :

. La première est la construction d'une image approchée des entrées à ε près

. La deuxième est f_S

. La troisième est la construction des sorties à Δ près



La formalisation faite dans ce paragraphe s'applique aussi bien aux machines synchrones qu'asynchrones.

III - 2.2. UTILISATION DE LA DERIVEE TEMPORELLE DANS UNE DEMARCHE DE CONCEPTION PAR SPECIFICATION PROGRESSIVE

Dans une démarche de conception progressive, il est nécessaire de :

- définir une machine abstraite,
- définir un comportement approché (ε,Δ) admissible de la machine abstraite.

Valider cette étape, c'est :

- s'assurer que toute fonction ayant un comportement à (ε,Δ) près de la machine abstraite, respecte les spécifications initiales.

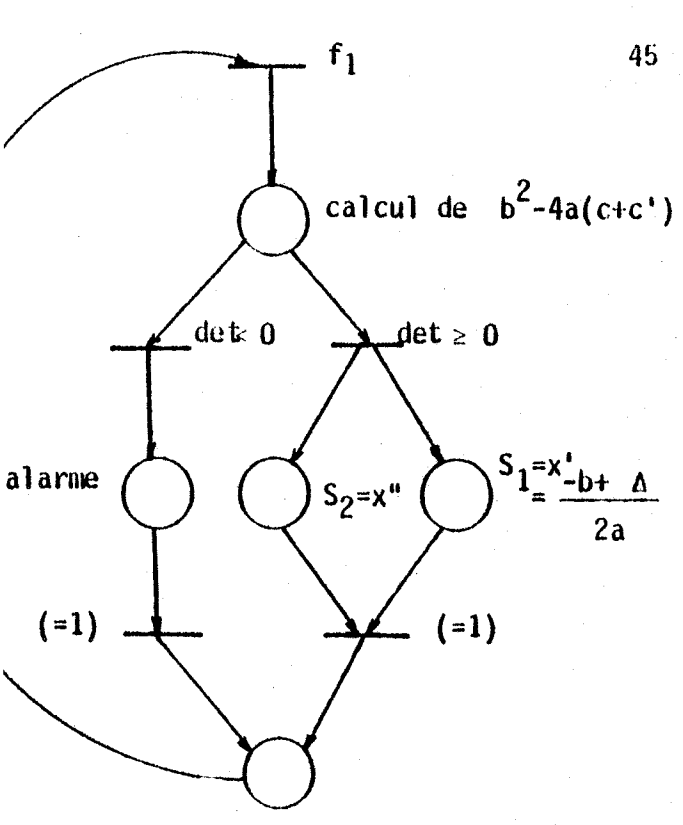
III - 3. UTILISATION D'UN MODÈLE PARTICULIER (LE GRAFCET)

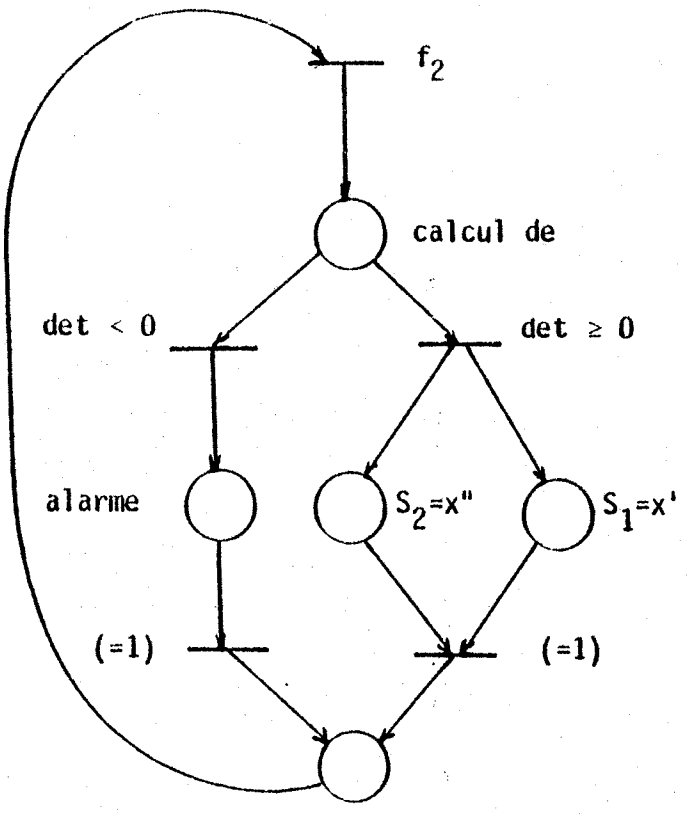
Le Grafcet tel qu'il est défini dans [9] est une machine abstraite déterministe. Lorsque le concepteur décrit son application à l'aide d'un GRAFCET G, il définit une fonction ma_G qui associe à toute histoire des entrées une et une seule histoire des sorties. Utiliser le grafcet dans une spécification progressive nécessitera la définition d'un comportement à (ε,Δ) près.

Voici deux exemples de définition d'un GRAFCET et de son comportement admissible approché.

Exemple 1

Les spécifications algorithmiques correspondant aux spécifications initiales données au paragraphe II - 2.1 (sous système de conduite de tir) peuvent être données par deux grafcets :





où f_1 et f_2 sont deux signaux de période T_1, T_2 signifiant les changements des variables (a,b,c) et c'.

Le système a trois sorties (S_1, S_2, Alarme).

Pour qu'une machine ayant un comportement approché à $\varepsilon = (\varepsilon_1, \varepsilon_2)$ $\Delta = (\delta_1, \delta_2, \delta_3)$ près soit admissible, il faut que :

$\varepsilon_1 < T_1$
$\varepsilon_2 < T_2$ en effet, on ne peut pas sauter une nouvelle valeur des variables.

$\varepsilon_1 + \delta_1 < \Delta_1$
$\varepsilon_1 + \delta_2 < \Delta_1$
$\varepsilon_2 + \delta_1 < \Delta_1$
$\varepsilon_2 + \delta_2 < \Delta_1$ ce qui correspond à la spécification des délais de réponse.

On remarquera que f_1 et f_2 ont des traitements complètement indépendants.

Exemple 2

- mesure de longueur de plot à l'aide d'une roue phonique.

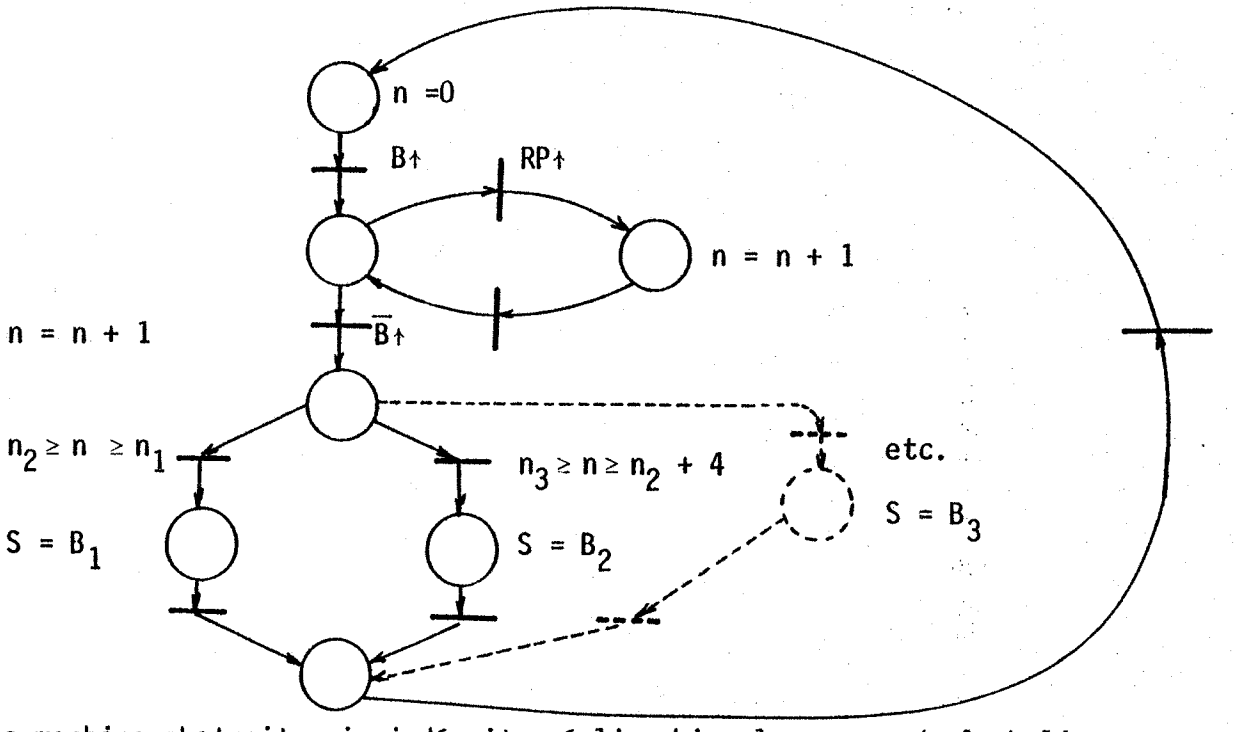
Une partie du régulateur de vitesse de métro a deux entrées :

- R_p la roue phonique envoie un signal tous les x centimètres
- B est un signal indiquant la présence ou l'absence d'un plot.

La sortie est le type de plot rencontré :

- $S = B_1$ si le plot mesure entre $n_1 x$ et $n_2 x$ centimères
- $S = B_2$ si le plot mesure entre $(n_2+4)x$ et $n_3 x$ centimètres
- $S = B_3$.

Le Grafcet décrivant cette partie du pilote est :



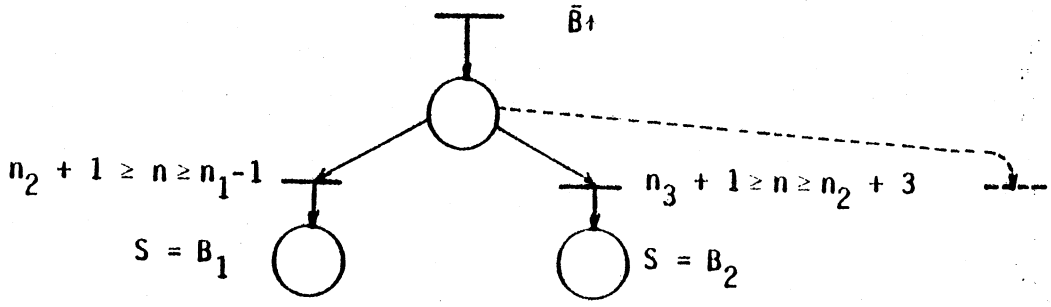
La machine abstraite ainsi décrite réalise bien la mesure de la balise.

Déterminons $\varepsilon = (\varepsilon_1, \varepsilon_2)$ ε_1 étant l'incertitude sur RP

ε_2 étant l'incertitude sur B

$\varepsilon_1 < T_1$ où T_1 est le temps minimal de l'envoi d'impulsion de la roue phonique. En effet, il ne faut pas "perdre" d'impulsion.

Déterminer ε_2 est plus délicat. La connaissance de B↑ et $\bar{B}$↑ doit être suffisamment précise pour ne pas fausser la sortie S. Si $\varepsilon_2 < T_1$ alors n prendra une valeur comprise entre $n_1 + 1$ et $n_1 - 1$, n_1 étant la valeur réelle.
La spécification de ε (T_1 T_1) obligera donc le concepteur à modifier son graphe au niveau de $\bar{B}$↑.



La spécification de Δ est en fait le délai de réponse admissible de la sortie.
On remarque sur les deux exemples que :

- trouver Δ est relativement simple, c'est en fait la définition du délai de réponse habituel,
- trouver ε est plus difficile.

Dans tous les cas, $\varepsilon_i < T_i$ où T_i est la période minimale de l'entrée e_i.
Puis il faut s'assurer que si e_i est "décalé" à une date comprise entre 0 et ε_i, le GRAFCET continue à décrire un comportement souhaité. La difficulté réside bien entendu dans le fait que le décalage de l'entrée entraîne l'inversion possible de l'ordre des entrées par rapport à l'ordre des entrées réelles.

III - 3.2. RECHERCHE DE ε POUR UN GRAFCET PARTICULIER

Si un Grafcet a un ensemble d'entrées $E = \{e_1, e_2, e_n\}$, s'assurer que 'un flou' de ε sur les entrées de ce grafcet est admissible. Ce qui revient à vérifier les propriétés suivantes :

1) si $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n)$ et si $e_1, e_2, \ldots, e_n$ ont des périodes minimales $(T_1, T_2, \ldots, T_n)$ alors $\varepsilon_i \leq T_i \quad \forall\ i$

Cette propriété traduit le fait qu'il ne faut pas "oublier" des valeurs de l'entrée e_i dans la machine à comportement approché.

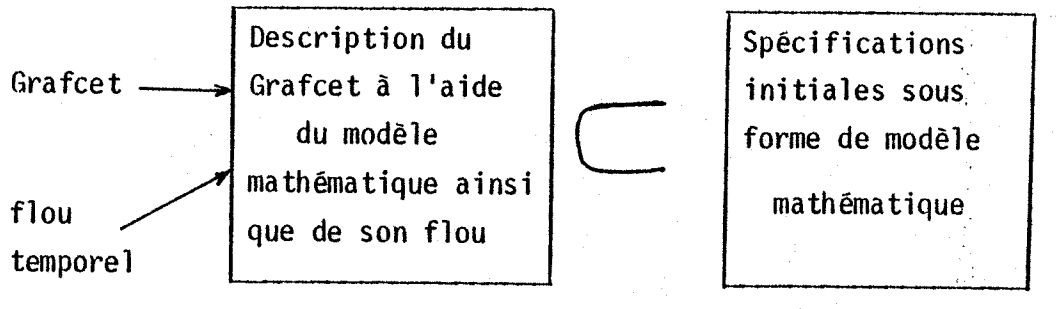
2) Pour tout e_i, pour toute partie de E composée de e_i et des entrées susceptibles d'arriver dans l'intervalle ε_i, l'inversion de l'ordre des entrées n'entraîne pas un comportement inadmissible.

Il est évident que cette deuxième propriété est plus difficile à démontrer, ce qui pose le problème de validation du comportement approché du GRAFCET.

III - 3.2. POSSIBILITE DE VALIDATION DU GRAFCET ET DE SON COMPORTEMENT APPROCHE

Il s'agit maintenant de s'assurer que les spécifications algorithmiques sous forme GRAFCET + flou temporel sont conformes aux spécifications initiales.

Pour des grafcets simples, il semble possible de décrire le GRAFCET à l'aide du modèle mathématique présenté en II.1., puis de s'assurer que la description obtenue est incluse dans la spécification initiale.



Il est évident que l'inclusion est difficile à démontrer dès que le grafcet est un peu complexe. Des techniques d'analyse sont en cours de développement [8].

Dans les autres cas, la seule validation possible est apportée par la simulation.

Cette simulation se fait en deux temps :

- simulation du GRAFCET idéal,
- simulation du GRAFCET à (ε, Δ) près.

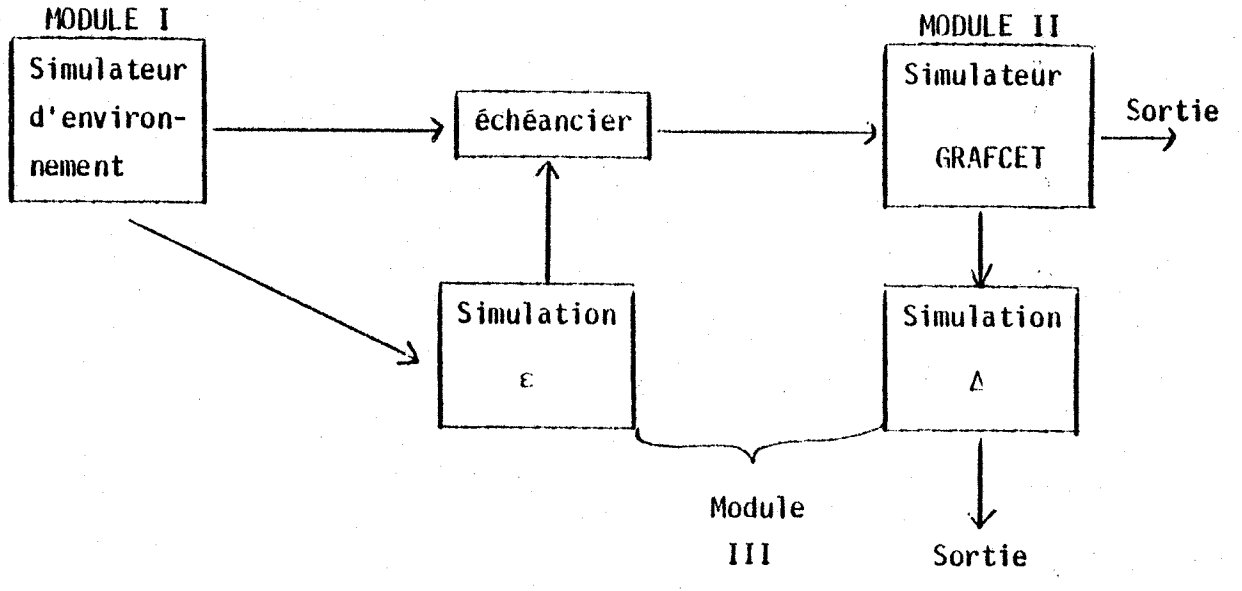
Cette simulation se fait à l'aide d'un simulateur (en cours de développement). Le simulateur comprend trois parties :

. simulation de la machine abstraite GRAFCET (déjà développé)

. simulation de l'environnement physique

. simulation du (ε,Δ)

Le simulateur peut être représenté par le schéma suivant :



Pour simuler le GRAFCET idéal, le concepteur utilise les modules 1, 2. Pour simuler le GRAFCET à (ε,Δ) près, il utilisera de plus le module III.
Les modules 1 et 3 sont en cours de développement. Il est à noter que l'environnement peut lui aussi être décrit à l'aide d'un GRAFCET. De ce fait, le module I appellera éventuellement le module II.

III - 4. CONCLUSION

Le choix d'un modèle abstrait synchrone ou asynchrone dépend du type de système à réaliser, et surtout de l'expérience du programmeur. Pour des systèmes soumis à des fortes contraintes de temps, de coût et pour un programmeur habitué à la programmation parallèle, le modèle asynchrone est plus adéquat. En effet, l'implémentation qui en sera dérivée sera certainement plus performante. Par contre, l'approche synchrone s'applique à une classe de systèmes en pleine extension, dont les caractéristiques sont plutôt l'exigence d'une grande sécurité et le fait d'être réalisé par des programmeurs d'application. Il reste que dans les deux approches, la validation analytique se pose.

CHAPITRE IV

IMPLANTATION FIDELE

Dans le chapitre I une implantation fidèle a été définie comme étant la construction d'une machine qui avertit l'environnement lorsqu'elle ne respecte pas les contraintes temporelles. Ce type d'implantation est particulièrement courante pour de grands systèmes (la grandeur rend difficile la construction de machine temporellement sûre) bénéficiant de la présence d'opérateurs humains (conduite de procédés industriels, etc...). Cette notion d'implantation apparaît d'ailleurs implicitement dans la plupart des langages évolués temps réel.(Instructions exprimant un déroutement en cas de non respect des délais de réponse).

La définition de comportement approché admissible d'une spécification algorithmique permet de définir rigoureusement ce qu'est une implantation fidèle.

"Une implantation fidèle est la réalisation d'une machine ayant soit un comportement approché admissible à ($\varepsilon\Delta$) près, soit avertissant l'environnement.

Une machine fidèle devra donc :

- Saisir les entrées à ε près (ou dans le cas contraire en avertir l'extérieur)
- Exécuter les tâches conformément à la machine abstraite
- Envoyer les sorties avant Δ (ou avertir l'extérieur dans le cas contraire)

Ce travail peut bien entendu être fait par un ou plusieurs processeurs.

IV - 2. IMPLANTATION MONOPROCESSEUR

On implantera sur un seul processeur les 3 processus suivants :

IV - 1.1. SAISIE DES ENTREES

Le processus d'acquisition des entrées (variables, signaux) doit être en mesure de dater ces entrées à ε près ou d'émettre une alarme. Bien entendu

deux solutions sont possibles :

Saisie des entrées par interruption

Saisie des entrées par scrutation

. *Par interruption*

Cette solution consiste à associer à chaque entrée une interruption. Cette solution permet de minimiser dans la plupart des cas l'intervalle de temps séparant l'arrivée d'un événement et sa prise en compte. Malheureusement en présence d'une forte charge (arrivée d'événements très rapprochés) le système s'écroule et il est difficile de connaître l'intervalle de temps séparant l'entrée réelle et la prise en compte. Cette solution entrainera donc des alarmes intempestives.

. *Par scrutation*

- L'approche par scrutation semble mieux adaptée,il suffira d'aller scruter les entrées à une fréquence compatible avec ε. Dans ce cas on est assuré du respect du ε sur les entrées.

IV - 1.2. FONCTIONNEMENT INTERNE

Il s'agit ici de réaliser un comportement conforme à la machine abstraite à partir de l'image des entrées. L'algorithme agira exactement comme un simulateur à partir de l'échéancier.

L'échéancier contient les événements externes avec leur date d'arrivée classés par le processus de gestion des entrées et les événements internes avec leurs dates théoriques engendrés en cours de calcul (instructions d'attente sur delais etc...).

Si le processeur devient libre à l'instant t et si la date associée à l'événement en tête de l'échéancier est antérieure à t, les tâches commandées par cet événement dans la machine abstraite sont exécutées avant toute nouvelle

consultation de l'échéancier.

IV - 1.3. EMISSION DES SORTIES

Pour toute sortie, il est nécessaire de s'assurer que Δ est respecté. Pour cela il suffit de comparer la date de l'événement ayant entrainé la sortie (date précisée dans l'échéancier) et la date réelle de la sortie; il s'agit en fait d'une sorte de chien de garde.

IV - 2. SOLUTIONS MULTIPROCESSEURS

IV - 2.1.

La première solution multiprocesseur qui vient à l'esprit est bien entendu la répartition des 3 processus sur 3 processeurs.

l'un pour la gestion des entrées

le deuxième pour la simulation de la machine abstraite

le troisième pour la surveillance des sorties.

Il est à noter que processeur ne veut pas forcément dire microprocesseur. En particulier il est possible de réaliser la gestion des entrées et la surveillance des sorties à l'aide de circuits spécialisés.

IV - 2.2. SOLUTION A DEGRE DE PARALLELISME SUPERIEUR

Lorsque le nombre d'alarme est trop élevé, il est nécessaire d'envisager des solutions augmentant le parallèlisme. Deux solutions peuvent être envisagées :

IV - 2.2.1.

L'une consiste à diviser le système à réaliser en plusieurs sous Systèmes. Chaque sous système est alors conçu séparément (définition d'une machine

abstraite réalisant le sous système et d'un comportement admissible approché). Chaque sous système voit donc le reste du système comme son environnement et définit donc un flou admissible par rapport aux autres sous systèmes. La définition de ce flou rend la réalisation facile.

Par contre, cela nécessite pour le concepteur un retour en arrière et une nouvelle spécification algorithmique.

IV - 2.2.2. L'autre solution consiste à parallèliser la simulation de la machine abstraite. Cette parallèlisation est plus ou moins facile selon le modèle utilisé (Synchrone-Asynchrone) car de nombreux problèmes de cohérence se posent.

IV - 3. EXEMPLE DE MACHINE PERMETTANT UNE IMPLANTATION FIDÈLE : L'INTERPRETEUR GRAFCET FIDÈLE

Pour permettre une implantation fidèle facile à partir du modèle grafcet (voir Chap. II) nous avons réalisé un interpréteur GRAFCET fidèle. Réaliser cet interpréteur revenant donc à réaliser une machine temps réel paramétrable (par Grafcet). Nous l'avons donc conçu à l'aide de la démarche proposée (spécification initiale - spécification algorithmique - spécification structurelle).

IV - 3.1. SPECIFICATION INITIALE DE L'INTERPRETEUR GRAFCET FIDELE

Nous avons voulu concevoir une machine extrêmement simple. Pour cela nous considérerons que le flou autorisé sur les n entrées sera le même pour toutes les entrées $\varepsilon = (\varepsilon_1, \varepsilon_1, \ldots, \varepsilon_1)$ et le délai de réponse sera le même pour toutes les sorties $\Delta = (\Delta_1, \Delta_1, \ldots, \Delta_1)$.

Pour spécifier proprement l'interpréteur, nous prendrons le formalisme proposé au chapitre 2. Soient $e_1(t)$, $e_2(t)$,...$e_n(t)$ les entrées de l'interpréteur

Soit $E'_1\ E'_2\ E'_n$ les variables représentant la "vision" des entrées par l'interpréteur. On pose $E'_i = (e'_i, \hat{e}'_i)$ E'_i est liée à e_i par la relation suivante quel que soit i :

$$\forall t\ \exists\ t_1\ '\ \in \left[t - \epsilon_i\ t\right] \text{ tel que } e'_i(\mu_{\hat{e}'_i}(t)) = e_i(t_1)$$

De plus $\forall\ i\ J \qquad \hat{e}_i(n) \neq \hat{e}_J(m)$

$\forall n\ m$

Relation qui exprime le fait que 2 événements ne peuvent se produire en même temps.

Nous avons vu (chap. II) comment est définie la machine abstraite Grafcet. La machine abstraite Grafcet est une fonction :

$$S = mag\left[E'\right]$$

où E' est la variable $\left[(e'_1, e'_2, \ldots e'_n), \Sigma\ \hat{e}'_1\right.$

et S la variable $\left[(s_1, s_2, \ldots s_m), \sum_{i=1}^{m} \hat{s}_i\right.$

Il s'agit maintenant de définir les relations liant les sorties réelles des sorties S de la machine abstraite :

$$\forall j \in (1..m)\ \exists\ \left[s'(j), a(j)\right] \text{ tel que}$$

$$s'(j) = s(j) \qquad (1)$$

$$a(j) = \left[\hat{s}'(j) - \hat{s}(j) > \Delta\right] \qquad (2)$$

(1) permet d'exprimer que pour toute sortie de la machine abstraite il y a

une sortie réelle

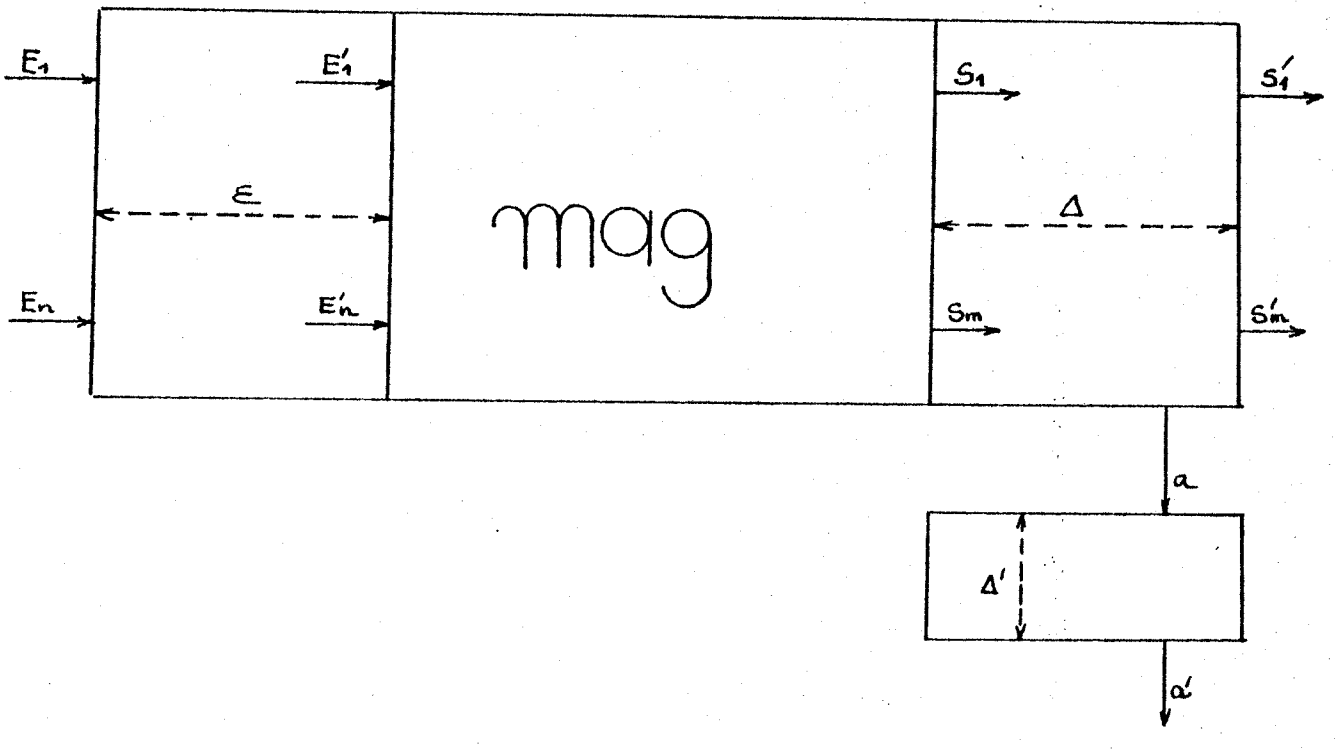
(2) permet d'exprimer la valeur de la fonction "alarme"

il faut maintenant définir quand il y aura une alarme réelle (notée a')

$$\forall m \; \exists \; a'(m) \text{ tel que } a(m) = a'(m)$$

$$\hat{a}(m) + \Delta' > \hat{a}'(m) > \hat{a}(m)$$

L'interpréteur Grafcet fidèle peut se résumer à l'aide du schéma suivant :



L'alarme aura lieu au plus tard Δ' après la situation de dépassement de Δ.

Cette spécification de l'interpréteur est complète, non ambigüe, paramétrée par $(\varepsilon,\Delta,\Delta')$ ε représentant la dérive temporelle autorisée par les entrées, Δ la dérive temporelle des sorties, Δ' le retard autorisé entre la situation d'alarme et l'alarme. Nous avons décidé de fixer ε et Δ' par construction mais de laisser la possibilité de l'utilisateur de fixer lui-même Δ (l'utilisateur fournira donc à l'interpréteur le grafcet et Δ) nous avons fixé $\Sigma\varepsilon$ à 1 ms et Δ' à 1 ms

IV - 3.2. SPECIFICATIONS ALGORITHMIQUES DE L'INTERPRETEUR

Il s'agit de donner ici une solution algorithmique de l'interpréteur. Nous avons vu qu'il fallait distinguer 3 grandes tâches :

- saisie des entrées (Tâche 1)
- machine abstraite GRAFCET (Tâche 2)
- chien de garde (Tâche 3)

Les tâches 1 et 3 sont relativement simples et il n'y a pas besoin de décomposition supplémentaire sur ces 2 tâches. La tâche 2 est composée de plusieurs fonctions :

- saisie de l'événement suivant
- calcul de l'évolution comprenant la recherche des transitions valides et le calcul des réceptivités et le franchissement
- calcul des tâches associées aux places

Ces tâches se partagent un certain nombre de données. Ce partage peut se décrire à l'aide du graphe de donnée suivant :

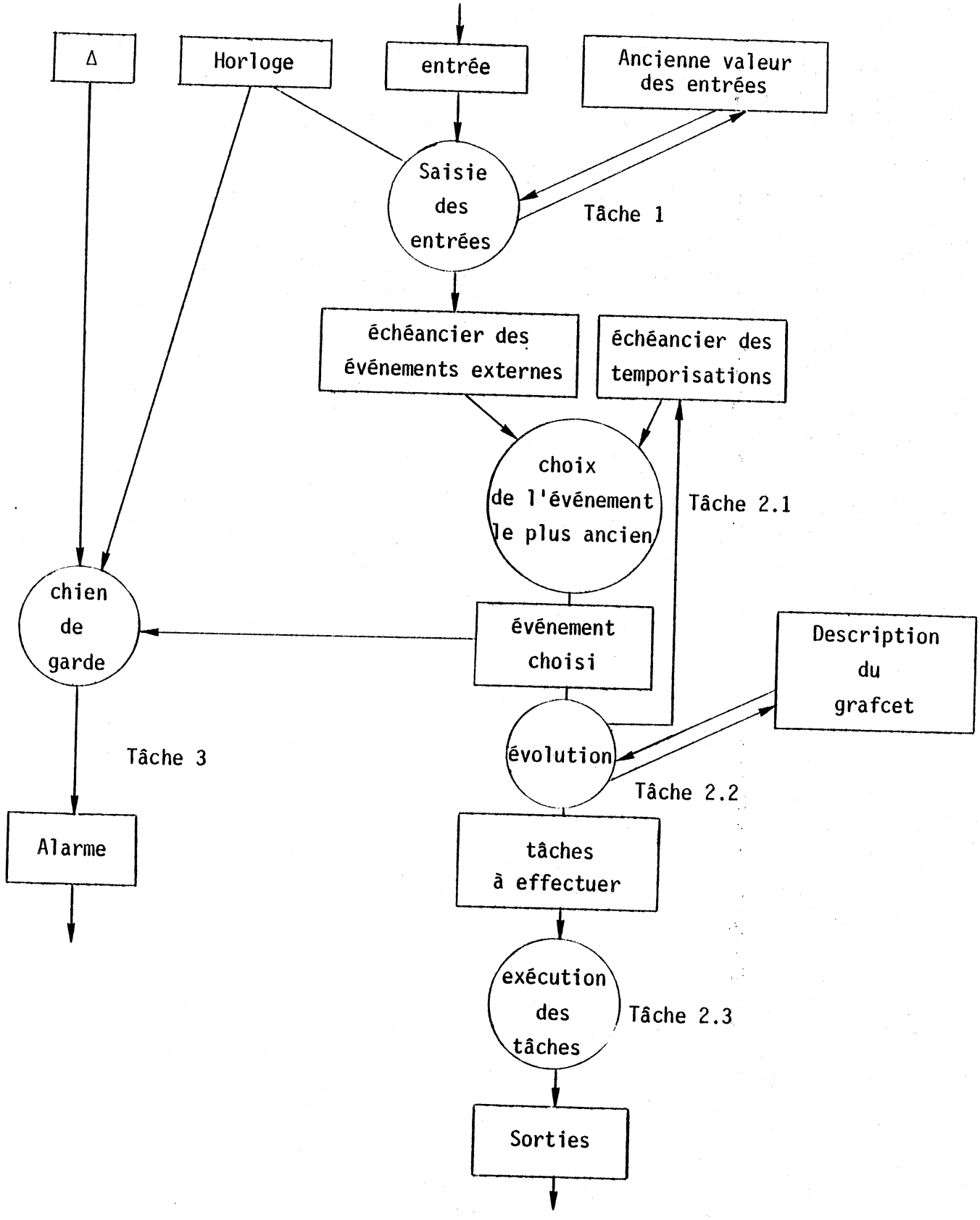

Δ
Horloge
entrée
Ancienne valeur des entrées
Saisie des entrées
Tâche 1
échéancier des événements externes
échéancier des temporisations
choix de l'événement le plus ancien
Tâche 2.1
chien de garde
événement choisi
Description du grafcet
évolution
Tâche 2.2
Tâche 3
Alarme
tâches à effectuer
exécution des tâches
Tâche 2.3
Sorties

Fig. 1

Remarques :

La tâche 1 (Saisie des entrées), a besoin :

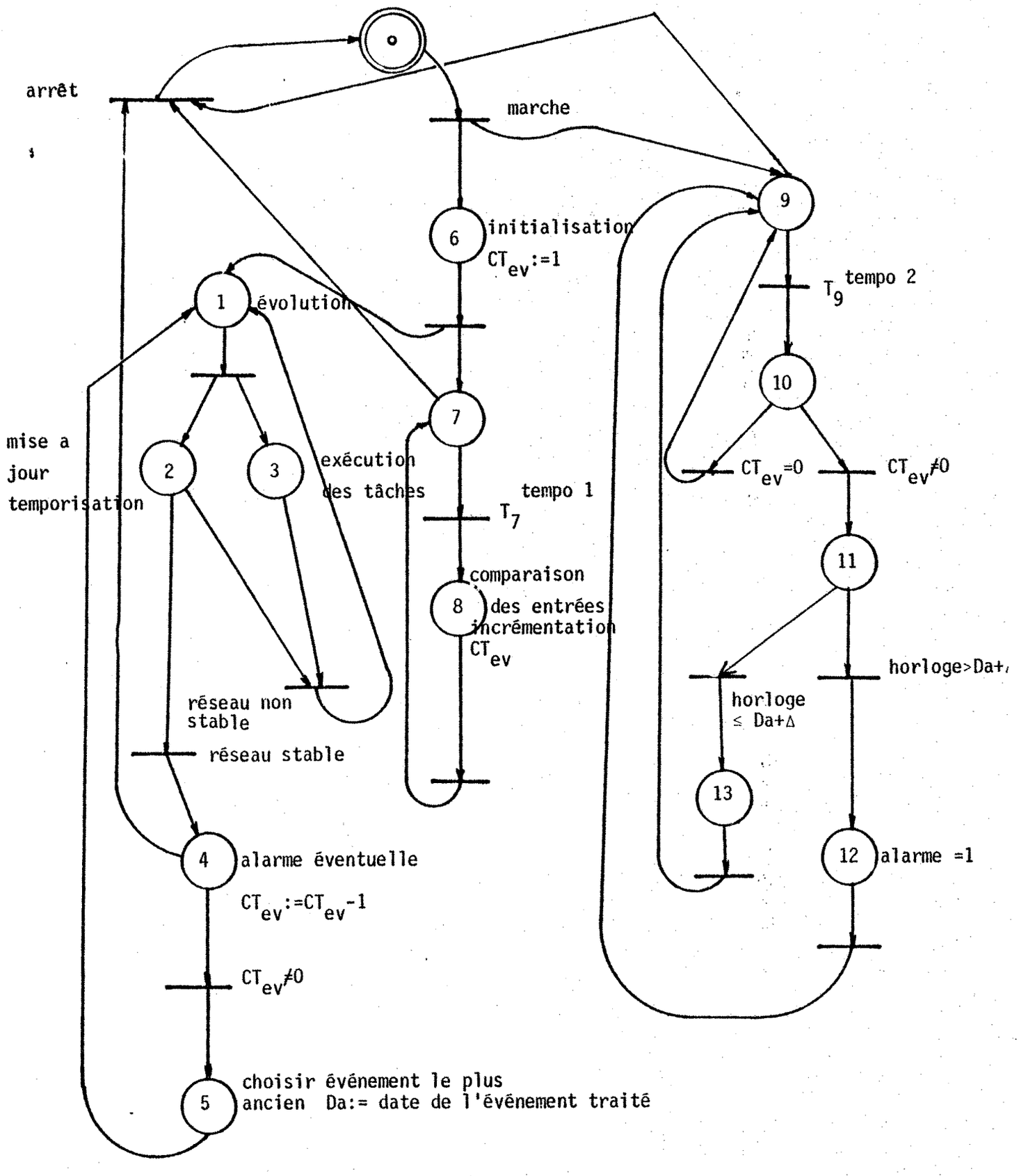
d'une horloge pour dater les entrées

d'une "photographie" des anciennes valeurs des entrées permettant ainsi de savoir quelles sont les entrées qui ont changé.

La tâche 2.1 doit choisir l'événement le plus ancien entre les événements "changement d'entrée externe" et les événements "fin de temporisation" en comparant leurs dates respectives.

La tâche 3 a besoin de l'horloge pour mesurer l'écart entre le temps exact et le temps de l'événement choisi. Si cet écart est supérieur à Δ il y aura alarme.

La synchronisation de ces taches peut être décrite à l'aide du GRAFCET suivant :



* commentaires pages suivantes

Figure 2

La synchronisation se fait à l'aide de CT_{ev} - CT_{ev} représentant le nombre d'événements + le nombre de temporisations. CT_{ev} est mis à jour par l'étape 8 (détection des entrées nouvelles) et par l'étape 2 (détection des temporisations). CT_{ev} est decrémenté à l'étape 4 (lorsque le réseau est stable un événement ou une temporisation peut être décompté car il vient d'être traité.

Tempo 1 et tempo 2 sont 2 temporisations. Pour que la machine abstraite respecte les spécifications, il faut que tempo 1 $\leq \varepsilon$ et tempo 2 $\leq \Delta'$.

On remarquera que l'alarme est donnée $\begin{bmatrix}\text{étape 4}\\ \text{étape 12}\end{bmatrix}$ lorsque l'événement le plus ancien (qui est forcément l'événement en cours de traitement) a une date inférieure à la date courante plus Δ. Ceci est une surspécification par rapport à la spécification comportementale car on enverra des alarmes lorsqu'un événement ne sera pas traité avant Δ, même si cet événement ne donne pas lieu à des sorties. La détection de la situation d'alarme se fait soit à la fin du traitement de l'événement (étape 4) soit en cours de traitement (étape 12). La détection en cours de traitement est obligatoire si on veut respecter la règle de l'émission d'alarme avant Δ' (car il n'est pas assuré de finir le traitement d'un événement avant l'état d'alarme + Δ').

- Description complète des tâches associées aux étapes

Etape 1

A cette étape est associé l'algorithme d'évolution décrivant un pas. Cet algorithme a pour paramètre un événement (soit e événement toujours présent soit x événement externe) et a pour but de déterminer l'état suivant du GRAFCET.

Soit X_i l'état courant d'un grafcet.

Cet algorithme se décompose en :

Déterminer les transitions validées par X_i

Déterminer les transitions franchissables parmi les transitions validées

Déterminer X_{i+1}, état suivant du grafcet

Déterminer la liste L_F des tâches à exécuter. Ces tâches sont celles associées à une étape appartenant à X_{i+1} et n'appartenant pas à X_i.

Etape 2

A cette étape est associée une tâche qui crée ou modifie l'échéancier propre auxtemporisations.Cet échéancier contient l'ensemble des temporisations avec leur date. De plus la tâche incremente le compteur CT_{ev} du nombre de temporisations faites.

Etape 3

A cette étape, est associée la tâche chargée d'exécuter les tâches de la liste L_F.

Etape 4

La tâche associée à l'étape 4 est décrite sur le graphe. Cette tâche a pour paramètre l'événement qui a déclenché l'évolution du graphe. Elle compare la date de cet événement avec la date courante - Δ et émet l'alarme éventuellement. De plus comme cette étape marque la fin du traitement de l'événement, on décremente le compteur d'événement.

Etape 5

La tâche associée choisit parmi l'ensemble des événements externes et l'ensemble des événements temporisations, l'événement le plus ancien. Elle fournit un et un seul événement ce qui permet de respecter la spécification : "2 événements ne sont pas traités en même temps par la machine GRAFCET".

Etape 6

C'est l'étape d'initialisation. La tâche associée comprend l'initialisation du compteur d'événement à 1 et la 1ère lecture des entrées, cette lecture permet de charger le tableau T_A tableau des "anciennes entrées" utile à l'étape 8

Etape 7 n'a pas de tâche associée

Etape 8

C'est l'étape de détection du changement d'entrée la tâche peut se décrire par :

Lire les entrées e(1), e(2) e(n) et les ranger dans le tableau T_N

$T_N(i) := e_i$

Pour tout $i \in \{1,\ldots,n\}$ faire

si $T_N(i) \neq T_A(i)$ alors

- ranger[i, T (i) date courante] dans l'échéancier
- incrementer CT_{ev}
- $T_A(i) := T_N(i)$

Les étapes 9-10-11-12-13 ont soit des tâches associées vides soit des tâches très simples décrites au niveau du graphe.

Le GRAFCET ainsi défini est bien une machine abstraite réalisant l'interpréteur GRAFCET fidèle. Le Grafcet de la figure 2 a la propriété de ne pas être synchronisé sur des événements d'entrées (hormis marche-arrêt). Comme pour toute spécification algorithmique il nous faut définir un comportement admissible. La définition du comportement approché admissible de la machine (ε_{Graf}, Δ_{Graf}) est relativement facile :

$\varepsilon_{Graf} \leq \frac{1}{1000}$ sec. ce temps permet de ne pas manquer des signaux marche-arrêt

Δ_{Graf} doit être tel que

(1) tempo 2 + $\Delta_{Graf} \leq \Delta'$ pour assurer l'émission de l'alarme avant Δ'.

Une machine ayant un comportement approché du Grafcet de la figure 2 à ($\frac{1}{1000}$, Δ_{Graf}) près (Δ_{Graf} étant lié par la relation 1) est une machine qui respecte les spécifications comportementales.

IV - 4. SPÉCIFICATIONS STRUCTURELLES DE L'INTERPRÉTEUR GRAFCET

Il s'agit ici de décrire l'implantation de l'interpréteur GRAFCET sur un matériel donné. Il s'agit d'une implantation sûre (on doit respecter absolument les délais de réponse). Ce problème sera donc traité à titre d'exemple au chapitre V.

CHAPITRE V

IMPLANTATION TEMPORELLEMENT SURE

L'implantation temporellement sûre est la réalisation d'un système (choix du matériel, implantation du programme sur ce matériel) de façon à ce qu'il respecte absolument les contraintes temps réel. Dans notre démarche de conception, c'est donc réaliser une machine ayant un comportement approché à (ε,Δ) près.

La saisie des entrées à ε près peut se faire de la même façon que pour l'implantation fidèle (voir chap. IV).

Le problème ici est de choisir le matériel et de réaliser l'ordonnancement des tâches sur ce matériel (Mémoire, processeurs) de telle façon que le comportement de la machine respecte Δ. Ce problème est en fait, pour chaque système à réaliser, un problème d'ordonnancement particulier. Etant données des tâches ayant des contraintes d'ordonnancement entre elles, il s'agit de trouver un ordonnancement (ou un algorithme d'ordonnancement) respectant les délais de réponse sur les tâches exécutant des sorties (délais exprimés à l'aide de Δ). Les problèmes d'ordonnancement sous cette forme ont été beaucoup étudiés et nous nous proposons de faire une rapide revue des différents travaux.

V - 1. LES PROBLEMES D'ORDONNANCEMENT

Le problème d'ordonnancement se pose dès que l'on veut allouer des ressources à des tâches de telle sorte que ces tâches soient exécutées en respectant certaines contraintes données.

Il est bien évident que ce problème ne se rencontre pas seulement en informatique. Deux exemples courants sont par exemple :

- la création d'un emploi du temps dans un lycée. Un certain nombre de tâches (heures de cours) doivent être effectuées avec un certain nombre de ressources (nombre de salles de classes, nombre de professeurs). Un certain nombre de contraintes impératives existent (un professeur de français n'enseigne

pas les mathématiques, on ne fait pas un cours de gym dans une salle de chimie) et un certain nombre de critères permettent de dire si l'emploi du temps est bon ou mauvais (attente des élèves et des professeurs entre deux cours, journée libre pour les élèves et professeurs... etc). Ce problème est typiquement un problème d'ordonnancement multicontraintes multicritères.

- l'organisation d'un tournoi de tennis (ou de tennis de table !) est un autre exemple de problème d'ordonnancement.Le critère peut être par exemple que le tournoi se termine le plut tôt possible.

On voit donc que le sujet est très varié et complexe. Pour simplifier le problème la plupart des études considère un seul type de ressource.
Ce problème en terme informatique peut être défini de la façon suivante [19] :

La ressource est appelée processeur. Il peut exister plusieurs processeurs de performances différentes formant l'ensemble des ressources.
L'ensemble des processeurs est habituellement noté $(P_1, P_2, \ , P_n)$.

Pour les tâches, il nous faut considérer le triplet (J, α, μ)

$J = \{J_1, J_2 \ , J_m\}$ est l'ensemble des tâches

- α est une relation partielle d'ordre entre les tâches si $J_r \ \alpha \ J_s$ l'exécution de J_s ne peut commencer avant la fin de J_r.

- μ est une relation de J dans $\mathbb{N}^n$ ou $\mathbb{R}^n$ selon les auteurs

$$\mu(J_i) = (t_{1i}, t_{2i}, \ , t_{ni})$$

t_{ki} indique le temps que met le processeur k à exécuter la tâche J_i.

Cette durée de tâche selon les auteurs est soit un entier, soit un réel ; Nous considérons dans la suite que $\mu(J_i)$ est un vecteur d'entiers.(En effet, dans un système informatique, la base de temps naturelle est fournie par le nombre de cycles machine).

Si l'ensemble des processeurs est composé de processeurs identiques on notera

$$\mu(J_i) = C_i \quad \text{qui signifie : } \mu(J_i) = (C_{1i}, C_{2i}, \quad , C_{ni})$$

*Ordonnancer un ensemble de tâches sur un système à multiprocesseurs revient donc à préciser pour chaque tâche J_i l'intervalle de temps durant lequel elle sera exécutée ainsi que le processeur d'exécution.

* Un algorithme d'ordonnacement est une procédure permettant d'ordonnancement un ensemble de tâches.

Pour mesurer les performances des algorithmes on utilise généralement un des deux critères suivants :

- le critère de date d'arrêt

. Si on suppose que d'ordonnancement part à la date 0 et que $f_i(S)$ indique la date à laquelle le job J_i est fini pour l'ordonnancement D le critère de date d'arrêt est $\omega(S) = \max_{1 \leq i \leq m} f_i(S)$

- le critère de date moyenne d'arrêt

$$\tilde{\omega}(S) = \frac{1}{m} \sum_{i=1}^{m} f_i(S)$$

Le problème d'ordonnacement d'un ensemble de tâches (J,α,μ) sur un ensemble de processeurs en vue de minimiser la date d'arrêt a été beaucoup étudié.

Ce problème est considéré comme un "NP hard" problème, c'est-à-dire comme un problème

n'ayant pas d'algorithme de résolution à facteur polynomial.

Plusieurs auteurs ont proposé des algorithmes polynomiaux pour des sous classes du problème d'ordonnancement [voir 23-19 pour un panorama de ces problèmes].

Le problème d'ordonnancement peut être posé aussi dans un autre sens qu'on pourrait appeler "recherche d'ordonnancements respectant les délais de réponse":

A chaque job J_i est associée une date d'autorisation de départ, date à partir de laquelle J_i peut commencer à être exécutée et une date d_i appelée date de fin d'exécution de la tâche J_i date qu'il ne faut pas dépasser.

Le problème est de trouver un ordonnancement respectant les contraintes sur un nombre minimal de processeurs (tous identiques dans ce cas).
Ce problème dans le cas où les tâches sont indépendantes est connu comme un problème NP complet.

Plusieurs algorithmes sous optimaux ont été référencés dans [23]. Un sous problème fort intéressant pour nous est le problème d'ordonnancement de tâches à requête périodique. Les chercheurs ont pour ce problème proposé des algorithmes d'ordonnancement préemptifs [19] permettant la conception de moniteur d'ordonnancement [19] [20] [21].

V - 2. APPLICATION DES ÉTUDES DES PROBLEMES D'ORDONNANCEMENT DANS L'IMPLANTATION SURE DES SYSTEMES TEMPS REEL

Pour étudier les problèmes d'ordonnancement les chercheurs ont fait un certain nombre d'hypothèses leur permettant de résoudre un problème particulier. La difficulté pour le concepteur de système temps réel sera donc d'adapter les théories à son cas particulier. Nous allons montrer les différentes approches possibles en fonction des différents cas.

V - 2.1. LES TACHES SONT A REQUETES STRICTEMENT PERIODIQUES

C'est en particulier souvent le cas pour des systèmes tels que les systèmes de contrôle ou des systèmes embarqués simples.

La première difficulté consistera à déterminer les temps maximums des tâches sur le matériel choisi. Ceci se fait à partir du programme associé à chaque tâche.

A partir de la connaissance de ces valeurs, 2 démarches sont possibles.

- Déterminer un ordonnancement statique des tâches, ordonnancement portant sur le plus petit commun multiple des périodes. Cet ordonnancement se fait soit à la main (dans les cas simples, et en particulier lorsque les périodes des tâches sont multiples entre elles) soit à l'aide d'un algorithme d'ordonnancement du type [20-21-19].

- Utiliser un algorithme d'ordonnancement des tâches pour faire un moniteur de tâche. Ces algorithmes peuvent être ceux présentés en [20] et [21] si les tâches sont interruptibles ou bien ceux présentés en V.2.1.1.2. (L'utilisation d'un algorithme d'ordonnancement ne se limite pas aux tâches à requête périodique et sera étudié plus en détail dans V.2.2.)

Quel que soit l'ordonnancement choisi, il s'agira aussi d'évaluer la place mémoire nécessaire pour l'implantation et, dans un système distribué, d'assurer la gestion partagée des mémoires. Peu d'études théoriques ont été faites sur ce sujet [24,25] . Chaque système semble devoir être traité comme un cas particulier.

V - 2.1.1. Exemple de l'interpréteur GRAFCET fidèle

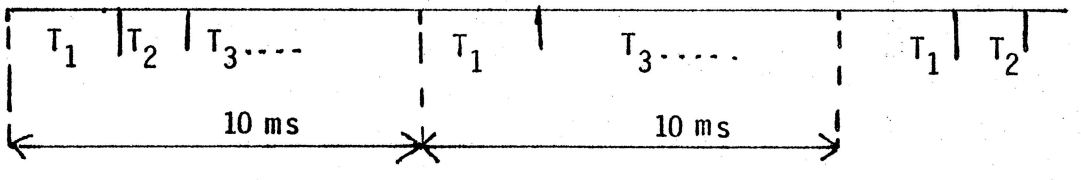
L'interpréteur Grafcet fidèle peut être vu comme un système à tâche périodique (voir chap. IV)

- 1 tâche T_1 (lecture d'entrée) de période Tempo 1 correspondant à l'étape 8 du grafcet
- 1 tâche T_2 (détection d'alarme) de période Tempo 2 correspondant au sous grafcet (étape 9, 10, 11, 12, 13)
- 1 tâche T_3 d'évolution du grafcet (étape, 1, 2, 3, 4, 5) dont on ne connaît pas la période mais qui doit se faire dès que possible.

Lors de l'implantation, nous avons pris pour faciliter l'ordonnancement

tempo 2 = 2 tempo 1 = 20 ms

- l'évaluation du temps de la tâche T_1 pour une implantation sur M6800 est de l'ordre de 1 ms pour 32 entrées de 1 à 8 bits. l'évaluation du temps de T_2 est de 50 µs.
- l'ordonnancement proposé est le suivant :



Un tel ordonnancement permet la réalisation d'un interpréteur Grafcet fidèle à (10 ms, Δ) près. L'alarme de dépassement de Δ étant donnée au maximum 20 ms après le dépassement.

- Evaluation de la place mémoire

  La place mémoire nécessaire pour les tâches T_1, T_2 est négligeable.

  Pour la tâche T_3, la place mémoire nécessaire dépend directement du Grafcet à implanter.

- la tâche T_3 et la tâche T_1 ont en commun l'échéancier, il nous faut évaluer la place mémoire de l'échéancier. Pour chaque événement d'entrée il faut 4 octets. Il y a au maximum 32 événements d'entrées en un temps ε. Comme un événement d'entrée est perimé après être resté Δ dans l'échéancier, il suffit de s'assurer de la conservation des renseignements pendant Δ.

La place mémoire nécessaire est donc de $\frac{\lceil \Delta \rceil}{\varepsilon} \times 128$ octets où $\frac{\lceil \Delta \rceil}{\varepsilon}$ est le premier entier supérieur ou égal au rationel $\frac{\Delta}{\varepsilon}$.

Remarque :

Dans cet exemple, l'ordonnancement était très simple. Dans la pratique, l'adaptation des fréquences des tâches de façon à ce qu'elles soient multiples entre elles est courante et simplifie souvent le problème.

V - 2.2. LES TACHES SONT A REQUETES NON STRICTEMENT PERIODIQUES

C'est bien entendu le cas général. La seule façon d'ordonnancer les tâches est de construire un moniteur d'ordonnancement de tâches s'appuyant sur une stratégie d'ordonnancement. La difficulté réside dans le fait qu'il faut valider les choix d'ordonnacement faits par ce moniteur.

V - 2.2.1. Etude de cas simples

- Le cas le plus simple est l'ordonnancement de tâches indépendantes (pas de contraintes d'ordonnancement entre elles) ayant les propriétés suivantes :

  - il existe un délai minimal T_i entre deux requêtes d'une même tâche J_i.
  - il est possible de définir un temps d'exécution maximal C_i de la tâche J_i.

Il existe deux types de moniteurs d'ordonnancement de tâches ayant ces propriétés.

V - 2.2.1.1. Les moniteurs basés sur un algorithme d'ordonnancement des tâches interruptibles. Ce sont les moniteurs susceptibles d'interrompre les tâches pour en lancer d'autres plus urgentes. L'étude de la validation des moniteurs de ce type sur monoprocesseur basés sur des stratégies différentes (à priorité fixe, à priorité liée aux dates critiques) a été faite en particulier dans [19] [20] [21].

V - 2.2.1.2. Les moniteurs basés sur un algorithme d'ordonnancement non interruptible. Peu d'études ont été faites sur ce sujet à notre connaissance. Il nous semble important d'étudier pourtant ce cas car les moniteurs basés sur un algorithme d'ordonnancement non interruptibles peuvent être très simples (donc peu coûteux au point de vue temps) et bien adaptés à une certaine classe d'application (fréquence de requête élevée, tâche simple) où les mécanismes d'interruptions peuvent être très pénalisants.

Nous avons donc étudié la validation "temps réel" de 3 stratégies d'ordonnancement appliqués à des systèmes répondant aux hypothèses suivantes :

H_1) l'application comprend m tâches indépendantes τ_1, τ_2, τ_m

H_2) le processeur choisi les exécute en un temps maximal C_1, C_2, C_m

H_3) les requêtes de ces tâches ont une période maximale T_1, T_2, T_m

H_4) chaque tâche doit être finie avant que ne soit écoulé un temps d_1, d_2, d_n après l'arrivée de la requête

Ces hypothèses sont toujours faites dans l'étude des validations des algorithmes préemptifs. Nous faisons aussi une hypothèse supplémentaire.

H_5) chaque tâche doit être exécutée avant l'arrivée d'une autre requête.

$d_i \leq T_i \quad \forall\, i \in [1...m]$

Les trois types d'algorithmes retenus sont :

Algorithme 1

Le choix de la requête est fait d'une façon aléatoire parmi les requêtes non traitées.

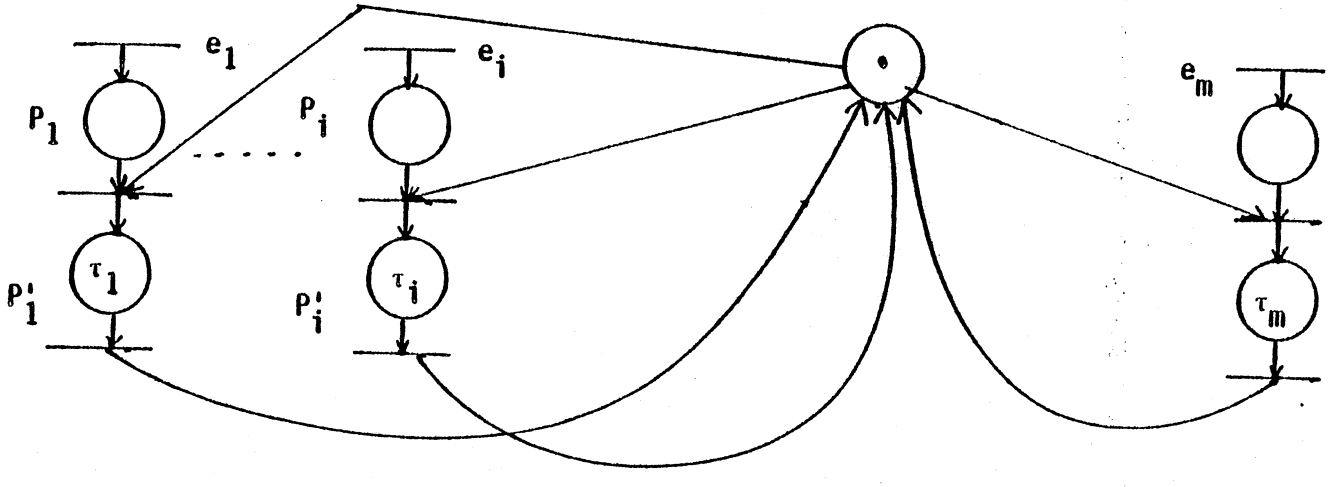
Algorithme 2 : les requêtes sont traitées dans leur ordre d'arrivée

Algorithme 3 : il existe une priorité préétablie parmi les tâches l'algorithme choisit parmi l'ensemble des requêtes non traitées, la requête de plus haute priorité.

Remarque : Le temps d'exécution de l'algorithme est négligeable par rapport à C_i.

A) Etude de la validation avec l'algorithme 1

L'algorithme 1 pourrait être décrit par le Réseau de Petri suivant :



où e_i est l'événement "requête de la tâche T_i".

Remarque : Dans le cas particulier courant où $d_i = T_i \ \forall_i$ les conditions de respect des contraintes temporelles peuvent s'écrire

$N(P_i) + N(P'_i) \leq 1$ avec $N(P_i)$ nombre de marques de P_i.

A.1) Une condition nécessaire et suffisante du respect des délais de réponse pour l'algorithme 1 est que :

$$\sum_{i=1}^{n} C_i \leq \min_{i=1}^{n} d_i \qquad (1)$$

Sous l'hypothèse 5 $d_i \leq T_i \quad \forall\, i$

Démonstration

. Condition nécessaire

Si il existe j tel que $\sum_{i=1}^{n} C_i > d_J$ alors lorsque les n requêtes arrivent à l'instant t, la requête J peut être traitée en dernier et la contrainte d_J ne sera pas respectée.

. Condition suffisante

Soit W(t) la fonction définie comme le temps de travail qu'il reste à effectuer à l'instant t pour satisfaire les requêtes arrivées strictement avant t.

Une condition suffisante du respect des délais de réponse est que :

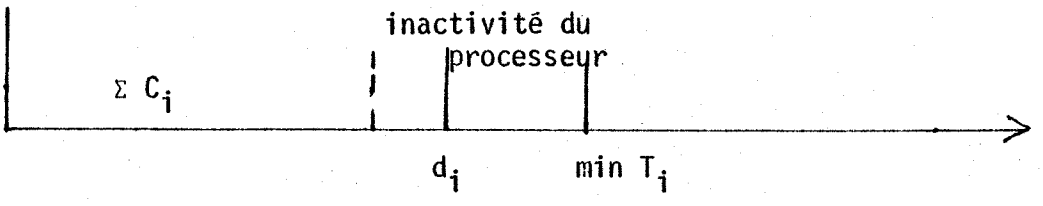
$$\forall t \; \exists\, t' \in \left]t,\; t + \min_{i=1}^{n}(d_i)\right] \text{ tel que } W_{t'} = 0 \qquad (2)$$

en effet si (2) est vérifiée une requête à l'instant t est sûre d'être exécutée avant t'.

La relation (1) implique (2). Ceci se démontre par recurrence.

. Supposons (1) vraie (2) est vérifiée pour $t \in [0d]$ où $d = \min_{i=1}^{n} d_i$

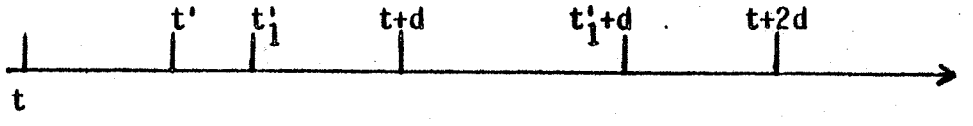
Dans cet intervalle la situation est :



On voit donc que $\forall\ t \in [0\ d]\ \exists\ t' = \min T_i$ tel que $W_{t'} = 0$.

Si (2) est vraie pour t, (2) est vraie pour t + d.

Soit t_1' le plus grand t' tel que $W(t') = 0$ et $t' < t + d$.



alors il existe $t'' \in [t_1', t_1' + d]$ tel que $W(t'') = 0$

comme t_1' est le plus grand t' tel que $W(t') = 0$ et $t' < t + d$

$t'' \in [t + d \quad t + 2\ d]$.

L'existence de t" se démontre par l'absurde. Si t" n'existe pas alors il n'y a pas eu de période d'inactivité entre t_1' et $t_1' + d$

$$W(t_1' + d) \leq W(t_1') + \sum_{i=1}^{n} C_i - d$$

$$W(t_1' + d) \leq \sum_{i=1}^{n} C_i - d$$

d'où $W(t_1' + d) \leq 0$. d'où t" existe = $t_1' + d$.

A.2) La condition (1) est une condition suffisante pour tout algorithme ne laissant pas oisif le processeur lorsqu'il existe une tâche à exécuter. En effet si l'algorithme 1 est validé, tout ordonnancement possible est validé.

B) Etude de l'algorithme 2 (premier arrivé, premier servi)

Dans cet algorithme, les requêtes sont traitées dans leur ordre d'arrivée. En cas d'arrivée simultanée, les requêtes sont soit classées dans un ordre d'arrivée aléatoire, soit dans un ordre pré-défini à l'avance.

B.1) La condition (1) est une condition nécessaire et suffisante du respect des contraintes temporelles dans le cas de l'utilisation de l'algorithme 2.

Démonstration

. Condition nécessaire

Soit J tel que $d_J = d$ si $\sum_{i=1}^{n} C_i > d$ posons $\sum_{i=1}^{n} C_i - d = 2\varepsilon$ alors si la requête de τ_J arrive à l'instant t et que toutes les autres arrivent à l'instant $t - \varepsilon$, la contrainte temporelle d_J n'est pas respectée.

. Condition suffisante

à cause de la remarque A.2 .

B.2) Sans l'hypothèse (5) une condition nécessaire et suffisante du respect des délais de réponse est

$$\begin{cases} \sum_{i=1}^{n} C_i \leqslant \min d_i & (1) \\ \sum_{i=1}^{n} \frac{C_i}{T_i} \leqslant 1 & (3) \end{cases}$$

pour l'algorithme 2.

. Condition nécessaire

(1) est nécessaire lorsque toutes les requêtes arrivent au même instant.

(3) est nécessaire car sans elle la fonction charge de travail $\to +\infty$ lorsque $t \to +\infty$ ce qui entraîne un non respect des contraintes.

. Condition suffisante

Il suffit de montrer que $\forall_J, \forall_n\ W(n, T_J) \leq d_J$.

Ceci se montre en deux temps.

1) les délais de réponse sont respectés jusqu'à la première période d'inactivité du processeur.

Dans ce cas on a :

$$W(n\ T_J) = \Sigma\left(\frac{n\ T_i}{\lfloor T_J \rfloor} + 1\right) C_i - \overset{n}{A}\ T_J$$

temps de travail requis — temps de travail fait

où $\frac{t}{\lfloor T_i \rfloor}$ est le plus petit entier inférieur à $\frac{t}{T_i}$

d'où :

$$W(n\ T_J) \leq \sum_{i=1}^{n} C_i + n\ T_J\left(\sum_{i=1}^{n} \frac{C_i}{T_i} - 1\right)$$

d'où :

$$W(n\ T_J) \leq \sum_{i=1}^{n} C_i \leq d_i$$

2) Soit $\left[t_1\ t_2\right]$ une période d'inactivité du processeur. Soit t_3 le début de la période d'inactivité suivante.

Pour tout i, pour tout n tel que $t_2 \leq nT_i \leq t_3$

$$W(n\ T_J) = W(t_2) + \sum_{i=1}^{n}\left(\frac{n\ T_J - t_2}{\lfloor\ T_i\ \rfloor} + 1\right) C_i + (t_2 - n\ T_J)$$

d'où $\quad W(n\ T_J) \leq \sum_{i=1}^{n} C_i + (n\ T_J - t_2)\left(\sum_{i=1}^{n} \frac{C_i}{T_i} - 1\right)$

d'où $\quad W(n\ T_J) \leq \sum_{i=1}^{n} C_i \leq d_i.$

C) Algorithme à priorité fixe

Il s'agit ici d'un algorithme où les tâches sont choisies selon un ordre de priorité fixé à l'avance. Pour cette étude on supposera

$$d_1 \leq d_2 \leq \ldots \leq d_n \leq \min T_i \qquad \text{(Hyp. 6)}$$

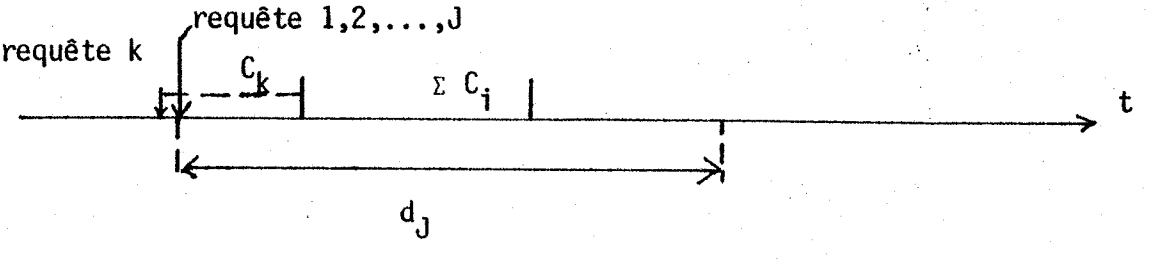
On montre facilement que la meilleure priorité à définir est celle qui est dans l'ordre des délais de réponse. La requête de la tâche τ_1 a la plus haute priorité, suivi de la requête de τ_2 ... jusqu'à la requête de τ_n. La démonstration est analogue à celle de [21] pour l'algorithme préemptif.

Pour cet algorithme et avec l'hypothèse 6 une condition nécessaire et suffisante de respect des délais de réponse est que :

$$\forall_J \quad \sum_{i=1}^{n} C_i + \max_{k=J+1}^{n} C_k \leq d_J$$

. Condition nécessaire

La condition correspond à l'arrivée de la requête de τ_J en même temps que celle de $\tau_1\ \tau_2 \ldots \tau_{J-1}$ et juste après celle d'une requête de priorité inférieure.



. Condition suffisante

Pour une tâche J, la démonstration de la condition suffisante est la même que pour l'algorithme 1 ramené au cas de J+1 tâches indépendantes, les tâches 1...J et la tâche k. En effet pour la tâche J, seule une tâche de

priorité inférieure intervient.

D) Remarque sur l'étude des algorithmes

On a pu voir que les algorithmes 1 et 2 ont des conditions de validation semblables sous l'hypothèse H_5. Ceci n'est pas vrai en l'absence de cette hypothèse (l'algorithme 1 a des conditions nécessaires et suffisantes plus contraignantes).

V- 2.2.2. *Validation des contraintes temporelles dans des cas plus complexes*

Il est évident que la plupart des systèmes ne répondent pas aux hypothèses des études théoriques. En particulier, l'hypothèse de tâches indépendantes est rarement vérifiée.

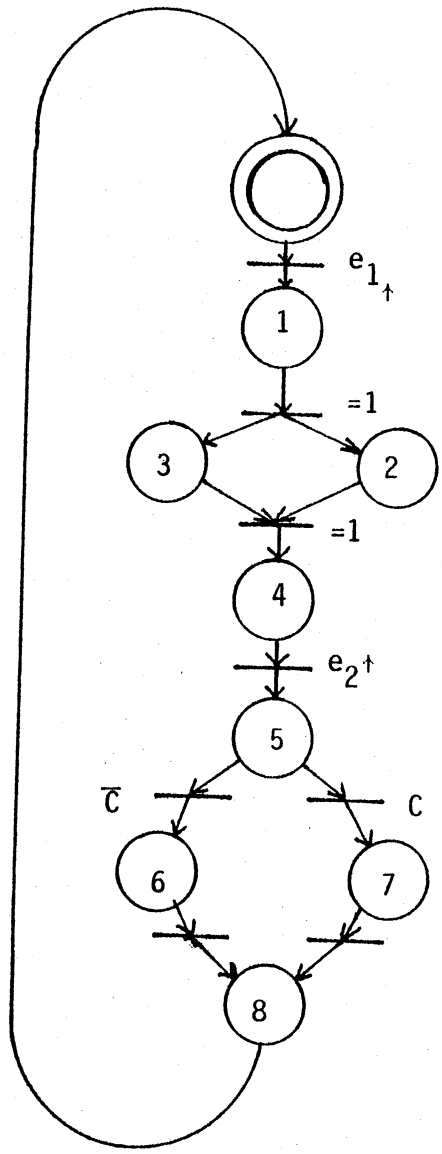
Un moniteur d'ordonnancement réel devra donc tenir compte de la dépendance entre ces tâches.

Pour valider de tels moniteurs sur une application, il faudra tenter de se ramener à un cas simple d'ordonnancement, cas simple devant être plus contraignant que la réalité. Bien entendu, il n'y a pas de méthode pour agir de la sorte. Nous nous proposons de développer une méthode de validation d'une application décrite en Grafcet en utilisant l'interpréteur Grafcet en tant que moniteur d'ordonnancement.

La démarche adoptée est la suivante :

1) Pour un grafcet particulier, nous associons à chaque entrée le temps maximal nécessaire à l'interpréteur pour restabiliser le grafcet après la prise en compte de l'entrée.

Ex.



Le grafcet a deux entrées e_1 e_2, il est possible de connaître une borne max. du temps de calcul associé aux étapes 1 2 ... 8. Soit C_1, $C_2 \ldots C_8$ ces temps de calcul. On connaît aussi une borne max. du temps de l'algorithme d'évolution Soit B cette borne.

e_1 entraînera un temps max. d'exec. de $C_1 + C_2 + C_3 + C_4 + 3\ B$

e_2 entraînera un temps max. d'exec. de $C_5 + \max(C_6\ C_7) + C_3 + 3\ B$.

2) Une fois ceci fait, il faut associer à toute entrée un délai de réponse à respecter pour le traitement de cette entrée. Ceci se fait en prenant le minimum des délais de réponses des sorties entraînées par cette entrée

soit $d_i = \min_{J \in S_i} \Delta_J$

où S_i représente l'ensemble des sorties éventuellement déclenchées par e_i.

3) L'interpréteur GRAFCET est en fait un moniteur appliquant une stratégie FIFO (alg. 2) non interruptible (une entrée est entièrement traitée avant la suivante). On peut donc appliquer la validation proposée en (V.2.1.1.2.B).

Ex. Reprenons l'exemple 1 :

Soit d_1 le délai de réponse défini sur e_1 et d_2 sur e_2. Une condition suffisante pour que l'interpréteur Grafcet n'émette pas d'alarme est :

$$\begin{cases} \dfrac{C_1 + C_2 + C_3 + C_4 + 3B}{T_1} + \dfrac{C_5 + \max(C_6\ C_7) + C_8 + 3B}{T_2} \leq 1 \\ \text{et} \quad \sum_{i=1}^{n} C_i + 6\ B \leq d_1 \text{ ou } d_2 \end{cases}$$

en négligeant le temps de saisie des entrées.

On voit donc qu'il est possible à partir d'un grafcet donné et de l'interpréteur Grafcet de s'assurer de la non émission d'alarme.

Pour pouvoir appliquer cette démarche à des Grafcet de la taille de ceux décrivant des applications industrielles,il faudrait développer des outils permettant :

- Le calcul des temps d'exécution des tâches associées aux étapes
- Le calcul du temps maximum d'exécution à partir d'une entrée connaissant le temps maximum d'exécution des tâches. Ceci semble pouvoir se faire automatiquement à l'aide d'une méthode semblable à celle développée dans [22] pour la recherche de flot dans un graphe de contrôle parallèle.

CHAPITRE VI

APPLICATION DE LA DEMARCHE DE CONCEPTION

A LA CONCEPTION DU POSTE D'AIGUILLAGE INFORMATISE

Ce travail s'est inscrit dans le cadre d'une collaboration entre la SNCF et notre groupe. Les résultats détaillés ont été donnés dans [26]. Nous présenterons ici les parties les plus significatives.
Les documents de départ étaient ceux fournis par la SNCF [27]. Il nous a paru utile d'en faire une synthèse informelle [VI.1]. Nous en avons tiré des spécifications de type comportemental [VI.2], puis algorithmique [VI.3].
L'implantation de ce système est alors quasi immédiate. Ceci grâce à l'interpréteur GRAFCET fidèle.

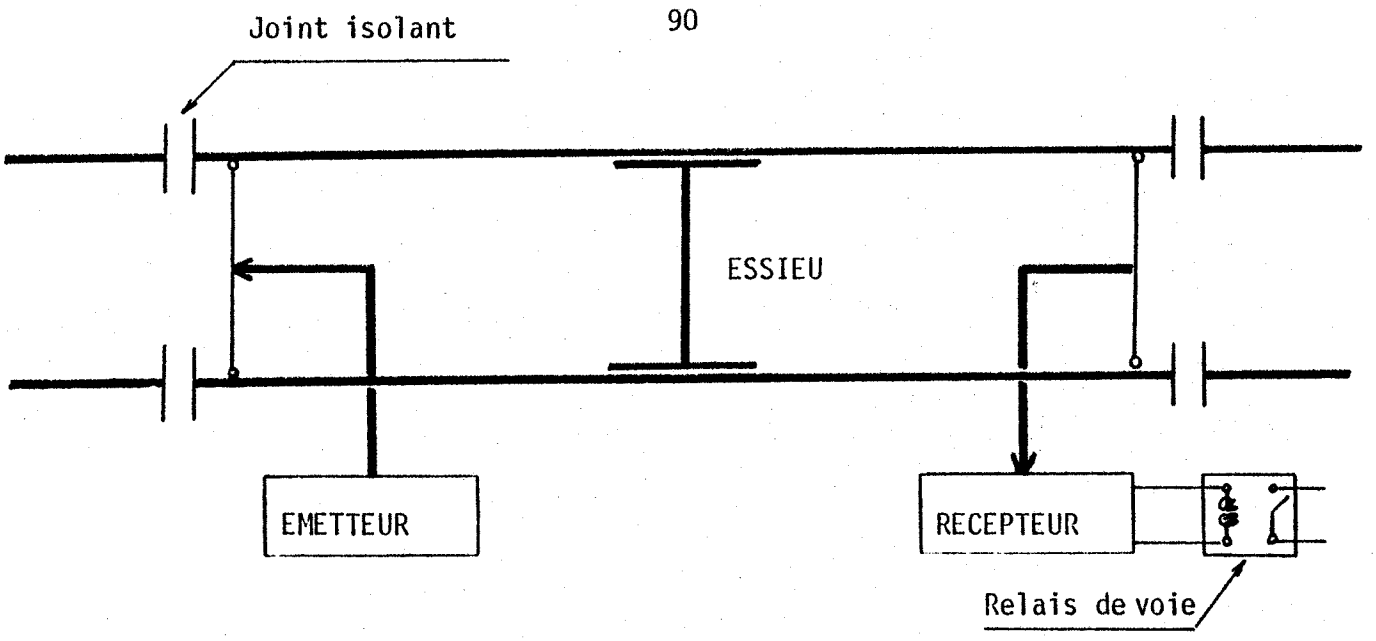
VI.1 - PRÉSENTATION D'UN POSTE D'AIGUILLAGE

Nous allons d'abord présenter quelques généralités sur le réseau SNCF, avant d'aborder le principe du poste d'aiguillage.

VI.1.1. GENERALITE SUR LE RESEAU SNCF

VI.1.1.1. Les voies

Le contrôle du trafic ferroviaire imposant de connaître à tout instant la position des trains, toutes les voies du réseau SNCF sont partitionnées en ZONES ISOLEES. Chacune de ces zones est munie d'un dispositif électrique, appelé CIRCUIT DE VOIE, permettant de détecter la présence d'un train.
Un circuit de voie est composé des deux rails limités électriquement à leurs extrémités par des joints isolants (d'où l'appellation de "zone isolée"). Ceux-ci sont reliés entre eux à l'une des extrémités par un émetteur de courant, et à l'autre, par un récepteur commandant un relais électromagnétique appelé RELAIS DE VOIE.



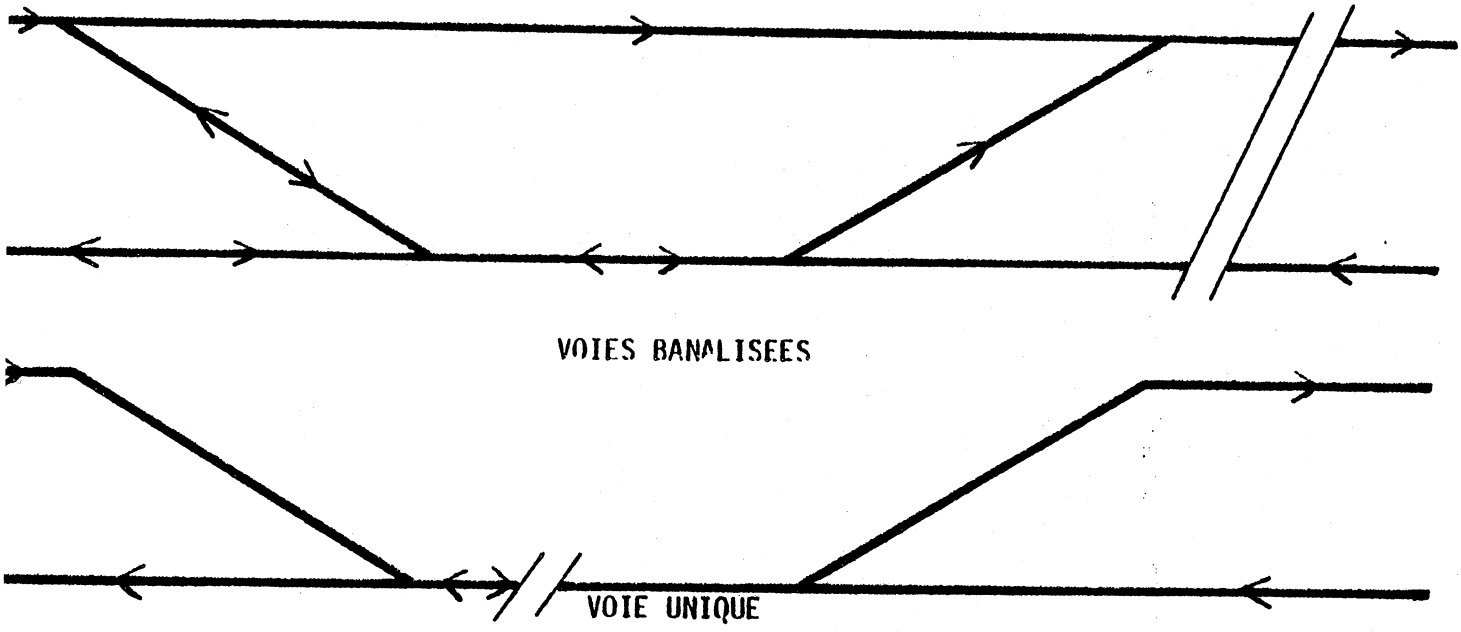
CIRCUIT DE VOIE

En l'absence de train, le récepteur reçoit une alimentation et peut exciter le relais de voie ; par contre, la présence d'un essieu court-circuite le récepteur et désexcite le relais.

Pour faciliter le contrôle du trafic, les zones isolées contiennent généralement peu d'aiguilles. Lorsque cela est possible, leur longueur est choisie voisine de 1500 mètres ; mais elle peut être réduite à quelques dizaines de mètres si les aiguilles sont très rapprochées, notamment au voisinage des gares.

Afin de permettre une circulation simultanée dans les deux sens, une ligne SNCF comporte de façon générale deux voies, équipées du dispositif précédemment décrit, chacune étant réservée à la circulation dans un sens.

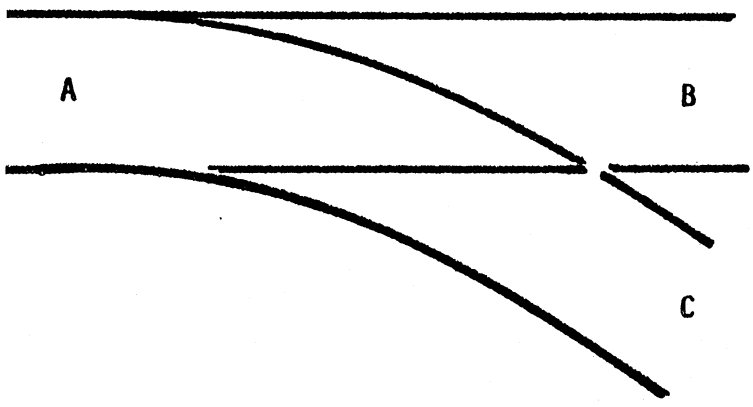
Cependant, pour augmenter la disponibilité du réseau, sur certaines portions de voie, le trafic peut s'effectuer indifféremment dans les deux sens : de telles voies sont dites VOIES BANALISEES. Si, de plus, elles ne contiennent pas d'aiguilles, elles sont dites VOIES UNIQUES car un train engagé sur une telle partie de voie ne peut être aiguillé et doit la parcourir entièrement sans effectuer de manoeuvres.



VI.1.1.2. Les aiguilles

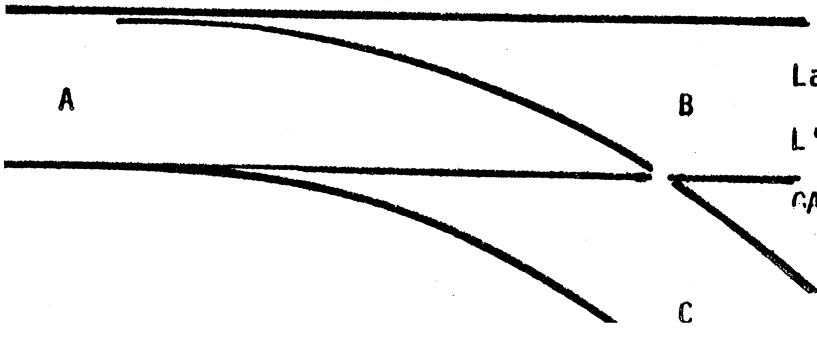
La fonction d'une aiguille est d'imposer une direction à un train au niveau d'une bifurcation, le principe étant d'assurer la plus parfaite continuité de la voie dans la direction choisie.

Deux situations peuvent se présenter :



La voie AC est reconstituée

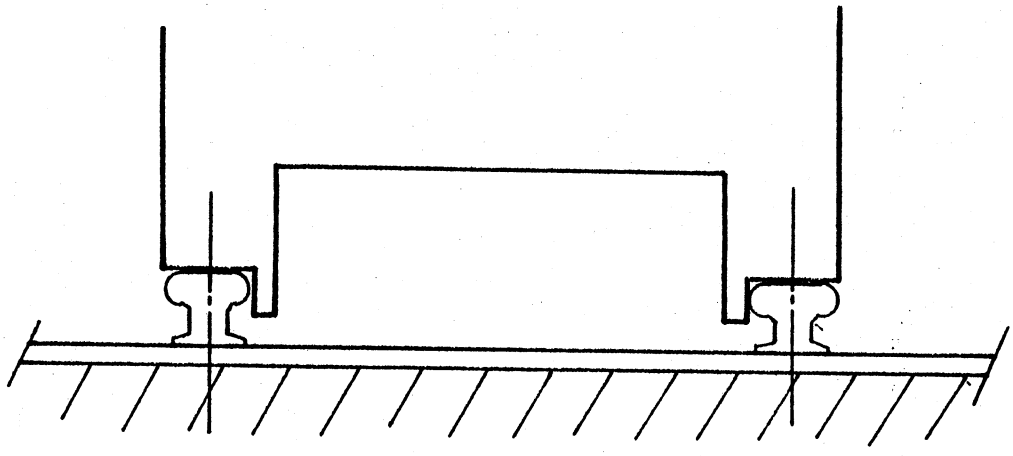
L'aiguille est dite en position DROITE.



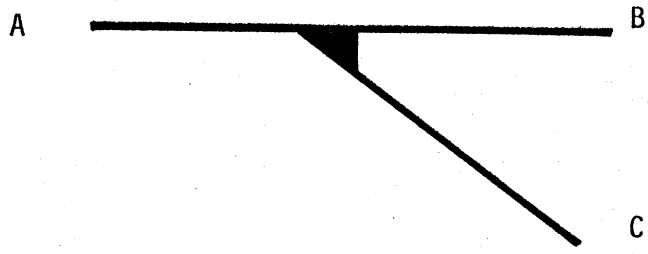
La voie AB est reconstituée.

L'aiguille est dite en position GAUCHE.

Dans ces deux situations, le bon aiguillage des trains est assuré par le profil particulier des roues et leur position sur les rails.



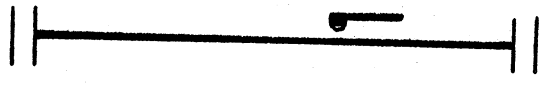
Une aiguille sera représentée par le symbole ci-dessous :



VI.1.1.3. Les pédales

Implantées sur certaines zones isolées, les pédales sont des dispositifs mécaniques permettant de détecter le passage d'un train. L'information fournie par une pédale est évidemment redondante avec celle du relais de voie correspondant ; ceci permet une augmentation de la sécurité en évitant toute erreur d'interprétation due à la défaillance d'un des dispositifs ou à la présence d'un corps étranger sur la voie.

Une pédale sera représentée par le symbole ci-dessous :



VI.1.1.4. La signalisation

Actuellement, la signalisation est principalement assurée par des feux de couleur, qui ont progressivement remplacé les signaux mécaniques.

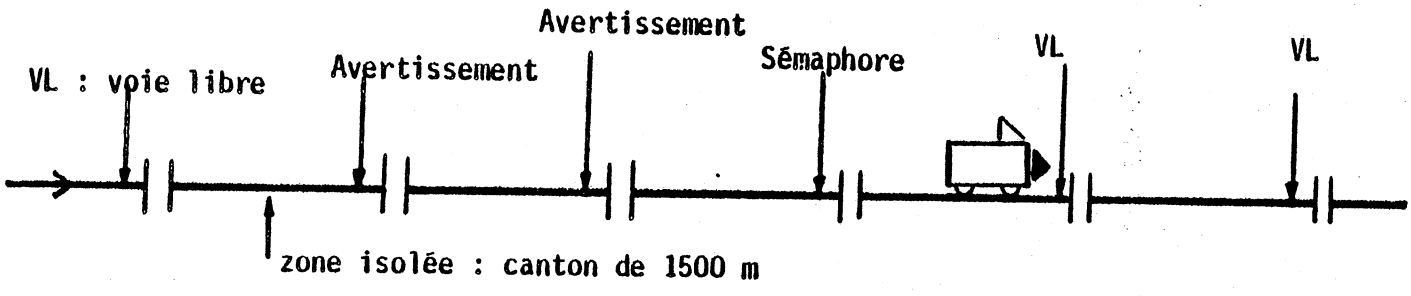
Sur l'ensemble du réseau, trois feux groupés au sein d'un même dispositif protègent chaque zone isolée :

- le SEMAPHORE (feu rouge) impose l'arrêt puis autorise une circulation à vue jusqu'au feu suivant.
- l'AVERTISSEMENT (feu jaune) commande un ralentissement permettant l'arrêt au feu suivant.
- le signal de VOIE LIBRE (feu vert) n'impose aucune restriction.

Ces différents signaux permettent le contrôle des voies suivant la LOGIQUE DE CANTONNEMENT :

Afin d'éviter les risques de rattrapage et de collision, l'entrée d'un train sur une zone isolée provoque la présentation (mise au rouge) du sémaphore protégeant cette zone et celle des avertissements protégeant les deux zones précédentes.

L'accès à toute autre zone n'est pas restreint : le signal de voie libre est présenté.



LOGIQUE DE CANTONNEMENT

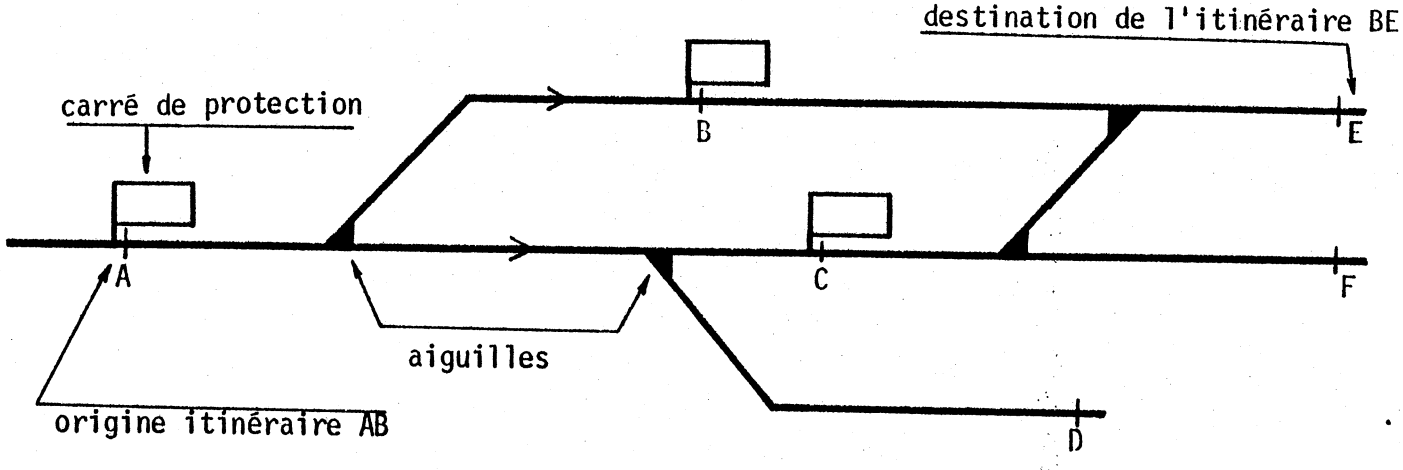
Dans les sections comportant des aiguillages, un signal supplémentaire permet d'imposer un arrêt impératif dans l'attente du bon positionnement des aiguilles : le carré (deux feux rouges verticaux).

Bien qu'en vigueur dans les zones d'aiguillage, la gestion des sémaphores, avertissements et feux de voie libre suivant la logique de cantonnement, n'est pas assurée par le poste, dont l'action ne porte que sur les carrés.

VI.1.2. PRINCIPE DU POSTE D'AIGUILLAGE

VI.1.2.1. Notion de poste à ITINERAIRES

Un itinéraire AB est un chemin particulier dans une section de voies comportant des aiguilles, permettant aux trains arrivant en A de se diriger vers B. Son accès est protégé par un carré.



Exemples d'itinéraires : AB, AC, AD, BE, CE, CF.

ITINERAIRES

Dans les postes actuels, un ensemble fixé d'itinéraires est mis à la disposition de l'aiguilleur. Celui-ci, pour assurer le passage d'un train sur un parcours choisi (par exemple AE), est amené à sélectionner un ou plusieurs itinéraires consécutifs (par exemple AC puis CE) par pression sur les boutons de commande d'itinéraires associés. Ces boutons ont une seule position active atteinte par pression, et possèdent chacun un voyant de contrôle éteint, clignotant ou allumé selon l'état de l'itinéraire associé. L'ensemble de ces boutons, ainsi qu'un ensemble de commutateurs, permettant la fermeture immédiate des carrés de protection, sont regroupés sur un même pupitre de commande. Ce pupitre est installé dans la salle d'aiguillage à proximité d'un tableau de contrôle optique (T.C.O.). Le T.C.O. est une image de la zone d'action du poste, montrant schématiquement le tracé des voies, l'emplacement des signaux et des aiguilles, ainsi que le découpage des zones isolées. Des voyants lumineux renseignent l'aiguilleur sur l'état des signaux, la position des aiguilles et l'état d'occupation des zones isolées, lui permettant ainsi de suivre le passage des trains.
Dans les anciens postes, l'aiguilleur était contraint de commander individuellement chacun des appareils (aiguilles, carrés) concernés par un parcours.
Désormais, les postes à itinéraires, installés de façon courante dans les gares d'une certaine importance, gèrent globalement l'ensemble des commandes des appareils concernés à partir de la pression de l'aiguilleur sur un bouton de commande d'itinéraire. Deux fonctions principales sont réalisées par le poste :

- l'établissement d'un itinéraire,
- la destruction d'un itinéraire demandé.

VI.1.2.2. Etablissement d'un itinéraire

Sous les termes "établissement d'un itinéraire", sont regroupées les trois étapes définies par la S.N.C.F. :

- enregistrement et préparation
- enclenchement
- contrôle.

Elles vont de la prise en compte de la commande de l'aiguilleur à l'ouverture du signal de protection, l'itinéraire étant prêt pour le passage d'un train.

VI.1.2.2.1. Enregistrement et préparation de l'itinéraire

La commande de l'itinéraire est réalisée manuellement, à partir du pupitre, par pression sur le bouton correspondant :

- l'itinéraire est dit ENREGISTRE
- le voyant de contrôle intégré au bouton clignote au blanc.

Le système cherche alors à positionner les organes de commande des aiguilles empruntées par l'itinéraire. Tant que l'un au moins d'entre eux est immobilisé par un autre itinéraire dans une position contraire, la commande est mémorisée dans l'attente de la levée des incompatibilités (les deux itinéraires sont alors dits INCOMPATIBLES).

VI.1.2.2.2. Enclenchement de l'itinéraire

Les organes de commande des aiguilles de l'itinéraire étant correctement positionnés, le système les immobilise dans leur position : ils sont dits ENCLENCHES.

L'itinéraire est FORME :

- le voyant de contrôle de l'itinéraire passe du blanc clignotant au blanc fixe.
- son tracé au T.C.O. s'allume au blanc fixe.

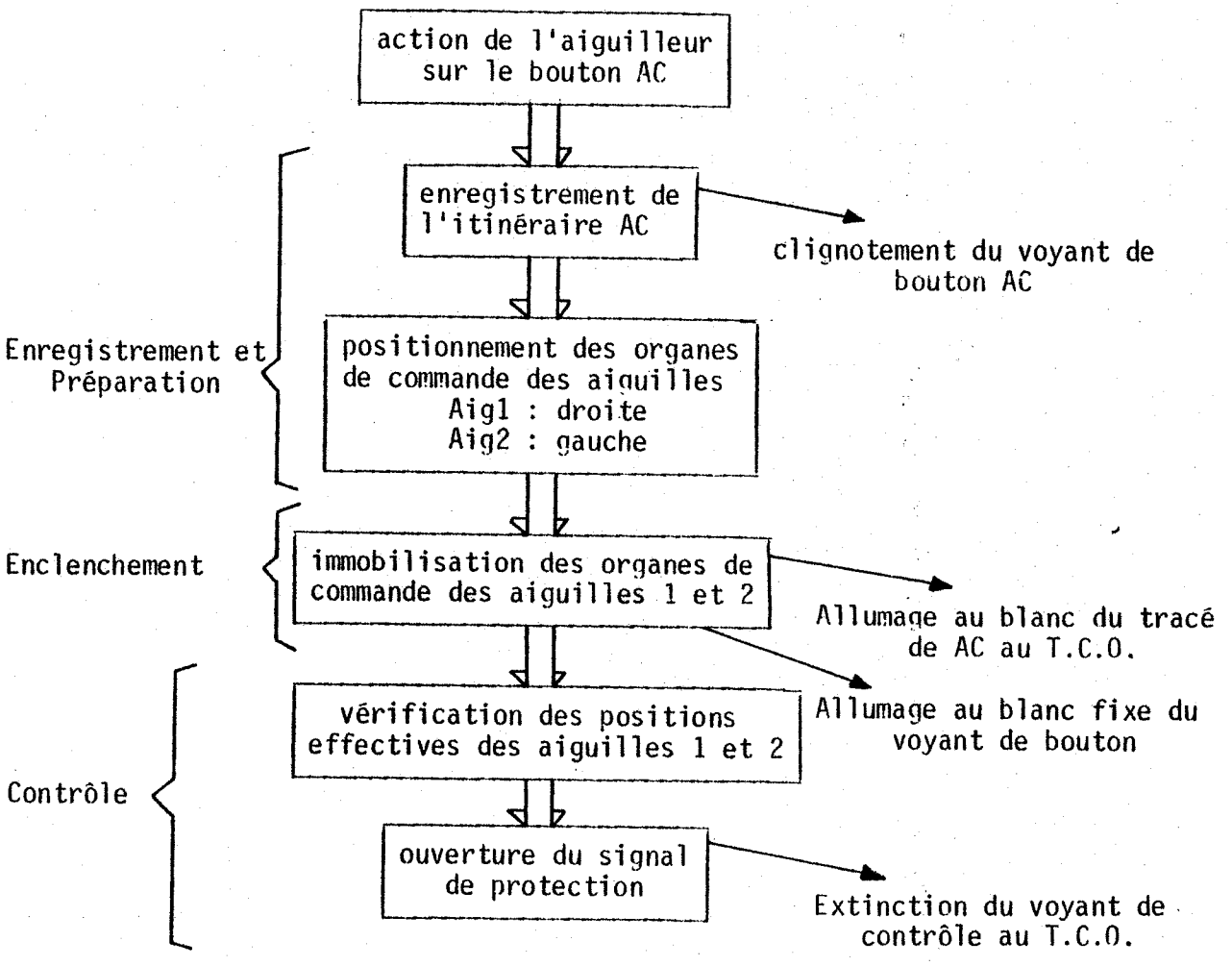
VI.1.2.2.3. Contrôle de l'itinéraire

Afin de s'assurer que les aiguilles ont correctement réagi aux commandes envoyées par le poste d'aiguillage, une vérification de la concordance entre leur position effective, sur le terrain, et l'indication portée par leurs organes de commande respectifs, est effectuée avant l'envoi de la commande d'ouverture du signal de protection.

D'autres contrôles, propres à la configuration de l'itinéraire, peuvent différer cette commande d'ouverture (cf. § II.5 Enclenchements entre itinéraires de sens contraires).

Après ouverture du signal de protection, entraînant l'extinction du voyant de carré au T.C.O., plus rien ne s'oppose au passage du train. Celui-ci est suivi, au T.C.O., par la mise des voyants du tracé au rouge.

SYNOPTIQUE



ETABLISSEMENT DE L'ITINERAIRE AC

VI.1.2.3. Possibilités de destruction des itinéraires

La destruction d'itinéraires, devenus inutiles, est nécessaire pour permettre la formation éventuelle d'autres itinéraires précédemment incompatibles. Elle peut s'effectuer de deux manières distinctes :

- automatiquement
- manuellement.

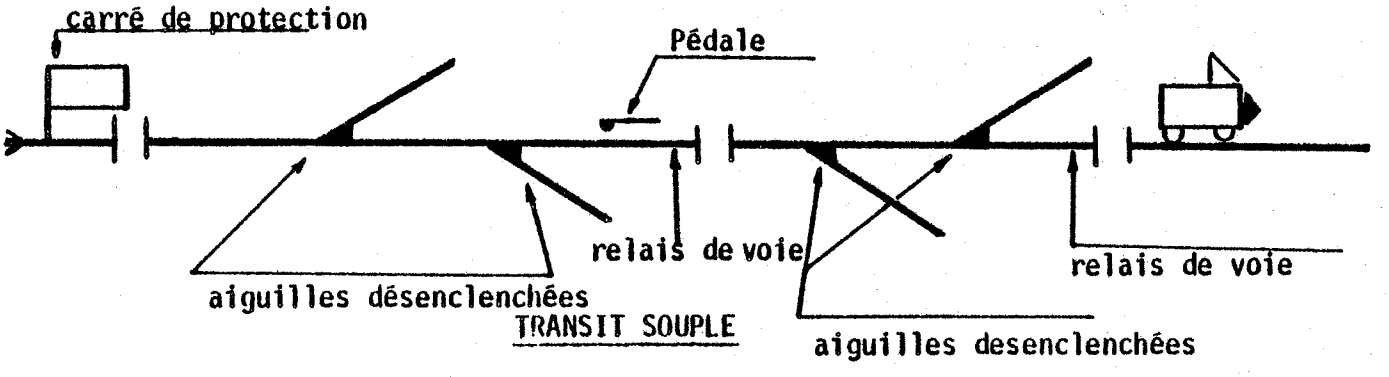
VI.1.2.3.1. Destruction automatique : Transit souple

La destruction de l'itinéraire est obtenue, sans intervention de l'aiguilleur, par le simple passage du train, suivant le principe du TRANSIT SOUPLE :
la succession des trois événements :

- occupation de la zone isolée située juste en aval du signal de protection (entraînant la fermeture immédiate de ce signal, ainsi que l'affichage au T.C.O. du contrôle de fermeture),
- attaque de la pédale placée sur cette zone,
- libération de cette zone,

qui utilise pleinement une redondance d'information en provenance du terrain, initialise le processus de destruction. Dès lors, l'itinéraire n'est plus considéré comme formé ; le voyant de table associé s'éteint et, après l'envoi sur le terrain d'une confirmation de fermeture du carré, les aiguilles de cette première zone sont libérées.
Cependant, les organes de commande des aiguilles des autres zones restent enclenchés et ne sont libérés, zone par zone, qu'après le passage du train à condition que :

- toutes les zones isolées, situées en amont sur l'itinéraire, aient été libérées,
- le train ait occupé la zone considérée et vienne de la libérer.



Les voyants de contrôle au T.C.O. de la zone isolée, allumés au rouge lors de son occupation par le train, s'éteignent à sa libération.
Ce procédé permet une formation plus rapide d'itinéraires incompatibles, tout en respectant les normes de sécurité au niveau des aiguilles.

VI.1.2.3.2. Destruction manuelle

La destruction manuelle permet à l'aiguilleur de détruire un itinéraire commandé par erreur, sans attendre le passage d'un train. Pour des raisons impératives de sécurité, cette destruction n'est pas exécutée lorsqu'un train va pénétrer sur l'itinéraire ou y circule déjà.

-- Destruction manuelle d'un itinéraire enregistré.

Elle est obtenue par une nouvelle pression sur le bouton associé à l'itinéraire. L'effacement de l'enregistrement est immédiat ; le voyant de bouton, qui clignotait au blanc, s'éteint.

-- Destruction manuelle d'un itinéraire formé.

.Itinéraires munis d'une ZONE D'APPROCHE.

Un itinéraire est dit muni d'une zone d'approche lorsque le système peut, en plus des informations portées par les circuits de voie des zones composant l'itinéraire, accéder à celles portées par les circuits de voie d'un ensemble de zones situées en amont du signal de protection. Dans ce cas, un voyant de contrôle au T.C.O., indique à l'aiguilleur la présence d'un train dans la zone d'approche. Ce dispositif permet de déterminer les cas où une destruction manuelle immédiate serait rendue dangereuse par la présence d'un train à proximité de l'origine de l'itinéraire : ce train risque en effet, à cause de sa vitesse, de ne pas pouvoir s'arrêter avant de pénétrer sur l'itinéraire.
Lorsque la zone d'approche est libre (voyant éteint), la destruction est obtenue immédiatement par pression sur le bouton associé à l'itinéraire ; le voyant de

table, allumé au blanc fixe, s'éteint.

Le système commande alors la fermeture ou la confirmation de fermeture du signal de protection et libère les organes de commande des aiguilles. Sur le T.C.O., le voyant de contrôle du carré de protection s'allume au rouge et le tracé de l'itinéraire s'éteint.

Lorsque la zone d'approche est occupée (voyant allumé au rouge), si le carré de protection est fermé, la procédure de destruction est analogue à la précédente ; sinon, l'aiguilleur doit commander la fermeture du signal de protection à l'aide d'une commande manuelle à usage limité, puis la destruction par pression sur le bouton associé à l'itinéraire. Le système temporise alors cette demande pendant trois minutes. A l'expiration de ce délai :

- soit l'itinéraire a été détruit par le passage du train n'ayant pu s'arrêter devant le signal de protection (Destruction Automatique), et la commande de Destruction Manuelle n'a plus raison d'être.
- soit le train s'est arrêté devant le signal de protection ; le système commande alors la confirmation de fermeture de ce signal et libère les organes de commande des aiguilles. Sur le T.C.O., le tracé de l'itinéraire, allumé au blanc fixe, s'éteint.

.Itinéraires non munis d'une ZONE D'APPROCHE.

Ce sont des itinéraires, empruntant essentiellement des voies de garage, pour lesquels deux cas sont à considérer :

- soit le signal de protection n'a pas été commandé à l'ouverture ; la destruction est immédiate par pression sur le bouton d'itinéraire.
- soit le signal de protection est ou a été commandé à l'ouverture ; pour obtenir la destruction, l'aiguilleur doit fermer ce signal à l'aide de la commande manuelle puis appuyer sur le bouton d'itinéraire. La destruction n'est effectuée par le système qu'après temporisation de la demande.

VI.1.2.4. Tracé permanent

Le principe de la destruction automatique contraint l'aiguilleur, lors du passage successif de trains sur un même itinéraire, à renouveler la commande de cet itinéraire.

Afin de réduire cette contrainte, pour certains itinéraires s'ajoute à la commande classique, vue précédemment, un mode de commande supplémentaire : le TRACE PERMANENT, obtenu par pression sur un bouton (T.P) voisin du bouton d'itinéraire (D.A.).

Ces deux modes de commandes ne diffèrent pas au niveau de l'établissement de l'itinéraire ; seule l'annulation de la Destruction Automatique et de la fermeture du signal de protection caractérise le tracé permanent. La protection contre d'éventuels rattrapages n'est plus alors assurée que par la gestion des sémaphores suivant la logique de cantonnement. Cette similitude permet d'obtenir la transformation d'un itinéraire, enregistré ou formé en mode Destruction Automatique, en un itinéraire en mode Tracé Permanent, par simple pression sur le bouton T.P. associé.

La destruction d'un itinéraire commandé en Tracé Permanent nécessite l'intervention de l'aiguilleur :

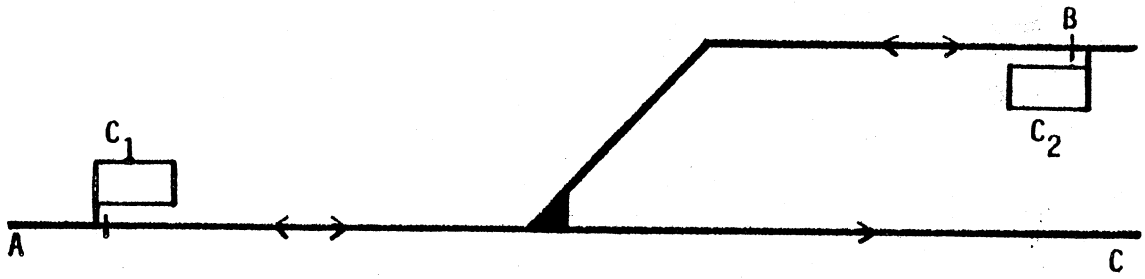
- une nouvelle pression sur le bouton T.P. provoque la transformation de l'itinéraire commandé en Tracé Permanent, en itinéraire commandé en Destruction Automatique.
- une pression sur le bouton d'itinéraire (D.A.) provoque sa Destruction Manuelle.

VI.1.2.5. Enclenchements entre itinéraires de sens contraires

Le but de ces enclenchements est d'interdire aux carrés origines de deux itinéraires de sens contraires, permettant l'accès à une même partie de voie banalisée, d'être ouverts simultanément. Un tel dispositif de sécurité est nécessaire car les

contraintes imposées lors de la formation n'empêchent pas que deux tels itinéraires soient formés.

Exemple :

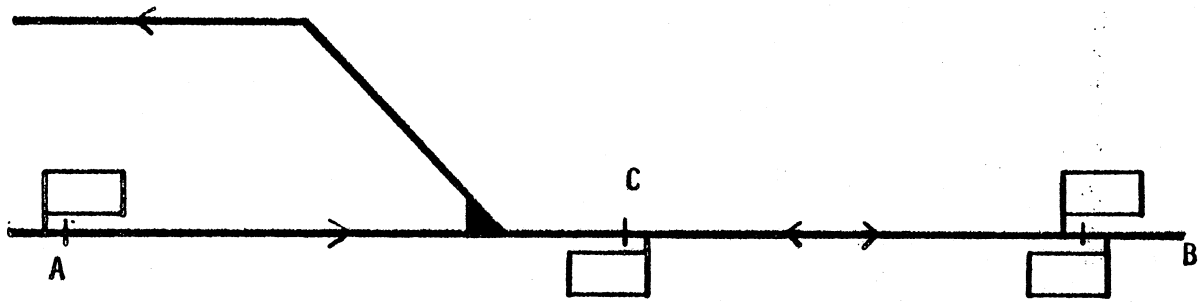


Les itinéraires AB et BA peuvent être formés.

C1 et C2 ne doivent pas être ouverts simultanément.

- Enclenchement de parcours banalisé.

Cet enclenchement protège les itinéraires de sens contraires possédant une partie de voie banalisée commune : ce sont en général un itinéraire et son inverse, ou deux itinéraires ayant une seule extrémité commune.



Les itinéraires AB et BC ont la partie de voie CB en commun.

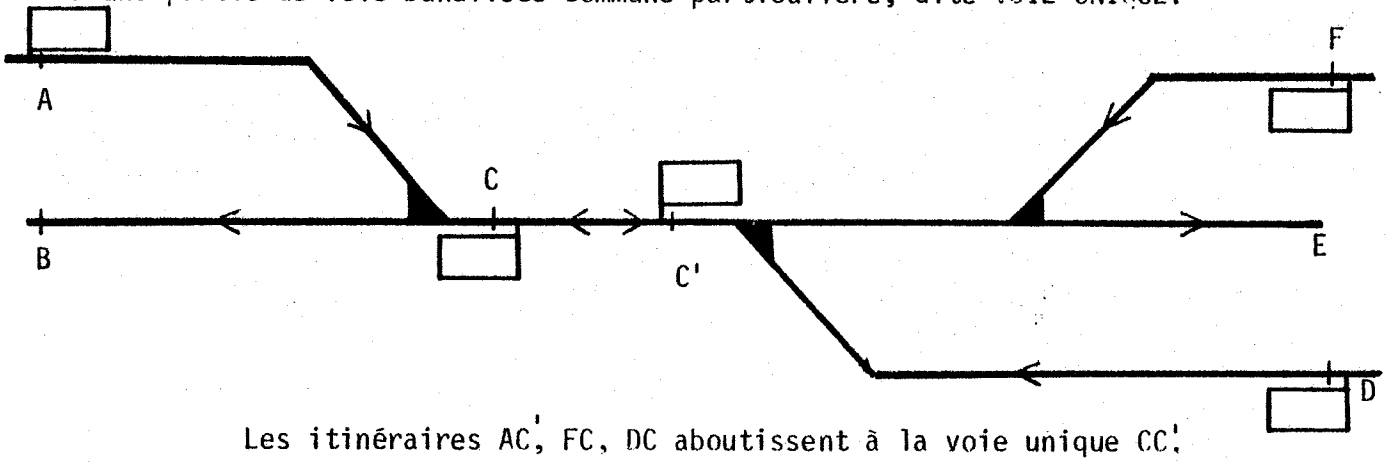
Leur seule extrémité commune est B.

Dans ce cas, aux conditions d'ouverture du carré vues précédemment (cf. §II.2.3 contrôle d'un itinéraire), s'ajoutent les contraintes suivantes :

- aucun itinéraire de sens contraire n'est ouvert
- aucun train circulant en sens contraire n'occupe un itinéraire de sens contraire.

- Enclenchement de voie unique.

Cet enclenchement protège les itinéraires de sens contraires aboutissant à une partie de voie banalisée commune particulière, dite VOIE UNIQUE.



Les itinéraires AC', FC, DC aboutissent à la voie unique CC'.

Dans ce cas, aux conditions d'ouverture du carré vues précédemment (cf. §II.2.3 contrôle d'un itinéraire), s'ajoutent les contraintes suivantes :

. aucun itinéraire de sens contraire aboutissant à la même voie unique n'est ouvert.

. aucun train circulant en sens contraire n'occupe un itinéraire de sens contraire ni la partie de voie unique.

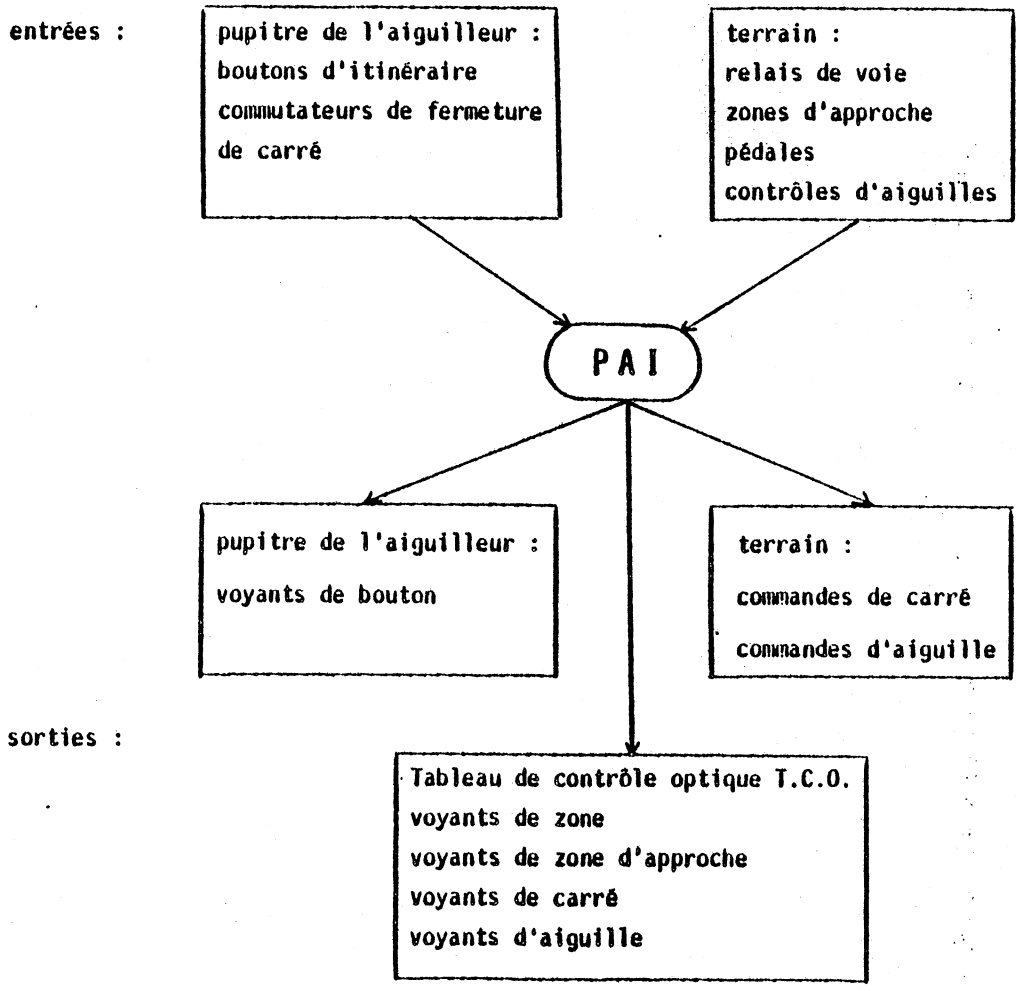
VI.1.3. CONCLUSION

Cette présentation décrit le principe général du poste d'aiguillage. Elle n'inclut pas les cas d'exceptions (refoulement...manoeuvres manuelles, etc). Malgré cette simplification, la lecture, la compréhension des documents SNCF et la rédaction de la présentation informelle, a représenté environ 2 mois/h de travail.
Partant de cette description, nous en avons tiré une spécification de type comportementale.

VI.2 - SPÉCIFICATION COMPORTEMENTALE DU PAI

On peut représenter le PAI par le graphe suivant :

Représentation symbolique du P.A.I.



Il paraît difficile de décrire le PAI sans faire entrer en jeu une certaine décomposition de ces fonctions principales, et d'utiliser pour cela un certain nombre de variables intermédiaires.

De plus, il faut donner une représentation mathématique des variables d'entrées et sorties, ainsi que des variables relatives à la partie du réseau commandé par le poste d'aiguillage. La représentation mathématique de ces variables est la suivante :

VI.2.1. REPRESENTATION MATHEMATIQUE DU RESEAU

Le réseau peut être défini comme :

- un ensemble de portions P de cardinalité nbp.

  la notion de portion est une notion qui n'est pas SNCF mais que nous avons introduite pour permettre une bonne description mathématique :

  . une portion est la partie élémentaire du réseau : elle contient au plus une aiguille et est inclue dans une zone isolée.

$$P = \{P_1, P_2 \dots P_{nbp}\}$$

- un ensemble de zones isolées de cardinalité nbz

$$Z = \{Z_1, Z_2 \dots Z_{nbz}\}$$

- un ensemble d'itinéraires IT de cardinalité nbit

$$IT = \{IT_1, IT_2 \dots IT_{nbit}\}$$

a) une zone isolée est un élément de $P(P)$

ex $Z_i = \{P_1, P_3, P_7\}$

de plus, $\forall i \; \forall j \; Z_i \cap Z_j = \emptyset$

b) un itinéraire est un élément de $P(P)$ ayant deux portions ou plus

$$IT_i = \{P_{k_1}, P_{k_2}, P_{k_{nbp(IT_i)}}\} \quad \text{avec } nbp(IT_i) \leq 2.$$

. pour tout itinéraire IT(i) la fonction $POS_i : P \to \{0,1,2,3\}$ associe à chaque portion, la position de son aiguille pour l'itinéraire IT_i

$POS_i(P_k) = 0 \Leftrightarrow P_k \in IT_i$ et P_k contient une aiguille qui doit être positionnée à gauche pour IT_i.

$POS_i(P_k) = 1 \Leftrightarrow P_k \in IT_i$ et P_k contient une aiguille qui doit être positionnée à droite pour IT_i.

$POS_i(P_k) = 2 \Leftrightarrow P_k \in IT_i$, mais ne contient pas d'aiguille

$POS_i(P_k) = 3 \Leftrightarrow P_k \notin IT_i$.

.A chaque itinéraire i est associé l'ensemble des zones isolées Z^i de cardinalité $nbz(IT_i) \geq 2$, tel que :

$$Z^i = \{Z_{J_1}, Z_{J_2}, \ldots Z_{J_{nbz(IT_i)}}\} \text{ avec } Z_{J_1} \cap IT_i \neq \emptyset$$

sur cet ensemble est définie la relation d'ordre total

$$\forall (Z_{J_l}, Z_{J_{l'}}) \in (Z^i)^2, Z_{J_l} \leq Z_{J_{l'}} \Leftrightarrow l \leq l'$$

. la fonction PREM associe à IT(i) sa première zone pour la relation $\leq$.

. la fonction DER associe à IT(i) sa dernière zone pour la relation $\leq$.

. pour tout itinéraire IT_i, la fonction $PRED_i$ $Z^i \rightarrow Z^i \cup \emptyset$ associe à chaque zone de Z^i la zone la précédant lorsqu'elle existe et $\emptyset$ sinon.

c) Carrés

L'ensemble des carrés de protection noté C est de cardinalité nbc

la fonction CAR : IT → C associe à chaque itinéraire son carré de protection.

d) Pédales

L'ensemble des pédales est noté PD, sa cardinalité est nbpd.

La fonction PED : IT → PD associe à chaque itinéraire sa pédale de destruction automatique.

e) Zone d'approche

L'ensemble des zones d'approche ZAP est de cardinalité nbzap

$$ZAP = \{ZAP_1, ZAP_2, \ldots, ZAP_{nbzap}\}.$$

La fonction ZAPPROCHE : IT → ZAP ∪ $\{ZAP_0\}$ associe à chaque itinéraire, s'il en possède une, sa zone d'approche, sinon ZAP_0.

f) contrôle d'aiguilles

L'ensemble des contrôles d'aiguilles noté KAG est de cardinalité nbkag.

La fonction Kontrolaig : $P \rightarrow KAG \cup \{KAG_0\}$ associe à chaque portion le contrôle d'aiguille associé, sinon KAG_0

On définit pour tout itinéraire i l'ensemble de ses contrôles d'aiguilles associé

$KAG^i = \{KAG_g$ tel que $\exists P_k \in IT_i$ Kontrolaig$(P_k) = KAG_g$

g) de la même façon, on définit des ensembles permettant de modéliser les voies uniques, le sens des itinéraires, etc...

Définissons maintenant les variables d'entrée, de sortie et intermédiaire du PAI.

VI.2.2. VARIABLES D'ENTREE

a) en provenance du terrain

- l'ensemble des relais de voies RV est en bijection avec l'ensemble Z des zones isolées. Le relais de voies associé à Z_J est noté RV_J.

  $RV_J = 0$ la zone isolée Z_J est libre

  $RV_J = 1$ la zone isolée Z_J est occupée.

- à tout élément PD_p de l'ensemble des pédales est associé une variable

  $PD_p = 0$ la pédale PD_p est relevée

  $PD_p = 1$ la pédale PD_p est attaquée.

- à tout élément ZAP_z est associé une variable

  $ZAP_z = 0$ ZAP_z est libre

  $ZAP_z = 1$ ZAP_z est occupée.

L'élément ZAP_0 a sa variable associée toujours égale à 1.

- à tout élément KAG_g est associée une variable

  KAG_g = 0 signifie l'aiguille est positionnée à gauche

  KAG_g = 1 signifie l'aiguille est positionnée à droite

L'élément KAG_0 a une variable associée de valeur toujours égale à 2.

b) en provenance du pupitre de l'aiguilleur

. l'ensemble des boutons d'itinéraire est bien entendu en bijection avec l'ensemble des itinéraires.

Le bouton associé à l'itinéraire IT_i est noté $BNIT_i$. On associe une variable

$BNIT_i$ = 0 signifie:il n'y a pas pression sur le bouton $BNIT_i$

$BNIT_i$ = 1 signifie:il y a pression sur le bouton $BNIT_i$

On utilisera en fait le compteur d'événement associé à cette variable.

. l'ensemble des commutateurs de fermeture de carré est en bijection avec l'ensemble des carrés.

FC_c = 1 signifie le carré C_c doit être mis au rouge

FC_c = 0 signifie le carré C_c est géré par les autres variables.

VI.2.3. VARIABLES INTERMEDIAIRES ET VARIABLES DE SORTIES

. L'ensemble état des portions de cardinalité nbp est en bijection avec l'ensemble des portions. Tout élément de "état des portions" a trois composantes (POS_R, $Reservée_k$, $bloquée_k$).

Pos_k = 0 la portion P_k possède 1 aiguille qui doit être positionnée à gauche

Pos_k = 1 la portion P_k possède 1 aiguille qui doit être positionnée à droite

Pos_k = 2 la portion P_k n'a pas d'aiguille

$Réserve_k$ = 0 la portion P_k n'est pas réservée

$Réserve_k$ = 1 la portion P_k est réservée

$Bloquée_k$ = 0 : si la portion P_k a une aiguille, elle est libre

$Bloquée_k$ = 1 : si la portion P_k a une aiguille, elle est bloquée.

. L'ensemble des compteurs de libérations CLIB est en bijection avec l'ensemble des itinéraires.

L'élément associé à IT_c est $CLIB_i$

Tout élément $Clib_i$ est un ensemble ordonné de compteur de libération de zone

$$IT_i = (Z_{J1}, Z_{J2}, \ldots, Z_{J_{nbz(IT_i)}})$$

$$CLIB_i = (Clib_i(Z_{J_1}), \ldots, Clib_i(Z_{J_l}), \ldots, Clib_i(Z_{J_{nbz(IT_i)}}))$$

où $Clib_i(Z_{J_l})$ compte le nombre de libération de la zone Z_{J_l} pour IT_i.

. Pour chaque carré C_c la variable associée C_c permet la commande du carré

$C_c = 0$ signifie carré au rouge

$C_c = 1$ signifie carré au vert.

. Chaque voyant de bouton d'itinéraire VBNIT(i) est commandé par sa variable associée

$VBNIT_i = 0$ voyant de bouton éteint

$VBNIT_i = 1$ voyant de bouton allumé clignotant

$VBNIT_i = 2$ voyant de bouton allumé fixe.

. Les ensembles IT DEMANDE, IT RESERVE, IT FORME, IT OUVERT, IT PART DETRUIT (itinéraire partiellement détruit) IT DETRUIT sont en bijection avec l'ensemble des itinéraires. Chaque élément de cet ensemble est une variable à valeur dans {0, 1}.

ex. $IT\ DETRUIT_i = 0 \Leftrightarrow$ l'itinéraire i n'est pas détruit

$IT\ DETRUIT_i = 1 \Leftrightarrow$ l'itinéraire i est détruit

. On peut ainsi définir les variables de sortie à destination du voyant de contrôle optique de la même façon.

Nous allons maintenant décrire les relations entre toutes les variables

VI.2.5. DEFINITION DES RELATIONS ENTRE LES VARIABLES

Nous décomposerons le PAI en une dizaine de sous-systèmes, chaque sous-système est en fait une relation plus ou moins complexe entre les variables. La décomposition en sous-systèmes est complètement arbitraire et est uniquement faite dans le but d'aider la spécification. Elle n'est en aucun cas une obligation de réalisation. Ces sous-systèmes sont :

- COM sous-système d'interprétation de la pression de l'aiguilleur sur un bouton d'itinéraire.

- C sous-système de gestion de la compatibilité d'itinéraire

- F sous-système de positionnement des aiguilles

- K Autorisation physique (ouverture du carré)

- DM Destruction d'itinéraire demandée par l'aiguilleur

- DA Destruction automatique d'itinéraire

- CONT Contrôle le bon fonctionnement des moteurs d'aiguilles et de carré.

- GEST TCO qui est la gestion du tableau contrôle optique. Ce sous-système n'est pas critique et nous n'en tiendrons pas compte dans le travail ultérieur.

On peut donner une vue d'ensemble des relations avec les variables à l'aide des diagrammes suivants :

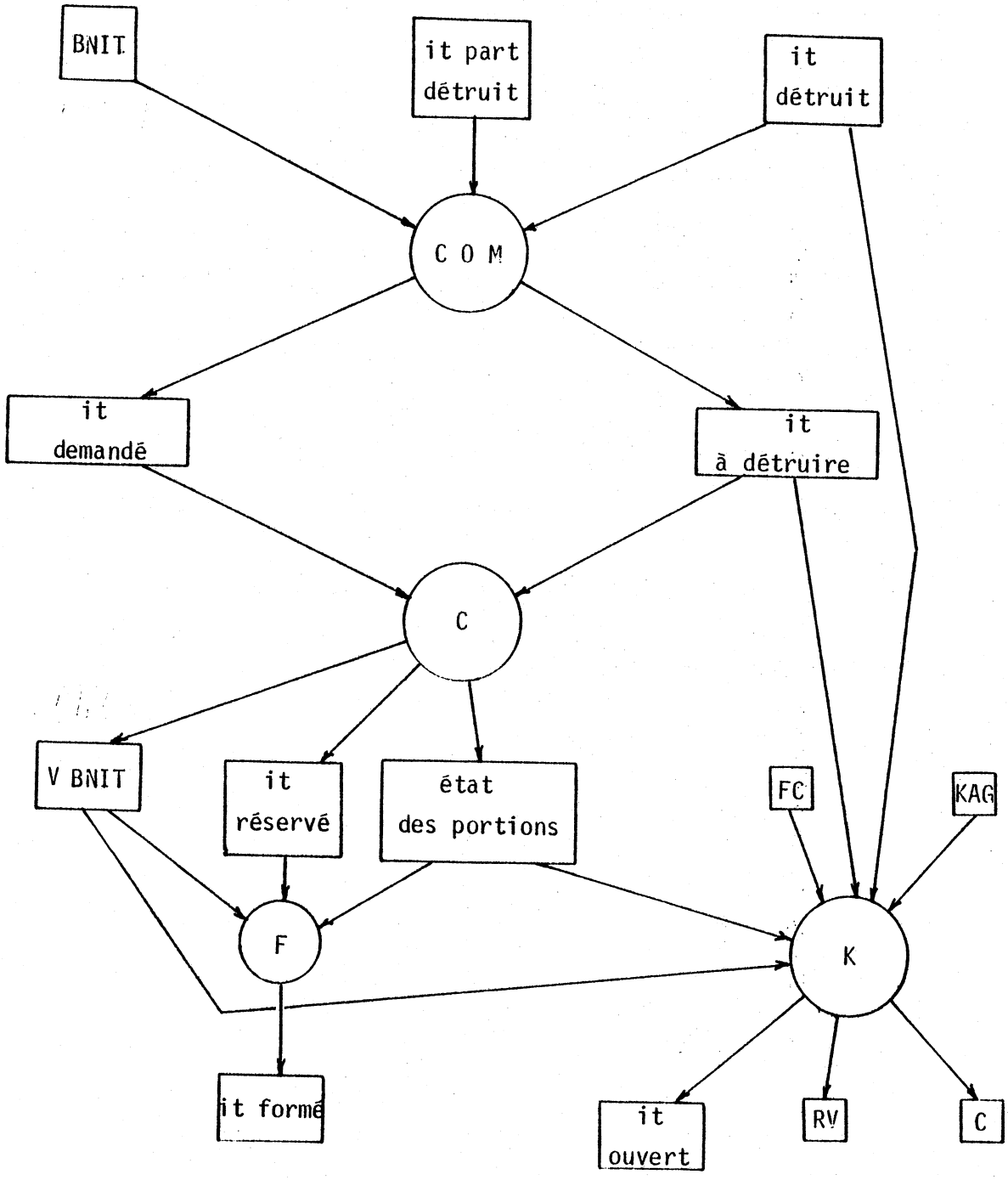


DIAGRAMME 1

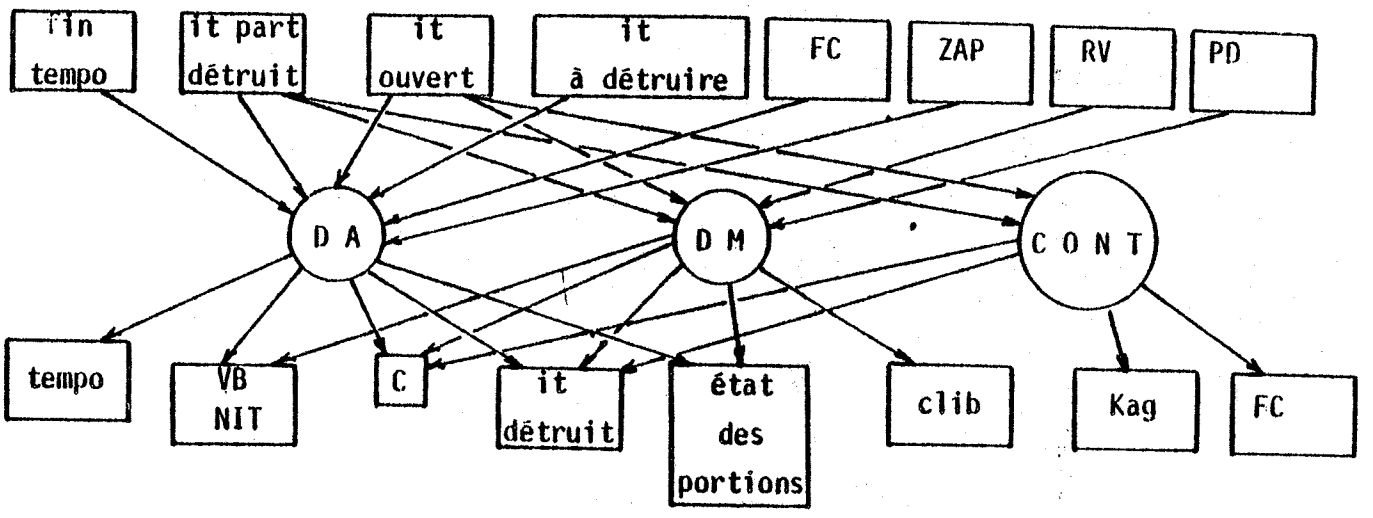


DIAGRAMME 2

On voit apparaître sur ces deux diagrammes un partage important des variables entre les différents sous-systèmes.

Pour chaque sous-système, il s'agit maintenant de décrire précisément les relations entrées-sorties. Ceci se fait comme précisé en II.2.2.

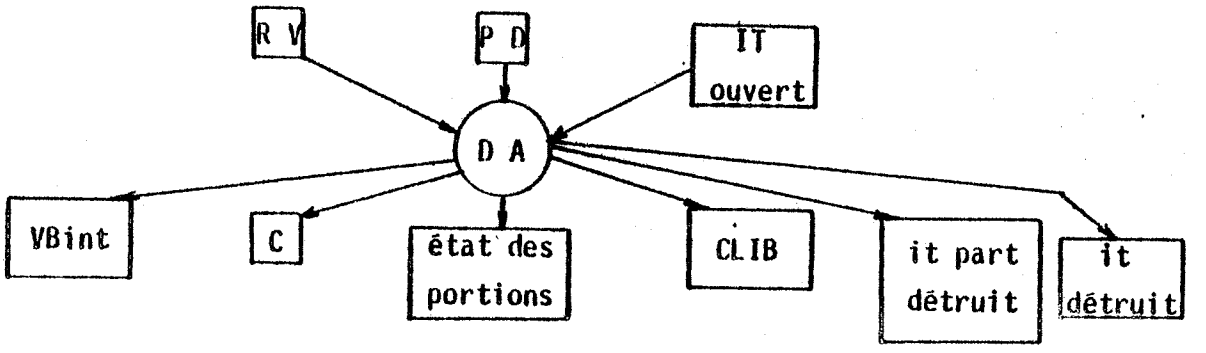
Chaque sous-système est défini par un ensemble de relations (ici de la forme $S = f(E)$).

Ces relations sont décrites par :

- une condition d'activation de f décrivant les relations temporelles entre S et E.
- la relation fonctionnelle liant S et E.

Nous allons donner ici un exemple de description des sous-systèmes :

- sous-système DA



Ce sous-système peut se décrire à l'aide de trois relations :

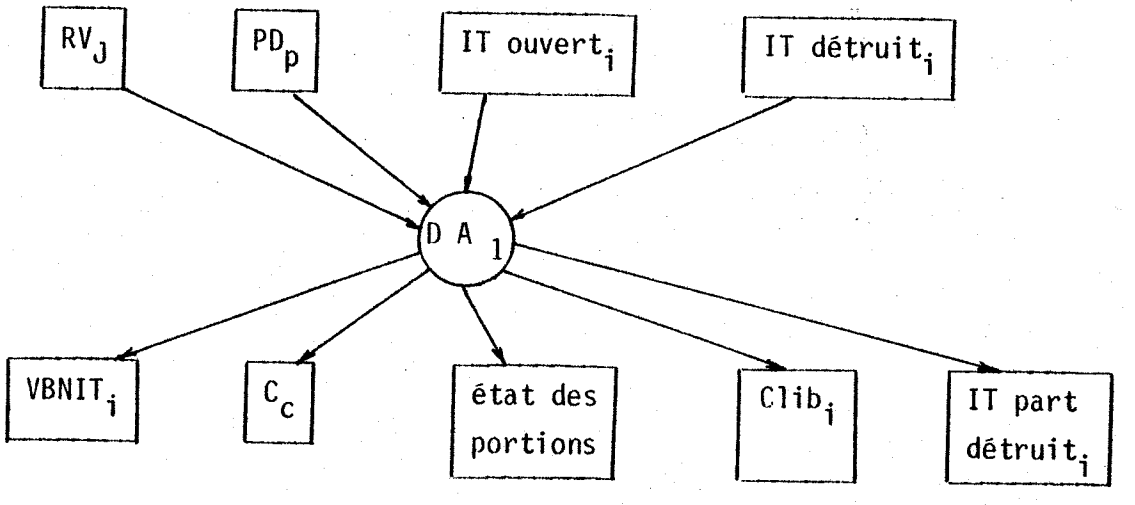
DA1 : Destruction automatique des premières zones

DA2 : Destruction automatique des dernières zones

DA3 : Destruction automatique des zones intermédiaires.

Chaque relation étant paramétrée par l'itinéraire IT_i.

A) Spécification de la relation DA_1 pour un itinéraire IT_i.



où $RV_J = RV(PREM(IT_i))$

RV_J est la variable associée à la première zone de l'itinéraire.

PD_p est la pédale de destruction automatique de l'itinéraire $PD_p = PED(IT_i)$

$C_c = CAR(IT_i)$ carré associé à l'itinéraire i.

. Expression de l'activation

chaque variable de sortie ($VBNIT_i$, C_c, état des portions, $clib_i$, it part détruit$_i$) sera calculée à l'arrivée de l'événement

$$RV_J\uparrow/ATT_i$$

RV_J étant l'événement associé au passage de 1 à 0 de la variable RV_J.

$ATT_i(t)$ est une condition : l'itinéraire IT_i est ouvert en attente de destruction automatique ou manuelle à l'instant t. $ATT_i(t)$ s'exprime en compteur de la façon suivante :

$Att_i(t) = [\mu_{it\ ouvert_i}{}^{\uparrow}(t) - \mu_{DMI_i\ it\ détruit_i}{}^{\uparrow}(t) - \mu_{DMFT_i\ it\ détruit_i}{}^{\uparrow}(t)$

$- \mu_{it\ part\ détruit_i}{}^{\uparrow}(t) > 0]$

c'est-à-dire ATT(i) est égal à 1 lorsque le compteur de l'événement "itinéraire (i) est devenu ouvert", est supérieur à la somme des compteurs des événements, "l'itinéraire a été détruit manuellement avec temporisation, sans temporisation, détruit automatiquement".

. Valeur de chaque sortie

(on note $DA_{1_i}\ X(p)$ la valeur de X calculée après la $p^{ième}$ occurrence de l'événement d'activation de DA_{1i}), et on note X la valeur de X à l'instant d'arrivée de l'événement. On a :

$DA_{1i} \cdot VBNIT_i\ (P_1) = [PD_p = 0]\ VBNIT_i$

$DA_{1i} \cdot VBNIT_i\ (P_1) = [PD_p = 0]$

$DA_{1i}\ C_c\ (P_1) = C_c\ [\overline{PD_p = 1}]$

$DA_{1i}\ CLIB_i\ ((PREM(IT_i))\ (P_1) = Clib_i\ (Prem(IT_i) + [PD_p = 1]$

Ceci traduit :

si la pédale PDP est enfoncée, alors le bouton est éteint.

L'itinéraire est partiellement détruit, le carré mis au rouge, le compteur de libération de la première zone de l'itinéraire est incrémenté.

Il reste à modifier "état des portions".

$DA_{1_i}\ \ Réservée_k\ (P_1) = Réservée_k - [(P_k \in Prem\ IT(i))_\wedge (Pos_i(P_k) \neq 3)_\wedge (PD_p = 1)]$

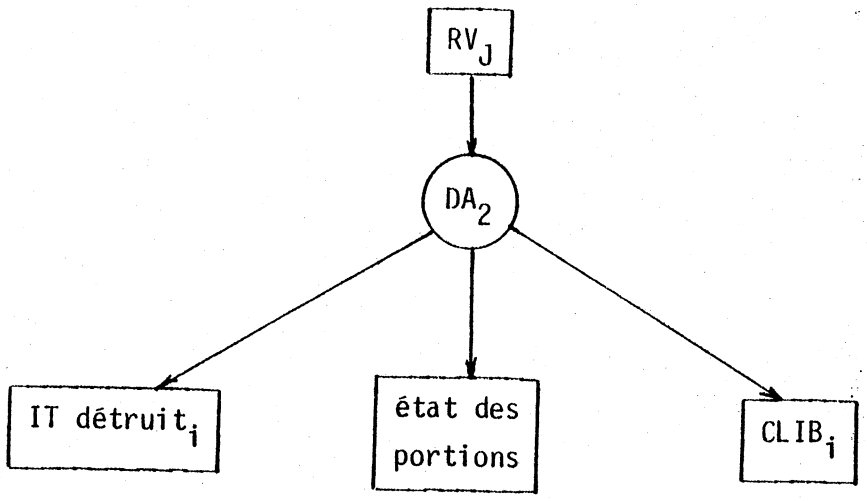
$DA_{1_i}\ \ Bloquée_k\ (P_1) = Bloquée_k \cdot [(\overline{P_k \in Prem\ IT(i))_\wedge (Pos_i\ P_k \neq 3)}$

$\wedge \overline{PD_p=1}_\wedge \overline{Réservée_k = 1}]$

Ce qui exprime que toute portion appartenant à la première zone est libérée et débloquée si elle n'est pas réservée par un autre itinéraire.

DA_{1i} est ainsi complètement explicité.

B) Spécification de la relation DA_2 pour un itinéraire IT_i donné



RV_J est la variable associée à la dernière zone de l'itinéraire IT_i.

. Expression de l'activation

Chaque variable de sortie sera réaffectée pour l'événement

$RV_J\downarrow$ passage de la variable RV_J de 1 à 0

Valeur des variables en sortie

Avec les mêmes notations que pour DA_1 on a :

DA_{2_i} . $IT\ détruit_i(P) = [Clib_i\ (Der(IT_i)) < Clib_i(Pred_i(Der(IT_i))]$

$DA_{2i}\ Clib_i(Der\ IT_i(P) = Clib_i\ (Der\ IT_i) + DA_{2_i}\ IT\ détruit_i(P)$

Le compteur de libération est incrémenté et l'itinéraire est détruit si le nombre de libérations de l'avant-dernière zone est supérieur au nombre de libérations de la dernière zone.

De plus, "état des portions" est modifié de la façon suivante :

$$DA_{2i}\ Bloquée_k(P) = Bloquée_k\ [Réservée_k = 1_\wedge(P_k \in Der\ IT_i)_\wedge Pos_i(P_k) \neq 3 \wedge(Da_{2i}\ IT\ détruit_i(P) = 1)]$$

$$DA_{2i}\ Réservée_k(P) = Réservée_k - [(P_k \in Der\ IT_i)_\wedge(Pos_i(P_k) \neq 3) \wedge (DA_{2i}\ IT\ détruit_i(P) = 1)]$$

Ceci indique que toute portion est libérée et débloquée si elle n'est pas réservée par un autre itinéraire.

La spécification de la relation DA_3 (destruction automatique de zone intermédiaire) est quasi semblable à DA_2.

- La spécification du PAI sous cette forme représente une quarantaine de pages et semble donc assez lourde à lire. En contre-partie, cela nous a permis de lever un certain nombre d'ambiguités et d'apporter des précisions.

VI.3 - SPÉCIFICATIONS ALGORITHMIQUES

Il s'agit ici de décrire une machine abstraite réalisant le PAI, puis de trouver le comportement approché admissible.

Nous donnons ici à titre d'exemple le GRAFCET réalisant la destruction d'itinéraire (sous système DA et DM).

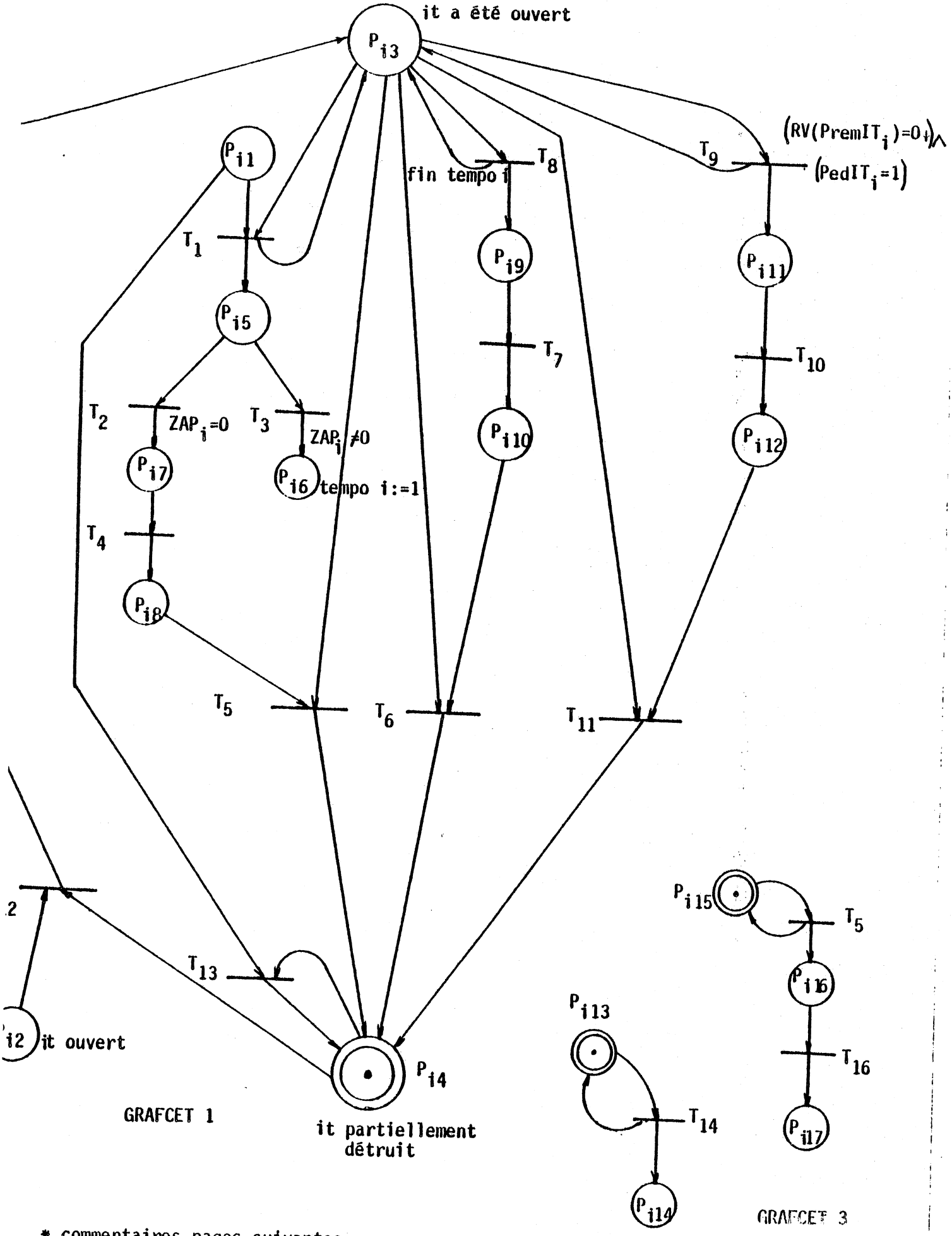
Le point de départ du GRAFCET représentant la destruction d'itinéraire, est l'existence des étapes P_{i1} et P_{i2}

P_{i1} est active lorsque l'itinéraire i est à détruire (ordre manuel de destruction).

P_{i2} est active lorsque l'itinéraire i est ouvert (carré ouvert).

Ces deux étapes sont activées par d'autres sous grafcet.

Partant de ces deux étapes, on peut décrire la destruction d'itinéraire à l'aide des Grafcets suivants :



* commentaires pages suivantes

- les étapes P_{i_3} et P_{i_4} ne peuvent être actives à tour de rôle.
- P_{i_3} active signifie qu'un itinéraire a été ouvert
- P_{i_4} active signifie soit que l'itinéraire n'a jamais été ouvert,
  soit que l'itinéraire a commencé à être détruit
  (représenté par une étape active en P_{i_8} ou $P_{i_{10}}$ ou $P_{i_{12}}$).

- la branche 1 P_{i1}, P_{i5}, P_{i6}, P_{i7}, P_{i8} du Grafcet 1 correspond au traitement de la demande de destruction manuelle.

T_1 est une transition associée à l'événement toujours vrai et c = 1

P_{i_5} n'a pas d'action associée

T_2 est une transition associée à l'événement toujours vrai et a pour condition :

Zapproche(IT(i)) = 0.

T_3 est une transition associée à l'événement toujours vrai et a pour condition

Zapproche(IT(i)) ≠ 0.

P_{i_6} a pour action [$Tempo_i$:= 1]

P_{i_7} a pour action

V_{BNIT_i} := 0

C_i := 0

pour k : 1 jusqu'à nbp faire

si $Pos_i(P_k)$ ≠ 3 alors | $Réservée_k$:= $Réservée_k$ - 1

| Si $réservée_k$ = 0

| alors $bloquée_k$ = 0

La branche 2 P_{i9}, P_{i10} du Grafcet 1 correspond au traitement de la fin de temporisation déclenché par P_{i6}.

T_6 est associée à l'évènement fin de temporisation.

P_{i_9} a pour action la même procédure que P_{i_7}.

La branche 3 P_{i11}, P_{i12} du Grafcet 1 décrit la destruction automatique de la zone.

T_9 a pour événement associé "RV(Prem|IT(i)) = 0↓ et pour condition Ped IT(i) = 1.

P_{i9} a pour action associée :

- $V_{BNIT_i} := 0$
- $C_c := 0$ où $C_c = car(it_i)$
- $Clib_i(Prem(IT_i)) := Clib_i(PremIT(i)) + 1$
- Pour k := 1 jusqu'à nbp faire
  - Si ($P_k \in Prem(IT_i)$) et ($Pos_i\ P_k \neq 3$) alors
    - $Réservée_k := Réservée_k - 1$
    - Si $Réservée_k = 0$ alors $bloquée_k := 0$

Le Grafcet 2 représente la destruction des zones intermédiaires.

T_{14} a pour événement $\underset{J \in L}{U} (RV_J = 0↓)$

où L est l'ensemble j énumérant l'ensemble des variables RV_J associé à l'itinéraire (à l'exception de la première et dernière zone).

La condition de T_{14} est $Clib[(Z_j) < Clib(pred\ Z_J)]$

- l'action associée à $P_{i_{14}}$ est
  - $Clib_i(Z_J) := Clib_i(Z_J) + 1$
  - pour k := 1 jusqu'à nbp faire
    - Si($P_k \in Z_J \wedge Pos_i(P_k) \neq 3$) alors
      - $Réservée_k := Réservée_k - 1$
      - Si $Réservée_k = 0$ alors $Bloquée_k := 0$

Le Grafcet 3 représente la destruction automatique de la dernière zone :

T_{15} a pour événement associé

RV(Der IT_i) = 0↓ indiquant le passage à 0 de la variable associée à la dernière zone.

La condition de T_{15} est semblable à celle de T_{14}.

L'action associée à $P_{i_{16}}$ est semblable à celle de $P_{i_{14}}$.

Les réceptivités des transitions non décrites sont en fait celles associées à l'évènement toujours vrai et à la condition égale à 1.

- Les étapes dont les actions n'ont pas été décrites sont des étapes n'ayant pas d'action.

Les étapes $P_{i_{14}}$ et $P_{i_{17}}$ sont nécessaires pour d'autres grafcets.

Définition du comportement approché admissible du Grafcet

Le grafcet de l'ensemble du PAI est en fait une machine abstraite parfaite.

Un comportement approché à (ε, Δ) près avec

$$\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n) \text{ où } \varepsilon_i = \frac{1}{100} \text{ sec. pour toutes les entrées}$$

$$\Delta = (\Delta_1, \Delta_2, \ldots, \Delta_n) \text{ où } \Delta_j = \frac{1}{100} \text{ sec. pour toutes les sorties}$$

est admissible.

En effet, $\Delta_J = \frac{1}{100}$ sec. temps de retard des sorties est tout à fait admissible compte-tenu du temps de réponse des appareils de voies, etc...

$\varepsilon_i = \frac{1}{100}$ sec. est admissible car :

- aucune entrée n'a une telle période,
- lorsque l'ordre de deux entrées différentes ont une importance, ces entrées sont séparées de plus d'$\frac{1}{100}$ sec.

VI.4 - IMPLANTATION DU PAI

Le PAI étant décrit sous forme GRAFCET, il suffirait ensuite d'utiliser l'interpréteur GRAFCET fidèle. Le matériel sur lequel est implanté l'interpréteur ne répond pas aux critères de sûreté de fonctionnement (fiabilité, sécurité). Un matériel étant en cours de définition pour répondre à ces critères, l'implantation de l'interpréteur GRAFCET fidèle est une voie pour permettre une implantation facile du logiciel sur ce matériel.

VI.5 - CONCLUSION SUR L'ÉTUDE DU PAI

Cette étude nous a montré que la démarche proposée permettait la levée d'ambiguités du cahier des charges. On peut noter que l'étape 1 (Spécification comportementale) est très longue mais facilite grandement l'étape 2 (Spécification algorithmique). L'étape 3 semble ensuite relativement facile. D'autres exemples sont en cours de traitement et semble valider ces conclusions.

CONCLUSION

CONCLUSION

La démarche que nous proposons repose sur un certain nombres de points fondamentaux :

1) La nécessité de spécifier rigoureusement le système temps réel sous forme "mathématique".
2) La nécessité de décrire non seulement une solution abstraite au problème mais aussi de définir un ensemble de solutions admissibles, ensemble défini à partir d'une solution abstraite et de la notion de comportement approché.
3) La nécessité d'implanter rigoureusement une solution définie en 2 ; ceci ne peut se faire d'une façon automatique que pour une implantation fidèle. Pour une implantation sûre, une étude cas par cas est nécessaire.

Enfin il est à souligner qu'un certain nombre d'outils sont soit développés (interpréteur GRAFCET fidèle) soit en cours de développement (Simulateur) permettant l'application pratique de cette méthode.

Cette démarche nous semble pouvoir être améliorée. En effet, elle présente une certaine faiblesse pour la validation des spécifications algorithmiques (simulation) il est évident qu'une validation analytique est nécessaire si on veut une démarche "totalement sûre".

Dans cette perspective, il nous paraît intéressant d'étudier la possibilité de décrire les spécifications algorithmiques à l'aide du même modèle que les spécifications comportementales puis d'étudier les possibilités de validation analytique.

[1] BARBACCI M.R
A comparison of register transfer languages for describing computers and digital systems. IEEE Trans. on computers, Vol. C.24 n°2, Fev.1975.

[2] DAHL 72
Dahl O.J., Dijkstra E.W., Hoare C.A.R.
Structured Programming. Academic Press, New York 1972.

[3] DIJKSTRA 76
Dijkstra E.W. A discipline of programming.
Prentice Hall series in Automatic Computation. New Jersey 1976.

[4] ARSAC 77
Arsac J. La construction de programmes structurés.
Dunod, Paris 1977.

[5] McLAREN 80
McLaren L. Evolving toward Ada in real time systems.
Proceeding of the ACM Sigplan Symposium on the ADA Programming Language. Dec. 9-11 1980.

[6] GIRARD 78
Girard B., Michel G. Les langages évolués pour le temps réel.
Nouvel Automatisme, Janvier-Février 1978, pp.35-41.

[7] CASPI 82
Caspi P., Halbwachs N., An approach to real time systems modeling
3e Int. Conf. on Distributed Computing Systems. Miami, Oct. 1982.

[8] CASPI 82
Caspi P., Halbwachs N., "Algebra of events : a model for parallel processing, Bellaire (Michigan) Aout 1982.

[9] MOALLA 81
Moalla M. Spécification et conception sûre d'automatismes discrets complexes, basées sur l'utilisation du GRAFCET et des réseaux de Petri. Thèse d'Etat INP Grenoble, juillet 1981.

[10] PETERSON 77
Peterson J.L. Petri Nets.
ACM Computer Surveys, Vol.9, n°3 Sept.1977, pp.223-251.

[11] MOALLA 78
Moalla M., Pulou J., Sifakis J. Synchronised Petri nets : a model for the description of non autonomous systems.
Mathematical foundations of computer sciences, Ed. Springer Verlag 1978, pp.374-383.

[12] Fortran temps réel. Draft Standard Industrial Real Time Fortran. ACM Sigplan Notices, Vol.16 n°7, juillet 1981.

[13] MOALLA 81
Moalla M., David R. Extension du Grafcet pour la représentation de systèmes temps réels complexes.
RAIRO Informatique. Vol.15, n°2 1981.

[14] PROCOL. Langage PROCOL. Manuel de référence Stéria.

[15] ADA. Evaluation du langage ADA. Groupe ADA de l'Afcet, version de travail n°1, Octobre 1981.

LE CALVEZ 79
[16] LE CALVEZ LISCH F. Gaellic Définition d'un langage de description globale des applications temps réel. Thèse de 3e cycle.
Université de Paris 7, 1979.

[17] CASPI 82
Caspi P., Halbwachs N., Pilaud D. Implémentation fidèle de langage temps réel.
Rapport Interne IMAG n°315, Septembre 1982.

[18] DAVID 82
David R., Deneux H. About the determinisin of safe interpreted Petri nets. Relations with the Grafcet model. Note LAG n°8223.

[19] DHALL 77
Dhall S.K. Scheduling Periodic-Time critical jobs on single processor and multi processor computing system.
Thesis of Department of Computer Science; University of Illinois, Urbana, 1977.

[20] LIU.CL LAYLAND 73
Liu.CL Layland J.W. Scheduling algorithms for multiprogramming in a hard real time environment (JACM 20-1 Janv. 1973).

[21] LABETOULLE 74
Labetoulle J. Ordonnancement des processus temps réel sur une ressource préemptive.
Thèse de 3e cycle, Université Paris VI, 1974.

[22] ROBACH 80
Robach Ch. Test et Testabilité de système informatiques.
Thèse d'Etat, INPG, 1980.

[23] BOURDON 82
Bourdon M. Une approche unifiée des problèmes d'ordonnancement statique discret.
DEA Informatique, INPG, Juin 1982.

[24] SLOWINSKI 81
Slowinski. L'ordonnancement des tâches préemptives sur des processeurs indépendants en présence de ressources supplémentaires.
RAIRO Informatique, 15-2, 1981.

[25] Tani MURATA 78
Tani Murata. Scheduling parallel computations with storage constraints.
12th Asilomar Conference on circuits systems and computers, IEEE, Novembre 1978.

[26] LAFFORGUE 82
Lafforgue S., Millioni F. Spécification d'un poste d'aiguillage informatisé.
Rapport de DEA Informatique INPG, Sept. 1982.

[27] Documents internes SNCF sur le PRS et PAI

[28] BERRY 82
Berry and all. Quelques primitives pour la programmation temps réel et leur sémantique mathématique.
Proceeding of real time, Data 82, Novembre 1982.

[29] MILNER 80
Milner R. A calculus of communicating systems.
Lecture notes in Computer Science n°92, Springer Verlag, 1980.

[30] JORRAND 81
Jorrand Ph. Sur les spécifications des processus comminucants : quelques bases pour l'analyse et la vérification.
Journée d'étude sur les protocoles et les systèmes distribués, Institut de Programmation, Paris, Juin 1981

[31] MILNER 82
Milner R. Calculi for synchrony and asynchrony.
Research report, Edimburgh University, Fevrier 1982.

[32] HENINGER 79
Heninger K.L. Specifying software requirements for complex systems : new techniques and their applications.
Conf. on Specifications of reliable software, IEEE n°79 CH 1401 9C, 1979.

[33] HOARE 69
Hoare C.A.R. An axiomatic basis of computer programming.
CACM, Vol. 12, n°10, Octobre 1969.

[34] ABRIAL 78
Abrial J.R. Z : a specification language.
IFIP, Tokyo, 1978.

[35] CAPLAIN 78
Caplain M. Langage de spécification.
Thèse d'état, Université de Grenoble, 1978.

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974,

VU les rapports de présentation de

. Madame G. SAUCIER, Professeur
. Monsieur R. DAVID, Maître de recherche

Monsieur PILAUD Daniel

est autorisé à présenter une thèse en soutenance pour l'obtention du titre de DOCTEUR de TROISIEME CYCLE, spécialité "Génie informatique".

Fait à Grenoble, le 12 novembre 1982

Le Président de l'I.N.P.-G.

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,