UNIVERSITE DE NANCY 1

UER SCIENCES MATHEMATIQUES

CONCEPTS ET OUTILS POUR L'ECRITURE DE MAQUETTES DANS LE PROJET MAESTRO

THESE

POUR L'OBTENTION DU

DOCTORAT DE TROISIEME CYCLE

EN INFORMATIQUE

SOUTENUE LE 22 MAI 1979 PAR

PHILIPPE KLEIN

UNIVERSITE DE NANCY 1
UER SCIENCES MATHEMATIQUES

CONCEPTS ET OUTILS POUR L'ECRITURE DE MAQUETTES DANS LE PROJET MAESTRO

THESE

POUR L'OBTENTION DU
DOCTORAT DE TROISIEME CYCLE
EN INFORMATIQUE

SOUTENUE LE 22 MAI 1979 PAR
PHILIPPE KLEIN

JURY :
PRESIDENT : M. C. PAIR
EXAMINATEURS : MM JC. DERNIAME
JF. DUFOURD
J. MAROLDT
MME C. ROLLAND

Je remercie M. le Professeur PAIR, président de l'INPL, de me faire le grand honneur de présider ce jury.

Je remercie vivement Madame ROLLAND pour la formation initiale à la recherche qu'elle m'a prodiguée, pour l'intérêt qu'elle a témoigné envers ce travail et pour l'honneur qu'elle me fait de participer au jury.

Que Monsieur J.C. DERNIAME trouve ici l'expression de ma gratitude pour la formation en informatique que je lui dois et pour sa participation au jury.

Mes remerciements vont aussi à Monsieur J. MAROLDT pour la sympathie et l'aide matérielle dont il a fait preuve en me confiant des enseignements et pour avoir accepté d'être membre du jury.

Je tiens à remercier tout particulièrement Monsieur J.F. DUFOURD qui est à l'origine du projet MAESTRO dans lequel s'insère ce travail. Je lui suis reconnaissant d'avoir su, par ses conseils et encouragements me faire progresser dans l'élaboration de cette thèse. Je le remercie d'avoir accepté de faire partie du jury.

Que les chercheurs de l'équipe MAESTRO soient ici remerciés pour leur amitié et leur entr'aide.

Je désire enfin remercier Mesdames LEFEVRE et MOREAU, l'administration du département GEA de l'IUT de Nancy ainsi que Madame MARCHAND, sans qui la réalisation matérielle de cette thèse n'aurait pas été possible.

SOMMAIRE

PLAN GENERAL

CHAPITRE I - INTRODUCTION

PLAN DETAILLE

CHAPITRE II - ARCHITECTURE GENERALE D'UNE MAQUETTE

PLAN DETAILLE

CHAPITRE III - LANGAGES ET OUTILS DE DESCRIPTION DE MAQUETTES

PLAN DETAILLE

III.1 - Langages d'écriture de maquettes
- III.1.1 - Choix du langage
- III.1.2 - Présentation de SIMULA

III.2 - Outils de synchronisation

III.3 - Réalisation de certains outils en SIMULA

III.4 - Conclusion

CHAPITRE IV - DETAIL DES COMPOSANTS

PLAN DETAILLE

IV.1 - Outils généraux
- IV.1.1 - Outils de synchronisation
- IV.1.2 - Procédures de service

IV.2 - Aspects logiques
- IV.2.1 - Les structures de données
- IV.2.2 - Les processus synchronisés
- IV.2.3 - Prise en compte du temps

IV.3 - Aspects physiques
- IV.3.1 - Les ressources
- IV.3.2 - Utilisation des calendriers

CHAPITRE V - ECRITURE D'UNE MAQUETTE

PLAN DETAILLE

V.1 - Méthode pour l'écriture de maquettes
- V.1.1 - Les outils généraux
- V.1.2 - Aspects logiques
- V.1.3 - Aspects physiques
- V.1.4 - Exécution d'une simulation

V.2 - Application - Exemple de maquette
- V.2.1 - Le choix du système
- V.2.2 - Architecture de la maquette
- V.2.3 - Ecriture de la maquette
- V.2.4 - Conclusion

CHAPITRE VI - MESURES SUR LES MAQUETTES ET EXPLOITATION DES RESULTATS

PLAN DETAILLE

VI.1 - Objectifs des mesures

VI.2 - Outils de mesures
- VI.2.1 - Les régions
- VI.2.2 - Les files
- VI.2.3 - Les ressources
- VI.2.4 - Principe d'utilisation

VI.3 - Mesures effectuées sur la maquette
- VI.3.1 - Mesures sur les files
- VI.3.2 - Mesures sur les processeurs
- VI.3.3 - Mesures sur les régions

VI.4 - Conclusion

CHAPITRE VII - CONCLUSION

CHAPITRE VIII - BIBLIOGRAPHIE

CHAPITRE IX - ANNEXES

CHAPITRE I

INTRODUCTION

CHAPITRE I -

INTRODUCTION

Notre travail s'inscrit dans le cadre du projet MAESTRO (Maquettes pour l'évaluation de systèmes d'information d'organisations). Aussi, nous ferons d'abord une présentation des grandes lignes de ce projet, afin de mieux mettre en évidence les caractéristiques propres de notre travail.

I.1 - PRESENTATION DU PROJET MAESTRO

L'objectif de ce projet est d'aider à mieux comprendre les systèmes d'information d'organisations par une amélioration des techniques d'analyse. Ces systèmes sont représentés au moyen de maquettes (modèles réduits), qui sont utilisées ensuite dans des sessions de simulation afin d'éprouver la correction fonctionnelle ou le comportement dynamique des systèmes décrits.

Avant d'étudier le projet proprement dit, il est nécessaire de rappeler quelques définitions simples pour permettre une bonne compréhension de notre travail : celles de système, modèle, simulation, d'une manière générale, et dans le contexte de l'informatique d'organisation.

I.1.1 - DEFINITIONS GENERALES

a) Les systèmes

De nombreux auteurs ont proposé une définition des systèmes. Par exemple :

"Un système est une combinaison de parties qui se coordonnent pour concourir à un résultat ou de manière à former un ensemble".

Nous intéressant essentiellement aux aspects structurels d'un système, nous adoptons plutôt la définition suivante, adaptée de celle de FISHMAN (FIS73) :

"Un système est un ensemble d'entités reliées, chacune caractérisée par des attributs qui peuvent eux-mêmes être reliés".

Certaines de ces interrelations (ou "fonctions") sont définies indépendamment du temps et sont dites statiques. D'autres, dont la définition dépend explicitement ou implicitement du temps, sont dites dynamiques. Collectivement, les attributs d'une entité définissent son état. L'état du système est défini par les états de certaines entités choisies, appelées : entités critiques. Un système peut, au cours du temps, passer par différents états. Si les changements d'états s'opèrent de manière discontinue, on les appelle événements. L'environnement d'un système est défini comme tout ce qui n'appartient pas au système (extérieur).

A partir de ces définitions, il est possible de donner une classification des systèmes :

- il existe des systèmes naturels et des systèmes fabriqués ;
- quand un système ne peut exister que dans un environnement particulier, il est dit ouvert, par contraste avec les systèmes clos qui peuvent exister dans n'importe quel environnement.

Pour étudier un système, il faut le délimiter, définir un domaine d'observation dans la réalité. Les objectifs de l'étude des systèmes sont essentiellement de comprendre sa structure, son évolution en étudiant ses changements d'états ; ceci aux fins de contrôle et éventuellement de prédiction.

La première étape de l'étude d'un système est souvent la construction d'un modèle.

b) Les modèles

"Un modèle est une représentation formelle de la théorie, ou un exposé formel d'observation empirique" (FIS73). Plus simplement, on considère un modèle comme une représentation simplifiée d'un système, qui doit fonctionner sur les mêmes principes que l'original en vraie grandeur (LAR79).

De même que pour les systèmes, on peut proposer une classification des modèles.

Dans son ouvrage (FOR61), FORRESTER propose une classification dont nous rappelons les grandes lignes.

- On distingue les modèles physiques (répliques à échelle réduite ; par exemple les maquettes de bateau) des modèles abstraits (les modèles mathématiques en font partie), et des modèles schématiques (diagrammes, cartes, etc...).
- On peut distinguer deux grandes classes de modèles mathématiques : dans un modèle analytique il est possible de déduire la solution du problème directement par sa représentation mathématique : par exemple la loi d'Ohm ; un modèle numérique se rapproche plutôt du procédé de calcul, ou algorithme, qui fournit un résultat pour une donnée précise : un exemple est l'intégration numérique.
- On distingue les modèles statiques et les modèles dynamiques. Dans un modèle statique, le temps n'intervient pas, ou le modèle est un instantané donnant la représentation de l'état du système à un instant donné. Par opposition, le temps intervient, explicitement ou implicitement, dans un modèle dynamique. Un exemple de modèle statique, pris dans les modèles mathématiques est la loi d'Ohm ; la relation fondamentale de la mécanique $F = m\Gamma$ est un modèle mathématique dynamique.

- Un autre critère de classification des modèles mathématiques concerne leurs aspects déterministes ou stochastiques. Les modèles déterministes n'utilisent que des relations certaines : par exemple la loi de Mariotte. Dans les modèles stochastiques, le hasard intervient au travers de lois de probabilités : par exemple les chaînes de Markov.

Il existe d'autres critères de classification des modèles ; nous n'avons rappelé, ici, que les principaux.

Très souvent, le besoin de reproduire la réalité du fonctionnement d'un système rend impossible la résolution analytique d'un modèle. Une approche possible de ces problèmes est leur résolution par simulation.

c) La simulation de systèmes

On entend par simulation de systèmes, le fait de représenter un système par un modèle symbolique, aisément manipulable et capable de produire des résultats numériques.

A l'extrême on pourrait utiliser le système lui-même comme modèle ; ainsi on ne perd aucune information quand au comportement. Cette approche, simulation par identité est lourde, rarement réalisable, et ne permet pas d'obtenir des résultats avec un délai de réponse suffisamment court.

Une autre manière de procéder est de garder le maximum d'aspects du système réel en excluant, toutefois, les éléments dont la présence rendrait une simulation par identité impossible. Cette méthode, appelée simulation par quasi-identité est également lourde et ne garantit pas une adéquation certaine au problème posé par l'étude du système réel.

Une simulation dite de laboratoire, plus facilement réalisable et moins coûteuse que les deux méthodes ci-dessus, consiste à ne retenir du système que ses caractéristiques essentielles. Les composants d'une telle simulation sont aussi divers que les personnes, les machines, les procédures opérationnelles, les fonctions mathématiques et les distributions de probabilités. FISHMAN considère dans (FIS73) deux types de simulations de laboratoire :

- les jeux opérationnels dans lesquels la machine est utilisée pour collecter, traiter et produire des informations, que les "joueurs" utilisent pour prendre des décisions ; la machine ayant, elle-même, la possibilité de jouer un rôle actif dans ces prises de décisions.
- les simulations homme-machine apportent un élément de plus dans cette méthode. Ici certaines personnes du laboratoire, parallèlement au travail de la machine, procèdent à la réduction des données et à leur analyse.

L'intérêt de cette méthode est, en général, la possibilité d'améliorer les systèmes existants grâce à la simulation. (Tests de nouvelles procédures, impacts de modifications, etc...).

Enfin, l'approche semblant la plus satisfaisante dans le domaine des simulations de systèmes, est la simulation par ordinateur uniquement (FIS73).

Si on exclut les personnes et les aspects technologiques des concepts de la simulation de laboratoire, et que l'on garde l'ordinateur, les règles opérationnelles, les fonctions mathématiques et les distributions de probabilités, on obtient les caractéristiques essentielles d'une simulation basée uniquement sur l'odinateur. Plusieurs avantages sont à l'actif de l'utilisation d'une telle méthode :

- il est possible de comprimer le temps, et ainsi de simuler en quelques minutes, ou quelques secondes, plusieurs années d'activités.

- à l'opposé, il est possible d'étendre le temps, ce qui peut être très pratique au cas où des changements d'états d'un système réel se produisent en grand nombre dans un laps de temps très court ; et ainsi sont difficilement observables sur ce système réel.

- un des aspects les plus importants d'une simulation est l'étude et le contrôle des sources de variation. En particulier, lorsqu'on procède à des mesures statistiques sur les corrélations entre les entrées et les sorties d'un système, on doit tenir expressément compte des sources de variations des informations dans le système. Quand on met au point une simulation sur ordinateur, il est indispensable d'y faire figurer toutes les sources de variations ainsi que leur degré de variation propre. Un avantage de cette méthode est donc la sécurité d'avoir un système complet car elle empêche l'oubli volontaire ou involontaire de sources de variations.

- les erreurs de mesures, en général dues à de mauvaises interprétations du réel, sont évitées grâce à cette méthode automatisée qui produit des résultats exempts de variations dues à des éléments (extérieurs) incontrôlables.

- l'utilisation d'ordinateurs en simulation permet l'enregistrement de nombreuses données, en particulier d'états du système, permettant d'en conserver une trace.

- par analyse des résultats obtenus, il est possible de relancer une simulation avec de nouveaux paramètres (techniques d'affinage successif).

En conclusion, on peut remarquer qu'une simulation par ordinateur devient intéressante à utiliser quand les méthodes analytiques connues ne peuvent mener à une solution satisfaisante. Cette méthode permet, en outre, de tester et d'évaluer de nouveaux systèmes, de proposer d'éventuels chan-

gements et améliorations aux systèmes existants sans investir d'énormes efforts, ni de gros moyens financiers, et sans influencer leur développement physique réel.

Toutefois, la plupart de ces méthodes de simulation sont basées sur l'idée de réduction des éléments du système. C'est là un des problèmes des plus difficiles à résoudre dans l'étude des systèmes. Nous reviendrons plus loin sur le problème du niveau de détail à considérer dans leur modélisation. Rappelons simplement que l'expérience montre qu'un luxe de détails trop important nuit souvent à l'étude des systèmes pour les raisons suivantes :

- il rend plus difficile l'étude des systèmes en noyant parmi des éléments secondaires, les éléments principaux ;

- il cache parfois des solutions intéressantes, par exemple analytiques, pour privilégier des solutions plus ponctuelles, par exemple, numériques ;

- il augmente la difficulté de résolution et les coûts. Le fait d'ajouter des détails revient à augmenter la programmation ; ce qui a pour effet de faire croître les coûts d'exécution.

En général, les utilisateurs commencent leur étude en expérimentant des modèles très grossiers ; ils ajoutent alors des détails tant que le modèle donne des résultats inadéquats. En tout état de cause, tout dépend du but que l'on poursuit. Un modèle n'est en général bon que pour réaliser des objectifs bien précisés.

Nous avons exposé dans ce début de chapitre, les principales notions en dégageant, pour chacune d'elles, les aspects les plus généraux. Ce qui suit montre comment les concepts de SYSTEME, MODELE et SIMULATION sont utilisés dans notre projet qui se situe dans le domaine de l'étude des systèmes d'information en organisations.

I.1.2 - DEFINITIONS APPLIQUEES AUX SYSTEMES D'INFORMATION

a) Les systèmes d'information en organisation

Un système d'information est un ensemble complexe d'informations et de moyens de traitement : collecte, mémorisation, transformation, restitution, communication d'informations dans l'organisation. On peut s'intéresser à tout le système d'une organisation, ou seulement à une partie que l'on appelle encore système d'information : les limites fixées sont arbitraires. On considère alors un système d'information comme un ensemble comprenant :

- tout ou partie d'un système informatique (système d'information automatique) ;
- une partie des procédures administratives de traitement (système d'information non automatique), plus ou moins liées au système automatique.

Nous verrons par la suite que l'on peut aussi étudier les systèmes d'information, d'une part d'un point de vue logique et d'un point de vue physique, d'autre part, d'un point de vue statique et d'un point de vue dynamique. Ainsi, d'une manière très générale, on peut dire qu'un système d'information (SI) est un système de processus coopérants (par l'intermédiaire de structures de données communes), et concurrents (pour l'obtention de ressources physiques). Nous retrouverons ces éléments ultérieurement quand nous discuterons l'adéquation de nos concepts et outils à la représentation du système d'information.

L'étude d'un ensemble aussi complexe qu'un système d'information est favorisée par une vue globale. C'est pourquoi il est bon d'étudier ce que sont des modèles généraux de tels systèmes.

b) Modèles de systèmes d'information

On peut simplement adopter la définition suivante :

- un modèle (ou une maquette) (*) de système d'information est une représentation à échelle réduite de ce système.

Les modèles de systèmes d'information que l'on rencontre sont schématiques ou symboliques, et ils visent des buts assez différents. Nous tentons de les illustrer maintenant en nous intéressant à certaines méthodes et techniques qui nous semblent caractéristiques des buts que l'on peut poursuivre en faisant des modèles.

α) Parmi les méthodes et les outils traditionnels permettant d'organiser la circulation de l'information dans les organisations, on peut citer d'abord des modèles graphiques ("schématiques" dans la classification) :

- les graphes de circulation d'information mettent en évidence des postes de travail et des transmissions de données entre ces postes, pour l'analyse ou la conception de SI. Ces graphes, très utiles aux organisateurs, ont fait l'objet de normalisations précises. Au départ très compliqué, le graphisme employé est devenu beaucoup plus simple dans les méthodes d'analyse actuelles (CORIG (MALLET (MAL71)), PROTEE (PRO70)), faites surtout pour l'informatique. Une étude de ce type de méthodes (par ex. (DUF76)) montre que, quelle que soit la qualité de la description adoptée, un certain nombre d'ambiguités ou d'imprécisions sont très difficiles à lever.

La principale remarque que l'on puisse faire sur les schémas de circulation est peut-être de proposer un modèle statique d'un système d'information. Seuls, des

---

(*) Remarque : On utilise ici les termes modèle et maquette avec la même signification.

commentaires peuvent apporter quelques indications quant au fonctionnement dynamique de ce système.

- les organigrammes : bien connus des informaticiens, ils ont fait l'objet d'études approfondies qui ont mis en évidence leurs avantages et leurs inconvénients pour l'écriture de programmes (cf. les schémas de programmes de LIVERCY (LIV78)). Mais ces schémas sont surtout utilisés, à l'heure actuelle, pour représenter, non plus au niveau d'un programme, mais de manière statique et globale, la partie automatique d'un système d'information. Ces modèles schématiques montrent que le plus souvent, la dynamique des systèmes n'est pas (ou peu) représentée dans les méthodes étudiées. Or, il est souhaitable de prendre en compte les aspects dynamiques dans une description de système car ils sont un élément essentiel à sa spécification et à sa compréhension. En effet, dans l'étude des systèmes existants, la représentation de ces aspects dynamiques dans des maquettes permet de détecter des vices de structure, d'évaluer le fonctionnement et le comportement dynamique de ces systèmes, c'est-à-dire de mieux les comprendre. Dans l'étude de systèmes nouveaux, elle aide à mieux spécifier ces systèmes, et, ultérieurement, à les construire et à les exploiter.

β) On pense alors naturellement au modèle de FORRESTER (FOR61), dont une des caractéristiques est de décrire la dynamique de toute une organisation. Rappelons simplement que le modèle de FORRESTER est un modèle mathématique qui se propose de mettre en équations les six flux qui traversent une entreprise : matériaux, argent, ordres, personnel, matériel, information (décision). Un tel "fluide" naît d'une "source" et disparaît dans un "puits". Entretemps, il a pu traverser des "réservoirs" où il a séjourné pendant une durée fonction de certains types de "délais".

Seul un léger degré d'indéterminisme peut être admis dans ce modèle : les flux d'entrée et les délais peuvent

être des processus aléatoires. Un tel modèle offre une vision de flux d'informations continus qui ne permettent pas de tenir compte des phénomènes essentiellement discrets du traitement de l'information. Il n'est pas possible d'exprimer le comportement particulier dû à un test ou à une décision de tel ou tel lot de données : tout cheminement d'information est figé.

En conclusion, ce modèle, orienté vers des études plutôt macroscopiques et à long terme de dynamique industrielle ou économique, ne semble pas répondre à nos objectifs de modélisation des systèmes d'information. (§ I.1.3)

δ) D'autres modèles peuvent être envisagés. En effet, depuis quelques années, on assiste à des propositions de modèles de systèmes d'information dont les buts sont surtout de chercher à spécifier et à concevoir un système avant de le réaliser. On peut citer le modèle MACSI (PEC75), ISDOS (TEI73) et son extension qui est actuellement à l'étude pour tenir compte des phénomènes de dynamique, dans l'esprit de ce que nous proposons dans notre travail (BOD78). D'autre part, des projets universitaires français, avec des préoccupations identiques, visent à mieux spécifier données et traitements d'un système d'information, en proposant des méthodes et des outils pour les concevoir, les réaliser, et les maintenir, citons : SCAPFACE (LUG75), GALION (FLO77), REMORA (ROL79).

c) La simulation de systèmes d'information

A notre connaissance, il n'existe que très peu de travaux visant à la simulation de systèmes d'information. On peut toutefois à nouveau citer les simulations continues de FORRESTER (FOR61) dont nous avons parlé précédemment.

Dans le projet MACSI (PEC75), on propose, entre

autres, une formalisation de la circulation de l'information dans une organisation. Le prolongement de ce projet, MACSI-P (GIR77), propose la réalisation de tels prototypes. Ceux-ci sont plutôt orientés vers l'étude de la sémantique des informations et de leur traitement, celle de la dynamique y étant prévue dans des simulations en vraie grandeur.

HURTUBISE dans (HUR76) développe la démarche SIG. Il s'agit d'une description des aspects sémantiques et dynamiques d'un système d'information de gestion. On s'intéresse aux temps d'exécution pour déduire les chemins critiques par la méthode PERT.

Après avoir tenté de faire sentir les objectifs que peuvent atteindre certaines méthodes proposées, nous allons maintenant préciser les buts du projet MAESTRO.

I.1.3 - OBJECTIFS DE LA MODELISATION DE SYSTEMES D'INFORMATION DU PROJET MAESTRO

Les buts que l'on peut assigner à une modélisation de systèmes d'information dans le projet MAESTRO sont de décrire, évaluer, prévoir.

a) Un modèle descriptif s'intéresse à la structure du système. Il tente de représenter les entités et les relations du système, éventuellement à différents niveaux. Utilisant un langage de programmation précis, codifié, l'écriture de maquettes, telle que nous l'envisageons, peut être assimilée à une description formelle. De ce point de vue, on a donc un modèle symbolique.

On peut disposer d'une description statique du système, mais il est souhaitable de prendre également en compte les aspects dynamiques en décrivant des modifications ordonnancées dans le temps, en tenant compte des phénomènes de parallélisme et de synchronisation des différentes

opérations. Ainsi on a un modèle dynamique. Comme nous l'avons vu, un système peut se décomposer en une partie automatique, et une partie non automatique (administrative). Les maquettes reflètent ces deux aspects, mais c'est pourtant d'après un autre aspect que nous scindons leur présentation en deux points de vue : logique et physique ;

- point de vue logique : on s'intéresse à la description de données et de processus ;
- point de vue physique : on examine l'implantation de ces processus dans l'organisation et l'usage des ressources physiques.

Cette description de système se fait, dans nos maquettes, avec un souci de décomposition modulaire en associant des composants standards diversifiés et des composants spécifiques au système décrit. Le langage de description évolué que nous avons choisi permet une utilisation, et même une extension aisée des composants déjà réalisés en fonction des particularités du système décrit. Nos maquettes peuvent servir à décrire des systèmes d'information :

- existants, afin d'obtenir sur eux une documentation claire, de les comparer à d'autres, et de faciliter leur évolution ;
- futurs, afin de les évaluer structurellement, de guider leur conception détaillée, leur réalisation et leur exploitation.

Le compilateur du langage vérifie la correction syntaxique de cette description.

b) Un modèle d'évaluation fonctionnelle aide à mette en évidence des vices de fonctionnement logique. Un mécanisme programmé réel peut être entièrement décrit dans une maquette, qui est ensuite éprouvée dans un environnement programmé bien choisi, en essayant de tenir compte de tous

les cas possibles. Ainsi, on peut tester, de manière qualitative, tel ou tel processus de modifications, alimenté par un générateur de données adéquat. Dans le cas d'une étude préalable d'un système, pour parvenir à un accord entre utilisateurs, concepteurs et informaticiens, sur les spécifications exactes du système à réaliser, on peut aussi être amené à illustrer la variété des informations saisies, traitées et restituées.

c) Un modèle d'évaluation quantitative aide à apprécier le comportement dynamique d'un système. En effet, on peut y décrire l'évolution du temps et l'utilisation des ressources physiques. En ce sens, un tel modèle est discret, numérique, dynamique, stochastique.

La méthode employée lorsque l'on fait de la simulation (ce qui est notre cas) est de faire des mesures sur certains composants du système éxécutés en quasi-parallélisme, grâce à une technique de simulation à événements discrets. Des caractéristiques du comportement du système sont déduites par analyse statistique des mesures ainsi recueillies. Les paramètres de fonctionnement doivent être, si possible, proches de la réalité (taux d'entrées, durée de travail des processeurs...).

Plusieurs types d'analyses peuvent être faites :

- on peut chercher les conditions d'obtention d'un équilibre du système. En particulier on vérifie que les files d'attente ne croissent pas démesurément, provoquant des "goulet d'étranglement". On supprime certains de ces effets néfastes grâce à des modifications du système.

- il est possible d'étudier l'évolution du système en régime transitoire afin, par exemple de détecter quand se produit un événement précis. Les problèmes statistiques inhérents à une telle étude, sont assez délicats. Généralement on est contraint de procéder à plusieurs simulations ou, si elle est possible, d'avoir recours à une méthode de points de régénération.

- un autre type de mesures est effectué en régime permanent. Un certain nombre de valeurs moyennes caractéristiques du système, sont alors obtenues avec un intervalle de confiance :

. des longueurs moyennes de files d'attente ;
. des taux moyens d'occupation de ressources ;
. ...

Dans ce cas, la simulation est le plus souvent unique, mais sur une période relativement longue. La valeur accordée aux résultats obtenus peut être absolue, ou relative. Dans le premier cas, on peut chiffrer assez précisément le coût du système (cf : audit informatique). Dans le deuxième cas, souvent dû à un manque d'informations sur les paramètres du système, on ne peut que procéder à des comparaisons avec des résultats obtenus sur d'autres modèles, de structure ou de paramètres différents ; on a alors un modèle explicatif.

La validation d'un modèle consiste à s'assurer qu'il est capable de rendre compte de situations (réelles ou fictives) que l'on connaît. L'ensemble des situations pour lesquelles ces vérifications ont été faites s'appelle le domaine de validation. Il peut être intéressant d'utiliser un modèle aux fins de prédiction. Cela consiste à tester le modèle dans des situations hypothétiques, différentes du domaine de validation, et suppose l'emploi d'un modèle robuste, c'est-à-dire capable de fournir des renseignements au-delà de son domaine de validation.

La modélisation peut, comme on vient de le voir, poursuivre des objectifs très divers et l'on pourrait penser avoir besoin de concepts différents selon le type d'objectifs, donc de modèles auxquels on s'intéresse, ou même selon le niveau de détail. Mais nous avons choisi l'unicité des concepts de base avec une diversification des éléments secondaires. De même, notre ambition est d'avoir un langage unique pour décrire les maquettes, nous laissant la possibilité d'utiliser des outils diversifiés, adaptés aux objectifs à satisfaire et écrits dans ce langage. Celui-ci doit permettre une vérification formelle des spécifications du modèle, une résolution de ce modèle pour des tests de fonctionnement logique et de comportement dynamique. Il doit, en outre, favoriser les mesures et l'analyse statistique. Il nous faut donc un langage de programmation apte à la description de systèmes, et à la simulation à événements discrets.

C'est donc dans ce contexte de recherche général que se situe le travail décrit dans cette thèse, et que nous présentons maintenant.

I.2 - PRESENTATION DE NOTRE ETUDE

I.2.1 - NOTRE TRAVAIL DANS LE PROJET

Le but de notre travail était de concevoir, réaliser et expérimenter un système logiciel permettant de décrire des systèmes d'information d'organisations sous forme de maquettes qui servent à évaluer et prédire leur comportement dynamique.

Dans un premier temps, notre travail a porté sur l'expression de la dynamique des systèmes d'information. C'est pourquoi, une partie importante de cette thèse traite de l'étude des mécanismes de synchronisation. La synchronisation, dans le projet, est basée sur le mécanisme de

moniteurs de HOARE (HOA74). La synchronisation, au niveau logique, est exprimée à travers la coopération des processus. Ceux-ci doivent parfois être déclenchés à des instants précis. C'est pourquoi nous introduisons une notion d'échéancier (secondaire). Au niveau physique, la synchronisation est l'expression de la concurrence des processus vis à vis de ressources physiques. Elle est, également, réalisée par des moniteurs. Un processus peut être bloqué en attente d'une ressource si celle-ci n'est pas dans un état satisfaisant; mais il peut également être bloqué si cette ressource n'est pas disponible. En effet, des ressources peuvent n'être disponibles que pendant certaines périodes en dehors desquelles aucun processus ne peut les utiliser (les occuper). On définit, alors, les calendriers de disponibilité des ressources.

Ces deux notions, échéancier et calendrier, introduites aux niveaux logiques et physiques, sont un apport important de ce travail dont elles sont, à notre connaissance, une originalité.

Une fois les concepts fondamentaux exprimant la dynamique résolus, notre travail a été de définir les autres composants nécessaires à l'écriture de maquette. C'est ainsi que nous avons été amenés à compléter le mécanisme général de moniteurs afin de l'adapter à chaque cas particulier rencontré. On dispose maintenant de plusieurs types de moniteurs spécialisés. Chaque type de ressources a été ensuite, conçu, réalisé séparément. Nous n'avons, pour l'instant, retenu que les types utiles à la réalisation de nos maquettes expérimentales, mais nous parlerons tout de même de ces ressources d'une manière générale.

Ces différents mécanismes ont été testés sur des petits systèmes expérimentaux. Mais il était bon de les tester également en vraie grandeur. C'est pourquoi nous

avons réalisé une maquette d'entreprise sur laquelle nous testons nos différents apports, d'une taille suffisamment importante pour nous servir de banc d'essais. Une fois les outils construits, nous pouvons introduire tout ce qui concerne l'aspect mesures et analyse statistique. Ce travail est en cours de réalisation par M. THAUREL (THA79).

I.2.2 - PLAN DE THESE

Après le rappel des notions fondamentales, la présentation du projet MAESTRO et la localisation de cette thèse dans ce projet, nous donnons le plan adopté.

Remarquons, avant tout, que la rédaction est basée sur la séparation logique-physique de l'étude des systèmes d'information. En effet on pourrait penser qu'il s'agit là d'un paradoxe, car si l'on veut parler de l'influence du temps, il nous semble qu'il faut nécessairement en parler au niveau physique. Mais nous pensons que cela aide à clarifier le plan, même si la conception d'une maquette ne suit pas cette présentation. Il ne s'agit là que d'un artifice de présentation.

Le deuxième chapitre, divisé en deux parties, présente l'architecture générale du modèle, sous son aspect logique, puis sous son aspect physique. Le premier donne une définition informelle de ce que l'on entend par structure de données et par processus synchronisés. On montre ensuite quels sont les impacts de l'introduction de la notion de temps au niveau logique, pour terminer par l'étude des problèmes posés par le niveau de détail. Le deuxième passe en revue quels sont les composants physiques susceptibles d'être rencontrés dans une modélisation de système d'information. On y trouve tout d'abord un récapitulatif des types de ressources utilisées, puis un exposé sur la modélisation de ces ressources. On développe l'aspect allocation, puis les différences essentielles dues à la modélisation par rapport à la réalité des systèmes.

Le chapitre 3 traite des langages et des outils de description. La première partie propose une classification des langages de simulation. Après avoir justifié le choix de SIMULA, nous en faisons une rapide présentation. La deuxième partie traite des principaux outils de synchronisation que l'on peut trouver dans la bibliographie. Nous justifions l'emploi des moniteurs comme outils de synchronisation par rapport aux autres mécanismes. Nous montrons, enfin, comment il est possible de réaliser, grâce à SIMULA, certains de ces outils de synchronisation.

Le chapitre 4 apporte plus de détails sur les divers composants de notre système. On y trouve, d'abord, une description des outils généraux (moniteurs, classes...), puis on retrouve les deux points de vue cités précédemment :

- les aspects logiques traitent du détail des structures de données et des processus synchronisés en expliquant comment ces composants sont réalisés dans le projet ; la prise en compte du temps à ce niveau, se traduisant par la notion d'échéancier, est également exposée en détail ;
- les aspects physiques montrent quels sont les types de ressources utilisés et comment ils ont été réalisés ; on expose également en détail le mécanisme de calendrier.

Le chapitre 5 propose la méthode à suivre pour écrire une maquette. C'est, schématiquement, une notice d'emploi du package orientée utilisateur. Dans ce but, on présente dans ce chapitre une réalisation concrète de maquette ; elle concerne la gestion des commandes clients et la facturation dans une entreprise commerciale.

Enfin, le chapitre 6 montre quelles sont les possibilités statistiques prévues dans le système. On y expose les différents types de mesures effectuées, les

outils conçus pour leur relevé et leur impact sur les classes existantes. L'exploitation des résultats est expliquée au travers de l'application présentée au chapitre précédent.

En conclusion, le chapitre 7 dresse un rapide bilan du travail fait dans le cadre du projet et tente de définir des perspectives d'avenir.

Les chapitres 8 et 9 comprennent respectivement la bibliographie et des annexes donnant quelques programmes écrits et testés à titre de documentation.

CHAPITRE II

ARCHITECTURE GENERALE D'UNE MAQUETTE

CHAPITRE II -

ARCHITECTURE GENERALE D'UNE MAQUETTE

Comme nous l'avons déjà précisé, nous développons ce chapitre en considérant d'abord les aspects logiques de l'architecture générale d'une maquette, puis ses aspects physiques. L'architecture d'une maquette doit refléter celle du système décrit, du moins jusqu'à un certain niveau de détail. Nous présentons ici, les différentes notions qui interviennent dans la description d'une maquette.

II.1 - POINT DE VUE LOGIQUE

De ce point de vue, une maquette est constituée de données, et de transformations effectuées sur ces données.

On trouve dans la littérature, de nombreuses définitions du terme "donnée" ; généralement ces définitions sont très imprécises, ou alors très formelles (J.L. REMY (REM74)). Ce travail n'étant pas de nature théorique, nous dirons simplement qu'une donnée comporte des ensembles (ou types) d'objets, des relations et des fonctions partielles dans ces ensembles. Une telle donnée caractérise une partie de l'état du système à un instant précis. Mais pour pouvoir parler de changement d'état, on doit considérer l'évolution des données du système. C'est ainsi qu'on introduit la notion d'ensemble de données D vérifiant certains axiomes (REM74). Pour donner un exemple simple, on peut citer l'ensemble des files d'attente d'objets d'un certain type. Si l'on veut garder une certaine généralité, on peut ne pas préciser, pendant un certain temps, de quel type d'objets il s'agit. Ceci conduit naturellement à la notion de type abstrait (LIS77, GUT77). Nous n'en parlerons pas ici de manière détaillée, mais nous y ferons allusion incidemment par la suite.

L'évolution des données dans le temps s'exprime à l'aide des concepts de modification et de processus.

II.1.1 - STRUCTURES DE DONNEES

On appelle modification dans l'ensemble D une fonction partielle (non partout définie) de D dans D.

On appelle structure de données un couple S = (D,M), où D est un ensemble de données vérifiant certains axiomes, et M un ensemble de modifications sur D.

Afin d'illustrer ces concepts, reprenons l'exemple précédent. On peut considérer la structure S = (D,M), où D est l'ensemble des files d'attente d'objets d'un certain type, et où M est l'ensemble des modifications sur D : par exemple, l'adjonction d'un élément en tête de file, la suppression d'un élément, le vidage d'une file ...

Cette structure de données apparaît souvent dans nos maquettes. Toutefois, on peut utiliser toutes sortes de structures, et une étude est en cours pour déterminer quelles sont les plus intéressantes à retenir, compte tenu de nos préoccupations : on trouvera dans (DUF78c) un développement sur ce sujet.

Ces structures servent à représenter dans les maquettes :

- des fichiers séquentiels,
- des fichiers en accès direct,
- des bases de données,
- des "boîtes à lettres",
- des fichiers manuels...

qui interviennent dans le système réel.

Dans ce paragraphe, on ne se soucie pas des problèmes posés par l'implantation physique de ces structures.

On peut aussi remarquer qu'il n'y a, en général, pas correspondance bijective entre les structures de données réelles et celles d'une maquette. Nous abordons là le problème posé par le niveau de détail et par conséquent les effets de la simplification et de la réduction.

La réduction sémantique des données dans une maquette consiste à prendre en compte exclusivement certaines structures de données ou certaines fonctions d'accès, par une sorte de projection du système sur un sous-espace.

Parfois, la réduction de certaines structures ou de certaines fonctions peut gêner la possibilité d'accès à une donnée intéressante du système. Ayant besoin de celle-ci dans le modèle réduit, on est amené à introduire de nouveaux éléments d'information qui, en quelque sorte, synthétisent la donnée qui a été perdue. Cette différence par rapport à la réalité est explicitée par l'exemple suivant : on verra, au paragraphe 5.2. qu'on considère des commandes avec chacune un attribut "nombre de lignes de commande". Dans la réalité, chaque ligne d'une commande est individualisée et traitée en tant que telle.

Bien que la réduction sémantique des données diminue, en général, considérablement le volume des informations, on est parfois amené à réduire encore les données du point de vue quantitatif. En effet, il est impensable, dans un modèle réduit, de manipuler un aussi grand volume de données que celui qui existe, le plus souvent, dans le système réel, et ceci, quels que soient les objectifs poursuivis par la modélisation. Les retombées de ces réductions se font sentir essentiellement lors de l'appréhension des durées de traitements, pour lesquels il faut tenir compte de leur complexité vis à vis du volume de données manipulées.

Les données partageables d'une maquette sont regroupées en structures protégées chacune par un moniteur, au sens de HOARE (HOA74). Les détails de l'organisation et de la gestion interne à un moniteur n'ont pas à être connus de l'extérieur. Seules certaines fonctions et procédures locales peuvent être appelées de l'extérieur par des processus pour accéder ou modifier les données protégées. Leur exécution est toujours faite en exclusion mutuelle, pour éviter des effets chaotiques. Quand les données accédées par un processus ne sont pas dans un état compatible avec sa poursuite, le processus est bloqué sur une condition dans le moniteur correspondant (il relâche alors son exclusivité), jusqu'à ce qu'un autre le délivre. Ce mécanisme de blocage sur condition permet de réaliser la synchronisation des processus. Dans un même système, il peut y avoir plusieurs moniteurs du même type, c'est à dire avec des structures de données et un comportement identiques.

Avant de donner un exemple, nous précisons la notion de processus.

II.1.2 - PROCESSUS SYNCHRONISES

Un processus (ou calcul) est une suite d'accès et de modifications sur des structures de données. Une définition plus proche d'une machine est donnée dans (CRO75) : "Un processus est une suite temporelle d'exécutions des instructions d'un programme".

Dans une maquette, un processus est l'exécution d'un "programme" qui crée, transmet, transforme ou reçoit des données. On peut distinguer des processus représentant deux grandes catégories de traitements de données du système réel :

- des "instances" de programmes informatiques,
- des traitements manuels ou intellectuels.

Souvent, les instances de programmes informatiques sont représentées par des processus déterministes. C'est le cas des traitements qu'on peut décrire tels qu'ils existent dans le système réel. Pour d'autres, dont la sémantique est réduite, ces processus peuvent être indéterministes ou stochastiques.

Par exemple, on considère la mise à jour d'un fichier de commandes dont chaque enregistrement comprend, en plus d'éléments annexes, un numéro de commande et le nombre de lignes de cette commande. Si on connait le nombre de lignes de la commande concernée par la mise à jour alors ce processus de mise à jour est déterministe ; dans le cas contraire, il faut générer, de manière aléatoire suivant une certaine distribution, un nombre de lignes et le processus devient indéterministe.

On retrouve là encore les problèmes dûs à la taille du système réel et au niveau de détail retenu pour une maquette.

Pour les traitements manuels, tels que ceux effectués dans les services administratifs des organisations, les processus qui les représentent sont assez souvent indéterministes. Fréquemment, les traitements rencontrés en organisation sont répétitifs : on les représente dans une maquette par des processus cycliques.

A un instant donné, un processus peut se trouver dans l'un des états suivants :

- actif, c'est à dire qu'une modification est en cours ou sur le point d'être effectuée ;
- passif (ou bloqué) dans le cas contraire.

Un processus est passif si l'état du système ne lui permet pas de poursuivre ses accès ou modifications. Quand ils sont tous réalisés, le processus est dit terminé.

Un processus peut se bloquer si une donnée, sur laquelle il veut faire une modification n'est pas dans un état favorable, ce qui se traduit par un blocage sur le moniteur qui la gère. De plus, comme on l'a précisé, on impose l'exclusion mutuelle entre certains accès ou modifications, ce qui est une autre source de blocage.

Un processus bloqué,non terminé, peut être débloqué dès que l'état du système lui permet d'effectuer la prochaine modification : il redevient alors actif.

Comme on l'a vu précédemment, on est contraint d'effectuer des réductions et des simplifications sur les processus.

Toutefois, la réduction sémantique des processus doit être faite très prudemment. En effet, ces simplifications, souvent dues à des réductions sémantiques de données, risquent d'éliminer un certain nombre de risques de blocage de processus. Ceci est dangereux, notamment quand on s'intéresse au fonctionnement ou au comportement dynamique du système.

De plus, la réduction du nombre de processus est souvent nécessaire car un système d'information réel en comporte souvent beaucoup trop pour qu'on puisse les reproduire tels quels dans une maquette. D'ailleurs, d'un point de vue logique, certains processus n'ont plus lieu d'exister, les données sur lesquelles ils travaillent ayant disparu de la maquette. Cette nécessaire prudence dans la réduction du nombre de processus est renforcée par le fait que certains utilisent des ressources. En supprimant purement et simplement de tels processus, on supprime des consommateurs de ressources, ce qui peut avoir des retombées sur le comportement des autres processus. Pour résoudre cette question, deux attitudes sont possibles :

- d'une part, on peut laisser subsister des processus parasites seulement en tant qu'utilisateurs de ressources ;

- d'autre part, il est souvent possible de les agréger à d'autres processus, dont on modifie le comportement dynamique en augmentant la consommation de ressources.

Le souci principal concernant la réduction du nombre de processus est, en général, d'obtenir des processus de taille optimale par agrégation de traitements d'informations.

Pour illustrer ce qui vient d'être dit par un exemple simple, présentons un petit modèle extrait d'une maquette de gestion des commandes-clients et de facturation d'une entreprise (que nous reprendrons par la suite plus en détail). Ceci nous permet d'introduire les notations graphiques utiles à ce niveau de la conception des maquettes.

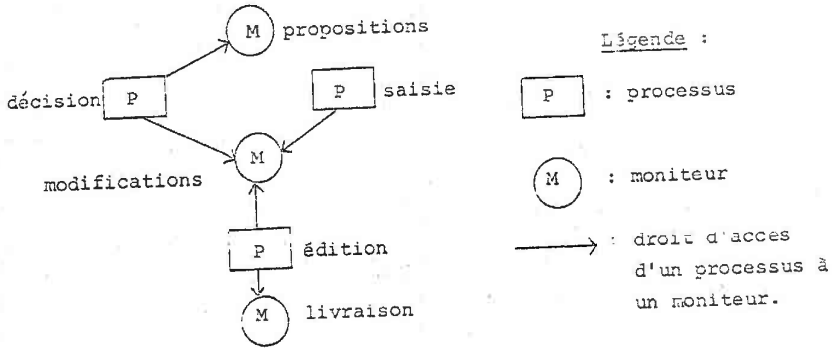


Figure 1.

Trois processus partagent une structure modifications : le processus décision y inscrit, ou non, des modifications de propositions de commandes à livrer, ce qui a pour effet de déclencher, soit un processus de saisie de ces modifications (sur console), soit un processus d'édition des bons de livraisons.

On peut résumer cette présentation des processus en reprenant la caractérisation qu'en a donné BRINCH HANSEN dans (BRI75). Un processus est un ensemble de structures de données privées et d'un programme séquentiel opérant sur ces données ou les données communes partageables. Un processus ne peut pas faire de transformations sur les données privées d'un autre processus. La synchronisation des processus est réalisée exclusivement par des moniteurs qui assurent le partage des données concernées.

Il est essentiel de s'attarder un peu sur la manière dont le temps est pris en compte dans nos maquettes.

II.1.3 - EVOLUTION D'UN SYSTEME DANS LE TEMPS

Les modifications effectuées par les processus n'interviennent pas n'importe quand : elles sont ordonnancées dans le temps.

Par exemple, il peut arriver, dans un service administratif que des traitements ne doivent commencer qu'à certaines heures de la journée : le tri du courrier peut être prévu dans une entreprise tous les matins à 10 heures. Ce processus reste donc bloqué jusqu'à ce que cette date soit atteinte, alors le tri éventuel est effectué, puis le processus retourne à l'état passif jusqu'à l'heure favorable suivante.

On est donc amené à introduire une notion d'échéancier d'activation des processus, dans lequel sont notées les différentes dates de déclenchement du processus considéré ; cet échéancier agit à ces instants pour libérer le processus par l'intermédiaire d'un moniteur dans lequel il est bloqué. La figure ci-dessous illustre ce mécanisme :

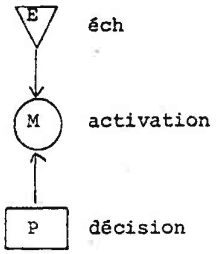


Figure 2.

Le processus de décision peut être libéré par l'intermédiaire du moniteur activation, à certaines dates précisées dans l'échéancier éch. Mais il peut l'être, en utilisant ce même moniteur, par un autre processus. Ceci permet l'utilisation des moniteurs comme seul moyen de synchronisation.

Si, dans l'exemple précédent, le tri du courrier est bien déclenché à 10 heures, le processus de tri reste en fait bloqué tant que le système n'est pas dans un état favorable, c'est à dire tant que le courrier n'est pas arrivé. Il devient actif quand c'est le cas, puis retourne à l'état passif une fois son travail terminé.

Avec l'exclusion mutuelle, qui se produit quand deux processus différents veulent avoir accès, ou modifier la même donnée en même temps, nous avons ici les trois principales causes de blocage de processus, au niveau logique, dans une maquette.

Nous pouvons résumer ce paragraphe en donnant l'architecture, au niveau logique, de l'extrait de maquette donné en exemple précédemment :

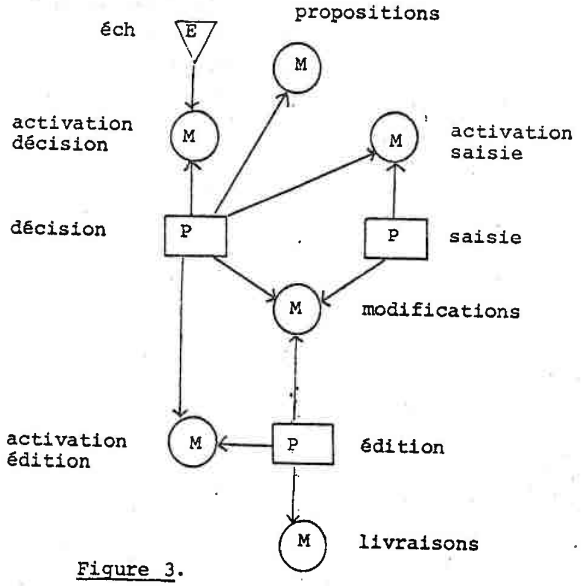


Figure 3.

Dans l'étude d'un système, on pourrait n'utiliser que ces concepts et tenter de prendre en compte la dynamique de ce système uniquement à travers des considérations du niveau logique, où l'exclusion mutuelle serait imposée sur les accès et les modifications, où chaque modification serait valuée par une durée d'exécution, et où on aurait retenu une politique d'ordonnancement des processus.

Mais ce serait s'éloigner considérablement de la réalité car il existe d'autres causes de blocage pour des processus. En effet, pour être exécutés, les processus ont besoin de ressources. Or le plus souvent, ces ressources sont en nombre restreint et les processus entrent en compétition (en concurrence) pour les obtenir. Pour étudier ces questions, nous devons nous placer à un niveau physique.

II.2 - POINT DE VUE PHYSIQUE

On tente d'abord de dresser un catalogue des ressources qui interviennent physiquement dans un système réel et qui sont nécessaires à l'enregistrement et à la transformation des données. Ces éléments prennent une large part dans la dynamique des systèmes ; ils permettent d'introduire la notion de durée d'une opération et sont sources de nouveaux blocages pour les processus.

Puis, on montre comment on les modélise et l'on tente d'établir une typologie des différents modes de représentation de ces ressources.

II.2.1 - RESSOURCES REELLES DU TRAITEMENT DE L'INFORMATION

On distingue, dans les systèmes réels, des ressources actives et des ressources passives.

a) Les ressources actives, ou processeurs, sont celles qui exécutent tout ou partie d'un processus. Elles se répartissent en moyens matériels de traitement et en moyens humains. Si un processus ne possède pas l'usage d'un tel processeur, il est nécessairement passif.

Les moyens matériels actifs sont des machines capables de fonctionner de manière autonome, au moins un certain temps, pour effectuer des opérations de traitement de données. Ce sont, par exemple, des ordinateurs ou des réseaux d'ordinateurs. Ils servent à la partie automatique du système.

Les moyens humains forment la partie non automatique. Ce sont essentiellement des personnes ou des groupements de personnes : par exemple ateliers, entrepôts, services administratifs, services informatiques...

b) Les ressources passives sont des moyens auxiliaires, utiles aux processus (ou aux processeurs) pour pouvoir se dérouler. Ils ne peuvent travailler de manière autonome, donc sans moyens actifs. Un processus en attente d'une de ces ressources se bloque. On peut distinguer deux types de ressources passives : les consommables et les réutilisables. Pour la partie automatique, ce sont les mémoires, les lignes téléphoniques, le papier,... ; pour la partie non automatique ce sont les matériels de bureau, les machines diverses, ...

On peut considérer, de ce point de vue, qu'une donnée, favorable à l'accomplissement d'une modification par un processus, est une ressource, en l'absence de laquelle le processus se bloque.

Dans la réalité, qu'elles soient actives ou passives, certaines ressources ne sont pas disponibles tout le temps pour le système que l'on considère. Ainsi il nous faut introduire la notion de calendrier comportant les périodes de disponibilité des ressources. Tout processus demandant une ressource en dehors de ses périodes de disponibilité, se bloque jusqu'au début de la prochaine période. Ce mécanisme introduit une nouvelle source de blocage. Inversement, à la fin d'une période, afin de prendre en compte la dynamique des systèmes, il est important de connaître les conditions de blocage induites par les ressources. C'est pourquoi nous parlons brièvement de la question de l'allocation de ressources aux processus dans les systèmes réels.

Il existe de multiples politiques d'allocation de ressources. Pour certaines d'entre elles, on peut complètement en préciser le principe : c'est le cas de l'allocation des ressources d'un ordinateur à des processus, où la politique définie pour le système d'exploitation est appliquée automatiquement.

D'autres sont beaucoup moins rigoureuses : c'est le cas de l'allocation de certaines personnes à certains travaux.

L'allocation des ressources peut être statique ou dynamique. Dans le premier cas, on alloue des ressources à des processus pendant une période sans se soucier de l'utilisation qu'ils peuvent en faire. Dans le deuxième cas, on alloue à la demande des ressources, c'est à dire que cette allocation n'est effective que si le processus concerné en a exprimé le besoin, et on lui retire la ressource dès que ce besoin n'existe plus.

L'allocation peut être centralisée ou décentralisée. Dans le cas centralisé, toutes les ressources sont contrôlées par un allocateur unique. Celui-ci les distribue aux différents processus. Dans le cas décentralisé, il existe plusieurs allocateurs contrôlant une ressource ou un ensemble de ressources. La première solution (stratégie globale) permet une certaine cohérence dans l'allocation des ressources mais elle est souvent plus complexe à mettre en oeuvre.

Un problème crucial de l'allocation des ressources, est d'éviter l'interblocage (ou deadlock). Deux processus peuvent, en effet, être dans des situations telles que chacun occupe des ressources tout en attendant celles possédées par l'autre : ces processus se bloquent indéfiniment. Cet interblocage peut mettre en jeu un nombre important de processus et de ressources. Le déblocage souvent très complexe, risque de faire perdre une certaine partie du système. Ce problème peut être évité par des contrôles dans un contexte d'allocation centralisée des ressources.

En dehors de ces deux modes d'allocation, on peut définir d'autres caractéristiques des ressources.

Une ressource possède n points d'accès ($n \geq 1$) si n processus peuvent l'utiliser au même instant ; pour n = 1 la ressource est appelée critique, pour $n > 1$ elle est dite partageable (à accès multiple).

Une ressource est dite préemptible si, étant occupée par un processus, elle peut être allouée à un processus demandeur, à condition que celui-ci ait une priorité plus forte que celui-là.

On se rend compte que l'allocation de ressources dans un système réel est extrêmement complexe. C'est pourquoi on est amené à en effectuer une modélisation.

II.2.2 - MODELISATION DES RESSOURCES

Nous dressons d'abord un catalogue des différents types de ressources employées dans les maquettes, puis nous présentons un essai de typologie des modes de travail pour montrer les différentes manières de représenter dans une maquette, des ressources du système réel.

a) Différents types de ressources

Qu'elles soient actives ou passives, humaines ou matérielles, les ressources d'une maquette se ramènent à des types bien précis dont nous donnons la liste et les principales caractéristiques.

1. Ressources critiques simples

Ce sont des ressources à un seul point d'accès, ou dans la terminologie des files d'attente : des files simples à un serveur. Elles sont préemptibles ou non. Dans le premier cas, elles sont utilisées par des clients avec priorités fixes pendant le service. Un processus occupant cette ressource en perd le contrôle quand un autre processus, de priorité plus élevée, demande cette même ressource ; le premier processus, interrompu, reprend son travail au point d'interruption dès que la ressource est libérée.

Les ressources critiques simples non préemptibles sont utilisées par des processus sans priorités, alors gérés selon le mode PAPS (premier arrivé - premier servi) ; ou avec priorités fixes, alors gérés suivant le principe qui désigne le premier servi comme étant celui qui a la plus forte priorité.

Qu'elles soient d'un type ou d'un autre, on peut ou non associer aux ressources critiques simples, le mécanisme de calendrier. Dans le premier cas, on y fixe les périodes de disponibilité de la ressource pendant lesquelles le serveur est utilisable, dans le second cas on considère le serveur comme toujours disponible.

2. Ressources à accès multiples

Ce sont des ressources à plusieurs points d'accès, c'est à dire utilisables par plusieurs processus en même temps. Nous ne limitons pas le nombre d'accès demandés par un processus à la fois : on obtient des files d'attente à plusieurs serveurs utilisés par des clients sans priorités et gérées suivant le mode PAPS, ou avec priorités fixes et gérées suivant la priorité la plus forte. On peut, ou non, leur associer le mécanisme de calendrier. On rattache à ce type de ressources celles qui sont utilisables par un nombre quelconque de processus appelées ressources à nombre infini de points d'accès.

3. Processeurs partagés

Ce sont des files d'attente à un serveur, celui-ci partageant son temps de manière équitable entre tous les processus utilisateurs selon une technique de "tourniquet" (round-robin). Les clients sont considérés sans priorités, le processeur partagé pouvant être ou non soumis à un calendrier.

Ces différents types de ressources servent à modéliser des ressources actives ou des ressources passives non consommables. Les ressources consommables ne sont pas considérées pour nous comme des causes de blocage possible : on se contente de comptabiliser leur consommation.

Bien d'autres types de ressources sont envisageables, mais nous limitons le nombre de type de ressources utilisables dans les maquettes à ceux qui nous ont semblé indispensables pour rendre compte simplement et efficacement du comportement d'un système.

b) Allocation des ressources

Nous avons vu en II.2.1 les différentes caractéristiques de l'allocation des ressources dans un système réel. Il est possible de combiner toutes ces possibilités dans une modélisation.

Mais, dans nos maquettes, les ressources sont généralement gérées indépendamment les unes des autres, de manière décentralisée, même si ce n'est pas toujours le cas dans la réalité. Ceci peut paraître une simplification abusive, mais les expériences ne manquent pas d'évaluations de systèmes réalisées très convenablement par des modèles à ressources décentralisées (Cf.: réseaux de files d'attente). Cette simplification paraît exclure de l'analyse des systèmes d'information, certains détails tels que ceux liés au fonctionnement des systèmes d'exploitation des ordinateurs où la gestion est souvent globale. Il faut cependant remarquer que l'échelle des temps est, dans ce cas, sans commune mesure avec l'échelle plus "humaine" utilisée dans les maquettes.

On est fréquemment amené à rassembler des ressources élémentaires pour des raisons de niveau de détail et de simplicité. C'est ainsi, comme nous le verrons plus loin, qu'un ordinateur peut être considéré comme une ressource à accès multiple infini. Là encore, le niveau de détail prévaut dans le choix de telle ou telle manière de modéliser.

Un problème, lié à ces simplifications, subsiste dans l'utilisation d'une allocation de ressources décentralisée : c'est celui de l'interblocage (ou deadlock).

Nous verrons, dans le chapitre consacré à la présentation détaillée des composants d'une maquette, qu'une restriction à l'utilisation des ressources permet d'éviter l'interblocage. En effet, pour des raisons techniques, il a été décidé que :

"Un processus utilisant une ressource ne peut pas être bloqué autre part que sur cette ressource". Il ne peut pas être bloqué en attente d'une autre ressource ou, plus généralement, dans un autre moniteur. Un processus ne peut donc pas utiliser plusieurs ressources "bloquantes" à la fois. Mais on remarque qu'un processeur du type "ressource à nombre infini d'accès sans calendrier" ne provoque aucun blocage pour les processus qui l'occupent : il est toujours libre pour n'importe qui. Ce type de ressource peut donc être utilisé en même temps que d'autres ressources. Cette restriction permet d'éviter l'interblocage puisqu'un processus possédant une ressource ne peut pas en attendre une autre.

Afin d'unifier la description, les ressources sont gérées par des moniteurs. Ainsi on peut introduire les ressources, avec éventuellement des calendriers dans les maquettes avec le graphisme adopté pour les moniteurs.

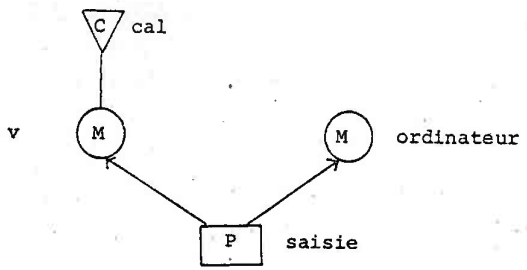


Figure 4.

Le processus de saisie est effectué par une personne v dont les périodes de disponibilité sont notées dans un calendrier cal. On remarque que la ressource "ordinateur" est représentée par son moniteur alors qu'il s'agit d'une ressource à nombre infini de points d'accès servant uniquement à de la comptabilité. Il nous faut faire particulièrement attention à ce que cette ressource soit utilisée en dehors de tout risque de blocage étranger à celle-ci, puisqu'elle possède un calendrier.

c) Autres simplifications au niveau physique

Ne pouvant pas prendre en considération des détails trop fins au niveau physique, nous sommes amenés à faire des hypothèses de simplification et de réduction sur les composants physiques du système.

Dans les maquettes que nous utilisons généralement, l'organisation physique des données et la mémorisation précise des programmes informatiques ne sont pas directement prises en compte, parce qu'intervenant la plupart du temps à une échelle trop fine.

Pour les données, par exemple, peu importe que l'on remplace dans une maquette, des fichiers physiques à rangement séquentiel, indexé ou calculé par des listes, tableaux ou autres structures, à condition d'y reproduire des fonctions d'accès et des modifications logiques ainsi qu'un comportement dynamique fidèle.

Pour les processus, on s'intéresse bien sûr à leur logique, mais également beaucoup à leur comportement dynamique. Certains phénomènes liés aux implantations physiques (attente pour défaut de page, pour entrées-sorties,...) sont totalement ignorés, se produisant à une échelle de temps sans rapport avec celle d'autres événements (dix-millièmes de seconde par rapport à l'heure ou au jour).

On remarque donc qu'il y a fréquemment un divorce très net entre les choix physiques du système et ceux d'une maquette, sans que cela entraîne des différences notoires de sémantique de données et de processus, ni, à un certain niveau au moins, de comportement dynamique.

Ces différences par rapport à la réalité et leurs conséquences (limitation des types de ressources,...) nous conduisent à proposer un essai de typologie des modes de travail qui montre comment on peut décider du type de ressources à utiliser pour représenter des cas bien précis.

d) Typologie des modes de représentation

Nous devons insister encore une fois sur le fait qu'un modèle est une représentation simplifiée de la réalité. Il n'essaie pas de décrire complètement les mécanismes internes, mais il s'arrête à un certain niveau de détail, le reste des mécanismes étant simulé par des procédures aléatoires.

Nous décrivons, ici, des travaux effectués par des personnes seules, des groupes de personnes, des processeurs matériels et des autres ressources.

1. Les personnes

- Une personne peut exécuter un seul processus (une tâche unique). Par exemple, une personne travaillant à la chaîne, ou encore un employé au guichet d'une banque, peuvent être considérés comme exécutant un seul processus. On représente alors ces personnes par des ressources critiques simples, sans priorités, éventuellement avec calendrier.

- Une personne peut aussi effectuer plusieurs tâches planifiées. C'est le cas, par exemple, d'un épicier qui, alternativement dans la journée, réceptionne des marchandises, les étiquette, fait la vente, passe les commandes, fait sa comptabilité... Si chaque période est réservée pour effectuer une tâche précise, on modélise cette personne par autant de ressources critiques simples, avec calendriers, que de tâches (de processus). Toutefois, il faut prendre garde à ce que les périodes de disponibilité des différents calendriers soient disjointes : en effet, une personne ne peut pas faire ici deux choses à la fois.

- Une personne peut effectuer des tâches multiples "à la demande" sans priorité. L'exemple le plus typique est celui d'une dactylo dans un secrétariat. Cette méthode de travail, très simple, pour laquelle on utilise une ressource critique simple, éventuellement avec calendrier, comporte quelques dangers : il existe le risque d'une monopolisation de la ressource par un gros travail (une thèse ?...), alors il peut se former une file d'attente importante ; on n'a pas la possibilité de distinguer les travaux urgents de ceux qui le sont moins.

- Si on introduit un système de priorités pour l'exemple précédent, le problème des travaux urgents est résolu. Si de plus, on instaure la préemption, alors on ne risque plus la monopolisation incontrôlable. On modélise cette ressource par une ressource critique simple préemptible (ou non) avec priorités fixes et éventuellement avec calendriers. On peut citer comme exemple une standardiste qui s'occupe également du tri du courrier : les appels téléphoniques sont prioritaires vis à vis du tri, et ils doivent le préempter, c'est à dire disposer de la standardiste, quand ils interviennent.

On peut ici remarquer que nos choix sur les types de ressources nous ont conduits à adopter une politique de priorités fixes. On aurait pu imaginer divers systèmes de priorités :

- priorité plus forte au travail le plus court,
- priorité variable, par exemple croissant avec le temps, et même devenant infinie après une certaine date (pour satisfaire les exigences de la planification d'obtention des résultats);
- ...

On peut considérer ces politiques comme faisant partie d'un niveau de détail dont nous ne tenons pas compte. On signale qu'il n'y a aucun obstacle technique à la réalisation de ce type de ressources. Le problème le plus ardu est de fixer les politiques de variation adéquates des priorités.

- Quand une personne partage son temps entre plusieurs tâches en n'accordant à chacune qu'un quantum de temps à la fois, on considère qu'elle exécute des tâches multiples en "temps partagé" (time-sharing). L'allocation du prochain quantum se fait suivant le mode PAPS. On utilise, pour modéliser, un processeur partagé avec éventuellement un calendrier.

Remarquons que là encore les politiques d'allocation sont multiples mais il semble que l'adoption d'un système de priorités est inutile si le quantum est correctement choisi ("assez" petit).

- Enfin, si une personne travaille avec des comportements différents, planifiés dans des périodes différentes, on associe à chacune d'elles une ressource fonctionnant dans l'un des modes précédents ; en prenant garde à ce que les périodes de disponibilité de chacun des calendriers soient disjointes.

2. Groupe de personnes

Un service peut être défini comme un groupe homogène de personnes interchangeables, c'est à dire, capables d'effectuer les mêmes tâches (un découpage judicieux d'un service administratif, par exemple, est parfois nécessaire pour satisfaire l'homogénéité). On associe un seul calendrier à un service. Dans la réalité, ce service exécute plusieurs processus ; il nous faut donc savoir comment les ressources sont partagées entre plusieurs demandeurs. Ce problème est très délicat car il est possible d'envisager une multitude de cas (en particulier si on distingue les personnes (ou accès) du groupe : cela entraîne une politique de service très compliquée). On considère donc le problème de manière plus globale et l'on peut proposer deux types de modélisation :

- un processeur partagé avec ou sans calendrier ;

- une ressource à accès multiples, avec priorités, avec préemption, et avec calendrier. C'est une ressource à n points d'accès utilisables simultanément par des personnes avec priorités, ayant la possibilité de préemption et ayant le mécanisme de calendrier. Les processus peuvent demander : p accès obligatoirement, p accès au plus ou le maximum d'accès disponibles.

A chaque arrivée d'un processus, on reconsidère la répartition des accès de la ressource entre les processus utilisateurs. Nous voyons tout de suite que ce type de ressources est assez complexe et donc difficile à mettre en oeuvre (un gros problème est lié à la répartition des accès, par exemple, lors de préemption : il faut utiliser une technique de ramasse-miettes). Il n'est d'ailleurs pas certain qu'elles aident à mieux approcher la réalité. C'est pourquoi elles ne font pas partie de nos types de ressources utilisables.

3. Autres ressources

Du point de vue du matériel, le comportement d'un système informatique est difficile à appréhender de manière globale. En monoprogrammation, on peut considérer un ordinateur comme une ressource critique ininterruptible, ce qui ne pose pas de problème. Mais ce mode de traitement tend à disparaître. En multiprogrammation ou en temps partagé, la question devient plus délicate. Des modèles de comportement ont été proposés (LER77), à l'aide de systèmes de files d'attente. Leurs hypothèses sont assez précises :

- sur les composants de la machine simulée,

- sur les caractéristiques de charge,

- sur les comportements même d'utilisateurs aux consoles.

Dans notre cas, ces modèles semblent trop fins et trop contraignants pour nous être utiles. Il nous faut considérer les choses de plus loin et nous contenter de temps de réponse approximatifs (ce sont ceux qui nous intéressent le plus en général).

On modélise un ordinateur soit à l'aide d'une ressource à accès multiples (en général infini), le temps du travail pour un processus peut être alors variable en fonction du degré de multiprogrammation ; soit à l'aide d'un processeur partagé, dans ce cas les problèmes dûs aux temps de réponse sont réglés automatiquement.

Dans le cas de la modélisation d'un réseau d'ordinateurs, on ne tient pas compte de la charge de ce réseau. Les temps de réponse sont purement aléatoires sans se soucier de l'utilisation du réseau : ce serait là un degré de finesse trop grand.

Les ressources passives consommables ne sont pas considérées comme génératrices de sources de blocage. On ne procède qu'à des mesures sur leur utilisation ou sur leur consommation. Elles pourront donc être représentées par des ressources à nombre infini d'accès sans calendrier.

Cette étude, au niveau physique nous amène à compléter la maquette extraite de la gestion des commandes clients et de la facturation présentée en début de chapitre. Ainsi par apport des ressources et de leurs éventuels calendriers, la maquette devient :

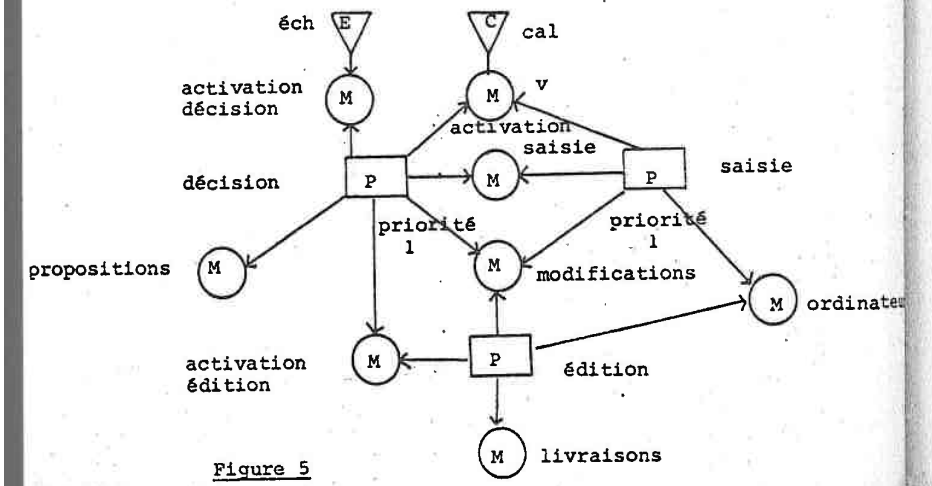


Figure 5

Le processus de décision, exécuté par la personne v avec la priorité 1 utilise les propositions de livraisons pour travailler comme on l'a indiqué précédemment. La saisie des modifications est exécutée par la même personne v sur ordinateur. L'édition des bons de livraisons est faite par ordinateur. La personne v est une ressource critique avec priorité, préemptible et avec calendrier (cal).

Ces schémas sont extraits d'une maquette dont l'ensemble est présenté dans ses détails au paragraphe 5.2.

En résumé de ce chapitre, nous pouvons donner une caractérisation plus précise des maquettes utilisées. Une maquette est un modèle réduit de système d'information. On essaie d'y conserver une structure d'information semblable, réduite et implantée physiquement de manière différente. Les processus y sont simplifiés et regroupés. On ne garde que les ressources physiques qui ont une influence déterminante sur le comportement du système, on les regroupe et on les décrit par des types en nombre limité. Ces ressources sont gérées indépendamment les unes des autres.

Nous avons, dans ce chapitre, mis en évidence l'architecture générale d'une maquette et les types de composants utilisés. Les chapitres suivants montrent comment les décrire dans un langage de programmation.

CHAPITRE III

LANGAGES ET OUTILS DE DESCRIPTION DE MAQUETTES

CHAPITRE III -

LANGAGES ET OUTILS DE DESCRIPTION DE MAQUETTES

Nous présentons, dans ce chapitre, les moyens disponibles pour écrire une maquette. Pour cela nous étudions les différents langages de simulation qui existent et regardons s'ils sont adaptés à notre problème. Nous expliquons pourquoi nous avons choisi le langage SIMULA après avoir présenté ce langage. La deuxième partie de ce chapitre est consacrée à l'étude des différents outils de synchronisation présents dans la bibliographie. Il ressort de cette étude que notre choix se porte sur les moniteurs, nous expliquons ce choix. Puis pour terminer ce chapitre, nous montrons comment réaliser, en SIMULA, certains des outils étudiés.

III.1 - LANGAGES D'ECRITURE DE MAQUETTES

III.1.1 - CHOIX DU LANGAGE

Pour mener à bien notre étude, il nous faut disposer d'un langage de description de systèmes capable de faire exécuter nos maquettes de manière simulée, en utilisant le "quasi-parallélisme" avec temps simulé.

En effet, on pourrait penser avoir besoin de deux langages :

- un pour la description de systèmes (incluant l'expression de la coopération des processus parallèles) ;
- un autre pour la simulation.

Mais on s'aperçoit, grâce à une étude des différents langages permettant la simulation, qu'il est possible de n'utiliser qu'un seul langage qui sert à la fois à la description de systèmes et à la simulation.

a) Les langages de simulation

Comme on l'a vu dans le paragraphe de caractérisation des systèmes, un système peut être discret ou continu. Dans un système, si après l'initialisation, les attributs varient de manière continue avec le temps, le système est dit continu ; sinon il est dit discret. D'après la classification donnée, on peut dire que nos modèles de systèmes sont dynamiques, stochastiques et discrets. C'est pourquoi, après avoir dit qu'il existe des langages spécifiques à l'étude des systèmes continus (CSMP, MIMIC ou SL/1 (VAU77)), nous ne reparlerons plus de ceux-ci pour nous intéresser uniquement à la simulation à événements discrets.

Une simulation à événements discrets concerne la modélisation sur ordinateur d'un système dont les changements d'états peuvent être représentés par un ensemble d'événements discrets.

Afin de présenter les différentes catégories de simulation à événements discrets, il faut rappeler les notions importantes dans l'élaboration de modèles de systèmes. Les plus spécifiques sont l'événement, le processus et l'activité. Un événement est un changement d'état du système, un processus est une suite d'événements ordonnés dans le temps et une activité est l'ensemble des opérations qui transforme l'état d'une entité.

Ces trois concepts sont à la base de la classification des démarches de simulation à événements discrets. On distingue la simulation par événements, la simulation par activités et la simulation par processus (ou par transaction). Chacune de ces approches ayant ses caractéristiques propres, leur développement met en jeu des langages de simulation à événements discrets spécifiques (DAH68). Nous étudions chacune de ces méthodes, en donnant leurs caractéristiques et les langages susceptibles d'être utilisés pour les mettre en oeuvre.

- L'approche par événements met en évidence le détail des opérations à effectuer quand un événement intervient. On associe une suite d'opérations (ou procédure) à chaque type d'événement. On perd dans cette approche, la notion de processus, car une procédure associée à un événement, se déroule dans sa totalité, sans blocage, avant de changer l'état du système. Comme exemples de langages utilisés pour mettre en oeuvre cette méthode, on peut citer GASP II (GAS69) qui est un package de sous-programmes FORTRAN spécialement écrits pour la simulation à événements discrets, dont l'utilisation est favorisée par le fait qu'il est basé sur un langage connu : FORTRAN ; un autre exemple est le langage SIMSCRIPT (MAR63) qui est peut-être le langage le plus facile à comprendre et qui permet la plus grande liberté dans l'écriture de calculs arithmétiques et logiques. Malheureusement il oblige à un long apprentissage et n'est pas ainsi beaucoup utilisé (FIS73). D'autres langages encore peuvent être utilisés, on en trouve une description dans (FIS73).

- La deuxième approche est celle par activités. Elle est basée sur le principe suivant : à chaque fois qu'un événement intervient, on passe en revue toutes les activités de la simulation pour déterminer quelles sont celles qui doivent commencer et celles qui doivent se terminer. On peut citer le langage CSL (BUX62) qui permet de travailler suivant cette méthode. L'exécution de modèles programmés avec ces langages est assez lente, mais un gros avantage est que ces modèles font ressortir les variables décisionnelles et sont donc facilement compréhensibles pour des non-informaticiens (VAU77).

- Enfin la troisième approche possible est celle de la simulation par processus ou par transaction (VAU77). Cette méthode met en évidence la progression dans le système d'une entité, depuis son arrivée (événement) jusqu'à son départ (événement). Cette approche réunit les caractéristiques très sophistiquées de l'approche par événement et les facilités de modélisation à un niveau conceptuel de l'approche par activités. La base de cette méthode est le processus, qui offre en plus de la notion de suites d'instructions (procédure) la possibilité de blocage. Plusieurs langages existent qui utilisent cette approche : on peut citer GPSS (IBM70), SIMULA (DAH66,DAH70), SØL (KNU64),.... Un des plus employé est GPSS qui est d'un usage assez aisé mais qui, semble-t-il, ne permet pas d'apporter beaucoup de détails sémantiques dans la simulation.

Certains langages, tels SIMULA, permettent de travailler suivant une quelconque de ces trois approches. Nous avons choisi, de part la nature même de notre travail, d'utiliser la méthode de simulation par processus ou transaction car elle permet de mieux rendre compte de la logique des transformations dans le système, parce qu'elle est plus naturelle (habituelle) en informatique, que les deux autres approches.

Partant de cette hypothèse, il nous a fallu trouver un langage qui puisse concilier la simulation par processus avec la description de système.

b) Les langages de description de systèmes

Parmi les langages de description de systèmes aptes à exécuter nos maquettes de manière simulée, par une technique de quasi-parallélisme avec temps simulé, on peut citer MODULA (WIR77) ou encore le langage SIMONE (KAU76,

BEZ76) qui est un langage PASCAL "parallèle" avec lequel les premiers essais ont été effectués (DUF76) (x). Ce langage présente les avantages de la simplicité et la sûreté de PASCAL, combinés avec la facilité d'expression de la coopération, grâce à une implémentation directe des moniteurs de HOARE (BRI75). Mais ce langage a l'inconvénient de rendre difficile le paramètrage des processus et des moniteurs. De plus on ne peut pas, dans ce langage, nommer explicitement les processus (en effet on utilise la séquence suivante pour lancer un processus : start new proc...). Ceci peut être un désavantage certain dans le cas de la préemption, où il nous faut "reconnaître" les processus.

Toutes ces restrictions font que nous choisissons le langage SIMULA. Son puissant concept de classes hiérarchisées permet très facilement l'adjonction de paramètres supplémentaires ; il est très facile d'y introduire les moniteurs de HOARE (BEZ75) ; enfin, mais ce n'est ici qu'une raison mineure, grâce aux procédures virtuelles de SIMULA, on a la possibilité de manipuler les types abstraits ($DUF78_c$). Une étude du langage nous en montre tous les avantages. Nous allons, simplement, rappeler les notions les plus intéressantes de ce langage afin d'aider à la compréhension de nos réalisations.

III.1.2 - PRESENTATION DE SIMULA

SIMULA (DAH66,DAH70) est une extension d'ALGOL60 dans laquelle le concept nouveau le plus important est celui des traitements quasi-parallèles.

Nous rappelons ici très brièvement les notions indispensables à la compréhension de nos réalisations techniques.

(x) On peut encore citer la tendance actuelle vers de nouveaux langages (BRI78,HOA78) qui suppriment toutes les variables globales pour ne conserver que les processus. Ces langages sont intéressants mais, pour l'instant, non opérationnels.

Tout d'abord, nous nous intéressons aux éléments descriptifs de SIMULA. La description d'un système, dans l'optique d'une programmation en SIMULA, se scinde en deux parties (VAU77) : une partie dite de décomposition du système et une partie de classification des composants obtenus. En effet, la décomposition du système fait apparaître des composantes individuelles. Or il est fréquent de trouver des composantes dont les structures de description se ressemblent. On peut alors répartir ces composantes en différentes CLASSES. C'est là un des puissants concepts du langage SIMULA, qui permet de définir des types d'objets paramétrés, tenant à la fois de la structure (PL/1, CØBØL, PASCAL...), de la procédure et de la coroutine (CON63), (VAU77). Une description de classe définit un concept et non un individu. On introduit une classe dans le système même s'il n'existe qu'un individu de cette classe. Dans d'autres cas le nombre de composantes peut être très grand.

Ces principes de décomposition et de classification sont à la base de la philosophie descriptive de SIMULA.

a) Les classes hiérarchisées

Nous ne donnons pas de description exhaustive d'une classe (cf CII72) mais nous ne retenons que la structure simplifiée suivante :

```
class nomdeclasse (paramètres formels) ;
                  spécifications de paramètres ;
begin
      déclarations des attributs ;
      actions à effectuer ;
end ;
```

C'est ainsi qu'on peut décrire une classe d'êtres humains de la manière suivante :


```
class humain (nom) ; text nom ;
begin
      integer âge ;
      boolean procedure mineur ; mineur := âge <18 ;
      âge := 0
end ; ¢ humain ¢
```

Cela définit une classe de nouveaux-nés, d'attributs nom (paramètre formel), âge et mineur (fonction procédure booléenne). On repère un objet (instance dynamique d'une classe) par un pointeur (une référence). Par exemple :

```
ref (humain) x,y ;
```

définit deux pointeurs x et y vers des objets de la classe humain. Tant que ces objets ne sont pas créés, les références x et y sont nulles :

```
x :- none ; y :- none ;
```

Pour créer effectivement un objet, avec passage de paramètres et exécution du corps de classe, on utilise le générateur new:

```
x :- new humain ('DUPONT')
```

crée un nouveau-né, appelé 'DUPONT' et repéré par x. Les attributs d'un objet sont accessibles de l'extérieur de l'objet en notation pointée :

```
x.âge
```

est une variable ayant pour valeur l'âge de la personne pointée par x ;

```
x.âge := x.âge + 1 ;
```

incrémente l'âge de x (chaque année par exemple).

A ce concept de classe vient s'ajouter celui de la hiérarchisation. En effet, on peut créer des sous-classes d'une classe par la méthode de préfixage, qui permet la concaténation des définitions de deux classes :

un objet d'une classe préfixée possède alors, outre les attributs propres à sa classe, tous ceux de la classe dont le nom figure en préfixe (DAH70). Les notions de préfixe et de sous-classes peuvent être généralisées transitivement.

Ainsi on peut définir, à partir de la classe humain, les classes homme et femme :

```
humain class homme ; null ;
humain class femme (nomjeunefille) ; text
                                    nomjeunefille ;
begin
        boolean procedure mariée ; mariée := not
                                (nomjeunefille ='␣') ;
end ;
```

Ainsi la séquence suivante déclare, puis crée, des objets des classes homme et femme :

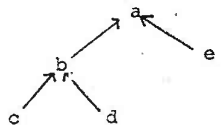
```
ref (homme) x ;
ref (femme) y ;
x :- new homme ('DUPONT') ;
y :- new femme ('DURAND', '␣␣') ;
```

x est un nouveau-né du sexe masculin s'appelant 'DUPONT', et y pointe vers un nouveau-né du sexe féminin non marié et s'appelant 'DURAND'.

On définit la séquence de préfixes d'une classe qui comporte un préfixe comme étant formée de ce préfixe et de la séquence de préfixes de ce dernier.

Ainsi l'exemple suivant illustre cet énoncé (CII72) :


```
class a ; ... ;
a class b ; ... ;
b class c ; ... ;
a class e ; ... ;
b class d ; ... ;
```



hiérarchie (structure en arbre)

figure 6

La séquence de préfixes de la classe d est b,a. Si on représente par aa, ab, ac, ad, ae respectivement les attributs des classes a, b, c, d, e, on peut schématiser les attributs des objets des différentes classes ainsi :

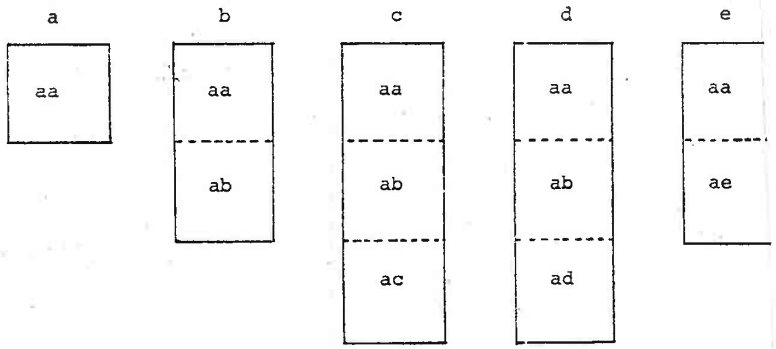


figure 7

On en déduit que les attributs d'un objet appartenant à la classe cl sont formés de l'union des attributs de cl et des attributs des classes de la séquence de préfixes de cl.

Ainsi, dans l'exemple précédent,

```
y.âge
```

donne l'âge de mademoiselle DURAND et

```
x.mineur
```

précise si oui ou non monsieur DUPONT est mineur.

Il existe de plus, dans SIMULA, des classes systèmes prédéfinies dont on peut utiliser les attributs dans un programme par préfixage de celui-ci par leur nom de classe ; ce sont les classes :

- SIMSET qui permet le traitement de listes ;

- SIMULATION qui permet la définition de processus et l'utilisation d'un échéancier standard.

b) Les traitements de listes en SIMULA

La classe SIMSET définit les éléments et les fonctions du traitement de listes linéaires chaînées et bilatères (symétriques). Chaque liste est composée d'une tête de liste et d'un nombre variable de liens. Une tête de liste est un objet de la classe HEAD ; un lien est un objet de la classe LINK. Ainsi, si l'on désire déclarer une liste de personnes appelée lpersonnes dans laquelle on veut ranger des objets de la classe humain, il faut procéder ainsi :

```
ref (head) lpersonnes ;
lpersonnes :- new head ;
```

et préfixer la classe humain par link afin de pouvoir ranger les objets de cette classe dans la file lpersonnes :

```
link class humain (nom) ; ...
begin
    ...
end ;
```

Les éléments d'une liste sont accessibles grâce à deux fonctions : suc (successeur) et pred (prédécesseur). Une liste peut donc être représentée par le schéma suivant : (CII72)

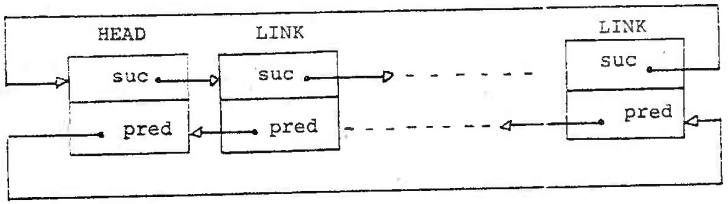


figure 8

Une liste vide est représentée par :

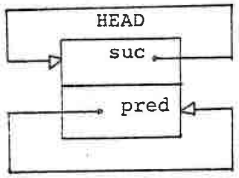


figure 9

L'ensemble des propriétés des listes est défini grâce aux trois classes attributs de la classe SIMSET :

- la classe LINKAGE, pour les éléments de liste, et ses deux sous-classes :

  . la classe HEAD pour les têtes de listes ;
  . la classe LINK pour les liens.

Grâce au principe de classes hiérarchisées, si une classe est préfixée par SIMSET, les attributs de SIMSET sont accessibles à l'intérieur de la classe. On peut donc utiliser les classes HEAD et LINK comme préfixes de classes décrivant des entités qui sont des têtes de listes ou des liens.

Un certain nombre de procédures définies dans SIMSET, permettent de manipuler les attributs des listes :

par exemple clear vide une liste de ses éléments ; cardinal, fournit le nombre de liens figurant dans la liste,...

Ainsi on utilise personne.cardinal pour savoir combien d'humains figurent dans la liste personne. D'autr procédures permettent les adjonctions ou les suppressions d'éléments dans des listes :

- x.into (personne) ; insère en queue de la liste personne le lien pointé par x
- x.precede (y) ; insère le lien pointé par x dans la file comportant le lien pointé par y et juste avant celui-ci (*)
- y.out ; supprime le lien pointé par y de la file dans laquelle il se trouvait (*)
- y.follow (x) ; insère le lien pointé par y juste après celui pointé par x dans la file où il se trouve (*).

Dans les cas d'insertion (into, precede, follow) si le lien à insérer appartenait déjà à une liste avant l'exécution de l'insertion, celle-ci le retire de la liste dont il faisait partie.

Les procédures first et last, utilisées en notation pointée avec des têtes de listes donnent respectivement le premier et le dernier lien d'une liste, empty indique si la liste est vide ou non.

(*) : Remarque : Une particularité du traitement des liste est qu'un lien ne peut appartenir à deux listes à la fois. On n'a donc pas besoin ici de préciser de quelle liste il s'agit puisqu'on sait où se trouve le lien pointé par y.

c) Le quasi-parallélisme

La classe SIMULATION est composée par l'ensemble des classes et des procédures permettant l'exécution de programmes de simulation. Elle est préfixée par la classe SIMSET, ainsi tous les attributs de celle-ci peuvent être utilisés dans la classe SIMULATION (CII72).

Simuler un système donné c'est ordonnancer dans le temps les différents processus intervenant dans ce système. A cet effet on est amené à associer à un processus ordonnancé, une date qui correspond au temps effectif où ce processus doit commencer son action. Cette date est désignée par evtime.

En SIMULA, la gestion de ces activations utilise un échéancier interne appelé SQS (sequencing set), dans lequel tous les processus ordonnancés sont rangés par evtime croissant.

A un moment donné de la simulation, on peut donc distinguer quatre types de processus :

- un processus unique dit actif qui est le premier processus de la SQS, c'est-à-dire un de ceux dont l'evtime est le plus faible. Il est repéré par le pointeur current ;
- un certain nombre, éventuellement nul, d'autres processus ordonnancés dans la SQS qu'on appelle suspendus ;
- un certain nombre, éventuellement nul, de processus ne figurant pas dans la SQS, mais dont l'action n'est pas terminée, et qui sont dits passifs (ou idle) ;
- un certain nombre, éventuellement nul, de processus dont l'action est terminée. Ces processus sont dits terminés au sens de la simulation.

Les entités principales intervenant dans la simulation sont donc les processus, objets de la classe PROCESS.

Cette classe étant préfixée par LINK, il en résulte que les processus ont toutes les caractéristiques des liens, et peuvent être chaînés dans des listes. Plusieurs fonctions spécifiques aux attributs d'un processus peuvent à tout moment donner l'état de celui-ci (actif, suspendu, idle, terminé), la date de sa mise en action (procédure evtime).

On dispose d'autre part de procédures d'activation activate et de réactivation reactivate. Ces procédures peuvent avoir un effet :

- immédiat ;
- à une date déterminée par addition d'un délai à la date courante (delay) ;
- à une date spécifiée (at) ;
- pour un processus donné, avant ou après un autre processus donné (before/after).

D'autres procédures permettent d'influer sur l'exécution des processus, par exemple :

- hold (t) termine la phase active du processus actif à cet instant (désigné par time) et le replace dans la SQS afin que ce processus soit réactivé à l'instant time+t ;
- passivate qui suspend le processus actif (current) en l'enlevant de la SQS et en reprenant l'exécution du processus le suivant dans la SQS ;
- wait (s) qui insère le processus actif dans la liste s puis le passive par un passivate ;

- cancel (x) qui rend passif le processus x s'il ne l'était déjà ;
- terminate termine le processus au sens de la simulation.

Toutes ces procédures donnent la possibilité de lancer ou d'arrêter l'activation d'un processus, la gestion proprement dite de la SQS (échéancier interne) étant automatique et transparente à l'utilisateur.

Pour utiliser les concepts de la simulation dans un programme, il suffit de préfixer le programme principal par SIMULATION. Ainsi on a accès à tous les attributs des classes SIMSET et SIMULATION.

Le langage SIMULA offre de plus la possibilité d'utilisation de procédures standards servant aux tirages de nombres aléatoires. On peut ainsi utiliser les tirages binaire (draw), uniformes (randint, uniform) gaussien (normal), exponentiel (negexp), de Poisson (poisson)...

Une procédure histo sert à la mise à jour d'un histogramme. De plus toute une bibliothèque de fonctions standards mathématiques est accessible par des procédures prédéclarées (max, sqrt...).

En résumé, on peut dire que SIMULA a de nombreuses qualités pour une description simple et claire des systèmes de processus coopérants. Son concept de classes hiérarchisées permet de fabriquer par extension des outils diversifiés.

Dans le paragraphe suivant, après une étude de différents outils de synchronisation, nous voyons comment en réaliser en SIMULA et en particulier comment on implémente les moniteurs de HOARE que nous développons par la suite.

III.2 - OUTILS DE SYNCHRONISATION

La définition d'une maquette, telle que nous l'avons donnée indique qu'une place importante y est faite aux questions de synchronisation et ordonnancement des processus.

Cet ordonnancement est réalisé partiellement par les processeurs dont nous avons parlé précédemment, mais les processus de nos maquettes peuvent être synchronisés de manière différente, tenant à la logique même du système, par la prise en compte des contraintes de "précédence" : un processus peut en effet être activé par un processus par l'envoi d'un "signal".

De multiples solutions ont été proposées pour résoudre les problèmes de synchronisation. Dans ce paragraphe nous en exposons quelques unes, sans toutefois être exhaustifs, en essayant de suivre l'évolution chronologique. Pour chacun des mécanismes nous donnons rapidement les caractéristiques afin de conclure le paragraphe en situant notre travail par rapport à ces études.

- Les coroutines

KNUTH dans (KNU68) fait un historique de la définition du concept de "coroutine". Il signale que ce terme a été utilisé pour la première fois par CONWAY en 1958, (CON63) après qu'il ait développé ce concept et qu'il l'ait appliqué initialement aux programmes d'assemblage et aux compilateurs.

Les coroutines constituent une généralisation du concept de sous-programmes en ce sens qu'il existe une complète symétrie entre les coroutines qui s'appellent mutuellement par opposition aux relations non symétriques entre les sous-programmes et le programme principal. Ainsi,

la principale différence entre les échanges programme - sous-programme et coroutine-coroutine tient dans le fait qu'un sous-programme est toujours lancé à son début, tandis que le programme principal ou les coroutines ont leur exécution reprise à la suite de l'endroit où ils se sont arrêtés précédemment. Des exemples d'utilisation peuvent être consultés dans (KNU68) et (CON63).

- FORK et JOIN sont deux primitives de synchronisation exprimant respectivement l'activation de plusieurs processus par un processus, et celle d'un processus par plusieurs autres (noeud de divergence et noeud de convergence). FORK à n arcs permet le déclenchement de n processus parallèles et JOIN peut être utilisé pour la synchronisation.

- Les événements de PL/1 (VEI72)

En PL/1 il est possible, en utilisant l'option task, d'initialiser des processus parallèles. Un événement est représenté par un symbole dont la déclaration peut être explicite (par l'attribut event) ou implicite lors de la création du processus (option task). Un processus qui exécute l'instruction wait (evt), où evt est un événement, se bloque si et seulement si l'événement n'est pas arrivé. L'affectation d'une valeur booléenne à un événement se fait par completion.

- Les réseaux de Petri (PET77) sont apparus vers 1962-63. Un réseau de Petri est un graphe biparti $G = (P,T,\Gamma)$ dont les sommets appartiennent soit à l'ensemble des places P, soit à l'ensemble des transitions T. Les arcs (branches) orientés sont définis par la relation binaire $\Gamma \subset PxT \cup TxP$: ils relient donc une place à une transition ou une transition à une place (GIR78).

Aux places, représentées par des cercles, on associe des ressources dont la disponibilité momentanée est représentée par les marques, points situés dans les places ; aux transitions représentées par des traits

on associe des actions qui consomment des ressources. La règle d'évolution des marques régit le déclenchement. Une transition est déclenchable si et seulement si toutes ses places d'entrée sont pleines. Le déclenchement, opération indivisible, consiste à enlever une marque à chaque place d'entrée et à ajouter une marque à chaque place de sortie.

Les réseaux de Petri permettent de représenter très facilement le parallélisme et la synchronisation des processus concurrentiels. Un autre avantage des réseaux de Petri est celui de la possibilité d'effectuer des preuves de bon fonctionnement et ainsi d'éviter tout interblocage (GIR78).

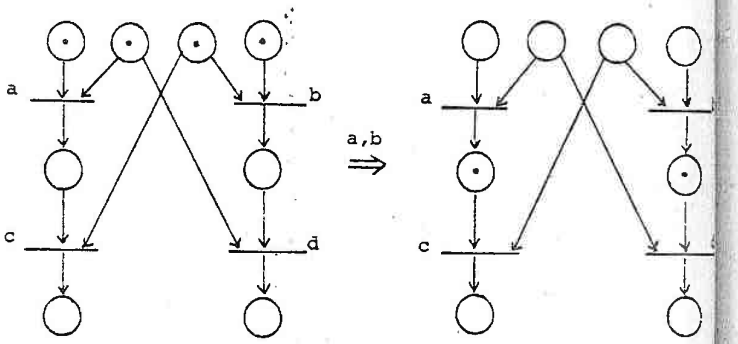


figure 10

La figure 10 illustre un cas d'interblocage. En effet, le déclenchement (mise à feu) des transitions a et b provoque le blocage du système car les transitions c et d ne pourront jamais être mises à feu.

Certains inconvénients peuvent être signalés quant à l'utilisation des réseaux de Petri. Il faut, notamment, disposer d'un mécanisme complexe d'indivisibilité pour les

règles de déclenchement (GIR78) ; de plus, une transition déclenchable n'est pas nécessairement mise à feu immédiatement : ceci est dû aux conflits qui se produisent quand deux ou plusieurs transitions déclenchables se partagent la même place d'entrée.

L'usage des réseaux de Petri pour décrire le contrôle d'un système semble, pour les néophytes, assez proche de celui des organigrammes pour décrire un algorithme séquentiel. Comme pour ceux-ci, il est bien sûr possible de structurer ces réseaux, de les écrire en couches successives, mais, de la même manière, on peut songer à leur abandon pour les langages de programmation structurée de systèmes de processus coopérants, dont nous parlons par ailleurs.

- Les sémaphores de DIJKSTRA (DIJ68)

Un sémaphore s est constitué d'une variable entière e(s) et d'une file d'attente f(s). La variable e(s) est la valeur du sémaphore initialement non négatif. On peut agir sur un sémaphore s par deux primitives qui sont des opérations indivisibles : P(s) et V(s) (CRO75).

Ce mécanisme est utilisé pour résoudre des problèmes généraux de synchronisation en ce sens qu'un signal d'activation est envoyé par la primitive V, ce signal étant attendu par la primitive P.

Un sémaphore est dit "privé" à un processus proc si seul ce processus peut effectuer l'opération P ; les autres processus ne pouvant agir sur le sémaphore qu'au moyen de la primitive V. Ce processus proc peut donc se bloquer, au moyen d'une primitive P, sur son sémaphore privé jusqu'à ce qu'un autre processus le "réveille" par une opération V sur ce sémaphore.

En combinant les sémaphores privés et les sémaphores d'exclusion mutuelle (CRO75) (chaque processus voit sa section critique encadrée par les opérations P et V sur un sémaphore

initialisé à 1), on peut réaliser des modèles de synchronisation plus complexes.

Ces mécanismes ne répondent pas à tous les besoins de la synchronisation ; en particulier, les primitives introduites permettent de traduire la coopération des processus de manière temporelle mais elles ne fournissent aucun dispositif de communication d'un processus à l'autre. Des propositions de sémaphores avec messages (SAA70) ont été faites pour éviter ces inconvénients mais ce sont là des mécanismes très lourds. HABERMAN dans (HAB72) estime qu'ils diluent la synchronisation et HOARE dans (HOA74) compare les sémaphores à des GØTØ !...

- ALGOL 68 (ALG72)

L'indépendance des calculs s'exprime, en ALGOL 68, par le calcul collatéral. Les propositions collatérales renferment, entre parenthèses tout ce qui doit être exécuté en parallèle. Le calcul synchronisé se fait à l'aide de sémaphores dont la définition a été donnée par DIJKSTRA. Il existe toutefois quelques différences mineures entre la définition des sémaphores d'ALGOL 68 et celle qu'en a donné DIJKSTRA.

- Les primitives Wait et Signal

Introduit par HABERMAN (HAB72), le mécanisme de synchronisation basé sur les primitives Wait et Signal ressemble fort à l'utilisation des sémaphores. L'idée générale est qu'un processus qui n'est pas autorisé à continuer, est forcé d'attendre jusqu'à recevoir un signal d'un autre processus ; à ce moment là, il pourra reprendre son exécution sur la prochaine instruction. L'exemple du modèle du producteur-consommateur (CRO75) illustre bien ce mécanisme, repris sous une autre forme dans les moniteurs

- Les moniteurs

Ce concept a été introduit par BRINCH HANSEN et repris par HOARE pour l'appliquer à la synchronisation dans les systèmes (HOA74). Un moniteur est une partie d'un système qui assure le partage de certaines ressources entre des processus. Il est composé de :

. variables locales qui donnent l'état des ressources contrôlées par le moniteur ;

. fonctions et procédures locales appelées dans le moniteur ;

. fonctions et procédures appelées depuis l'extérieur du moniteur et qui définissent les différents types d'accès aux ressources contrôlées.

L'usage d'un moniteur se fait en exclusion mutuelle : à tout moment, il y a au plus un processus actif à l'intérieur d'un moniteur donné. En général dans un système, plusieurs moniteurs ont la même structure, on les regroupe en classes.

Si, lors de la demande d'une ressource protégée par un moniteur, cette ressource n'est pas libre, l'usager est mis dans une file d'attente que l'on appelle "condition". A ce moment, l'exclusion mutuelle sur le moniteur est relâchée. On peut agir sur une file d'attente-condition par essentiellement deux primitives : signal et wait. Signal provoque la réactivation immédiate d'un processus en attente, si la file est vide cette primitive n'a aucun effet. Wait provoque l'insertion du processus dans la file d'attente. On verra au paragraphe IV-1 qu'on définit d'autres primitives afin de faciliter la gestion d'une condition.

Ce mécanisme n'est pas exempt de critiques. En effet KESSELS dans (KES77) propose un mécanisme de synchronisation dérivé des moniteurs en y incluant l'attente conditionnelle. Ceci a l'avantage de ne pas mettre en jeu

explicitement des signaux (présents dans les files d'attent KESSELS reprend le mécanisme du wait until avec la restric tion que la condition ne peut pas dépendre de variables locales à une procédure du moniteur. Ainsi minimise-t-il les coûts d'implémentation et d'exécution. Ces problèmes existent dans le Wait Until.

- L'attente conditionnelle

Dans le cadre de la simulation VAUCHER a dévelop le mécanisme de l'attente conditionnelle (WAIT UNTIL). Dans (VAU73), il expose les principes du Wait Until traditionnel L'implantation classique de ce mécanisme comporte quatre éléments :

. une liste globale dans laquelle est rangé tout processus en attente conditionnelle ;

. un "moniteur" qui s'exécute après chaque événement ; ce moniteur réveille tour à tour chaque process en attente dans la liste pour qu'il réévalue sa condition ;

. une variable booléenne qui informe le moniteur qu'un événement conditionnel a été déclenché et que le balayage de la liste d'attente doit être repris au début ;

. enfin une procédure Wait Until qui provoque la mise en attente d'un processus jusqu'à ce que la condition soit vraie.

La notion d'attente conditionnelle décrite par le Wait Until permet donc une spécification naturelle de la synchronisation dans les modèles de simulation à événements discrets. Ce mécanisme est conceptuellement simple puisqu'il n'y a pas d'envoi de signaux et puisqu'il permet de décrire du parallélisme de manière très pratique et très claire. Mais on s'aperçoit très vite que son emploi entraîne une très grande inefficacité de par son implantation très volumineuse et coûteuse (il faut réévaluer chaque condition à chaque événement). Conscient de ces désavantages, VAUCHER propose dans (VAU78) une amélioration

par accélération du Wait Until. Cette méthode est basée sur les concepts de PARTITION et de PARTITION RAPIDE. Une partition regroupe les événements susceptibles de se déclencher mutuellement et crée une liste d'attente distincte gérée par son propre moniteur pour chaque regroupement. Cette technique a été adoptée pour améliorer la vitesse du Wait Until sans changer son comportement fonctionnel. Mais il s'agit toujours de mécanismes demandant un effort de structuration de la part de l'utilisateur, et dont l'implémentation est encore monstrueuse. Un autre désavantage de l'attente conditionnelle est que, bien que facilitant l'allocation multiple, elle ne permet pas de faire de la préemption.

- Expression de chemins

La proposition de cette nouvelle méthode de synchronisation a été faite par HABERMAN (HAB75). La spécification de la synchronisation des processus à l'aide de ces expressions de chemins est expliquée dans (CAM74). Cette méthode est basée sur une séparation entre la description des actions (exécution d'une procédure par un processus) et leurs conditions d'exécution (synchronisation). Chaque expression de chemin donne les procédures qui doivent être synchronisées avec la manière de le faire (séquence, sélection, répétition, exécution simultanée). Chaque expression de chemin est implémentée par un contrôleur qui décide quand une procédure, demandée par un processus, est autorisée à commencer, ce qui a pour effet de permettre au processeur invocateur de continuer à travailler. Le système doit donc générer pour chaque procédure, un prologue dans lequel le processus demande à être exécuté, et un épilogue dans lequel le processus notifie au contrôleur la fin de son travail afin que celui-ci puisse réveiller d'éventuels processus en attente.

- Les mots de synchronisation

Introduits par ROUCAIROL dans (ROU78), la notion de mots de synchronisation est une généralisation des compteurs de VERJUS. Ayant constaté que l'utilisation des compteurs (ou des sémaphores) peut conduire à des situations de blocage permanent, ou à privilégier certaines classes de processus, il propose l'emploi d'opérations sur des mots de synchronisation. Ceux-ci décrivent l'exécution d'un programme par la suite (ou trace) des événements ou des opérations qu'il réalise. Alors une instruction de synchronisation apparaît comme une opération agissant directement sur la forme d'un mot d'exécution. L'emploi de primitives indivisibles portant sur les mots (ajout, efface, test) induit donc la synchronisation.

L'intérêt d'une telle méthode est de pouvoir disposer "d'une histoire" de ce qui s'est passé dans le système, sous forme de trace.

- La synchronisation par des langages de haut niveau (PUL78, RAY78) est actuellement à l'étude, les propositions faites concernant les problèmes où la séparation des expressions de contrôle et de traitement est possible. La partie traitement comporte les descriptions des opérations liées à un type abstrait, alors que la partie contrôle contient les règles d'utilisation de ces opérations.

- Les concepts avancés de synchronisation

Parmi ceux-ci, on peut citer les extensions des commandes gardées de DIJKSTRA (DIJ75) faites pour les processus communicants de HOARE (HOA78), et les processus distribués de BRINCH-HANSEN (BRI78). Ceux-ci éliminent les structures de données partagées en unifiant les notions de moniteurs et de processus. La synchronisation y est assurée par les entrées-sorties grâce à la communication entre les processus concurrents. Ces notions nouvelles conduisent à des programmes plus clairs mais sous-entendent un certain nombre de problèmes d'implémentation expliqués dans (HOA78).

III.3 - REALISATION DE CERTAINS OUTILS EN SIMULA

Afin d'appuyer, encore, la justification de notre choix de SIMULA, nous allons montrer comment on peut réaliser certains des mécanismes, parmi les plus intéressants, décrits dans le paragraphe précédent.

- Les coroutines se traduisent facilement grâce aux instructions detach et resume de SIMULA. Detach permet de suspendre l'exécution d'un objet, reprise par l'intermédiaire d'une instruction resume. Ainsi, une classe de coroutine ccoroutine s'écrira :

```
class ccoroutine ;
begin
      :
      detach ;
      :
      resume ;
      :
end ; $ ccoroutine $
```

- Les sémaphores sont implémentés en SIMULA à l'aide d'une classe sema dont la structure est :

```
class sema (s) ; integer (s) ;
begin
      ref (head) q ;
      q :- new head ;
end ; $ sema $
```

On décrit les opérations P et V de la manière suivante :

```
procedure P (sem) ; ref (sema) sem ;
inspect sem do begin
        s := s-1 ;
        if s < 0 then current.intc (q) ;
end ; $ P $
```

```
procedure V (sem) ; ref (sema) sem ;
inspect sem do begin
        s := s+1 ;
        if s ⩽ 0
        then begin
                integer i, n ;
                ref (process) pt ;
                n := randint (1,q.cardinal,U) ;
                pt :- q.first,
                for i := 2 step 1 until n do
                        pt :- pt.suc ;
                activate pt delay 0 ;
             end ;
end ; ⧼ V ⧽
```

La procédure V est ici décrite en tenant compte du fait que le processus à réactiver est choisi au hasard dans la file d'attente q. Si la discipline de choix est PAPS alors V s'écrit :

```
procedure V (sem) ; ref (sema) sem ;
inspect sem do begin
        s := s+1
        if s ⩽ 0 then
            activate q.first delay 0 ;
end ; ⧼ V ⧽
```

- L'implantation du Wait Until en SIMULA a été montrée par VAUCHER dans (VAU77) pour GPSSS. Elle comprend :

. la liste waitq ;
. la procédure wait until ;
. un "moniteur" (superviseur) activé après chaque événement qui à son tour réactive chaque transition dans waitq afin qu'elle réévalue sa condition.
. une variable globale action qui signale au "moniteur" qu'un événement conditionnel a été déclenché et qu'un nouveau balayage de Waitq doit être effectué.


```
ref (head) waitq ;
boolean action ;
ref (wait moniteur) moniteur ;
procedure wait until (condition) ; name condition ;
                    boolean condition ;
begin
     if not condition then
     begin
          activate moniteur after nextev ;
          priority.into (waitq) ;
          passivate ;
          while not condition do passivate ;
          out ;
          action := true ;
     end
end ; ⧼ wait until ⧽
process class wait moniteur ;
begin
     ref (transaction) p ;
  boucle :
     action := false ;
     p :- waitq.first ;
     while p =/= none do
     begin
          activate p ;
          if not action then p :- p.suc
                        else begin
                             p :- waitq.first ;
                             action := false
                             end
     end ;
     if waitq.empty then passivate
                    else reactivate current
                         after nextev ;
     goto boucle ;
end ; ⧼ wait moniteur ⧽
```

- Le mécanisme de moniteurs de HOARE dont le détail est donné en IV.1 est implémenté en SIMULA sous forme d'une classe :

```
class monitor ;
begin
      ref (head) urgentq, enterq ;
      boolean busy ;
      procedure enter ;... ;
      procedure leave ;... ;
      enterq :- new head ;
      urgentq :- new head ;
      busy := false
end ; ¢ monitor ¢
```

- Les réseaux de Petri

Dans un réseau de Petri, on associe aux places (cercles) des ressources (conditions, événements, opérande) (GIR78), et aux transitions (traits) des actions (processus). On remarque qu'il appartient au concepteur des réseaux de décomposer au besoin les opérations, ainsi le réseau ci-dessous peut-il être décomposé en séparant les débuts et fins de déclenchement, en introduisant un délai d'exécution de la transition initiale.

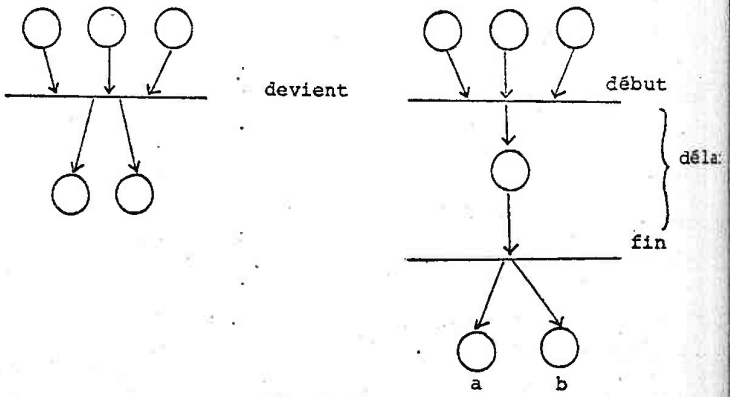


figure 11

On peut alors écrire ce mécanisme en SIMULA en introduisant une attente du processus (de son début) sur trois compteurs (ou trois moniteurs). Le délai est représenté par un HØLD dans le corps du processus et la fin consiste à envoyer deux signaux afin de valider les places a et b.

III.4 - CONCLUSION

Nous venons d'étudier sommairement un certain nombre de mécanismes de synchronisation. Notre choix s'est porté sur les moniteurs de HOARE car, entre autres qualités, ils permettent de construire beaucoup d'autres mécanismes. Les raisons pour lesquelles nous n'avons pas adopté les autres mécanismes peuvent se résumer à des questions pratiques. En effet, les réseaux de Petri, par exemple, nous auraient obligés à définir tout un langage d'interprétation, de plus ceux-ci ne nous conviennent guère de par leur ressemblance aux organigrammes et leurs inconvénients cités plus haut. Les sémaphores, comme on l'a vu, sont des outils un peu dépassés qui peuvent s'avérer dangereux si leur utilisation n'est pas correcte. Nous écartons le principe du Wait et Signal pour les mêmes raisons, étant très proche des sémaphores. Les inconvénients de l'attente conditionnelle qui nous ont poussés à ne pas utiliser ce mécanisme sont essentiellement liés aux coûts d'implémentation et d'utilisation. Comme on l'a dit, la notion d'expression de chemins est agréable car elle permet de bien séparer la description des actions de leurs conditions d'exécution, mais là encore il nous faudrait écrire un traducteur. Enfin les concepts avancés proposés par HOARE et BRINCH HANSEN posent un certain nombre de problèmes d'implantation, étant basés sur le mécanisme de commandes gardées dont l'indéterminisme les rend très difficiles à réaliser.

En conclusion, et au vu de leurs références, il nous semble que les moniteurs sont particulièrement bien adaptés à nos applications bien qu'ils ne soient pas exempts de critiques (KES77) et qu'il nous faille les adapter pour mieux rendre compte de la synchronisation dans nos maquettes. En effet, nous sommes obligés d'introduire de nouveaux types de conditions afin, par exemple, de prendre en compte les phénomènes de préemption.

Le chapitre suivant donne des détails sur ces transformations ainsi que sur chacun des composants des maquettes.

CHAPITRE IV

DETAILS DES COMPOSANTS

CHAPITRE IV -

DETAILS DES COMPOSANTS

Ce chapitre a pour but de présenter en détail, chacun des composants utilisables dans les maquetttes. Pour chacun d'eux, nous exposons comment nous les avons réalisés en SIMULA et quelles sont leurs contraintes d'utilisation.

Tout d'abord nous nous intéressons aux outils généraux formant la base des composants ; puis notre étude se scinde en deux parties : les composants décrivant les aspects logiques de la représentation de systèmes par des maquettes puis ceux qui décrivent les aspects physiques.

Tous les outils présentés ont été regroupés dans une classe unique appelée MAESTRO dont on verra l'utilité au chapitre 5.

IV.1 - OUTILS GENERAUX

Les outils présentés ici ont été réalisés en SIMULA, en utilisant les attributs des classes systèmes. En effet la classe MAESTRO, dont nous détaillons le contenu dans le paragraphe V.1., est préfixée par SIMULATION.

Parmi les outils généraux d'écriture de maquettes, détaillons d'abord le mécanisme de moniteurs tel qu'il a été introduit par HOARE (HOA74, cf. § III.2), sa réalisation en SIMULA et les extensions que nous avons faites sur les conditions. Ensuite nous décrivons les procédures utilitaires.

IV.1.1 - OUTILS DE SYNCHRONISATION

a) Les moniteurs

En SIMULA, un moniteur est sensiblement l'équivalent d'un objet d'une classe dont les attributs seraient accédés en exclusion mutuelle. On peut donc définir dans ce langage des types de moniteurs paramétrés, sous-classe de la classe monitor dont le schéma est le suivant : (on trouvera en annexe 1 la réalisation complète) :

```
        link class monitor (nom) ; text nom ;
        begin
           ref (head) enterq, urgentq ;
(1)        boolean busy
(2)        procedure enter ;...;
(3)        procedure leave ;...;
           enterq :- new head ;
           urgentq :- new head ;
           busy := false
        end ; $ monitor $
```

(1) busy : booléen, vrai si et seulement si un processus occupe le moniteur ;

(2) procédure d'entrée dans le moniteur : blocage du processus appelant si busy est vrai ;

(3) procédure de sortie du moniteur : libération d'un processus bloqué à cause de l'exclusion mutuelle ou, s'il n'y en a pas, mise de busy à faux.

b) Les conditions

On a vu, dans le chapitre précédent, que lors d'une demande d'une ressource protégée par un moniteur, si cette ressource n'est pas libre, le processus demandeur est mis en attente dans une file appelée condition. L'exclusion mutuelle sur le moniteur est relâchée.

On peut agir sur une file d'attente - condition par diverses primitives dont le rôle est expliqué ci-dessous. La structure de la classe SIMULA associée aux conditions est la suivante :

```
(1)     class condition (m) ; ref (monitor) m ;
(2)     begin
(3)        ref (head) waitq ;
(4)        procedure cwait ;...;
(5)        procedure csignal ;...;
(6)        procedure sigleave ;...;
(7)        procedure csignalg ;...;
(8)        boolean procedure empty ;...;
(9)        waitq :- new head ;
        end ; $ condition $
```

(1) entête de la classe condition associée au moniteur m ;

(3) déclaration de la file waitq, file d'attente sur la condition ;

(4) l'appel de cette procédure permet de ranger le processus appelant dans la file waitq ;

(5) l'appel de cette procédure libère le premier processus en attente dans waitq ;

(6) l'appel de cette procédure est équivalent à un csignal sur la condition suivi de la sortie du moniteur;

(7) cette procédure fait csignal sur tous les processus en attente dans waitq ;

(8) cette fonction-procédure, à résultat booléen, permet de savoir si waitq est vide ou non ;

(9) création de la file waitq.

Les procédures notées en (6), (7) et (8) ont été définies afin de faciliter la gestion d'une condition.

La déclaration et la création d'une condition, par exemple libre, à l'intérieur d'une sous-classe de monitor, est réalisée ainsi :

```
ref (condition) libre ;
libre :- new condition (this monitor) ;
```

c) Les conditions avec priorités et délais (DUFF)

Le mécanisme précédent s'avère insuffisant dès lors que l'on se préoccupe du phénomène de préemption, pour des processus avec priorités.

Dans ce cas, un processus qui entre dans un moniteur pour demander une ressource doit pouvoir l'obtenir même si celle-ci est déjà occupée par un autre processus, à condition que sa priorité soit plus forte. Bien sûr, le deuxième processus doit être mis en attente derrière une condition avec priorité et le premier processus doit acquérir la ressource. Dans le contexte de quasi-parallélisme, on supposera, et ceci est une restriction importante, qu'un processus qui tient une telle ressource se trouve toujours dans l'échéancier SQS.

On est donc amené à introduire des conditions particulières, avec priorités et délais, appartenant à la classe conditionp, dont nous donnons la structure :

```
(1)   class conditionp (m) ; ref (monitor) m ;
      begin
(2)     ref (head) waitq ;
(3)     procedure cwait (pr,p) ; ref (process) pr ;
                  real p ; ... ;
(4)     procedure csignal ; ... ;
(5)     procedure waitp (pr, p, dt) ; ref (process) p ;
                  real p, dt ; ... ;
(6)     boolean procedure cempty ; ... ;
(7)     ref (process) procedure cfirst ; ... ;
(8)     waitq :- new head ;
      end ; ¢ conditionp ¢
```


(1) déclaration de l'entête de la classe conditionp, avec priorités et délais, associée au moniteur m ;

(2) déclaration de la file waitq dont les éléments, ordonnés par priorités décroissantes, appartiennent à la classe :

```
link class élément (pr, p, dt) ;
ref (process) pr ; real p, dt ; null ;
```

où pr est un processus, p sa priorité et dt le temps résiduel (délai) avant réveil (il est en effet nécessaire de conserver ce temps pour les processus qui sont préemptés et qui seront réordonnancés ultérieurement) ;

(3) l'appel de cette procédure permet de mettre l'élément (pr, p, dt) à sa place dans la file waitq, pr est un processus, p sa priorité et dt est, soit égal à -1 pour le cas où pr est le processus courant, soit égal à pr.evtime-time (temps résiduel) pour un processus pr préempté ; le processus pr est sorti de la SQS si besoin est;

(4) l'appel de la procédure csignal permet de sortir l'élément situé en tête de la file waitq (donc un de ceux de plus forte priorité), et de réordonnancer son processus pr dans la SQS, de manière qu'il ne soit réveillé qu'après la durée dt (si dt > 0) ou 0 (si dt = -1, le processus signalant ne se bloquant dans urgentq (file des urgences) que dans ce cas (HOA74)).

(5) procédure de rangement appelée par cwait ;

(6) l'appel de cempty permet de savoir si waitq est vide ou non ;

(7) l'appel de cette procédure délivre le premier processus de waitq.

Ainsi, quand on veut utiliser le mécanisme de moniteurs pour protéger des objets, il suffit de préfixer la classe servant à les décrire, par monitor. On a alors accès aux primitives enter et leave, et également la possibilité de créer des conditions.

IV.1.2 - PROCEDURES DE SERVICES

Un certain nombre de déclarations (de classes et de procédures) ont été définies afin de faciliter la gestion des différents mécanismes et outils de la simulation.

a) La classe date

link class date (jo,he,mi) ; integer jo,he,mi ; null

Un objet de cette classe représente une date sous la forme jours, heures, minutes.

b) La procédure lecturedate

procedure lecturedate (h,fin) ; boolean fin ;
real h ;....;

l'appel de cette procédure provoque la lecture (sur carte) d'une date (objet de la classe date), h est un réel représentant la date convertie en nombre d'heures et fin indique si oui ou non on est à la fin de fichier (repérée par une '*').

c) La procédure écri

procedure écri (t) ; real t ;....;

l'appel de cette procédure permet d'écrire en clair (c'est à dire sous la forme jours, heures, minutes) une date t donnée en nombre d'heures).

d) Les procédures de conversions

ref (date) procedure convnh (nh) ; real nh ;...;

permet de convertir une date nh en nombre d'heures en une occurence de la classe date (dtra) référenciée par convnh.

real procedure convjhm (da) ; ref (date) da ;...;

convertit une date (j, h, m) en un nombre d'heures.

IV.2 - ASPECTS LOGIQUES

Nous scindons cette présentation en trois parties :

les structures de données,
les processus,
et la prise en compte du temps.

IV.2.1 - LES STRUCTURES DE DONNEES

Comme cela a été dit au paragraphe II.1.1., il est possible de représenter n'importe quelle structure de données. Ces structures sont implémentées grâce aux classes de SIMULA préfixées, ou non, par la classe monitor suivant que l'on désire, ou non, y introduire les phénomènes de blocage.

Par exemple, dans cette réalisation, nous nous servons beaucoup de la structure de files (ou listes). Celle-ci est décrite par une classe donnant les attributs du type de file :

```
(1)     monitor class tfile (mesure, équilibre, inter) ;
        boolean mesure, équilibre ; real inter ;
        begin
(2)       ref (head) q ;
          ---
(3)       procedure entrer (t) ; ref (transaction) t ;... ;
(4)       procedure sortir (t) ; ref (transaction) t ;... ;
(5)       procedure majvol ;...;
(6)       procedure réinit ;...;
          ---
(7)       q :- new head ;
          ---
        end ; ¢ tfile ¢
```

La structure de cette classe est essentiellement voulue par les mesures statistiques ; on la détaille dans le chapitre VI.

La déclaration (2) et l'initialisation (7) servent à créer réellement la liste (file d'attente) concernée.

IV.2.2 - LES PROCESSUS SYNCHRONISES

Dans nos maquettes, un processus est un objet de la classe process ou d'une de ses sous-classes. Par exemple, il peut appartenir à la classe processus dont la structure est :

```
process class processus (nom, priorité, t) ;
text nom ; real priorité,t ; null ; $ processus $
```

où nom est l'identification de ce processus, priorité sa priorité éventuelle, et t un paramètre de sa durée de travail.

Nous avons vu que la synchronisation des processus se fait en général par des moniteurs qui assurent le partage des données qu'ils protègent. En particulier, un processus peut être activé par l'intermédiaire d'un moniteur appelé moniteur d'activation. Les signaux de déclenchement des processus de traitement sont rangés dans des boîtes ou dans des files d'attente protégées par ce mécanisme de moniteur d'activation. Nous avons ressenti le besoin de créer trois types de moniteurs d'activation :

- Le type 1 définit des moniteurs d'activation qui ne mémorisent pas les signaux qui arrivent quand il n'y a aucun processus en attente de signal.

- Les moniteurs d'activation de type 2 mémorisent tous les signaux de déclenchement, même si aucun processus n'attend de signal. Ils sont dits "à mémoires multiples".

- Enfin le type 3 introduit des moniteurs d'activation qui ne mémorisent qu'un seul signal d'activation à la fois, qu'il y ait ou non un processus en attente de signal. Ils sont dits à "mémoire unique".

Le choix du type de moniteur d'activation à utiliser dépend en général, du type de structure d'information sur laquelle travaille le processus à activer. Nous développerons cet aspect par la suite.

Le fonctionnement d'un moniteur d'activation peut se schématiser ainsi :

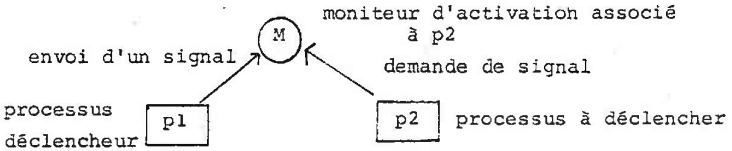


figure 12

La structure de la classe définissant les moniteurs d'activation est la suivante :

```
(1)     monitor class actmonitor (type) ; integer type ;
        begin
(2)       ref (head) actq ;
(3)       ref (condition) déblocage ;
(4)       procedure acquérir (d) ; name d ;
                    ref (message) d ;...;
(5)       procedure envoyer (d) ; ref (message) d ;...;
          actq :- new head ;
          déblocage :- new condition (this monitor)
        end ; $ actmonitor $
```

(1) entête de la classe du moniteur d'activation du type 1, 2 ou 3 ;

(2) actq est la file (dans le cas 2) ou la boîte (dans les cas 1 et 3) qui mémorise les messages d'activation (ou signaux), objets de la classe message

```
link class message (no) ; integer no ; null ; $ message $
```

où no est un numéro d'indentification de message ;

(3) la condition déblocage sert à bloquer les processus en attente d'un signal d'activation ;

(4) l'appel de cette procédure par un processus provoque l'activation de celui-ci avec le message d, s'il existe un élément dans la file (ou boîte) actq, et bloque le processus sinon ;

(5) l'appel de cette procédure par un processus provoque l'envoi d'un message (ou signal) d'activation dans la file (ou boîte) du moniteur d'activation du processus à activer. A ce niveau on mémorise ou non le message en fonction du type du moniteur d'activation.

Ainsi, si un processus p1 de la classe a veut activer un autre processus p2 de la classe b choisie cyclique, il lui faut envoyer un message d dans la file d'un moniteur d'activation actm associé à p2. La structure des processus est alors la suivante :

```
processus class a ;
begin
   ---
   actm.envoyer (d) ;
   ---
end ; ∮ a ∮
```

```
processus class b ;
begin
   while true do
   begin
      ---
      actm.acquerir (d) ;
      ---
   end ;
end ; ∮ b ∮
```

IV.2.3 - PRISE EN COMPTE DU TEMPS

Nous avons vu, dans la présentation de SIMULA, que l'on dispose d'un échéancier interne : la SQS. Une restriction importante de celui-ci est qu'il n'est pas possible d'y insérer, à un moment donné, plusieurs "notices d'événements" associées à un même processus. Or il nous paraît très utile de pouvoir disposer de plusieurs notices, notamment pour des processus dont on sait d'avance qu'ils doivent être activés à dates déterminées. Nous introduisons donc dans le système, un échéancier général secondaire, sorte de fusion de divers "échéanciers privés", associés chacun à un processus, et contenant ses dates d'activation.

Cet échéancier général est cyclique car, dans les problèmes de simulation de systèmes d'information en organisations, la plupart des procédures se reproduisent de manière répétitive.

Un processus d'activation alpha, associé à cet échéancier, permet d'envoyer des signaux aux moniteurs d'activation des processus concernés afin de les débloquer : ce processus d'activation scrute continuellement l'échéancier et gère automatiquement les changements de cycles.

Nous donnons ci-après la structure de la classe échéancier.

```
(1)   class échéancier ;
      begin
(2)      ref (head) echq ;
(3)      integer per, hdp ;
(4)      ref (activation) alpha ;
(5)      procedure viderech (a) ; ref (actmonitor) a ;...;
(6)      procedure modifech  (a) ; ref (actmonitor) a ;...;
(7)      ref (process) procedure prochevnotice ;...;
(8)      echq :- new head ;
(9)      alpha :- new activation (this échéancier) ;
      end ; ∮ échéancier ∮
```

(1) entête de la classe échéancier ;

(2) déclaration de la liste echq (échéancier) dont les éléments, ordonnés par dates croissantes, appartiennent à la classe evnotice :

```
link class evnotice (datact, actm) ;
real datact ; ref (actmonitor) actm ; null ;
```

où datact est la date d'activation du processus associé au moniteur d'activation actm ;

(3) déclaration des variables locales à l'échéancier : per est la longueur d'un cycle en nombre entier d'heures, hdp est l'heure du début du cycle en cours ;

(4) déclaration du processus scrutateur alpha ;

(5) l'appel de viderech (a) permet de supprimer de l'échéancier toutes les notices evnotice associées au moniteur d'activation a ;

(6) l'appel de modifech (a) permet de modifier l'échéancier par substitution aux anciennes evnotices associées au moniteur d'activation a, des nouvelles evnotices dont les dates d'activation sont lues sur cartes ; au cas où il n'existe pas encore d'evnotices associées à a, cette procédure est équivalente à une adjonction avec interclassement.

Ces deux procédures permettent donc la construction et la mise à jour de l'échéancier. Elles réordonnancent le processus scrutateur alpha.

L'utilisation du mécanisme d'échéancier est possible après avoir procédé à sa création :

```
ref (échéancier) éché ;
éché :- new échéancier ;
```

IV.3 - ASPECTS PHYSIQUES

Comme cela a déjà été dit, ces aspects regroupent les notions de ressources et de leur allocation.

VI.3.1 - LES RESSOURCES

Nous avons vu en II.2.2. que notre étude est limitée à un certain nombre de types de ressources dont nous donnons le détail.

a) Les ressources critiques simples

Une ressource critique peut être assimilée à un booléen, appelé occupé, protégé par un moniteur, avec une condition appelée libre. Un processus demandant cette ressource se bloque sur la condition libre si la ressource est occupée.

Quand un processus libère la ressource, il signale aux éventuels processus en attente qu'il n'a plus l'utilité de la ressource, afin de débloquer l'un d'entre eux : le premier dans la file associée à la condition libre (politique FIFØ ou PAPS). La classe prédéfinie calrc a la structure suivante :

```
(1)     calendrier class calrc ;
        begin
(2)        boolean occupé ;
(3)        ref (condition) libre ;
(4)        procedure acquérir ;...;
(5)        procedure libérer ;...;
(6)        libre :- new condition (this monitor) ;
           ---
(7)        occupé :- false ;
        end ; ¢ calrc ¢
```

(1) entête de la déclaration de la classe définissant la ressource critique (le mécanisme de calendrier et son utilisation seront définis au paragraphe suivant) ;

(2) occupé indique si la ressource est occupée à un instant donné ;

(3) la condition libre bloque le processus demandant la ressource si elle est occupée ;

(4) par l'appel de cette procédure, un processus peut acquérir la ressource si occupé est faux ; sinon, il se bloque dans libre ;

(5) l'appel de cette procédure libère la ressource et signale la condition libre, afin de donner le contrôle à l'éventuel premier processus de la file d'attente.

On verra dans le paragraphe suivant (IV.3.2.) comment utiliser une telle ressource avec, ou non le mécanisme de calendrier. Il en est de même pour les autres types de ressources.

b) Les ressources critiques avec priorité

Ces ressources sont également implantées sous forme d'une classe à l'aide de deux éléments :

- le booléen occupé qui symbolise la ressource critique ;
- la condition libre sur laquelle les processus demandeurs de la ressource sont bloqués, si celle-ci n'est pas libre. Mais, à la différence de la ressource critique simple, il faut ici employer une condition avec priorité, de telle manière qu'on donne le contrôle de la ressource, une fois libérée, au processus ayant la plus grande priorité. La structure de la classe des ressources critiques avec priorité est la suivante :

```
(1)     calendrier class calrcp ;
        begin
(2)        boolean occupé ;
(3)        ref (conditionp) libre ;
(4)        procedure acquérir ;...;
(5)        procedure libérer ;...;
(6)        libre :- new conditionp (this monitor) ;
(7)        occupé := false ;
        end ; ∉ calrcp ∉
```

(1) entête de la classe définissant une ressource critique avec priorité ;

(2) booléen précisant si oui ou non la ressource est occupée ;

(3) condition avec priorité de blocage des processus demandant la ressource alors qu'elle est occupée ;

(4) demande d'acquisition de la ressource; si elle est occupée, on bloque le processeur demandeur dans libre ;

(5) tout processus appelant cette procédure libère la ressource et signale la condition libre, afin de donner la ressource au premier processus en attente ayant la plus forte priorité.

L'utilisation de ce type de ressources est également détaillée au paragraphe IV.3.2.

c) Les ressources critiques préemptibles avec priorités

Il s'agit, ici, de gérer une ressource critique, de telle manière que, même si elle est occupée par un processus, elle puisse être accordée à un processus demandeur, pourvu qu'il ait une priorité plus élevée. Le premier processus est alors préempté et mis en attente dans la file d'une condition avec priorités et délais. Lors d'un signal sur cette condition, le processus, à qui on accorde la ressource ainsi libérée, est le premier parmi ceux ayant la plus forte priorité.

La structure de la classe de ressources critiques, avec priorité préemptibles calrcpp est la suivante :

```
(1)     calendrier class calrcpp ;
        begin
(2)        ref (processus) occupant ;
(3)        ref (conditionp) tour ;
(4)        procedure acquérir ;...;
(5)        procedure libérer ;...;
(6)        occupant :- none ;
           tour :- new conditionp (this monitor)
        end ; ∉ calrcpp ∉
```

(1) entête de la classe calrcpp ;

(2) occupant est le processus qui utilise actuellement la ressource critique ; c'est un objet de la classe processus.

(3) tour est la condition avec priorité et délais permettant de bloquer les processus non prioritaires ;

(4) l'appel de acquérir donne la ressource critique au processus le plus prioritaire (entre l'occupant et le processus qui a appelé la procédure), et met l'autre en attente dans tour ;

(5) l'appel de libérer libère la ressource et la donne au premier processus de tour, s'il existe ;

(6) occupant est initialisé à none, adresse de l'objet vide ;

d) Les ressources à accès multiples

L'attribut quantité d'une telle ressource en fixe le nombre d'accès. Par exemple, un service de perforation comportant trois personnes interchangeables a une quantité égale à 3. Un processus peut demander un ou plusieurs accès à cette ressource.

* Dans le cas d'une ressource à accès multiple, pour laquelle un processus peut demander le nombre d'accès ($\geqslant 1$) qu'il désire, la condition ressuffi bloque effectivement les processus dont le nombre d'accès demandé est supérieur au "stock" d'accès restants. Le problème le plus important, pour ce type de ressource, se pose lors de la libération d'un certain nombre (n) d'accès par un processus. A ce moment, il faut donner des accès (suivant le nombre demandé) aux processus en attente dans la condition ressuffi, dans la limite de la quantité nouvelle disponible (ancienne quantité en stock plus quantité libérée). Pour cela, il suffit de réveiller chacun des processus bloqués sur ressuffi afin d'allouer le plus possible d'accès pour optimiser le nombre de processus travaillant avec cette ressource. La structure de la classe calram est la suivante :

```
(1)   calendrier class calram (quantité) ; real quantité ;
      begin
(2)      ref (condition) ressuffi ;
(3)      procedure acquérir (n) ; real n ;...;
(4)      procedure libérer (n) ; real n ;...;
         ---
(5)      ressuffi :- new condition (this monitor) ;
      end ; ∅ calram ∅
```


(1) entête de la classe calram, quantité étant le nombre d'accès libres à un instant donné (total à l'initialisation) ;

(2) condition de nombre d'accès insuffisant, sur laquelle se bloquent les processus demandant la ressource tant qu'il n'y a pas assez d'accès pour qu'ils continuent ;

(3) l'appel de cette procédure permet au processus appelant de disposer de n accès si ils sont libres, sinon il est bloqué sur ressuffi ;

(4) l'appel de cette procédure provoque la libération de n accès et le signale à tous les processus en attente sur ressuffi afin d'en réveiller le maximum.

Seul ce type de ressources à accès multiples a été réalisé pour notre première application. Nous allons étudier maintenant comment on pourrait réaliser d'autres types de ressources à accès multiples et quelles seraient les difficultés rencontrées.

* Nous n'avons décrit jusqu'ici, que les ressources à accès multiples, sans priorités ni préemption. Ceci est dû au fait que nous n'avons réalisé que les ressources utiles en priorité. C'est ainsi que les ressources à accès multiples avec priorité et avec préemption n'ont pas été implémentées. Pour ce type de ressources, il existe des problèmes de choix dans la politique d'activation des processus mis en attente, que nous illustrons dans l'exemple ci-dessous :

Soit une ressource à accès multiple, avec priorité, préemptible, ayant comme quantité 5, et les processus :

p1, utilisant 1 accès avec priorité 10,
p2, utilisant 2 accès avec priorité 9,
p3, utilisant 6 accès avec priorité 2,
p4, utilisant 4 accès avec priorité 1,

à un moment donné, de telle manière qu'il reste deux accès non utilisés.

Si, à cet instant, un processus p5 demande 8 accès avec priorité 8, il faut préempter p4 et p3 pour lancer p5, mais il reste alors 4 accès non utilisés, donc on pourrait relancer p4, et ainsi de suite ...

On pense qu'une telle ressource est réalisable avec un mécanisme de ramasse-miettes. Quoiqu'il en soit, n'en ayant pas l'utilité, cette ressource n'a pas été implémentée.

★ Un type particulier de ressources à accès multiples a été réalisé afin de disposer de processeurs disponibles pour n'importe quel processus (cf. II.2.2.a.2). Ces ressources sont dites à accès multiples infinis. Elles ne servent qu'à comptabiliser des temps de réponse. Ce sont des objets de la classe calraminf dont la structure est :

```
(1)     calendrier class calraminf ;
        begin
(2)        procedure acquérir ;...;
(3)        procedure libérer ;...;
        end ; $ calraminf $
```

(1) entête de la classe préfixée par la classe calendrier afin de pouvoir utiliser ce mécanisme (Cf.IV.3.2.) ;

(2) (3) ces procédures ne comportent que des actions nécessaires aux prises de mesures.

e) Les processeurs partagés

Le problème est de définir un type de processeur qui peut être éventuellement partagé entre plusieurs processus, chacun travaillant pendant un certain quantum de temps, en exclusion mutuelle sur cette ressource. Le processeur est donc une ressource critique (booléen occupé) protégé par un moniteur, avec une condition libre.

La structure d'une classe calcpp de processeurs partagés est la suivante :

```
(1)     calendrier class calcpp (k) ; real k ;
        begin
(2)        boolean occupé ;
(3)        integer n ;
(4)        ref (condition) libre ;
(5)        procedure acquérir (x,qu) ; name qu ;
                     real x, qu ;...;
(6)        procedure libérer (x,qu) ; name x ;
                     real x, qu ;...;
(7)        occupé := false ;
(8)        n := 0 ;
(9)        libre :- new condition (this monitor) ;
        end ; $ calcpp $
```

(1) entête de la classe définissant un processeur partagé, dont k est l'intervalle de temps à partager ;

(2) occupé est un booléen indiquant si le processeur est utilisé à un instant donné ;

(3) n représente le nombre de processus qui se partagent le processeur à un instant donné ;

(4) condition sur laquelle se bloque les processus demandeurs, si le processeur n'est pas libre ;

(5) l'appel de cette procédure provoque, si le processeur est libre, l'occupation par le processus appelant, pour une durée égale à qu = inf (x, k/n) où x est la durée résiduelle du travail du processus ;

(6) l'appel de cette procédure provoque la libération de la ressource, la mise à jour du temps résiduel, et un signal sur la condition libre.

La procédure utiliserpp facilite l'utilisation de ce type de processeur :

```
procedure utiliserpp (t,pp) ; real t ;
            ref (calcpp) pp ;
begin
   real x, qu ;
   if t > 0
   then begin
      pp.n := pp.n + 1 ;
      x := t ;
      ---
      while x > 0 do
      begin
         pp.acquérir (x,qu) ;
         hold (qu) ;
         pp.libérer (x,qu)
      end ;
      ---
   end
end ; ¢ utiliserpp ¢
```

Ainsi, un processus, qui désire utiliser ce type de processeur, aura la structure suivante :

```
processus class p (t,pp) ; real t ;
            ref (calcpp) pp ;
begin
   ---
   while true do
   begin
      ---
      utiliserpp (t,pp) ;
      ---
   end ;
end ; ¢ p ¢
```


IV.3.2 - UTILISATION DES CALENDRIERS

a) Réalisation du mécanisme de calendrier

La nécessité de ce mécanisme se fait sentir par le fait que certains processeurs ne sont pas toujours disponibles pour des processus dans la maquette qu'on réalise. On peut définir, pour un processeur donné, des périodes de disponibilité, dont les dates de début et de fin sont notées dans un calendrier propre à ce processeur. Ce calendrier comporte les dates délimitant des périodes sur un cycle : tout comme l'échéancier présenté précédemment, les calendriers sont gérés de manière cyclique.

Ils sont déclarés comme étant des moniteurs, et ce sont eux qui, globalement, protègent un processeur qui utilise ce mécanisme. Afin de standardiser les notations, nous introduisons un paramètre b, booléen qui précise si oui ou non on désire utiliser le mécanisme de calendrier pour un processeur donné.

Le principe de fonctionnement est le suivant : on dispose de deux files :

- notcalq est la liste des notices du calendrier, objets de la classe :

```
link class notcal (début, fin) ; real début, fin ; null ;
```

qui représentent les limites des périodes de disponibilité des ressources ;

- noticeq est la liste des processus qui sont passés dans le calendrier (et qui utilisent, ou tentent d'utiliser une ressource protégée) avec une notice de la classe :

```
link class notice (pr, dt) ; ref (processus) pr ;
            real dt ; null ;
```

où pr est le processus utilisant la ressource et dt est égal à -1, pour un processus qui n'a pas encore travaillé avec la ressource, et égal à la durée résiduelle d'exécution pour un processus interrompu en fin de période de disponibilité de la ressource.

Une condition travail bloque les processus en dehors des périodes de disponibilité ; un booléen heuretravail indique si, à un instant donné, on se trouve dans une période de disponibilité de la ressource.

Pour gérer les différentes périodes de disponibilité, on dispose de deux processus de scrutation de notcalq :

- le premier est appelé processus de désactivation ; il est du type :

process class désact (cal) ; ref (calendrier) cal ;

il est chargé, en fin de période de disponibilité de la ressource protégée par le calendrier cal, de désactiver les processus qui utilisent cette ressource en la leur retirant, et en calculant leur durée résiduelle d'exécution ; il est activé à la fin de chaque période de disponibilité de la ressource concernée ;

- le second est appelé processus d'activation ; il est du type :

process class act (cal) ; ref (calendrier) cal ;

il est chargé, en début de périodes de disponibilité, de redonner le contrôle de la ressource aux processus qui se sont faits bloquer en fin de période, et ceci pour une durée correspondant au temps résiduel ; il est en outre chargé de faire un signal (csignalg) sur la condition travail afin de donner la ressource à d'éventuels processus arrivés pendant que celle-ci n'était pas disponible (cas où heuretravail := false). Enfin, ce processus gère la reproduction des périodes de disponibilité sur les différents cycles.

Les processus d'activation et de désactivation procèdent également à la mise à jour du booléen heuretravail.

La structure de la classe calendrier est la suivante :

```
(1)   ressource class calendrier (b, termine) ;
      boolean b, termine ;
      begin
(2)      ref (head) noticeq, notcalq ;
(3)      ref (condition) travail ;
(4)      integer jdp ;
(5)      boolean heuretravail ;
(6)      procedure calconsulter ;...;
(7)      procedure calsortir ;...;
(8)      procedure constcalendrier ;...;
         ---
(9)      if b then begin
                   noticeq :- new head ;
                   notcalq :- new head ;
                   travail :- new condition (this monitor) ;
                   constcalendrier
                        end
                   else heuretravail := true ;
      end ; ¢ calendrier ¢
```

(1) entête de la classe calendrier préfixée par la classe

```
monitor class ressource ;...;
```

définissant des procédures de prises de mesures statistiques (cf. Chapitre VI) ; b est un booléen précisant si oui ou non on désire utiliser le mécanisme de calendrier ; et le booléen termine indique que le processus de désactivation associé au calendrier doit laisser les processus terminer leur travail en cours en fin de période de disponibilité de la ressource concernée (en effet, dans la réalité, il est impensable (sauf peut être lors de la frappe d'une thèse...) qu'une secrétaire interrompe brutalement son travail au beau milieu d'une phrase). Ces booléens seront initialisés suivant les besoins de l'utilisateur ;

(2) déclaration des deux listes ;

(3) condition de blocage en dehors des périodes de disponibilité ;

(4) jdp est utilisé pour définir le jour de début d'un cycle ;

(5) heuretravail indique si on se trouve, ou non, dans une période de disponibilité à un instant donné ;

(6) l'appel de calconsulter par un processus permet, éventuellement de bloquer celui-ci sur travail ; quand il sera débloqué, on créera, pour lui, une notice dans noticeq ;

(7) l'appel de calsortir fait sortir de noticeq la notice qui correspond au processus invocateur ;

(8) cette procédure construit le calendrier associé à une ressource en lisant, sur cartes, les dates de début et de fin des périodes de disponibilité ; cette liste de dates est rangée dans notcalq ; de plus cette procédure met à jour jdp et lance les processus d'activation et de désactivation ;

(9) Au cas où b est vrai, on initialise les attributs de la classe calendrier avant de construire le calendrier lui-même.

b. Utilisation des ressources

Chacun des types de ressources présentés dans ce chapitre est utilisable avec, ou sans le mécanisme de calendrier. Le principe des classes hiérarchisées :

```
link       class monitor (nom) ;...;
monitor    class ressource ;...;
ressource  class calendrier (b, termine) ;...;
calendrier class calrc ;...;
calendrier class calrcp ;...;
calendrier class calrcpp ;...;
calendrier class calram (q) ;...;
calendrier class calraminf ;...;
calendrier class calcpp (k) ;...;
```


permet de créer des occurences (objets) de chacunes des classes de ressources avec comme paramètres (attributs) dans l'ordre :

- un nom (du type texte) ;

- un booléen b (utilisation ou non du mécanisme de calendrier) ;

- un booléen termine (laisser ou non un processus terminer son travail en fin de période) ;

- pour les ressources à accès multiples : le nombre d'accès ; pour les processeurs partagés : l'intervalle de temps à partager.

Ainsi, une ressource à accès multiples, appelée ram, utilisant un calendrier, ne permettant pas aux processus de terminer leur travail en fin de période et ayant 4 accès est déclarée et créé ainsi :

```
ref (calram) res ;
res :- new calram ('ram', true, false, 4) ;
```

Après avoir détaillé chacun des composants intervenant dans nos maquettes, nous allons montrer comment les utiliser plus précisément afin d'effectuer une simulation d'un système. Un exemple de taille importante est présenté afin d'aider à la compréhension de l'utilisation des mécanismes.

CHAPITRE V

ECRITURE D'UNE MAQUETTE

CHAPITRE V -

ECRITURE D'UNE MAQUETTE

Le but de ce chapitre est de présenter une méthode d'écriture de maquettes. Pour cela, nous montrons comment utiliser chacun des composants décrits au chapitre IV, quelles sont les contraintes à respecter, puis, pour illustrer le tout, nous détaillons une application de taille importante où entrent en jeu la plupart des notions définies.

V.1 - METHODE POUR L'ECRITURE DE MAQUETTES

Après avoir présenté au chapitre IV les différents composants d'une maquette, nous voyons comment on peut les utiliser.

Toutes les notions introduites ont été réalisées sous forme de classes hiérarchisées et de procédures SIMULA. Elles sont toutes regroupées dans une classe unique : la classe MAESTRO. Grâce aux possibilités de précompilation, avec création de prologue de la nouvelle version SIMULA mise au point par l'IRIA et la CII-HB, la classe MAESTRO est cataloguée en bibliothèque. Puisque cette classe est préfixée par la classe SIMULATION, classe système de SIMULA (cf. III.1.2.), en préfixant son programme de simulation par la classe MAESTRO, l'utilisateur a accès à tous les attributs, des classes systèmes de SIMULA et des notions nouvelles de MAESTRO. La structure de la classe prédéfinie MAESTRO est la suivante :

Remarque : afin d'en alléger le texte, on ne rappelle pas ici les paramètres de chacune des classes ou des procédures, le lecteur pourra se reporter aux chapitres IV et VI où il trouvera tous les renseignements nécessaires.

Ne figurent ici que les composants de la classe MAESTRO utiles à l'écriture d'une maquette. Toutes les classes et procédures internes aux composants non indispensables à l'utilisateur ne sont pas décrites ici.

Pour de plus amples renseignements, l'annexe 3 donne la liste des mots réservés de la classe MAESTRO afin d'éviter toute double déclaration involontaire.

```
simulation class maestro
begin
   link class monitor ;...;
   class condition ;...;
   class conditionp ;...;
   monitor class actmonitor ;...;
   class échéancier ;...;
   ---
   calendrier class calrc ;...;
   calendrier class calrcp ;...;
   calendrier class calrcpp ;...;
   calendrier class calram ;...;
   calendrier class calraminf ;...;
   calendrier class calcpp ;...;
   procedure utiliserpp ;...;
   ---
   link class trégion ;...;
   link class tfile ;...;
   message class transaction ;...;
   ---
   process class processus ;...;
   procedure passe ;...;
   procedure simul ;...;
   ---
   inner
end ; $ maestro $
```

Un listing représentant cette classe est donné en annexe 2.

Nous reprenons le principe de séparation logique/physique pour analyser l'utilisation de chacun des composants.

V.1.1 - LES OUTILS GENERAUX

Comme nous l'avons dit, si l'utilisateur désire inclure des phénomènes de blocage dans ses structures de données, il peut préfixer les déclarations de classes par le nom de classe monitor. Toutefois il faut remarquer que s'il n'existe pas de blocage de processus (donc ici pas de HOLD) à l'intérieur du moniteur (ce qui arrive fréquemment en quasi-parallélisme), il n'a pas besoin d'utiliser ce mécanisme sinon pour une question de normalisation des notations.

V.1.2 - ASPECTS LOGIQUES

L'utilisateur a toute liberté pour créer les types de structure de données qu'il désire.

S'il veut créer des files d'attente, il peut utiliser la classe prédéfinie tfile. Ainsi, par la même occasion, il peut obtenir, de manière automatique, des mesures sur ces files (cf. chapitre 6).

Les éléments pouvant faire partie de ces files sont des objets de la classe transaction dont la structure est :

```
message class transaction (pds) ; real pds ;...;
      $ transaction $
```

où pds est un poids et où le corps de classe concerne des initialisations de mesures. La classe message ayant comme paramètre un numéro de message (entier), toute transaction doit avoir un numéro et un poids.

Comme nous l'avons dit en IV.2.2., un processus de la maquette peut faire partie de la classe processus dont les paramètres sont : un nom, une priorité et un paramètre de la durée d'exécution. Afin d'utiliser les mécanismes de synchronisation par moniteurs d'activation, la structure d'un processus p peut être la suivante :

```
(1)     processus class p ;...;
        begin
        ---
(2)       while true do
(3)       begin
(4)         actm.acquérir (d) ;
(5)         ---
          end
        end ; $ p $
```

Le processus est cyclique dans ce cas et pour pouvoir exécuter son corps (5), il doit attendre un message dans un moniteur d'activation actm. Ce message peut être envoyé, soit par un autre processus par :

```
actm.envoyer (d),
```
(*)

soit par l'intermédiaire du mécanisme d'échéancier. C'est alors le processus dit d'activation qui se charge de cet envoi.

Pour utiliser l'échéancier d'activation des processus, après l'avoir créé (cf.IV.2.3.), on doit y placer les différentes notices de calendrier propres à chaque processus. Pour cela, on doit d'abord déclarer et créer un moniteur d'activation actm qui est réservé à l'activation d'un (ou plusieurs) processus (s'ils ont les mêmes échéances d'activation) :

```
ref (actmonitor) actm ;
actm :- new actmonitor ('nom', type) ;
```

L'échéancier étant créé et appelé éché dans la classe MAESTRO, on y place les notices d'activation en appelant la procédure de construction :

```
éché.modifech (actm) ;
```

(*) dans ce cas on peut parfois assimiler le signal d'activation à un message utilisable par le processus p. (voir V.2.)

Comme nous l'avons vu en IV.2.3., cette procédure a pour but de construire l'échéancier privé associé au moniteur d'activation actm en lisant sur cartes les horaires d'activation ; ainsi, par exemple, si le processus p doit être activé à 10 h puis à 15h 30, la carte de donnée a la forme suivante :

```
0  10  0  0  16  30 *
```

Les dates y sont inscrites sous la forme jours, heures, minutes, la suite de dates se terminant par le caractère '*'.

Dans le cas de l'utilisation du mécanisme de l'échéancier pour un processus, celui-ci doit être conçu de manière cyclique.

V.1.3 - ASPECTS PHYSIQUES

L'utilisation de ressources dans une maquette est simplifiée de par la présence, dans la classe MAESTRO, d'un certain nombre de type prédéfinis. Ce sont :

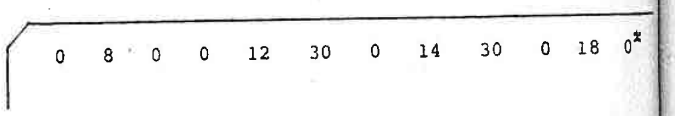
- les ressources critiques simples (calrc) ;
- les ressources critiques avec priorité (calrcp)
- les ressources critiques avec priorité préemptibles (calrcpp) ;
- les ressources à accès multiples (calram) ;
- les ressources à accès multiples infinis (calraminf) ;
- les processeurs partagés (calcpp).

Toutes ces ressources peuvent utiliser le mécanisme de calendrier. Pour cela, il faut, lors de la création effective de l'objet ressource de la classe concernée, passer en paramètre la valeur vraie pour le booléen b.

Par exemple :

ressource 1 :- new calrc ('ressource 1', true, false)

initialise une ressource critique avec calendrier qui ne laisse pas les processus terminer leur travail en fin de période. Cette initialisation provoque, si b est vrai, la construction du calendrier associé à la ressource par lecture des notices de calendriers sur cartes. Ainsi, le processeur de type calrc ci-dessus étant libre de 8h à 12h 30 et de 14h 30 à 18h, la carte de données nécessaire à la construction du calendrier est :



On remarque, qu'ici aussi, les dates sont données en clair (jours, heures, minutes) et que la liste se termine par une étoile.

Nous rappelons que l'utilisateur, et lui seul en est reponsable, doit faire attention à plusieurs points :

- aucun processus ne peut utiliser plus d'une ressource (processeur) à la fois, sauf s'il n'existe aucun risque de blocage ;

- l'utilisation de l'option termine des calendriers est à sa seule appréciation. En effet, il serait maladroit de laisser terminer son travail en fin de période de disponibilité de la ressource r à un processus dont la durée d'exécution est fixée, par exemple, à 24 heures, si la ressource r n'est disponible que pendant 1 heure dans la journée. La répercussion sur le fonctionnement du système pourrait être grave.

Un exemple d'utilisation d'un processeur partagé figure en annexe 4.

V.1.4 - EXECUTION D'UNE SIMULATION

Après avoir déclaré, créé et initialisé tous les constituants de sa maquette, suivant les principes énoncés dans les trois premiers paragraphes de ce chapitre, l'utilisateur peut lancer l'exécution de la maquette décrite.

Pour cela, il dispose de la procédure simul dont la structure est :

```
(1)     procedure simul (duréésimul, editresgen, équi-
                         libre, bfgen) ;
(2)     real duréésimul ;
(3)     boolean editresgen, équilibre, bfgen ;
        begin
          ---
(4)       hold (duréésimul) ;
          ---
        end ; ∅ simul ∅
```

(1) entête de la procédure simul où duréésimul est un entier donnant la durée de la simulation (nombre entier d'heures) et où les booléens (3) servent à définir quels types de résultats (par des mesures) on désire obtenir (cf. chapitre VI).

(4) après avoir initialiser le système (pour les mesures), le hold provoque le départ de la simulation.

Grâce à cette procédure, l'utilisateur a la possibilité d'effectuer plusieurs simulations du même système. Elle lui permet également de ne pas tenir compte des mesures prises pendant une phase transitoire (début de la simulation). Il lui suffit de provoquer deux appels de simul ainsi :

```
simul (phase, false, false, false) ;
simul (duréesimul, editresgen, équilibre, bfgen) ;
          (cf. chapitre VI).
```

Pour illustrer ce paragraphe donnant la méthode d'utilisation de la classe MAESTRO, nous détaillons une application de taille importante (KLE78).

V.2 - APPLICATION - EXEMPLE DE MAQUETTE

L'application choisie est une partie du système de gestion d'une entreprise de taille moyenne. En raison du manque de mesures, donc de valeurs numériques précises sur l'entreprise, l'application réalisée ne se veut, dans un premier temps, que descriptive. Une fois tous les paramètres connus, elle pourra servir à l'évaluation du comportement dynamique du système étudié.

V.2.1 - LE CHOIX DU SYSTEME

L'entreprise choisie, en vue de la modélisation d'une partie de son système d'information, est une PME de distribution de la banlieue nancéienne. Succursale d'une grosse entreprise française, elle peut, du point de vue de sa gestion, être considérée comme une entité indépendante. Elle emploie environ cinquante personnes.

Nous avons choisi une entreprise moyenne de préférence à une grosse parce que la modélisation d'une grosse entreprise aurait accru la complexité de notre modèle, sans en accroître nécessairement l'intérêt. Pourtant, dans une PME, on se heurte parfois à une certaine difficulté pour définir la tâche des individus ou des groupes d'individus. Ceci influence bien sûr le niveau de détail de la description, mais comme on l'a déjà dit, un modèle global est parfois préférable à un modèle détaillé, notamment quand il s'agit d'une étude du comportement dynamique du sytème.

V.2.2 - ARCHITECTURE DE LA MAQUETTE

Ce paragraphe présente une maquette complète. Le choix du système à modéliser s'est porté sur le système de gestion des commandes (écrites et téléphonées) et sur la facturation. Les autres secteurs d'activité de l'entreprise ne sont envisagés : ils sont considérés, dans une première approximation, comme n'ayant pas d'interactions, pour ce qui concerne la dynamique, avec le système que nous avons retenu.

Nous présentons ici, une proposition de modélisation (dont on donne plus de détails dans (KLE78)). Il est évident qu'il peut en exister d'autres. On trouvera notamment dans (DUF79b) une autre forme de la modélisation d'une partie du même système.

a) Architecture générale

Notre système complet se décompose en cinq parties distinctes :

- la prise de commandes écrites,
- la prise de commandes téléphonées,
- le traitement administratif,
- le traitement physique,
- la facturation.

α) Prise de commandes écrites

A partir d'une liste de bons de commandes (lbc), la personne r codifie ces bons pour en déduire une liste de bons de commandes identifiés (lbci).

Elle les saisit sur ordinateur pour former une liste de commandes du jour (lcojo). Le schéma de ce traitement est donné figure 13. (*)

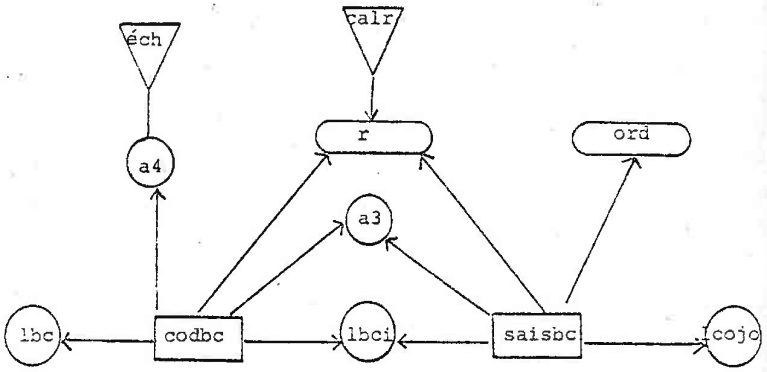


figure 13

β) Prise de commandes téléphonées

La personne v traite directement, à l'aide d'une console, l'appel téléphonique (bapl) qui est, soit une commande, soit une demande de renseignements. Elle en déduit, le cas échéant, une commande transmise à lcojo. Le schéma se trouve en figure 14.

(*) Pour des raisons de clarté des schémas de maquettes, les processeurs sont représentés par ( ) en remplacement du ◯ des moniteurs. Sur tous ces schémas figurent les échéanciers et calendriers détaillés aux paragraphes b) et c).

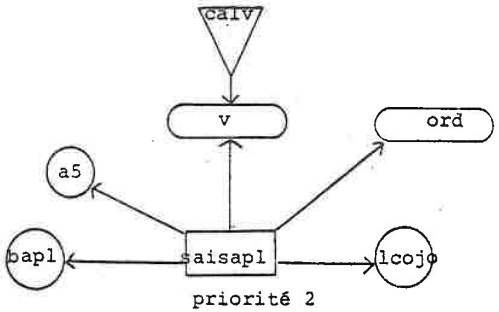


Figure 14

γ) Traitement administratif

Ce traitement est la partie la plus complexe de notre maquette. En effet, à ce niveau, les commandes initiales sont découpées en plusieurs parties :

- les lignes de commandes proposées à la livraison
- les lignes de commandes mises en attente (pour des raisons d'insuffisance de stock par exemple).

Ce travail est effectué par l'ordinateur qui, à partir d'une copie (lcojo') de la liste des commandes du jour (lcojo) et de la liste des commandes en attente des jours précédents (lcoat) établit une liste de propositions de livraisons (lpl) et une nouvelle liste de commandes en attente (nlcoat). Le détail de ce traitement est donné au paragraphe b) γ).

La deuxième partie de ce traitement administratif est effectuée par la personne v qui procède à une décision de livraison sur chacun des éléments de lpl, pour en déduire les propositions de livraison modifiées (lplm) ou, s'il n'y en a pas, pour lancer l'édition des bons de livraisons (lbl).

Si lplm n'est pas vide, v, à l'aide d'une console, effectue une saisie des modifications dont l'ordinateur tient compte (tout au moins dans la réalité) pour éditer de nouvelles propositions de livraisons qui commencent un autre cycle. Cette boucle proposition-décision-saisie ne peut s'effectuer au maximum que trois fois. Ces traitements sont également détaillés en b) γ), leur schéma est en figure 15.

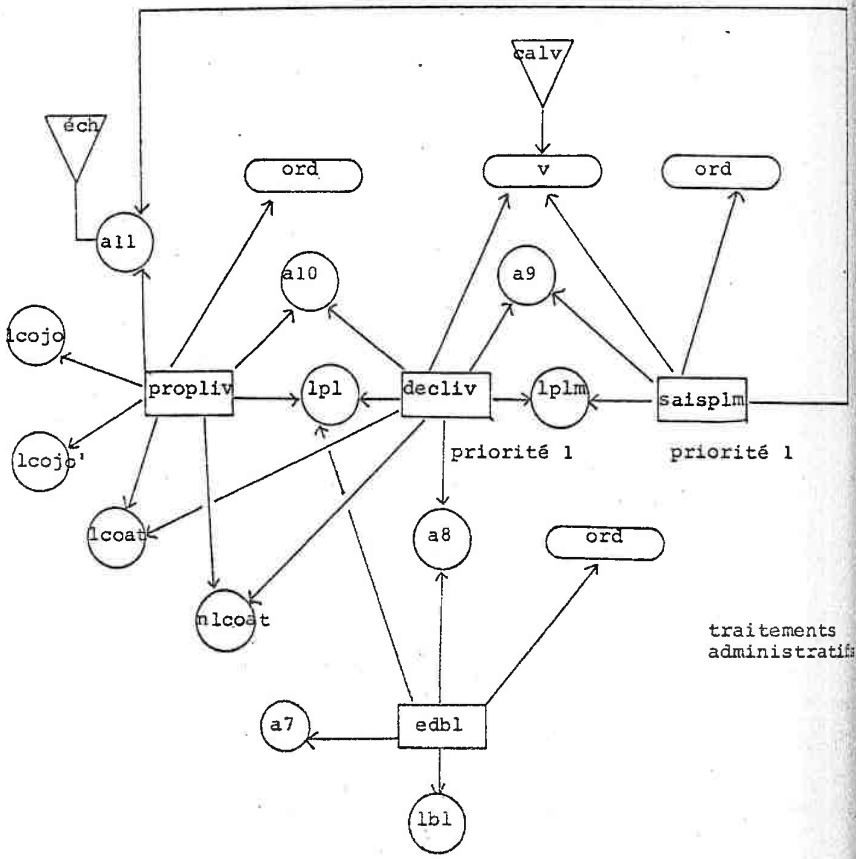


Figure 15

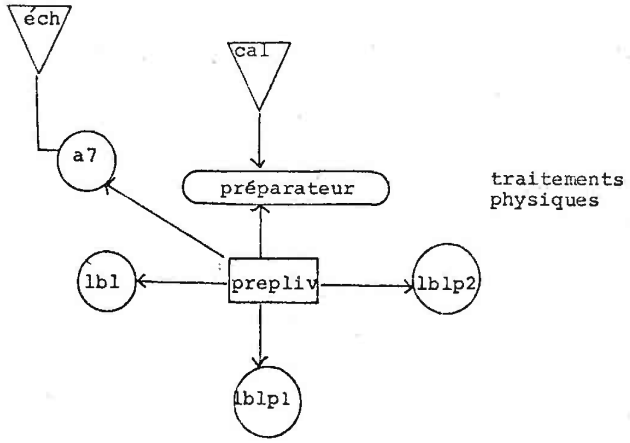


Figure 16

δ) Les traitements physiques

Ces traitements consistent en une préparation des livraisons par le magasin. A partir de la liste des bons de livraisons, le préparateur construit deux listes de bons de livraisons : d'une part pour l'envoi aux clients (lblp2), d'autre part, pour la suite des opérations administratives (lblp1). Le schéma est donné figure 16.

ε) La facturation

Ici encore, le traitement est divisé en quatre parties :

- décision de facturation,
- saisie des bons de livraison,
- facturation proprement dite,
- ventilation des factures.

La décision de facturation (essentiellement due aux délais de paiement et à l'introduction du port éventuel ...) est basée sur le même modèle que la proposition de livraison. C'est à dire qu'on décide ici, quelles sont de la liste des bons de livraison préparés (lblpl) et de la liste des bons de livraison non facturables (lblnf) des jours précédents, les factures qui pourront former la liste des bons de livraisons préparés (lblp). Cette décision est prise par la personne val qui saisit les éléments de lblp par une console pour en déduire la liste des bons de livraisons facturables (lblf). La modélisation est basée sur le même principe que pour les propositions de livraisons à la seule différence qu'ici on ne travaille plus au niveau de la ligne de commande mais au niveau de la commande elle-même. La facturation proprement dite, faite par l'ordinateur, consiste à établir une liste de factures (lf) à partir de la liste des bons de livraisons facturables (lblf). Cette liste de factures est alors ventilée par le service de facturation vers la comptabilité (lfcpta) et les clients (lfcl). Les schémas sont donnés en figures 17 et 18.

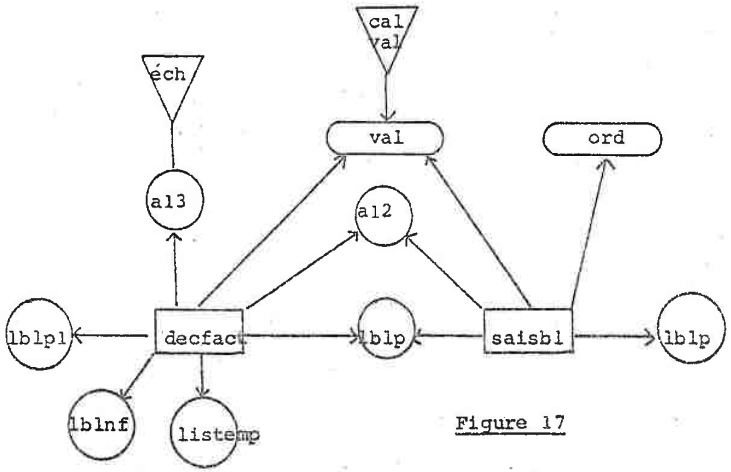


Figure 17

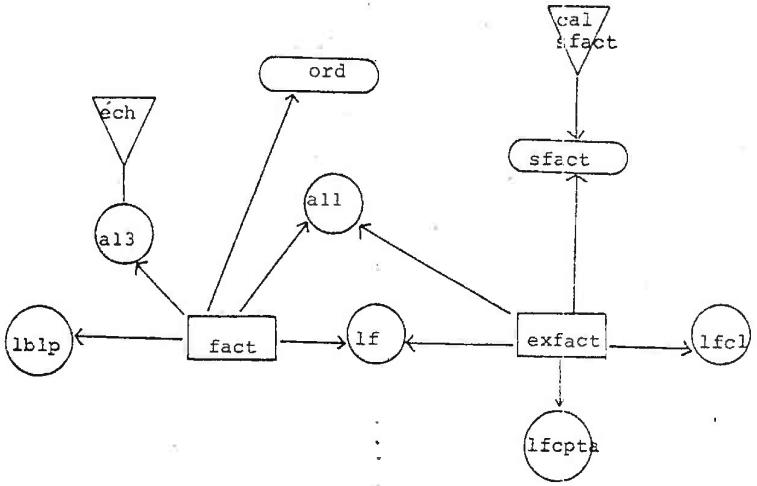
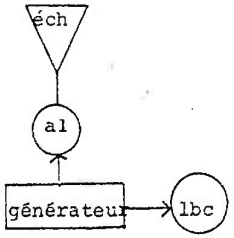


Figure 18

γ) Clôture du système

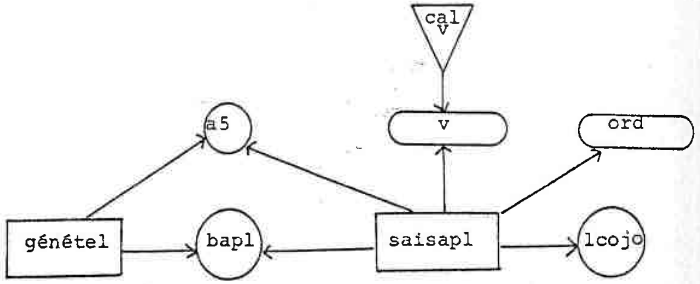
Nous complétons la maquette pour en faire un modèle de simulation clos. Ainsi, en entrée, nous sommes amenés à adjoindre un traitement assimilé à une arrivée de courrier. Il génère des bons de commandes afin d'alimenter la liste des bons de commandes (lbc). De plus, un générateur d'appels téléphoniques permet d'alimenter la boîte d'appels (bapl). Les caractéristiques techniques de ces traitements sont données au paragraphe b) γ). Toutefois nous devons attirer l'attention sur le fait que toutes les données (lois de services et temps de réponse) sur lesquelles nous travaillons sont très approximatives. Une étude plus approfondie, actuellement en cours (THA79) nous permettra de disposer de paramètres mieux fondés. Le but est ici de valider les outils que nous avons conçus et réalisés, d'apprécier leur puissance de description et leur efficacité.

La clôture du système en sortie est assurée par deux processus simulant l'envoi de courrier aux clients et au service de comptabilité. Ils n'ont pas un intérêt primordial pour la simulation du système sinon par le fait qu'ils vident les files lblp2, lfcl et lfcpta et ainsi nous évitent le risque de déborder de l'espace mémoire dont nous disposons. Les schémas de clôture du système sont donnés figure 19.



clôture en entrée

génération de commandes écrite



génération des commandes téléphonées

Clôture en sortie

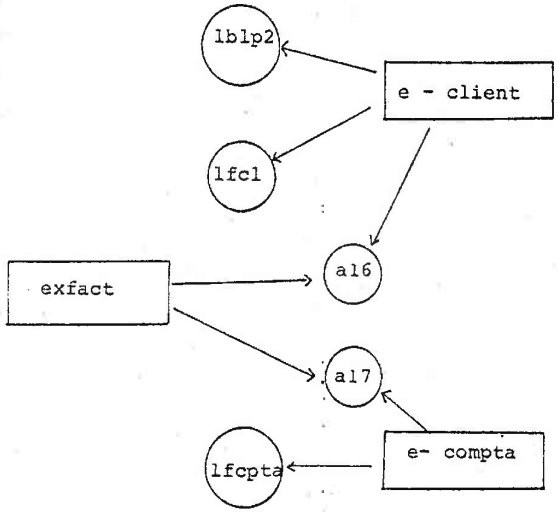


figure 19

b) Aspects logiques

Reprenant le plan adopté, nous étudions ici, les structures de données, les processus, l'intervention du temps au niveau logique dans notre maquette.

α) Les structures de données

Les données de notre système sont soit gérées par des files, soit par des boîtes. Elles peuvent être du type consommables ou réutilisables. Dans notre application nous pouvons considérer ces deux types comme formés :

- pour les ressources consommables, par le contenu :

. des files intermédiaires : lbc, lbci, lcojo, lpl, lplm, lbl, lblpl, lblp, lblf et lf.

. des boîtes ou files des moniteurs d'activation, et notamment de la boîte bapl pour laquelle on remarque que l'information (message) est assimilée au signal de déclenchement d'un processus.

- pour les ressources réutilisables : il s'agit de fichiers représentés aussi dans les maquettes par des files donnant l'une de leurs versions. Ce sont :

lcoat et lblnf pour les fichiers "permanents",

lcojo, listemp et nlcoat pour les fichiers "temporaires".

Les files lblp2, lfcl et lfcpta sont également considérées comme des fichiers à archiver ou à ventiler. Ce sont donc des ressources réutilisables.

Toutes ces files contiennent des messages (objets de sous-classes de la classe : message class transaction) qui peuvent être de trois types :

1. Des bons de commande (de la classe bc) comprenant la date d'arrivée dans le système, le numéro affecté à la commande, le type (écrite ou téléphonée) et le nombre de lignes commandées.

2. Des bons de commande en attente (de la classe bca) comprenant les mêmes éléments que bc avec en plus le nombre de lignes en attente.

3. Des bons de livraison (de la classe bcl) ayant les mêmes caractéristiques que bca avec en plus le nombre de lignes à livrer.

On retrouve les éléments de type bc dans les files lbc, lbci, lcojo et lcojo', ceux du type bca dans les files nlcoat et lcoat et ceux du type bcl dans les autres files. Une initialisation (avant toute simulation) de certaines files (fichiers "permanents" lcoat et lblnf) permet de simuler le fonctionnement du système. Celle-ci est effectuée à partir de mesures approximatives.

β) Les processus

Nous ne donnons pas ici de détails sur chacun des processus du système d'autant que certains ne font presque que du transfert d'information. Nous nous contentons de donner, dans le tableau 20 les caractéristiques essentielles des processus utilisés, en précisant pour chacun :

- un moniteur d'activation associé,
- sa durée d'exécution (loi des temps de service),
- son mode d'activation,
- la ou les ressources processeurs utilisés,
- sa priorité.

Comme on l'a indiqué, les lois de services ne sont ici qu'indicatives. Des études plus fines, dans l'organisation, doivent permettre de définir des lois de probabilités qui reflèteront mieux la réalité.

Le fonctionnement interne des processus est très souvent assez simple. Toutefois, des problèmes ont été rencontrés dans la modélisation des traitements administratifs (cf. (KLE78) § III.3.4.). C'est pourquoi nous rappelons ici le fonctionnement de cette partie de la maquette.

Le principe de base du travail de Propliv est l'hypothèse de conservation du flux des informations à travers ce traitement. Cela sous entend que si le cardinal de lcojo' est N, celui de lcoat étant M, on doit avoir la relation :

$$(a * N) + (b * M) = N = \text{cardinal de lpl}$$

où a et b sont des proportions de choix d'éléments dans lcojo et lcoat. En fixant un pourcentage, par exemple a, on obtient le second par la formule :

$$b = \frac{N * (1 - a)}{M}$$

NOM	moniteur d'activation associé	temps d'exécution	mode d'activation	ressources utilisées 1	2	priorité
générateur	a1 type1	0	échéancier à 9H30			1
génétel	/	0	processus de Poisson de moyenne $\lambda = \frac{18}{7}$ qd V travaille	(V)		1
codbc	a4 type1	50s/commande 25s/ligne	échéancier à 10H	r		1
saisbc	a3 type1	R:25s/commande 25s/ligne ord:7s/commande 7s/ligne	par codbc	r	ord	1
saisapl	a5 type1	2,5mn de conversation 30s/commande 25s/ligne	par génétel	v	ord	2
propliv	a11 type3	5s/ligne	échéancier à 15H ou par saisplm	ord		1
decliv	a10 type1	1ère fois : 20s/ligne 2ème fois : 5s/ligne 3ème fois : 1s/ligne	par propliv	v		1
saisplm	a9 type1	V:15s/ligne modifiée ord:7s/ligne modifiée	par decliv	v	ord	1
edbl	a8 type1	15mn	par decliv	ord		1
prepliv	a7 type1	2mn/ligne	échéancier à 9H ou par edbl	prepa		1
decfact	a13 type1	10s/commande 5s/ligne	échéancier à 14H	val		1
saisbl	a12 type1	val:15s/ligne 20s/commande ord:7s/ligne 7s/commande	par decfact	val	ord	1
fact	a15 type1	30mn	échéancier à 17H	ord		1
exfact	a14 type1	20s/facture	par fact	sfact		1
e-client	a16 type2	0	par exfact			1
e-compta	a17 type2	0	par exfact			1

figure 20 : tableau donnant les caractéristiques des processus de la maquette

Les coefficients a et b étant calculés au début du travail de propliv, on peut alors définir quelles sont les lignes à proposer et celles à refuser. Quand propliv a terminé son travail, il active le processus de décision de livraison (decliv) qui décide si oui ou non il apporte des modifications au travail de propliv.

- S'il n'y a pas de modifications, le contrôle de la liste lpl est donné immédiatement au processus d'édition des bons de livraisons (edbl) qui déduit la liste de ces bons (lbl) par copie de la liste lpl.

- S'il y a des modifications à effectuer, un certain pourcentage de lignes de commandes de lpl sont choisies pour constituer ces modifications. Ensuite v, avec la priorité 1, construit la liste des propositions de livraisons modifiées lplm, et saisit ses éléments sur ordinateur avant de redonner le contrôle à propliv pour une nouvelle boucle. Celle-ci peut au maximum être exécutée trois fois.

La décision ou non de poursuivre une boucle est prise au niveau de decliv avec :

- pour la première boucle, 50 % de chance,

- pour la deuxième boucle, 20 % de chance,

- pour la troisième boucle, 5 % de chance d'être effectuée.

Les temps de réponse (ou de travail) diminuent à chaque tour. On rappelle que dans un premier temps ces paramètres ne sont qu'approximatifs.

De nombreux problèmes dus à cette modélisation nous ont amenés à utiliser une liste lcojo'. Les raisons de ces choix sont plus largement expliqués dans (KLE78).

Nous avons vu, au paragraphe IV.2.2 que chaque processus de l'application peut être activé par l'intermédiaire d'un moniteur d'activation. Nous disposons de trois types de moniteurs d'activation:

- le type 1 sans mémorisation,
- le type 2 à mémoire multiple,
- le type 3 à mémoire unique.

Le choix d'un type précis de moniteur d'activation pour un processus donné dépend du type de structure de données sur laquelle travaille ce processus. Pour simplifier le problème, on peut dire que les ressources informations traitées dans la maquette par un processus peuvent être classées en 4 types : boîte unique, files dont on traite un élément, files "figées" (c'est à dire n'évoluant pas pendant le travail du processus) et files normales ou "évolutives" dans lesquelles on autorise les modifications (adjonction, suppression...) pendant le travail du processus. Ainsi on peut dire que :

- un processus traitant une boîte unique a un moniteur d'activation de type 1. Souvent il y a double emploi entre un signal et un message : on assimile alors l'un à l'autre ;

- un processus traitant une file élément par élément est activé par un moniteur d'activation du type 2 ;

- un processus utilisant une liste figée la traite au cours de son activation jusqu'à épuisement. On utilise donc ici un moniteur d'activation du type 3 ;

- enfin tout processus travaillant sur une liste normale ou "évolutive" doit correspondre à un moniteur d'activation de type 1. En effet, quand ce processus redemande un signal d'activation, il doit avoir épuisé la liste.

Les tableaux et les fichiers sont assimilables à des ressources réutilisables du type files dont on traite un élément. Les processus traitant ces types d'informations doivent posséder un moniteur d'activation à mémoire multiple pour qu'ils puissent prendre en compte toutes les éventuelles modifications.

Cette classification n'est qu'un essai, une proposition afin de montrer comment il est possible de raisonner. L'emploi d'un type plutôt qu'un autre est à la charge de l'utilisateur.

γ) L'échéancier d'activation des processus

Il nous permet de préciser les dates fixes d'activation des processus de notre système. La période de l'échéancier a été prise d'une journée. Les processus de notre maquette utilisant l'échéancier sont :

- la génération de bons de commandes écrits, devant être activé à 9H 30 tous les matins ;
- la codification des bons de commandes commence à 10H chaque matin ;
- les propositions de livraison à 15H ;
- la préparation des livraisons à 9H ;
- la décision des facturations à 14H ;
- la facturation à 17H tous les jours.

L'échéancier d'activation a donc la forme décrite figure 21.

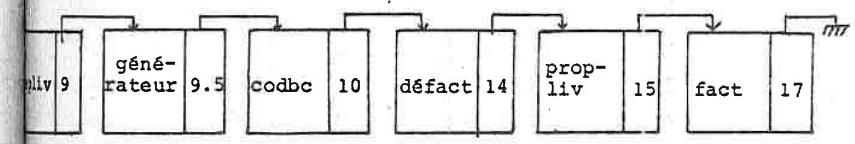


figure 21

Sur ce schéma figurent les noms des processus à activer. En réalité nous avons, dans l'échéancier, une référence aux moniteurs d'activation concernés.

c) Aspects physiques

Nous considérons, ici, les ressources-processeurs prises en compte dans notre maquette ainsi que leur allocation.

α) Les ressources-processeurs

Nous utilisons, dans notre maquette, quatre types de processeurs :

1. Des ressources critiques avec calendrier (calrc). Elles sont composées par toutes les personnes qui effectuent, seules, certains traitements. Nous trouvons dans cette catégorie les personnes r, préparateur et val.

2. Une ressource critique préemptible, avec priorité et calendrier (calrcpp). C'est la personne v. Elle est chargée, dans le modèle, de différents traitements. D'une part elle s'occupe des décisions de livraisons puis de leur saisie, et d'autre part elle répond au téléphone et enregistre les commandes téléphonées. Cette dernière activité est prioritaire vis à vis des autres traitements. Cela justifie l'emploi d'une calrcpp.

3. Une ressource à accès multiple avec calendrier (calram). C'est le service de facturation sfact composé de deux personnes. On dispose donc de deux accès à cette ressource.

4. Une ressource à accès multiple infini sans calendrier qui représente l'ordinateur. Comme on l'a vu cela évite d'éventuels blocages. Cette ressource permet uniquement de prendre en compte un temps de réponse dans le système.

Ces ressources ont été définies ainsi dans un premier temps. Mais quand nous disposerons d'éléments supplémentaires sur l'organisation de l'entreprise, ce modèle pourra être modifié (en particulier pour préparateur, sfact et l'ordinateur).

La modularité, l'indépendance des éléments d'une maquette dans notre projet nous permettent d'envisager ces changements de manière aisée.

Les caractéristiques de chacun des processeurs utilisés sont résumées dans le tableau de la figure 22.

β) Les calendriers

Les différentes ressources (processeurs) utilisées dans le système représentent, pour la plupart, des personnes ou des services dont on connait l'emploi du temps dans une journée. On peut donc utiliser ici le mécanisme de calendrier pour tenir compte de la disponibilité de ces processeurs, pour le sous-système considéré.

Ainsi, r travaille de 10H à 12H, puis de 14H à 15H tous les jours. v, le préparateur et le service de facturation travaillent de 9H à 12H et de 14H à 18H tous les jours. De son côté val ne travaille que de 14H à 16H 30. On peut donc associer un calendrier, précisant les périodes de disponibilité, à chacune de ces ressources.

L'ordinateur est supposé disponible 24H sur 24, on ne lui associe donc pas de calendrier.

Pour chacun des processeurs utilisant le mécanisme de calendrier, on a la possibilité de préciser si oui ou non on permet au processus utilisant ce processeur de terminer son travail en cours en fin de période de disponibilité. Cette option assure une modélisation et une simulation plus proche de la réalité des organisations.

On verra sur la figure 22 quelles sont les options adoptées pour chacun des processeurs de l'application.

NOM	TYPE	Nbre d' accès	Option laissant terminer le travail en cours	Heures de disponibilité
r	calrc	1	OUI	10H à 12H et 14H à 15H
ord	calraminf	"∞"	/	Toujours
v	calrcpp	1	NON	9H à 12H et 14H à 18H
préparateur	calrc	1	OUI	9H à 12H et 14H à 18H
val	calrc	1	OUI	14H à 16H 30
sfact	calram	2	OUI	8H à 12H et 14H à 18H

Figure 22

Tableau donnant les caractéristiques des ressources

V.2.3 - ECRITURE DE LA MAQUETTE

La structure du programme permettant l'exécution de la maquette décrite dans ce chapitre est la suivante :

```
begin
   maestro begin
   ---      définition des transactions (bc, bca et bcl)
   ---      définition des processus

   ---      déclaration des variables de travail
   ---      déclaration des files d'attente
   ---      déclaration des régions
   ---      déclaration des processeurs
   ---      déclaration des moniteurs d'activation

   ---      initialisation des variables de travail
   ---      création des régions
   ---      création des files
   ---      création des ressources (éventuellement,
            construction des calendriers)
   ---      création des moniteurs d'activation
   ---      construction de l'échéancier
   ---      initialisation du contenu des files-fichiers

   ---      activation de tous les processus
   ---      lancement de la simulation (procédure simul)
   end ∮ maestro ∮
end ∮ programme ∮ //
```

On peut se reporter à l'annexe 5 pour avoir le listing complet de cette application.

V.2.4 - CONCLUSION

Nous concluons ce chapitre en précisant que nous avons présenté ici, une manière de modéliser le système de gestion des commandes et de facturation de cette entreprise. Il existe d'autres manières de faire cette modélisation (DUF79b). Les valeurs des paramètres utilisées pour les lois de service et les temps de réponse ne sont, pour l'instant, qu'approximatives. La maquette construite et testée n'a donc de valeur que pour une validation des outils conçus et réalisés dans l'optique d'une étude du comportement dynamique (qui n'est pas encore faite). Des études plus poussées sur les paramètres sont actuellement en cours (THA79).

Cette maquette a été testée du point de vue de son fonctionnement et des résultats ont été obtenus. Nous les présentons dans le chapitre suivant.

CHAPITRE VI

MESURES SUR LES MAQUETTES ET EXPLOITATION DES RESULTATS

CHAPITRE VI -

MESURES SUR LES MAQUETTES ET EXPLOITATION DES RESULTATS

Ce chapitre présente les mesures effectuées sur les maquettes. Nous ne développons pas, ici, les aspects statistiques ni l'exploitation des résultats. Ces études sont actuellement en cours par M. THAUREL (THA79).

Nous ne donnons que des résultats indiquant des mesures globales synthétisant le comportement dynamique du système.

Nous présentons, tout d'abord, les outils de prises de mesures mis au point par M. THAUREL, qui s'inspirent partiellement de ceux de GPSSS (VAU77) ; puis nous montrons les résultats obtenus sur la maquette présentée au chapitre V.

VI.1 - OBJECTIFS DES MESURES

L'exécution de maquettes de systèmes d'information en organisations vise, ici, à évaluer le comportement dynamique de ces systèmes. Le but est d'obtenir des mesures permettant d'apprécier :

- les délais de réponse ;
- la charge de certains composants ;
- l'équilibre du système décrit.

Nous ne détaillons pas, ici, ces notions. On trouvera de plus amples renseignements dans (THA79). Précisons toutefois que les mesures moyennes n'ont de sens que si on en connaît la précision, c'est-à-dire qu'il est souhaitable d'avoir pour chacune d'elles un intervalle de confiance; et que si les processus sous-jacents sont quasi-stationnaires.

Pour effectuer les mesures, l'utilisateur a à sa disposition deux possibilités (outils) :

- des processus spécialisés ajoutés au modèle ;
- des procédures spécialisées appelées dans les processus.

L'appréciation des temps de réponse est réalisée grâce à des entités appelées régions de traitement. Cette notion est analogue aux "queues" de GPSS (IBM70) et aux "régions" de GPSSS (VAU77). Une région est une entité incorporée au système pour collecter des résultats statistiques et qui n'a aucune influence sur le déroulement de la simulation. Un temps de réponse est alors défini comme la durée séparant l'entrée d'une information (transaction cf VI.2) dans une région, de la sortie de cette même information de cette région. L'utilisateur doit définir les régions de son système de manière correcte. Il est souhaitable que ces régions soient disjointes. Un type d'information considéré ne doit pas subir de transformations fondamentales dans cette région, et ainsi doit conserver ses caractéristiques. En effet, nous exposons par la suite des mesures effectuées sur les régions de notre maquette (§ VI.3.3). Or, les transactions entrant dans la région "administrative" sont des bons de commande (type bc) et les transactions en sortant sont des bons de livraison (type bcl). A un bon de commande entré peuvent correspondre plusieurs bons de livraison en sortie. L'information, dans ce cas, a subi des transformations dans la région et il nous faut définir sur quel type de transactions les mesures d'entrée et de sortie sont effectuées, sinon on risque d'aboutir à des résultats sans aucune valeur statistique.

VI.2 - OUTILS DE MESURE

Afin de séparer nettement la prise de mesure, l'édition et l'analyse statistique, nous utilisons des fichiers :

- le fichier FGEN qui contient les résultats généraux concernant les ressources, les régions et les files ; il est constitué, éventuellement, à la fin de la simulation et sert pour une édition séparée ;
- le fichier FVØL qui contient les mesures constituant les séries chronologiques relatives aux volumes d'information.

Une description détaillée de ces fichiers peut être consultée dans (THA79).

Les mesures effectuées concernent les régions, les files, et les processeurs.

Dans les régions et les files, on s'intéresse aux entrées et sorties de transactions, objets de la classe message dont la structure est :

```
(1)     link class messages (no) ; integer no ; null ;
                                          ¢ message ¢
(2)     message class transaction (pds) ; real pds ;
(3)     begin
           ...
        end ; ¢ transaction ¢
```

(1) entête de la classe message où no est un entier permettant de repérer le message ;

(2) entête de la classe transaction ou pds définit un poids ;

(3) déclaration et mise à jour de variables servant à effectuer des mesures (cf (THA79)).

VI.2.1 - LES REGIONS

Une transaction entre ou sort d'une région, ces mouvements sont comptabilisés par les procédures de la classe trégion dont la structure est :

```
(1)     link class trégion (nom) ; text (nom) ;
        begin
(2)       ...
(3)       procedure entrer (tr) ; ref (transaction) tr ;...;
(4)       procedure sortir (tr) ; ref (transaction) tr ;...;
          ...
        end ; ¢ trégion ¢
```

(1) entête de la classe trégion où nom est un libellé permettant d'identifier la région ;

(2) déclaration de compteurs locaux ;

(3) l'appel de cette procédure permet de mettre à jour les variables locales lors de l'entrée d'une transaction dans la région ;

(4) l'appel de cette procédure met à jour les variables locales lors de la sortie d'une transaction de la région.

Pour créer une région, on utilise la suite d'instructions :

```
ref (trégion) rg ;
rg :- new trégion ('rg') ;
```


VI.2.2 - LES FILES

Nous avons vu, au § IV.2.1, que la classe tfile est conçue essentiellement pour faciliter le dénombrement des transactions dans ces files. On peut obtenir des résultats généraux relatifs à ces files en positionnant le booléen mesure à vrai. Il est également possible de définir quelles sont, parmi toutes les files du système, celles qui serviront à étudier l'équilibre de celui-ci (cf THA79). Pour cela on positionne le booléen équilibre.

La structure de la class tfile est :

```
(1)     monitor class tfile (mesure, équilibre,inter) ;
        boolean mesure, équilibre ; integer inter ;
        begin
(2)       ref (head)q ;
          ...
(3)       procedure entrer (tr) ; ref (transaction) tr ;...;
(4)       procedure sortir (tr) ; ref (transaction) tr ;...;
          ...
(5)       q :- new head ;
          ...
        end ; ¢ tfile ¢
```

(1) entête de la classe tfile, où mesure est un booléen positionné à vrai si on désire obtenir des résultats généraux sur la file considérée ; équilibre précise si cette file sert à l'étude de l'équilibre du système et dans ce cas, inter donne l'intervalle de temps entre chaque mesure sur cette file participant à la définition de l'équilibre ;

(2) déclaration de la file contenant les transactions ;

(3) l'appel de cette procédure met à jour les variables et compteurs locaux lors de l'entrée d'une transaction dans la file :

(4) l'appel de cette procédure provoque la mise à jour des variables locales lors de la sortie d'une transaction de la file ;

(5) création effective de la file q.

La création d'un objet de type file appelé lbc dont on désire des résultats généraux mais qui ne sert pas à définir l'équilibre du système est faite ainsi :

```
ref (tfile) lbc ;
lbc :- new tfile ('lbc', true, false, 0) ;
```

si la file lbci sert à l'équilibre en prenant des mesures toutes les 24 heures et en obtenant également des résultats généraux, elle est créée ainsi :

```
ref (tfile) lbci ;
lbci :- new tfile ('lbci', true, true, 24) ;
```

VI.2.3 - LES RESSOURCES

Nous définissons une classe ressource permettant des prises de mesures et des résultats lors de l'acquisition et de la libération des ressources-processeurs du système. Cette classe permet de préfixer les différentes définitions des types de ressources. Sa structure est :

```
(1)     monitor class ressource ;
        begin
(2)        ...
        end ; $ ressource $
```


(1) entête de la classe

(2) déclaration de procédures permettant l'initialisation et la mise à jour des variables et compteurs locaux aux ressources. Ces procédures ne sont pas manipulées par l'utilisateur, en effet la prise de mesure sur les ressources est automatique à l'acquisition et à la libération par un processus. Nous ne les détaillons donc pas ici. Pour plus amples renseignements, le lecteur peut consulter (THA79).

VI.2.4 - PRINCIPE D'UTILISATION

a) Prises de mesures

Toutes les prises de mesures sont effectuées automatiquement, sauf celles concernant les entrées et les sorties de transactions de files et de régions. Pour illustrer celles-ci, prenons par exemple :

- commande référence un objet de la classe transaction ;
- facturation référence un objet de la classe trégion ;
- lcojo référence un objet de la classe tfile.

Considérant la transaction référencée par commande, les prises de mesures lors des entrées et sorties des files et des régions se font ainsi :

- entrée dans la région facturation : facturation.entrer (commande) ;
- sortie de la région facturation : facturation.sortir (commande) ;
- entrée dans la file lcojo : lcojo.entrer (commande) ;
- sortie de la file lcojo : lcojo.sortie (commande).

b) Lancement de l'exécution de la maquette

Comme nous l'avons dit au § V.1.4, pour lancer une simulation, on appelle la procédure simul :

simul (duréesimul, éditrésgén, équilibre, bfgen) ; où duréesimul est un nombre réel d'heures donnant la durée de la simulation, éditrésgén précise si les résultats généraux doivent être édités en fin de simulation, équilibre indique si on recherche l'équilibre du système et bfgen est vrai si les résultats généraux sont édités ultérieurement en utilisant le fichier FGEN.

Ainsi, pour lancer une simulation sur 3 jours, en définissant l'équilibre du système et en ayant une édition ultérieure des résultats, on utilise l'appel suivant :

```
simul (72, false, true, true) ;
```

c) Trace des processus

Pour contrôler l'évolution du système et plus précisément pour obtenir une trace des processus dans le système, l'utilisateur dispose de la procédure passe dont la structure est :

```
(1)     procédure passe (lib, débute) ; text lib ;
                                        boolean débute ;
        begin
           ...
        end ; $ passe $
```

(1) entête de procédure où lib est un texte (en général un nom de processus) et débute un booléen vrai si on désire imprimer un message indiquant le début de travail d'un processus et faux s'il s'agit de la fin. La date de l'appel de la procédure est également écrite en clair. Par exemple : l'appel


```
passe ('codbc', true) ;
```

effectué à 2H 27 provoque l'impression :

```
0 2 27 DEBUT DE TRAVAIL DE CODBC
```

l'appel à 3H 47 de

```
passe ('saisbc', false) ;
```

provoque l'impression de :

```
0 3 47 FIN DE TRAVAIL DE SAISBC
```

VI.3 - MESURES EFFECTUEES SUR LA MAQUETTE

Nous montrons, dans ce paragraphe, un exemple de résultats statistiques obtenus à partir de la simulation de la maquette présentée au paragraphe V.2. Ces résultats ne sont que généraux et une analyse plus fine, en cours de réalisation ne sera présentée que lors de prochains rapports (THA79).(*) Remarquons en outre que ce sont les résultats d'une simulation dont les paramètres (lois de services, temps de réponse...) ne sont qu'approximatifs. Ces mesures portent essentiellement sur :

- les processeurs,
- les files,
- les régions.

Elles sont éditées grâce à un programme utilisant le fichier FGEN construit lors de la simulation. On trouvera dans l'annexe 6 un listing montrant une trace constatant l'évolution des processus dans le système lors de la simulation (obtenue par appels de la procédure passe).

(*) Nous montrons ici quels sont les types de mesures effectuées pour chacun des composants d'un système ; puis nous présentons les résultats obtenus par la simulation de la maquette décrite en V.2.

Tous ces résultats ont été obtenus, à titre d'exemple, par une simulation de notre maquette sur 3 jours ; (en effet, cette simulation n'est qu'une démonstration montrant ce qu'il est possible d'obtenir, les résultats n'ayant que peu de valeurs statistiques).

VI.3.1 - MESURES SUR LES FILES

Les résultats généraux que l'utilisateur peut obtenir (en positionnant le booléen mesure de la classe tfile à vrai) sur les files, concernent :

- le nombre d'entrées des transactions dans la file concernée ;
- le nombre de sorties de ces transactions ;
- les contenus maximum
  moyen
  et actuel de la file ;
- le nombre de retours à zéro (file vide).

Il est en outre, possible d'obtenir d'autres résultats concernant les files qui font partie de la définition de l'équilibre du système. Ces résultats indiquent :

- le nombre de mesures effectuées sur cette file ;
- leur périodicité ;
- les évaluations des volumes d'information :

  . moyen,
  . maximum,
  . minimum ;

- ainsi que le volume d'information dans la file en fin de simulation.

Les résultats présentés dans les tableaux des figures 23 et 24 ont été obtenus par l'exécution de la maquette présentée au chapitre V, dans laquelle les files lcoat et lblnf ont été initialisées à 50 éléments (mesures approximatives dans l'entreprise).

FILE (S)	ENTREES	SORTIES	CONTENU DE LA FILE			RET A ZERO
			MAXIMUM	MOYEN	ACTUEL	
LBC	45	45	15	0.677	0	3
LBCI	45	45	15	1.082	0	3
LCOJO	61	50	24	2.634	11	0
LPL	138	138	65	44.425	0	3
LBL	138	113	55	14.377	25	2
LCOAT	106	89	50	28.036	17	2
LBLP1	113	104	65	18.539	9	2
LBLP2	113	0	113		113	0
LBLNF	110	63	88	31.592	47	0
LBLP	107	107	66	1.392	0	2
LBLF	107	107	66	2.552	0	2
LF	107	107	66	0.999	0	2
LFCL	107	0	107		107	0
LFCPTA	107	0	107		107	0

figure 23

MESURES			EVALUATION DES VOLUMES D INFORMATION : VOLUME			
FILE (S)	NOMBRE	INTERVALLE	MINIMAL	MAXIMAL	MOYEN	ACTUEL
LBC	3	1 0 0	0	0	0	0
LCOJO	3	1 0 0	0	0	0	0
LCOAT	3	1 0 0	3017	3588	3398	3588
LBLP1	3	1 0 0	145	3770	2401	3770
LFCL	3	1 0 0	0	4004	1347	4004

figure 24

VI.3.2 - MESURES SUR LES PROCESSEURS

Les prises de mesures sur les processeurs sont réparties en trois catégories :

- mesures sur les ressources critiques,
- mesures sur les ressources à accès multiples,
- mesures sur les processeurs partagés.

a) Ressources critiques

Pour chaque ressource critique, qu'il s'agisse de ressource critique simple, de ressource critique avec priorité ou de ressource critique avec priorité, préemptible (cf IV.3.1), nous mesurons les éléments suivants :

- le nombre de processus servis par cette ressource,
- la durée totale pendant laquelle cette ressource est disponible durant le temps de simulation,
- le taux d'utilisation de cette ressource $\left(\frac{\text{durée d'utilisation}}{\text{durée disponible}}\right)$,
- enfin l'état (libre ou occupé) de la ressource en fin de simulation.

Les résultats de ces mesures sur notre maquette sont donnés dans les tableaux 25 et 26.

b) Ressources à accès multiples

Pour ces types de ressources (cf IV.3.1) nous mesurons :

- le nombre total d'accès total disponible pour la ressource,

- les nombres . maximum
  . moyen
  . en fin de simulation, d'accès utilisés
- les taux d'utilisation . de la ressource
  . des accès

Les résultats obtenus figurent sur le tableau 27.

Pour les ressources à accès multiples, on se contente de mesures relatives aux nombres d'utilisateurs :

- servis pendant la simulation
- moyen
- maximum
- et en fin de simulation.

Les résultats obtenus sur l'ordinateur, seule ressource à accès multiples de notre maquette, sont donnés dans le tableau 28.

c) Processeurs partagés

Les mesures sur ces ressources, présentées au paragraphe IV.3.1, concernent :

- les nombres d'utilisateurs servis
  maximum
  et en fin de simulation,
- ainsi que le taux d'utilisation de ces ressources.

N'ayant pas de processeur partagé dans la maquette présentée, on pourra trouver un exemple de résultat dans (THA79).


```
**************************
** RESULTATS SIMULATION **
**************************

NO PASSAGE  :  1

TEMPS SIMULATION  :  0  0  0

DUREE SIMULATION  :  3  0  0
```

CALRC	PROCESSUS SERVIS	DUREE DISPONIBLE	TAUX UTILISATION	ETAT ACTUEL
ID	48	0 9 0	0.703	LIBRE
VAL	110	0 5 0	0.546	LIBRE
PREPA	3	0 21 0	0.556	OCCUPE

figure 25

```
**************************
** RESSOURCE TYPE CALRCP **
**************************
```

CALRCPP	PROCESSUS SERVIS	DUREE DISPONIBLE	TAUX UTILISATION	ETAT ACTUEL
✓	54	0 21 0	0.130	LIBRE

figure 26

CALRAM		NOMBRE ACCES UTILISES			TAUX UTILISATION	
		MAXI.	MOYEN	ACTUEL	CALRAM	ACCES
SFACT	2.00	1.00	0.013	0.00	0.013	0.01

figure 27

AUCUNE RESSOURCE TYPE CALCPP
****

	NOMBRE D UTILISATEURS			
PAMINF	SERVIS	MOYEN	MAXI.	ACTUFL
ORDINATEUR	517	0.11	3	0

figure 28

VI.3.3 - MESURES SUR LES REGIONS

A partir des mesures prises à l'entrée, et à la sortie d'une transaction d'une région, on obtient des résultats concernant :

- le nombre d'entrées de transactions dans la région ;
- le nombre de sorties de cette région ;
- les contenus . moyen
  . maximum
  . en fin de simulation (actuel) de la région ;
- enfin, le temps de séjour . moyen
  . maximum
  . minimum des transactions dans la région.

Les différentes régions de notre maquette ont été définies en correspondance des cinq étapes de la gestion d'une commande, à savoir :

- prise de commandes écrites (région : "écrites") ;
- prise de commandes téléphonées (région : "téléphonée") ;
- traitement administratif (région : "administratifs")
- traitement physique (région : "physique") ;
- facturation (région : "facturation").

Les résultats obtenus sur notre maquette sont présentés dans le tableau de la figure 29.

Les remarques énoncées à propos de la définition des régions (cf § VI.1) sont illustrées ici par la région "administratifs" où les transactions entrant sont des bons de commandes et leur sortie est signalée qu'à partir du moment où il n'existe plus de lignes en attente.

REGION (S)	ENTREES	SORTIES	CONTENU DE LA REGION			TEMPS DE SEJOUR					
			MAXIMUM	MOYEN	ACTUEL	MAXIMUM			MOYEN		MIN
ECRITES	45	45	15	1.764	0	0	4	50	0	2 49	0 1
TELEPHONEES	60	16	44	0.000	44	0	0	0	0	0 0	0 0
ADMINISTRATIF	100	83	66	36.653	17	2	16	8	1	7 47	0 15
PHYSIQUE	114	113	1	0.583	1	0	15	13	0	0 22	0 0
FACTURATION	154	107	115	36.528	47	2	17	45	1	0 34	0 3

figure 29

VI.4 - CONCLUSION

Nous avons présenté ici les résultats bruts (*) d'une première version de la maquette décrite au § V.2 qui utilise les classes définies dans le chapitre IV. Les différents outils de prise de mesures seront améliorés dans une prochaine phase de notre travail (THA79). De meilleures données (lois de services, temps de réponse) sont en cours de collecte dans l'entreprise. Ainsi, la validation du modèle pourra certainement être prouvée.

(*) Remarque : Les résultats donnés dans ce chapitre concernent parfois des moyennes. Il nous faut signaler que, pour que ces mesures aient un sens pour procéder à leur analyse, nous devons évaluer leur dispersion, par exemple en calculant des intervalles de confiance.

CHAPITRE VII

CONCLUSION

Le travail présenté dans cette thèse avait pour but essentiel de concevoir et de réaliser des outils susceptibles d'être utilisés pour l'écriture de maquettes de systèmes d'information et pour leur exécution simulant le fonctionnement des systèmes décrits. Il nous semble que l'ensemble ainsi défini est cohérent et que l'on dispose d'un logiciel d'une grande puissance de description.

Dans cette conclusion, nous faisons, avec un rappel des points principaux, une synthèse du travail effectué. Puis nous formulons certaines critiques vis à vis des choix entrepris, pour en déduire des améliorations et des extensions possibles. Nous terminons enfin sur quelques perspectives d'avenir pour le projet MAESTRO.

La description de l'architecture générale, telle que nous la voyons, est fondée essentiellement sur celle du partage d'informations entre des processus et celle de leur concurrence pour obtenir des ressources. Il est bien entendu que ce n'est qu'une approche parmi d'autres (voir notamment les méthodes exposées au § I.1), mais c'est elle qui nous est apparue la plus naturelle pour aborder la modélisation de systèmes d'information en organisations.

L'approche choisie pour résoudre les problèmes de synchronisation des processus est originale dans le domaine de la simulation de systèmes généraux. Nous avons choisi comme unique outil d'expression du parallélisme, de la synchronisation et de la concurrence, le mécanisme de moniteurs de HOARE. Là encore, il s'agit d'une solution parmi d'autres (cf. § III.2), mais, outre qu'elle nous a permis de rendre compte assez facilement de tous nos problèmes de synchronisation, elle conduit à une réalisation aisée en SIMULA67. Toutefois, nous avons été amenés à modifier les moniteurs pour tenir compte de certains phénomènes tels que ceux liés à la préemption et aux calendriers. On

pourra peut-être dire qu'il aurait été plus simple d'utiliser des activations directes de processus (par exemple au moyen des instructions (RE) ACTIVATE de SIMULA) mais on sait le danger que représente une utilisation de mécanismes d'aussi bas niveau : il suffit de penser aux ALLERA de la programmation traditionnelle... En outre, les moniteurs ont l'avantage, par rapport à ces activations directes, de rendre, pour un processus, le mode de déclenchement totalement transparent.

Nous avons choisi SIMULA comme langage de simulation et de description de systèmes car il permet de construire des outils paramétrés facilement extensibles grâce à la notion de classes hiérarchisées. La manipulation de listes et le quasi-parallélisme y sont facilement utilisables, et ce langage semble permettre de pratiquer une sorte de programmation abstraite grâce aux procédures virtuelles.

De plus, les nouvelles versions de SIMULA comprennent des améliorations non négligeables. On y offre la possibilité de fabriquer son propre environnement catalogué (par exemple pour nous, la classe MAESTRO) et même, maintenant, des facilités de compilation séparée. En outre, ces nouvelles versions introduisent dans la définition des classes des listes d'attributs non protégés, seuls accessibles de l'extérieur du corps de classe (grâce à la clause "NOT HIDDEN PROTECTED"). Grâce à SIMULA, nous pouvons donc réaliser des outils généraux permettant d'en fabriquer d'autres, particuliers à chaque système à modéliser. Enfin, autre point intéressant pour les systèmes d'information, il y a maintenant de nombreuses études et réalisations sur la définition et l'implantation de bases de données en SIMULA...

En résumé, notre système a pour avantages sa facilité d'utilisation pour un informaticien, sa puissance de description et la possibilité d'extension aisée de nos différents outils. Toutefois, il est certain qu'un utilisateur doit avoir de bonnes connaissances dans les domaines de la modélisation et de la simulation de systèmes de processus coopérants il doit en outre bien connaître SIMULA.

Mais ce qui a été présenté ne constitue en fait qu'une première version expérimentale. Cela nous amène à quelques réflexions critiques sur ce qui a été fait.

Si la manière de concevoir l'architecture générale d'une maquette semble satisfaisante, en revanche quelques doutes peuvent apparaître sur l'utilisation des moniteurs. Comme on l'a vu, ils sont suceptibles d'améliorations (cf. KESSELS (KES77)), mais, en outre, ils ne sont certainement pas la seule solution à notre type de problèmes. On peut en effet s'interroger sur les nouvelles idées dans ce domaine, notamment sur les nouveaux concepts introduits par BRINCH HANSEN (BRI78) et HOARE (HOA78) pour résoudre les problèmes de synchronisation. Il est possible qu'alors, l'architecture générale soit légèrement différente et que, par exemple, nous n'ayons plus de données partagées passives et qu'elles soient remplacées par des processus. Mais ces nouvelles notions ne sont pas encore opérationnelles puisqu'elles contiennent des problèmes non résolus, notamment au niveau de l'implantation physique. Il nous semble donc plus sûr de nous en tenir à nos choix pour l'instant.

L'utilisation de SIMULA est également sujette à critiques. En effet, on peut douter des possibilités d'écrire d'emblée un modèle complet en SIMULA. On considère que ce n'est pas un langage d'assez haut niveau, ne permettant pas des définitions suffisamment statiques des objets utilisés.

Nous abordons alors le problème de la facilité d'utilisation. Il est clair que SIMULA est un langage de spécialistes (informaticiens) et il serait alors souhaitable de disposer d'un langage plus général qui enroberait tous nos mécanismes et qui serait accessible par n'importe quel utilisateur, même "non-informaticien". La tendance actuelle est d'utiliser de moins en moins les langages procéduraux pour s'orienter davantage vers des langages de spécification. Sans doute un progrès supplémentaire serait-il d'avoir un langage permettant l'écriture et la modification de maquettes de manière interactive par une communication conversationnelle homme-machine.

Les améliorations ci-dessus concernent plutôt une utilisation plus aisée du système de programmation de maquettes. D'autres, plus ponctuelles, concernent les composants définis dans le chapitre IV.

Ces composants suffisent, moyennant parfois quelques astuces de conception (cf § IV.3.1), pour écrire la majeure partie des maquettes. Cependant nous avons tous les outils pour pouvoir définir facilement d'autres composants : par exemple, l'écriture de nouveaux types de ressources ou la modification structurelle de ceux qui existent ne semble guère poser de problèmes.

Pour les processeurs, il serait peut-être souhaitable de disposer de mécanismes gérant les horaires variables (par exemple réalisés par des processus parasites ou par des modifications dynamiques et aléatoires des calendriers). De la même manière, nous pourrions systématiser, par des outils spécialisés, les arrêts de travail de certains processeurs, de manière aléatoire (maladie...) ou cyclique (week-end, vacances...). Au niveau des processus, nous pourrions compléter leurs caractéristiques en permettant des changements dynamiques de priorités. Nous augmenterions ainsi la puissance de description de notre système. Dans la même optique, il serait souhaitable d'améliorer les mécanismes d'échéancier et de calendriers pour faciliter les modifications dynamiques.

Cela leur procurerait une meilleure souplesse d'utilisation. Il faudrait aussi prévoir la possibilité de définir des calendriers associés à des ressources dont les périodes de disponibilité seraient différentes en fonction des processus utilisant ces ressources (dans notre première version, il est possible de concevoir ce mécanisme en créant autant de ressources que de processus différents à condition que les différentes périodes de disponibilité soient disjointes). Enfin, il serait bon d'approfondir nos réflexions sur d'autres points. Par exemple, grâce à la confrontation à l'expérience, il faudrait systématiser les mécanismes de synchronisation par une typologie plus précise des informations (cf. moniteurs d'activation de différents types).

Toutes ces propositions d'améliorations nous amènent à rappeler une remarque importante : il serait faux de croire que, plus un système est détaillé, meilleur il est. Dans notre cas, quand nous essayons d'effectuer une évaluation quantitative par des études statistiques, nous éprouvons de grandes difficultés si le système comporte un luxe de détails, notamment s'il y a trop d'éléments influant sur la dynamique de ce système. C'est pourquoi, avec toutes ces améliorations, il faut être très prudent et ne pas transformer les maquettes en monstres inexploitables.

Pour terminer, nous voudrions que ces études débouchent sur la définition d'une méthode de modélisation par niveaux. Souvent, lors de l'analyse ou de la conception d'un système d'information, à partir d'une même description générale, on désire procéder soit à une évaluation du fonctionnement du système, soit à une évaluation quantitative de son comportement dynamique. Quel serait alors le niveau de détail à prendre en compte dans cette description générale, par une maquette, avant de la détailler de deux manières

différentes correspondant à ces deux types d'objectifs ? Un effort de réflexion doit être fait sur les principes et les critères de réduction des informations, des processus, des ressources et sur la décomposition du système en différents niveaux avec des échelles de temps différentes.

Il nous semble que l'originalité de ce travail est qu'il est une des premières tentatives de modélisation globale de systèmes d'information et qu'il constitue une nouvelle approche de la construction de modèles de simulation.

CHAPITRE VIII

BIBLIOGRAPHIE

BIBLIOGRAPHIE PAR THEMES

1. Etudes des mécanismes de synchronisation

ALG72, BEZ75, BRI75, BRI78, CAM74, CON63, CRO75, DIJ68, DIJ75, GIR78, HAB72, HAB75, HOA74, HOA78, KES77, PET77, PUL78, RAY78, SAA70, VAU73, VAU77, VAU78, VEI72.

2. Etudes des systèmes d'information

BOD78, DUF78d, FLO77, FOR61, GIR77, HUR76, LUG75, MAL71, PEC75, PRO70, ROL79, TEI73.

3. Langages de simulation

BUX62, CII72, DAH66, DAH68, DAH70, FIS73, GAS69, IBM70, KNU64, MAR63, VAU77.

4. Langages de description de systèmes

BEZ76, KAU76, WIR77.

5. Divers - Ouvrages généraux

CRO75, FIS73, GUT77, KNU68, LAR79, LER77, LIS77, LIV78, REM74, VEI72.

6. Publications de Maestro

DUF76, DUF78a, DUF78b, DUF78c, DUF78d, DUF79a, DUF79b, KLE78, THA79.

(ALG72) Groupe ALGOL de l'AFCET. Définition du langage algorithmique ALGOL 68. Ed. Hermann. Paris (1972)

(BEZ75) BEZIVIN J. Concepts et méthodes pour la production de systèmes opératoires. Rennes. Thèse de spécialité. (1975)

(BEZ76) BEZIVIN J. et al. SIMONE : Manuel de programmation. IRIA/SESORI. Rocquencourt. (1976)

(BOD78) BODART F., PIGNEUR Y. Spécifications dynamiques d'un système d'informations. Inforsid séminaire gr. 2 Cergy. (1978)

(BRI75) BRINCH HANSEN P. The programming language Concurrent Pascal. International Summer School. Münich (juillet 1975)

(BRI78) BRINCH HANSEN P. Distributed processes : A concurrent programming concept. CACM vol. 21 n°11. (nov. 1978)

(BUX62) BUXTON J.N., LASKI J.G. Control and Simulation Language. Computer journal, 5 n°5. (1962)

(CAM74) CAMPBELL R.H., HABERMANN A.N. The specification of process synchronization by path expressions. Lecture notes in Computer Science. Vol 16. Springer Verlag. (1974)

(CII72) CII-HB. SIMULA sous Siris 7/8 : Manuel d'utilisation. Louveciennes. (1972)

(CON63) CONWAY M.E. Design of a separable Transition - Diagram Compiler. CACM vol.6 p.396-408. (1963)

(CRO75) CROCUS. Systèmes d'exploitation des ordinateurs. Ed. Dunod. Paris (1975)

(DAH66) DAHL O.J., NYGAARD K. SIMULA, an Algol-based simulation language. CACM vol.9 n°9. (1966)

(DAH68) DAHL O.J. Discrete event simulation language. In Programming Languages. Ed. Genuys Academic Press. (1968)

(DAH70) DAHL O.J. et al. The SIMULA 67 Common Base Language. Pub. S 22. Norvegian Computing Center. Oslo. (1970)

(DIJ68) DIJKSTRA E.W. Cooperating Sequential Processes. In Programming Languages. Ed. Genuys Academic Press. (1968)

(DIJ75) DIJKSTRA E.W. Guarded Commands, non-determinacy and formal derivation of programs. CACM vol. 18 n° 8. (aout 1975)

(DUF76) DUFOURD J.F. Modèle dynamique de système de traitements d'informations en organisation administrative. CRIN publication n° 76-R-006. Nancy (1976)

(DUF78a) DUFOURD J.F., KLEIN P. Ordonnancement des processus dans le projet MAESTRO. BIGRE n°9. Rennes. (avril 1978)

(DUF78b) DUFOURD J.F., KLEIN P. Projet MAESTRO : Outils de base pour l'écriture de maquettes. CRIN n° 78-R-029. Nancy.(1978)

(DUF78c) DUFOURD J.F. Types abstraits, modèle relationnel et langage SIMULA. CRIN n° 78-R-027. Nancy.(1978)

(DUF78d) DUFOURD J.F., HERTSCHUH N., KLEIN P. Le projet MAESTRO. Congrès AFCET. Gif sur Yvette. (novembre 1978)

(DUF79a) DUFOURD J.F. Modélisation des systèmes répartis dans les organisations. BIGRE à paraître. Rennes. (1979)

(DUF79b) DUFOURD J.F. Maquettes de Systèmes d'Information. Conférence CIPS/DPMA/FIQ. Canada. (1979)

(FIS73) FISHMANN G.S. Concepts and methods in discrete event digital simulation. Ed. Wiley and sons. New-York. (1973)

(FLO77) FLORY A. Un modèle et une méthode pour la conception logique d'une base de données. Thèse d'état. Lyon. (1977)

(FOR61) FORRESTER J.W. Industrial dynamics. Ed. M.I.T. Press. Cambridge, Mass. (1961)

(GAS69) PRITSKER A.A., KIVIAT P.J. Simulation with GASP II. Prentice-Hall. Englewood Cliffs. N.J. (1969)

(GIR77) GIRAUDIN J.P. Le projet MACSI-P. Thèse de troisième cycle. Grenoble. (1977)

(GIR78) GIRAULT C. Réseaux de Petri et synchronisation de processus. Journées informatiques de Nice. (1978)

(GUT77) GUTTAG J.V. Spécification algébrique de types abstraits. Bulletin de l'AFCET, GROPLAN n° 2. (1977)

(HAB72) HABERMANN A.N. Synchronization of communicating processes. CACM vol. 15 n°3. (1972)

(HAB75) HABERMANN A.N. Path expressions. Carnegie Mellon University. (1975)

(HOA74) HOARE C.A.R. Monitors : An operating system structuring concept. CACM vol.17 n° 10. (1974)

(HOA78) HOARE C.A.R. Communicating Sequential Processes. CACM vol.21 n°8. (Aout 1978)

(HUR76) HURTUBISE R. Informatique et information. Ed. Agence d'Arc. Ottawa. et Ed. d'Organisation. Paris. (1976)

(IBM70) IBM. General Purpose Simulation System/360. User's manual GH 20-0326. White Plains. N.Y. (1970)

(KAU76) KAUBISCH W.H., PERROTT R.H., HOARE C.A.R. Quasiparallel programming. Software practice and experience. vol.16. Ed. Wiley and sons. New-York. (1976)

(KES77) KESSELS J.L.W. An alternative to event queues for synchronization in monitors. CACM vol.20 n°7. (1977)

(KLE78) KLEIN P., DUFOURD J.F. Projet MAESTRO : Exemple de maquette : Gestion des commandes et facturation dans une entreprise commerciale. CRIN publication n° 78-R-077. Nancy. (1978)

(KNU64) KNUTH D.E. et al. SOL-A symbolic language for general-purpose system simulation. IEEE Transactions on electronic computers. (1964)

(KNU68) KNUTH D.E. The art of computer programming, vol.1 Ed. Addison Wesley. (1968)

(LAR79) LAROUSSE. Petit Larousse illustré. Paris. (1979)

(LER77) LEROUDIER J. Systèmes adaptatifs à mémoire virtuelle. Thèse d'état. Grenoble. (1977)

(LIS77) LISKOV B. et al. Abstraction mechanisms in CLU. CACM vol.20 n°8. (1977)

(LIV78) LIVERCY. Théorie des programmes. Ed. Dunod. Paris (1978)

(LUG75) LUGUET J. SCAPFACE - Système de conception automatisée et progressive d'un système de base de données fondé sur l'analyse conversationnelle des états de sortie. Thèse d'état. Toulouse. (1975)

(MAL71) MALLET R.A. La méthode informatique. Ed. Hermann. Paris. (1971)

(MAR63) MARKOWITZ H.M. et al. SIMSCRIPT-A simulation programming language. Prentice-Hall.N.J. (1963)

(PEC75) PECCOUD F. Le projet MACSI - Méthode d'aide à la conception de systèmes d'information. Thèse d'état. Grenoble. (1975)

(PET77) PETERSON J.L. Petri nets. Computing surveys. Vol.9 n°3 (1977)

(PRO70) PROTEE. Système d'analyse. Société d'informatique et de systèmes. Paris. (1970)

(PUL78) PULOU J. Un outil pour la spécification de la synchronisation dans les langages de haut niveau. RAIRO vol.12 n°4. (1978)

(RAY78) RAYNAL M. Une expression de la synchronisation pour les types abstraits. RAIRO vol.12 n°4. (1978)

(REM74) REMY J.L. Structures d'information. Formalisation des notions d'accès et de modifications de données. Thèse de troisième cycle. Nancy. (1974)

(ROL79) ROLLAND C. et al. Projet REMORA. Séminaires et thèses sur le projet Remora. Nancy. (1974-1979)

(ROU78) ROUCAIROL G.P. Mots de synchronisation. RAIRO vol.12 n°4. (1978)

(SAA70) SAAL H.J., RIDDLE W.E. Communicating semaphore. SLAC CGTM, 117. Stanford. (1970)

(TEI73) TEICHROEW D. et al. An introduction to computer based information processing system. ISDOS Working paper n°72. (1973)

(THA79) THAUREL M. Aspects statistiques du projet MAESTRO. Thèse de troisième cycle. Nancy. (à paraître)

(VAU73) VAUCHER J. A wait until algorithm for general purpose simulation language. Proc. Winter Simulation Conference. San Fransisco. (1973)

(VAU77) VAUCHER J., BRATLEY P. Cours de simulation. Ecole d'été de l' AFCET. Montréal. (1977)

(VAU78) VAUCHER J., DAVEY D. Accélération du wait until pour l'ordonnancement conditionnel en simulation. Congrès AFCET. Gif sur Yvette. (1978)

(VEI72) VEILLON F., CAGNAT J.F. Cours de programmation en langage PL/1. Vol.3 : Concepts avancés. Coll. U Armand Colin. Paris. (1972)

(WIR77) WIRTH N. MODULA : A programming language for modular multiprogramming. Software practice and experience. Vol.17. ed. Wiley and sons. New-York. (1977)

CHAPITRE IX

ANNEXES

PLAN DETAILLE

ANNEXE 1.

EXEMPLE DE FONCTIONNEMENT DU MECANISME DE MONITEUR

Nous présentons un exemple de programme illustrant le mécanisme de moniteur de HOARE (HOA74) réalisé en SIMULA. Nous avons choisi un système de producteurs-consommateurs (CRO75).

On dispose d'une file (ou liste) illimitée, appelée tampon, protégée par un moniteur qui assure l'exclusion mutuelle d'accès au tampon.

Par appel de la procédure recevoir, un processus consommateur reçoit un message (numéro) si tampon n'est pas vide, sinon il se bloque sur la condition nonvide. Par appel de la procédure envoyer, un processus producteur envoie un message (numéro) dans tampon, et peut ainsi débloquer un consommateur en attente sur nonvide.

L'exemple décrit utilise la classe MAESTRO comme préfixe. On y dispose de :

- trois producteurs appelés 'PRODUCTEUR1', 'PRODUCTEUR2' et 'PRODUCTEUR3' qui produisent respectivement des messages numérotés 1, 2 et 3. Chacun est activé toutes les deux unités de temps (heures par exemple) ;

- deux consommateurs appelés 'CONSOMMATEUR1' et 'CONSOMMATEUR2' qui tentent de consommer un message toutes les trois unités de temps.

Les résultats sont présentés sous deux formes :

- consommation d'un message :
  heure (en jours - heures - minutes), nom du consommateur, numéro du message consommé.

- production d'un message :
  heure, nom du producteur, numéro du message produit.

Le listing source et les résultats sont les suivants :

Remarque : La classe MAESTRO se trouve en annexe 2.

```
BIN
NESTPO BEGIN
 MONITOR CLASS TAMPON ;
 BEGIN
       INTEGER N;
       REF (HEAD) TAB ;
       REF (CONDITION) NONVIDE ;
       PROCEDURE ENVOYER (F) ; INTEGER F ;
       BEGIN
             THIS MONITOR.ENTER ;
             NEW MESSAGE (F).INTO (TAB) ;
             N := N + 1 ;
             NONVIDE.SIGLEAVE
       END ; $ ENVOYER $
       PROCEDURE RECEVOIR (G) ; NAME G ; INTEGER G ;
       BEGIN
             REF (MESSAGE) D ;
             THIS MONITOR.ENTER ;
             IF TAB.EMPTY THEN NONVIDE.CWAIT ;
             D :- TAB.FIRST ;
             G := D.NO ;
             TAB.FIRST.OUT ;
             N := N - 1 ;
             THIS MONITOR.LEAVE
       END ; $ RECEVOIR $
       N := 0 ;
       TAB :- NEW HEAD ;
       NONVIDE :- NEW CONDITION ( THIS MONITOR ) ;
END ; $ TAMPON $
PROCESSUS CLASS CONSOMMATEUR (TAMP);REF (TAMPON)TAMP;
BEGIN
       INTEGER F ;
       WHILE TRUE DO
       BEGIN
             TAMP.RECEVOIR (F) ;
             OUTTEXT ('                    ') ;
             ECRI (TIME) ;
             OUTTEXT (NOM) ;
             OUTINT (F,2) ;
             OUTIMAGE ;
             HOLD (T)
       END
END ; $ CONSOMMATEUR $
PROCESSUS CLASS PRODUCTEUR (N,TAMP); INTEGER N; REF (TAMPON) TAMP ;
BEGIN
       WHILE TRUE DO
       BEGIN
             TAMP.ENVOYER(N) ;
             OUTTEXT ('                    ') ;
             ECRI (TIME) ';
             OUTTEXT (NOM) ;
             OUTINT (N,2) ;
             OUTIMAGE ;
             HOLD (T)
```

```
        END
    END ; $ PRODUCTEUR $
    REF (TAMPON) TAMP ;
    REF (PRODUCTEUR) P1,P2,P3 ;
    REF (CONSOMMATEUR) C1,C2 ;
    TAMP :- NEW TAMPON (' TAMPON ') ;
    P1 :- NEW PRODUCTEUR ('  PRODUCTEUR1  ',1,2,1,TAMP) ;
    P2 :- NEW PRODUCTEUR ('  PRODUCTEUR2  ',1,2,2,TAMP ) ;
    P3 :- NEW PRODUCTEUR ('  PRODUCTEUR3  ',1,2,3,TAMP) ;
    C1 :- NEW CONSOMMATEUR ('  CONSOMMATEUR1  ',1,3,TAMP) ;
    C2 :- NEW CONSOMMATEUR ('  CONSOMMATEUR2  ',1,3,TAMP)  ;
    ACTIVATE C1 QUA PROCESS ;
    ACTIVATE C2 QUA PROCESS ;
    ACTIVATE P1 QUA PROCESS ;
    ACTIVATE P2 QUA PROCESS ;
    ACTIVATE P3 QUA PROCESS ;
    HOLD (50)
  END
END $ PROGRAMME $#
```



```
 0  0    CONSOMMATEUR1   1
 0  0  0   PRODUCTEUR1   1
 0  0    CONSOMMATEUR2   2
 0  0  0   PRODUCTEUR2   2
 0  0  0   PRODUCTEUR3   3
 0  2  0   PRODUCTEUR1   1
 0  2  0   PRODUCTEUR2   2
 0  2  0   PRODUCTEUR3   3
 3  0    CONSOMMATEUR1   3
 3  0    CONSOMMATEUR2   1
 0  4  0   PRODUCTEUR1   1
 0  4  0   PRODUCTEUR2   2
 0  4  0   PRODUCTEUR3   3
 6  0    CONSOMMATEUR1   2
 6  0    CONSOMMATEUR2   3
 0  6  0   PRODUCTEUR1   1
 0  6  0   PRODUCTEUR2   2
 0  6  0   PRODUCTEUR3   3
 8  0  0   PRODUCTEUR1   1
 8  0  0   PRODUCTEUR2   2
 8  0  0   PRODUCTEUR3   3
 9  0    CONSOMMATEUR1   1
 9  0    CONSOMMATEUR2   2
10  0  0   PRODUCTEUR1   1
10  0  0   PRODUCTEUR2   2
10  0  0   PRODUCTEUR3   3
12  0    CONSOMMATEUR1   3
12  0    CONSOMMATEUR2   1
12  0  0   PRODUCTEUR1   1
12  0  0   PRODUCTEUR2   2
12  0  0   PRODUCTEUR3   3
14  0  0   PRODUCTEUR1   1
14  0  0   PRODUCTEUR2   2
14  0  0   PRODUCTEUR3   3
15  0    CONSOMMATEUR1   2
15  0    CONSOMMATEUR2   3
16  0  0   PRODUCTEUR1   1
16  0  0   PRODUCTEUR2   2
16  0  0   PRODUCTEUR3   3
18  0    CONSOMMATEUR1   1
18  0    CONSOMMATEUR2   2
18  0  0   PRODUCTEUR1   1
18  0  0   PRODUCTEUR2   2
18  0  0   PRODUCTEUR3   3
20  0  0   PRODUCTEUR1   1
20  0  0   PRODUCTEUR2   2
20  0  0   PRODUCTEUR3   3
21  0    CONSOMMATEUR1   3
21  0    CONSOMMATEUR2   1
22  0  0   PRODUCTEUR1   1
```

```
    0 22  0    PRODUCTEUR2   2
    0 22  0    PRODUCTEUR3   3
1   0  0     CONSOMMATEUR1   2
1   0  0     CONSOMMATEUR2   3
    1  0  0    PRODUCTEUR1   1
    1  0  0    PRODUCTEUR2   2
    1  0  0    PRODUCTEUR3   3
    1  2  0    PRODUCTEUR1   1
    1  2  0    PRODUCTEUR2   2
    1  2  0    PRODUCTEUR3   3
1   3  0     CONSOMMATEUR1   1
1   3  0     CONSOMMATEUR2   2
    1  4  0    PRODUCTEUR1   1
    1  4  0    PRODUCTEUR2   2
    1  4  0    PRODUCTEUR3   3
1   6  0     CONSOMMATEUR1   3
1   6  0     CONSOMMATEUR2   1
    1  6  0    PRODUCTEUR1   1
    1  6  0    PRODUCTEUR2   2
    1  6  0    PRODUCTEUR3   3
    1  8  0    PRODUCTEUR1   1
    1  8  0    PRODUCTEUR2   2
    1  8  0    PRODUCTEUR3   3
1   9  0     CONSOMMATEUR1   2
1   9  0     CONSOMMATEUR2   3
    1 10  0    PRODUCTEUR1   1
    1 10  0    PRODUCTEUR2   2
    1 10  0    PRODUCTEUR3   3
1  12  0     CONSOMMATEUR1   1
1  12  0     CONSOMMATEUR2   2
    1 12  0    PRODUCTEUR1   1
    1 12  0    PRODUCTEUR2   2
    1 12  0    PRODUCTEUR3   3
    1 14  0    PRODUCTEUR1   1
    1 14  0    PRODUCTEUR2   2
    1 14  0    PRODUCTEUR3   3
1  15  0     CONSOMMATEUR1   3
1  15  0     CONSOMMATEUR2   1
    1 16  0    PRODUCTEUR1   1
    1 16  0    PRODUCTEUR2   2
    1 16  0    PRODUCTEUR3   3
1  18  0     CONSOMMATEUR1   2
1  18  0     CONSOMMATEUR2   3
    1 18  0    PRODUCTEUR1   1
    1 18  0    PRODUCTEUR2   2
    1 18  0    PRODUCTEUR3   3
    1 20  0    PRODUCTEUR1   1
    1 20  0    PRODUCTEUR2   2
    1 20  0    PRODUCTEUR3   3
1  21  0     CONSOMMATEUR1   1
1  21  0     CONSOMMATEUR2   2
    1 22  0    PRODUCTEUR1   1
    1 22  0    PRODUCTEUR2   2
    1 22  0    PRODUCTEUR3   3
2   0  0     CONSOMMATEUR1   3
2   0  0     CONSOMMATEUR2   1
    2  0  0    PRODUCTEUR1   1
    2  0  0    PRODUCTEUR2   2
    2  0  0    PRODUCTEUR3   3
```


ANNEXE 2.

LA CLASSE MAESTRO

Nous donnons ici, la majeure partie de la classe prédéfinie MAESTRO utile à l'écriture de maquettes. Seules figurent les déclarations de classes que l'utilisateur doit connaître pour pouvoir écrire ses maquettes.

La liste des classes décrites ici se trouve page V.2.

```
***    LA CLASSE MAESTRO (PROLOGUE PRECOMPILE)
***


LATION CLASS MAESTRO ;
GIN

***
***  LA CLASSE MONITOR
***
INK CLASS MONITOR (NOM) ; TEXT NOM ;
EGIN
    REF (HEAD) ENTERQ , URGENTQ ;
    BOOLEAN BUSY ;
    PROCEDURE ENTER ;
    BEGIN
          IF BUSY THEN WAIT (ENTERQ) ;
          BUSY := TRUE ;
    END ; $ ENTER $
    PROCEDURE LEAVE ;
    BEGIN
          IF NOT URGENTQ.EMPTY
          THEN BEGIN
                ACTIVATE URGENTQ.FIRST QUA PROCESS AFTER CURRENT ;
                URGENTQ.FIRST.OUT
          END
          ELSE IF NOT ENTERQ.EMPTY
                THEN BEGIN
                      ACTIVATE ENTERQ.FIRST QUA PROCESS AFTER CURRENT
                      ENTERQ.FIRST.OUT
                END
                ELSE BUSY := FALSE
    END ; $ LEAVE $
    ENTERQ :- NEW HEAD ;
    URGENTQ :- NEW HEAD ;
    BUSY := FALSE
ND ; $ MONITOR $
***
***  LA CLASSE CONDITION
***
LASS CONDITION (M) ; REF (MONITOR) M ;
EGIN
    REF (HEAD) WAITQ ;
    PROCEDURE CWAIT ;
    BEGIN
          IF NOT M.URGENTQ.EMPTY
          THEN BEGIN
                ACTIVATE M.URGENTQ.FIRST QUA PROCESS AFTER CURRENT ;
                M.URGENTQ.FIRST.OUT
          END
          ELSE IF NOT M.ENTERQ.EMPTY
                THEN BEGIN
```

```
                        ACTIVATE M.ENTERQ.FIRST QUA PROCESS AFTER CURRE
                        M.ENTERQ.FIRST.OUT
                  END
                  ELSE M.BUSY := FALSE ;
            WAIT (WAITQ)
      END ; $ CWAIT $
      PROCEDURE CSIGNALG ;
      BEGIN
            WHILE NOT WAITQ.EMPTY DO
            BEGIN
                  ACTIVATE WAITQ.FIRST QUA PROCESS AFTER CURRENT ;
                  WAITQ.FIRST.OUT ;
                  WAIT (M.URGENTQ)
            END
      END ; $ CSIGNALG $
      PROCEDURE SIGLEAVE ;
      BEGIN
            IF NOT WAITQ.EMPTY
            THEN BEGIN
                  ACTIVATE WAITQ.FIRST QUA PROCESS AFTER CURRENT ;
                  WAITQ.FIRST.OUT ;
                  WAIT (M.URGENTQ)
            END ;
            IF NOT M.URGENTQ.EMPTY
            THEN BEGIN
                  ACTIVATE M.URGENTQ.FIRST QUA PROCESS AFTER CURRENT ;
                  M.URGENTQ.FIRST.OUT
            END
            ELSE IF NOT M.ENTERQ.EMPTY
                  THEN BEGIN
                        ACTIVATE M.ENTERQ.FIRST QUA PROCESS AFTER CURR
                        M.ENTERQ.FIRST.OUT
                  END
                  ELSE M.BUSY := FALSE ;
      END ; $ SIGLEAVE $
      BOOLEAN PROCEDURE EMPTY ;
      EMPTY := WAITQ.EMPTY ;
      WAITQ :- NEW HEAD
END ; $ CONDITION $
**********
**********  LA CLASSE CONDITIONP
**********
CLASS CONDITIONP (M) ; REF (MONITOR) M ;
BEGIN
      REF (HEAD) WAITQ ;
      PROCEDURE CWAIT(PR,P) ; REF (PROCESS) PR ; REAL P ;
      BEGIN
            IF PR == CURRENT
            THEN BEGIN
                  IF NOT M.URGENTQ.EMPTY
                  THEN BEGIN
                        ACTIVATE M.URGENTQ.FIRST QUA PROCESS AFTER CURR
                        M.URGENTQ.FIRST.OUT
                        END
                  ELSE IF NOT M.ENTERQ.EMPTY
                        THEN BEGIN
                              ACTIVATE M.ENTERQ.FIRST QUA PROCESS AFTER
                              M.ENTERQ.FIRST.OUT
                              END
                        ELSE M.BUSY := FALSE ;
```



```
                  WAITP (PR,P,-1)
                  END
            ELSE WAITP (PR,P,PR.EVTIME - TIME)
      END ; $ CWAIT $
      PROCEDURE CSIGNAL ;
      BEGIN
            IF NOT WAITQ.EMPTY
            THEN BEGIN
                  REF (ELEMENT) X ;
                  X :- WAITQ.FIRST ;
                  X.OUT ;
                  IF X.DT = -1
                  THEN BEGIN
                        ACTIVATE X.PR QUA PROCESS AFTER CURRENT ;
                        WAIT (M.URGENTQ)
                        END
                  ELSE IF X.DT = 0
                        THEN REACTIVATE X.PR QUA PROCESS AFTER CURRENT
                        ELSE REACTIVATE X.PR QUA PROCESS DELAY X.DT PRIOR;
            END ;
      END ; $ CSIGNAL $
      PROCEDURE WAITP (PR,P,DT) ; REF (PROCESS) PR ; REAL P,DT ;
      BEGIN
            REF (ELEMENT) X,Y ;
            X :- WAITQ.FIRST ;
            WHILE (IF X == NONE THEN FALSE ELSE P <= X.P) DO X :- X.SUC ;
            Y :- NEW ELEMENT (PR,P,DT) ;
            IF X == NONE
            THEN Y.INTO (WAITQ)
            ELSE Y.PRECEDE (X) ;
            CANCEL (PR)
      END ; $ WAITP $
      REF (PROCESS) PROCEDURE CFIRST ;
      CFIRST :- WAITQ.FIRST QUA ELEMENT.PR ;
      BOOLEAN PROCEDURE CEMPTY ; CEMPTY := WAITQ.EMPTY ;
      WAITQ :- NEW HEAD
; $ CONDITIONP $

*
*  LA CLASSE ACTMONITOR
*
ITOR CLASS ACTMONITOR (TYPE) ; INTEGER TYPE ;
IN
      REF (HEAD) ACTQ ;
      REF (CONDITION) DEBLOCAGE  ;
      PROCEDURE ACQUERIR (D) ; NAME D ; REF (MESSAGE) D ;
      BEGIN
            THIS MONITOR.ENTER ;
            IF ACTQ.EMPTY
            THEN DEBLOCAGE.CWAIT ;
            D :- ACTQ.FIRST ;
            D.OUT ;
            THIS MONITOR.LEAVE
      END ; $  ACQUERIR  $
      PROCEDURE ENVOYER (D) ; REF (MESSAGE) D ;
      BEGIN
            THIS MONITOR.ENTER ;
            IF NOT ( DEBLOCAGE.EMPTY AND ( TYPE = 1 OR ( TYPE = 3 AND NOT
            ACTQ.EMPTY ) ) )
            THEN D.INTO (ACTQ) ;
            DEBLOCAGE.SIGLEAVE
```

```
        END ; $  ENVOYER  $
        ACTQ :- NEW HEAD ;
        DEBLOCAGE :- NEW CONDITION (THIS MONITOR)
    END ; $  ACTMONITOR  $
%%%%%%%%%%
%%%%%%%%%%  LA CLASSE ECHEANCIER
%%%%%%%%%%
    CLASS ECHEANCIER ;
    BEGIN
        REF (HEAD) ECHQ ;
        INTEGER PER.HDP ;
        REF (ACTIVATION) ALPHA ;
        PROCEDURE VIDERECH (A) ; REF (ACTMONITOR) A ;
        BEGIN
            REF (EVNOTICE) X ;
            X :- ECHQ.FIRST ;
            WHILE X =/= NONE DO
            BEGIN
                REF (EVNOTICE) Y ;
                Y :- X.SUC ;
                IF X.ACTM == A THEN X.OUT ;
                X :- Y
            END ;
        END ; $ VIDERECH $
        REF (EVNOTICE) PROCEDURE PROCHEVNOTICE ;
        BEGIN
            REF (EVNOTICE) X ;
            X :- ECHQ.FIRST ;
            WHILE (X =/= NONE AND X.DATACT < TIME) DO
                X :- X.SUC ;
            PROCHEVNOTICE :- X
        END ; $ PROCHEVNOTICE $
        PROCEDURE MODIFECH (A) ; REF (ACTMONITOR) A ;
        BEGIN
            BOOLEAN FIN ;
            REAL D ;
            REF (EVNOTICE) X ;
            VIDERECH (A) ;
            LECTUREDATE (D,FIN) ;
            X :- ECHQ.FIRST ;
            WHILE (NOT FIN AND X =/= NONE ) DO
                IF D >= X.DATACT
                THEN X :- X.SUC
                ELSE BEGIN
                    NEW EVNOTICE (D,A).PRECEDE (X) ;
                    LECTUREDATE (D,FIN)
                END ;
            WHILE NOT FIN DO
            BEGIN
                NEW EVNOTICE (D,A).INTO (ECHQ) ;
                LECTUREDATE (D,FIN)
            END ;
            INIMAGE ;
            ALPHA.COUR :- PROCHEVNOTICE ;
            REACTIVATE ALPHA AT ALPHA.COUR.DATACT + HDP
        END ; $ MODIFECH $
        ECHQ :- NEW HEAD ;
        ALPHA :- NEW ACTIVATION (THIS ECHEANCIER )
    END ; $ ECHEANCIER $
```



```
%%%
%%%  DECLARATIONS DES CLASSES DE RESSOURCES
%%%

%%%
%%%  LA CLASSE CALRC
%%%
LENDRIER CLASS CALRC ;
EGIN
    BOOLEAN OCCUPE ;
    REF (CONDITION) LIBRE ;
    PROCEDURE ACQUERIR ;
    BEGIN
        THIS MONITOR.ENTER ;
        IF B THEN CALCONSULTER ;
        IF OCCUPE THEN LIBRE.CWAIT ;
        STATACQUERIR (1) ;
        OCCUPE := TRUE ;
        THIS MONITOR.LEAVE
    END ; $ ACQUERIR $
    PROCEDURE LIBERER ;
    BEGIN
        THIS MONITOR.ENTER ;
        OCCUPE := FALSE ;
        STATLIBERER (1) ;
        STATSERVIS ;
        IF B THEN CALSORTIR ;
        LIBRE.SIGLEAVE
    END ; $ LIBERER $
    LIBRE :- NEW CONDITION (THIS MONITOR ) ;
    STATINIT (1) ;
    INTO (CALRCQ) ;
    OCCUPE := FALSE
D ; $ CALRC $
%%%
%%%  LA CLASSE CALRAMINF
%%%
LENDRIER CLASS CALRAMINF ;
EGIN
    PROCEDURE ACQUERIR ;
    BEGIN
        IF B THEN CALCONSULTER ;
        STATACQUERIR (1)
    END ; $ ACQUERIR $
    PROCEDURE LIBERER ;
    BEGIN
        STATLIBERER (1) ;
        IF B THEN CALSORTIR ;
        STATSERVIS
    END ; $ LIBERER $
    STATINIT (1) ;
    INTO (RAMINFQ) ;
D ; $ RAMINF $
%%%
%%%  LA CLASSE CALRAM
%%%
LENDRIER CLASS CALRAM (QUANTITE) ; REAL QUANTITE ;
EGIN
```

```
        REF (CONDITION) RESSUFFI ;
        PROCEDURE ACQUERIR (N) ; REAL N ;
        BEGIN
              THIS MONITOR.ENTER ;
              IF B THEN CALCONSULTER ;
              IF N > QUANTITE
              THEN RESSUFFI.CWAIT ;
              QUANTITE := QUANTITE - N ;
              STATACQUERIR (N) ;
              THIS MONITOR.LEAVE
        END ; $ ACQUERIR $
        PROCEDURE LIBERER (N) ; REAL  N ;
        BEGIN
              THIS MONITOR.ENTER ;
              QUANTITE := QUANTITE + N ;
              STATLIBERER (N) ;
              STATSERVIS ;
              IF B THEN CALSORTIR ;
              RESSUFFI.SIGLEAVE
        END ; $  LIBERER  $
        STATINIT (QUANTITE) ;
        INTO (CALRAMQ ) ;
        RESSUFFI :- NEW CONDITION (THIS MONITOR)
  END ; $ CALRAM $
**********
**********  LA CLASSE CALCPP
**********
  CALENDRIER CLASS CALCPP (K) ; REAL K ;
  BEGIN
        BOOLEAN OCCUPE ; INTEGER N ;
        REF (CONDITION) LIBRE ;
        PROCEDURE REQUERIR (X,QU) ; NAME QU ; REAL X,QU ;
        BEGIN
              THIS MONITOR.ENTER ;
              IF OCCUPE THEN LIBRE.CWAIT ;
              QU := MIN ( X , K/N ) ;
              OCCUPE := TRUE ;
              STATACQUERIR (1) ;
              IF THIS CALENDRIER.B THEN CALCONSULTER ;
              THIS MONITOR.LEAVE
        END ; $ REQUERIR $
        PROCEDURE LIBERER (X,QU) ; NAME X ; REAL X,QU ;
        BEGIN
              THIS MONITOR.ENTER ;
              X := X - QU ;
              IF X = 0  THEN N := N - 1 ;
              IF NOT LIBRE.EMPTY
              THEN LIBRE.SIGLEAVE
              ELSE BEGIN
                    OCCUPE := FALSE ;
                    STATLIBERER (1) ;
                    THIS MONITOR.LEAVE
              END ;
              IF THIS CALENDRIER.B THEN CALSORTIR ;
        END ; $ LIBERER $
        OCCUPE := FALSE ;
        N := 0 ;
        LIBRE :- NEW CONDITION (THIS MONITOR) ;
        THIS CALENDRIER.ACCES := 1
  END ; $ CPP $
```



```
**
**  LA CLASSE CALRCP
**
ENDRIER CLASS CALRCP ;
IN
  BOOLEAN OCCUPE ;
  REF (CONDITIONP) LIBRE ;
  PROCEDURE ACQUERIR ;
  BEGIN
        THIS MONITOR.ENTER ;
        IF B THEN CALCONSULTER ;
        IF OCCUPE
        THEN LIBRE.CWAIT (CURRENT,CURRENT QUA PROCESSUS.PRIORITE) ;
        STATACQUERIR (1) ;
        OCCUPE := TRUE ;
        THIS MONITOR.LEAVE
  END ; $ ACQUERIR $
  PROCEDURE LIBERER ;
  BEGIN
        THIS MONITOR.ENTER ;
        IF LIBRE.CEMPTY
        THEN BEGIN
                STATLIBERER (1) ;
                STATSERVIS ;
                OCCUPE := FALSE
             END
        ELSE LIBRE.CSIGNAL ;
        IF B THEN CALSORTIR ;
        THIS MONITOR.LEAVE
  END ; $ LIBERER $
  STATINIT (1) ;
  INTO (CALRCPQ) ;
  OCCUPE := FALSE ;
  LIBRE :- NEW CONDITIONP (THIS MONITOR)
; $ CALRCP $
*
*  LA CLASSE CALRCPP
*
ENDRIER CLASS CALRCPP ;
IN
  REF (PROCESSUS) OCCUPANT ;
  REF (CONDITIONP) TOUR ;
  PROCEDURE ACQUERIR ;
  BEGIN
        REF (NOTICE) X ;
        THIS MONITOR.ENTER ;
        IF B THEN CALCONSULTER ;
        IF OCCUPANT =/= NONE
        THEN BEGIN
              IF OCCUPANT.PRIORITE < CURRENT QUA PROCESSUS.PRIORITE
              THEN TOUR.CWAIT(OCCUPANT,OCCUPANT.PRIORITE)
            ELSE TOUR.CWAIT(CURRENT,CURRENT QUA PROCESSUS.PRIORITE);
             END
        ELSE STATACQUERIR (1) ;
        OCCUPANT :- CURRENT ;
        THIS MONITOR.LEAVE
  END ; $ ACQUERIR $
  PROCEDURE LIBERER ;
  BEGIN
        THIS MONITOR.ENTER ;
```

```
            IF TOUR.CEMPTY
            THEN BEGIN
            STATLIBERER (1) ;
            STATSERVIS ;
                  OCCUPANT :- NONE
                  END
            ELSE BEGIN
                  OCCUPANT :- TOUR.CFIRST ;
                  TOUR.CSIGNAL
                  END ;
            IF B THEN CALSORTIR ;
            THIS MONITOR.LEAVE
      END ; $ LIBERER $
      STATINIT (1) ;
      INTO (CALRCPPQ) ;
      TOUR :- NEW CONDITIONP (THIS MONITOR)
END ; $ CALRCPP $



**********
**********  LA CLASSE TREGION
**********
      LINK CLASS TREGION (NOM) ; TEXT NOM ;
      BEGIN
            INTEGER RGENT , RGSORT , RGMAX ;
            REAL RGTPSAT , RGTPSMAX , RGTPSMIN ;
            PROCEDURE ENTRER (TR) ; REF (TRANSACTION) TR ;
            BEGIN
                  INSPECT TR DO
                  BEGIN
                  HDEBRG := TIME ;
                  RGENT := RGENT + 1 ;
                  IF RGENT - RGSORT > RGMAX THEN RGMAX := RGENT - RGSORT
                  END ;
            END ; $ ENTRER $
            PROCEDURE SORTIR (TR) ; REF (TRANSACTION) TR ;
            BEGIN
                  REAL TPSE ;
                  INSPECT TR DO
                  BEGIN
                  TPSE := TIME - HDEBRG ;
                  RGSORT := RGSORT + 1 ;
                  IF RGSORT = 1 THEN RGTPSMIN := RGTPSMAX := TPSE ;
                  IF TPSE < RGTPSMIN THEN RGTPSMIN := TPSE ;
                  IF TPSE > RGTPSMAX THEN RGTPSMAX := TPSE ;
                  RGTPSAT := RGTPSAT + TPSE
                  END ;
            END ; $ SORTIR $
            PROCEDURE REINIT ;
            BEGIN
                  RGMAX := RGENT := RGENT - RGSORT ;
                  RGSORT := 0 ;
                  RGTPSMIN := RGTPSMAX := RGTPSAT := 0 ;
            END ; $ INIT $
            IF NOM.LENGTH > 15 THEN NOM :- NOM.SUB (1,15) ;
            INTO (REGIONQ )
      END ; $ TREGION $
**********
**********  LA CLASSE TFILE
```



```
***
MONITOR CLASS TFILE (MESURE,EQUILIBRE,INTER) ;
REAL INTER ;
BOOLEAN MESURE,EQUILIBRE ;
BEGIN
      REF (HEAD) Q ;
      INTEGER FAPRE,FAMAX,FANUL,FASORT,NBMES,NOFILE ;
      REAL TVOL,FATPSAT,VOL,VOLMIN,VOLMAX,TPSE ,HMOD,FAPDS ;
      PROCEDURE ENTRER (T) ;REF (TRANSACTION) T ;
      BEGIN
            INSPECT T DO
            BEGIN
                  HDEBF := TIME ;
                  MAJVOL (1,PDS) ;
            END ;
            IF FAPRE > FAMAX THEN FAMAX := FAPRE ;
      END ;
      PROCEDURE SORTIR (T) ;REF (TRANSACTION) T ;
      BEGIN
            INSPECT T DO
            BEGIN
                  TPSE := TIME - HDEBF ;
                  MAJVOL (-1,PDS) ;
            END ;
            IF FAPRE = 0 THEN FANUL := FANUL + 1 ;
            FASORT := FASORT + 1 ;
            FATPSAT := FATPSAT + TPSE ;
      END ;
      PROCEDURE  MAJVOL (NBUNIT,PDSUNIT); INTEGER NBUNIT ; REAL PDSUNIT;
      BEGIN
            VOL := VOL + (TIME - HMOD) * FAPDS ;
            HMOD := TIME ;
            FAPDS := FAPDS + NBUNIT * PDSUNIT ;
            FAPRE := FAPRE + NBUNIT ;
       END ;
      PROCEDURE MESUREFILE ;
      BEGIN
            NBMES := NBMES + 1 ;
            IF NBMES = 1 THEN VOLMIN := VOLMAX := VOL ;
            VOLMIN := MIN (VOL,VOLMIN) ;
            VOLMAX := MAX (VOL,VOLMAX) ;
            TVOL := TVOL + VOL ;
      END ; $ MESURE $
       PROCEDURE REINIT ;
       BEGIN
            VOLMIN := VOLMAX := VOL ;
            NBMES := 0 ;
            FASORT := FANUL := 0 ;
            FATPSAT := 0 ;
            FAMAX := FAPRE ;
            TVOL := 0 ;
       END ;
       Q :- NEW HEAD ;
       IF NOM.LENGTH > 15 THEN NOM :- NOM.SUB (1,15) ;
       HMOD := TIME ;
       IF EQUILIBRE OR MESURE  THEN  INTO (FILEQ) ;
       IF EQUILIBRE
       THEN BEGIN
                  NOFILE := NBFILE := NBFILE + 1 ;
                  ACTIVATE NEW MESUREVOL(THIS TFILE) QUA PROCESS
```

```
                                        DELAY INTER ;
                END;
     END ; $ TFILE $
**********
**********  LA CLASSE TRANSACTION
**********
     MESSAGE CLASS TRANSACTION (PDS) ; REAL PDS ;
     BEGIN
          REAL HDEBF , HDEBRG ;
          NBMESS := NBMESS + 1 ;
     END ; $ TRANSACTION $
**********
**********  LA CLASSE PROCESSUS
**********
     PROCESS CLASS PROCESSUS (NOM,PRIORITE,T) ; TEXT NOM ; REAL PRIORITE ;
                                                          REAL T. ;
     NULL ;


**********
**********  PROCEDURE DE SERVICE PASSE
**********
     PROCEDURE PASSE (LIB,DEBUTE) ; TEXT LIB ; BOOLEAN DEBUTE ;
     BEGIN
          ECRI (TIME) ;
          IF DEBUTE THEN OUTTEXT('    DEBUT DE TRAVAIL DE   ')
                    ELSE OUTTEXT('           FIN DE TRAVAIL DE    ') ;
          OUTTEXT (LIB) ;
          OUTIMAGE
     END ; $ PASSE $


**********
**********  PROCEDURE UTILISERPP (PROCESSEURS PARTAGES)
**********
      PROCEDURE UTILISERPP (T,PP) ; REAL T ; REF (CALCPP) PP ;
     BEGIN
          REAL X,QU ;
          IF T > 0
          THEN BEGIN
          PP.N := PP.N + 1 ;
          X := T ;
          PP.STATUTILISATEUR (1) ;
          WHILE X > 0 DO
          BEGIN
                PP.REQUERIR (X,QU) ;
                HOLD (QU) ;
                PP.LIBERER (X,QU)
          END ;
          PP.STATUTILISATEUR (-1) ;
          PP.STATSERVIS ;
          END
     END ; $ UTILISERPP $



**********
**********  PROCEDURE DE LANCEMENT DE LA SIMULATION
**********
```



```
ROCEDURE SIMUL (DUREESIMUL,EDITRESGEN,EQUILIBRE,BFGEN) ;
EAL DUREESIMUL ;
OOLEAN EDITRESGEN, EQUILIBRE, BFGEN ;
EGIN
     IF EQUILIBRE THEN BOOL := TRUE ELSE BOOL := FALSE ;
     DURSIM := DUREESIMUL ;
     DEBSIMUL := TIME ;
     PAGE ;
     U := U + 9741 ;
     NBPAS := NBPAS + 1 ;
     IF NBPAS > 1 THEN REINITIALISATION ;
     IF EQUILIBRE THEN OUVRIRFVOL ;
     HOLD (DUREESIMUL) ;
     IF EDITRESGEN THEN STATGEN ;
     IF BFGEN THEN CONSTFGEN ;
     IF EQUILIBRE THEN FVOL.CLOSE
END ;  $  SIMUL  $


**  DECLARATIONS ET INITIALISATIONS DES
**  VARIABLES GLOBALES A LA CLASSE
**  MAESTRO


NER ;


**
**  FIN DE LA CLASSE MAESTRO
**
 $ MAESTRO $
```

ANNEXE 3.

MOTS RESERVES DE LA CLASSE MAESTRO

Nous donnons ici la liste par ordre alphabétique des identificateurs (noms de classes, et variables globales) utilisés dans la classe MAESTRO, afin d'éviter à l'utilisateur les ennuis causés par les doubles définitions.

ACT
ACTIVATION
ACTMONITOR
BOOL
BIDON
CALCPP
CALCPPQ
CALENDRIER
CALRAM
CALRAMINF
CALRAMPQ
CALRAMPPQ
CALRAMQ
CALRC
CALRCP
CALRCPP
CALRCPQ
CALRCPPQ
CALRCQ
CONDITION
CONDITIONP
CONSTFGEN
CONVJHM
CONVNH
CPPQ
DATE
DEBSIMUL
DESACT
DTRA
DURSIM
ECHE
ECHEANCIER
ECRI
ECRIFVOL
ELEMENT
EVNOTICE
FILEQ
FVOL
HDP
HISTOGRAMME
HISTOQ
LECTUREDATE
MESSAGE
MESUREVOL
MONITOR
NBFILE
NBMESS
NBPAS
NOTCAL
NOTICE
OUVRIRFVOL
PASSE
PER
PROCESSUS
RAMINFQ
REGIONQ
REINITIALISATION
RESSOURCE
SEPETOILE
SIMUL
STATGEN
TFILE
TRANSACTION
TREGION
U
UTILISERPP

ANNEXE 4.

EXEMPLE D'UTILISATION DES PROCESSEURS PARTAGES

Nous présentons dans cette annexe un exemple d'utilisation de ressources de type processeurs partagés avec et sans calendrier.

Pour cela nous préfixons notre programme par MAESTRO. Nous utilisons deux processeurs :

- PARTAGE est un processeur partagé sans calendrier ;
- CALPART est un processeur partagé avec calendrier

dont les périodes de disponibilité sont :

2 à 7, 9 à 12, 15 à 19, 21 à 1.1, 1.4 à 1.7, 1.9 à 1.13.
(Les dates sont données en jour, heure, minute).

Chaque processeur a un intervalle de temps à partager égal à 6 heures.

On définit une classe de processus (utilisant ces processeurs) appelée p. Chacun des processus signale ses début et fin de travail (par la procédure passe) global ainsi que les début et fin de possession du processeur considéré.

Ainsi les informations suivantes :

```
0  6  0    DEBUT DE TRAVAIL DE P3
0  8  0    DEBUT
0 10  0    FIN
0 10  0    FIN DE TRAVAIL DE P3
```

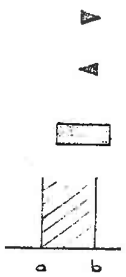
signifient que le processeur P3 a demandé pour la première fois le processeur à 6 h., qu'il a effectivement travaillé avec lui de 8 h à 10 h. et qu'il a quitté le système, son travail terminé à 10 h.

On utilise 12 processus dont les caractéristiques sont :

nom du processus	processeur partagé utilisé	durée de travail	date d'activation
p1	partage	7	0
p2	partage	2	3
p3	partage	2	6
p4	partage	6h 30	6h 30
p5	partage	4h 30	15h 30
p6	partage	2	19
p7	calpart	5	0
p8	calpart	3	4
p9	calpart	3	6h 30
p10	calpart	4h 30	12
p11	calpart	2	14
p12	calpart	3h 30	1j. 4h

Le listing du programme et les résultats obtenus sont donnés ci-après.

On peut schématiser ces résultats par les figures suivantes :

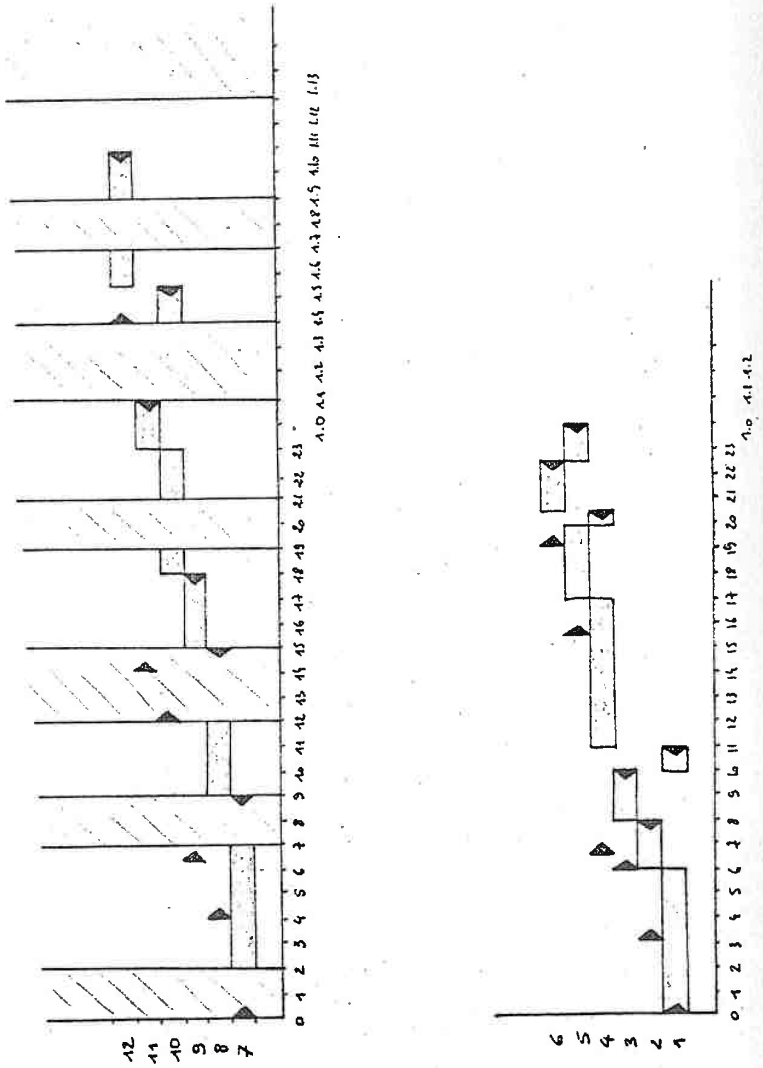


activation du processus (DEBUT DE TRAVAIL DE)

desactivation (FIN DE TRAVAIL DE)

travail du processus avec le processeur, période encadrée par DEBUT et FIN

période d'indisponibilité du processeur.




```
000     BEGIN
000     MAESTRO BEGIN
001      PROCESSUS CLASS P (PP) ; REF (CALCPP) PP ;
002      BEGIN
002         PASSE (NOM,TRUE) ;
003         UTILISERPP (T,PP) ;
003         PASSE (NOM,FALSE) ;
003      END ; $ P $
003      REF (CALCPP) PARTAGE ;
002      REF (CALCPP) CALPART ;
002      REF (P) P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12;
002      PARTAGE :- NEW CALCPP ('PARTAGE',FALSE,FALSE,6) ;
002      CALPART :- NEW CALCPP ('CALPART',TRUE,FALSE,6) ;
002      P1 :- NEW P ('P1',1,7,PARTAGE) ;
002      P2 :- NEW P ('P2',1,2,PARTAGE) ;
002      P3 :- NEW P ('P3',1,2,PARTAGE) ;
002      P4 :- NEW P ('P4',1,6.5,PARTAGE) ;
002      P5 :- NEW P ('P5',1,4.5,PARTAGE) ;
002      P6 :- NEW P ('P6',1,2,PARTAGE) ;
002      P7 :- NEW P ('P7',1,5,CALPART) ;
002      P8 :- NEW P ('P8',1 ,3,CALPART) ;
002      P9 :- NEW P ('P9',1,3,CALPART) ;
002      P10 :- NEW P ('P10',1,4.5,CALPART) ;
002      P11 :- NEW P ('P11',1,2,CALPART) ;
002      P12 :- NEW P ('P12',1,3.5,CALPART) ;
002      ACTIVATE P1 QUA PROCESS ;
002      ACTIVATE P2 QUA PROCESS AT 3 ;
002      ACTIVATE P3 QUA PROCESS AT 6 ;
002      ACTIVATE P4 QUA PROCESS AT 6.5 ;
002      ACTIVATE P5 QUA PROCESS AT 15.5 ;
002      ACTIVATE P6 QUA PROCESS AT 19 ;
002      ACTIVATE P7 QUA PROCESS ;
002      ACTIVATE P8 QUA PROCESS AT 4 ;
002      ACTIVATE P9 QUA PROCESS AT 6.5 ;
002      ACTIVATE P10 QUA PROCESS AT 12 ;
002      ACTIVATE P11 QUA PROCESS AT 14 ;
002      ACTIVATE P12 QUA PROCESS AT 28 ;
002      HOLD (72)
002     END $ MAESTRO $
002     END $ PROGRAMME $ #
```

```
LE CALENDRIER ASSOCIE A LA RESSOURCE   CALPART  EST LE SUIVANT :
 0  2  0    0  7  0    0  9  0    0 12  0    0 15  0    0 19  0    0 21  0  1
 1  4  0    1  7  0    1  9  0    1 13  0
 0  0  0       DEBUT DE TRAVAIL DE    P1
 0  0  0 P1    DEBUT
 0  0  0       DEBUT DE TRAVAIL DE    P7
 0  2  0 P7    DEBUT
 0  3  0       DEBUT DE TRAVAIL DE    P2
 0  4  0       DEBUT DE TRAVAIL DE    P8
 0  6  0 P1    FIN
 0  6  0       DEBUT DE TRAVAIL DE    P3
 0  6  0 P2    DEBUT
 0  6 30       DEBUT DE TRAVAIL DE    P4
 0  6 30       DEBUT DE TRAVAIL DE    P9
 0  7  0 P7    FIN
 0  7  0 P8    DEBUT
 0  8  0 P2    FIN
 0  8  0 P3    DEBUT
 0  8  0             FIN DE TRAVAIL DE    P2
 0  9  0             FIN DE TRAVAIL DE    P7
 0 10  0 P3    FIN
 0 10  0 P1    DEBUT
 0 10  0             FIN DE TRAVAIL DE    P3
 0 11  0 P1    FIN
 0 11  0 P4    DEBUT
 0 11  0             FIN DE TRAVAIL DE    P1
 0 12  0 P8    FIN
 0 12  0       DEBUT DE TRAVAIL DE    P10
 0 12  0 P9    DEBUT
 0 14  0       DEBUT DE TRAVAIL DE    P11
 0 15  0             FIN DE TRAVAIL DE    P8
 0 15 30       DEBUT DE TRAVAIL DE    P5
 0 17  0 P4    FIN
 0 17  0 P5    DEBUT
 0 18  0 P9    FIN
 0 18  0 P10   DEBUT
 0 18  0             FIN DE TRAVAIL DE    P9
 0 19  0       DEBUT DE TRAVAIL DE    P6
 0 20  0 P5    FIN
 0 20  0 P4    DEBUT
 0 20 30 P4    FIN
 0 20 30 P5    DEBUT
 0 20 30             FIN DE TRAVAIL DE    P4
 0 22 30 P6    FIN
 0 22 30 P5    DEBUT
 0 22 30             FIN DE TRAVAIL DE    P6
 0 23  0 P10   FIN
 0 23  0 P11   DEBUT
 1  0  0 P5    FIN
 1  0  0             FIN DE TRAVAIL DE    P5
 1  1  0 P11   FIN
 1  1  0             FIN DE TRAVAIL DE    P11
 1  4  0 P10   DEBUT
 1  4  0       DEBUT DE TRAVAIL DE    P12
 1  5 30 P10   FIN
 1  5 30 P12   DEBUT
 1  5 30             FIN DE TRAVAIL DE    P10
 1 11  0 P12   FIN
 1 11  0             FIN DE TRAVAIL DE    P12
EXECUTION TERMINEE
**************************************************************************
```


ANNEXE 5.

LISTING DE L'APPLICATION PRESENTEE AU CHAPITRE V.

Ce listing est l'image du programme de la maquette présentée au chapitre V. On y trouve successivement :

- la définition des transactions,
- la définition des processus,
- les déclarations des variables de travail,
- les déclarations des files,
- les déclarations des régions,
- les déclarations des processeurs,
- les déclarations des moniteurs d'activation,
- les initialisations des variables de travail,
- les créations des régions,
- les créations des files,
- les créations des processeurs,
- les créations des moniteurs d'activation,
- la création de l'échéancier,
- les initialisations des files fichiers,
- l'activation des processus,
- le lancement de la simulation,
- enfin les données des calendriers et de l'échéancier.

Ce programme illustre l'utilisation de la classe MAESTRO.

```
**** MODELE EXPERIMENTAL

**** UTILISATION DE LA CLASSE MAESTRO


ESTRO BEGIN
****
****  DEFINIION DES TRANSACTIONS
****
TRANSACTION CLASS BC (DAT,TYPE,LIGN) ; REAL DAT ; CHARACTER TYPE ;
INTEGER LIGN ; NULL ;
BC CLASS BCA (LA) ; INTEGER LA ; NULL ;
BCA CLASS BCL (LL) ; INTEGER LL ; NULL ;
****
****  DEFINITION DES PROCESSUS
****
PROCESSUS CLASS GENERATEUR ;
BEGIN
      WHILE TRUE DO
      BEGIN
            INTEGER I ;
            REF (MESSAGE) D ;
            A1.ACQUERIR (D) ;
                                                        PASSE (NOM,TRUE)
            FOR I := 1 STEP 1 UNTIL 15 DO
            BEGIN
            REF (BC) X;
                  X :- NEW BC (NUMEROTATION,0,TIME,"E",10) ;
                  ECR.ENTRER (X) ;
                  X.INTO (LBC.Q) ;
                  LBC.ENTRER (X)
            END ;
                                                        PASSE(NOM,FALSE)
      HOLD (0)
      END ;
END ; $ GENERATEUR $


PROCESSUS CLASS GENETEL (V) ; REF (CALRCPP) V ;
BEGIN
      WHILE TRUE DO
      BEGIN
            CHARACTER C ;
            REF (BC) BAPL ;
            IF V.HEURETRAVAIL
            THEN BEGIN
            IF DRAW (8/18,U) THEN C := "T"
                             ELSE C := "D" ;
                  BAPL :- NEW BC (NUMEROTATION,0,TIME,C,4) ;
                  OUTTEXT ('**********************************') ;
```

```
                ECRI (TIME) ;
                OUTTEXT ('    APPEL TELEPHONIQUE  ') ;
                ECRI (BAPL.DAT) ;
                OUTCHAR (BAPL.TYPE) ;
                OUTINT (BAPL.NO,5) ;
                OUTINT (BAPL.LIGN,3) ;
                OUTIMAGE ;
                BIDON :- BAPL ;
                TEL.ENTRER (BAPL) ;
          A5.ENVOYER (BIDON)
          END ;
          HOLD (NEGEXP (18/7,U))
    END
END ; $ GENETEL $


PROCESSUS CLASS CODBC (R) ; REF (CALRC) R ;
BEGIN
    WHILE TRUE DO
    BEGIN
          REF (MESSAGE) D ;
          REF ( BC ) X ;
          REAL DUREE ;
          A4.ACQUERIR (D) ;
          R.ACQUERIR ;
                                                    PASSE (NOM,TRU
          FOR X:- LBC.Q.FIRST WHILE X =/= NONE DO
          BEGIN
                DUREE := 0.013 + ( X.LIGN * 0.006 ) ;
                HOLD (DUREE) ;
                LBC.SORTIR (X) ;
                X.INTO (LBCI.Q) ;
                LBCI.ENTRER (X)
          END ;
                                                    PASSE(NOM,FAL
          R.LIBERER ;
          A3.ENVOYER (BIDON)
    END ;
END ; $ CODBC $


PROCESSUS CLASS SAISBC (R,ORD) ; REF (CALRC) R ; REF (CALRAMINF) ORD
BEGIN
    WHILE TRUE DO
    BEGIN
          REF (MESSAGE) D ;
          REF ( BC ) X ;
          REAL DUREE ;
          A3.ACQUERIR (D) ;
                                                    PASSE (NOM,TR
          FOR X :- LBCI.Q.FIRST WHILE X =/= NONE DO
          BEGIN
          DUREE := 0.006 + ( X.LIGN * 0.006) ;
          R.ACQUERIR ;
          HOLD (DUREE ) ;
          R.LIBERER ;
          DUREE := 0.002 + ( X.LIGN * 0.002) ;
          ORD.ACQUERIR ;
```



```
          HOLD (DUREE) ;
    ORD.LIBERER ;
          LBCI.SORTIR (X) ;
          X.INTO (LCOJO.Q) ;
          ECR.SORTIR (X) ;
          LCOJO.ENTRER (X)
    END ;
                                                    PASSE(NOM,FALSE)
END
D ; $ SAISBC $


OCESSUS CLASS SAISAPL (V,ORD) ; REF (CALRCPP) V ; REF (CALRAMINF) ORD ;
GIN
    WHILE TRUE DO
    BEGIN
          REF (MESSAGE) X ;
          REAL DUREE ;
          AS.ACQUERIR (X) ;
                                                    PASSE (NOM,TRUE) ;
          IF X QUA BC.TYPE = "D"
          THEN DUREE := 0.04
          ELSE BEGIN
                X QUA BC.INTO (LCOJO.Q) ;
                LCOJO.ENTRER (X QUA TRANSACTION ) ;
                TEL.SORTIR (X QUA BC) ;
                DUREE := 0.04 + 0.008 + (X QUA BC.LIGN * 0.007)
                END ;
          ORD.ACQUERIR ;
          HOLD (DUREE) ;
                                                    PASSE(NOM,FALSE) ;
          ORD.LIBERER ;
          V.ACQUERIR ;
          HOLD (DUREE) ;
          V.LIBERER
    END
D ; $ SAISAPL $


OCESSUS CLASS PROPLIV (ORD) ; REF (CALRAMINF) ORD ;
GIN
    WHILE TRUE DO
    BEGIN
          REF (BCA) ZA ;
          REF (BCL) ZL ;
          REF ( BC ) X ;
          REF (MESSAGE) D ;
          REAL A,B ;
          INTEGER N,M,I,NLL,NLA ;
          REF (BCA) Y ;
          A11.ACQUERIR (D) ;
                                                    PASSE (NOM,TRUE) ;
          FOR X :- LCOJO.Q.FIRST WHILE X =/= NONE DO
          X.INTO (LCOJOPRIM.Q) ;
          A := 0.4 ;
          X :- LCOJOPRIM.Q.FIRST ;
          N := 0 ;
          WHILE X =/=  NONE DO
```

```
BEGIN
      N := N + X.LIGN ;
      X :- X.SUC
END ;
Y :- LCOAT.Q.FIRST ;
M := 0 ;
WHILE Y =/= NONE DO
BEGIN
      M := M + Y.LA ;
      Y :- Y.SUC
END ;
IF M = 0 THEN B := 0 ELSE
B := N * ( 1 - A ) / M ;
Y :- LCOAT.Q.FIRST ;
WHILE Y =/= NONE DO
BEGIN
      NLL := 0 ;
      FOR I := 1 STEP 1 UNTIL Y.LA DO
          IF DRAW (B,U) THEN NLL := NLL + 1 ;
      NLA := Y.LA - NLL ;
      IF NOT NLA = 0 THEN
      BEGIN
            ZA :- NEW BCA (Y.NO,Y.POS,Y.DAT,Y.TYPE,Y.LIGN,
                           NLA ) ;
          ZA.INTO (NLCOAT.Q) ;
          IF COMPTEUR = 0 THEN LCOAT.ENTRER(ZA) ;
      END ;
      IF NOT NLL = 0 THEN
      BEGIN
            ZL :- NEW BCL (Y.NO,Y.POS,Y.DAT,Y.TYPE,Y.LIGN,
                           NLA,NLL ) ;
          ZL.INTO (LPL.Q) ;
          IF COMPTEUR = 0 THEN LPL.ENTRER (ZL) ;
          END ;
          IF COMPTEUR = 0 THEN LCOAT.SORTIR (Y) ;
       ORD.ACQUERIR ;
       HOLD ( NLL * T ) ;
       ORD.LIBERER ;
       Y :- Y.SUC
 END ;
 X :- LCOJOPRIM.Q.FIRST ;
 WHILE X =/= NONE DO
 BEGIN
       NLL := 0 ;
       FOR I := 1 STEP 1 UNTIL X.LIGN DO
           IF DRAW (A,U) THEN NLL := NLL + 1 ;
       NLA := X.LIGN -NLL ;
       IF NOT NLA = 0 THEN
       BEGIN
            ZA :- NEW BCA (X.NO,X.POS,X.DAT,X.TYPE,X.LIGN,
                           NLA ) ;
          ZA.INTO (NLCOAT.Q) ;
          IF COMPTEUR = 0 THEN LCOAT.ENTRER (ZA) ;
       END ;
       IF NOT NLL = 0 THEN
       BEGIN
            ZL :- NEW BCL (X.NO,X.POS,X.DAT,X.TYPE,X.LIGN,
                           NLA,NLL ) ;
          ZL.INTO (LPL.Q) ;
          IF COMPTEUR = 0 THEN LPL.ENTRER (ZL) ;
```



```
          END ;
       IF COMPTEUR = 0
       THEN BEGIN
            LCOJO.SORTIR (X) ;
            ADM.ENTRER (X)
       END ;
       ORD.ACQUERIR ;
       HOLD (NLL * T ) ;
       ORD.LIBERER ;
       X :- X.SUC
    END ;
                                                  PASSE(NOM,FALSE)
    A10.ENVOYER (BIDON)
END
ND ; $ PROPLIV $


ROCESSUS CLASS DECLIV (V) ; REF (CALRCPP) V ;
EGIN
    WHILE TRUE DO
    BEGIN
       REF (MESSAGE) D ;
       REAL A,B,C ;
       A10.ACQUERIR (D) ;
       V.ACQUERIR ;
                                                  PASSE (NOM,TRUE)
       COMPTEUR := COMPTEUR + 1 ;
       IF COMPTEUR = 1
       THEN BEGIN
            A := 0.5 ; B := 0.1 ; C := 0.006
            END
       ELSE IF COMPTEUR = 2
            THEN BEGIN
                 A := 0.2 ; B := 0.05 ; C := 0.0013
                 END
            ELSE BEGIN
                 A := 0.05 ; B := 0.01 ; C := 0.0002
                 END ;
       IF DRAW (A,U) AND COMPTEUR < 4
       THEN BEGIN
            REF (BCA) Z ;
            REF (BCL) X,Y ;
            INTEGER NLL,I ;
            X :- LPL.Q.FIRST ;
            NLL := 0 ;
            WHILE X =/= NONE DO
            BEGIN
                 FOR I := 1 STEP 1 UNTIL X.LL DO
                     IF DRAW (B,U) THEN NLL := NLL + 1 ;
                 X :- X.SUC
            END ;
            HOLD (NLL*C) ;
            FOR X :- LPL.Q.FIRST WHILE X =/= NONE DO
            BEGIN
                 X.INTO (LPLM.Q) ;
                 LPLM.ENTRER (X)
            END ;
            BIDON.NO := NLL ;
                                                  PASSE(NOM,FALSE) ;
```

```
            A9.ENVOYER (BIDON)
                  END
            ELSE BEGIN
                  REF (BCA) Z ;
                  COMPTEUR := 0 ;
                  LCOJOPRIM.Q.CLEAR ;
                  LCOAT.Q.CLEAR ;
                  FOR Z :- NLCOAT.Q.FIRST WHILE Z =/= NONE DO
                  Z.INTO (LCOAT.Q) ;
                                                    PASSE(NOM,FALSE)
            A8.ENVOYER (BIDON)
                  END ;
            V.LIBERER
      END
END ; $ DFCLIV $


PROCESSUS CLASS SAISPLM (ORD,V) ; REF (CALRAMINF) ORD ; REF (CALRCPP) V ;
BEGIN
      WHILE TRUE DO
      BEGIN
            REF (MESSAGE) D ;
            A9.ACQUERIR (D) ;
                                                    PASSE (NOM,TRUE)
            NLCOAT.Q.CLEAR ;
            LPLM.Q.CLEAR ;
            V.ACQUERIR ;
            HOLD (D.NO * 0.004) ;
            V.LIBERER ;
            ORD.ACQUERIR ;
            HOLD (D.NO * 0.002) ;
            ORD.LIBERER ;
                                                    PASSE(NOM,FALSE)
            A11.ENVOYER (BIDON)
      END
END ; $ SAISPLM $


PROCESSUS CLASS EDBL (ORD) ; REF (CALRAMINF) ORD ;
BEGIN
      WHILE TRUE DO
      BEGIN
            REF (MESSAGE) D ;
            REF (BCL) X ;
            A8.ACQUERIR (D) ;
                                                    PASSE (NOM,TRUE)
            FOR X :- LPL.Q.FIRST WHILE X =/= NONE DO
                  BEGIN
                        LPL.SORTIR (X) ;
                        X.INTO (LBL.Q) ;
                         IF X.LA = 0 THEN ADM.SORTIR (X) ;
                        LBL.ENTRER (X)
                  END ;
            ORD.ACQUERIR ;
            HOLD (T) ;
            ORD.LIBERER ;
                                                    PASSE(NOM,FALSE)
            A7.ENVOYER (BIDON)
```



```
      END
) ; $EDBL $


CESSUS CLASS EXFACT (SFACT) ; REF (CALRAM) SFACT ;
GIN
      WHILE TRUE DO
      BEGIN
            REF (MESSAGE) D ;
            REF (BCL) X ;
            A14.ACQUERIR (D) ;
            SFACT.ACQUERIR (1) ;
                                                    PASSE (NOM,TRUE)
            FOR X :- LF.Q.FIRST WHILE X =/= NONE DO
            BEGIN
                  IF X.LA = 0
                  THEN BEGIN
                      OUTIMAGE ;
                      ECRI (TIME) ;
                      OUTTEXT ('    LA FACTURE NUMERO  ') ;
                     OUTINT (X.NO,4) ;
                      OUTTEXT('  EST ENTIEREMENT LIVREE **************');
                      IF X.DAT > 0
                      THEN BEGIN
                           OUTTEXT ('   TEMPS PASSE DANS LE SYSTEME : ');
                           ECRI (TIME - X.DAT)
                           END ;
                      OUTIMAGE ;
                      OUTIMAGE
                      END ;
                  HOLD (0.006) ;
                  LF.SORTIR (X) ;
                  X.INTO (LFCL.Q ) ;
                  LFCL.ENTRER (X) ;
                  FACU.SORTIR (X) ;
                  X.INTO (LFCPTA.Q) ;
                  LFCPTA.ENTRER (X) ;
            END ;
                                                    PASSE(NOM,FALSE)
            A16.ENVOYER (D) ;
            A17.ENVOYER (D) ;
            SFACT.LIBERER (1)
      END
) ; $ EXFACT $


CESSUS CLASS FACT (ORD) ; REF (CALRAMINF) ORD ;
GIN
      WHILE TRUE DO
      BEGIN
            REF (MESSAGE) D ;
            REF (BCL) X ;
            A15.ACQUERIR (D) ;
                                                    PASSE (NOM,TRUE) ;
            FOR X :- LBLF.Q.FIRST WHILE X =/= NONE DO
                  BEGIN
                        LBLF.SORTIR (X) ;
                        X.INTO (LF.Q) ;
```

```
                    LF.ENTRER (X)
               END ;
          ORD.ACQUERIR ;
          HOLD (T) ;
          ORD.LIBERER ;
                                                    PASSE(NOM,FALSE
          A14.ENVOYER (BIDON)
     END
END ; $ FACT $


PROCESSUS CLASS SAISBL (ORD,VAL) ; REF (CALRAMINF) ORD ; REF (CALRC) VA
BEGIN
     WHILE TRUE DO
     BEGIN
          REF (MESSAGE) D ;
          REF (BCL) X ;
          A12.ACQUERIR (D) ;
                                                    PASSE (NOM,TRUE
          FOR X :- LBLP.Q.FIRST WHILE X =/= NONE DO
          BEGIN
               VAL.ACQUERIR ;
               HOLD (X.LL * 0.004 + 0.006) ;
               VAL.LIBERER ;
               ORD.ACQUERIR ;
               HOLD (X.LL * 0.002 + 0.002) ;
               ORD.LIBERER ;
               LBLP.SORTIR (X) ;
               X.INTO (LBLF.Q) ;
               LBLF.ENTRER (X)
          END ;
                                                    PASSE(NOM,FALSE
     END
END ; $ SAISBL $


PROCESSUS CLASS DECFACT (VAL) ; REF (CALRC) VAL ;
BEGIN
     WHILE TRUE DO
     BEGIN
          REF (MESSAGE) D ;
          REAL A,B ;
          REF (BCL) X ;
          INTEGER N,M ;
          A13.ACQUERIR (D) ;
          VAL.ACQUERIR ;
                                                    PASSE (NOM,TRUE
          A := 0.4 ;
          N := LBLP1.Q.CARDINAL ;
          M := LBLNF.Q.CARDINAL ;
          B := N * ( 1 - A ) / M ;
          FOR X :- LBLP1.Q.FIRST WHILE X =/= NONE DO
          BEGIN
               FACU.ENTRER (X) ;
               LBLP1.SORTIR (X) ;
               IF DRAW (A,U)
               THEN BEGIN
                    X.INTO (LBLP.Q) ;
```



```
                    LBLP.ENTRER (X)
                    END
               ELSE BEGIN
                    X.INTO (LISTEMP.Q) ;
                    LBLNF.ENTRER (X)
                    END ;
               HOLD ( X.LL * 0.0013 + 0.0026 )
          END ;
          FOR X :- LBLNF.Q.FIRST WHILE X =/= NONE DO
          BEGIN
               IF DRAW (B,U)
               THEN BEGIN
                    LBLNF.SORTIR (X) ;
                    X.INTO (LBLP.Q) ;
                    LBLP.ENTRER (X)
                    END
               ELSE X.INTO (LISTEMP.Q) ;
               HOLD (X.LL * 0.0013 + 0.0026 )
          END ;
                                                    PASSE(NOM,FALSE)
          FOR X :- LISTEMP.Q.FIRST WHILE X =/= NONE DO
                    X.INTO (LBLNF.Q) ;
          VAL.LIBERER ;
          A12.ENVOYER (BIDON)
     END
D ; $ DECFACT $


OCESSUS CLASS PREPLIV (PREPARATEUR ) ; REF (CALRC) PREPARATEUR ;
GIN
     WHILE TRUE DO
     BEGIN
          REF (MESSAGE) D ;
          REF (BCL) X ;
          A7.ACQUERIR (D) ;
          PREPARATEUR.ACQUERIR ;
                                                    PASSE (NOM,TRUE)
          FOR X :- LBL.Q.FIRST WHILE X =/= NONE DO
          BEGIN
               PHY.ENTRER (X) ;
               HOLD (X.LL * 0.032) ;
               LBL.SORTIR (X) ;
               X.INTO (LBLP1.Q) ;
               LBLP1.ENTRER (X) ;
               PHY.SORTIR (X) ;
               X.INTO (LBLP2.Q) ;
               LBLP2.ENTRER (X) ;
          END ;
                                                    PASSE(NOM,FALSE)
          PREPARATEUR.LIBERER
     END
D ; $ PREPLIV $


OCESSUS ECLIENT ;
GIN
     WHILE TRUE DO
     BEGIN
```

```
            REF (BCL) X ;
            REF (MESSAGE) D ;
            A16.ACQUERIR (D) ;
                                                    PASSE (NOM,TRU
            FOR X :- LBLP2.Q.FIRST WHILE X =/= NONE DO
            BEGIN
                  LBLP2.SORTIR (X) ;
                  X.OUT
            END ;
            FOR X :- LFCL.Q.FIRST WHILE X =/= NONE DO
            BEGIN
                  LFCL.SORTIR (X) ;
                  X.OUT
            END ;
                                                    PASSE (NOM,FAL
      END
END ; $ ECLIENT $

*

PROCESSUS ECOMPTA ;
BEGIN
      WHILE TRUE DO
      BEGIN
            REF (BCL) X ;
            REF (MESSAGE) D ;
            A17.ACQUERIR (D) ;
                                          PASSE (NOM,TRUE) ;
            FOR X :- LFCPTA.Q.FIRST WHILE X =/= NONE DO
            BEGIN
                  LFCPTA.SORTIR (X) ;
                  X.OUT
            END ;
                                          PASSE (NOM,FALSE) ;
      END
END ; $ ECOMPTA $



**********
**********  DECLARATION DES VARIABLES DE TRAVAIL
**********
      INTEGER I ;
      INTEGER ORDRE ;
      INTEGER COMPTEUR ;
      INTEGER LIMLBLNF , LIMLCOAT ;
**********
**********  DECLARATION DES FILES D'ATTENTE
**********
      REF (TFILE) LBC,LBCI,LCOJO,LPL,LPLM,LBL,LCOAT,NLCOAT,LBLP1,
      LBLP2,LBLNF,LBLP,LISTEMP,LBLF,LF,LFCL,LFCPTA,LCOJOPRIM ;
**********
**********  DECLARATION DES REGIONS
**********
      REF (TREGION) ECR,TEL,ADM,PHY,FACU ;
**********
********** DECLARATION DES PROCESSEURS
**********
```



```
FF (CALRAMINF) ORD ;
FF (CALRC) R ;
FF (CALRCPP) V ;
FF (CALRC) PREPA ;
FF (CALRC) VAL;
FF (CALRAM) SFACT ;
***
***  DECLARATION DES MONITEURS D'ACTIVATION
***
EF (ACTMONITOR) A1,A3,A4,A5,A7,A8,A9,A10,A11,A12,A13,A14,A15 ,
16,A17 ;


***
***  INITIALISATION DES VARIABLES DE TRAVAIL
***
IMLBLNF := 50 ;
IMLCOAT := 50 ;
OMPTEUR := 0 ;
RDRE := 0 ;



***
*** CREATION DES REGIONS
***
CR :- NEW TREGION ('ECRITES') ;
EL :- NEW TREGION ('TELEPHONEES') ;
DM :- NEW TREGION ('ADMINISTRATIF') ;
HY :- NEW TREGION ('PHYSIQUE') ;
ACU :- NEW TREGION ('FACTURATION') ;
***
***  CREATION DES FILES
***
BC :- NEW TFILE ('LBC',TRUE,TRUE,24) ;
BCI :- NEW TFILE ('LBCI',TRUE,FALSE,24) ;
COJO :- NEW TFILE ('LCOJO',TRUE,TRUE,24) ;
COJOPRIM :- NEW TFILE ('LCOJOPRIM',FALSE,FALSE,24) ;
PL :- NEW TFILE ('LPL',TRUE,FALSE,24) ;
PLM :- NEW TFILE ('LPLM',FALSE,FALSE,24) ;
BL :- NEW TFILE ('LBL',TRUE,FALSE,24) ;
COAT :- NEW TFILE ('LCOAT',FALSE,TRUE,24) ;
LCOAT :- NEW TFILE ('NLCOAT',FALSE,FALSE,24) ;
BLP1 :- NEW TFILE ('LBLP1',TRUE,TRUE,24) ;
BLP2 :- NEW TFILE ('LBLP2',TRUE,FALSE,24) ;
BLNF :- NEW TFILE ('LBLNF',TRUE,FALSE,24) ;
BLP :- NEW TFILE ('LBLP',TRUE,FALSE,24) ;
ISTEMP :- NEW TFILE ('LISTEMP',FALSE,FALSE,24) ;
BLF :- NEW TFILE ('LBLF',TRUE,FALSE,24) ;
F :- NEW TFILE ('LF',TRUE,FALSE,24) ;
FCL :- NEW TFILE ('LFCL',TRUE,TRUE,24) ;
FCPTA :- NEW TFILE ('LFCPTA',TRUE,FALSE,24) ;
***
***  CREATION DES RESSOURCES
***
RD :- NEW CALRAMINF ('ORDINATEUR',FALSE,TRUE) ;
 :- NEW CALRC ('R',TRUE,TRUE) ;
 :- NEW CALRCPP ('V',TRUE,FALSE) ;
AL :- NEW CALRC ('VAL',TRUE,TRUE) ;
FACT :- NEW CALRAM ('SFACT',TRUE,TRUE,2) ;
```

```
      PREPA :- NEW CALRC ('PREPA',TRUE,FALSE) ;
**********
**********  CREATION DES MONITEURS D'ACTIVATION
**********
      A1 :- NEW ACTMONITOR ('GENERATEUR',1) ;
      A3 :- NEW ACTMONITOR ('SAISBC',1) ;
      A4 :- NEW ACTMONITOR ('CODBC',1) ;
      A5 :- NEW ACTMONITOR ('SAISAPL',1) ;
      A7 :- NEW ACTMONITOR ('PREPLIV',1) ;
      A8 :- NEW ACTMONITOR ('EDBL',1) ;
      A9 :- NEW ACTMONITOR ('SAISPLM',1) ;
      A10 :- NEW ACTMONITOR ('DECLIV',1) ;
      A11 :- NEW ACTMONITOR ('PROPLIV',3) ;
      A12 :- NEW ACTMONITOR ('SAISBL',1) ;
      A13 :- NEW ACTMONITOR ('DECFACT',1) ;
      A14 :- NEW ACTMONITOR ('EXFACT',1) ;
      A15 :- NEW ACTMONITOR ('FACT',1) ;
      A16 :- NEW ACTMONITOR ('ECLIENT',2) ;
      A17 :- NEW ACTMONITOR ('ECOMPTA',2) ;
**********
********** CREATION DE L'ECHEANCIER
**********
      ECHE.MODIFECH (A1) ;
      ECHE.MODIFECH (A4) ;
      ECHE.MODIFECH (A7) ;
      ECHE.MODIFECH (A11) ;
      ECHE.MODIFECH (A13) ;
      ECHE.MODIFECH (A15) ;
**********
********** INITIALISATION DES FILES-FICHIERS
**********
      FOR I := 1 STEP 1 UNTIL LIMLCOAT DO
      BEGIN
            REF (BCA) X ;
            X :- NEW BCA (NUMEROTATION,4,TIME,"A",4,2) ;
            X.INTO (LCOAT.Q) ;
            ADM.ENTRER (X) ;
            LCOAT.ENTRER(X)
      END ;
      FOR I := 1 STEP 1 UNTIL LIMLBLNF DO
      BEGIN
            REF (BCL) X ;
            X :- NEW BCL (NUMEROTATION,2,TIME,"A",4,2,2) ;
            X.INTO (LBLNF.Q) ;
            FACU.ENTRER (X) ;
            LBLNF.ENTRER (X)
      END ;
**********
**********  ACTIVATION DES PROCESSUS
**********
      ACTIVATE NEW GENERATEUR ('GENERATEUR',1,0) QUA PROCESS ;
      ACTIVATE NEW SAISBC ('SAISBC',1,0,R,ORD) QUA PROCESS ;
      ACTIVATE NEW CODBC ('CODBC',1,0,R) QUA PROCESS ;
      ACTIVATE NEW SAISAPL ('SAISAPL',2,0,V,ORD) QUA PROCESS ;
      ACTIVATE NEW GENETEL ('GENETEL',1,0,V) QUA PROCESS ;
      ACTIVATE NEW PREPLIV ('PREPLIV',1,0,PREPA) QUA PROCESS ;
      ACTIVATE NEW EDBL ('EDBL',1,0.25,ORD) QUA PROCESS ;
      ACTIVATE NEW PROPLIV ('PROPLIV',1,0.0015,ORD) QUA PROCESS ;
      ACTIVATE NEW DECLIV ('DECLIV',1,0,V) QUA PROCESS ;
      ACTIVATE NEW SAISPLM ('SAISPLM',1,0,ORD,V) QUA PROCESS ;
```



```
ACTIVATE NEW SAISBL ('SAISBL',1,0,ORD,VAL) QUA PROCESS ;
ACTIVATE NEW DECFACT ('DECFACT',1,0,VAL) QUA PROCESS ;
ACTIVATE NEW EXFACT ('EXFACT',1,0,SFACT) QUA PROCESS ;
ACTIVATE NEW FACT ('FACT',1,0.5,ORD ) QUA PROCESS ;
ACTIVATE NEW ECLIENT ('ECLIENT',1,0) QUA PROCESS ;
ACTIVATE NEW ECOMPTA ('ECOMPTA',1,0) QUA PROCESS ;



****
****  LANCEMENT DE LA SIMULATION
****
SIMUL (72,TRUE,TRUE,TRUE)
0 $ BLOC MAESTRO $
0  $  PROGRAMME  $ #



****
****  DONNEES DE LA SIMULATION
****  LES CALENDRIERS
****
0 0 12 0 0 14 0 0 15 0*
 0 12 0 0 14 0 0 18 0*
0 0 16 30*
 0 12 0 0 14 0 0 18 0*
 0 12 0 0 14 0 0 18 0*
****
****  L'ECHEANCIER
****
0*
0*
*
0*
0*
0*
```

ANNEXE 6.

RESULTATS : TRACE DES PROCESSUS

Nous présentons ici des résultats obtenus par simulation du système décrit dans le chapitre V.

Ces résultats concernent la trace des différents processus obtenue par l'utilisation de la procédure passe. Pour chaque processus, on trouve ses dates (sous la forme jour, heure, minute) de début et de fin d'activité. Les calendriers relatifs à chacun des processeurs utilisant ce mécanisme figurent au début. Un léger bilan de fin de journée précise quelles sont les transactions (bons de livraisons) qui quittent le système (pour chacunes, autres que celles ayant participé à l'initialisation, on précise combien de temps elles sont restées dans le système).

```
LE CALENDRIER ASSOCIE A LA RESSOURCE   R  EST LE SUIVANT :
 0 10  0   0 12  0   0 14  0   0 15  0
LE CALENDRIER ASSOCIE A LA RESSOURCE   V  EST LE SUIVANT :
 0  9  0   0 12  0   0 14  0   0 18  0
LE CALENDRIER ASSOCIE A LA RESSOURCE   VAL  EST LE SUIVANT :
 0 14  0   0 16 30
LE CALENDRIER ASSOCIE A LA RESSOURCE   SFACT  EST LE SUIVANT :
 0  8  0   0 12  0   0 14  0   0 18  0
LE CALENDRIER ASSOCIE A LA RESSOURCE   PREPA  EST LE SUIVANT :
 0  9  0   0 12  0   0 14  0   0 18  0
```

```
  0  9  0            FIN DE TRAVAIL DE   PREPLIV
********************************  0  9 10     APPEL TELEPHONIQUE    0  9 10 T  101  4
  0  9 10      DEBUT DE TRAVAIL DE   SAISAPL
  0  9 14            FIN DE TRAVAIL DE   SAISAPL
********************************  0  9 23     APPEL TELEPHONIQUE    0  9 23 D  102  4
  0  9 23      DEBUT DE TRAVAIL DE   SAISAPL
  0  9 25            FIN DE TRAVAIL DE   SAISAPL
  0  9 30      DEBUT DE TRAVAIL DE   GENERATEUR
  0  9 30            FIN DE TRAVAIL DE   GENERATEUR
  0 10  0      DEBUT DE TRAVAIL DE   CODBC
********************************  0 10  5     APPEL TELEPHONIQUE    0 10  5 D  118  4
  0 10  5      DEBUT DE TRAVAIL DE   SAISAPL
  0 10  7            FIN DE TRAVAIL DE   SAISAPL
********************************  0 10 23     APPEL TELEPHONIQUE    0 10 23 D  119  4
  0 10 23      DEBUT DE TRAVAIL DE   SAISAPL
  0 10 26            FIN DE TRAVAIL DE   SAISAPL
********************************  0 10 41     APPEL TELEPHONIQUE    0 10 41 D  120  4
  0 10 41      DEBUT DE TRAVAIL DE   SAISAPL
  0 10 43            FIN DE TRAVAIL DE   SAISAPL
  0 11  5            FIN DE TRAVAIL DE   CODBC
  0 11  5      DEBUT DE TRAVAIL DE   SAISBC
********************************  0 11 13     APPEL TELEPHONIQUE    0 11 13 D  121  4
  0 11 13      DEBUT DE TRAVAIL DE   SAISAPL
  0 11 15            FIN DE TRAVAIL DE   SAISAPL
********************************  0 11 56     APPEL TELEPHONIQUE    0 11 56 D  122  4
  0 11 56      DEBUT DE TRAVAIL DE   SAISAPL
********************************  0 11 56     APPEL TELEPHONIQUE    0 11 56 T  123  4
  0 11 56   SIGNAL SUPPRIME ...SAISAPL
  0 11 58            FIN DE TRAVAIL DE   SAISAPL
  0 14  0      DEBUT DE TRAVAIL DE   DECFACT
  0 14 15            FIN DE TRAVAIL DE   DECFACT
  0 14 15      DEBUT DE TRAVAIL DE   SAISBL
  0 14 15            FIN DE TRAVAIL DE   SAISBL
  0 14 21            FIN DE TRAVAIL DE   SAISBC
********************************  0 14 53     APPEL TELEPHONIQUE    0 14 53 T  124  4
  0 14 53      DEBUT DE TRAVAIL DE   SAISAPL
  0 14 57            FIN DE TRAVAIL DE   SAISAPL
  0 15  0      DEBUT DE TRAVAIL DE   PROPLIV
  0 15 13            FIN DE TRAVAIL DE   PROPLIV
  0 15 13      DEBUT DE TRAVAIL DE   DECLIV
  0 15 15            FIN DE TRAVAIL DE   DECLIV
  0 15 15      DEBUT DE TRAVAIL DE   SAISPLM
  0 15 17            FIN DE TRAVAIL DE   SAISPLM
  0 15 17      DEBUT DE TRAVAIL DE   PROPLIV
********************************  0 15 25     APPEL TELEPHONIQUE    0 15 25 T  125  4
  0 15 25      DEBUT DE TRAVAIL DE   SAISAPL
  0 15 49            FIN DE TRAVAIL DE   EDBL
  0 15 49      DEBUT DE TRAVAIL DE   PREPLIV
********************************  0 15 50     APPEL TELEPHONIQUE    0 15 50 D  128  4
  0 15 50   SIGNAL SUPPRIME ...SAISAPL
  0 15 50            FIN DE TRAVAIL DE   SAISAPL
********************************  0 15 53     APPEL TELEPHONIQUE    0 15 53 T  129  4
  0 15 53   SIGNAL SUPPRIME ...SAISAPL
********************************  0 16  9     APPEL TELEPHONIQUE    0 16  9 D  130  4
  0 16  9      DEBUT DE TRAVAIL DE   SAISAPL
  0 16 11            FIN DE TRAVAIL DE   SAISAPL
********************************  0 16 34     APPEL TELEPHONIQUE    0 16 34 T  131  4
  0 16 34      DEBUT DE TRAVAIL DE   SAISAPL
  0 16 39            FIN DE TRAVAIL DE   SAISAPL
********************************  0 16 45     APPEL TELEPHONIQUE    0 16 45 D  132  4
  0 16 45      DEBUT DE TRAVAIL DE   SAISAPL
  0 16 47            FIN DE TRAVAIL DE   SAISAPL
********************************  0 16 56     APPEL TELEPHONIQUE    0 16 56 D  133  4
  0 16 56      DEBUT DE TRAVAIL DE   SAISAPL
  0 16 58            FIN DE TRAVAIL DE   SAISAPL
  0 17  0      DEBUT DE TRAVAIL DE   FACT
********************************  0 17  2     APPEL TELEPHONIQUE    0 17  2 D  134  4
  0 17  2      DEBUT DE TRAVAIL DE   SAISAPL
  0 17  5            FIN DE TRAVAIL DE   SAISAPL
  0 17 30            FIN DE TRAVAIL DE   FACT
  0 17 30      DEBUT DE TRAVAIL DE   EXFACT
  0 17 30            FIN DE TRAVAIL DE   EXFACT
********************************  0 17 53     APPEL TELEPHONIQUE    0 17 53 T  135  4
  0 17 53      DEBUT DE TRAVAIL DE   SAISAPL
  0 17 58            FIN DE TRAVAIL DE   SAISAPL
  1  9  0   SIGNAL SUPPRIME ...PREPLIV
  1  9 30      DEBUT DE TRAVAIL DE   GENERATEUR
  1  9 30            FIN DE TRAVAIL DE   GENERATEUR
********************************  1  9 45     APPEL TELEPHONIQUE    1  9 45 D  151  4
  1  9 45      DEBUT DE TRAVAIL DE   SAISAPL
  1  9 47            FIN DE TRAVAIL DE   SAISAPL
  1 10  0      DEBUT DE TRAVAIL DE   CODBC
********************************  1 10  1     APPEL TELEPHONIQUE    1 10  1 D  152  4
  1 10  1      DEBUT DE TRAVAIL DE   SAISAPL
  1 10  3            FIN DE TRAVAIL DE   SAISAPL
********************************  1 10 12     APPEL TELEPHONIQUE    1 10 12 D  153  4
  1 10 12      DEBUT DE TRAVAIL DE   SAISAPL
********************************  1 10 13     APPEL TELEPHONIQUE    1 10 13 T  154  4
  1 10 13   SIGNAL SUPPRIME ...SAISAPL
```

```
  1 10 36      DEBUT DE TRAVAIL DE   SAISAPL
  1 10 38            FIN DE TRAVAIL DE   SAISAPL
********************************  1 10 40      APPEL TELEPHONIQUE     1 10 40 D  156  4
  1 10 40   SIGNAL SUPPRIME ...SAISAPL
  1 11  5            FIN DE TRAVAIL DE   CODRC
  1 11  5      DEBUT DE TRAVAIL DE   SAISRC
********************************  1 11 16      APPEL TELEPHONIQUE     1 11 16 T  157  4
  1 11 16      DEBUT DE TRAVAIL DE   SAISAPL
  1 11 21            FIN DE TRAVAIL DE   SAISAPL
********************************  1 11 40      APPEL TELEPHONIQUE     1 11 40 D  158  4
  1 11 40      DEBUT DE TRAVAIL DE   SAISAPL
  1 11 42            FIN DE TRAVAIL DE   SAISAPL
  1 11 57            FIN DE TRAVAIL DE   PREPLIV
********************************  1 11 59      APPEL TELEPHONIQUE     1 11 59 D  159  4
  1 11 59      DEBUT DE TRAVAIL DE   SAISAPL
  1 12  1            FIN DE TRAVAIL DE   SAISAPL
  1 14  0      DEBUT DE TRAVAIL DE   DECFACT
  1 14 21            FIN DE TRAVAIL DE   SAISRC
********************************  1 14 27      APPEL TELEPHONIQUE     1 14 27 D  160  4
  1 14 27      DEBUT DE TRAVAIL DE   SAISAPL
  1 14 29            FIN DE TRAVAIL DE   SAISAPL
  1 14 38            FIN DE TRAVAIL DE   DECFACT
  1 14 38      DEBUT DE TRAVAIL DE   SAISBL
********************************  1 14 53      APPEL TELEPHONIQUE     1 14 53 D  161  4
  1 14 53      DEBUT DE TRAVAIL DE   SAISAPL
  1 14 55            FIN DE TRAVAIL DE   SAISAPL
  1 15  0      DEBUT DE TRAVAIL DE   PROPLIV
********************************  1 15 12      APPEL TELEPHONIQUE     1 15 12 T  162  4
  1 15 12      DEBUT DE TRAVAIL DE   SAISAPL
  1 15 14            FIN DE TRAVAIL DE   PROPLIV
  1 15 14      DEBUT DE TRAVAIL DE   DECLIV
  1 15 14            FIN DE TRAVAIL DE   DECLIV
  1 15 14      DEBUT DE TRAVAIL DE   EDBL
  1 15 17            FIN DE TRAVAIL DE   SAISAPL
********************************  1 15 24      APPEL TELEPHONIQUE     1 15 24 D  163  4
  1 15 24      DEBUT DE TRAVAIL DE   SAISAPL
  1 15 27            FIN DE TRAVAIL DE   SAISAPL
  1 15 29            FIN DE TRAVAIL DE   EDBL
  1 15 29      DEBUT DE TRAVAIL DE   PREPLIV
  1 15 59            FIN DE TRAVAIL DE   SAISBL
********************************  1 16 14      APPEL TELEPHONIQUE     1 16 14 D  164  4
  1 16 14      DEBUT DE TRAVAIL DE   SAISAPL
  1 16 16            FIN DE TRAVAIL DE   SAISAPL
  1 17  0      DEBUT DE TRAVAIL DE   FACT
********************************  1 17  3      APPEL TELEPHONIQUE     1 17  3 T  165  4
  1 17  3      DEBUT DE TRAVAIL DE   SAISAPL
  1 17 30      LA FACTURE NUMERO     2  EST ENTIEREMENT LIVREE **************

  1 17 30      LA FACTURE NUMERO     4  EST ENTIEREMENT LIVREE **************

  1 17 30      LA FACTURE NUMERO     8  EST ENTIEREMENT LIVREE **************

  1 17 31      LA FACTURE NUMERO    10  EST ENTIEREMENT LIVREE **************

  1 17 31      LA FACTURE NUMERO    11  EST ENTIEREMENT LIVREE **************

  1 17 32      LA FACTURE NUMERO    13  EST ENTIEREMENT LIVREE **************

  1 17 32            FIN DE TRAVAIL DE   SAISAPL

  1 17 32      LA FACTURE NUMERO    16  EST ENTIEREMENT LIVREE **************

********************************  1 17 32      APPEL TELEPHONIQUE     1 17 32 T  168  4
  1 17 32   SIGNAL SUPPRIME ...SAISAPL
  1 17 32      LA FACTURE NUMERO    20  EST ENTIEREMENT LIVREE **************

  1 17 33      LA FACTURE NUMERO    21  EST ENTIEREMENT LIVREE **************

  1 17 33      LA FACTURE NUMERO    22  EST ENTIEREMENT LIVREE **************

  1 17 33      LA FACTURE NUMERO    24  EST ENTIEREMENT LIVREE **************

  1 17 34      LA FACTURE NUMERO    27  EST ENTIEREMENT LIVREE **************

  1 17 34      LA FACTURE NUMERO    31  EST ENTIEREMENT LIVREE **************

  1 17 35      LA FACTURE NUMERO    36  EST ENTIEREMENT LIVREE **************
```

```
  1 17 36      LA FACTURE NUMERO    40  EST ENTIEREMENT LIVREE **************

  1 17 36      LA FACTURE NUMERO    44  EST ENTIEREMENT LIVREE **************

  1 17 36      LA FACTURE NUMERO    45  EST ENTIEREMENT LIVREE **************

  1 17 37      LA FACTURE NUMERO    48  EST ENTIEREMENT LIVREE **************

********************************  1 17 40      APPEL TELEPHONIQUE     1 17 40 D  169  4
  1 17 40      DEBUT DE TRAVAIL DE    SAISAPL
  1 17 42             FIN DE TRAVAIL DE    SAISAPL
  1 17 53             FIN DE TRAVAIL DE    EXFACT
********************************  1 17 56      APPEL TELEPHONIQUE     1 17 56 D  170  4
  1 17 56      DEBUT DE TRAVAIL DE    SAISAPL
  1 17 58             FIN DE TRAVAIL DE    SAISAPL
  2  9  0   SIGNAL SUPPRIME ...PREPLIV
********************************  2  9 11      APPEL TELEPHONIQUE     2  9 11 T  171  4
  2  9 11      DEBUT DE TRAVAIL DE    SAISAPL
  2  9 15             FIN DE TRAVAIL DE    SAISAPL
********************************  2  9 26      APPEL TELEPHONIQUE     2  9 26 D  172  4
  2  9 26      DEBUT DE TRAVAIL DE    SAISAPL
  2  9 28             FIN DE TRAVAIL DE    SAISAPL
  2  9 30      DEBUT DE TRAVAIL DE    GENERATEUR
  2  9 30             FIN DE TRAVAIL DE    GENERATEUR
********************************  2  9 56      APPEL TELEPHONIQUE     2  9 56 T  188  4
  2  9 56      DEBUT DE TRAVAIL DE    SAISAPL
  2 10  0      DEBUT DE TRAVAIL DE    CODBC
  2 10  0             FIN DE TRAVAIL DE    SAISAPL
********************************  2 10 18      APPEL TELEPHONIQUE     2 10 18 D  189  4
  2 10 18      DEBUT DE TRAVAIL DE    SAISAPL
  2 10 21             FIN DE TRAVAIL DE    SAISAPL
********************************  2 10 42      APPEL TELEPHONIQUE     2 10 42 D  190  4
  2 10 42      DEBUT DE TRAVAIL DE    SAISAPL
  2 10 44             FIN DE TRAVAIL DE    SAISAPL
********************************  2 10 46      APPEL TELEPHONIQUE     2 10 46 D  191  4
  2 10 46   SIGNAL SUPPRIME ...SAISAPL
  2 11  5             FIN DE TRAVAIL DE    CODBC
  2 11  5      DEBUT DE TRAVAIL DE    SAISBC
********************************  2 11 10      APPEL TELEPHONIQUE     2 11 10 D  192  4
  2 11 10      DEBUT DE TRAVAIL DE    SAISAPL
  2 11 12             FIN DE TRAVAIL DE    SAISAPL
********************************  2 11 37      APPEL TELEPHONIQUE     2 11 37 T  193  4
  2 15  0      DEBUT DE TRAVAIL DE    PROPLIV
  2 15 14             FIN DE TRAVAIL DE    PROPLIV
  2 15 14      DEBUT DE TRAVAIL DE    DECLIV
  2 15 17             FIN DE TRAVAIL DE    DECLIV
  2 15 17      DEBUT DE TRAVAIL DE    SAISPLM
  2 15 20             FIN DE TRAVAIL DE    SAISPLM
  2 15 20      DEBUT DE TRAVAIL DE    PROPLIV
  2 15 34             FIN DE TRAVAIL DE    PROPLIV
  2 15 34      DEBUT DE TRAVAIL DE    DECLIV
  2 15 35             FIN DE TRAVAIL DE    DECLIV
  2 15 35      DEBUT DE TRAVAIL DE    SAISPLM
********************************  2 15 38      APPEL TELEPHONIQUE     2 15 38 T  194  4
  2 15 38      DEBUT DE TRAVAIL DE    SAISAPL
  2 15 40             FIN DE TRAVAIL DE    SAISPLM
  2 15 40      DEBUT DE TRAVAIL DE    PROPLIV
  2 15 42             FIN DE TRAVAIL DE    SAISAPL
********************************  2 15 44      APPEL TELEPHONIQUE     2 15 44 D  195  4
  2 15 44   SIGNAL SUPPRIME ...SAISAPL
  2 15 44             FIN DE TRAVAIL DE    SAISBL
********************************  2 15 45      APPEL TELEPHONIQUE     2 15 45 T  196  4
  2 15 45   SIGNAL SUPPRIME ...SAISAPL
********************************  2 15 52      APPEL TELEPHONIQUE     2 15 52 T  197  4
  2 15 52      DEBUT DE TRAVAIL DE    SAISAPL
  2 15 53             FIN DE TRAVAIL DE    PROPLIV
  2 15 53      DEBUT DE TRAVAIL DE    DECLIV
  2 15 53             FIN DE TRAVAIL DE    DECLIV
  2 15 53      DEBUT DE TRAVAIL DE    SAISPLM
  2 15 54             FIN DE TRAVAIL DE    SAISPLM
  2 15 54      DEBUT DE TRAVAIL DE    PROPLIV
  2 15 57             FIN DE TRAVAIL DE    SAISAPL
  2 16  8             FIN DE TRAVAIL DE    PROPLIV
  2 16  8      DEBUT DE TRAVAIL DE    DECLIV
  2 16  8             FIN DE TRAVAIL DE    DECLIV
  2 16  8      DEBUT DE TRAVAIL DE    EDBL
********************************  2 16 19      APPEL TELEPHONIQUE     2 16 19 T  198  4
  2 16 19      DEBUT DE TRAVAIL DE    SAISAPL
  2 16 23             FIN DE TRAVAIL DE    EDBL
  2 16 23      DEBUT DE TRAVAIL DE    PREPLIV
  2 16 23             FIN DE TRAVAIL DE    SAISAPL
********************************  2 16 34      APPEL TELEPHONIQUE     2 16 34 T  199  4
  2 16 34      DEBUT DE TRAVAIL DE    SAISAPL
  2 16 38             FIN DE TRAVAIL DE    SAISAPL
********************************  2 16 39      APPEL TELEPHONIQUE     2 16 39 D  200  4
  2 16 39   SIGNAL SUPPRIME ...SAISAPL
```

```
 2 16 50      DEBUT DE TRAVAIL DE   SAISAPL
 2 16 52            FIN DE TRAVAIL DE   SAISAPL
 2 17  0      DEBUT DE TRAVAIL DE   FACT
********************************  2 17  7     APPEL TELEPHONIQUE    2 17  7 D  202  4
 2 17  7      DEBUT DE TRAVAIL DE   SAISAPL
 2 17  9            FIN DE TRAVAIL DE   SAISAPL
********************************  2 17 19     APPEL TELEPHONIQUE    2 17 19 D  203  4
 2 17 19      DEBUT DE TRAVAIL DE   SAISAPL
 2 17 21            FIN DE TRAVAIL DE   SAISAPL
 2 17 30            FIN DE TRAVAIL DE   FACT
 2 17 30      DEBUT DE TRAVAIL DE   EXFACT

 2 17 30      LA FACTURE NUMERO     5  EST ENTIEREMENT LIVREE **************

 2 17 30      LA FACTURE NUMERO    46  EST ENTIEREMENT LIVREE **************

 2 17 30      LA FACTURE NUMERO   103  EST ENTIEREMENT LIVREE **************   TEMPS PASSE DANS LE SYSTEME :   2

 2 17 31      LA FACTURE NUMERO   107  EST ENTIEREMENT LIVREE **************   TEMPS PASSE DANS LE SYSTEME :   2

 2 17 31      LA FACTURE NUMERO   108  EST ENTIEREMENT LIVREE **************   TEMPS PASSE DANS LE SYSTEME :   2

********************************  2 17 31     APPEL TELEPHONIQUE    2 17 31 D  204  4
 2 17 31      DEBUT DE TRAVAIL DE   SAISAPL
 2 17 31      LA FACTURE NUMERO   110  EST ENTIEREMENT LIVREE **************   TEMPS PASSE DANS LE SYSTEME :   2

 2 17 32      LA FACTURE NUMERO   112  EST ENTIEREMENT LIVREE **************   TEMPS PASSE DANS LE SYSTEME :   2

 2 17 32      LA FACTURE NUMERO   115  EST ENTIEREMENT LIVREE **************   TEMPS PASSE DANS LE SYSTEME :   2

 2 17 32      LA FACTURE NUMERO   116  EST ENTIEREMENT LIVREE **************   TEMPS PASSE DANS LE SYSTEME :   2
 2 17 33            FIN DE TRAVAIL DE   SAISAPL
 2 17 36      LA FACTURE NUMERO     6  EST ENTIEREMENT LIVREE **************
```

```
   2 17 37      LA FACTURE NUMERO    26  EST ENTIEREMENT LIVREE **************

   2 17 37      LA FACTURE NUMERO    28  EST ENTIEREMENT LIVREE **************

   2 17 38      LA FACTURE NUMERO    29  EST ENTIEREMENT LIVREE **************

   2 17 38      LA FACTURE NUMERO    32  EST ENTIEREMENT LIVREE **************

   2 17 38      LA FACTURE NUMERO    38  EST ENTIEREMENT LIVREE **************

   2 17 39      LA FACTURE NUMERO    39  EST ENTIEREMENT LIVREE **************

   2 17 39      LA FACTURE NUMERO    41  EST ENTIEREMENT LIVREE **************

   2 17 40      LA FACTURE NUMERO    43  EST ENTIEREMENT LIVREE **************

   2 17 40      LA FACTURE NUMERO    47  EST ENTIEREMENT LIVREE **************
   2 17 44            FIN DE TRAVAIL DE   EXFACT
********************************  2 17 51     APPEL TELEPHONIQUE    2 17 51 D  205  4
   2 17 51      DEBUT DE TRAVAIL DE   SAISAPL
   2 17 54            FIN DE TRAVAIL DE   SAISAPL
EXECUTION TERMINEE
******************************************************************************
```

NOM DE L'ETUDIANT : Monsieur KLEIN Philippe

NATURE DE LA THESE : Doctorat de 3e cycle : Informatique

VU, APPROUVE

et PERMIS D'IMPRIMER

NANCY LE 08 MAI 1979 4065

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

M. BOULANGE