UNIVERSITÉ DE NANCY I — U.E.R. DE SCIENCES MATHÉMATIQUES

CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY

MAQUETTES POUR EVALUER LES SYSTEMES D'INFORMATION DES ORGANISATIONS

THÈSE

présentée et soutenue publiquement le 22 Janvier 1980
devant la Commission d'Examen

A L'UNIVERSITÉ DE NANCY I

pour l'obtention du grade de

DOCTEUR ÈS-SCIENCES MATHEMATIQUES

par

Jean-François DUFOURD

Rapporteurs : MM. C. PAIR ..... Président
F. PECCOUD
C. ROLLAND

Examinateurs : MM. F. BODARD
M. CREHANGE
J.-C. DERNIAME
M. GRIFFITHS
D. POTIER

UNIVERSITÉ DE NANCY I — U.E.R. DE SCIENCES MATHÉMATIQUES

CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY

MAQUETTES POUR EVALUER LES SYSTEMES D'INFORMATION DES ORGANISATIONS

THÈSE

présentée et soutenue publiquement le 22 Janvier 1980
devant la Commission d'Examen

A L'UNIVERSITÉ DE NANCY I

pour l'obtention du grade de

DOCTEUR ÈS-SCIENCES MATHEMATIQUES

par

Jean-François DUFOURD

Rapporteurs : MM. C. PAIR ..... Président
F. PECCOUD
C. ROLLAND

Examinateurs : MM. F. BODARD
M. CREHANGE
J.-C. DERNIAME
M. GRIFFITHS
D. POTIER

à la mémoire de ma mère

à mon père

Je tiens à remercier ici les membres du jury :

C. Pair, Président de l'Institut National Polytechnique de Lorraine et Directeur du Centre de Recherche en Informatique de Nancy, pour l'honneur qu'il me fait en présidant ce jury ; il y a près de dix ans, il me suggérait de me lancer dans le domaine de l'informatique appliquée à la gestion : "tout y est à faire" disait-il ; j'espère que mon travail, qui a toujours bénéficié de son exceptionnelle compétence scientifique et de son soutien permanent, aura quelque peu contribué à combler le manque dont il parlait ;

F. Peccoud, Professeur à l'Université de Grenoble, précurseur français de l'informatique appliquée à la gestion ; je lui dois une base solide pour l'enseignement de l'analyse et mon goût pour les problèmes concrets ; ses travaux sur la conception des systèmes sont à l'origine de nôtre projet ;

C. Rolland, Professeur à l'Université de Paris -1- Sorbonne, un des pionniers de l'informatique d'organisation, qui dirige l'équipe à laquelle j'appartiens ; depuis des années, je bénéficie de son appui efficace et de ses excellents conseils : sans elle, ce travail n'aurait certainement pas abouti ;

F. Bodard, Professeur à l'Institut Universitaire N. D. de la Paix à Namur, spécialiste des systèmes d'information ; il s'est intéressé à notre approche qui complète sur certains points la recherche poursuivie actuellement par son équipe ;

M. Créhange, Professeur à l'Université de Nancy - 2, spécialiste des bases de données, qui honore l'établissement où j'enseigne ; ses travaux ne sont pas étrangers à la manière dont nous envisageons ici la structuration et la représentation de l'information ;

J.C. Derniame, Professeur à l'Université de Nancy - 1, un des initiateurs de l'informatique à Nancy ; je lui dois mes connaissances en informatique générale ; nos conversations amicales répétées et les contacts extérieurs qu'il a su me faire établir ont eu une influence capitale sur notre projet ;

M. Griffiths, Professeur à l'Université de Nancy - 2 et Directeur de l'Institut Universitaire de Calcul Automatique de Lorraine, dont la réputation internationale rejaillit bénéfiquement sur toute l'université nancéienne ; il nous a guidé dans le choix d'outils d'écriture de systèmes, au début de notre travail ;

D. Potier, Responsable scientifique à l'IRIA, spécialiste de l'évaluation des systèmes informatiques, qui a assisté aux débuts de notre projet ; il pourra constater que notre tentative d'appliquer certains des principes d'évaluation qu'il connaît bien aux systèmes d'information ne va pas sans difficulté.

J'associe dans l'expression de ma gratitude :

les personnes qui ont travaillé à la réalisation de notre projet :

N. Hertschuh, pour sa compétence en informatique, en gestion, et pour son amitié ;

P. Klein, qui a conçu et réalisé une grande part des outils-système ;

M. Thaurel, qui s'est intéressée à l'évaluation quantitative, qui a conçu et mis en oeuvre les outils de mesure et d'analyse statistique ;

les chercheurs du CRIN dont l'élan vers une théorie de l'informatique toujours plus achevée, allant de pair avec des développements pratiques importants, est à l'origine d'un environnement scientifique de valeur ;

les membres de l'IUCAL pour leur compétence technique et leur cordialité ;

les enseignants de l'IUT pour l'atmosphère amicale et favorable au travail qu'ils ont su entretenir ;

M.F. Lefèvre et M. Moreau dont j'ai mis la patience et la bonne humeur à rude épreuve lors de travaux de secrétariat ;

D. Marchand pour l'excellence de la réalisation finale de ce document.

Je sais gré le CNRS et l'IRIA de leur contribution déterminante à notre projet par le contrat 77165 de l'ATP n° 102 "Informatique d'Organisation".

Enfin, je remercie Ghislaine pour la solide philosophie qu'elle s'est forgée à propos des tâches universitaires, pour son secours moral et son soutien matériel.

J'embrasse Delphine dont le sourire et la fraîcheur nous illuminent depuis plus de quatre années.

Que tous les autres membres de ma famille soient remerciés pour leur compréhension, leur dévouement et leur appui constant.

PLAN de THESE

MAQUETTES pour EVALUER les SYSTEMES d'INFORMATION des ORGANISATIONS

INTRODUCTION

"Schikaneder :

Halt, wo wollen's den hin, Maestro ?"

(W.A. Mozart et E. Schikaneder,
Le Directeur de Théâtre)

"Schikaneder (à Mozart) :

Halte ! Où voulez-vous donc aller, Maestro ?"

En dépit du nombre considérable de systèmes d'information, parfois fort complexes, qui sont conçus, implantés ou exploités chaque jour dans les organisations, il existe fort peu de méthodes pour les appréhender globalement, pour les analyser ou les concevoir efficacement.

En ce domaine, la science est encore balbutiante, la théorie à peine émergeante, les connaissances éparses et empiriques. Il est vrai que c'est un sujet très vaste qui tient à la fois des sciences exactes, humaines, expérimentales, et qui est tributaire d'une évolution formidable de la technologie.

Il est dangereux de réduire le système d'information d'une organisation au seul sous-système automatique, de faire uniquement porter sur celui-ci les efforts d'étude, en négligeant ses interactions avec l'organisation tout entière. On l'a bien vu dans le passé avec l'implantation artificielle de systèmes parfois brillamment conçus, mais n'ayant qu'un lointain rapport avec les besoins en informations ou encore impossibles à alimenter en données, et, finalement, tournant à vide ou rejetés par les utilisateurs supposés, comme des corps étrangers.

L'objectif essentiel de ce travail est d'aider à accroître la connaissance que l'on a des systèmes d'information, à mieux comprendre leur structure, leur fonctionnement logique et leur comportement dynamique par une amélioration des méthodes et outils de description et d'analyse. Mais on peut entrevoir des retombées dans d'autres domaines, notamment celui de la conception des systèmes.

- La première partie de cette thèse présente, d'une manière un peu théorique, notre champ d'étude et la manière dont nous l'avons abordé.

Tout d'abord, il est nécessaire de dire ce que nous entendons par système d'information d'une organisation (chapitre 1). Pour nous, c'est un ensemble complexe d'informations et de moyens de traitement incluant : collecte, mémorisation, transformation, restitution, communication d'information. On peut s'intéresser à tout le

système ou seulement à une partie, mais aussi à certains éléments, extérieurs à l'organisation proprement dite, dont l'appréhension revêt de l'importance. On voit donc que l'on peut fixer très souplement les limites de ce que nous appelons système d'information, et qui comprend :

. 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.

Pour étudier une telle entité, il nous a semblé nécessaire d'en avoir une vision globale, et donc d'en faire des modèles généraux (chapitre 2). Les nôtres sont des modèles symboliques, numériques, dynamiques et discrets. Ils s'écrivent dans un langage de programmation, d'où leur nom de maquettes logicielles ou, plus simplement, maquettes.

Il est bien sûr possible de faire beaucoup de modèles du même système d'information, de différents niveaux, avec des caractéristiques diverses, un modèle ne valant que pour satisfaire des objectifs bien précisés à l'avance (chapitre 3) :

a) Une maquette peut être descriptive de l'architecture d'un système, pour mettre en évidence ses composants et ses relations.

b) Une maquette peut servir à tester le bon fonctionnement logique du système décrit, avec un souci d'ordre qualitatif.

c) Une maquette peut aider à apprécier le comportement dynamique d'un système, évaluer et prévoir ses performances, dans un souci d'ordre quantitatif.

- La deuxième partie présente les notions et les outils que l'on utilise pour concevoir et écrire des maquettes.

On pourrait penser avoir besoin de concepts très différents selon le type de modèles auquel on s'intéresse et même selon le niveau de détail. Nous montrons qu'il n'en est rien et que l'on a une unité des notions de base, avec une diversification des éléments secondaires qui en dérivent.

Il en est bien sûr de même du langage d'écriture de ces maquettes (chapitre 4). Il doit permettre :

. une vérification formelle des spécifications d'un modèle (catégories a, b et c)

. des tests de fonctionnement logique (catégories b et c)

. la résolution par simulation d'un modèle (catégorie c) pour tester quantitativement le comportement dynamique du système décrit, avec possibilité de mesures et d'analyses statistiques.

Ceci conduit naturellement à un langage de programmation apte à :

. la description de systèmes

. la simulation (ici, à événements discrets).

Nous avons finalement retenu le langage SIMULA 67 dont nous soulignons les avantages et les inconvénients compte-tenu de nos objectifs.

Par ailleurs, une maquette est un modèle dynamique de système. Il est donc nécessaire d'avoir des outils pour y exprimer le déroulement en parallèle et la synchronisation des activités de traitement d'information. A cette fin, nous avons largement utilisé un mécanisme standard, les moniteurs (de C.A.R. HOARE et P. BRINCH HANSEN), que nous avons été amenés à étendre pour tenir compte de certains cas d'espèces, par exemple les phénomènes de préemption.

D'un point de vue architectural (chapitre 5), une maquette est composée de processus

. coopérants, par l'intermédiaire de structures de données communes partageables

. concurrents pour l'obtention de ressources physiques.

Son écriture consiste à assembler des composants que l'on a pu standardiser et d'autres qui lui sont particuliers. Cette étude a été l'occasion de concevoir et réaliser un système logiciel, le système MAESTRO, où se trouvent intégrés les outils standard de description et ceux servant aux prises de mesures et à l'analyse statistique (chapitre 6).

- La troisième partie concerne l'utilisation de maquettes.

A partir de ce système logiciel, des maquettes peuvent être écrites à des fins d'évaluation structurale (chapitre 7) ou d'évaluation fonctionnelle qualitative (chapitre 8).

Si l'on désire utiliser une maquette pour étudier quantitativement le comportement dynamique d'un système par simulation (chapitre 9), on retrouve les problèmes inhérents à cette approche : alimentation en données, prise de mesures, analyse statistique, difficultés de valider et de tester la robustesse. Une maquette validée, et jugée suffisamment robuste par rapport à certaines hypothèses, permet de faire de l'analyse et de la prédiction en régime stationnaire ou transitoire.

On mesure l'intérêt de maquettes de systèmes d'information pour la gestion d'une entreprise. Elles aident à appréhender globalement un système, à apprécier son fonctionnement, elles suggèrent des modifications que l'on peut ensuite tester "en laboratoire", sans risque de compromettre l'équilibre parfois précaire du système réel. Enfin, elles permettent une appréhension des coûts d'un système, un des domaines de l'"audit" interne des entreprises.

Cependant, la démarche pour la fabrication d'une maquette est longue et délicate. Le plus souvent, elle se fait par approximations successives

au fur et à mesure que progresse la connaissance sur l'organisation. En ce sens, une maquette est un guide pour accroître cette connaissance.

Enfin, il ne nous semble pas sain de se lancer directement, comme on le fait souvent à l'heure actuelle, dans la conception et la réalisation d'un système d'information, sans s'être assuré qu'il est viable et que ses performances sont acceptables, pour la charge actuelle comme pour la charge future présumée. C'est pourquoi la construction et l'utilisation de maquettes nous paraît appartenir pour l'avenir à la première étape de la conception, contribuant à fixer à la fois l'architecture d'un nouveau système, son organisation administrative et le matériel souhaitable. Ceci à condition de ne pas négliger les aspects humains et économiques des choix que l'on réalise.

Nous n'avons pas la vanité de croire que tous les problèmes sont ainsi en passe d'être résolus. Notre travail n'est encore qu'une exploration dans un domaine très vaste. Visant essentiellement à apporter des outils pour améliorer les méthodes d'analyse des systèmes d'information des organisations, il est une étape nécessaire vers une véritable théorie.

Cette thèse a été voulue aussi lisible et complète en elle-même que possible. Elle s'adresse à un public très large d'informaticiens, de statisticiens, d'organisateurs et de gestionnaires. On ne s'étonnera donc pas d'y trouver des rappels substantiels de concepts, méthodes ou techniques parfois propres à l'une des disciplines couvertes et pouvant être ignorés ou oubliés par certains lecteurs.

Ce travail a bénéficié du soutien du C.N.R.S. et de l'I.R.I.A. dans le cadre de l'ATP n° 102 Informatique d'Organisation, grâce au contrat IRIA n° 77 - 165 sur le projet MAESTRO :

MAquettes pour l'Evaluation de SysTèmes d'infoRmation d'Organisations

Partie I

APPROCHE de la MODELISATION des SYSTEMES d'INFORMATION

Chapitres :

1. SYSTEMES d'INFORMATION.
2. MODELES de SYSTEMES d'INFORMATION.
3. OBJECTIFS et HYPOTHESES de TRAVAIL.

Chapitre 1 - SYSTEMES d'INFORMATION

"Uriel :

Der erste Tag entstand.
Verwirrung weicht, und Ordnung keimt empor.
Erstarrt entflicht der Höllengeister Schar ..."

(J. Haydn, La Création (d'après Lendley et Milton (Le Paradis Perdu)) , 1ère partie, air n° 2).

"Uriel :

Ce fut le premier jour.
Trouble et confusion reculent, et l'ordre naît.
La troupe des esprits infernaux s'enfuit..."

Chapitre 1 - SYSTEMES d'INFORMATION

1. SYSTEMES

1.1 Définition d'un système.

1.2 Etats d'un système.

1.3 Environnement d'un système.

1.4 Sous-systèmes.

1.5 Objectifs de l'étude des systèmes.

2. SYSTEMES d'INFORMATION

2.1 Définition intuitive.

2.2 Le système d'information logique.

- 2.2.1 Information et donnée ; point de vue statique
- 2.2.2 Modification des informations ; point de vue dynamique.
- 2.2.3 Evolution dans le temps d'un système d'information logique.
- 2.2.4 Etats d'un système d'information logique.

2.3 Le système d'information physique.

- 2.3.1 Mémoires.
- 2.3.2 Représentation physique des données et des processus.
- 2.3.3 Ressources du traitement de l'information.
- 2.3.4 Hiérarchie des ressources.
- 2.3.5 Allocation des ressources.
- 2.3.6 Système d'information physique.

Dans ce chapitre, nous délimitons notre domaine d'étude.

Il est nécessaire de rappeler brièvement quelques généralités sur les systèmes (1), avant de parler de systèmes d'information (2).

1 - SYSTEMES

Notre but n'est pas de faire ici une synthèse de la littérature considérable qui existe sur ce sujet, et nous nous contentons de quelques définitons intuitives simples, suffisantes pour une bonne compréhension de notre travail.

1.1 Définition d'un système

De nombreux auteurs ont proposé la leur. Celle de LAROUSSE (LAR 75) est la suivante :

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

Cette définition très générale laisse supposer qu'un système peut avoir un objectif ou ne pas en avoir. Nous laisserons de côté cette question pour ne nous intéresser qu'aux aspects "mécaniques" d'un système, en adoptant la définition suivante, adaptée de celle donnée par B. LANGEFORS (LAN 73) :

> Un système est un ensemble d'objets identifiables (ou entités) qui sont reliés entre eux.

Certaines de ces liaisons sont définies indépendamment du temps et sont dites statiques. Celles dont la définition dépend explicitement ou implicitement de l'écoulement du temps sont dites dynamiques.

D'une certaine façon, les liaisons entre objets peuvent être aussi considérées comme des objets (abstraits).

1.2 Etats d'un système

Les entités et les liaisons peuvent prendre différentes valeurs dont l'assemblage constitue l'ensemble d'états d'un système.

Quand les changements d'états s'opèrent brusquement, de manière discontinue, on les appelle événements.

1.3 Environnement d'un système

On définit l'environnement (ou l'extérieur) d'un système comme tout ce qui n'appartient pas au système.

Si cet environnement peut avoir une influence sur l'état du système, par des liaisons externes, on dit que le système est ouvert ; dans le cas contraire, on dit qu'il est clos.

1.4 Sous-systèmes

On peut, à l'intérieur d'un système donné, en délimiter des parties que l'on appelle sous-systèmes.

On retrouve pour un sous-système tout ce qui a été dit pour le système complet, notamment sur l'ensemble d'états (souvent plus restreint) et les liaisons avec l'environnement (qui comprend maintenant aussi les autres sous-systèmes).

Il est parfois plus commode de considérer séparément les sous-systèmes, et les liaisons qu'ils possèdent, que de s'attaquer d'emblée à l'étude du système tout entier.

1.5 Objectifs de l'étude des systèmes

Délimiter un système, c'est définir un domaine d'observation et d'étude dans la réalité.
On cherche alors à comprendre la structure de ce domaine, son évolution par l'étude de ses changements d'états, le plus souvent à des fins de contrôle ou de prédiction.
C'est, avec une optique particulière, ce que nous tenterons de faire pour les systèmes d'information.

2 - SYSTEMES D'INFORMATION

Ici encore la littérature est abondante et la synthèse difficile. Pour notre part, nous examinons les systèmes d'information à la lumière de ce que nous venons de rappeler brièvement de la "théorie générale" des systèmes.

2.1 Définition intuitive

Nous retiendrons, celle de N.C. CHURCHILL, C.H. KRIEBEL et A.C. STREDY, citée dans (LEM 73) :

> Un système d'information d'organisation est une "combinaison formalisée de ressources humaines et informatiques résultant de la collecte, de la mémorisation, de la recherche, de la communication et de l'utilisation des données, en vue de permettre un management efficace des opérations au sein d'une organisation".

Cette définition se pose donc en termes de moyens et d'objectifs. A nouveau, nous laissons de côté le but à atteindre : "management efficace...", pour ne

nous intéresser qu'aux aspects "mécaniques" du système d'information : "combinaison formalisée ... données".

Si l'on se reporte à la définition d'un système proposée en (1.1), il convient d'abord d'identifier, dans un système d'information, des entités et des liaisons entre ces entités, puis de s'intéresser à ses changements d'états.

La présentation que nous allons en faire maintenant n'est pas neutre : elle reflète une vision subjective du système d'information, ce que l'on pourrait appeler notre "réel perçu" pour reprendre une expression à la mode. Sans doute est-il possible de percevoir différemment cette réalité, de dégager d'autres entités et d'autres liaisons, mais cette vision est celle qui nous a semblé la plus naturelle pour un système en fonctionnement.

Il ne s'agit pas non plus, malgré la définition ci-dessus, d'une présentation formalisée complète d'un système d'information, mais souvent d'une approche pragmatique. En effet, si la partie automatisée d'un système est tout naturellement formalisée, il n'en est pas de même de la partie non automatisée. On tentera pourtant par la suite de modéliser celle-ci avec des concepts proches de ceux qui permettent d'appréhender celle-là. C'est pourquoi nous développons quand c'est possible un point de vue formel.

Nous cherchons maintenant à dresser une sorte de catalogue des éléments d'un système d'information, en donnant à chacun la définition la plus précise possible. Il nous servira à introduire nos objectifs et hypothèses de travail.

Quand on décrit un système, on peut le faire à différents niveaux, en prenant en compte plus ou moins de détails. Les économistes le savent bien, qui parlent de vision micro- ou macro-scopique. Ici, nous ne nous fixons pas a priori de niveaux de description, mais nous reviendrons ultérieurement sur cette importante question.

Tout au long de ce travail, nous serons amenés à distinguer des aspects logiques et physiques, comme nous le faisons maintenant.

2.2 Le système d'information logique

Fondamentalement, un système d'information logique est constitué de données réparties sur différents sites et de transformations effectuées sur ces données. Nous précisons ici ces différentes notions.

2.2.1 Information et donnée ; point de vue statique

Beaucoup d'auteurs différencient les termes "donnée" et "information", mais ils le font de manières assez variées.

- Ainsi, dans (LEM 73), J.L. LEMOIGNE distingue plusieurs niveaux d'informations (ou "moments") selon le nombre de traitements qu'il a fallu faire pour les obtenir. Une donnée serait alors une information primaire (ou de moment 1), c'est-à-dire qu'aucun traitement n'aurait accompagné sa saisie intiale.

Mais l'auteur signale à juste titre le flou et la subjectivité qui s'attache à la notion de "traitement", ainsi que la relativité de la "saisie" de données.

- D'autres, comme B. LANGEFORS (LAN 73), considèrent qu'un message revêt deux aspects :

. un aspect syntaxique, qui ne s'attache qu'à la forme du message et en fait une donnée,

. un aspect sémantique, qui s'attache à sa signification pour une personne et lui confère le caractère d'une information.

Depuis de nombreuses années, on cherche à trouver des structures de données adéquates à la conception et à l'implantation de systèmes informatiques :

- E.F. CODD a proposé un modèle de données relationnel (COD 70) qui a été largement étudié, développé, et complété, par exemple par P.P. CHEN (CHE 76) ; il sert de base aux études actuelles sur les systèmes de gestion de bases de données (AST 76) ;

- Une autre approche est à l'ordre du jour :
celle des types abstraits (par exemple GUT 77, GUT 78) ; elle sert à spécifier axiomatiquement les objets qui interviennent dans un problème informatique, ainsi que leurs relations ; la tendance naturelle de cette approche est d'aller vers des définitions de plus en plus statiques, c'est-à-dire où l'on considère qu'un identificateur ne peut repérer qu'un seul et même objet au cours du temps, par opposition à des définitions dynamiques.

Nous reviendrons sur ces deux approches au chapitre 6.

- La notion d'information a fait aussi l'objet d'études théoriques s'appuyant sur la logique mathématique.

Ainsi dans (PAI 74) et (REM 74), C. PAIR et J.L. REMY distinguent information et donnée. Une information y est un objet "idéal" permettant d'exprimer des affirmations, plus précisément, l'ensemble des théorèmes d'un système formel propositionnel. Une donnée est alors un objet "concret" interprétant (au sens de la logique) une information. On s'y préoccupe aussi des modifications que peuvent subir les informations. Cette manière de voir nous semble adéquate à notre approche des systèmes en fonctionnement, au moins pour leurs parties automatiques. Elle permet en effet d'exprimer leurs caractères statiques et leur évolution dynamique.

Nous développons dans ce paragraphe une formalisation, qui reprend celle de (PAI 74) et (REM 74) en la généralisant, notamment par l'introduction de schémas relationnels[1] (KRE 67), comme cela a déjà été fait par M. CREHANGE dans (CRE 75).

(1) En réalité, une fonction n-aire à valeur dans {vrai, faux} est assimilable à une relation. Sans que cela soit dit explicitement, la formulation de (PAI 74) et (REM 74) inclut donc aussi les relations.

a) Donnée

Rappelons tout d'abord la définition généralement admise :

> Une donnée est un m+2-uple $(E_1, \ldots, E_m, \Lambda, \pi)$, où les E_j $(j = 1, \ldots, m)$ sont des ensembles non nécessairement distincts d'objets, Λ un ensemble de fonctions partielles[2] définies dans des ensembles de la forme $E_{i_1} \times \ldots \times E_{i_q}$ $(q \geq 0$ et $1 \leq i_j \leq m)$, à valeur dans l'un des E_j, et π un ensemble de relations définies dans des ensembles de la forme $E_{i_1} \times \ldots \times E_{i_q}$ $(q \geq 1$ et $1 \leq i_j \leq m)$.

Exemple 1.1

On considère ici une donnée décrivant, sous forme volontairement simplifiée, l'état des commandes-clients d'une entreprise.

Les ensembles d'objets sont client, commande, ligne (de commande), produit, $\mathbb{N}$ (les entiers naturels), $\mathbb{R}$, $\mathbb{R}_+$, $\mathbb{R}_+^*$ (les nombres réels, réels positifs ou nul, réels strictement positifs), CH (les chaînes de caractères alphanumériques), caractères (un ensemble d'objets caractérisant les produits).

Les fonctions partielles de cette donnée sont les suivantes :

p	: *ligne → produit*	: *produit commandé dans une ligne*
nl	: *ligne →* $\mathbb{N}$	: *n° de ligne*
q	: *ligne →* $\mathbb{R}_+^*$	: *quantité commandée*
c	: *ligne → commande*	: *commande contenant une ligne*
s	: *ligne → ligne*	: *fonction successeur dans une liste linéaire de lignes*
tl	: *→ ligne*	: *tête de cette liste (fonction o-aire): premier élément*
np	: *produit →* $\mathbb{N}$	: *n° de produit*
st	: *produit →* $\mathbb{R}_+$	: *quantité en stock*
car	: *produit → caractères*	: *caractéristiques d'un produit*
nc	: *commande →* $\mathbb{N}$	: *n° de commande*
cl	: *commande → client*	: *client ayant passé une commande*
l	: *commande → ligne*	: *1ère ligne de commande dans la liste ci-dessus*
cde	: *N → commande*	: *commande de n° donné.*

(2) il est possible de considérer toujours de vraies fonctions, en introduisant un symbole de non-définition $\perp$. On prolonge alors les fonctions partielles hors de leur domaine de définition en leur y faisant prendre la valeur $\perp$.

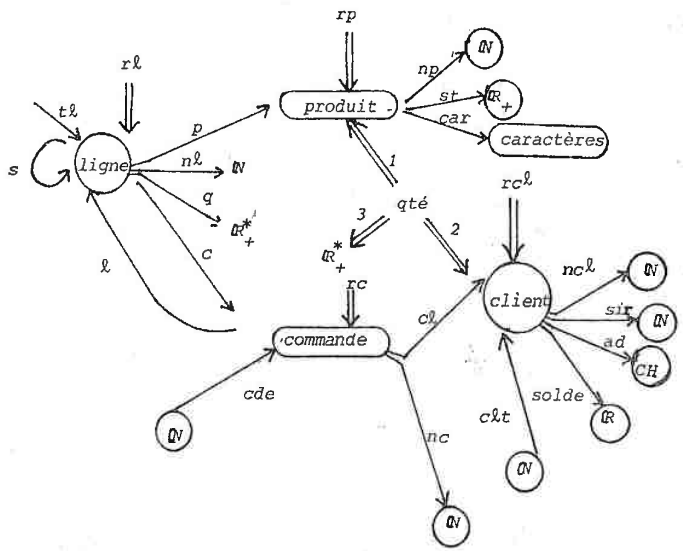
ncl	: client → $\mathbb{N}$	: n° de client
sir	: client → $\mathbb{N}$	: n° de SIRENE (INSEE) d'un client
ad	: client → CH	: adresse d'un client
solde	: client → $\mathbb{R}$	: solde du compte d'un client
clt	: $\mathbb{N}$ → client	: client de n° donné

Les relations suivantes définissent, à un instant précis, les objets recensés par la donnée parmi les objets (potentiels) des ensembles de base cités précédemment.

$rl \subset$ ligne	: ensemble des lignes commandées
$rp \subset$ produit	: ensemble des produits au catalogue
$rc \subset$ commande	: ensemble des commandes
$rcl \subset$ client	: ensemble des clients
qté $\subset$ produit × client × $\mathbb{R}_+^*$	: quantité commandée par un client pour un produit

Ces fonctions et relations ne sont définies qu'à un instant donné : elles font place à d'autres quand le temps s'écoule.

Nous ne discuterons pas l'opportunité de leur choix, ni leur "minimalité", qui sont ici hors de notre propos.



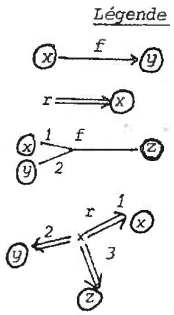


Figure 1.1 - Diagramme sagittal des ensembles, fonctions et relations

□

Une telle donnée peut caractériser l'état du système d'information à un instant donné. Mais si l'on veut pouvoir parler de changement d'état, ou modification, il faut aller plus loin et considérer qu'une donnée est un élément d'une "structure de données". Pour faire des raisonnements sur une telle structure, on lui associe un système formel de la manière suivante.

b) Système formel

Nous ne rappelons ici que sommairement les notions de logique mathématique utilisées dans ce travail. On trouvera des exposés plus complets, bien sûr dans les ouvrages de logique (KRE 67, SCH 73...), mais aussi dans les articles et thèses qui traitent de la théorie formelle de l'information (PAI 74, REM 74, FIN 74...).

> Un système relationnel est un système formel $\mathcal{F} = (\mathcal{L}, F, \mathcal{X}, R)$ vérifiant les conditions énoncées dans les paragraphes suivants ; les formules de F sont des prédicats typés du premier ordre sur un ensemble de formules atomiques A.

i) Alphabet $\mathcal{L}$

On se donne un entier m qui est le nombre de types d'objets du système. L'alphabet $\mathcal{L}$ est défini par :

$\mathcal{L} = L \cup P \cup V \cup \{ \equiv \} \cup \{ (,), \neg, \supset \} \cup \Omega$

L : ensemble des symboles fonctionnels

P : ensemble des symboles relationnels

$V = \bigcup_{j=1}^{m} V_j$: ensemble des variables, V_j désignant l'ensemble des variables de type j

$\equiv$: symbole d'égalité

$\neg$ et $\supset$: symboles de négation et d'implication du calcul des propositions

Ω : ensemble disjoint des précédents et en correspondance bijective avec V : l'élément de Ω correspondant à la variable x est noté $\exists x$.

On note :

[m] : l'ensemble des entiers naturels compris entre 1 et m :

$$[\, m \,] = \{ n \in \mathbb{N} \; ; \; 1 \leq n \leq m \}$$

$[\, m \,]^*$: le monoïde libre sur [m]

λ : le mot vide de $[\, m \,]^*$

$[\, m \,]^+$: l'ensemble des suites finies d'éléments de [m] :

$$[\, m \,]^+ = [\, m \,]^* - \{ \lambda \}$$

Les symboles de L et P sont typés, c'est-à-dire :

. Il existe une application source de L dans $[\, m \,]^*$. La longueur q du mot source (f) s'appelle arité de f, et f est dit symbole q-aire. Si q = 0, f est o-aire et est dit symbole de constante. On définit aussi une application but de L dans [m]. A partir de source et but, on construit une application profil de L dans $[\, m \,]^* \times [\, m \,]$, telle que profil (f) = (source (f), but (f)).

. De la même manière, il existe une application profil de P dans $[\, m \,]^+$. La longueur de profil (r) s'appelle arité de r.

d) Interprétation d'une information ; donnée

Soit une interprétation de L et P, $(E_1, \ldots, E_m, \text{int})$, qui, comme nous l'avons dit, s'étend naturellement aux schémas de S et T.

On montre à nouveau facilement que cette interprétation s'étend à toute formule de F dans l'ensemble des relations du domaine d'interprétation $E = \bigcup_{i \in [m]^+} E_i$, où $E_i = E_{i_1} \times \ldots \times E_{i_m}$ pour $i = (i_1, \ldots, i_n)$.

La liaison entre information et donnée s'exprime alors par la définition suivante :

> Si $I \subset F$ est une information de $\mathcal{F}$, et si toute formule $\varphi \in I$ sans variable est interprétée par int comme l'ensemble du domaine d'interprétation E, alors on dit que int est une réalisation de I et que le m + 2 - uple $(E_1, \ldots, E_m, \text{int}(L), \text{int}(P))$ est une donnée qui réalise l'information I.

Exemple 1.2

Nous reprenons l'exemple 1.1 avec les mêmes notations. Mais, cette fois, les noms des ensembles de base, des fonctions et relations, sont considérés comme des symboles de types, des symboles fonctionnels et relationnels d'un système formel, la donnée de l'exemple 1.1 étant une réalisation particulière d'une information de ce système.

Pour définir celui-ci, il suffit de donner la liste de ses axiomes propres. Afin de ne pas alourdir inutilement la présentation, nous n'en donnons que quelques-uns et nous ne "typons" pas les variables :

Injectivité des fonctions

$(np \,.\, x \equiv xp \,.\, y \supset x \equiv y)$

$(nc \,.\, x \equiv nc \,.\, y \supset x \equiv y)$

Interdépendance des relations

$(rl \,.\, x \supset (rp \,.\, p \,.\, x \wedge rc \,.\, c \,.\, x))$

Surjectivité des fonctions

$(rc \,.\, x \Leftrightarrow \exists \; n \; cde \,.\, n \equiv x)$

Domaine de définition des fonctions

$(r\ell \ . \ x \leftrightarrow \exists \ y\ c \ . \ x \equiv y)$

Ruptures de séquences :

$(\exists\ x\ s \ . \ y \equiv \ell \ . \ x \supset \quad \neg \ c \ . \ y \equiv c \ . \ s \ . \ y)$

□

e) Informations consistantes et complètes

Nous ne rappelons ici que les définitions de la consistance et de la complétude. On trouvera dans (REM 74) de plus amples développements.

Une information I est consistante si $I \neq F$.

Une information est complète si elle est maximale dans l'ensemble des informations consistantes.

S'il est possible, voire indispensable, de considérer d'une telle manière les données que manipule un calculateur, il est totalement exclu de le faire exhaustivement pour des données administratives, traitées "à la main".

Sans doute l'analyse que l'on en fait n'est souvent pas assez poussée et l'on gagnerait beaucoup en rigueur en cherchant à leur appliquer des principes analogues. Ainsi enrichirait-on la connaissance que l'on a des règles qui régissent l'information que l'on manipule, et parviendrait-on parfois à plus de simplicité et de sûreté dans les systèmes administratifs.

Mais il y a bien des zones d'incertitude et de subjectivité, qui contribuent à rendre si difficile la gestion d'une entreprise et à empêcher sa mécanisation complète. Ce que nous voulons dire, c'est qu'il est possible de réduire un peu cette zone de flou, à condition peut-être de ne pas vouloir entrer dans trop de détails et d'admettre une certaine dose d'indéterminisme. Autrement dit, on peut chercher à modéliser les données traitées manuellement avec des concepts analogues à ceux que nous venons de présenter. Ceci fait partie de nos objectifs.

f) Equations récursives dans une information ; définition de nouveaux symboles fonctionnels et relationnels

Dans une information I, il est possible de définir, grâce à une équation récursive, un nouveau symbole fonctionnel f, ou relationnel r, n'appartenant pas à L ou P. Un tel symbole est naturellement interprété comme une nouvelle fonction ou une nouvelle relation du domaine d'interprétation.

La théorie du point fixe étendue aux informations (REM 74) montre que, notamment à la condition que I soit une information complète, cette nouvelle fonction ou relation est nécessairement le plus petit point fixe de la fonctionnelle associée à l'équation récursive dans l'interprétation.

Si l'on suppose que f est de profil (i, j) (resp. r de profil i), il est possible, sous certaines conditions, de montrer que f u (resp. r u) où $u \in S_i$ est sans variable, est calculable : c'est-à-dire qu'il existe une démonstration, ou un calcul, de $f\ u \equiv v$ avec $v \in S_j$ sans variable (resp. r u).

D'une manière plus générale, il est possible de définir ou de calculer de nouveaux symboles fonctionnels ou relationnels grâce à un système d'équations récursives (REM 74).

g) Définition d'une question ; logique du 2ème ordre

Les paragraphes précédents ont montré qu'un système formel fondé sur le calcul des prédicats du ler ordre typé permet de construire de nouvelles fonctions ou relations dans le domaine d'interprétation. Ceci correspond bien, dans la réalité, à des questions que l'on se pose sur une donnée.

Ceci est toutefois insuffisant : souvent, on veut obtenir des résultats synthétiques, que ne permet pas de rendre cette théorie : par exemple, le cardinal d'une relation (voir exemple 1 3).

Il devient alors nécessaire de considérer un 2ème niveau de système formel, lié à celui que nous venons de construire dans une logique du 2ème ordre. Dès que l'on touche à cette notion en informatique, on sait que des difficultés apparaissent et qu'il faut se restreindre considérablement.

On se limite généralement, au 2ème ordre, à un système uniquement fonctionnel où, dans l'interprétation :

. les variables prennent leurs valeurs parmi les relations du domaine interprétant les formules du 1er ordre ;

. les schémas fonctionnels correspondent à des fonctions agissant sur ces relations et à valeurs dans le domaine.

On se borne aussi, au 2ème ordre, à un système propositionnel, c'est-à-dire sans quantificateur, dans lequel il est bon d'avoir, à côté de symboles typés, d'autres qui ne le sont pas (KRE 67), ou qui ne le sont que partiellement (exemple 1.3).

Un tel système du 2ème ordre est sous-jacent à la pluplart des langages d'interrogation que l'on rencontre dans les systèmes de gestion de bases de données.

Nous ne détaillons pas plus ce point, qui conduirait à des développements assez analogues aux paragraphes précédents. On y montrerait aussi que, dans un tel système, il est possible de définir de nouveaux symboles fonctionnels par des systèmes d'équations récursives.

Nous nous contentons de donner un exemple.

Exemple 1.3

Nous reprenons le système formel de l'exemple 1.2. Au 2ème ordre, on considère de nouveaux types : ceux-ci s'interprètent comme des ensembles de parties de produits cartésiens d'ensembles de base.

On a, par exemple, l'ensemble des relations (parties) de :
- l'interprétation de client int (client) : $\mathfrak{P}$ (int (client))
- int (produit) × int (client) × $\mathbb{R}$: $\mathfrak{P}$ (int (produit) × int (client) × $\mathbb{R}$).

Symboles non typés ou partiellement typés du 2ème ordre

CARD est un symbole fonctionnel de source un type quelconque du 2ème ordre et but le type particulier du 1er ordre qui s'interprète comme l'ensemble des entiers naturels $\mathbb{N}$;

$\emptyset$ est une constante non typée (ensemble vide) ;
MOINS est un symbole fonctionnel s'interprétant comme la différence ensembliste ;
CHOIX est un symbole fonctionnel de source un type du 2ème ordre et but un type du 1er ordre.

CARD est défini par l'équation récursive :

$$CARD \,.\, X \equiv \underline{si}\ X \equiv \emptyset\ \underline{alors}\ 0\ \underline{sinon}\ +\,.\ CARD\ .\ MOINS\ .\ X\ .\ CHOIX\ .\ X\ .\ 1$$

où + est un symbole fonctionnel du 1er ordre interprété comme l'addition entière, et X une variable non typée du 2ème ordre.

Symboles typés

TSOLDE est un symbole fonctionnel de source le type interprété comme $\mathfrak{P}$(int (client)) et de but le type du 1er ordre interprété comme $\mathbb{R}_+$. Il est défini par l'équation récursive :

$$TSOLDE \,.\, X \equiv \underline{si}\ X \equiv \emptyset\ \underline{alors}\ 0$$
$$\underline{sinon}\ \dot{+}.TSOLDE.MOINS.X.CHOIX.X.CHOIX.X.solde$$

où $\dot{+}$ est l'addition réelle et X une variable typée : TSOLDE.X s'interprète comme le total des soldes des comptes des clients de l'ensemble int (X).

□

h) Informations et données réparties

Les données d'une organisation sont réparties géographiquement entre un certain nombre de sites k = 1, ..., K.

Comme on l'a vu précédemment, on peut associer au site k un système formel $\mathcal{F}_k$, dit local, un ensemble d'informations $\mathfrak{I}(\mathcal{F}_k)$, une fonction et un domaine d'interprétation locaux.

Naturellement se pose la question importante de la consistance (ou cohérence sémantique) globale d'une telle information répartie. On se place alors dans un système formel $\mathcal{F}$ global, ou extension (REM 74), commun à tous les $\mathcal{F}_k$, pour les faire "coopérer".

Les axiomes de $\mathcal{F}$ expriment la cohérence sémantique entre les différents systèmes locaux.

Ces concepts sont très près de ceux de schémas locaux et globaux des systèmes de gestion de bases de données réparties (cf. par exemple ADIBA (ADI 78)).

Une autre façon de voir est de partir d'emblée d'un système formel global $\mathcal{F}$, et de le "répartir" sur différents sites.

Exemple 1.4 :

Supposons 2 sites (2 points de vente), où l'on a 2 systèmes formels identiques à celui étudié dans l'exemple 1.2, avec les mêmes symboles de types, les mêmes symboles fonctionnels et relationnels, mais indicés par les numéros de sites 1 et 2.

On construit un système formel global qui les fait se correspondre. Pour simplifier, on suppose que seuls les produits et leurs caractéristiques doivent concorder.

Si produit, caractéristiques, $\mathbb{N}$, $\mathbb{R}_+$ sont des types globaux,

rp ⊂ produit : le symbole produits enregistrés,

np : produit → $\mathbb{N}$: le symbole n° de produit,

st : produit → $\mathbb{R}_+$: le symbole de quantité en stock,

car : produit → caractéristiques : le symbole caractéristiques des produits,

v : produit → $\mathbb{R}_+$: le symbole quantité vendue,

tous ces symboles étant "globaux",

on a alors par exemple les axiomes globaux suivants :

$$(np1 \,.\, x1 \equiv np2 \,.\, x2 \supset car1 \,.\, x1 \equiv car2 \,.\, x2)$$

exprime que des produits de mêmes numéros ont mêmes caractéristiques sur les 2 sites ;

$$(np1 \,.\, x1 \equiv np2 \,.\, x2 \supset \exists\, x \quad np \,.\, x \equiv np1 \,.\, x1)$$

exprime la concordance entre nomenclatures locales et globales ;

$$(np \,.\, x \equiv np \,.\, y \supset x \equiv y)$$

est l'injectivité de np ;

$$((np \,.\, x \equiv np1 \,.\, x1 \wedge np \,.\, x \equiv np2 \,.\, x2) \supset st \,.\, x \equiv \dot{+} \,.\, st1 \,.\, x1 \,.\, st2 \,.\, x2)$$

où $\dot{+}$ est un symbole fonctionnel qui s'interprète comme l'addition réelle, exprime que la quantité en stock est égale à la somme des quantités dans les 2 sites.

□

Il est rare que l'information que l'on a dans une organisation soit consistante, au sens donné ici à ce terme, à cause des délais de mise à jour. On cherche bien sûr à tout faire pour les diminuer. Cependant, l'expérience montre que l'on s'accomode souvent assez bien d'un certain niveau d'incohérence globale, sans quoi toute gestion deviendrait impossible.

On montre ainsi l'insuffisance de la théorie, fondée sur les systèmes formels, que nous venons d'esquisser : la grande puissance des règles d'inférence entraîne en effet que la moindre incohérence locale rend inconsistante, donc inutilisable (en théorie heureusement !) toute l'information. Sans doute conviendrait-il d'approfondir cette question.

Notre travail n'est pas une recherche sur la théorie formelle des systèmes d'information. Si les notions présentées précédemment aident à en préciser la véritable nature, elles ne doivent pas être un carcan ou un masque pour les développements que nous allons faire.

Aussi, par la suite, nous "fusionnons" le niveau formel et le niveau interprétatif, en nous plaçant, pour un système donné, dans une seule interprétation. Désormais, un symbole ou un schéma fonctionnel ou relationnel correspond toujours, à un moment donné, à une seule fonction ou relation de l'unique domaine d'interprétation. Ainsi se trouvent confondues les notions de donnée et d'information, comme cela est fait dans beaucoup d'études (voir par exemple le système PIVOINES de M. CREHANGE (CRE 75)).

2.2.2 Modification des informations ; point de vue dynamique

L'information (ou la donnée) d'une organisation est appelée à évoluer dans le temps. On exprime cette évolution à l'aide des concepts de modification, de calcul ou processus.

a) Notion de modification ; structure d'information

Etant donné un système formel[4] $\mathcal{F}$, et $\mathcal{I}(\mathcal{F})$ l'ensemble des informations de $\mathcal{F}$, on considère les applications partielles (non partout définies) $\mathcal{K}(\mathcal{F})$ de $\mathcal{I}(\mathcal{F})$ dans lui-même :

$$\mathcal{K}(\mathcal{F}) = \bigcup_{X \subset \mathcal{I}(\mathcal{F})} \{ X \longrightarrow \mathcal{I}(\mathcal{F}) \}$$

> On appelle structure d'information un couple $\mathcal{S} = (\mathcal{I}(\mathcal{F}), \mathcal{M})$, où $\mathcal{M} \subset \mathcal{K}(\mathcal{F})$ est appelé ensemble des modifications élémentaires de $\mathcal{S}$.

A partir de cet ensemble de modifications élémentaires, on montre (REM 74) que l'on peut définir de nouvelles modifications, de manière statique, par récursivité, de manière analogue à celle qui permet de définir de nouvelles fonctions ou relations (2.2.1.f).

(4) On peut se placer au ler ou au 2ème ordre (voir 2.2.1.g).

Si le système formel $\mathcal{F}$ est réparti sur différents sites, la structure $\mathcal{S} = (\mathcal{I}(\mathcal{F}), \mathcal{M})$ est dite répartie elle aussi.

Si certains axiomes du système sont susceptibles d'être modifiés, d'autres en revanche doivent rester immuables : on les appelle contraintes d'intégrité (voir par exemple (DAT 75)) ou axiomes propres (voir 2.2.1.b).

Exemple 1.5

Reprenons l'exemple 1.4. On peut trouver pour le système formel proposé de nombreuses modifications, par exemple celles qui consistent à ajouter ou supprimer un objet. Ces modifications peuvent d'ailleurs être regroupées, selon leur nature, en des schémas de modifications (FIN 74) ou modifications paramétrées. Ainsi, parmi les modifications définitives,

ajouter_entête_cde 1 (clx, ncx, lx) ajoute sur le site 1 un entête de commande concernant le client clx, avec le n° de commande ncx et avec une 1ère ligne désignée par lx ;

supprimer_entête_cde 1 (ncx) supprime l'entête de la commande de n° ncx sur le site 1.

On aurait de la même manière des adjonctions ou suppressions de produits, clients, lignes de commandes, localement ou globalement.

□

b) Notion de calcul

On considère une structure d'information $\mathcal{S} = (\mathcal{I}(\mathcal{F}), \mathcal{M})$, éventuellement répartie sur différents sites.

> Un calcul est une suite, finie ou infinie, de modifications élémentaires de $\mathcal{M}$.

De la même manière qu'ont été définis des schémas fonctionnels, il est possible de définir des schémas de calculs[5] ou calculs paramétrés

(5) en réalité des schémas d'ensembles de calculs (cf. FIN 74).

(FIN 74). A cette notion correspond intuitivement celle de programme (au sens habituel du mot), et à calcul celle d'élaboration de programme : on dit aussi processus.

Dans une organisation, un programme correspond donc à un "modèle" d'enchaînement logique de modifications à faire subir à l'information.

Exemple 1.6

Dans l'exemple 1.5, on peut imaginer comme schémas de calculs ceux qui consistent à ajouter de nouvelles commandes, ou à supprimer des commandes qui seraient livrées, pour un site : par exemple ajouter_commandes1, supprimer_commandes1 pour le site 1. Ce sont des schémas de calculs qui utilisent des modifications élémentaires, par exemple ajouter_entête_cde1, supprimer_entête_cde1 vues précédemment, ainsi que les adjonctions et suppressions de lignes de commande. Nous ne les décrirons pas en détail. □

c) Structure d'information privée d'un calcul

Pour être exécutées, les modifications élémentaires d'un calcul, qui agissent sur l'information d'un système, sont souvent redécoupées en modifications plus fines qui agissent sur une information particulière au processus. Celui-ci est le seul à pouvoir accéder à cette information et à pouvoir la modifier.

On dit qu'à chaque processus est associée une structure d'information privée ou propre. On peut répéter pour elle tout ce qui a été dit sur la structure d'information générale. Par opposition, on qualifie celle-ci de commune ou partageable, parce qu'elle peut être accédée ou modifiée par tous les processus.

d) Système d'information logique

Un système d'information logique est constitué :
- d'une structure d'information partageable $\mathcal{S} = (\mathcal{Y}(\mathcal{F}), \mathcal{M})$, éventuellement répartie sur différents sites

- d'un domaine $(E_1, \ldots, E_m)$ et d'une fonction d'interprétation
- d'un ensemble $\mathcal{P}$ d'élaborations de programmes (ou processus), fini ou dénombrable.

Certaines des élaborations de programmes de $\mathcal{P}$ peuvent provenir du même programme (schéma de calculs).

Une de nos préoccupations essentielles étant l'étude de l'évolution dans le temps d'un système d'information, nous nous penchons maintenant sur les instants où se produisent les modifications.

2.2.3 Evolution dans le temps d'un système d'information logique

Nous nous intéressons ici aux principes qui régissent l'évolution dans le temps d'un système d'information.

On considère les processus d'un système d'information logique dont les schémas (programmes) sont différents ou non (plusieurs processus pouvant avoir le même schéma) et qui sont finis ou infinis.

Les modifications effectives n'interviennent pas n'importe quand : elles sont ordonnancées dans le temps. On peut tenter d'étudier le moment où elles s'effectuent en tenant compte du fait qu'elles ne sont pas instantanées. Leur durée dépend d'un certain nombre de facteurs que nous serons amenés à expliciter.

Pour l'instant, nous allons raisonner comme si l'on avait tenu compte de ces durées une fois pour toutes, au moment de la définition du système, et nous ne les considérerons plus explicitement à ce niveau logique.

- On peut alors prendre le temps parmi les entités du système, comme un symbole fonctionnel de constante (fonction o-aire) $t : \rightarrow \mathbb{R}_+$, où $\mathbb{R}_+$ (ensemble des réels ≥ 0) représente l'échelle de temps.

Dans le cadre d'un système réparti se pose le problème de la relativité du temps à cause du phénomène de "dérive" des horloges sur les différents sites (LEL 77). Ceci pourrait se traduire ici par l'introduction de plusieurs symboles et axiomes (contraintes d'intégrité) exprimant comment les re-synchroniser. Ne se préoccupant pas de questions aussi fines et liées à la technologie, notre travail ignore celle-ci et considère dès lors un temps absolu, donné par une horloge unique, à laquelle tous les processus peuvent accèder sans pouvoir la modifier ("variable exogène").

Dans ces conditions, et puisque nous avons défini une modification comme une fonction partielle, sa réalisation peut n'être possible que pour certains états initiaux de l'information, notamment ceux liés aux valeurs du temps, par exemple pour un déclenchement à des dates déterminées. Ceci est bien sûr générateur de blocages pour les processus.

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

. actif : une modification est en cours d'élaboration ;

. passif (ou inactif ou bloqué) : dans le cas contraire.

Ainsi, un processus se trouve dans l'état passif si l'état du système d'information logique (y compris le temps) ne lui permet pas de poursuivre ses modifications, ou quand il est terminé, c'est-à-dire quand toutes ses modifications ont été réalisées. Un processus bloqué non terminé peut être débloqué (rendu actif) dès que l'état du système lui permet d'entreprendre la prochaine modification.

A chaque instant, il est possible de repérer l'endroit où le processus est parvenu, c'est-à-dire la modification en cours : cet endroit fait partie de l'état du processus. Dans le cas d'un programme informatique, si l'on regarde les choses finement, ceci correspond à la notion de numéro d'instruction, ou compteur ordinal.

- Une difficulté essentielle surgit lorsque deux processus sont actifs simultanément pour traiter le même élément d'information : on sait que l'effet produit peut être imprévisible (cf. par exemple CRO 75).

Pour se garantir contre ce genre de phénomène, il faut imposer dans certains cas l'exclusion mutuelle de deux modifications. Ceci se traduit par une relation binaire $\mathcal{E}x$ dans l'ensemble des modifications élémentaires $\mathcal{M}$: $\mathcal{E}x \subset \mathcal{M} \times \mathcal{M}$. Cette relation symétrique et non transitive est construite par une analyse soignée de l'information et des modifications.

La question est alors de savoir comment ordonnancer dans le temps les processus : entre deux processus dont les modifications s'excluent à un instant, lequel exécuter et lequel faire attendre ? Toutes les politiques de choix sont possibles : premier arrivé - premier servi, avec priorités fixes ou variables, avec préemption, au hasard, globalement ou localement... On voit ainsi que l'exclusion mutuelle entre modifications est source de nouveaux blocages pour les processus.

Exemple 1.7

Deux processus p1 et p2, issus respectivement des schémas précédents (exemple 1.6) ajouter_commandes1 et supprimer_commandes1, peuvent se trouver bloqués parce que l'état du système ne permet pas leur continuation.

Ainsi, si l'on a décidé d'entrer les commandes tous les jours ouvrables à 17 heures sur le site 1, p1 reste bloqué jusqu'à ce que cette date soit atteinte. Alors les entrées éventuelles sont effectuées, puis le processus retourne à l'état passif jusqu'à l'heure favorable suivante.

De même, si l'on supprime des commandes du site 1 dès que c'est possible, le processus p2 reste passif tant que le système n'est pas dans un état favorable, c'est-à-dire tant qu'il n'y a pas de commandes à supprimer. Il devient actif quand c'est le cas, puis retourne à l'état passif une fois le travail effectué.

On a ici les deux causes habituelles de blocage d'un processus sur l'état du système.

On doit adjoindre les causes de blocage dues à l'exclusion mutuelle des modifications : par exemple, ajouter_entête_cdel et supprimer_entête_cdel sont exclusives pour des raisons évidentes de sécurité, ce qui est source de nouveaux blocages pour les processus considérés.

□

2.2.4 Etats d'un système d'information logique

En théorie, il est exclu de vouloir observer un système d'information en cours de modification, parce que l'information n'y est pas stable. On ne peut le faire qu'à un instant où aucune modification ne se déroule.

Dans ce cas, l'état d'un système d'information est caractérisé par
- l'état de l'information,
- l'état des processus.

Mais on sait bien qu'en réalité, on ne se prive pas de consulter de l'information localement, à un endroit où elle apparaît stable, même si, en d'autres lieux, elle est en plein bouleversement. Ceci montre encore une fois la faiblesse et l'incomplètude de la description que nous venons d'esquisser, mais aussi toute la difficulté d'approcher ces questions.

On pourrait s'en tenir là et tenter de prendre en compte la dynamique d'un système uniquement à travers des considérations d'un niveau logique où :
- l'exclusion mutuelle serait imposée entre certaines modifications ;
- chaque modification serait valuée par une durée d'exécution arbitraire;
- une politique d'ordonnancement des processus dans le temps aurait été retenue.

Mais ce serait s'éloigner considérablement de la réalité et ignorer qu'il y a bien d'autres sources de blocages possibles pour des processus, qui tiennent aux moyens physiques de traitement.

2.3 - Le système d'information physique

Il comprend toutes les ressources nécessaires à l'enregistrement des données et à leur transformation ; on y distingue des mémoires, des moyens passifs et des moyens actifs de traitement.

Ces éléments interviennent pour une large part dans la dynamique du système d'information : ils confèrent aux modifications leur durée et ils sont source de blocages supplémentaires pour les processus.

Ici encore, nous tentons de dresser un catalogue des éléments physiques, sans préjuger du niveau de détail choisi ultérieurement pour leur représentation dans des maquettes.

2.3.1 Mémoires

Chaque mémoire de l'organisation a pour rôle l'enregistrement à chaque instant d'une partie de donnée locale ou globale, reflétant ainsi un état de l'organisation

On distingue les mémoires informatiques et les autres.

a) Mémoires informatiques

La description des différents types de mémoires d'ordinateurs, adressables ou non, centrales ou secondaires, est bien connue des informaticiens. A ce propos, on se reportera par exemple à C. PAIR (PAI 71), où sont formalisés les différents modes d'accès et de modification des mémoires.

b) Autres mémoires de l'organisation

Ce sont celles qui correspondent au rangement de données dans les services de l'organisation, sur différents types de supports : feuilles de papier, fiches cartonnées, cartes perforées, etc...

Il est évidemment exclu d'appréhender exactement en termes mathématiques les modes de rangement sur ces différents supports, une telle organisation présentant toujours un certain caractère de flou. Cependant, en approfondissant l'étude de l'organisation, on peut souvent trouver des structures de rangement sous-jacentes, à rapprocher des mémoires de machines adressables ou non, finies ou infinies.

Exemple 1.8

Une chemise cartonnée, un classeur, un livre ou une armoire ne sont-ils pas comparables à des fichiers séquentiels, ou à accès direct quand est constitué un système d'indexage ?

□

Bien qu'elle soit sujette à bien des approximations par rapport à la réalité, nous ferons dans nos maquettes l'hypothèse que l'on peut modéliser aussi accès et modifications pour les supports traditionnels de l'information.

2.3.2 Réprésentation physique des données et des processus

a) Données

La question qui est posée ici est celle de l'implantation physique dans les mémoires de l'organisation de données qui évoluent dans le temps.

Nous n'examinons que le cas de l'informatique, où, pour l'usager, l'implantation est souvent réalisée automatiquement par l'intermédiaire d'un langage de programmation.

C'est pour mieux étudier cette question qu'ont été lancés les récents travaux théoriques sur la représentation des types abstraits (J.V. GUTTAG (GUT 78), M.C. GAUDEL, G. TERRINE (GAU 78), J.P. FINANCE (FIN 78)...).

Avec la prise en compte des réseaux et de la répartition des données, cette question rebondit et s'amplifie : elle fait l'objet d'études importantes dans le cadre de projets actuels (cf. SIRIUS (SIR 78)).

On considère le plus souvent pour les informations (ADI 78) :

- un schéma externe qui est approximativement le système formel réparti que nous avons présenté ;
- un schéma interne qui est la carte de l'implantation permanente dans les mémoires de ce système, généralement réduit à des symboles de base et à des axiomes ; les valeurs des schémas fonctionnels ou relationnels ou les théorèmes s'en déduisent à volonté : n'étant souvent utiles que temporairement (cf. 2.2.2), on leur alloue si nécessaire des espaces de mémoire temporaires.

Ces deux schémas, eux-mêmes codés et implantés en mémoire, peuvent subir aussi des modifications au cours du temps.

b) Modifications et processus

La représentation physique des données conduit naturellement à définir en termes d'accès et de modification de mémoires les modifications et processus logiques du système.

Ceux qui sont automatiques sont ensuite eux-aussi représentés physiquement sous forme d'instructions-machine, le plus souvent par l'intermédiaire d'un langage de programmation, ce qui nous ramène aux considérations précédentes. On conserve généralement ces programmes sur des mémoires auxiliaires, mais on leur alloue une partie de la mémoire centrale pour l'exécution des processus qui leur correspondent. Ces implantations en mémoire sont variables au cours du temps.

2.3.3 Ressources du traitement de l'information

Parmi les ressources dont ont besoin les processus pour évoluer, on distingue : les données, les mémoires, des moyens "passifs" et "actifs" de traitement d'information.

a) Les données

Puisqu'un processus peut être bloqué en attente d'un état convenable des données, il est courant de considérer celles-ci comme des ressources.

Un état permettant à un processus p1 de poursuivre son évolution a pu être engendré par un processus p2 : on dit alors que p1 et p2 coopèrent. D'une manière générale, on dit que tous les processus d'un système sont coopérants.

b) Les mémoires

Pour évoluer, un processus a besoin de mémoire :

- comme espace propre, pour sa structure de données privée et ses instructions quand c'est un processus informatique ;
- comme espace commun, pour accèder à des données ou ranger des résultats de la structure de données commune.

Dans les deux cas, cette nécessité d'espace est génératrice de nouveaux blocages pour les processus :

- par manque de mémoire disponible,
- par exclusion mutuelle d'accès : souvent, par précaution, il n'y a qu'un point d'accès (voir 2.3.5) à une mémoire.

c) Moyens passifs de traitement d'information

Ce sont des éléments auxiliaires, indispensables au traitement de l'information, mais qui ne peuvent pas travailler de manière autonome.

Exemple 1.9

Du matériel de bureau divers, du papier, une ligne téléphonique, un clavier de machine à écrire ou de télétype, une tête de lecture-écriture, etc... peuvent être considérés comme des ressources passives.

□

En l'absence d'une telle ressource qui lui serait nécessaire, un processus se bloque.

Ces moyens passifs ne peuvent jamais être utilisés sans un moyen actif de traitement, dont ils peuvent souvent être considérés comme des constituants (2.3.4).

d) Moyens actifs de traitement (ou processeurs)

Si un processus ne possède pas l'usage d'au moins une telle ressource active, il est nécessairement passif.

Ils se répartissent en moyens matériels et humains.

- Moyens matériels actifs

Ce sont des organes capables de fonctionner de manière autonome, au moins pendant un certain temps, pour effectuer une opération de traitement d'information :

Exemple 1.10

Le bloc logico-arithmétique d'un ordinateur, un photocopieur, un ordinateur considéré globalement, etc... peuvent être considérés comme des moyens actifs, à différents niveaux hiérarchiques (2.3.4).

□

- Moyens humains

Certains traitements d'information sont exécutés par des personnes, avec le secours éventuel de moyens matériels.

Exemple 1.11

On peut s'intéresser à des personnes prises individuellement, lorsqu'elles effectuent un travail spécifique.

On peut aussi considérer globalement des groupes de personnes ou des services, quand on ne veut pas entrer dans les détails.

□

Les processus d'un système d'information partagent donc un ensemble de ressources de natures différentes pour l'obtention desquelles ils sont dits concurrents.

2.3.4 Hiérarchie des ressources

On pourrait tenter de dresser une liste détaillée des ressources élémentaires intervenant dans le traitement de l'information. Elle risquerait d'être fort longue, et il n'est pas certain qu'elle apporterait beaucoup pour une étude globale de systèmes d'information.

De plus, ces différentes ressources élémentaires ne sont pas indépendantes : elles sont souvent des éléments constitutifs d'autres ressources de niveau plus élevé.

Exemple 1.12

Si l'on considère qu'une bascule, un bus ou un circuit additionneur, etc..., sont des ressources élémentaires (ce qui est d'ailleurs discutable), l'important est sans doute, à un certain niveau, que ces éléments fassent partie d'une ressource appelée unité centrale. Mais on peut aussi dire que celle-ci appartient à une ressource que l'on nomme ordinateur.

□

Dans ces conditions, où s'arrêter ?

Certainement quand on a regroupé les ressources élémentaires en blocs pouvant être alloués tout entier à des processus.

De tels blocs qui sont semblables peuvent éventuellement faire partie d'une même unité : on dit qu'ils sont des accès d'une même ressource.

2.3.5 Allocation des ressources

Il existe de multiples politiques d'allocation des ressources. Certaines d'entre elles peuvent être complètement précisées : c'est le cas pour l'allocation des ressources d'un ordinateur à des processus, où la politique définie pour le système d'exploitation (operating system) est appliquée automatiquement. D'autres sont moins rigoureuses et gardent un certain caractère de flou : c'est le cas pour l'allocation de personnes à certains travaux, où la politique, qui a pu être définie (avec précision ou pas) au moment de l'organisation des services et du travail, est plus ou moins respectée, où l'on s'adapte aux circonstances et bouscule parfois (et heureusement !) les règles ou recommandations.

a) L'allocation des ressources est statique ou dynamique : dans le premier cas, on alloue des ressources à un processus pendant une certaine période, indépendamment de l'usage qu'il en fait. Dans le deuxième, ou allocation à la demande, on ne cherche à allouer des ressources à un processus que s'il en a réellement besoin et on les lui retire dès que ce besoin n'existe plus.

b) Une ressource est dite partageable avec n points d'accès ($n \geq 1$) si elle peut être attribuée au même instant à n processus au plus. Une ressource à un seul point d'accès est dite critique.

Une ressource est sujette à préemption (ou réquisition) si elle peut être retirée à un processus, alors que celui-ci n'a pas exprimé le désir de s'en désaisir.

c) L'allocation des ressources peut être décentralisée ou centralisée.

Dans le cas décentralisé, il existe des allocateurs contrôlant chacun plusieurs ressources. Dans le cas centralisé, toutes les ressources sont contrôlées par le même allocateur. Cette solution, de stratégie globale, permet plus de cohérence mais est plus difficile à mettre en œuvre.

- Les politiques d'allocation décentralisées dépendent beaucoup de la nature des ressources attribuées et il est vain de tenter de les citer toutes.

. Pour une ressource critique, quelques politiques sont :

PAPS : premier arrivé-premier servi ("FIFO : first in - first out") ;

DAPS : dernier arrivé-premier servi ("LIFO : last in - first out") ;

règle du "tourniquet" ("Round - robin" (KLE 76)) : donner à tour de rôle aux processus demandeurs l'usage de la ressource pendant une fraction (quantum) de temps, fixe ou variable ;

avec priorités (fixes ou variables) ;

avec préemption ou non, quand la question se pose ;

etc...

. Pour une ressource partageable à n points d'accès ($n > 1$), avec des processus demandant p accès ($1 \leq p \leq n$) :

satisfaire le plus grand nombre possible de processus dans l'ordre de leur arrivée (PAPS) ;

idem, mais dans l'ordre inverse (DAPS) ;

idem, mais dans l'ordre de leurs priorités (fixes ou variables) ;

avec ou sans réquisition ;

etc...

- Nous ne parlons des stratégies globales que pour signaler le problème important de l'interblocage ("deadlock").

Deux processus peuvent être en situation telle que chacun accapare des ressources tout en attendant celles possédées par l'autre : ces processus se bloquent alors indéfiniment. On dit qu'ils sont interbloqués.

L'interblocage peut mettre en jeu un nombre important de processus et de ressources. Le déblocage de tous les processus (guérison) peut être très complexe, voire impossible sans destruction d'une partie des processus interbloqués.

On conçoit alors tout l'intérêt d'une stratégie globale d'allocation qui assure la prévention de ce phénomène.

Ces techniques existent et sont décrites dans les ouvrages sur les systèmes d'exploitation : dans CROCUS (CRO 75), on distingue prévention statique (demande globale, allocation hiérarchisée de ressources, communication hiérarchisée de processus (P. BRINCH HANSEN (BRI 73)), et prévention dynamique (algorithme du banquier (E.W. DIJKSTRA (DIJ 67), A.N. HABERMANN (HAB 69))).

d) Allocation dans les systèmes d'information

- Dans un système informatique, c'est le système d'exploitation (operating system) qui est chargé de l'allocation des ressources aux processus : c'est un ensemble de processus dont on sait la grande complexité. On peut considérer qu'il appartient à la partie automatique du système d'information physique, d'autant que ses processus participent au partage général des ressources et influencent en cela le comportement du système d'information.

- Pour la partie non automatique (ou administrative) de ce système, nous avons déjà signalé l'incertitude qui caractérise la définition des ressources et leur allocation.

Ceci est d'ailleurs compliqué par le fait que les moyens que possède une entreprise ne sont pas disponibles en permanence : c'est le cas du personnel avec les congés, les horaires de travail (fixes ou variables), les arrêts pour maladie, les grèves, etc... La description formelle exhaustive de l'allocation tient donc ici de la gageure et l'on doit se contenter d'approximations importantes dans des modèles.

2.3.6 Système d'information physique

On voit donc qu'un système d'information physique est d'une nature très complexe. Pour résumer ce qui vient d'être dit, on peut adopter la définition générale suivante :

> Un système d'information physique est un système de processus coopérants, concurrents vis à vis d'un ensemble de ressources.

Il nous a semblé souhaitable de chercher à simplifier cette réalité, notamment pour mieux prendre en compte le "non-formalisé".

"Cavaradossi :

La vidi ieri, ma fu puro
caso. A pregar qui venne ....
non visto la ritrassi"

(G. Puccini, La Tosca, acte 1).

"Cavaradossi (peintre) :

Je l'ai vue hier, mais par pur
hasard. Elle venait faire sa prière ...
J'ai peint son portrait sans me faire remarquer"

Chapitre 2 - MODELES de SYSTEMES d'INFORMATION

1. MODELES

   1.1 Définition intuitive.

   1.2 Classification des modèles.

   1.3 Niveau de détail.

   1.4 Objectifs d'une modélisation.

   1.5 Validation et robustesse d'un modèle de prédiction.

   1.6 Simulation informatique d'un système.

2. DEFINITION INTUITIVE d'un MODELE de SYSTEME d'INFORMATION

3. PANORAMA des MODELES pour les SYSTEMES d'INFORMATION

4. MODELES ORIENTES VERS l'ANALYSE des SYSTEMES d'INFORMATION

   4.1 Méthodes et outils traditionnels.

   4.2 Modèle de FORRESTER.

   4.3. Autres modèles.

Nous rappelons brièvement ce que l'on entend par modèle d'une manière générale (1), puis nous l'appliquons aux systèmes d'information (2). Après avoir dressé un panorama des modèles pour ces systèmes (3), nous étudions ceux qui sont destinés plus précisément à leur analyse (4).

1 - MODELES

L'utilisation de modèles est largement répandue comme un moyen pour étudier des systèmes complexes.

1.1 Définition intuitive

Il existe de nombreuses définitions de ce terme. Celle de ROBERT (ROB 73) est la suivante :

> Un modèle est la représentation simplifiée d'un système.

Après une définition semblable, LAROUSSE (LAR 75) précise :

> Le modèle réduit doit fonctionner sur les mêmes principes que l'original en vraie grandeur.

Nous adopterons cette définition et ferons nôtre cette règle. Nous les préciserons ultérieurement dans le cadre des systèmes d'information.

1.2 Classification des modèles

Dans son ouvrage "Industrial Dynamics", FORRESTER (FOR 61) propose une classification intéressante des modèles dont nous rappelons les grandes lignes.

- On distingue tout d'abord des modèles physiques et des modèles abstraits. Les premiers sont des répliques physiques, souvent à échelle réduite, d'objets à l'étude.

Exemple 2.1

Des maquettes d'avions, testées en souffleries, ou de bateaux éprouvées dans des bassins de carènes, sont des modèles physiques. □

Les seconds sont constitués par des symboles assemblés selon la syntaxe d'un langage.

Exemple 2.2

Une description en français est un modèle abstrait. □

Les modèles mathématiques, qui utilisent un langage plus précis, moins ambigü, font partie de cette classe.

- On distingue des modèles statiques et des modèles dynamiques. Dans un modèle statique, le temps n'intervient pas.

Exemple 2.3

La loi de MARIOTTE $p V = nr T$ pour les gaz parfaits est un modèle statique (mathématique). □

Dans un modèle dynamique, le temps intervient, explicitement ou implicitement.

Exemple 2.4

La loi $F = m \Gamma$, relation fondamentale de la mécanique classique, régit la dynamique du solide. □

- Parmi les modèles mathématiques, on distingue ceux qui sont analytiques et ceux qui sont numériques.

Les premiers peuvent donner des résultats généraux.

Exemple 2.5

Les deux modèles ci-dessus, qui lient entre elles des variables dont l'une quelconque prend une valeur déterminée quand les autres ont des valeurs fixées, sont analytiques. □

Les seconds sont plutôt des procédés de calculs, ou algorithmes, qui fournissent un résultat pour une donnée précise.

Exemple 2.6

La méthode de NEWTON pour le calcul approché d'un zéro d'une fonction réelle dérivable est un modèle numérique. □

- Parmi les modèles mathématiques, on distingue aussi ceux qui sont déterministes de ceux qui sont stochastiques.

Les premiers n'utilisent que des relations certaines.

Exemple 2.7

La loi de MARIOTTE est un modèle déterministe. □

Les seconds font intervenir le hasard par des lois de probabilité.

Exemple 2.8

Les chaines de MARKOV constituent un modèle stochastique. □

Il existe d'autres critères de classification que nous ne développons pas ici, et qui s'appliquent d'ailleurs aussi aux systèmes étudiés : stabilité, stationnarité ...

1.3 Niveau de détail

La représentation d'un système par un modèle peut se faire de manière plus ou moins détaillée, selon que l'on prend en compte dans le modèle plus ou moins d'entités et de relations du système.

Un plus grand détail permet de faire intervenir un plus grand nombre de sources de variation et semble permettre de mieux approcher le système réel.

Mais l'expérience montre qu'un luxe de détails trop important est souvent nuisible :

- il rend plus difficile l'étude du système en noyant des éléments essentiels parmi des éléments secondaires ;
- il accroît la difficulté de la résolution du modèle ;
- il éloigne parfois de solutions intéressantes, par exemple analytiques, pour privilégier des solutions plus ponctuelles, par exemple numériques.

En tout état de cause, tout dépend du but que l'on poursuit : on peut faire, à partir du même système, des modèles très divers, à des niveaux de détails différents. Un modèle n'est en général bon que pour réaliser des objectifs précis. Un modèle qui serait universel n'existe pas, ou alors ce serait le système étudié lui-même, ce qui ramène au point de départ...

1.4 Objectifs d'une modélisation

Le but d'un modèle est de décrire, d'évaluer ou de prévoir.

- Un modèle descriptif s'intéresse à la structure du système. Il cherche à représenter, éventuellement à différents niveaux, les entités et les relations du système, pour en faciliter la compréhension et la communication à autrui.

Exemple 2.9

Des plans d'architecture (à différents niveaux : plans-masse, plans techniques ...) ou des schémas de sciences naturelles (à différents niveaux : oeil, loupe, microscope ...) permettent de mieux "visualiser" un système statique. Un dessin animé représentant le fonctionnement d'une chaîne de production aide à comprendre sa dynamique.

- Un modèle d'évaluation fonctionnelle qualitative sert à vérifier qu'un système dynamique fonctionne convenablement, conformément à ses spécifications logiques.

On peut chercher à le prouver mathématiquement (cf. "preuves de programmes" en informatique). A défaut, on construit souvent un modèle

du système qui permet, par des tests, de détecter des vices de fonctionnement.

Exemple 2.10

En électronique, on est parfois conduit à faire des modèles programmés de circuits, et l'on teste sur ordinateur qu'ils réalisent bien les fonctions logiques que l'on attend d'eux. □

On peut mettre dans cette classe les modèles de "faisabilité", dont le rôle est de s'assurer, par des techniques similaires, qu'un mécanisme est correct, avant de le fabriquer en vraie grandeur, éventuellement en série.

- Un modèle dynamique d'évaluation quantitative aide à faire apprécier le comportement dans le temps du système qu'il décrit.

. Si le modèle est analytique, on obtient des valeurs numériques, éventuellement soumises à probabilités, en appliquant des formules générales.

. Sinon, on obtient des valeurs numériques en appliquant des algorithmes ; on en répète parfois plusieurs fois l'exécution quand le modèle est stochastique, afin d'obtenir des séries chronologiques.

Ces résultats ont parfois une valeur "absolue", qui reflète assez bien la réalité, éventuellement dans un intervalle de confiance. Mais, ils n'ont parfois qu'une valeur "relative", permettant uniquement de comparer une situation à d'autres situations. Dans le deuxième cas, on utilise plutôt le modèle pour comprendre un phénomène, pour mettre en évidence ses entités et relations fondamentales : on dit qu'on a un modèle explicatif.

Exemple 2.11

On s'intéresse actuellement beaucoup aux modèles de systèmes d'exploitation. Ainsi, le phénomène d'"écroulement" peut être expliqué par un modèle assez grossier, qui met en évidence le rôle joué par le degré de multiprogrammation et l'algorithme de pagination de la mémoire (CRO 75).

Mais si l'on veut des renseignements quantitatifs précis, notamment pour prévenir l'écroulement, on doit construire des modèles plus fins (GEL 73). □

- Un modèle prévisionnel est un modèle dans lequel les valeurs de certaines entités ou relations sont fixées pour représenter une situation hypothétique différente de la réalité actuelle ou connue du système.

Un tel modèle permet d'obtenir des résultats que l'on suppose correspondre à ce qui se passerait réellement si le système était dans cette situation. En ce sens, on dit que l'on fait de la prédiction.

Exemple 2.12

Dans les entreprises, on fait souvent, même s'ils ne sont pas toujours bien explicités, des modèles de prévision de ventes, de trésorerie, de renouvellement de matériel, etc ... □

On peut s'intéresser soit à un comportement global dans le temps du système placé dans telle ou telle situation, soit à la survenance d'un événement précis.

Naturellement, il n'existe pas de limite bien franche entre ces différentes catégories : un modèle peut toucher à plusieurs d'entre elles. Mais, si l'on ne veut pas courir le risque de l'inefficacité, il importe de bien préciser ce que l'on attend du modèle que l'on définit. Ceci doit être lié avec la recherche d'un niveau de détail convenable, ainsi que nous le suggérions précédemment.

1.5 Validation et robustesse d'un modèle de prédiction

L'utilisation d'un modèle de prédiction demande beaucoup de précautions.

Tout d'abord, il est nécessaire de le valider, c'est-à-dire de s'assurer qu'il est capable de rendre compte de situations (réelles

ou hypothétiques) que l'on connaît. L'ensemble des situations pour lesquelles cette vérification est faite s'appelle le domaine de validation !

Exemple 2.13

On vérifie que la loi de MARIOTTE s'applique (au moins de manière approchée) à l'oxygène, à l'hélium, au méthane dans une certaine plage de températures. □

Naturellement, l'usage d'un modèle de prédiction suppose que l'on cherche à savoir ce qui se passe dans une situation qui se trouve en dehors du domaine de validation. On suppose alors que le modèle est encore valide dans cette situation et qu'il est capable d'y fournir de bons résultats.

Un modèle qui possède cette qualité est dit robuste.

Exemple 2.14

La loi de MARIOTTE est un modèle robuste puisqu'elle s'applique à tous les gaz parfaits, dans de larges plages de températures. □

Mais, attention !, il n'est pas certain que tous les résultats que l'on obtient en dehors du domaine de validation soient corrects : souvent, il faut préciser vis à vis desquels d'entre eux un modèle est robuste et, naturellement, essayer de délimiter un domaine de robustesse.

1.6 Simulation informatique d'un système

La simulation consiste à remplacer l'étude d'un système par l'étude d'un de ses modèles, en faisant l'hypothèse que les résultats que l'on obtient sur celui-ci sont transposables à celui-là.

La simulation qui nous intéresse ici est la simulation à l'aide d'un modèle programmé, c'est-à-dire d'un modèle codé dans un langage de programmation approprié, qui est ensuite exécuté par un calculateur, sans aucune intervention externe.

Ceci nécessite donc l'usage d'un modèle clos[1].

(1) un modèle est aussi un système : la définition d'un système clos s'applique donc aussi à un modèle.

Les avantages d'un tel procédé sont multiples :

- on peut, si le modèle est dynamique, dilater le temps ou, au contraire, le comprimer ;
- il est possible d'arrêter et de reprendre la simulation à volonté ;
- plusieurs "répliques" de la même simulation sont possibles : si le modèle est stochastique, ceci permet de faire des statistiques ;
- la modification du modèle est très simple, aussi bien pour les entités que pour les relations ;
- on peut expérimenter, tester, évaluer des systèmes nouveaux ou des modifications sur des systèmes existants.

Nous verrons cependant par la suite que cette procédure présente quantité de difficultés.

2 - DEFINITION INTUITIVE d'un MODELE de SYSTEME d'INFORMATION

En reprenant les définitions de système d'information et de modèle données précédemment, on peut adopter la définition générale suivante :

> Un modèle (ou une maquette) de système d'information est une représentation symbolique de ce système.

Cette représentation peut être statique ou dynamique, analytique ou numérique, déterministe ou stochastique ...

Le niveau de détail (le "facteur d'échelle") peut être plus ou moins grand selon ce que l'on désire représenter, selon les objectifs et les facteurs que l'on veut retenir.

Les informaticiens ou les organisateurs ressentent le besoin de représenter les systèmes d'information, en cours de définition, de construction ou d'exploitation, sur lesquels ils travaillent. Ils le font dans des optiques différentes, en utilisant des méthodes et techniques diverses : certaines très simples et pragmatiques ne visent souvent que des objectifs descriptifs, d'autres plus sophistiquées sont utilisées à des fins d'évaluation et de prédiction.

Nous allons maintenant, sans vouloir être exhaustif (qui pourrait y prétendre dans un domaine aussi vaste ?), passer en revue certaines d'entre elles, qui nous semblent caractéristiques des visées que l'on peut avoir en faisant des modèles. Nous les confronterons aux objectifs dégagés en (1.4), ce qui nous permettra de bien situer les propositions faites dans ce travail.

3 - PANORAMA des MODELES pour les SYSTEMES d'INFORMATION

Depuis quelques années, on assiste à une floraison de propositions de modèles pour les systèmes d'information. Tous ne sont cependant pas destinés aux mêmes buts : le groupe de travail n° 2 d'INFORSID a cherché à en faire une typologie autour de quatre mots-clés : analyse, conception, réalisation, utilisation de systèmes d'information (INF 76). Cette classification est utile pour dégager les caractéristiques des modèles proposés, bien qu'aucun d'entre eux ne se limite strictement à une seule classe.

- Les modèles d'analyse de systèmes d'information, auxquels se rattache plutôt ce que nous présentons ici, seront développés dans le paragraphe suivant (4).

- Les modèles de conception sont de loin ceux sur qui a été porté l'effort principal, parce que le problème est fondamental et que le domaine est très vaste.

Cela va de modèles pour spécifier des données et des programmes à des modèles pour concevoir des systèmes tout entiers. Nous ne parlons ici que de ce dernier aspect, les autres étant mentionnés incidemment tout au long de notre étude.

Ainsi, le modèle MACSI de F. PECCOUD (PEC 75) aide à décrire les parties automatiques ou non d'un système. La partie MACSI2 et le langage OASIS permettent de conduire des sessions de simulation interactive

avec les futurs utilisateurs, ce qui les pousse à mieux exprimer leurs besoins. La version MACSI - P (GIR 75) permet de construire des "prototypes fonctionnellement équivalents" aux systèmes définitifs à construire.

Le projet ISDOS, dirigé par D. TEICHROEW (TEI 72), a pour ambition de fournir un langage de spécification de problèmes (PSL). Une extension de ce projet est à l'étude pour tenir compte des phénomènes de dynamique, dans l'esprit de ce que nous proposons dans notre travail (BOD 78).

Les constructeurs et sociétés de service sont à la recherche de moyens de plus en plus rigoureux pour définir leurs systèmes avant de les réaliser : on peut citer par exemple DB/DC (PEL 78).

Le but du projet REMORA, lancé par C. ROLLAND (ROL 74 - 79), est de donner un modèle, une méthode, un langage et des outils pour concevoir un système d'information. Le modèle de base est de type relationnel (COD 70, CHE 76) et il a été étendu, pour inclure des notions de dynamique, en un schéma conceptuel de système d'information (FOU 78).

D'autres travaux français ont des préoccupations voisines : on peut citer ceux de H. TARDIEU (TAR 79) et les projets universitaires SCAPFACE (LUG 75), GALION (FLO 77), LAPAGE (CAV 79) . . .

- les modèles de réalisation ressortent souvent moins bien des études qui ont été faites sur les systèmes d'information. Cependant, pratiquement toutes celles que nous venons de mentionner ont aussi comme but de réaliser des systèmes conformes aux spécifications données à la conception.

Le projet ISDOS se propose de générer automatiquement des programmes, grâce à un analyseur (PSA) de spécifications données en PSL.

C'est le cas aussi dans le projet REMORA, où la réalisation et l'implantation des données et programmes est guidée par un automate conversationnel.

Le projet LAPAGE propose un "menu" de méthodes utilisables pour développer un système.

- Les modèles d'utilisation sont les moins développés. Ils doivent guider les usagers "finals" dans leur accès aux systèmes d'information, grâce à des procédures interactives clairement définies.

Les modèles d'utilisation globale sont à rapprocher des modèles d'analyse des systèmes d'information. En effet, l'étude des systèmes fonctionnant dans leur environnement doit aider à décrire, évaluer et optimiser les modes d'exploitation.

Comme modèles, on ne peut guère citer à l'heure actuelle que :
. des automates d'états finis, qui guident dans l'utilisation interactive de systèmes conversationnels,
. des graphes de circulation d'information, comme ceux dont nous allons parler en (4.1) pour la description de systèmes existants.

Nous mettons maintenant l'accent sur les modèles pouvant contribuer à une analyse de système d'information.

4 - MODELES ORIENTES vers l'ANALYSE des SYSTEMES d'INFORMATION

Ils cherchent à décrire, et éventuellement évaluer, des systèmes réels en situation de fonctionnement.

4.1 Méthodes et outils traditionnels

Depuis très longtemps, et bien avant l'apparition de l'informatique, les personnes qui avaient la responsabilité d'organiser la circulation de l'information dans les organisations utilisaient des modèles graphiques, parmi lesquels on peut citer :

a) des diagrammes précisant la structure de l'organisation et mettant en évidence des relations hiérarchiques, de conseil ou fonctionnelles (cf. par exemple O. GELINIER (GELI 69)).

b) des schémas décrivant des structures de groupes de travail, sur des idées de participation, de délégation, de direction par objectifs, de cogestion, d'autogestion, ... (GELI 69).

En effet, de la structure de l'entreprise, notamment de celle du système décisionnel, dépend, en grande partie, la forme du système d'information, sorte de "système d'irrigation".

c) des graphes de circulation d'information, mettant en évidence des postes de travail, où s'effectuent des opérations de traitement d'information, et des transmissions de données entre ces postes (REI 71). Ces graphes ont beaucoup intéressé les organisateurs qui ont tenté de les codifier, en fixant des symboles pour chaque type d'opérations, manuelles ou automatiques, telles que : tri, classement, interclassement, ventilation, etc ...

On peut citer, par exemple, la normalisation du SCOM (SCO 73), ou celle de BARTANABE - GRÜN (voir par exemple les travaux d' H. HABRIAS sur les schémas d'analyse (HAB 77)). La multiplicité des symboles rend très précise la codification, mais alourdit beaucoup des schémas qui prennent des allures de "labyrinthes avec obstacles".

Avec l'apparition de l'informatique, les circuits administratifs s'allègent. On ressent moins le besoin d'une codification étroite et tatillonne du travail administratif, ce qui a pour conséquence d'alléger les outils. Les symboles sont limités à quelques-uns et l'on donne une vue plus globale des systèmes d'information. On trouve de tels outils dans la plupart des méthodes d'analyse commerciales, sous une forme sensiblement analogue (COU 73).

A titre d'exemple, examinons la proposition de la méthode CORIG (MAL 71) de décrire la circulation de l'information par des "procédures", définies comme des réponses de l'organisation à des événements. Une procédure est représentée par un schéma qui met en évidence des "interventions" (rectangles), accomplies par différents "agents d'activité" (colonnes) ; elles sont liées par un cheminement d'information (flèches), ou par l'intermédiaire de "documentations" manuelles (boîtes) ou automatiques (cercles).

Exemple 2.15 Tenue de stock

Nous présentons ici un exemple, très schématique et partiel, de système d'information pour tenir un stock sur ordinateur.

Cinq agents d'activité sont en présence : un service de comptabilité, un service de perforation, 2 magasins, un service ordinateur. Le système est partie en temps différé (communication comptabilité-ordinateur), partie en temps réel (communication ordinateur-magasins, qui disposent chacun d'une console).

Les procédures prises en compte dans ce système sont les suivantes :

- *pr1 : à l'événement "création, modification ou suppression d'un produit", on déclenche la procédure de mise à jour de la documentation "produits", en différé.*
- *pr2 : à l'événement "modification de la quantité d'un produit" dans le magasin 1, on déclenche la procédure de mise à jour de la documentation "produits", en temps réel.*
- *pr3 : comme pr2, mais avec magasin 2.*

Remarque : dans le cas où, dans pr2 ou pr3, une chute de stock en dessous du seuil critique ou une rupture de stock a été décelée, on note cet incident dans la documentation "incidents".

- *pr4 : à l'événement "fin de journée", on liste les incidents, on les envoie à la comptabilité, qui les examine et établit des bons de commandes aux fournisseurs.*

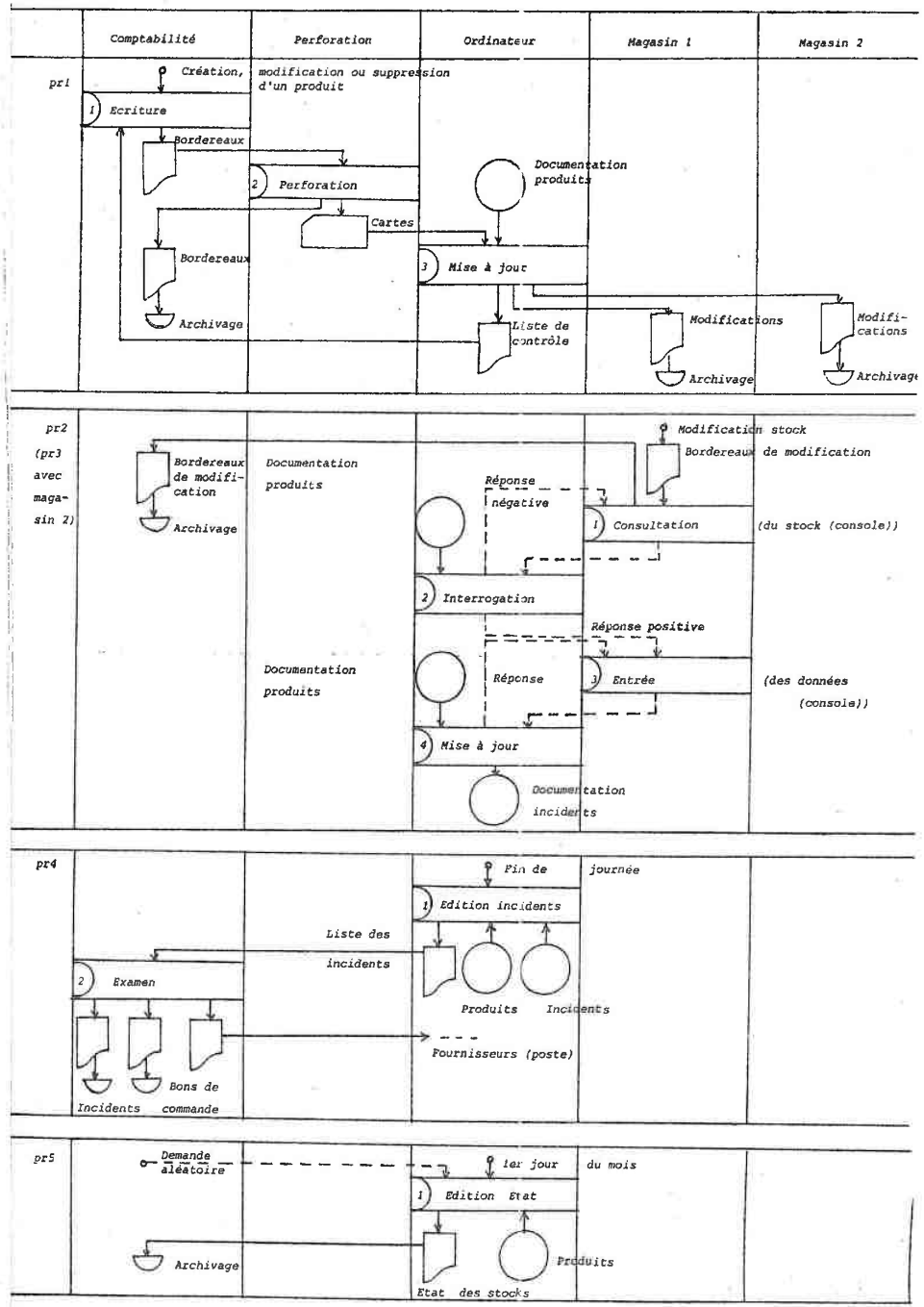


Figure 2.1 - Schémas de circulation pour les procédures pr1, pr2, pr3, pr4, pr5

- pr5 : à l'événement "1er jour du mois" ou à la demande, l'ordinateur édite un état du stock pour la comptabilité.

□

Nous ne discutons pas le bien-fondé de ces procédures, mais simplement la forme de leur présentation.

Ce type de schémas est en général accompagné de fiches précisant, pour chaque intervention (rectangle), la liste des opérations à effectuer par l'agent d'activité concerné. Cette liste, qui doit donner toutes les conditions et caractéristiques des traitements d'information, est rédigée en langue naturelle : elle comporte presque nécessairement des imprécisions ou des ambigüités. Nous ne donnerons pas celle qui se rapporterait à l'exemple ci-dessus et dont la lecture serait fastidieuse.

De tels schémas de procédures ont le mérite de donner une vue globale du système d'information, de mettre en évidence des interlocuteurs (agents d'activité), des taches à accomplir (interventions), des informations permanentes (documentations) ou temporaires, ainsi que les liaisons entre les taches. Ils ont aussi l'avantage de faire réfléchir aux conditions de déclenchement des procédures et à l'architecture du système.

Cependant, même si les commentaires qui les accompagnent sont très fournis, ils laissent parfois planer un certain nombre de doutes. Ainsi, dans l'exemple précédent, on peut faire les remarques suivantes.

- On ne voit pas bien si l'on traite des lots d'information ou des informations individuelles. Par exemple, on peut se demander si la procédure pr1 est effectivement déclenchée pour chaque création, modification ou suppression d'un produit, ou si l'on doit attendre d'avoir un lot de telles transactions. De même, il n'est pas clair que la transmission des bordereaux à la perforation doive se faire par lots, ni même la transmission des cartes à l'ordinateur... Dans le cas où des lots sont constitués, il importe aussi de savoir quelle est leur taille.

- Si le travail se fait par lots, on ne voit pas si l'on doit maintenir la séparation en lots distincts quand se constitue une file d'attente, ou au contraire n'en faire plus qu'un seul.

- On ne sait pas toujours quand déclencher une tâche donnée, cette question étant d'ailleurs liée aux précédentes. Par exemple, dans prl, le cycle de mise à jour des produits par l'ordinateur reste à préciser : est-il journalier, hebdomadaire ... ? D'une façon générale, la périodicité ne doit-elle pas varier avec la charge et, dans ce cas, comment l'apprécier ?

Il est évident que l'on ne peut répondre à de telles questions que si l'on a des éléments statistiques sur la charge en données et ses variations temporelles. De même, que signifie au juste le terme "fin de mois", qui manque de précision ?

- Les différents agents d'activité travaillent en parallèle, mais on ne sait pas trop dans quelles conditions. Ainsi, la perforation des cartes peut-elle se faire en même temps que l'écriture des bordereaux ? Ceci est d'ailleurs lié aux remarques précédentes.

- Quand le même agent d'activité peut choisir entre plusieurs interventions à un moment donné, quelle est la règle qui permet de dire si elles se dérouleront effectivement en parallèle, ou si elles devront se succéder, et dans quel ordre ? Ainsi, parfois, la comptabilité devra se partager ou choisir entre l'écriture des bordereaux (prl) et l'examen des incidents (pr4).

Les schémas de circulation donnent donc, à un certain niveau, une vue statique du système d'information, toutes les imprécisions que nous avons mentionnées concernant plutôt sa dynamique. Si l'on désire définir complètement le système, tous ces petits mystères doivent être levés dans les commentaires informels qui accompagnent les schémas. Gageons que ce ne sera pas chose très facile et qu'il subsistera bien, çà et là, quelques incertitudes.

Aussi serait-il plus sûr d'avoir un guide pour exprimer cette dynamique, et même, encore plus judicieusement, de la prendre en compte dans une description formelle du système d'information, dont elle est un élément très important.

On pense alors au modèle de FORRESTER, dont l'ambition est de décrire la dynamique de toute une organisation. Nous rappellerons brièvement ses principes dans le prochain paragraphe et nous dirons quelle peut être son utilité dans notre problématique.

d) des organigrammes. Cet outil traditionnel, longtemps utilisé en informatique pour décrire des algorithmes, a fait l'objet d'études approfondies qui ont mis en évidence ses avantages et ses inconvénients. Ce domaine est en fait celui des schémas de programmes (GRE 75, LED 75...) sur lequel on trouve une synthèse dans (LIV 78).

Son usage disparaît pour l'analyse des problèmes au profit d'autres méthodes plus sûres de construction et de formulation d'algorithmes (cf. les méthodes de programmation structurée (DDH 72), comme la méthode déductive (PAI 79)).

Mais il subsiste comme outil de représentation de chaînes de programmes, sous une forme un peu différente. Il permet de mettre en évidence, de façon très globale, des informations et des traitements.

En analyse traditionnelle, on distingue deux niveaux de représentation (cf. PAM 69, REI 71) :

- des organigrammes "logiques" aidant à visualiser des "fichiers logiques" et des "unités logiques de traitement", c'est-à-dire des entités où n'interviennent pas de contraintes matérielles ;

- des organigrammes "organiques" présentant des "fichiers physiques" et des "programmes", c'est-à-dire des entités redécoupées en fonction de contraintes matérielles.

Ces schémas sont surtout utilisés pour représenter de manière statique et globale la partie automatique d'un système d'information.

4.2 Le modèle de J.W. FORRESTER (FOR 61, FOR 68)

a) Présentation générale

C'est un modèle mathématique, qui a été utilisé pour des études de dynamique industrielle ou économique, dont on connaît le retentissement (rapports MEADOWS (MEA 72) sur la croissance).

Nous ne donnerons ici que ses traits essentiels, quand il est employé pour décrire une organisation. Il favorise alors la mise en équations des différents "flux" qui la traversent : matériaux, argent, personnel, équipement, ordres, information.

Un tel "fluide" naît d'une "source" et disparaît dans un "puits". Entretemps, il a pu traverser des "réservoirs" dont le débit de sortie est modulé grâce à des variables d'état du système, selon certaines règles précisées à l'avance.

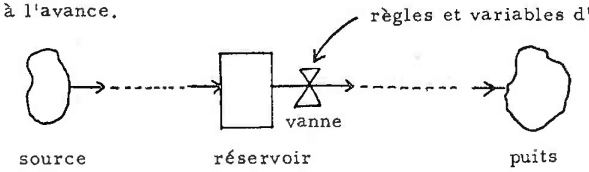


Figure 2.2 Modèle de FORRESTER - Schéma de principe

L'organisation est donc vue comme un réseau hydraulique, avec différents fluides dont les écoulements continus interagissent.

Les réservoirs symbolisent des traitements qui retiennent les particules plus ou moins longuement : leur durée est fonction de "délais" d'ordre n ($n \geq 1$).

Les équations du modèle sont discrétisées par rapport au temps. Ayant fixé la loi d'écoulement des fluides aux sources, il est possible de calculer les états successifs du système, en faisant progresser le temps pas à pas, selon une méthode de simulation continue (chapitre 4).

On admet dans ce modèle un certain degré d'indéterminisme : les flux d'entrée et les délais peuvent être des processus aléatoires.

b) Analyse des délais

Nous approfondissons ici les délais, qui confèrent au système sa dynamique. Dans la suite, on notera pour un réservoir (figure 2.3), et pour $t \in \mathbb{R}_+$:

a (t) : flux entrant dans le réservoir à l'instant t

b (t) : flux sortant du réservoir à l'instant t

a et b sont des fonctions ou, d'une manière plus générale, des distributions (au sens de L. SCHWARTZ, par exemple dans (SCH 65)),

v (t) : le volume contenu dans le réservoir à l'instant t

k : une constante de délai

A : la transformée de LAPLACE (SCH 65) de la distribution a

$$A(s) = \int_0^{+\infty} a(t)\, e^{-st}\, dt$$

pour $s \in \mathbb{C}$, avec $\mathrm{Re}(s) \geq s_0 \in \mathbb{R}$, tel que $\int_0^{+\infty} |a(t)|\, e^{-s_0 t}\, dt < +\infty$

B : la transformée de LAPLACE de la distribution b :

$$B(s) = \int_0^{+\infty} b(t)\, e^{-st}\, dt$$

pour $s \in \mathbb{C}$, avec $\mathrm{Re}(s) \geq s_0 \in \mathbb{R}$, tel que $\int_0^{+\infty} |b(t)|\, e^{-s_0 t}\, dt < +\infty$

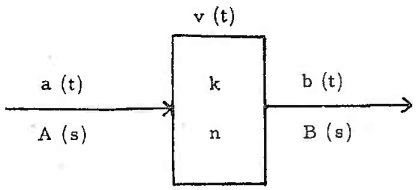


Figure 2.3 - Notations pour un réservoir

On suppose ici que a et b sont des processus déterministes et que $b(t) = 0$ pour $t < 0$.

- Délais d'ordre 1

Un délai d'ordre 1 définit un débit de sortie par des conditions aux limites et la formule :

$$b(t) = \frac{v(t)}{k}$$

En remarquant que :

$$\frac{dv}{dt}(t) = a(t) - b(t)$$

où $\frac{dv}{dt}$ est la fonction ou distribution dérivée de v, on en déduit l'équation différentielle :

$$k \frac{db}{dt} + b = a$$

où $\frac{db}{dt}$ est la fonction ou distribution dérivée de b.

En utilisant la transformée de LAPLACE, pour $Re(s) \geq s_0$, et compte tenu du fait que $b(o-) = 0$:

$$ks\,B(s) + B(s) = A(s)$$

$$B(s) = \frac{A(s)}{ks+1}$$

En prenant la transformée de LAPLACE inverse, quand c'est possible, on exprime b (t) en fonction de a (t).

On ne peut pas exprimer la sémantique des données, ni celle de leurs transformations, et tout cheminement d'information est figé : on n'admet pas qu'à la sortie d'un réservoir il y ait plusieurs flux dirigés vers différentes destinations.

On a ici une vue très macroscopique qui n'autorise pas la prise en compte de ressources ayant une influence sur la dynamique par des blocages et des synchronisations.

Celle-ci s'exprime d'ailleurs très globalement par des délais de différents types, qu'il semble difficile d'utiliser toujours dans le cadre du traitement de l'information. L'hypothèse de la proportionnalité du flux de sortie d'un réservoir au niveau de ce réservoir nous paraît souvent ne pas pouvoir être retenue : comment imaginer, par exemple, qu'une machine travaille d'autant plus vite que le volume des données à traiter est plus élevé ?

Ce modèle s'apparente à une approximation "fluidique" ou par "diffusion" (KLE 76) d'un système de files d'attente, bien que, dans ces dernières, le rythme de travail des serveurs soit indépendant du débit d'entrée.

Orienté vers des études plutôt macroscopiques de dynamique industrielle ou économique, il ne semble pas répondre à nos objectifs de modélisation des systèmes d'information (chapitre 3).

4.3 Autres modèles

D'autres modèles ont vu le jour ces dernières années pour décrire ou évaluer des systèmes d'information en fonctionnement.

Ainsi A. CHECROUN dans (CHE 76) propose de décrire des systèmes administratifs, dans leurs aspects automatisés ou non, par des

modèles dynamiques. Ceux-ci seraient ensuite utilisés dans des simulations destinées à valider la structure des modèles et tester différentes options de réorganisation. On retrouvera ces objectifs parmi les nôtres, mais, la méthode d'approche, les concepts et la formulation adoptés sont différents. Ce travail semble d'ailleurs s'être orienté dernièrement vers la description de systèmes, en laissant pour l'instant de côté la simulation.

L'approche SIG, proposée par R. HURTUBISE dans (HUR 76), combine à la fois l'étude détaillée des informations et des traitements, dans des réseaux de communication, et l'analyse de performances. Les procédures administratives sont décrites dans un "cahier des réseaux". Une simulation du système d'information de type PERT est entreprise, afin de déterminer des chemins critiques et d'optimiser les circuits. C'est assurément aussi un des objectifs que nous allons présenter.

Chapitre 3 - OBJECTIFS et HYPOTHESES de TRAVAIL

"Tamino :

Zu Hilfe ! Zu hilfe ! Sonst bin ich verloren"

(W.A. Mozart et E. Schikaneder, La Flûte Enchantée, acte 1, scène 7).

"Tamino (face au dragon) :

A l'aide ! A l'aide ! Sinon je suis perdu"

Chapitre 3 - OBJECTIFS et HYPOTHESES de TRAVAIL

1. OBJECTIFS de ce TRAVAIL

1.1 Description de systèmes.

1.2 Evaluation fonctionnelle qualitative.

1.3 Evaluation quantitative du comportement dynamique.

1.4 Prédiction.

2. HYPOTHESES de REDUCTION

2.1 Le système d'information logique.

2.1.1 Réduction sémantique des informations.

2.1.2 Limitation pour les fonctions et relations.

2.1.3 Limitation pour les formules.

2.1.4 Réduction quantitative des informations.

2.1.5 Réduction sémantique des processus.

2.1.6 Réduction du nombre des processus.

2.2 Le système d'information physique.

2.2.1 Les mémoires ; implantation physique des données et des processus.

2.2.2 Simplification concernant le partage des ressources.

2.2.3 Simplification concernant les processeurs.

2.2.4 Simplification concernant les ressources passives.

Dans les chapitres précédents, nous avons :

- montré la complexité des systèmes d'information en faisant ressortir, sous différents éclairages, ce qu'il nous semble possible d'exprimer de manière précise en ce domaine et, au contraire, tout ce qui reste assez flou, notamment au sujet du non automatique (et pour cause !) ;

- dégagé l'apport des tentatives de modélisation globale de systèmes, qui ont porté, jusqu'à présent, surtout sur la spécification et la conception de systèmes d'information automatique, dans une optique relativement statique ;

- montré que peu est fait en ce qui concerne l'évaluation du fonctionnement logique et du comportement dynamique de ces systèmes.

Dans ce chapitre, nous précisons d'abord quels sont les objectifs qui nous poussent à utiliser des modèles réduits (ou maquettes) pour étudier les systèmes d'information, sous leurs différents aspects (1). Nous développons ensuite (2) les hypothèses de travail que nous avons retenues pour proposer un cadre conceptuel, des outils, un mode d'emploi, qui feront l'objet des chapitres suivants.

1 - OBJECTIFS de ce TRAVAIL

En reprenant la classification des objectifs du chapitre précédent (1.4), nous précisons ceux que nous avons assignés à notre travail de modélisation de systèmes d'information en fonctionnement.

1.1 Description de systèmes

L'écriture d'un modèle réduit, telle que nous l'envisageons, est une description formelle de système d'information : elle utilise un

langage précis, et même, un langage de programmation. D'où le nom de maquette logicielle, ou plus simplement maquette, donné à un tel modèle.

On y appréhende les aspects statiques, mais aussi dynamiques d'un système : on donne la possibilité de décrire des modifications ordonnancées dans le temps, en tenant compte de phénomènes de parallélisme et de synchronisation.

Une maquette reflète la partie automatique, comme la partie non automatique (administrative) d'un système, c'est-à-dire en est une description globale.

Elle reproduit les aspects logiques d'un système par certains côtés (description de données et processus), et physiques par d'autres (utilisation de ressources concrètes).

On peut décrire un système par une maquette avec un niveau de détail plus ou moins fin, affectant aussi bien les données, les processus, que les ressources du système.

La description est modulaire : elle se fait par assemblage de composants standard diversifiés et de composants propres au système décrit. Il serait d'ailleurs bon que le langage utilisé autorise une extension facile de composants déjà réalisés.

Ainsi, la conception et l'écriture de maquettes logicielles aident à décrire un système d'information :

- existant : pour avoir sur lui une documentation claire, mieux l'apprécier, le comparer à d'autres, et aider à le faire évoluer ;
- futur : pour faciliter son évaluation, guider ensuite sa conception détaillée, sa réalisation et son exploitation.

La description ainsi obtenue sous la forme d'un programme est soumise à un compilateur qui en vérifie la correction formelle.

1.2 Evaluation fonctionnelle qualitative

- Une maquette de système d'information aide à mettre en évidence des vices de fonctionnement logique de ce système.

Elle est exploitée, après avoir été convenablement paramétrée, de manière que tous les cas y soient envisagés.

On peut ainsi détecter des écarts par rapport aux spécifications données, ainsi que des blocages imprévus.

De tels essais sont envisageables à un quelconque niveau de détail dans la description.

- Lors de l'étude préalable d'un système, il est intéressant de construire une maquette et de la faire fonctionner. On montre ainsi la variété des informations saisies, traitées, restituées, à de futurs utilisateurs, afin de les amener, avec les organisateurs et informaticiens, à un consensus sur les caractéristiques exactes du système à réaliser.

1.3 Evaluation quantitative de comportement dynamique

Une maquette dynamique telle que nous l'imaginons permet d'étudier les performances d'un système, puisqu'on y décrit l'évolution du temps et l'utilisation de ressources physiques.

Une telle évaluation de type quantitatif est opérée grâce à des mesures faites sur certains composants de la maquette en fonctionnement, grâce à une technique de simulation.

Ces mesures sont ensuite analysées statistiquement pour en déduire des caractéristiques de comportement.

On se place ici dans des conditions "normales" (ou supposées telles) du fonctionnement du système réel, c'est-à-dire avec des taux d'entrées

et des allures de travail des processeurs assez proches de la réalité. Plusieurs types d'analyses sont envisageables :

- on peut s'intéresser aux conditions d'obtention d'un état d'équilibre du système. On vérifie notamment que des volumes de données ne croissent pas démesurément au cours du temps, ou, en d'autres termes, qu'il n'y a pas de "goulet d'étranglement". Dans le cas contraire, un tel modèle aide à en détecter la cause et à tester des modifications du système visant à supprimer les anomalies.

- On peut chercher à étudier le comportement d'un système en régime transitoire, c'est-à-dire en dehors d'un état d'équilibre, notamment pour mieux "dimensionner" certains composants, ou pour savoir quand se produit un événement précis. Par exemple, on suit l'évolution dans le temps de certaines données afin de prévoir un espace de rangement optimum, ou d'accroître momentanément les moyens (personnes, machines) affectés à un travail.

- En régime permanent, il est possible d'obtenir des valeurs moyennes, caractéristiques du comportement du système à l'équilibre, avec des intervalles de confiance :

. volume moyen de données
. taux moyen d'occupation de ressources
. temps de réponse moyen.

Dans certains cas, où l'on estime que le modèle est très proche de la réalité, on accorde aux résultats obtenus une valeur absolue. Ceci aide, entre autres, à chiffrer assez précisément le coût d'exploitation du système : en ce sens, la construction et l'usage de telles maquettes guident les sessions d'audit informatique ou les études préalables.

Dans d'autres cas, où la confiance en les données recueillies pour alimenter le modèle est plus faible, on n'accorde aux résultats obtenus

qu'une valeur relative. Ces résultats sont alors appréciés par comparaison à d'autres, obtenus dans les mêmes conditions, sur des modèles de structures ou de paramètres différents. Une telle manière de procéder est un guide pour choisir entre plusieurs configurations possibles d'un système, lorsqu'on le remet en cause lors d'une étude préalable ou après détection d'un vice de fonctionnement.

- Enfin, une démarche analogue peut aider à mettre en évidence le rôle joué par tel ou tel facteur dans l'explication d'un phénomène. Par exemple, on peut montrer ainsi l'influence, dans la formation d'une file d'attente ou l'allongement d'un temps de réponse, de la manière d'ordonnancer les travaux dans un service administratif.

1.4 Prédiction

En s'écartant du fonctionnement "normal" du système, on peut étudier ce qui se passerait s'il se trouvait confronté à telle ou telle situation hypothétique.

Ceci suppose un modèle robuste capable de fournir des renseignements au-delà de son domaine de validation.

Plusieurs types d'expériences peuvent être faites selon les hypothèses envisagées : modification des paramètres de l'environnement, des paramètres internes, ou de la structure du modèle.

Ces modifications sont envisagées statiquement, avant l'utilisation de la maquette en simulation, ou dynamiquement, c'est-à-dire pendant l'exécution de la maquette.

Les techniques utilisées pour ces études doivent être les mêmes que celles dont on se sert lors d'évaluations quantitatives, dans une situation proche du fonctionnement habituel du système, en régime permanent ou transitoire.

La modélisation que nous envisageons poursuit donc plusieurs buts différents.

Il est exclu qu'une maquette unique puisse prétendre à elle-seule les satisfaire tous : il faut sans doute, pour chaque catégorie et niveau d'objectifs, fabriquer une maquette adaptée, sur laquelle on fait une série d'expériences précises.

Est-ce à dire que les différentes maquettes n'ont aucun lien, par exemple de filiation, entre elles ? Sans doute pas ... En fait, notre ambition est d'avoir un langage unique pour décrire les maquettes des diverses catégories, mais aussi de proposer des outils diversifiés adaptés aux différentes classes d'objectifs à satisfaire. Elle est aussi de montrer comment construire et exploiter au mieux ces maquettes.

A terme, ces réflexions peuvent peut-être déboucher sur une méthode de conception progressive de systèmes d'information fondée sur l'idée de maquette, c'est-à-dire de modèle réduit programmé opérationnel.

2 - HYPOTHESES de REDUCTION

Une maquette est un modèle, c'est-à-dire une image en réduction d'un système d'information. Avant d'étudier en détail l'architecture et les composants d'une maquette, il nous semble important de préciser nos hypothèses de réduction.

Nous examinons successivement celles concernant ce que nous avons appelé système d'information logique et physique, sous deux aspects : sémantique et quantitatif.

2.1 Le système d'information logique

2.1.1 Réduction sémantique des informations

Quand elles sont formalisées, la réduction sémantique[1] des informations porte d'abord sur la prise en compte exclusive de certains types d'informations, symboles fonctionnels et relationnels, parmi ceux que l'on peut mettre en évidence dans le système réel. Il faut en mesurer les conséquences sur le système formel et, lorqu'on se place dans une interprétation, sur les fonctions et relations : on a ici en quelque sorte une projection du système sur un sous-espace.

Quand elles ne sont pas formalisées, les informations traitées réellement "à la main" sont tout de même, dans une maquette, représentées de manière réduite selon les mêmes principes. Ce que nous disons s'applique aussi à elles.

Nous avons déjà dit que nous confondons niveau formel et interprétatif, en nous plaçant dans une interprétation unique. A nouveau, les mêmes symboles servent aux deux usages.

a) Fonctions

Etant donné un symbole fonctionnel f de profil (i, j), $i \in [m]^*$, $j \in [m]$, plusieurs cas peuvent se présenter :

- le type j n'est pas retenu dans le modèle : dans ce cas, le symbole fonctionnel f en est supprimé aussi ;

- le type j est retenu dans le modèle, mais certains types constitutifs de i ne le sont pas : dans ce cas, f est supprimé et non remplacé, ou remplacé par une fonction indéterministe[2], notée aussi f, de profil $(\bar{i}, j)$ où $\bar{i}$ est la restriction de i aux types conservés dans le modèle ;

---

(1) On pourrait à ce propos relancer la vieille querelle de la limite entre sémantique et syntaxe. Puisque les transformations que nous envisageons portent sur les mots d'un langage formel, on peut tout aussi bien parler de réduction syntaxique.

(2) Il s'agit ici d'une fonction probabiliste ou, si l'on préfère, d'une variable aléatoire.

- le type j et les types de i sont retenus : f peut être conservée ou supprimée.

b) Relations

Etant donné un symbole relationnel r de profil i ∈ [m]$^+$:

- si certains types de i seulement sont retenus dans le modèle, r est supprimé et non remplacé, ou remplacé par sa projection, notée aussi r, sur $\bar{i}$, restriction de i aux types conservés ;

- si tous les types de i sont retenus, r peut être conservé ou supprimé.

c) Schémas fonctionnels et relationnels

Les schémas où apparaissent des symboles fonctionnels ou relationnels supprimés, et éventuellement remplacés, subissent des modifications d'une nature analogue : suppression, et remplacement éventuel avec introduction possible d'indéterminisme.

d) Formules

- Les formules atomiques dans lesquelles un schéma fonctionnel ou relationnel est supprimé sont elles-mêmes supprimées, les autres étant conservées, et éventuellement modifiées par modification des schémas.

- Les formules non atomiques dans lesquelles interviennent des formules atomiques supprimées sont simplifiées ; celles où interviennent des formules atomiques modifiées sont elles-mêmes modifiées en conséquence.

- La question de la répercussion des simplifications sur les axiomes du système est très délicate. En effet, des suppressions de types, ou de symboles fonctionnels ou relationnels, peuvent aboutir à l'incomplétude de l'information, ce qui peut être désastreux. Ce problème, lié aux notions d'extension et de restriction introduites par J. L. REMY dans

(REM 74), doit sans doute être examiné cas par cas. Pour certains d'entre eux, un remède est d'introduire de nouveaux types ou symboles.

e) Introduction de nouveaux types ou symboles permanents

La suppression de certains types risque de faire perdre la possibilité d'accéder à une information intéressante dont on peut avoir besoin dans le modèle réduit. D'où la nécessité d'introduire parfois de nouveaux types et de nouveaux symboles fonctionnels ou relationnels permanents dans la maquette (voir exemple 3.1), qui synthétisent en quelque sorte l'information qui a été perdue. Cette manière de faire est utile aussi pour compléter une information que la réduction a rendue incomplète.

Exemple 3.1

En reprenant l'exemple 1.2 (ch. 1, 2.1.d), on peut arbitrairement réduire le système formel proposé pour un site.

Ainsi, si l'on supprime les types ligne et produit, on est conduit, par exemple :

- à supprimer les symboles fonctionnels : p, nℓ, q, c, s, tℓ, np, st, car, ℓ ;

- à supprimer les symboles relationnels : rℓ , rp ;

- à supprimer le symbole relationnel qté, car sa projection sur (client, $\mathbb{R}_+$) n'a plus aucun sens ;

- à introduire un nouveau symbole fonctionnel nlignes de profil (commande, $\mathbb{N}$), pour conserver l'indication du nombre de lignes figurant dans une commande.

Les autres éléments restent inchangés (figure 3.1).

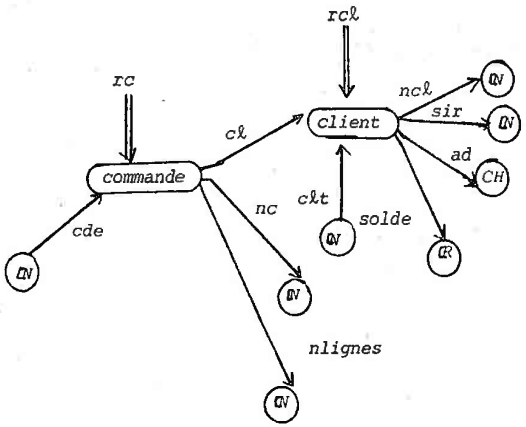


Figure 3.1 - Réduction des informations

Nous ne développons pas la réduction des formules, notamment des axiomes, qui nous semble ici particulièrement simple. □

2.1.2 Limitation pour les fonctions et relations

Dans les outils que nous proposons aux utilisateurs, nous avons implicitement certaines limitations quant à l'usage de relations et fonctions définies en extension. Il nous semble utile de discuter ici ces restrictions, en montrant comment il est possible de s'en accomoder.

Définir une relation n-aire en extension consiste à donner tous les n-uples de cette relation. De même, définir en extension une fonction d'arité n consiste à se donner tous les (n + 1)-uples du graphe de cette fonction.

Par la suite, nous n'autorisons la définition en extension que de relations 1-aires (unaires). De plus, la définition d'une fonction n-aire en extension passe nécessairement par une telle relation.

La question est donc de savoir se ramener toujours à des relations unaires et de savoir fabriquer des fonctions n-aires. Nous le faisons systématiquement en introduisant de nouveaux types pour les n-uples :

Etant donnés des types $T_1, \ldots, T_n$ intervenant dans une fonction ou une relation n-aire, on introduit un nouveau type T et des fonctions $f_1, \ldots, f_n$, telles que $f_k : T \to T_k$, pour k = 1, ..., n, avec :

$$((f_1 . x \equiv f_1 . y \wedge \ldots \wedge f_n . x \equiv f_n . y) \supset x \equiv y)$$

a) Relations n-aires

Soit un symbole relationnel r de profil $i = (i_1, \ldots, i_n)$.

On introduit un nouveau type T, un symbole relationnel unaire r' de profil T et n symboles fonctionnels unaires f_k, k = 1, ..., n, respectivement de profil (T, i_k), k = 1, ..., n.

Remplacer le symbole r par ces symboles unaires revient à introduire l'axiome :

$$(r . x_1 \ldots x_n \leftrightarrow \exists x (r' . x \wedge f_1 . x \equiv x_1 \wedge \ldots \wedge f_n . x \equiv x_n)).$$

Ceci montre que l'on supprime l'exigence de symboles n-aires par l'introduction de nouveaux types.

b) Fonctions n-aires

Soit un symbole fonctionnel f de profil (i, j), $i = (i_1, \ldots, i_n)$.

On introduit un nouveau type T, une relation unaire r de profil T, n symboles fonctionnels unaires f_k, k = 1, ..., n, respectivement de profil (T, i_k), k = 1, ..., n, et un symbole fonctionnel unaire f' de profil (T, j).

On se donne de plus l'axiome :

$$f . x_1 \ldots x_n \equiv y \leftrightarrow \exists x (r . x \wedge f_1 . x \equiv x_1 \ldots \wedge f_n . x \equiv x_n \wedge f' . x \equiv y).$$

On voit que la relation r donne en fait le domaine de définition de f (ou f').

Exemple 3.2

Dans l'exemple 1.2 (ch. 1, 2.1.d), on peut tenter de "supprimer" le symbole relationnel 3-aire qté.

On introduit un nouveau type vente, un symbole relationnel qtév unaire de profil vente, et 3 symboles fonctionnels pro, di, qv de profils respectifs (vente, produit), (vente, client), (vente, $\mathbb{R}_+$), avec l'axiome :

$$(qté\ .\ x_1\ x_2\ x_3 \leftrightarrow \exists\ x\ (qtév.x \wedge pro.x \equiv x_1 \wedge clé\ .\ x \equiv x_2 \wedge qv.x \equiv x_3))$$

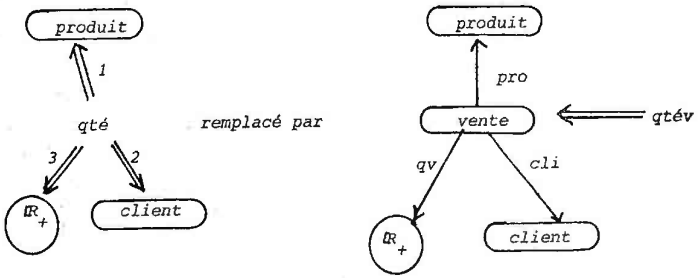


Figure 3.2 - Remplacement d'un symbole n-aire

Ainsi, dans l'optique de cette simplification, les relations (unaires) définies en extension ne servent qu'à délimiter les domaines de définition des fonctions. □

2.1.3 Limitation pour les formules

Nous avons souligné dans le chapitre 1 l'utilisation du calcul des prédicats comme base d'un langage de formules et d'interrogation d'informations.

En réalité, on se limite le plus souvent à quelques constructions syntaxiques permettant une implantation physique relativement aisée : c'est le cas notamment du langage ALPHA de CODD (COD 72) ou de son équivalent algébrique, avec les opérations de projection, jonction, division..., pour l'interrogation des bases de données relationnelles.

Pour notre part, nous limitons aussi les primitives, dans des outils de description systématique (chapitre 6).

2.1.4 Réduction quantitative des informations

Bien que la réduction sémantique puisse diminuer considérablement le volume d'un système d'information, il est parfois nécessaire de procéder à une réduction quantitative supplémentaire. Celle-ci consiste, au niveau formel, à diminuer le nombre des axiomes qui servent à définir, au niveau interprétatif, les relations en extension, c'est-à-dire les n-uples (ou 1-uples, voir (2.1.2)). En effet, il est hors de question de vouloir, dans un modèle réduit, manipuler la masse considérable de données que l'on trouve généralement dans un système réel, quel que soit l'objectif que l'on poursuit.

Cette réduction quantitative a un certain nombre de conséquences, notamment lors de l'appréhension des durées des traitements, pour lesquels il faut tenir compte de leur complexité vis à vis du volume des données manipulées.

Ce que nous disons ici s'applique aussi à la représentation d'informations non formalisées dans le système réel.

Exemple 3.3

Si, dans un système de gestion de commandes, on traite, n commandes pendant une durée absolue f (n) et que l'on décide de n'en traiter que n' = n/10 dans la maquette, alors, pour conserver au système un comportement dynamique correct, il faut considérer que le traitement de n' commandes se fait en un temps absolu f (10 x n'). □

2.1.5 Réduction sémantique des processus

La réduction sémantique des informations d'un système

dans une maquette induit naturellement celle des modifications élémentaires (permanentes ou temporaires) et celle des processus automatiques.

Comme dans le cas des fonctions d'accès, certaines modifications sont supprimées, d'autres sont transformées, avec introduction éventuelle d'indéterminisme.

Mais, en modélisant un système, on réduit aussi son espace d'états et l'on risque d'éliminer certaines causes importantes de blocage de processus : on doit donc y être très attentif.

Les processus non automatiques de traitement d'information sont décrits dans une maquette de la même manière que les autres. Il s'agit pour eux de trouver directement un modèle de comportement adéquat, que l'on exprime avec les mêmes concepts et outils.

2.1.6 Réduction du nombre des processus

Un système d'information réel comporte de très nombreux processus qu'il est souvent difficile de reproduire tels quels dans une maquette, même réduits sémantiquement.

Certains d'entre eux n'ont d'ailleurs plus de raison d'être, d'un point de vue logique, l'information sur laquelle ils portent ayant disparu dans la maquette.

La tentation est donc forte de les supprimer purement et simplement. C'est cependant dangereux si l'on s'intéresse au comportement dynamique du système, parce que l'on supprime des consommateurs de ressources ayant une influence sur le déroulement des autres processus.

Deux attitudes sont alors possibles :

- on laisse subsister ces processus "parasites" seulement en tant qu'utilisateurs de ressources ;

- on les agrège à d'autres processus, dont on modifie le comportement dynamique en augmentant leur consommation en ressources.

Cette dernière façon de faire peut d'ailleurs être utilisée pour réduire systématiquement le nombre de processus d'une maquette : on agglomère des traitements d'information, le plus souvent devant se dérouler séquentiellement, pour obtenir des processus de taille optimale.

Il faut alors remarquer que, pour la partie automatique d'un système, il n'y a pas correspondance bijective entre les processus du système réel (instances de programmes) et ceux d'une maquette, ceux-ci apparaissant souvent comme des regroupements de ceux-là.

Pour la partie non automatique, où les traitements d'information réels sont plus flous, on tente aussi de "tailler" les processus de façon optimale, avant de les décrire formellement en réduction.

2.2 Le système d'information physique

2.2.1 Les mémoires ; implantation physique des données et des processus

Dans les maquettes que nous considérons généralement, nous nous plaçons à un niveau tel que l'organisation physique des données et l'implantation physique des processus informatiques ne sont pas directement pris en compte, parce qu'intervenant la plupart du temps à une échelle trop fine.

Tout d'abord, des données qui, en réalité, sont implantées sur des mémoires quelconques, le sont souvent dans une maquette en mémoire centrale (ou virtuelle). De plus, il peut y avoir une différence sensible entre les manières d'organiser cette implantation dans le système réel et le modèle. Dans celui-ci, on se préoccupe finalement plus de l'organisation logique des données et la dynamique est fixée par d'autres considérations que leur implantation physique ; nous les développons par la suite.

Exemple 3.4

Peu importe que l'on remplace, dans une maquette, des fichiers physiques à rangement séquentiel, indexé ou calculé, par des listes, des arbres ou des tableaux, pourvu que l'on puisse y reproduire :

- les fonctions, les relations et les modifications logiques qui subsistent de la structure de données générale ;

- un comportement dynamique fidèle, à l'échelle de la perception humaine, des processus qui les utilisent, comportement finalement relativement indépendant des structures physiques de données adoptées dans la maquette. □

De même, les processus, qui sont physiquement répartis et implantés dans l'organisation, cohabitent dans la maquette en mémoire centrale (ou virtuelle). Comme pour les données, on se préoccupe de leur logique, mais on se soucie aussi beaucoup de leur comportement dynamique, affecté par des blocages du niveau logique et par l'utilisation de ressources.

Mais, certains phénomènes liés aux implantations physiques sont souvent ignorés totalement.

Exemple 3.5

Les attentes des processus réels sur ordinateur pour défaut de page, pour entrées-sorties sur disque,... qui se produisent à des échelles de temps de l'ordre du dix-millième de seconde ne sont pas admissibles dans un modèle où d'autres événements interviennent à l'ordre de la minute, de l'heure, du jour. □

Nous avons ici parlé essentiellement de la représentation physique des données et processus formalisés. Pour les autres, l'implantation en mémoire de leurs modèles est artificielle et entièrement soumise aux choix d'implantation du compilateur du langage de programmation utilisé.

En résumé, on peut affirmer qu'il peut y avoir un divorce très net entre les choix physiques du système réel et ceux d'une maquette, sans que cela entraîne des différences notoires de sémantique des données et processus, ni, à un certain niveau au moins, de comportement dynamique (réel pour le système, simulé pour la maquette).

2.2.2 Simplifications concernant le partage des ressources

Dans nos maquettes, les ressources physiques sont toujours gérées indépendamment les unes des autres, de manière décentralisée, même si ce n'est pas toujours le cas dans la réalité.

On perd ici la possibilité d'une prévention de l'interblocage, qui est laissée aux soins de l'utilisateur. Nous verrons cependant que les contraintes imposées à celui-ci sont telles que ce phénomène a très peu de chance de de se produire.

De plus, ceci paraît exclure certains niveaux de détail dans l'analyse des systèmes d'information, comme ceux liés au fonctionnement des systèmes d'exploitation d'ordinateurs, où la gestion est souvent globale. Il ne manque pourtant pas d'exemples, dans la littérature, où des phénomènes de ce niveau assez fin ont été approchés par des modèles à gestion de ressources décentralisée, comme des réseaux de files d'attente gérés indépendamment (cf. GEL 75c).

Ce n'est d'ailleurs pas à un tel niveau que sont généralement construites nos maquettes : l'échelle de temps des événements qui les affectent est le plus souvent l'échelle "humaine", comme nous le faisions remarquer précédemment.

On est donc amené, non seulement à rassembler des ressources élémentaires en "unités d'allocation" (chapitre 1), mais encore à poursuivre cette agrégation en ressources plus conséquentes, pour des raisons de

niveau de détail et de simplicité. Ce faisant, on accroît les chances d'aboutir finalement à des ressources que l'on peut considérer comme gérées indépendamment les unes des autres, ce qui est notre hypothèse.

Exemple 3.5

Ainsi, si l'on peut voir un ordinateur comme un réseau de files d'attente, un regroupement ultime doit permettre, sans doute au prix d'une perte de précision du modèle, de le considérer comme une ressource (un processeur) unique. □

2.2.3 Simplification concernant les processeurs

Qu'ils soient humains ou matériels, les processeurs d'un système réel se modélisent par quelques types de ressources (nous dirons aussi processeurs) dont nous donnons les principaux.

a) Ressources critiques

Ce sont des ressources à un seul point d'accès ou, dans la terminologie des files d'attente, des files à un serveur :

- utilisé par des clients sans priorités et gérés selon le mode PAPS (premier arrivé - premier servi), ou avec priorités fixes pendant le service ;

- préemptible avec reprise d'un processus interrompu au point d'interruption ("résume" (KLE 76)), ou non ;

- avec calendrier, où l'on fixe des périodes pendant lesquelles le serveur est utilisable, ou non, c'est-à-dire où l'on considère le serveur comme toujours disponible.

b) Ressources à accès multiples

Ce sont des ressources avec plusieurs points d'accès, c'est-à-dire utilisables par plusieurs processus en même temps ; on a donc ici des files d'attente à plusieurs serveurs :

- utilisés par des clients sans priorités et gérés selon le mode PAPS, ou avec priorités fixes ;

- préemptible, avec reprise, ou non ;

- avec calendrier ou non, comme pour a).

On peut rattacher à cette classe de ressources les ressources utilisables par un nombre quelconque de processus, ou ressources à "nombre infini" de points d'accès.

c) Processeurs partagés

Ce sont des files à un serveur, celui-ci partageant son temps de manière équitable entre tous les processus utilisateurs, selon une technique de "tourniquet" ("round-robin" (KLE 76)).

- Les clients sont sans priorités et gérés selon le mode PAPS, ou avec priorités fixes ;

- Le processeur est soumis ou non à un calendrier, comme pour a).

Bien d'autres types de ressources sont envisageables : il n'est qu'à se référer, par exemple, à l'étonnante variété du panorama dressé par L. KLEINROCK dans (KLE 76). Mais, pour l'instant, nous avons limité les modèles programmés utilisables de manière standard à ceux que nous avons rencontrés, ou qui nous semblent susceptibles de rendre compte simplement et efficacement du comportement des processeurs réels.

2.2.4 Simplification concernant les ressources passives

L'hypothèse qui est généralement faite sur les ressources passives est qu'elles ne doivent pas affecter de façon primordiale le comportement du système. Aussi, dans les maquettes, on n'en tient généralement pas compte, sinon pour la forme, et uniquement dans le but d'évaluer leur utilisation.

Dans ce cas, on se ramène pour elles à des ressources à nombre infini de points d'accès et l'on cumule les temps d'utilisation des divers accès.

C'est dire à nouveau que nous nous plaçons le plus souvent à un niveau de détail qui permet de négliger la plupart des contraintes d'implémentation physique, hormise l'utilisation de processeurs globaux pouvant bloquer les processus.

En résumé, on peut dire qu'une maquette est un modèle réduit de système d'information. C'est aussi un système de processus coopérants, mais complètement formalisé, alors qu'un système d'information ne l'est que partiellement. On tente d'y conserver une structure d'information "semblable", mais réduite et souvent implantée physiquement de manière différente. Les processus y sont simplifiés et regroupés. Des ressources physiques, on ne garde que celles qui ont une influence déterminante sur le comportement du système ; on les regroupe, on les décrit par des types de ressources en nombre limité et on les gère indépendamment les unes des autres.

Partie II

MAQUETTES LOGICIELLES de SYSTEMES d'INFORMATION

Chapitres :

4. LANGAGES et OUTILS de DESCRIPTION et de SIMULATION de SYSTEMES.

5. ARCHITECTURE et DESCRIPTION GENERALES d'une MAQUETTE de SYSTEME d'INFORMATION.

6. ANALYSE des COMPOSANTS d'une MAQUETTE.

Partie II

MAQUETTES LOGICIELLES de SYSTEMES d'INFORMATION

Nous avons longuement hésité sur l'ordre de présentation des différents éléments composant cette partie : convenait-il de décrire d'abord les concepts sur lesquels reposent les modèles que nous proposons, ou plutôt le langage qui nous permet de les exprimer et de les rendre opérationnels ?

Nous avons finalement choisi la deuxième solution, qui, bien que pouvant laisser croire (il n'en est rien...) les concepts présentés liés au langage, permet de faire usage de celui-ci pour rendre plus précise (plus formelle) la description des différentes nctions.

Chapitre 4 - LANGAGES et OUTILS de DESCRIPTION et de SIMULATION de SYSTEMES

"Mustapha :

Mi saltella il cuor nel petto.
Che dolcezza di parlar ! "

(G. Rossini, L'Italienne à Alger, acte 1, scène 7)

"Mustapha :

Mon coeur palpite de joie.
Quel charmant langage !"

Chapitre 4 - LANGAGES et OUTILS de DESCRIPTION et de SIMULATION de SYSTEMES

1. CHOIX d'un LANGAGE et d'OUTILS.

1.1 Langages de simulation et langages de description de systèmes.

1.1.1 Langages de simulation.

1.1.2 Langages de description de systèmes.

1.2 Description du parallélisme et de la synchronisation.

1.2.1 Les coroutines.

1.2.2 Les réseaux de PETRI.

1.2.3 Les événements et tâches de PL/1.

1.2.4 Les sémaphores ; les primitives wait et signal.

1.2.5 Les propositions collatérales d'ALGOL 68 ; les primitives fork et join.

1.2.6 Les moniteurs.

1.2.7 L'attente conditionnelle ; un compromis moniteur-attente conditionnelle.

1.2.8 Les expressions de chemin.

1.2.9 Vers d'autres outils d'expression du parallélisme et de la synchronisation.

1.3 Le choix de SIMULA et des moniteurs.

1.3.1 Langage de description et de simulation.

1.3.2 Expression du parallélisme et de la synchronisation.

2. PRESENTATION du LANGAGE SIMULA 67.

2.1 Les classes hiérarchisées.

2.1.1 Déclaration des classes.

2.1.2 Déclaration d'un pointeur.

Nous dressons d'abord (1) un rapide panorama des langages de description de systèmes et des langages de simulation ; nous discutons ensuite les concepts généraux et les outils d'expression du parallélisme et de la synchronisation de processus ; nous justifions enfin le choix du langage que nous avons retenu : SIMULA 67.

Puis nous rappelons (2) les idées essentielles de ce langage : les notions de classes hiérarchisées, d'attributs virtuels, les facilités pour le traitement de listes (classe SIMSET) et pour la simulation (classe SIMULATION).

Enfin nous présentons en détail (3) le concept de base que nous avons choisi pour exprimer le partage de ressources par des processus : les moniteurs (au sens de HOARE).

Ce chapitre se veut un rappel de notions souvent bien connues des informaticiens, même si elles sont sans cesse débattues depuis quelques années (sur le parallélisme et la synchronisation), mais aussi d'idées moins bien diffusées et parfois redécouvertes sous des formes un peu différentes (sur le langage SIMULA notamment).

Le lecteur familier avec telle ou telle d'entre elles peut sauter sans préjudice les paragraphes qui lui correspondent.

1 - CHOIX d'un LANGAGE et d'OUTILS

1.1 Langages de simulation et langages de description de systèmes

Nous avons cherché à nous appuyer sur un langage qui soit à la fois adapté à :

- la description de systèmes de processus coopérants,
- la simulation à événements discrets par une technique d'exécution en quasi-parallélisme (parallélisme simulé sur mono-processeur).

Ce langage devait posséder de multiples qualités :

- reposer sur des concepts simples en nombre réduit ;

- avoir une syntaxe agréable ;

- posséder une sémantique précise ;

- assurer une programmation modulaire descendante ("top-down") ou ascendante ("bottom-up") ;

- permettre la fabrication de composants assez généraux, facilement paramétrables et extensibles ;

- autoriser une "mise en catalogue" de ces composants, avec, de préférence, une compilation séparée ;

- être associé à une bibliothèque de sous-programmes adaptés à la simulation.

- ...

Les langages pouvant se parer de toutes ces qualités sont peu nombreux, voire inexistants. Nous avons cependant refusé de nous lancer dans la définition et la compilation d'un nouveau langage pour plusieurs raisons :

- ce n'est pas notre domaine de recherche ;

- on gaspille souvent trop de temps à fabriquer un produit pouvant s'avérer mal adapté et dépassé quand il devient opérationnel ;

- d'autres équipes (que nous citons ultérieurement) se sont attaquées à un travail similaire ;

- la question du langage, pour importante qu'elle soit en tant que guide de la pensée, est tout de même secondaire en face de celle de la découverte des concepts fondamentaux du problème que l'on traite.

Nous avons alors adopté une démarche pragmatique consistant à

- faire, autant que possible, une étude des langages qui nous semblaient se rapprocher du profil défini ci-dessus ;

- choisir un de ces langages pour tenter d'exprimer nos modèles ;

- noter les difficultés rencontrées et les améliorations à apporter ;

- nous fonder sur cette expérience pour, dans une étape ultérieure, proposer un langage mieux adapté à notre domaine.

Nous résumons dans les paragraphes suivants l'étude que nous avons menée sur les langages de description et simulation de systèmes.

1.1.1 Langages de simulation

Les langages de simulation se divisent en deux catégories, selon qu'ils sont destinés à la simulation continue ou à la simulation à événements discrets (ou plus simplement : discrète).

- La première (BRA 77) permet de simuler des systèmes où les changements d'états paraissent s'opérer de manière continue, le plus souvent modélisés par des équations différentielles sur la variable temps.

Dans certains cas, principalement lorsque le nombre d'équations est limité et les coefficients constants, il est possible de résoudre analytiquement de tels modèles, sinon, on utilise des techniques de résolution numérique, et parmi elles, la simulation continue.

Un simulateur continu est simplement un programme pour la résolution numérique (par discrétisation) de systèmes d'équations différentielles, possédant notamment plusieurs routines d'intégration.

Les langages suivants permettent de fabriquer des simulateurs continus :

SL/1, CSMP, qui sont tous deux des macro-générateurs de FORTRAN (BRA 77), MIMIC (BRA 77), DYNAMO (PUG 63), qui est le langage développé à partir du modèle de FORRESTER (chapitre 2), etc...

Nous n'en dirons pas plus sur ce sujet, puisque ce n'est pas le type de modèles et de simulation que nous avons retenu. Le lecteur intéressé pourra consulter par exemple (NAY 66).

- La deuxième permet de simuler des systèmes où les changements d'états s'opèrent de manière discrète, ou, tout au moins, sont vus comme tels dans un modèle. C'est ce type que nous avons retenu pour nos maquettes de systèmes d'information.

Il existe plusieurs manières de considérer un modèle à événements discrets, et nous citerons trois approches : par événements, par activités, par processus, termes dont nous rappelons les définitions, du point de vue de la simulation.

Un événement, comme nous l'avons déjà dit, est un changement d'état du modèle.

Un processus est une séquence d'événements ordonnés dans le temps. Ceci correspond bien à la notion de suite de modifications que nous avons rencontrée pour le système d'information (chapitre 1).

Une activité est une collection logique d'opérations pouvant changer l'état du système et dont le déclenchement est soumis à une condition.

Exemple 4.1

Pour mieux illustrer ces différentes notions, nous considérons un modèle de file d'attente simple avec un seul serveur (figure 4.1). Dans notre contexte, ceci peut correspondre, par exemple, à un traitement de messages (clients) par une personne (serveur) à un guichet.

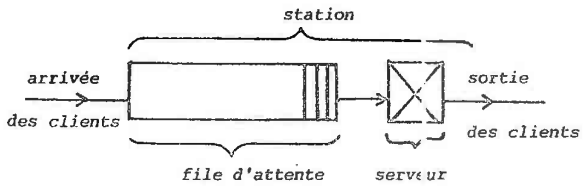


Figure 4.1 - File d'attente simple à 1 serveur

Les événements sont ici de deux types : arrivée d'un client et fin de travail du serveur pour un client.

Pour ce qui concerne les processus, trois manières de voir peuvent être envisagées :

- on considère comme entités actives les clients et comme entité passive le serveur : on associe à chaque client un processus qui est la suite des événements liés à ce client dans le modèle : arrivée, début et fin de service pour ce client ;

- on considère comme entités passives les clients et comme seule entité active le serveur : on associe à celui-ci un processus qui est une suite cyclique d'événements : entrée en service du client suivant, fin de service de ce client, etc...

- on considère à la fois les clients et le serveur comme des processus ; ceci correspond à une tendance actuelle en modélisation (voir 1.2.9).

Il y a plusieurs manières, de voir les activités :

- on peut n'en considérer qu'une seule : l'activité générale de service des clients : pour chacun d'eux, introduction dans le modèle, attribution, puis libération du serveur ;

- on peut en considérer deux : une activité d'introduction des clients et une de service proprement dit.

□

Dans les deux premières approches (par événements et par processus), la simulation est fondée sur la gestion dynamique d'un échéancier : ensemble de "notices d'événements" ordonnées par dates d'occurrences croissantes

et, à dates égales, grâce à des règles d'ordonnancement que l'on peut préciser. Cette gestion est réalisée automatiquement par un noyau de synchronisation dont le rôle est de parcourir l'échéancier dans l'ordre croissant, et de faire exécuter des changements d'états correspondant aux événements dont les notices sont rencontrées. Le temps courant simulé est toujours fixé par la date de la notice dont l'événement est en cours de traitement. Bien entendu, un changement d'état peut consister à modifier l'échéancier par suppression ou adjonction de notices d'événements.

Dans la troisième approche (par activités), la simulation est fondée sur la gestion dynamique d'horloges associées à certaines entités du modèle. Un noyau de synchronisation détermine à chaque étape la valeur minimale repérée par ces horloges qui devient le temps courant de la simulation. Puis, les conditions de déclenchement de toutes les activités sont évaluées successivement, de manière à modifier l'état du système et donc, éventuellement, les valeurs repérées par les horloges.

Nous examinons ces trois approches en nous appuyant sur l'exemple 4.1.

a) Approche par événements

Les événements sont regroupés par types : à chacun d'eux, on associe une procédure susceptible de modifier l'état du modèle, y compris l'échéancier. Quand le noyau de synchronisation doit traiter une notice d'événement, il appelle la procédure associée au type de cet événement.

Exemple 4.2

Dans l'exemple 4.1, il faudrait écrire deux procédures : l'une pour les événements du type arrivée de clients, l'autre pour ceux du type fin de service. □

Les langages les plus connus, construits selon cette approche, sont SIMSCRIPT I (KAR 65) et II (KIV 69), extensions de FORTRAN ou PL/1 pour la simulation, et GASP (PRIT 69).

b) Approche par processus

Elle utilise la notion de coroutine (CON 63) ou processus, dont nous serons amenés à reparler par la suite (1.2). Disons simplement ici que ce sont des instances (blocs d'appel) de procédures, du même niveau, dont l'exécution peut être interrompue, puis reprise ultérieurement. On peut simuler ainsi le déroulement en parallèle de certaines actions sur un ordinateur mono-processeur : c'est ce que l'on appelle quasi-(ou pseudo-) parallélisme. Lorsqu'il traite une notice d'événement, le noyau de synchronisation débloque un processus, le processus courant. Celui-ci accomplit alors certains changements d'états, éventuellement sur l'échéancier, avant de se terminer ou de s'interrompre à nouveau pour une durée déterminée (dans ce cas, il se réordonnance dans l'échéancier) ou non (dans ce cas, il doit attendre qu'un autre processus le libère).

Exemple 4.3

Dans le modèle de file d'attente ci-dessus (exemple 4.1) :

- si l'on considère les clients comme des entités actives, on crée un un type de coroutines clients décrivant le cheminement d'un client dans le modèle ; l'arrivée d'un client consiste en l'activation d'une nouvelle coroutine de ce type, qui s'exécute en parallèle (ou plutôt en pseudo-parallélisme) avec les autres, simulant ainsi la progression conjointe des clients dans le modèle ; notons que la première action d'une telle coroutine peut être de préparer la coroutine simulant la progression du client suivant, chaque arrivant créant ainsi son propre successeur ;

- si l'on considère comme seule entité active le serveur, on lui associe une coroutine cyclique, dont chaque cycle consiste à prendre un client dans la file, à le servir, puis à le faire sortir du modèle ; on lui adjoint alors une coroutine cyclique de génération de clients et d'alimentation de la file d'attente ;

on a ici deux vues duales du même système ;

- si l'on considère comme entités actives les clients et le serveur, on leur associe des coroutines dont le rôle est sensiblement le même pour les clients que dans la première approche, et pour le serveur que dans la deuxième.

□

Les langages les plus connus, construits selon cette approche, sont :

- GPSS (IBM 70, SCH 72), qui permet de décrire le cheminement de "transactions" dans un modèle où se trouvent des entités passives, des ressources critiques ("facilities") ou multiples ("storages"), dont les accès sont alloués aux transactions selon diverses politiques ;

- SIMULA 67 (DAH 66, DAH 70) avec ses puissants concepts de classes et de processus, dont nous reparlerons abondamment (2).

- Les langages très récents fondés sur les processus séquentiels communicants (CSP (HOA 78)) ou distribués (BRI 78), sur lesquels nous reviendrons en (1.2.9).

c) Approche par activités

On détermine certaines entités auxquelles on associe des horloges. On considère alors certaines activités, procédures dont l'appel est soumis à certaines conditions.

Exemple 4.4

Dans le modèle de l'exemple 4.1, on attache une horloge au prochain client et une au serveur.

- On peut ne retenir qu'une seule activité, le service général d'un client, soumis à une condition vraie si et seulement si, à la date courante considérée (minimum des temps des horloges), un client arrive ou le serveur achève de servir un client. A un instant donné de la simulation, si la condition est vraie (elle l'est toujours ici), le noyau de synchronisation fait exécuter l'activité, qui modifie l'état du système (incrémente les horloges,

notamment) ; puis il réévalue la condition et, si elle est à nouveau vraie, refait exécuter l'activité, et ainsi de suite, jusqu'à ce que la condition soit fausse et que le noyau fasse progresser le temps courant.

- On peut aussi considérer deux activités. La première consiste à introduire les clients dans le modèle et est soumise à la condition : temps courant correspondant à la date d'arrivée d'un client (horloge prochain client). La deuxième consiste à servir les clients et est soumise à la condition : temps courant correspondant à la date de fin de service d'un client (horloge serveur).

□

D'une manière générale, quand il y a plusieurs activités, un pas de la simulation, à temps courant constant, consiste à parcourir la liste des activités, dans un ordre à préciser, en évaluant pour chacune sa condition de déclenchement. Si cette condition est vraie, le noyau fait exécuter l'activité, puis recommence à scruter les conditions au début de la liste des activités. Et ainsi de suite, jusqu'à ce qu'il n'y ait plus aucune condition vraie, que le noyau parvienne à la fin de la liste des activités, et donc au terme du pas de simulation ; il fait alors progresser le temps courant en consultant les horloges.

Les langages les plus connus qui utilisent cette approche, sont CSL (BUX 66) et SIMON (HIL 67). Mais il semble que cette manière de voir assez originale soit un peu en déclin et cherche un second souffle.

La bibliographie sur les langages de simulation est très importante. Nous avons donné un certain nombre de références qui peuvent être complétées par des articles ou ouvrages plus généraux. Ainsi, O.J. DAHL, dans (DAH 68), compare plusieurs langages de simulation à événements discrets ; G. FISHMAN, dans (FIS 73), développe les différentes approches dont nous avons parlé et présente quelques langages ; B.P. ZEIGLER, dans (ZEI 76), reprend ces différentes méthodes dans le cadre d'une théorie de la modélisation et de la simulation, d'une manière plus formelle.

1.1.2 Langages de description de systèmes

Le développement, dans les années 60, des grands calculateurs, avec de nombreuses ressources centralisées, a nécessité l'écriture de systèmes d'exploitation ("operating systems") volumineux et très complexes. On a dès lors cherché des langages permettant d'aider à décomposer (ou au moins à exprimer la décomposition de) ces systèmes en modules relativement autonomes, bien spécifiés, reliés par des interfaces bien définis. Ces modules devant parfois s'exécuter en parallèle, on a voulu exprimer leur coopération et leur concurrence de la manière la plus claire possible.

Ceci a été à l'origine d'une tendance consistant à s'éloigner progressivement des langages de bas niveau (langages d'assemblage) et tenter d'utiliser de plus en plus des langages de haut niveau pour écrire un système. Cette écriture peut être envisagée soit comme une manière de bien spécifier le produit à obtenir, soit comme une écriture définitive du système, ce qui est plus ambitieux et nécessite un compilateur générant un code adapté à la machine-cible, parfois un trans-compilateur ("cross-compiler").

Ainsi, certains essais ont été faits pour utiliser ALGOL 60 ou PL/1 (COR 69), puis on a semblé préférer créer des langages mieux adaptés, tels que LIS : Langage d'Implémentation de Systèmes (ICH 72). D'autres tentatives ont eu lieu avec SIMULA 67 (DAH 70).

Le succès rencontré par le langage PASCAL (WIR 71), appartenant à la famille d'ALGOL, simple et avec une sémantique précise (HOA 72), a conduit certains concepteurs de systèmes à le prendre comme base de langages de description. A PASCAL ont été souvent adjointes les notions de processus (ou coroutines), pour décrire le parallélisme, et de moniteur (voir 1.2.6), pour décrire l'exclusion mutuelle et la synchronisation,

conformément aux idées de programmation structurée, maintes fois exprimées, par exemple dans (DDH 72) ou (OPE 72). C'est ainsi que sont apparus des langages de description de systèmes tels que CONCURRENT PASCAL (BRI 75), SIMONE (KAU 76) ou MODULA (WIR 76). Ils ont tous comme premier but l'écriture de systèmes, immédiatement opérationnels dans un contexte artificiel de quasi-parallélisme incorporé au logiciel, ce qui permet de vérifier le bon fonctionnement des mécanismes logiques décrits.

D'autres langages avec des idées proches ont vu le jour : LEST (RET 77) utilisant la notion de "connecteur", SESAME (CHEV 76), celle de chemin de synchronisation (1.2.8), etc.... De nombreux projets proposent de concevoir, décrire et implanter "proprement" des systèmes : SOLO (BRI 76), XIMONE (BEZ 78), etc...

Avec le développement récent des grands réseaux d'ordinateurs interconnectés, mais aussi des petits réseaux de micro-ordinateurs, ces problèmes revêtent encore plus d'acuité.

Ainsi l'on voit poindre des langages permettant de décrire des systèmes opératoires, ou des applications, sous forme de processus répartis implantés sur ces réseaux. C'est le sens des propositions d'acteurs ("actors") de C. HEWITT (HEW 77), de processus séquentiels communicants (CSP) de C.A.R. HOARE (HOA 78) et de processus distribués de P. BRINCH HANSEN (BRI 78). Tous les trois refusent la notion de structures de données globales partagées par des processus, pour garder seulement celle de processus, en unifiant en quelque sorte les concepts de moniteurs et de processus. Dans les deux derniers la description des actions est faite de manière indéterministe, en utilisant la notion de commandes gardées de E.W. DIJKSTRA (DIJ 75), avec blocages éventuels.

Les programmes ainsi obtenus sont très élégants, mais leur implantation réelle semble poser quelques problèmes pratiques, dès lors que l'on

s'éloigne de l'hypothèse forte suivante : à chaque processus est affecté en permanence et en exclusivité un processeur qui l'exécute. Il est vrai que ceci est admissible pour un réseau de micro-processeurs.

Ces nouvelles idées ont influencé la conception du langage GREEN (HON 70a, HOA 79b) retenu par le DOD des USA, sous la dénomination ADA, pour les systèmes temps réels.

La forme d'un langage de description de systèmes peut dépendre assez fortement des outils utilisés pour décrire le parallélisme et la synchronisation. Nous développons maintenant cet aspect avant d'expliquer notre choix.

1.2 Description du parallélisme et de la synchronisation

Nous passons rapidement en revue les différents concepts qui ont été proposés pour décrire les phénomènes de parallélisme et de synchronisation dans les systèmes.

1.2.1 Les coroutines (ou processus)

Dans (KNU, p. 226), D. KNUTH fait un bref historique de la notion de coroutine (ou processus). Il signale que ce terme a été utilisé pour la première fois par M.E. CONWAY en 1958, après qu'il ait développé ce concept et qu'il l'ait appliqué initialement à la construction de programmes d'assemblage. La première publication expliquant cette notion date de 1963 (CON 63) : elle est illustrée par la fabrication d'un compilateur COBOL à un passage.

Les coroutines constituent une généralisation de la notion de sous-programmes ("subroutines"). Au contraire de la relation dissymétrique qui existe entre un programme principal et un sous-programme, il y a

une symétrie complète entre coroutines, en ce sens qu'elles peuvent s'appeler l'une l'autre.

Exemple 4.5 (tiré de (KNU 68))

Supposons que l'on a deux coroutines A et B ; en programmant A, on doit penser que B est un sous-programme, mais, en programmant B, on doit aussi penser que A est un sous-programme. C'est-à-dire que, dans A, on utilise une instruction de branchement "JMP B" (dans le langage MIX de KNUTH) pour activer la coroutine B. Dans B l'instruction "JMP A" est utilisée pour réactiver A. Quand une coroutine est activée, elle reprend l'exécution de son programme au point où elle avait été suspendue. A et B constituent donc une "équipe" de composants du même niveau.

Ici, on considère que les exécutions des deux coroutines ne peuvent pas être simultanées : le branchement à B interrompt l'exécution de A et inversement. On a en réalité une exécution de A et B en quasi-parallélisme.

□

Comme nous le verrons par la suite, cette notion a été reprise sous des formes diverses. Dans le cadre d'une programmation structurée, on peut critiquer l'utilisation de branchements explicites à des coroutines, analogues aux aller à de la programmation séquentielle.

1.2.2 Les réseaux de PETRI

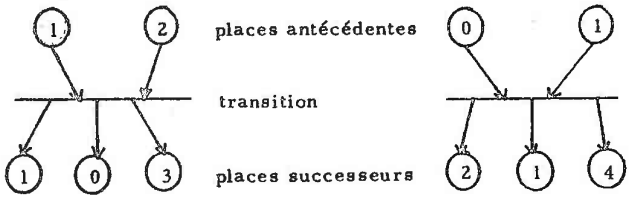
La théorie des réseaux de PETRI a été développée à partir des travaux de C.A. PETRI qui, dans sa thèse (PET 72), présente un nouveau modèle de flux d'information dans les systèmes. Ce modèle est fondé sur les concepts d'asynchronisme et de concurrence des composants d'un système et l'idée que les relations entre ces composants peuvent être représentées par un graphe (ou réseau). Nous reprenons ici quelques points d'articles de base sur cette question (PET 77, GIR 78).

Un graphe de PETRI (P∪T, Γ) comprend 2 sortes de nœuds : les éléments de P sont des places (représentées par des cercles) et ceux de T sont des transitions (représentées par des barres). Ses arcs, définis par la relation $\Gamma \subset P \times T \cup T \times P$, relient une place à une transition ou une transition à une place. Un marquage des places est une application $P \to \mathbb{N}$, où $\mathbb{N}$ est l'ensemble des entiers naturels : on dit qu'à une place, on associe un certain nombre de marques. Dans la suite, on suppose que P et T sont des ensembles finis.

Le marquage évolue dans le temps selon la règle suivante, dite de déclenchement (ou de "mise à feu") d'une transition :

Si toutes les places antécédentes d'une transition t (les éléments de $\Gamma^{-1}(t)$) ont un nombre de marques strictement positif, alors t est dite déclenchable ; le déclenchement peut alors intervenir (pas nécessairement instantanément) : c'est une opération indivisible qui consiste à diminuer de 1 le nombre de marques de chaque place antécédente et à augmenter de 1 le nombre de marques des places successeurs (éléments de $\Gamma(t)$).

La figure 4.2 illustre la mise à feu d'une transition.



a. Marquage avant mise à feu b. Marquage après mise à fe

(on note dans chaque cercle (place) le nombre de marques)

Figure 4.2 - Mise à feu d'une transition

Aux places, on associe des ressources dont la disponibilité momentanée est représentée par leur nombre de marques. Aux transitions, on associe des actions qui consomment et produisent des ressources.

Les réseaux de PETRI sont utilisés pour exprimer le contrôle (coopération et concurrence) dans les systèmes où les actions (modifications) peuvent s'effectuer en parallèle. Ils servent pour l'analyse de ce contrôle, en permettant d'en étudier certaines propriétés. A titre d'exemple, nous en présentons quelques unes, qui nous semblent adaptables à notre domaine.

Exemple 4.6

- Un réseau est k-borné si, pour chaque place, le nombre de marques est toujours inférieur ou égal à k ; un réseau 1-borné est dit sage ; cette question est importante, car elle conditionne parfois une implantation physique correcte de compteurs.

- Un réseau est conservatif si la somme des nombres de marques dans le réseau est constante.

- Une transition est morte pour un marquage si elle ne peut plus être déclenchable après que ce marquage soit atteint ; une transition est vivante si elle est, ou peut devenir, déclenchable après tout marquage du réseau : l'importance de ces questions vient de considérations sur la modélisation des systèmes, elles sont liées à celle d'interblocage (chapitre 1) ; ainsi, il est non seulement important qu'une transition soit déclenchable dans un marquage donné, mais aussi qu'elle puisse être déclenchable après tous les marquages que l'on peut atteindre après ce marquage : si ceci n'est pas vrai, il est possible d'atteindre un marquage dans lequel la transition est morte, ce qui peut signifier une étreinte fatale (ou interblocage).

- Un marquage m' est atteignable à partir d'un marquage m s'il existe une suite de déclenchements de transitions qui, à partir de m, conduit à m'; on montre que le problème de l'atteignabilité (un ensemble fini de marquages est-il un sous-ensemble de l'ensemble des marquages atteignables du réseau ?) est équivalent au problème de vivacité (toutes les transitions sont-elles vivantes ?) et qu'il est décidable.

□

Ces propriétés peuvent être vérifiées, notamment par des techniques d'algèbre linéaire, quelquefois, malheureusement, avec des algorithmes de complexité exponentielle.

On trouvera de plus amples détails dans (PET 77) et (GIR 78).

L'usage de réseaux de PETRI pour décrire le contrôle d'un système à composants parallèles fait irrésistiblement penser à celui des organigrammes pour exprimer les algorithmes séquentiels. Comme pour ceux-ci, il est bien sûr possible de structurer les réseaux, de les écrire en niveaux successifs. Mais, de la même manière, on peut songer à leur abandon au profit de langages de programmation structurée de systèmes de processus coopérants, tels ceux dont nous parlons par ailleurs.

Les réseaux de PETRI sont aussi utilisés pour exprimer la sémantique de mécanismes de synchronisation (LAU 75).

1.2.3 Les événements et taches de PL/1

Certains langages de programmation possèdent des extensions orientées vers le calcul parallèle et la synchronisation. C'est le cas de PL/1, sous le système OS 360/MVT (VEI 72).

Il est possible d'y déclarer des identificateurs d'événements. Chacun est implémenté en deux parties : un indicateur de fin booléen ayant la valeur 1 si l'événement s'est produit et 0 sinon ; un indicateur d'état entier ayant la valeur 0 pour l'état normal et une valeur différente pour un état anormal. On accède ou modifie les deux indicateurs d'un événement identifié par x en appelant completion (x) ou status (x).

On définit en outre, par des appels de procédures un peu particuliers, des coroutines appelées taches, qui peuvent s'exécuter en pseudo-parallélisme.

Une tâche peut attendre la réalisation de n événements parmi une liste ℓ d'événements en exécutant l'instruction d'attente : wait (ℓ) (n).
On a ainsi la possibilité de définir des systèmes assez complexes de tâches, synchronisées par completion et wait.

Mais on peut reprocher à ces notions un certain manque de clarté et de sûreté dûs à une sémantique trop floue. De plus, des restrictions importantes ne facilitent pas toujours la définition d'un système : par exemple, un même événement ne peut pas être attendu par plus d'une tâche, et il n'y a pas de possibilité d'enregistrer automatiquement plusieurs signaux d'activation.

1.2.4 Les sémaphores ; les primitives wait et signal

En 1967, E.W. DIJKSTRA introduit le très important concept de sémaphore (DIJ 67).

Un sémaphore s (CRO 75) est constitué d'une variable entière e (s) et d'une file d'attente f (s) dont la politique de gestion est laissée à la guise du programmeur. A la création du sémaphore, e (s) prend une valeur e_o (s) ≥ 0 et la file f (s) est vide.

On peut agir sur un sémaphore s par les deux primitives suivantes, qui sont des opérations indivisibles (CRO 75) :

```
P (s)  : début
            e (s) := e (s) - 1 ;
            si  e (s) < 0 alors
               début  co on suppose que cette primitive est exécutée par
                                 le processus r  co
                  état (r) := bloqué ;
                  mettre le processus r dans la file f (s)
               fin
         fin
```

```
V (s) : début
          e (s) := e (s) + 1 ;
          si e (s) ≤ 0 alors
            début
               sortir un processus de la file f (s) ;
               co soit q le processus sorti co
               état (q) := actif
            fin
        fin
```

Ce mécanisme favorise l'écriture de systèmes assez complexes de processus coopérants ou concurrents. Il ne répond cependant pas à tous les besoins, notamment celui de communiquer, d'un processus à l'autre, autre chose qu'un signale de déclenchement. C'est pourquoi d'autres propositions ont été faites, comme les sémaphores avec messages (SAA 70).

Dans (HAB 72), A.N. HABERMANN propose un mécanisme de synchronisation fondé sur l'emploi de deux primitives wait et signal, qui ressemblent fort aux procédures P et V. Il les utilise pour prouver certains résultats fondamentaux, notamment sur les systèmes producteurs-consommateurs.

De tels mécanismes sont souvent jugés aujourd'hui trop primitifs et d'un usage hasardeux (CAM 74) ; certains les trouvent trop près de l'instruction allerà ("go to") de la programmation séquentielle (HOA 74).

1.2.5 Les propositions collatérales d'ALGOL 68 ; les primitives fork et join

En ALGOL 68, le parallélisme s'exprime grâce aux propositions collatérales et la synchronisation peut être réalisée par des sémaphores.

Une proposition collatérale (WIJ 78, GAA 72) est une suite parenthésée de propositions unitaires séparées par des virgules.
Les exécutions de ces propositions unitaires sont mêlées dans le temps d'une manière quelconque.

Notons que ces propositions collatérales recouvrent aussi bien des déclarations que des traitements parallèles.

Le programmeur peut synchroniser des traitements grâce aux sémaphores, objet du mode structuré séma (WIJ 78, GAA 72 (10.4)), qui sont initialisés par l'opération unaire /.

Aux primitives P et V de DIJKSTRA correspondent les opérations unaires ↑ et ↓, dont la sémantique est légèrement différente.

L'utilisation ou non du symbole par devant une collatérale conduit à la notion de niveau de synchronisation (CAA 73) :

- si l'on n'utilise pas le symbole par : le blocage d'une proposition unitaire dans la collatérale bloque toutes les autres ;

- si l'on utilise le symbole par : le blocage d'une proposition unitaire dans la collatérale n'a pas d'influence sur l'élaboration des autres.

L'élaboration d'une proposition collatérale est terminée quand toutes les élaborations des propositions unitaires qui la composent sont achevées. Ces propositions ont donc en quelque sorte un lieu de rendez-vous commun.

Ce n'est plus le cas quand, quittant ALGOL 68, on utilise des primitives de synchronisation telles que fork et join. La primitive fork (fourche) permet d'initialiser plusieurs traitements parallèles. Des lieux de rendez-vous peuvent être définis de manière quelconque par des étiquettes dans le programme. La primitive join (e, k) (jonction), exécutée par un traitement en parallèle, lui impose d'aller à l'étiquette e, où il doit en retrouver k-1 autres avant que la séquence d'instruction étiquetée par e puisse être exécutée.

Ces primitives permettent de décrire des systèmes aux relations plus complexes, mais elles accroissent considérablement les risques d'erreurs dans les programmes et rendent pratiquement impossible toute preuve. On ajoutera, comme pour la suppression des allerà au profit des boucles dans la programmation structurée, qu'un effort supplémentaire d'analyse permet toujours d'éviter l'emploi de tels mécanismes.

1.2.6 Les moniteurs

La notion de moniteur est née des réflexions conjointes de chercheurs sur la conception de systèmes d'exploitation : E.W. DIJKSTRA, P. BRINCH HANSEN, C.A.R. HOARE ... Elle a pris successivement plusieurs formes avant d'aboutir à une version à peu près stable : section critique conditionnelle, secrétaire ... Elle est cependant encore un peu différente chez HOARE (HOA 74) et BRINCH HANSEN (BRI 75). Nous nous appuyons ici sur la première pour en faire une présentation assez informelle ; nous signalons à l'occasion les différences avec la deuxième. La longueur de ce paragraphe tient au fait que, par la suite, nous utilisons constamment cette notion.

Un moniteur est l'association d'une structure de données locale, de procédures et fonctions d'accès ou de modifications de ces données, qui sont appelées de l'extérieur par des programmes.

Nous reprenons les notations adoptées dans le langage SIMONE (KAU 76). La déclaration d'un moniteur a la forme :

```
nomdemoniteur : monitor
    begin
        ... déclaration de données locales ...
        procedure nom de procédure (... paramètres formels ...) ;
            begin ... corps de procédure ... end ;
        ... déclarations d'autres procédures ou fonctions ...
        ... initialisation des données locales ...
    end
```


Naturellement, les corps des (fonctions-) procédures peuvent avoir aussi des données locales.

Pour appeler de l'extérieur une (fonction-) procédure d'un moniteur, on utilise la notation pointée :

nomdemoniteur . nomdeprocédure (.. paramètres effectifs ...)

Il est possible de déclarer plusieurs moniteurs avec la même structure, grâce à la fabrication de modèles de moniteurs en SIMONE (BEZ 76), ou de classes de moniteurs paramétrées en SIMULA (voir 4.3).

Un programme utilisateur d'un moniteur n'a en principe pas à savoir comment sont organisées les données locales au moniteur : il n'y a accès et il ne peut les modifier que par l'intermédiaire des (fonctions-) procédures associées au moniteur, ce qui est un gage de sécurité.

Pour éviter des effets chaotiques possibles à l'intérieur d'un moniteur, on impose que l'exécution des appels des (fonctions-) procédures soit faite en exclusion mutuelle : quand un appel a lieu, on peut bloquer son exécution pour qu'elle ne commence qu'après la fin de l'exécution de l'appel précédent.

Les moniteurs sont utilisés pour décrire des allocateurs de ressources et pour exprimer la synchronisation de programmes parallèles (ou processus). Pour ce faire, ils utilisent un mécanisme de condition.

Une condition est une file d'attente dans laquelle on peut bloquer des processus et qui est déclarée à l'intérieur d'un moniteur.

En SIMONE, une déclaration de condition a la forme :

nomdecondition : condition

On peut agir sur une telle file d'attente-condition par l'intermédiaire de deux procédures, wait et signal, appelées à l'intérieur d'une procédure ou fonction d'un moniteur.

L'appel nomdecondition . wait bloque dans la file d'attente le processus invocateur, qui relâche son exclusivité sur le moniteur.

L'appel nomdecondition . signal libère immédiatement un des processus en attente dans la file ; s'il n'y en a pas, cet appel n'a pas d'effet. Notons que, pour maintenir l'exclusion mutuelle, le processus "signalant" est bloqué momentanément jusqu'à ce que le processus "signalé" quitte le moniteur.

Dans (BRI 75), une condition ne peut contenir qu'un processus à la fois, ce qui semble suffire à la plupart des cas rencontrés dans les systèmes d'exploitation, mais cause parfois certaines difficultés : quand au plus n processus sont susceptibles d'être bloqués pour une raison analogue, on déclare alors un tableau de n conditions.

Exemple 4.7

Pour reprendre un exemple simple (HOA 74), nous déclarons un moniteur chargé d'allouer une ressource critique, acquise et libérée par un nombre inconnu de processus clients, grâce à l'appel des procédures acquérir et libérer. Le booléen occupé détermine si la ressource est occupée ou non. Si une tentative est faite pour acquérir la ressource quand elle est occupée, le processus en cause est bloqué sur la condition nonoccupé. Il pourra être ultérieurement libéré par un appel de signal d'un autre processus sur cette condition. On initialise occupé à faux.

```
ressourcecritique : monitor
    begin
        occupé : boolean ;
        nonoccupé : condition ;
        procedure acquérir ;
        begin
            if occupé then nonoccupé . wait ;
            occupé := true
        end ;
```



```
        procedure libérer ;
        begin
            occupé := false ;
            nonoccupé . signal
        end ;
        occupé := false
    end
```

□

La politique de gestion des files d'attente-conditions n'est pas précisée : l'implémenteur est libre de son choix. Dans l'exemple ci-dessus, cette gestion pourrait être premier arrivé - premier servi, ce qui garantit notamment à chaque processus d'être exécuté.

Notons que le moniteur ressourcecritique simule en fait un sémaphore binaire (DIJ 67), ce qui est une preuve simple que le concept de moniteur avec conditions est au moins aussi puissant que les sémaphores.

Pour plus de détails sur ces notions, on consultera notamment (HOA 74) : on y montre l'interprétation complète du mécanisme en terme de sémaphores, divers exemples classiques et quelques extensions. Nous y reviendrons nous-même par la suite.

1.2.7 L'attente conditionnelle ; un compromis moniteur - attente conditionnelle

Nous avons déjà signalé que certains langages de simulation, fondés sur le concept d'activité, possèdent un noyau de synchronisation qui parcourt continuellement une liste de conditions de déclenchement des activités d'un programme. Quand une activité a été exécutée, le noyau revient au début de la liste d'activité et recherche séquentiellement une condition vraie permettant le déclenchement de l'activité correspondante (voir 1.1.1.c).

Si un tel mécanisme est difficilement concevable pour implanter des systèmes parallèles en vraie grandeur, il peut être en revanche utilisé pour les spécifier ou pour des modèles de simulation en quasi-parallélisme.

En effet, dans un processus décrit avec ce mécanisme, on n'a qu'une primitive permettant d'attendre des ressources, waituntil, qui est appelée de la manière suivante :

waituntil (condition)

où condition est une expression booléenne vraie si et seulement si le processus peut acquérir les ressources nécessaires à la poursuite de son exécution. Aucun besoin ici, comme dans les sémaphores ou les moniteurs par exemple, d'un mécanisme signalant la libération de ressources, ce qui fait éviter des erreurs de programmation.

Le waituntil a été notamment étudié par J. VAUCHER et adjoint au langage SIMULA (VAU 73).

La contre-partie de cette simplicité d'expression est une certaine lourdeur à l'exécution.

Dans un article (KES 77), J. L.W. KESSELS propose un compromis entre les moniteurs et l'attente conditionnelle. Il consiste à ne faire évaluer au noyau de synchronisation, après un blocage de processus, que des expressions booléennes associées aux conditions. Ceci permet de générer automatiquement des "signal" sur les conditions, mais en moins grand nombre que dans le waituntil classique. J. VAUCHER en a tenu compte pour l'amélioration qu'il propose dans (VAU 78).

1.2.8 Les expressions de chemin

Avec les expressions de chemin de R.H. CAMPBELL et A.N. HABERMANN (CAM 74), une idée neuve se fait jour : celle de

séparer les mécanismes de synchronisation des algorithmes sur lesquels ils agissent.

Ici, la synchronisation est spécifiée directement par description de la manière dont le corps d'une procédure (prise comme unité) est autorisé à être exécuté en relation avec d'autres, indépendamment des processus invocateurs.

Une expression de chemin décrit spécifiquement la synchronisation permise entre exécutions de procédures et prohibe toutes les autres, c'est-à-dire qu'un processus qui essaie d'exécuter une des procédures doit attendre jusqu'à ce que la combinaison de circonstances spécifiée dans l'expression se produise.

Les expressions de chemins sont construites à partir de deux schémas fondamentaux et de deux schémas additionnels qui utilisent la notion d'action.

Une action est l'exécution d'une procédure par un processus. Les deux schémas fondamentaux sont la séquence et la sélection d'actions.

Une séquence d'actions permet à chacune d'elles d'apparaître dans l'ordre indiqué. Si p, q, r sont des noms de procédures, l'expression de chemin

p ; q ; r

impose que les trois procédures soient exécutées séquentiellement (l'une après l'autre) dans la séquence donnée. Elles peuvent bien sûr être invoquées par des processus différents.

Une sélection d'actions permet à l'une d'entre elles seulement d'être exécutée. Ainsi, l'expression

p, q, r

spécifie qu'une sélection doit être faite entre p, q et r. Le processus essayant d'exécuter la procédure sélectionnée est autorisé à continuer,

tandis que des processus tentant d'exécuter les procédures non sélectionnées sont différés. La sélection d'une procédure est faite parmi celles qui ont été invoquées par les processus, de manière purement aléatoire.

Pour simplifier l'implémentation, on autorise une seule apparition d'un nom de procédure dans une expression (sinon, on a des "repeat-paths" (LAU 75)).

Les deux schémas de base peuvent être composés pour former des expressions de chemin plus complexes, grâce au parenthésage (CAM 74).

Les deux schémas additionnels sont la répétition et l'exécution simultanée.

La répétition permet à une expression complètement terminée d'être répétée indéfiniment. Ainsi, l'expression

path p end

permet à la procédure p d'être exécutée plusieurs fois, par exemple par des processus cycliques.

L'exécution simultanée indique que différents processus peuvent exécuter concurremment des procédures données. L'expression

{ p }

autorise la procédure p à être exécutée par plusieurs processus en simultanéité. Une fois qu'un processus a commencé à exécuter p, d'autres processus peuvent faire de même sans être différés, pourvu qu'il s'agisse d'exécutions en retard (incomplètes) de p. Aussitôt que la dernière de celles-ci est finie, l'expression de chemin est considérée comme terminée et d'autres processus invoquant p sont différés.

Dans (CAM 74), on dit comment combiner ces différents schémas pour exprimer des synchronisations complexes. De plus, on montre comment structurer un programme grâce à l'introduction de types, comparables

aux classes de SIMULA ou aux moniteurs, dans lesquels on fait figurer ces expressions de chemin, mais en dehors des fonctions et procédures synchronisées. On prouve de plus que la puissance de ce mécanisme est au moins celle des sémaphores de DIJKSTRA.

Du point de vue de l'implantation, chaque expression de chemin donne naissance à un contrôleur : si une procédure individuelle synchronisée est invoquée par un processus, le contrôleur décide si l'exécution de cette procédure peut commencer, et donc le processus continuer. Ce mécanisme agit comme suit :

- chaque procédure est complétée par un prologue et un épilogue ;

- un processus qui exécute le prologue s'enquiert auprès du contrôleur s'il est autorisé à continuer ; le contrôleur décide ; finalement, quand le processus exécute l'épilogue de la procédure, il prévient le contrôleur qui peut alors reprendre d'autres processus différés.

C'est en principe le compilateur qui, en utilisant les outils de synchronisation existants (notamment les sémaphores), génère automatiquement les contrôleurs, les prologues et les épilogues.

Pour plus de détails, on lira (CAM 74) et (LAU 75) où la sémantique des expressions de chemin est exprimée par des réseaux de PETRI.

1.2.9 Vers d'autres outils d'expression du parallélisme et de la synchronisation

L'idée de séparer des procédures l'expression du parallélisme et de la synchronisation, c'est-à-dire du contrôle, a été reprise sous différentes formes.

On la retrouve dans la description de la concurrence entre processus par des compteurs chez J.P. VERJUS (VER 78). De celle-ci dérive une expression de la synchronisation pour les types abstraits de M. RAYNAL

(RAY 78), qui s'intéresse à la synchronisation des opérations n-aires sur les types.

D'autres chercheurs pensent qu'il faut, pour certains problèmes, garder une sorte d'historique de l'exécution des programmes : c'est le cas de G. ROUCAIROL (ROU 78) qui utilise des mots de synchronisation. Les langages obtenus ne sont d'ailleurs pas sans rappeler ceux qui sont engendrés par les réseaux de PETRI (PET 77, LAU 75).

Nous avons déjà signalé l'intérêt suscité par les recherches menées à partir des commandes gardées de Dijkstra (DIJ 75) pour exprimer des algorithmes indéterministes : celles de C.A.R. HOARE (HOA 78), de P. BRINCH HANSEN (BRI 78), M. SINTZOFF (SIN 77)...

Exemple 4.8

A titre d'illustration, nous présentons brièvement les concepts imaginés par P. BRINCH HANSEN (BRI 78) pour décrire des systèmes de processus distribués.

Un programme concurrent consiste en un nombre fixé de processus séquentiels qui sont exécutés simultanément. Un processus est défini comme suit :

process nom
variables propres
procédures communes
instruction d'initialisation

Un processus ne peut avoir accès qu'à ses propres variables. Il n'y a pas de variable commune. Mais un processus peut appeler des procédures communes définies à l'intérieur de lui-même ou d'autres processus. Une procédure est définie comme suit :

proc nom (paramètres d'entrée # paramètres de sortie)
variables locales
instruction

Un processus p peut appeler une procédure r définie à l'intérieur d'un autre processus q comme suit :

call q . r (expressions, variables)

L'exécution est contrôlée par des commandes gardées et des régions gardées.

Les commandes gardées ont la syntaxe et la signification suivantes :

if b1 : s1 | b2 : s2 | ... end
do b1 : s1 | b2 : s2 | ... end

- Instruction if : si certaines des conditions b1, b2, ..., sont vraies, alors on sélectionne l'une d'entre elles bi et l'on exécute l'instruction si qui suit ; sinon, on arrête le programme.

- Instruction do : tant que certaines des conditions sont vraies, on sélectionne l'une d'entre elles arbitrairement et on exécute l'instruction correspondante.

Les régions gardées, qui permettent de bloquer des processus, ont la syntaxe et la signification suivante :

when b1 : s1 | b2 : s2 | ... end
cycle b1 : s1 | b2 : s2 | ... end

Une région gardée impose à un processus d'attendre jusqu'à ce que l'état de ses variables rende possible un choix arbitraire entre différentes instructions. Si aucun de ces choix n'est possible dans l'état courant, le processus diffère l'exécution de la région gardée.

- Instruction when : attendre jusqu'à ce qu'une condition soit vraie et exécuter l'instruction correspondante.

- Instruction cycle : répéter indéfiniment l'instruction when.

Si plusieurs conditions sont vraies simultanément à l'intérieur d'une commande ou région gardée, le choix des instructions qu'exécute la machine est imprévisible.

□

Avec ces mécanismes de base, il est possible de décrire la communication de processus ou l'allocation de ressources d'une manière très

agréable. On illustre dans (BRI 78) comment le concept de processus distribués unifie les notions de classes, moniteurs et processus. On montre aussi qu'il inclut bon nombre des outils dont nous avons précédemment parlé : procédures, coroutines, classes, moniteurs, processus, sémaphores, expressions de chemin ...

Pour plaisantes qu'elles soient, ces notions posent beaucoup de problèmes pratiques parmi ceux que nous avons déjà mentionnés pour les autres mécanismes.

Mais c'est ce principe de "rendez-vous" repris de C S P (HOA 78) qui a été retenu dans le langage GREEN - ADA (HON 79 a, HON 79 b) pour exprimer la synchronisation et la communication entre processus. Un mécanisme (wait, signal) et des sémaphores y sont aussi définis de manière standard. Dans le rapport (HON 79 b), une comparaison est faite avec d'autres concepts, parmi ceux dont nous avons parlé.

Ces propositions reposent sur le principe de communications directes entre les processus, chacun devant connaître l'identité de ses interlocuteurs (QUE 79). On peut leur préférer plus d'indéterminisme grâce à des communications indirectes par des "canaux" joignant dans les processus des "ports" étiquetés. C'est une des idées récentes développées par R. MILNER dans une algèbre de comportements de communications (MIL 78). On peut imaginer encore d'autres moyens de communication utilisant la diffusion : notons que l'usage de données partagées peut en être considéré comme une forme (QUE 79).

On constate donc toute la richesse du domaine de l'expression du contrôle et de la communication dans les systèmes. C'est un sujet au coeur des débats en informatique et l'on sent poindre des idées qui devraient à terme permettre de mieux décrire ces phénomènes, notamment d'une manière plus statique.

1.3 Le choix de SIMULA et des moniteurs

Nous justifions ici nos options concernant le langage de description de maquettes et les outils qui permettent d'y exprimer le parallèlisme et la synchronisation.

1.3.1 Langage de description et de simulation

Parmi la quantité des langages présentés en (1.1), il semble difficile de faire un choix. Le nôtre s'est finalement arrêté sur SIMULA 67 qui possède certaines des qualités que nous recherchions (1.1) : c'est un langage de description de systèmes et de simulation à événements discrets, extension d'ALGOL 60, notamment par trois notions : celles de classe, de processus et de liste.

La notion de classe hiérarchisée permet de fabriquer progressivement des composants de plus en plus complexes, selon une démarche modulaire descendante ; elle autorise la réutilisation des classes d'objets existantes dans une démarche ascendante. Cette notion favorise en outre la description de structures de données, telles que celles que nous avons envisagées dans le chapitre I, en regroupant des fonctions, relations et modifications, et en cachant éventuellement leurs représentations concrètes, comme on le fait pour les types abstraits (voir chapitre 6).

La description des processus y est aussi particulièrement simple, grâce à une classe-système spécialisée, la classe process, qui permet de fabriquer des types de processus paramétrés.

Les listes servent à implanter les structures de données ou à réaliser les mécanismes internes de synchronisation.

Pour ce qui est de la simulation proprement dite, SIMULA autorise en fait trois types d'approches que nous avons présentées pour la

simulation discrète :

- approche par événements : chaque procédure associée à un type d'événements, dont nous avons parlé, est réalisée par un processus qui peut être ordonnancé dans l'échéancier standard ;

- approche par processus : c'est l'approche la plus naturelle en SIMULA qui permet les deux vues duales précédemment décrites, selon ce que l'on considère comme entités actives et passives ; c'est cette méthode que nous avons retenue ;

- approche par activités : elle est possible en SIMULA 67 si l'on introduit une primitive d'attente conditionnelle waituntil, avec un mécanisme d'ordonnancement approprié comme l'a montré J. VAUCHER (VAU 73, VAU 78) ; naturellement, on n'utilise pas d'horloges, comme dans les langages fondés sur les activités, mais l'échéancier standard et le noyau de synchronisation de SIMULA.

Ce langage possède en outre quelques fonctions mathématiques et statistiques qui facilient la tache de programmation.

De plus en plus, les systèmes SIMULA permettent de compiler séparément des classes-prologues qui peuvent être cataloguées et réutilisées comme environnements prédéfinis de programmes. C'est le cas notamment pour le nouveau compilateur de la machine IRIS 80 que nous utilisons.

L'utilisation d'autres langages opérationnels présente de nombreuses difficultés. Pour ne parler que des plus courants, G P S S et SIMSCRIPT se prêtent assez mal à l'écriture de programmes où les structures de données manipulées sont complexes. Ils ne favorisent pas la création de nouveaux outils, tels que ceux dont nous aurons besoin. En revanche, leur utilisation s'avère souvent plus efficace quand on désire faire de longues exécutions de programmes pour obtenir des mesures significatives. Mais

pour nous c'est malgré tout un objectif secondaire. On peut, pour couper court, ajouter que l'on ne dispose pas de compilateurs pour ces langages sur IRIS 80.

D'autres langages, tels CONCURRENT PASCAL, SIMONE ou MODULA auraient pu être envisagés. Nous avons expérimenté le premier pour décrire nos premières maquettes (DUF 76) : il s'est révélé mal commode à l'usage, essentiellement pour une question de manque de possibilité de paramétrage des composants et d'impasse pour l'expression de phénomènes tels que la préemption de ressources. Bien qu'un compilateur SIMONE existe sur IRIS 80, nous avons renoncé (provisoirement ?) à l'utiliser.

Enfin, la nouvelle génération des langages fondés sur les expressions de chemins, les commandes et régions gardées... n'a pas encore fourni de produit opérationnel, ce qui est essentiel pour ne pas rester à un niveau purement spéculatif et concrétiser des idées.

Le choix de SIMULA apparaît donc à l'heure actuelle comme celui de la sécurité. Avant de rappeler les notions essentielles de ce langage en (2), nous justifions le choix des moniteurs.

1.3.2 Expresssion du parallèlisme et de la synchronisation

Le choix est rendu difficile par la variété des mécanismes (1.1), surtout qu'un certain nombre d'entre eux peuvent être réalisés facilement en SIMULA 67.

C'est le cas pour les coroutines, représentées par des objets de classes et les instructions detach et resume (DAH 70). C'est le cas des sémaphores, en créant une classe sémaphore et les procédures P et V (VAU 71). C'est le cas aussi pour les moniteurs dont la forme de la déclaration est voisine de celle des classes de SIMULA, et qui peuvent être réalisés grâce à des classes moniteur et condition (BEZ 75), comme nous le ferons par

la suite. Les mécanismes des réseaux de PETRI n'y échappent pas et ils peuvent d'ailleurs être programmés facilement par des moniteurs. Nous avons déjà signalé que l'attente conditionnelle avait été étudiée et implantée en SIMULA (VAU 73, VAU 78).

En revanche, une séparation des procédures et du contrôle comme dans les expressions de chemin est impossible à prendre en compte directement. Il en est bien sûr de même des concepts que nous avons présentés en (1.2.9), notamment les commandes et régions gardées, qui, pour nous ici, restent du domaine du futur.

Ce que nous avons dit dans la première partie de notre travail sur la description logique d'un système devrait nous conduire assez naturellement vers un mécanisme d'attente conditionnelle. En fait, pour avoir une plus grande efficacité et utiliser mieux les possibilités de structuration de SIMULA, c'est probablement vers un compromis attente conditionnelle-moniteur tel celui de J.L.W. KESSELS (KES 77) qu'il faudrait se tourner. Cependant, pour exprimer au niveau physique la concurrence pour l'obtention de ressources, on constate que cet outil est insuffisant, par exemple quand on tient compte de phénomènes de préemption.

Nous nous sommes alors fixés une fois pour toutes les moniteurs comme mécanisme de base. On objectera qu'ils ne permettent pas non plus, dans leur forme originelle, d'allouer des ressources avec une politique de préemption. C'est exact, mais il est possible en SIMULA 67 de tourner la difficulté grâce à des "conditions" particulières, que nous appelons par la suite "avec priorités et délais".

Les justifications données ici peuvent paraître un peu trop "programmatoires". C'est le risque que nous courions en adoptant cet ordre dans la présentation. Par la suite, quand notre modèle aura été exposé, nous aurons d'autres arguments sur la facilité d'utilisation, qui compléterons ces critères techniques.

2 - PRESENTATION du LANGAGE SIMULA 67

Avant d'être un langage de simulation, SIMULA (DAH 66, DAH 70) est un langage de haut niveau, extension d'ALGOL 60. Nous rappelons ici, parmi les concepts particuliers qu'il introduit, ceux qui sont utiles à la compréhension de ce travail : les classes hiérarchisées (2.1) avec les attributs virtuels, les classes-systèmes pour la manipulation de listes (2.2) et la simulation (2.3).

2.1 Les classes hiérarchisées

2.1.1 Déclaration des classes

La déclaration d'une classe SIMULA crée un nouveau type paramétré d'objets tenant à la fois de la structure (COBOL, PL/1, PASCAL ...), de l'instance d'appel de procédure et de la coroutine. En notation de BACKUS généralisée, elle prend à peu près la forme d'une déclaration de procédure ALGOL 60 (CII 72) :

< déclaration de classe > ::= { < préfixe > } < partie principale >

< partie principale > ::= class < nom de classe >
{< partie paramètres formels > ;
{ < partie valeur > }
{< partie spécification >} {< partie virtuelle >}
< corps de classe >

La forme et le rôle de < préfixe > et < partie virtuelle > seront précisés aux paragraphes suivants.

< partie paramètres formels > est la liste parenthésée des paramètres formels de la classe, < partie spécification > en donne le type et < partie valeur > le mode de passage. Nous nous conformons ici à l'implémentation

(CII 72), puisque c'est celle que nous utilisons. Notons simplement qu'il n'est pas possible de passer des procédures en paramètres de classes.

< corps de classe > a la forme d'un bloc ALGOL 60, mais avec la possibilité d'introduire, dans la suite des instructions, la "pseudo-instruction" inner, dont nous reparlerons.

Exemples 4.9

```
La déclaration
class être (nom) ; text nom ;
begin
      integer âge ;
      procédure modifâge (n) ; integer n ;
          âge := âge + n ;
      âge := 0
end
```

crée une classe d'êtres, avec le paramètre formel nom de type text (chaine de caractères) ; le corps de classe déclare l'entier âge, la procédure modifâge de paramètre entier n, et initialise âge à 0.

```
La déclaration
class complexe (x, y) ; real x,y ;
begin
      real module, argument ;
      module := sqrt (x * x + y * y) ;
      argument := arctan (y/x) ;
end
```

crée une classe de nombres complexes (non imaginaires purs) de parties réelles x (nécessairement non nulles),et imaginaires y ; le corps de classe déclare et calcule le module et l'argument d'un nombre complexe.

□

On dit que les paramètres formels, les entités spécifiées dans la partie virtuelle (2.1.7), celles déclarées dans le bloc principal du corps de classe sont les attributs de la classe.

2.1.2 Déclaration d'un pointeur

La déclaration

ref (< nom de classe >) x

crée un pointeur, désigné par x, vers un objet de la classe < nom de classe > selon le schéma de la figure 3 :

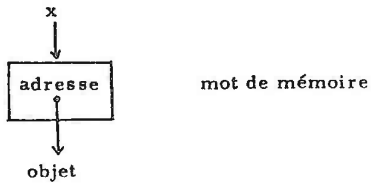


Figure 3 - Repérage d'un objet par un pointeur

Par la déclaration, il n'y a cependant pas création d'un objet, mais seulement d'un pointeur vers un objet vide, d'adresse none, qui appartient à toutes les classes.

Quand il n'y aura pas d'ambiguïté possible, on parlera, par abus de langage, de l'"objet" x, du "pointeur" x, ou de l'"objet référencé par" x.

2.1.3 Création d'objets

Pour créer un objet d'une classe, on utilise (de manière un peu comparable au loc d'ALGOL 68 (WIJ 68, GAA 72), le générateur d'objets new, dans des expressions de la forme :

new < nom de classe > { < partie paramètres effectifs > }

Ceci a pour effet :

- de créer un bloc d'espace-mémoire pour un nouvel objet,
- de réaliser le passage des paramètres conformément aux spécifications de la déclaration de classe,

- d'exécuter le corps de classe pour le nouvel objet ; notons qu'après cette exécution, le corps de classe subsiste.

Exemple 4.10

Avec la déclaration de la classe complexe de l'exemple 4.9 (2.1.1) la création du nombre complexe 2 + 3.5 i s'écrit :

new complexe (2, 3.5)

□

2.1.4 Affectation d'un objet à un pointeur

L'affectation d'un objet à un pointeur est notée :-.

Deux pointeurs différents peuvent désigner le même objet.

Exemple 4.11

En reprenant la déclaration de la classe être de l'exemple 4.9 (2.1.1), on peut écrire :

```
(1) ref (être) x, y ;
(2) x :- new être ('LA BETE DES VOSGES') ;
(3) y :- x ;
(4) x :- none ;
```

(1) déclare 2 pointeurs x et y vers des objets de la classe être ;

(2) crée un objet être et l'affecte au pointeur par x ;

(3) affecte l'objet repéré par le pointeur désigné par x au pointeur désigné par y (il y a "dérepérage" du membre de droite, au sens ALGOL 68) ;

(4) fait pointer sur l'objet vide le pointeur désigné par x, sans rien changer pour y.

Un objet qui, en cours d'exécution, n'est plus référencé par aucun pointeur, disparait et son espace-mémoire sera récupéré par un "ramasse-miettes".

□

2.1.5 Attributs d'un objet ; accès aux attributs

Les attributs d'un objet sont les valeurs pour cet objet des attributs de la classe à laquelle appartient l'objet.

Les attributs d'un objet non vide sont accessibles de l'extérieur de l'objet en utilisant la notation pointée (comme en PL/1 ou PASCAL), par l'intermédiaire d'un identificateur désignant un pointeur vers cet objet.

Exemple 4.12

Après l'exécution de la séquence d'instructions de l'exemple 4.10 (2.1.4) il est possible d'accéder aux attributs de l'objet repéré par le pointeur désigné par y :

y.nom délivre la valeur 'LA BETE DES VOSGES'
y.âge délivre la valeur 0.
L'appel y.modifâge (2) ajoute 2 à y.âge qui prend la valeur 2.

□

On a parfois trouvé abusif d'avoir ainsi accès, de l'extérieur, sans aucune protection, à tous ces attributs. La tendance actuelle est en effet plutôt de cacher à l'utilisateur d'une classe l'organisation interne des objets (types abstraits). Préoccupé par cette question, le "SIMULA development group" a, dès 1975, approuvé une extension du langage appelée "hidden protected specification" (PAL 76). On précise dans l'entête de classe, derrière ces mots-clés, la liste des attributs qui ne sont pas accessibles de l'extérieur, ou (exclusif), derrière not hidden protected, la liste de ceux qui le sont. Ceci répond donc bien au souci manifesté d'accroître la fiabilité et la sécurité de la programmation.

2.1.6 Classes hiérarchisées ; "inner"

On utilise ici la notion de < préfixe > introduite en (2.1.1). En préfixant la déclaration d'une classe b par le nom d'une classe a, on obtient en réalité pour b la "concaténation" des deux descriptions ;

la classe b est dite sous-classe de a.

On note schématiquement la déclaration d'une classe a par :

```
class a (pa) ; va ;
begin
      da ;
      ia ;
      inner ;
      fa
end
```

où pa désigne la liste des paramètres formels, va la partie virtuelle (on omet ici les parties spécification et valeur pour les paramètres formels), da les déclarations du bloc principal, ia les instructions de ce bloc avant inner et fa les instructions après inner.

La déclaration d'une sous-classe b de a s'écrit :

```
a class b (pb) ; vb ;
begin
      db ;
      ib ;
      inner ;
      pb
end
```

Cette déclaration est en fait "équivalente" à la déclaration concaténée :

```
class b (pa, pb) ; va ; vb ;
begin
      da ; db ;
      ia ; ib ;
      inner ;
      fb ; fa
end
```

C'est ce que l'on appelle hiérarchisation ou préfixage.

Un objet de la classe b est créé par un générateur

```
new b (pae, pbe)
```

où pae et pbe désignent les listes de paramètres effectifs correspondant à pa et pb.

Un objet de la classe b appartient aussi à la classe a (l'inverse est faux) et peut être repéré par un pointeur du type ref (a). Parmi les attributs d'un objet de b, on trouve ceux des objets de a (l'inverse est faux). La hiérarchisation peut être réalisée avec plusieurs niveaux de préfixes.

Exemple 4.13

Ainsi, en utilisant comme préfixe le nom de la classe être de l'exemple 4.9 (2.1.1), on peut créer des sous-classes de la manière suivante :

```
être class personne ;
begin
      integer scolarité ;
      ref (personne) conjoint ;
      boolean procedure mineur ; mineur := âge < 18
end ;
être class animal (poids) ; real poids ; null ;
animal class vache ;
begin
      integer litrage
end
```

La hiérarchie peut être schématisée comme suit :

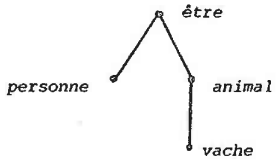


Figure 4.4 - Hiérarchie des classes

Une personne est un être, avec, comme attributs supplémentaires : l'entier (niveau de) scolarité, un pointeur vers un autre objet de la classe personne désigné par conjoint (définition récursive), et la fonction-procédure à résultat booléen mineur.

Un animal a un paramètre formel de plus qu'un être : le réel poids.

Une vache a un attribut interne supplémentaire à un animal : l'entier litrage (de lait par jour).

Notons que null est l'instruction vide.

□

Enfin, il est aussi possible de préfixer un bloc dans un programme SIMULA de la manière suivante :

< préfixe > { < partie paramètres effectifs > } < bloc >

Ceci permet d'utiliser dans le bloc tous les attributs d'un objet de la classe donnée en préfixe, créé avec les paramètres effectifs précisés.

2.1.7 Entités virtuelles

Les entités virtuelles sont déclarées dans un entête de classe (2.1.1) grâce à une expression de la forme

virtual : < spécification >

où < spécification > est une liste de déclarations d'entités virtuelles. Ces entités permettent d'accéder, à un niveau de préfixe donné, à des attributs qui ne seront déclarés qu'à des niveaux de préfixe inférieurs. Celles qui nous intéressent ici sont des (fonctions-) procédures.

Exemple 4.14

La déclaration

```
class nombre ; virtual : real procedure module ; null
```

crée un type nombre avec l'entité virtuelle module. A ce niveau, il est possible d'utiliser cet attribut sans avoir à écrire la fonction-procédure

correspondante, par exemple dans une fonction-procédure qui fait le produit des modules de deux nombres :

```
real procedure prodmod (x, y) ; ref (nombre) x, y ;
    prodmod := x . module * y . module
```

On peut par exemple déclarer deux sous-classes de nombres réels et de nombres complexes, dans lesquels on écrit effectivement le corps de la fonction-procédure module :

```
nombre class réel (a) ; real a ;
begin
    real procedure module ; module := abs (a)
end ;
nombre class complexe (a, b) ; real a, b ;
begin
    real procedure module ; module := sqrt (a * a + b * b)
end
```

L'appel

```
prodmod (new réel (-2), new complexe (-3, 4))
```

délivre alors la valeur 10.

□

On voit donc que, associé à la notion de classes hiérarchisées, le concept d'attribut virtuel permet de réaliser une sorte de programmation abstraite. Ici, on l'envisage à l'intérieur d'un même programme, ce qui ne présente qu'un intérêt de meilleure structuration et d'écriture progressive. Mais dans certains systèmes SIMULA où l'on peut compiler séparément des classes, cela devient un outil extrêmement puissant, comparable à ceux développés dans le cadre des études sur les types abstraits. Nous reviendrons ultérieurement sur ces notions (chapitre 6).

2.2 Les listes ; la classe-système SIMSET

Avec les concepts définis précédemment, il est possible de créer

soi-même des structures de données très diverses, notamment des listes. Mais SIMULA met à la disposition des utilisateurs un environnement particulier, sous la forme de la classe-système précompilée SIMSET, qui permet de créer et manipuler des listes linéaires chainées bilatères (PAI 71).

L'ensemble des propriétés des listes est défini par les trois classes attributs de la classe SIMSET : la classe linkage (propriétés communes aux têtes et liens), la classe head (têtes) et la classe link (liens).

Si une classe ou un bloc a SIMSET pour préfixe, les classes attributs de SIMSET sont alors accessibles à l'intérieur de la classe ou du bloc : on peut utiliser head comme préfixe de classes d'objets qui sont des têtes de listes et link comme préfixe de classes d'objets qui sont des liens.

Nous donnons ici la structure de la classe SIMSET (CII 72) en rappelant simplement les éléments qui nous seront utiles pour la compréhension de la suite :

```
class SIMSET ;
begin
    class linkage ;
    begin
        ...
        ref (link) procedure suc ; ... ;
        ref (link) procedure pred ; ... ;
        ...
    end ; $ linkage $
    linkage class head ;
    begin
        ref (link) procedure first ;... ;
        ref (link) procedure last ; ... ;
```



```
        boolean procedure empty ; ... ;
        integer procedure cardinal ; ... ;
        procedure clear ; ... ;
        ...
    end ; $ head $
    linkage class link ;
    begin
        procedure out ;... ;
        ...
        procedure into (s) ; ref (head) s ; ... ;
    end ; $ link $
end
```

Dans la classe head, les appels des fonctions-procédures first et last rendent respectivement le premier et le dernier lien de la liste ; l'appel de la fonction-procédure empty rend la valeur true si et seulement si la liste est vide, et celui de la procédure clear vide la liste.

Dans la classe link, l'appel de la procédure out fait sortir le lien de la liste à laquelle il appartient (s'il n'appartient à aucune liste, cette procédure est sans effet) ; l'appel de la procédure into a le même rôle que le précédent, puis il place le lien en queue de la liste en paramètre s. Un lien ne peut appartenir qu'à une seule liste au maximum.

Dans la classe linkage, les appels des fonctions-procédures suc et pred rendent respectivement le lien successeur et le lien prédécesseur d'une tête ou d'un lien (si ces liens n'existent pas, elles rendent l'objet vide).

Exemple 4.15

Si l'on veut créer une liste d'êtres tels que ceux de l'exemple 4.9 (1.1.1), on doit d'abord faire de chaque être un lien, grâce à un préfixage de la déclaration de être par link :

```
link class être (nom) ; text nom ; ... ;
```

La liste d'êtres ℓ peut être simplement déclarée par :

ref (head) ℓ

Notons qu'ici, on n'a pas d'assurance que ℓ ne contiendra pas d'autres liens que des êtres.

□

2.3 La simulation ; la classe-système SIMULATION

Pour exécuter des programmes de simulation, l'utilisateur dispose d'un environnement particulier, sous la forme de la classe-système précompilée SIMULATION. Celle-ci est préfixée par SIMSET, si bien qu'en préfixant le bloc de son programme principal par SIMULATION, l'utilisateur peut accéder à tous les attributs de la classe SIMSET, en plus de ceux appartenant spécifiquement à la classe SIMULATION.

L'évolution d'un système dans le temps est simulée par l'exécution en quasi-parallélisme de processus (ou coroutines), selon une technique de simulation dirigée par événements, fondée sur la gestion dynamique d'un échéancier ou SQS ("sequencing set"), comme nous l'avons expliqué en (1.1.1).

A cet effet, à un processus ordonnancé (dans l'échéancier) est associée une date qui correspond au temps effectif où le processus doit (re) commencer son action : cette date est désignée dans la suite par evtime. Les processus ordonnancés sont rangés dans l'échéancier par evtimes croissants.

A un instant donné de l'exécution d'un programme de simulation, on a :

- un processus unique, dit actif (ou courant), qui est le premier processus de l'échéancier, c'est-à-dire celui dont l'evtime est le plus faible ;

- un certain nombre (éventuellement nul) d'autres processus, dits suspendus, qui sont ordonnancés dans l'échéancier ;

- un certain nombre (éventuellement nul) de processus, dits passifs, qui ne figurent pas dans l'échéancier ;

- un certain nombre (éventuellement nul) de processus, dits terminés, dont l'action est terminée et qui ne peuvent plus redevenir actifs ou suspendus.

L'organisation interne de l'échéancier (souvent sous forme d'arbre balancé) n'a pas à être connue du programmeur, qui gère l'activation de ses processus grâce à certaines fonctions et procédures prédéfinies de la classe simulation.

La structure de cette classe est la suivante (CII 72) :

```
SIMSET class SIMULATION ;
begin
    link class process ; ... ;
    ref (process) procedure current ; ... ;
    real procedure time ; ... ;
    procedure hold (t) ; real t ; ... ;
    procedure passivate ; ... ;
    procedure wait (s) ; ref (head) s ; ... ;
    procedure cancel (x) ; ref (process) x ; ... ;
    procedure terminate (x) ; ref (process) x ; ... ;
    ... procédures d'activation de processus ...
    process class mainprogram ; .. ;
    ref (mainprogram) main ;
    main :- new mainprogram ;
    ... initialisation de l'échéancier ...
    ...
end
```

La classe process (voir détail ci-après) est une sous-classe de link : les processus sont donc des liens qui peuvent être rangés dans des files

d'attente (en même temps qu'ils peuvent d'ailleurs figurer dans l'échéancier).

L'appel de la fonction-procédure current rend la référence au premier processus de l'échéancier, c'est-à-dire au processus actif.

L'appel de la fonction-procédure time rend le temps courant de la simulation, c'est-à-dire l'evtime du processus actif.

La procédure hold permet de bloquer le processus qui l'invoque dans l'échéancier pendant un temps t.

L'appel de la procédure passivate sort le processus appelant de l'échéancier et le rend passif.

L'appel de la procédure wait range le processus invocateur dans la file d'attente s et le "passive".

La procédure cancel permet de rendre passif le processus x, s'il ne l'était pas déjà.

L'appel de la procédure passivate termine le processus x.

Différentes procédures d'activation peuvent être aussi utilisées pour ordonnancer un processus : elles sont présentées sous forme d'instructions :

{re} activate x

permet d'activer immédiatement le processus x (s'il est dans l'échéancier, on utilise reactivate) ;

{re} activate x delay t {prior}

permet d'activer x après un temps simulé t : le processus x est rangé après tous ceux de même evtime dans l'échéancier si prior n'est pas précisé, avant sinon ;

{re} activate x at t {prior}

permet d'activer x au temps t simulé ; prior a le même rôle que précédemment ;

{re} activate x (after / before) y

permet d'ordonnancer x juste après (after) ou juste avant (before) un processus y ;

Le programme principal est lui aussi considéré comme un processus que l'on peut ordonnancer : on lui associe en effet un processus désigné par main, objet de la classe de processus mainprogram.

Un processus possède un certain nombre d'attributs regroupés dans la classe process et auxquels on accède par certaines fonctions-procédures. Vue d'un utilisateur, la structure de la classe process est la suivante :

```
link class process ;
begin
    boolean procedure terminated ; ... ;
    boolean procedure idle ; ... ;
    ref (process) procedure nextev ;
    real procedure evtime ; ... ;
    ...
end
```

L'appel de la fonction procédure terminated rend le booléen true si et seulement si le processus est terminé.

L'appel de la fonction-procédure idle rend le booléean true si et seulement si le processus est passif.

L'appel de la fonction-procédure nextev rend la référence au processus suivant dans l'échéancier s'il existe, sinon none.

L'appel de la fonction-procédure evtime rend l'evtime (date d'activation) du processus s'il est ordonnancé, sinon son appel constitue une erreur.

On peut aisément créer une classe paramétrée de processus possédant tous ces attributs grâce au principe de hiérarchisation des classes (2.1.6).

Notons que la création d'un processus par le générateur new ne suffit pas à lancer l'exécution du corps de processus : elle doit être suivie d'une instruction d'activation (activate) au bénéfice de ce processus.

3 - REALISATION des MONITEURS en SIMULA

Pour illustrer ce qui vient d'être dit, et parce que ce mécanisme va nous être très utile par la suite, nous présentons la réalisation en SIMULA 67 des moniteurs de HOARE (HOA 74). Celle-ci a été proposée par J. BEZIVIN dans (BEZ 75) sous la forme de deux classes : monitor et condition. La première peut être écrite :

```
class monitor ;
begin
    boolean busy ;
    ref (head) enterq, urgentq ;
    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
end $ monitor $
```

Le booléen busy est true si et seulement si le moniteur est occupé, c'est-à-dire un processus en a obtenu l'exclusivité.

La file enterq est destinée à faire attendre les processus appelant une (fonction-) procédure du moniteur, alors que celui-ci est occupé.

La file urgentq retient les processus "signalant", c'est-à-dire ceux qui ont fait un signal sur une condition, en libérant effectivement un processus "signalé".

La procédure enter décrit l'entrée d'un processus dans le moniteur : si celui-ci est occupé, le processus se bloque dans enterq ; quand il est débloqué, il occupe le moniteur (il met busy à true).

La procédure leave décrit la sortie d'un processus du moniteur : si la file urgentq est non vide, il débloque un processus (ici le premier) de

cette file ; dans le cas inverse, si la file enterq est non vide, le processus sortant débloque un processus (ici le premier) de cette file sinon il libère le moniteur (il met busy à false).

Ici, la politique de gestion des files est premier arrivé-premier servi (PAPS), mais elle aurait pu être autre, notamment telle que le processus libéré soit purement aléatoire.

Notons que l'on appelle qualification l'expression "qua process" : elle a pour effet de "modifier" (au sens ALGOL 68) le type de l'objet désigné par l'identificateur qualifié, et d'en faire un process, auquel on peut appliquer les procédures d'activation (BIR 73).
La classe condition peut être écrite :

```
class condition (m) ; ref (monitor) m ;
begin
    ref (head) waitq ;
    procedure  cwait ;
    begin
        m . leave ;
        wait (waitq)
    end ; $ cwait $
    procedure  csignal ;
    begin
        if not waitq . empty
        then
            begin
                activate waitq . first qua process after current ;
                waitq  first . out ;
                wait (m . urgentq)
            end
    end ; $ signal $
```



```
    boolean procedure cempty ; cempty := waitq . empty ;
    waitq :- new head ;
end $ condition $
```

Le paramètre formel m de l'entête désigne le moniteur auquel est attachée la condition[1].

La file waitq permet de faire attendre des processus sur la condition.

L'invocation de la procédure cwait[2] par un processus, libère le moniteur (m . leave) puis bloque le processus invocateur dans waitq.

L'appel par un processus de la procédure csignal[2], libère un processus (ici, le premier) en attente sur la condition (processus "signalé"), puis bloque le processus invocateur ("signalant") dans urgentq.

L'appel de la procédure cempty[2] rend la valeur true si et seulement si waitq est vide.

Ici encore, le mode de gestion de la file waitq est PAPS et il aurait pu en être autrement.

Dans (HOA 74), C.A.R. HOARE donne une interprétation du mécanisme de moniteur avec des sémaphores : en utilisant une classe de sémaphores, nous aurions pu utiliser telle quelle cette traduction, mais la programmation est plus immédiate ici.

---

(1) le paramètre m aurait pu être évité en déclarant la classe condition à l'intérieur de la déclaration de la classe monitor. L'écriture adoptée alourdit très légèrement la création des objets condition (voir par exemple la ressourcecritique ci-après), mais présente l'avantage d'autoriser une adjonction plus naturelle de nouvelles classes de conditions, notamment celles avec priorités ou préemption (voir chapitre 6).

(2) La procédure wait est déjà utilisée en SIMULA pour les listes de processus (2.3) : pour éviter les confusions, le wait des conditions est devenu cwait ; de même, signal et empty sont devenus csignal et cempty.

Exemple 4.16

Une classe de ressources critiques telle que celle envisagée dans l'exemple 4.6 (1.2.6) peut être écrite :

```
monitor class ressourcecritique ;
begin
    boolean occupé ;
    ref (condition) nonoccupé ;
    procedure acquérir ;
    begin
        this monitor . enter ;
        if occupé then nonoccupé . cwait ;
        occupé := true
        this monitor . leave
    end ; $ acquérir $
    procedure libérer ;
    begin
        this monitor . enter ;
        occupé := false ;
        nonoccupé . csignal ;
        this monitor . leave
    end ; $ libérer $
    occupé := false ;
    nonoccupé :- new condition (this monitor)
end $ ressourcecritique $
```

Cette expression est plus lourde que celle de (1.2.6), par la nécessité de programmer explicitement l'exclusion mutuelle (enter, leave) et d'attacher la condition au moniteur.

Notons que this monitor est une "auto-référence" à un objet moniteur (BIR 73).

□

Dans un contexte de quasi-parallélisme tel celui de SIMULA, il est bien entendu que l'exclusion mutuelle maintenue sur les procédures du moniteur ressourcecritique est inutile. Elle se justifierait seulement dans le cas où l'exécution de chaque procédure aurait une certaine durée simulée, notamment aléatoire.

On pourrait ainsi souvent faire l'économie de ce mécanisme d'exclusion, mais, pour une plus grande généralité, nous avons préféré le maintenir systématiquement.

Chapitre 5 - ARCHITECTURE et DESCRIPTION GENERALES d'une MAQUETTE de SYSTEME d'INFORMATION

"Wotan :

Vollendet das ewige Werk :
auf Berges Gipfel
die Götterburg
prächtig prahlt
der prangende Bau ! "

(R. Wagner, L'Or du Rhin, acte 1, scène 2)

"Wotan :

Achevée est l'oeuvre éternelle :
là-haut sur les cimes
le château des dieux
resplendit
dans son faste et sa gloire ! "

Chapitre 5 - ARCHITECTURE et DESCRIPTION GENERALES d'une MAQUETTE de SYSTEME d'INFORMATION

1. ARCHITECTURE GENERALE d'une MAQUETTE de SYSTEME d'INFORMATION

1.1 Point de vue logique.

1.1.1 Structures de données.

1.1.2 Protection des structures de données ; moniteurs.

1.1.3 Processus.

1.1.4 Echéanciers.

1.2 Point de vue physique.

1.2.1 Types de ressources.

1.2.2 Modélisation des ressources.

1.2.3 Calendriers de disponibilité.

1.2.4 Description des ressources.

1.3 Exemple de maquette : gestion de commandes-clients.

2. ECRITURE d'une MAQUETTE

2.1 Point de vue logique.

2.1.1 Structures de données et moniteurs.

2.1.2 Processus.

2.1.3 Echéanciers secondaires.

2.2 Point de vue physique.

2.2.1 Ressources non préemptibles et sans calendrier.

2.2.2 Ressources préemptibles ou avec calendrier.

Nous nous préoccupons dans ce travail à la fois de traitements automatiques de l'information (informatique) et de traitements non automatiques (procédures administratives de l'environnement "organisationnel" de l'informatique). Les premiers peuvent être formalisés complètement, alors que les seconds ne le sont que par des modèles plus ou moins approximatifs (chapitre 1).

Cependant, dans une maquette, la manière de les décrire en "réduction" (chapitre 3) ne diffère pas fondamentalement, et c'est plutôt sur un autre critère que nous découpons un système d'information : l'antagonisme logique-physique.

Dans la première partie (1) de ce chapitre, nous présentons les concepts généraux qui permettent de construire des maquettes, successivement des deux points de vue : logique et physique.

Dans la deuxième (2), nous montrons comment ces notions sont décrites en SIMULA pour donner des maquettes programmées opérationnelles.

Cette présentation s'inspire de travaux sur la description de systèmes (C.A.R. HOARE (HOA 75) et P. BRINCH HANSEN (BRI 75)).

1 - ARCHITECTURE GENERALE d'une MAQUETTE de SYSTEME d'INFORMATION

Structurellement, une maquette de système d'information peut être définie comme un système complètement formalisé de processus coopérants, par l'intermédiaire de structures de données partageables, et concurrents, pour l'obtention de ressources physiques.

Nous adoptons successivement deux points de vue :

. logique : description de structures de données, de processus et de la coopération des processus ;

. physique : description de ressources, de leur allocation et de la concurrence entre processus.

Ces notions sont illustrées par un exemple pris dans la gestion d'une entreprise.

1.1 Point de vue logique

Nous décrivons ici des structures de données partageables protégées par des moniteurs, des processus coopérants, ainsi que les conditions d'activation des processus.

1.1.1 Structures de données

La structure de données (ou d'information) globale $\mathcal{S} = (\mathcal{Y}, \mathcal{M})$ de l'organisation est partagée en sous-structures contenant chacune certaines relations, fonctions et modifications élémentaires, régies par des axiomes.

Une de ces sous-structures correspond le plus souvent à une partie de la donnée effectivement enregistrée sur un site à un certain moment.

Certaines d'entre elles, qui se "ressemblent", peuvent être à leur tour considérées comme des objets d'un même type de structures, que les classes de SIMULA 67 peuvent servir à décrire assez facilement.

Le principe de hiérarchisation par préfixage aide à "enrichir" (REM 79) progressivement ces types. De plus, l'utilisation d'attributs virtuels peut permettre de cacher des choix d'implantation physique, qui sont repoussés le plus tard possible, conformément à l'un des principes des types abstraits (chapitre 6).

Dans une maquette, des structures modélisent l'information commune partageable (fichiers manuels, informatiques, ou base de données centralisée ou répartie) du système réel. La sémantique en est réduite, par suppression ou agrégation de types, fonctions ou relations pouvant introduire de l'indéterminisme (chapitre 3). Les simplifications ou regroupements que l'on opère ainsi peuvent conduire à des structures assez différentes, par leur morphologie, de celles du système initial.

Les détails de l'organisation et de la gestion internes à une telle structure de données n'ont en principe pas à être connus de l'extérieur : seules certaines fonctions et procédures locales peuvent être appelées de l'extérieur par des processus pour exécuter des modifications.

1.1.2 Protection des structures de données ; moniteurs

Il est parfois possible d'autoriser que plusieurs processus accèdent ou modifient la donnée d'une structure au même instant. Dans ce cas on n'a pas besoin d'utiliser de mécanisme protecteur.

Mais, souvent, pour les raisons que nous avons déjà soulignées, on doit maintenir une exclusion mutuelle entre certains accès ou modifications de la donnée d'une structure. Dans ce cas, nous avons trouvé commode d'utiliser le mécanisme de moniteur, au sens de C.A.R. HOARE (HOA 74), pour protéger l'accès aux structures de données. Ce mécanisme permet en outre d'organiser le blocage ou le déblocage de processus, suivant l'état de la donnée, grâce aux conditions. Nous supposons connues ces notions présentées au chapitre 4 (1.2.6).

Plusieurs cas peuvent se présenter :

- on désire maintenir une exclusion mutuelle totale entre toutes les fonctions et procédures d'accès ou de modification d'une structure de données, prises deux à deux ; on peut alors utiliser le mécanisme d'exclusion des moniteurs, en "englobant" la structure dans un moniteur, c'est-à-dire

en considérant que la structure fait partie du moniteur (figure 5.1) ;

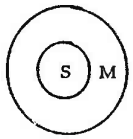


Figure 5.1 - Moniteur (M) englobant une structure de données (S)

par la suite, nous ne représentons plus la structure à l'intérieur du moniteur.

- on n'impose qu'une exclusion mutelle partielle entre certaines fonctions ou procédures d'accès ou de modification ; on sépare alors nettement la structure à protéger partiellement du mécanisme protecteur, le moniteur (figure 5.2) ; celui-ci est consulté avant toute tentative d'accès ou de modification à la structure par un processus, qui obtient l'autorisation de passer ou reste bloqué ; le moniteur est encore consulté après le travail du processus dans la structure pour libérer des processus éventuellement en attente.

Figure 5.2 - Moniteur (M) protégeant une structure de données (S)

Exemple 5.1

C'est ainsi que l'on résout le classique problème des "lecteurs et écrivains" avec un moniteur. Des processus "lecteurs" en nombre quelconque peuvent accéder simultanément au même enregistrement d'un fichier, mais un processus "écrivain" qui met à jour cet enregistrement doit y avoir un

accès exclusif. Pour écarter la possibilité de "famine" d'un écrivain, on impose qu'un nouveau lecteur ne puisse pas commencer à travailler sur l'enregistrement s'il y a un écrivain qui attend. De même, pour éviter la "coalition" des écrivains contre les lecteurs, tous les lecteurs qui attendent la fin d'une écriture ont priorité sur l'écrivain suivant. La solution de ce problème est présentée dans (HOA 74), avec un moniteur à deux conditions (OKtoread, OKtowrite) et quatre procédures externes (startread, endread, startwrite, endwrite).

□

Exemple 5.2

La figure (5.3) montre un petit modèle extrait d'une maquette de gestion de commandes-clients et de facturation d'une entreprise (voir exemple 5.5 (1.3)).

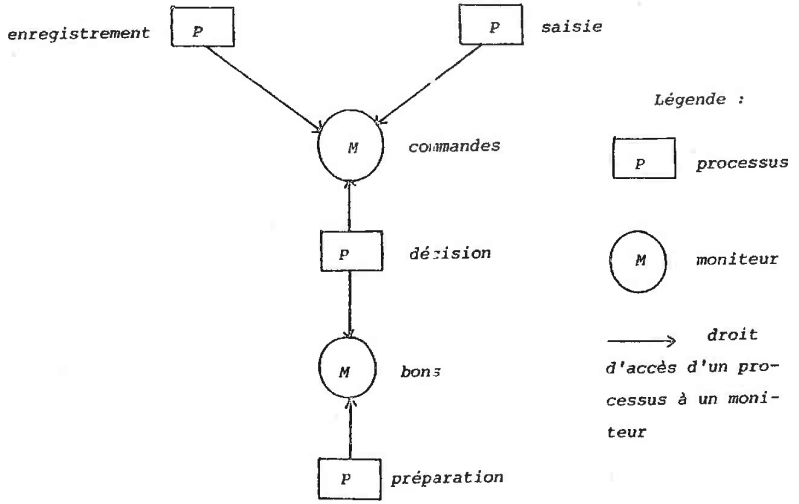


Figure 5.3 - *Partage de structures de données englobées dans des moniteurs.*

Trois processus, enregistrement, saisie et décision, transforment la structure commandes, en exclusion mutuelle, par adjonction, suppression ou modification de commandes. Il n'y a pas ici de condition de blocage supplémentaire.

Le processus décision inscrit en outre dans la structure bons des ordres de livraison qui sont utilisés par le processus préparation, selon un schéma classique de producteur-consommateur : le tampon étant ici supposé infini, la seule condition de blocage pour le processus préparation est que le tampon soit vide.

□

Plusieurs moniteurs qui se ressemblent peuvent être considérés comme objets d'un même type de moniteurs, éventuellement paramétré. C'est par exemple le cas des tampons infinis (ou files d'attente) qui reviennent couramment dans les applications et que l'on considère comme des objets d'un type de files.

1.1.3 Processus

Un processus est constitué d'une structure de données propre et d'un programme séquentiel. Celui-ci peut opérer sur les données privées du processus, sur les données communes, éventuellement protégées par des moniteurs, mais pas sur les données privées d'un autre processus.

Dans une maquette, ils modélisent les traitements d'information automatiques, manuels et intellectuels, du système d'information, comme l'a montré l'exemple de la figure 5.3.

Plusieurs processus peuvent être du même type, c'est-à-dire issus du même texte de programme paramétré.

Nous avons déjà vu deux causes de blocage de processus : l'exclusion mutuelle d'accès ou l'arrêt sur condition dans un moniteur. En gestion, il

est courant de vouloir déclencher un processus à certaines dates, d'où un concept d'échéancier (1.1.4).

Une question importante est de savoir s'il convient de considérer un nombre de processus fixé à l'avance, ou de s'autoriser à en créer et à en supprimer dynamiquement. Pour une question de simplicité et de sécurité, nous penchons plutôt vers la première solution, en considérant souvent des processus cycliques, c'est-à-dire répétant indéfiniment la même séquence d'instructions et se bloquant de temps à autre dans des moniteurs. Nous n'interdisons cependant pas la génération et la destruction dynamiques de processus quand c'est absolument nécessaire : SIMULA 67 le permet sans peine grâce au générateur new et à la procédure terminate (voir 4.2.3). Mais il faut bien voir que ceci correspond la plupart du temps aux deux approches duales de la modélisation par processus (5.1.1), l'une consistant à prendre comme processus les serveurs (ce qui est plutôt notre façon de voir) et l'autre les clients. Parfois il est nécessaire de combiner les deux approches dans la même maquette, et quelquefois, le choix de l'une ou de l'autre dépend du niveau de détail considéré (chapitres 7 et 9).

1.1.4 Echéanciers

Conceptuellement, un échéancier (secondaire) contient une liste de couples (date d'activation, référence à un moniteur) rangés par dates d'activation croissantes et gérés dynamiquement, ainsi qu'un processus scrutateur de cette liste. Quand le temps courant devient égal à une date d'activation, le processus scrutateur déclenche une procédure du moniteur associé, qui peut avoir pour effet, par un signal, de libérer un processus bloqué sur condition dans le moniteur.

Nous parlons ici d'échéancier secondaire parce que ce mécanisme complète celui d'échéancier standard de SIMULA 67 (chapitre 4 (2.3)).

Naturellement, l'organisation interne d'un échéancier est complètement cachée à l'utilisateur qui n'y accède que par certaines fonctions et procédures. On voit donc ici que, pour déclencher des processus à des instants déterminés, on n'utilise à nouveau que le principe des moniteurs.

Exemple 5.3

La figure 5.4 illustre la notion d'échéancier.

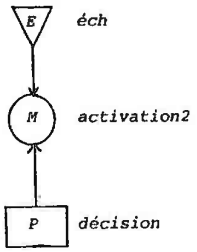


Figure 5.4 - Echéancier

Le processus décision (de livraison) peut être libéré par l'intermédiaire du moniteur activation2, à certaines dates précisées dans l'échéancier éch. En utilisant ce même moniteur, il peut l'être aussi par un autre processus. Ceci permet d'utiliser les moniteurs comme seul moyen de synchronisation.

□

Pour tenir compte d'autres blocages de processus, il nous faut nous placer au niveau physique.

1.2 Point de vue physique

Nous considérons successivement les différents types de ressources physiques retenues dans les maquettes, leur modélisation et leur description.

1.2.1 Types de ressources

On distingue des ressources actives et des ressources passives.

- Les ressources actives, ou processeurs, sont celles qui modélisent les agents d'exécution de tout ou partie des processus. Selon le niveau de détail considéré, elles représentent :

. pour la partie automatique : des ordinateurs, des réseaux d'ordinateurs, de micro-processeurs... ;

. pour la partie non automatique : des personnes ou des services de l'organisation (ateliers, entrepôts, services administratifs...), uniquement pour l'aspect traitement de l'information.

Naturellement, le sous-système de traitement de l'information est lié aux autres (production, finances, gestion du personnel, du matériel...), particulièrement pour leur comportement dynamique : c'est ce qu'avait bien vu FORRESTER (FOR 61) en proposant son modèle des six flux interactifs. Ici, nous voulons ignorer au maximum les interconnexions entre les sous-systèmes, ou plutôt, les intégrer globalement soit sous forme de processus parasites, soit en augmentant les temps de traitement par des temps de latence ("overheads") plus importants.

- Les ressources passives représentent des moyens auxiliaires, consommables ou réutilisables, dont ont besoin les processus (ou les processeurs) pour pouvoir se dérouler : mémoires, papier, matériel de bureau...

Pour nous qui regardons souvent les choses à un niveau assez global, leur prise en compte n'affecte le plus souvent pas la dynamique des maquettes. Il s'agit ici beaucoup plus de chercher à établir une sorte de comptabilité de certains moyens utilisés, en supposant qu'ils sont toujours en quantité suffisante. Il nous paraît en effet raisonnable de considérer qu'il n'y a pas blocage par manque de telles ressources, ce qui nous entraînerait

dans des considérations secondaires ou de trop bas niveau. D'ailleurs, l'intégration de ces moyens pour la dynamique a dû se faire déjà à un niveau plus élevé, au moment où l'on s'interroge sur les performances des processeurs considérés globalement et incluant ces ressources passives.

1.2.2 Modélisation des ressources

Dans une maquette, les ressources sont toujours gérées indépendamment les unes des autres, de manière décentralisée (chapitre 3).

Ceci peut paraître une simplification abusive, mais nous avons déjà signalé que les expériences ne manquent pas d'évaluations quantitatives de systèmes réalisées convenablement par des modèles à gestion de ressources décentralisée, comme des réseaux de files d'attente (KLE 75, 76).

Qu'elles soient actives ou passives, les ressources sont considérées de la même manière par les processus. Pour reprendre la terminologie des files d'attente, les ressources sont des serveurs, demandés par les processus qui sont des clients, et gérés indépendamment dans des stations de différents types :

- files d'attente à un serveur (ou ressources critiques), avec ou sans priorités, préemptibles ou non ;
- files d'attente à plusieurs serveurs (ou ressources à accès multiples), avec les mêmes caractéristiques ;
- processeurs partagés, par exemple gérés selon la technique du "tourniquet" ;
- ...

Toutes sortes de politiques de gestion de files d'attente sont évidemment envisageables (KLE 75, 76). Une des difficultés essentielles est de

choisir parmi cette immense variété les quelques modèles qui permettent de bien représenter l'allocation des ressources dans les systèmes d'information (chapitres 6 et 7).

Mais les ressources d'un système d'information ne sont pas toujours disponibles pour lui : on fixe alors dans un calendrier les périodes pendant lesquelles une ressource peut être délivrée à un processus de la maquette correspondante.

1.2.3 Calendriers de disponibilité

Conceptuellement, le calendrier attaché à une ressource contient une liste de couples (dates de début et fin d'intervalles de disponibilité) rangés par dates de début croissantes et gérés dynamiquement, ainsi que deux processus scrutateurs de cette liste. Un processus "activateur" rend la ressource disponible à la date de début d'un intervalle de disponibilité, alors qu'un processus "désactivateur" bloque l'usage de cette ressource à la date de fin. Ceci peut avoir pour effet de bloquer des processus utilisateurs, par une sorte de préemption.

1.2.4 Description des ressources

D'un point de vue conceptuel, les données partagées (1.1.1) peuvent être considérées comme des ressources. On a alors envie d'utiliser pour celles-ci le même mécanisme protecteur et allocateur que pour celles-là : les moniteurs.

S'il s'agit de décrire l'allocation de ressources non préemptibles (avec ou sans priorité) et sans calendriers, ceci est très facile en utilisant la notion standard de moniteur et de condition.

Dans le cas contraire cette notion est insuffisante, mais, sous certaines hypothèses que nous verrons (chapitre 6), il est possible de

tourner la difficulté grâce à l'introduction de conditions particulières, avec priorités et délais.

Ceci permet d'affirmer que tous les types de ressources introduits sont gérés par des moniteurs, et unifie la description des systèmes d'information.

Un processus qui doit utiliser une ressource la demande par l'appel d'une procédure d'acquisition externe du moniteur associé. Pour la rendre, il fait appel à une procédure de libération, dans des conditions similaires.

Exemple 5.4

La figure 5.5 illustre les utilisations successives, par le processus saisie, du processus ord et du "processeur" y, une personne.

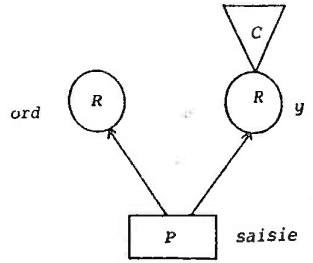


Figure 5.5 - Usage de ressources physiques

Le processus y est par exemple une ressource critique préemptible (saisie possède une priorité), dont le temps de disponibilité est fixé par un calendrier. Le processeur ord est par exemple une ressource à "nombre infini de points d'accès", utilisée uniquement à des fins de comptabilité d'utilisation. □

Souvent, à cause de problèmes de verrouillage trop importants, on impose qu'un processus ne puisse utiliser un processeur, sauf cas particulier, qu'en dehors de tout risque de blocage étranger au processeur.

Dans l'exemple ci-dessus, il n'y a pas de risque de cette sorte pour l'utilisation de y, en raison de la nature de la ressource ordinateur. En revanche, au cas où y serait préempté, il y a risque de comptabiliser l'utilisation de ord de manière abusive, en attendant le retour de l'usage de y. Il faudrait alors, pour s'en sortir, un mécanisme d'allocation plus global. Nous ne le souhaitons pas et préférons imposer, au risque de nous éloigner un peu de la réalité ou de compliquer les mesures, la restriction énoncée précédemmert.

1.3 Exemple (5.5) de maquette : gestion de commandes-clients

Pour illustrer les notions précédentes, nous présentons la structure d'une petite maquette de système d'information (figure 5.6). Elle représente, à échelle réduite, la gestion des commandes-clients d'une entreprise commerciale que nous avons étudiée. Les exemples précédents en sont extraits et l'on trouvera dans (KLE 79) une maquette plus complète qui la contient, avec quelques variantes.

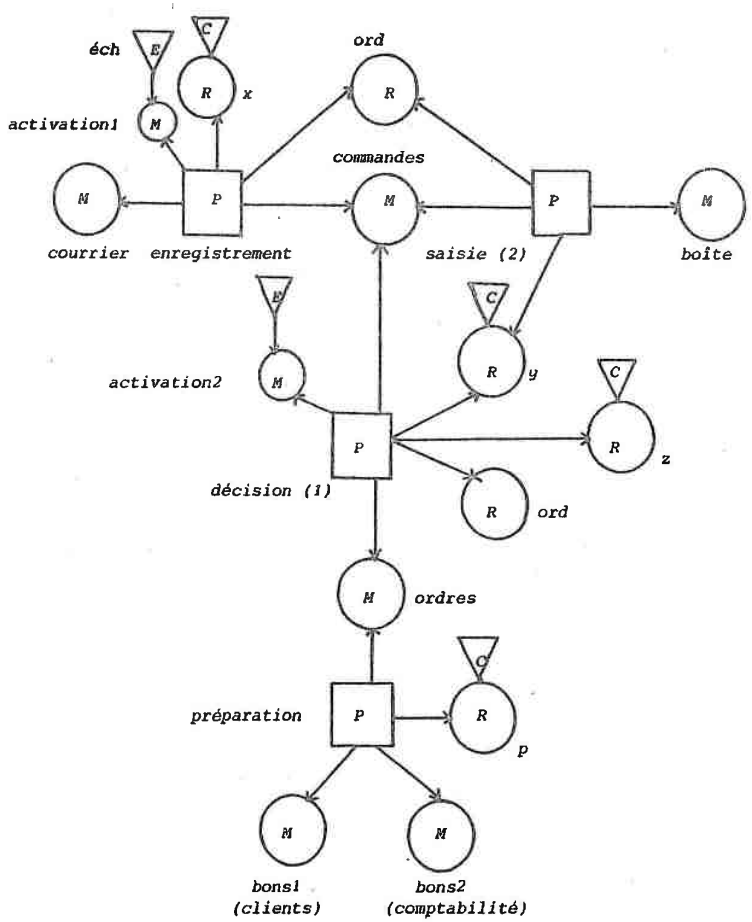
Le processus enregistrement puise des lettres dans la structure courrier et met à jour la structure commandes. Il est déclenché journellement à des heures précisées dans l'échéancier éch ; par l'intermédiaire d'une structure[1] activation1 Il est exécuté par une personne x et un ordinateur ord.

Le processus saisie répond à des appels téléphoniques qui parviennent dans la structure boite et apporte immédiatement des modifications dans la structure commandes, grâce à une console et selon un mode conversationnel. Son déclenchement se produit aléatoirement à chaque appel téléphonique concernant les commandes. Il est exécuté par une personne y et par l'ordinateur ord.

Le processus décision inscrit des ordres de liaison dans la structure ordres, à partir de la structure commandes qu'il met à jour. Il est déclenché tous les jours à une heure précisée par l'échéancier éch, grâce

---

(1) Il s'agit ici d'un "moniteur d'activation" (voir chapitre 6 (1.2.2)).



Légende :

- S — structure de données
- E — échéancier (secondaire)
- M — moniteur protégeant une structure de données
- R — moniteur protégeant une ressource physique
- P — processus
- C — calendrier associé à une ressource physique
- ⟶ droit d'accès d'un processus à une structure de données ou à un moniteur

Figure 5.6 - Architecture générale d'une maquette

à la structure[1] activation2. Il est exécuté par deux personnes, y et z, et par l'ordinateur ord.

Le processus préparation récupère ces ordres de livraison, prépare la marchandise, complète les ordres et les scinde en deux parties. L'une est placée dans la structure bons1 destinée à l'envoi au client, l'autre dans la structure bons2 destinée à la comptabilité. Il est exécuté par une personne p.

Tous ces processus sont cycliques et le détail plus ou moins complexe de leur fonctionnement sera décrit par la suite.

Les structures de données sont toutes supposées globalement accédées ou modifiées en exclusion mutuelle, et incluses dans des moniteurs. Leur détail sera donné ultérieurement.

Comme nous l'avons déjà dit, l'ordinateur ord est vu comme une ressource à nombre infini de points d'accès.

Les personnes x, z et p sont considérées ici comme des ressources critiques non préemptibles, protégées chacune par un calendrier qui fixe leurs périodes de disponibilité.

En revanche la personne y est représentée comme une ressource critique préemptible avec calendrier. Deux processus de la maquette peuvent la demander : saisie et décision. Il est donc nécessaire de leur affecter des priorités : celles-ci sont respectivement 2 et 1. Saisie peut éventuellement préempter y utilisée par décision, ce qui correspond à une réponse immédiate à un appel téléphonique.

Ceci est une des modélisations possibles du système de gestion de commandes. En effet, pour un objectif et un niveau de détail déterminés, il existe plusieurs manières de concevoir des maquettes avec des propriétés identiques.

□

(1) Il s'agit ici d'un "moniteur d'activation" (voir chapitre 6 (1.2.2)).

2 - ECRITURE d'une MAQUETTE

Nous donnerons dans le chapitre suivant tous les détails nécessaires à l'écriture d'une maquette. Ici, nous voulons simplement montrer comment on structure globalement une description en SIMULA 67 à l'aide de composants standard, que nous présentons plus précisément au chapitre 6, ou de composants propres au système que l'on modélise. La présentation qui suit reprend le même ordre et les mêmes exemples que précédemment.

2.1 Point de vue logique

2.1.1 Structures de données et moniteurs

Les classes du langage SIMULA 67 permettent de décrire des types de structures de données paramétrés. Nous discuterons plus tard de la méthode de structuration qui conduit à ces descriptions, ne considérant ici que son résultat : des classes paramétrées représentant les structures de données d'un système d'information.

Ces structures sont ou non englobées dans un moniteur, comme nous l'avons fait remarquer précédemment.

Exemple 5.6

Pour le modèle de l'exemple 5.5, on peut créer une classe de moniteurs ccommandes par une déclaration dont la structure est :

monitor class ccommandes ; begin ... end

La déclaration de commandes est alors :

ref (ccommandes) commandes

La création de la structure associée et son affectation s'écrivent :

commandes :- new commandes

□

2.1.2 Processus

La classe process de la classe-système SIMULATION est utilisée pour déclarer par préfixage des classes de processus paramétrées. Ces paramètres peuvent être de natures diverses selon les applications considérées, mais on retrouve couramment des attributs permettant d'obtenir la priorité ou le temps d'exécution des processus.

Exemple 5.7

Dans notre modèle, on peut créer une classe de processus cenregistrement, que nous supposons non paramétrée, par une déclaration de la forme :

process class cenregistrement ; begin ... end

La déclaration du processus enregistrement est alors :

ref (cenregistrement) enregistrement

La création effective d'un tel processus et son affectation s'écrivent :

enregistrement :- new cenregistrement

Rappelons que, pour être effectivement exécuté, un processus doit être obligatoirement (re-) lancé par une instruction activate (chapitre 4).

□

2.1.3 Echéanciers secondaires

Ce mécanisme est réalisé grâce à une classe standard échéancier où sont notamment déclarés la liste de couples (date d'activation, référence à moniteur) et le processus scrutateur, dont nous avons parlé précédemment.

Plusieurs échéanciers, avec des caractéristiques distinctes, permettent des périodicités différentes pour certains déclenchements (cf. chapitre 6).

Exemple 5.8

Dans notre modèle, la déclaration de l'échéancier éch s'écrit :

ref (échéancier) éch.

Sa création, avec l'initialisation de sa liste de couples par lecture, est réalisée par une instruction de la forme :

éch :- new échéancier (... paramètres effectifs ...)

□

2.2 Point de vue physique

2.2.1 Ressources non préemptibles et sans calendrier

Qu'elles soient de n'importe lequel des types que nous avons précédemment cités (actives ou passives), les ressources non préemptibles et sans calendrier peuvent être contrôlées par des moniteurs identiques à ceux que nous avons présentés. Ordinairement, la gestion des files d'attente y est réalisée avec la politique premier arrivé - premier servi. Il est cependant très facile d'avoir une politique de gestion avec priorités, comme celle qui a été présentée dans (HOA 74).

Ainsi, des classes de stations (ou de ressources) standard ont été déclarées et sont à la disposition des utilisateurs dans une classe précompilée (2.3).

2.2.2 Ressources préemptibles ou avec calendrier

Ici, la question est plus délicate, car il faut se donner le

moyen d'interrompre un (ou plusieurs) processus qui ont l'usage d'une telle ressource. En cas de préemption, cette interruption est due à un processus plus prioritaire que celui (ou ceux) qui ont l'usage de la ressource. En cas de fin de période de disponibilité, l'interruption est réalisée par le mécanisme de calendrier.

Il est donc nécessaire de conserver la liste des processus qui ont l'usage d'une telle ressource.

Au moment de la reprise (restitution de la ressource aux processus interrompus dans le cas de préemption, ou début de période de disponibilité dans le cas d'un calendrier), il faut pouvoir redonner la ressource aux processus interrompus. Dans le cas d'une simulation comme nous la pratiquons, il faut avoir gardé le temps résiduel d'utilisation de la ressource, pour pouvoir faire une reprise au point d'interruption ("preemptive-resume discipline" (KLE 76)).

On conçoit alors la nécessité de référencer les processus, d'altérer un peu le mécanisme standard de moniteur et d'avoir des facilités pour gérer l'échéancier standard du langage de simulation. Ceci n'est pas possible avec les langages courants qui possèdent la notion de moniteur (CONCURRENT PASCAL, SIMONE, MODULA, déjà cités), mais l'est avec le langage SIMULA 67.

Ainsi a été créée une classe particulière de conditions avec priorités et délais, la classe conditionp, qui permet de réaliser la préemption (chapitre 6).

2.2.3 Calendriers

Une station possédant des périodes de disponibilité est globalement contenue dans un objet appartenant à la classe calendrier, elle-même sous-classe de la classe monitor, par une déclaration de la forme :

monitor class calendrier ...

Un tel objet contient une liste de paires de bornes d'intervalles de disponibilité, une liste de références aux processus ayant traversé le calendrier, et une condition sur laquelle sont bloqués les nouveaux arrivants aux instants d'inactivité. Deux processus internes scrutateurs lancent ou bloquent les processus aux début et fin de chaque intervalle de temps.

2.2.4 Ressources

Les ressources sont regroupées dans des stations qui appartiennent à des sous-classes de la classe calendrier (donc de monitor). Il est cependant possible d'inhiber ce mécanisme en valorisant convenablement un paramètre à la création des objets.

Un certain nombre de classes standard de ressources ont été écrites et sont disponibles pour les utilisateurs : ressources critiques ou à accès multiples, avec priorités ou non, avec préemption ou non, processeurs partagés...

D'autres peuvent être écrites assez facilement pour des besoins particuliers à partir des outils de base disponibles.

Exemple 5.9

La classe standard calrcp de ressources critiques préemptibles, et protégées par un calendrier, a un entête de la forme :

calendrier class calrcp

Le processeur y du modèle (exemple 5.5) est alors déclaré :

ref (calrcp) y

Sa création, avec l'initialisation de sa liste de paires de bornes d'intervalles de disponibilité par lecture, est réalisée par :

y :- new calrcp (... paramètres effectifs ...)

□

2.3 La classe MAESTRO ; structure d'un programme-maquette

2.3.1 La classe MAESTRO

Tous les mécanismes standard du système logiciel que nous venons de présenter sont regroupés dans une classe unique, la classe MAESTRO. Ils sont complétés par des outils de mise au point et de mesure pour l'analyse de comportement dynamique, qui figurent eux-aussi dans cette classe et que nous présenterons ultérieurement (chapitres 8, 9, annexe C).

La classe MAESTRO est compilée et cataloguée sur disque, afin de servir de prologue de programmathèque pour les programmes utilisateurs. Elle est préfixée par la classe SIMULATION puisqu'elle en utilise les attributs. Ceux-ci sont donc aussi disponibles pour l'utilisateur de la classe MAESTRO. Sa structure est la suivante :

```
SIMULATION class MAESTRO ;
begin
    ...
    déclaration des mécanismes :
    de moniteur,
    de condition simple ou avec priorités et délais,
    d'échéancier,
    de calendrier,
    de ressources ;
    ...
    déclarations d'autres classes et procédures visibles
    ...
end
```

2.3.2 Ecriture d'un programme-maquette

Pour avoir accès aux outils de la classe MAESTRO, un programme utilisateur doit avoir son bloc principal (au 2ème niveau) préfixé par MAESTRO :

```
begin
    MAESTRO begin
    < corps du programme utilisateur >
    end
end #
```

Exemple 5.10

Ainsi, le programme correspondant à la maquette de l'exemple 5.5 (1.3) a la structure suivante :

```
begin
    MAESTRO begin
        ...
        monitor class ccommandes ; ... ;
        ... autres déclarations de classes de moniteurs ...
        process class cenregistrement ; ... ;
        ... autres déclarations de classes de processus ...
        ---------------------------------------------------
        ref (ccommandes) commandes ;
        ... autres déclarations de moniteurs ...
        ref (cenregistrement) enregistrement ;
        ... autres déclarations de processus ...
        ---------------------------------------------------
        ref (échéancier) éch ;
        ---------------------------------------------------
        ref (calrcp) y ;
        ... autres déclarations de ressources ...
        ---------------------------------------------------
```



```
        ...
        commandes :- new ccommandes ;
        ...
        enregistrement :- new cenregistrement ;
        ...
        éch :- new échéancier ... ;
        ...
        y :- new calrcp ... ;
        ... autres créations d'objets ...
        ...
        ---------------------------------------------------
        ...
        activate enregistrement qua process ;
        ... autres initialisations ...
        ...
    end
end #
```

Nous donnons par la suite plus de détails sur cet exemple ; le texte complet de cette maquette est donné en annexe B.

□

Chapitre 6 - ANALYSE des COMPOSANTS d'une MAQUETTE

"Le Choeur :

Et que les marteaux
Pelles et ciseaux
Achèvent le reste
de nos travaux "

(H. Berlioz, Benvenuto Cellini, acte 2, scène 18)

Chapitre 6 - ANALYSE des COMPOSANTS d'une MAQUETTE

1. POINT de VUE LOGIQUE.

   1.1 Structures de données.

      1.1.1 Approche pragmatique.

      1.1.2 Approche systématique.

         a) Modèle relationnel et langage algébrique.

         b) Réalisation d'un modèle relationnel abstrait en SIMULA 67.

         c) Implantation des relations.

         d) Exemple d'utilisation.

         e) Prologue standard.

         f) Liaisons avec les types abstraits.

         g) Conclusion.

   1.2 Mécanismes de protection et de blocage.

      1.2.1 Files d'attente.

      1.2.2 Moniteurs d'activation.

   1.3 Processus ; durée de vie des processus.

   1.4 Echéanciers secondaires.

2. POINT de VUE PHYSIQUE.

   2.1 Compléments sur les moniteurs ; conditions avec priorités et délais.

   2.2 Calendriers de disponibilité de ressources.

   2.3 Différents types de stations (ou ressources).

      2.3.1 Ressources critiques simples : stations à un serveur avec une file PAPS.

2.3.2 Ressources critiques avec priorités : stations à un serveur avec une file à priorités.

2.3.3 Ressources à accès multiples : stations à n serveurs avec une seule file.

2.3.4 Ressource à nombre infini de points d'accès : stations à nombre de serveurs infini.

2.3.5 Ressources critiques avec priorités préemptibles : stations à un serveur et à une file avec priorités et préemption.

2.3.6 Processeurs partagés : stations avec un serveur partagé selon la méthode du tourniquet.

3. STRUCTURE de la CLASSE MAESTRO.

Ce chapitre a pour but de présenter en détail les composants qui interviennent dans la conception et l'écriture de maquettes.

Certains d'entre eux sont standard : ils font partir du prologue MAESTRO et peuvent être utilisés tels quels dans des maquettes. D'autres doivent être fabriqués en fonction du problème à résoudre : nous tentons d'esquisser pour eux une méthodologie de construction qui s'appuie sur les concepts de SIMULA 67 et sur des outils généraux que nous proposons.

Souvent, aux composants standard sont melés des éléments qui servent à collecter des mesures pour l'analyse statistique. Dans ce chapitre, nous en faisons abstraction, à l'exception de certains paramètres, pour considérer uniquement les mécanismes propres à la description de systèmes d'information. Les outils de mesure feront l'objet du chapitre 9.

La présentation nous conduit à nouveau à distinguer les aspects logiques (1) et physiques (2) d'un système. Nous résumons ensuite (3) les composants standard, avec la structure de la classe MAESTRO. Le texte complet de cette classe est donné en annexe C.

1. POINT de VUE LOGIQUE

On montre ici comment décrire des structures de données (1.1), leurs mécanismes de protection et de blocage (1.2), des processus et leurs moyens d'activation (1.3), notamment par des échéanciers (1.4).

1.1 Structures de données

Il existe deux manières d'approcher l'implantation des structures de données au moyen de SIMULA 67. L'une, que nous disons pragmatique,

consiste à choisir directement un moyen de regrouper et de représenter relations, fonctions et modifications, d'une manière tout à fait "classique". L'autre, que nous disons systématique, veut conduire à plus de méthode pour effectuer cette implantation, à travers des "niveaux d'abstraction".

La deuxième approche est probablement celle qui aura notre faveur à l'avenir, notamment à cause du développement des types abstraits : nous montrons d'ailleurs comment réaliser progressivement et à échelle réduite un modèle relationnel de données semblable à celui de CODD. Mais, il nous faut dire que toutes les implantations de structures que nous avons réalisées à l'heure actuelle dans des maquettes l'ont été directement avec la première approche.

1.1.1 Approche pragmatique

On conçoit qu'il n'y ait pas grand'chose à dire de cette approche où l'on tente naturellement de "faire pour le mieux". La plus simple est sans doute de montrer sur un exemple comment implanter directement une structure. Rappelons seulement que, conformément à ce qui a été dit sur les simplifications au chapitre 3, toute relation n-aire se ramène à une relation unaire avec introduction de nouvelles fonctions.

Exemple 6.1

Nous reprenons l'exemple 1.1 (chapitre 1), tout d'abord pour ce qui concerne les fonctions et les relations.

Chaque ligne (de commande) et ses images par certaines fonctions définies dans l'ensemble de base ligne sont représentées par un objet d'une classe, que nous appelons aussi ligne, et qui est un lien (préfixage par link) :

```
link class ligne ;
begin
    ref (produit) p ;
    integer nl ;
    real q ;
    ref (commande) c
end
```


produit et commande désignent aussi des classes de produits et de commandes ; de même client désigne une classe de clients.

La relation rl, les fonctions 1er élément tl et successeur s, sont réalisées par une liste nl et des fonctions-procédures :

```
ref (head) rl ;
ref (ligne) procedure tl ; tl :- rl . first qua ligne ;
ref (ligne) procedure s (x) ; ref (ligne) x ;
    s :- x . suc qua ligne ;
```

La classe produit peut être écrite :

```
link class produit ;
begin
    integer np ;
    real    st ;
    text    car
end
```

La relation rp est par exemple réalisée par une liste d'objets de la classe produit :

```
    ref (head) rp
```

La classe commande est déclarée comme suit :

```
class commande ;
begin
    integer nc ;
    ref (client) cl ;
    ref (ligne) l
end
```

La relation rc et la fonction cde sont par exemple réalisées comme un tableau d'objets de la classe commande (on suppose qu'il y en a 1 000 au maximum) :

```
ref (commande) array (1 : 1000) cde
```

Il faut se souvenir que rc désigne en fait l'ensemble de toutes les commandes, c'est-à-dire des éléments du tableau qui sont "pertinents" (ce terme étant à définir). Une implantation plus réaliste serait sans doute faite avec l'utilisation d'une fonction de "hash-coding".

De même, on peut déclarer une classe client :

```
class client ;
begin
      integer ncl, sir ;
      text ad ;
      real solde
end
```

La relation rcl et la fonction clt peuvent être réalisées d'une manière analogue à la relation rc et à la fonction cde, par exemple par le tableau (où l'on suppose qu'il y a 100 clients au maximum) :

```
ref (client) array (1 : 100) clt
```

On peut faire sur cette implantation les mêmes remarques que sur celle des commandes ci-dessus.

D'après ce que nous avons rappelé du chapitre 3, la relation qté doit donner lieu à la déclaration d'une classe a :

```
link class a ;
begin
      ref (produit) pro ;
      ref (client) cli ;
      real qv
end
```

et être représentée par une liste :

```
ref (head) qté
```

□

Dans ces descriptions, les axiomes permanents, ou contraintes d'intégrité, ne sont pas pris en compte. Ils doivent l'être dans l'écriture des modifications qui leur sont associées. Du fait de l'imbrication des composants de la structure et des axiomes, la modification d'un élément peut avoir des répercussions sur d'autres et compliquer l'écriture des procédures de modification.

Exemple 6.2

La procédure correspondant à ajouter_entête_ cde1 de l'exemple 1.5 (chapitre 1) peut s'écrire :

```
procédure ajouterentêtecde (clx, ncx, lx) ;
      ref (client) clx ;  integer ncx ; ref (ligne) lx ;...;
```

Elle ajoute un entête de commande concernant le client clx, avec le n° de commande ncx et une première ligne désignée par lx, doit faire de multiples vérifications avant d'effectuer la mise à jour, ou de la rejeter :

- *vérifier que le client clx référencé existe dans rcl,*
- *vérifier que l'entier ncx est compris entre 1 et 1 000, et que ce numéro n'existe pas encore comme image de la fonction nc,*
- *vérifier que l'élément ligne référencé par lx n'est pas l'objet vide,*

...

□

Tous les composants considérés (à l'exclusion des déclarations de classes) peuvent être regroupés, avec toutes les modifications, dans un même objet appartenant à une classe unique.

Exemple 6.3

On peut regrouper dans la classe gestioncommandes les structures de données et modifications définies dans les exemples 6.1 et 6.2 :

```
class  gestioncommandes ;
```

```
begin
    ref (head) rl ;
    ref (ligne) procedure tl ; ... ;
    ref (ligne) procedure s (x) ; ... ;
    ref (head) rp ;
    ref (commande) array (1 : 1 000) cde ;
    ref (client) array (1 : 100) clt ;
    ref (head) qté ;
    ...
    procedure ajouterentêtecde (clx, ncx, lx) ; ... ;
    ... autres procédures de modification ...
    ... initialisations ...
end
```

On appelle initialisations l'ensemble des instructions nécessaires à la création d'un objet de la classe gestioncommandes. En effet, à partir de ce moment, il est possible de créer plusieurs structures de ce type, par le générateur new, ce qui pourrait correspondre, par exemple, à la description des données de différents sites.

□

On conçoit sur cet exemple toute la difficulté qu'il y a parfois à écrire des procédures de modification qui préservent la consistance (ou la cohérence) d'une donnée quand sa structure est complexe. C'est la raison pour laquelle nous nous sommes intéressés dans ce travail à des moyens plus puissants pour structurer les données, comme les modèles relationnels et les mécanismes d'abstraction, où l'on tente de localiser le mieux possible les différents éléments, notamment les modifications.

1.1.2 Approche systématique

Beaucoup de recherches récentes sur les langages de programmation font une large part à la notion d'abstraction sous différentes

formes : CLU (LIS 77), MESA (GES 77), ALPHARD (SHA 77), MEFIA (SCH 78, CUN 79), GREEN-ADA (HON 79a), etc... Un des buts du raisonnement par niveaux d'abstraction est d'améliorer la structuration des données et des programmes, en utilisant des techniques d'analyse descendante, par raffinement, partant de spécifications mathématiques pour aboutir progressivement à une implantation.

Une démarche analogue est possible en SIMULA 67, comme nous l'avons déjà souligné, grace aux notions de classes hierarchisées et de procédures virtuelles : d'ailleurs, les concepts de types, modules, "clusters", "forms", univers, "packages" ... qui apparaissent dans les langages précédemment cités s'en inspirent plus ou moins, comme ne manquent jamais de le rappeler leurs concepteurs.

Nous cherchons à tirer profit de ces idées pour décrire mieux et plus généralement les structures de données que nous manipulons dans les maquettes. Dans le paragraphe précédent, nous avons montré toute la difficulté qu'il y a actuellement à vouloir gérer des structures complexes. Sans doute les thoéries sur l'abstraction ont-elles fait des progrès importants ces dernières années, mais il n'y a pas, à notre connaissance, de tentatives très concluantes sur de gros problèmes réels.

Pour faciliter les choses dans une première étape, il est raisonnable de se tourner sur des restrictions du modèle que nous avons présenté : par exemple, le modèle relationnel de CODD (COD 70).

Après un rappel de ce modèle, nous amorçons sa description avec différents niveaux d'abstraction autorisés par SIMULA 67, puis nous soulignons la différence qui existe entre les approches classes (de SIMULA 67) et types abstraits.

Il n'est pas question ici de la description d'un système de gestion de base de données "en vraie grandeur", mais simplement de ce qui pourrait en être le noyau, dans un système "en réduction", où toutes les données

sont gérées en mémoire centrale et dans un seul programme ! Ce petit système a été programmé et testé sur l'ordinateur IRIS-80 de l'IUCAL (voir Annexe A).

Nous disons en conclusion ce que pourrait être une implantation réelle utilisable pour de véritables applications, et nous faisons part de travaux qui vont dans ce sens.

a) Modèle relationnel et langage algébrique (COD 70)

(i) Relation

Une relation n-aire r, définie sur des ensembles $E_1, \ldots, E_n$ non nécessairement distincts, est un sous-ensemble du produit cartésien $E_1 \times \ldots \times E_n$: $r \subset E_1 \times \ldots \times E_n$.

$E_1, \ldots, E_n$ sont appelés composantes de r.

Sur les relations, il est possible d'effectuer certaines opérations, qui correspondent à des formules du calcul des prédicats du ler ordre : projection, jonction, division, union, différence, ...

(ii) Projection

Si r est une relation définie sur $E_1, \ldots, E_n$ et σ une permutation de $\{1, \ldots, n\}$: $\sigma : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\}$, la projection de r sur $E_{\sigma(1)}, \ldots, E_{\sigma(p)}$ $(p \leq n)$ est une relation sur $E_{\sigma(1)}, \ldots, E_{\sigma(p)}$ définie par :

$$\text{proj}_{\sigma, p}(r) = \{(x_{\sigma(1)}, \ldots, x_{\sigma(p)}) ; \exists x_{\sigma(p+1)} \ldots \exists x_{\sigma(n)} (x_1, \ldots, x_n) \in r\}$$

Cette relation correspond à une interprétation de la formule du calcul des prédicats du ler ordre :

$$\exists x_{\sigma(p+1)} \ldots \exists x_{\sigma(n)}\ r(x_1, \ldots, x_n)$$

(iii) Jonction

Si r est une relation définie sur $E_1, \ldots, E_n$, et s une relation définie sur $F_1, \ldots, F_p$ avec $E_i = F_j$, la jonction de r et s par rapport à E_i et F_j est une relation sur $E_1, \ldots, E_n, F_1, \ldots, F_{j-1}, F_{j+1}, \ldots, F_p$ définie par :

$$\text{join}_{i,j}(r, s) = \{(x_1, \ldots, x_n, y_1, \ldots y_{j-1}, y_{j+1}, \ldots, y_p) ;$$
$$(x_1, \ldots, x_n) \in r \text{ et } (y_1, \ldots, y_p) \in s \text{ et } x_i = y_j\}$$

Cette relation correspond à une interprétation de la formule du calcul des prédicats du ler ordre :

$$(r(x_1, \ldots, x_i, \ldots, x_n) \wedge s(y_1, \ldots y_{j-1}, x_i, y_{j+1}, \ldots, y_p)).$$

(iv) Division

Si r est une relation définie sur $E_1 \ldots, E_n$, et s une relation définie sur $E_{\sigma(s)}, \ldots, E_{\sigma(p)}$ $(p \leq n)$, où σ est une permutation sur $\{1, \ldots, n\}$, la division de r par s est une relation sur $E_{\sigma(p+1)}, \ldots, E_{\sigma(n)}$ définie par :

$$\text{div}_{\sigma, p}(r, s) = \{(x_{\sigma(p+1)}, \ldots, x_{\sigma(n)}) ; (x_{\sigma(s)}, \ldots, x_{\sigma(p)}) \in s$$
$$\supset (x_1, \ldots, x_n) \in r\}$$

Cette relation correspond à une interprétation de la formule du calcul des prédicats du ler ordre :

$$\forall x_{\sigma(s)} \ldots \forall x_{\sigma(p)} (s(x_{\sigma(s)}, \ldots, x_{\sigma(p)}) \supset r(x_1, \ldots, x_n))$$

(v) Union (insertion d'un élément)

Si r est une relation définie sur $E_1, \ldots, E_n$ et $a = (a_1, \ldots, a_n)$ un élément de $E_1 \times \ldots \times E_n$, l'union de r et a est une relation sur $E_1, \ldots, E_n$ définie par :

$$\text{union}(r, a) = r \cup \{a\}$$

Cette relation correspond à une interprétation de la formule du calcul des prédicats du ler ordre :

$$(r\ (x_1, \ldots, x_n)\ \vee (x_1 \equiv a_1 \wedge \ldots \wedge x_n \equiv a_n))$$

(v) Différence (soustraction d'un élément)

Dans les mêmes conditions que ci-dessus, la différence de r et a est une relation sur $E_1, \ldots, E_n$ définie par :

différence (r, a) = r - {a }.

Cette relation correspond à une interprétation de la formule du calcul des prédicats du ler ordre :

$$(r\ (x_1, \ldots, x_n) \wedge \neg (x_1 \equiv a_1 \wedge \ldots \wedge x_n \equiv a_n))$$

b) Réalisation d'un modèle relationnel abstrait en SIMULA 67

Nous montrons ici comment réaliser un modèle où l'on peut manipuler des relations, et définir les opérations précédentes, sans se soucier de la manière précise dont sont implantées les relations.

(i) Objets abstraits

Les n-uples que nous traitons dans les relations appartiennent à des sous-classes de la classe objet. Pour donner ultérieurement la possibilité de les mettre dans les listes, nous préfixons sa déclaration par link :

```
link class objet ;
virtual : text procedure clé ; null
```

A chaque objet est attaché une clé qui l'identifie et qui est un texte ; ceci n'est en rien restrictif car on peut fabriquer en SIMULA 67 un texte avec de multiples éléments entiers, réels, ...

Deux n-uples appartenant à la même sous-classe d'objet sont considérés comme égaux s'ils ont la même clé. La procédure "abstraite" égal permet de les comparer :

```
boolean procédure égal (x, y) ; ref (objet) x, y ;
    égal := x . clé = y . clé
```

On doit en effet user de ce détour car, et c'est un inconvénient de SIMULA 67 (relevé par J. VAUCHER dans (VAU 79)), les objets n'ont pas de valeur, et ni l'affectation (:=), ni la comparaison (=), ne s'appliquent à eux. Seules exceptions : les types simples prédéfinis : integer, real, text, ...

(ii) Relations abstraites

Une relation est alors un ensemble d'objets différents (selon cette définition), qui appartiennent tous à la même sous-classe de la classe objet.

Sur les relations peuvent être définies des opérations, mettant en jeu une seule relation parmi les opérandes. On introduit alors dans la déclaration d'une classe relation certaines procédures, qui sont virtuelles, parce que, à ce niveau, on ne désire pas les implanter réellement. On laisse le soin à l'implémenteur de les définir complètement à un niveau inférieur.

On a par exemple la déclaration suivante :

```
class relation ;
    virtual : procedure union, soustraction ;
              ref (objet) procedure élément, premier,
              suivant ;
begin
    integer card
end
```

L'entier card donne le cardinal de la relation : il est initialisé à 0 à la création de la relation.

Les procédures union et soustraction permettent d'ajouter ou de supprimer un n-uple, conformément à a).

Les fonctions-procédures qui les suivent permettent d'accèder à un n-uple (ou à none, référence de l'objet vide) :

. élément permet d'accèder à un n-uple connaissant sa clé ;
. premier permet d'obtenir un "premier" n-uple ;
. suivant permet d'obtenir le "successeur" d'un n-uple.

Il est en effet nécessaire de se donner le moyen d'énumérer ainsi les n-uples d'une relation : ceci est réalisé au moyen d'une relation d'ordre arbitraire, qui transforme toute simple relation en un objet plus riche, qu'on peut appeler une séquence, ou liste linéaire "abstraite", parce que l'ensemble des valeurs y est quelconque.

(iii) Autres opérations du modèle relationnel

A ce niveau d'abstraction, et à partir des opérations qui viennent d'être définies, il est possible d'écrire un certain nombre de (fonctions-) procédures qui réalisent les autres opérations algébriques du modèle relationnel. On a ici une sorte d'abstraction "fonctionnelle" (JAC 78). On trouvera le texte complet de ces procédures en annexe A.

- Projection d'une relation

Elle est réalisée par la procédure :

```
procédure proj (r, t, aff) ; name s ; ref (relation) r, s ;
        ref (objet) procedure aff ; ... ;
```

r est le nom de la relation initiale et t celui de la projection. A un niveau concret d'utilisation, le seul effort demandé à un programmeur est d'écrire une fonction-procédure aff, qui réalise la projection d'un n-uple de r

sur un n-uple de t, et qui est une simple suite d'affectations. Tout le reste, notamment la détection des "doubles" est réalisé automatiquement au niveau abstrait.

- Jonction de deux relations

Elle est réalisée par la procédure :

```
procedure join (r, s, t, conc) ; name t ; ref (relation) r, s, t ;
        ref (objet) procedure conc ; ... ;
```

r et s sont les deux relations initiales et t leur jonction. A un niveau concret d'utilisation, le programmeur doit écrire une fonction-procédure conc, qui réalise la jonction d'un n-uple de r et d'un n-uple de s en un n-uple de t quand celle-ci est possible, sinon le résultat rendu est none.

- Division de deux relations

Elle est réalisée par la procédure :

```
procedure divide (r, s, t, coup) ; name t ; ref (relation) r, s, t ;
        procedure coup ; ... ;
```

r et s sont les deux relations initiales et t leur division. A un niveau concret d'utilisation, le programmeur doit écrire une procédure coup, qui réalise la coupure d'un n-uple de r en un n-uple de s et un n-uple de t, et qui est une simple suite d'affectations.

L'usage de ces opérations est extrêmement simple, moyennant l'écriture de (fonctions-) procédures triviales à un niveau concret.

(iv) Autres opérations

Il est très facile de définir et utiliser d'autres procédures abstraites travaillant sur des relations abstraites, pour réaliser des opérations ensemblistes diverses : par exemple, union, intersection, complémentation

disjonction ..., pour des relations contenant des objets du même type.

L'opération de tri d'une relation est elle-aussi facile à réaliser à un niveau abstrait, par des procédures de la forme :

```
procedure tri (r, t, arg, sens) ; name t ; ref (relation) r, t ;
    text procedure arg ; character sens ; ... ;
```

r est la relation initiale et t la relation triée. arg est la fonction qui délivre l'argument de tri des n-uples, et sens est un caractère (C ou D) qui indique s'il s'agit d'un tri croissant ou décroissant.

Divers algorithmes de tri peuvent être utilisés ici, en fonction notamment du nombre d'éléments intervenant dans la relation à trier, et ils conduisent à plusieurs procédures de tri.

(v) Clés secondaires

Il est naturellement possible de déclarer d'autres fonctions-clés dans des objets pour avoir des indicatifs secondaires, mais l'égalité de deux objets est toujours déterminée par l'égalité de leurs clès primaires.

c) Implantation des relations

On ne peut naturellement pas rester au niveau d'abstraction précédent, dès lors que l'on veut utiliser de véritables relations. Il faut implanter les (fonctions-) procédures virtuelles précédentes, considérant que les n-uples sont rangés dans des listes, des tableaux, des arbres, ... On déclare des classes de relations concrètes, préfixée par relation, de manière à pouvoir utiliser pour elles les procédures définies au niveau abstrait.

Exemple 6.4

A titre d'illustration, nous donnons ci-dessous le schéma d'une implantation de relations en listes linéaires chaînées :


```
relation class liste ;
begin
    ref (head) ℓ ;
    procedure union (a) ; ref (objet) a ; ... ;
    procedure soustraction (a) ; ref (objet) a ; ... ;
    ref (objet) procedure élément (c) ; text c ; ... ;
    ref (objet) procedure premier ; ... ;
    ref (objet) procedure suivant (x) ; ref (objet) x ; ... ;
    ℓ :- new head
end
```

On peut noter que cette déclaration reste largement abstraite, puisqu'elle ne manipule que des objets abstraits et leurs clés, mais elle impose un rangement des objets dans la liste ℓ. On a ici une sorte d'abstraction "structurale" (JAC 78).

□

Il est bien sûr possible, dans une implantation donnée, de compléter les procédures virtuelles du niveau abstrait relation par d'autres (fonctions-) procédures, adaptées par exemple à des accès particuliers. De la même manière, c'est à un niveau concret que peuvent être programmés des contrôles d'intégrité pour des modifications.

d) Exemple d'utilisation (6.5)

Nous reprenons ici un exemple proposé par C.J. DATE (DAT 75), avec des fournisseurs, des produits et des commandes, et qui a servi pour le test des programmes ci-dessus (annexe A).

On définit différents types de n-uples :

```
objet class supplier (sd, sname, status, city) ; $ fournisseurs $
    text sd, sname, city ; integer status ;
begin
    text procedure clé ; clé :-sd ;
end ; $ supplier $
```

```
objet class ville (city) ; text city ;          $ villes $
begin
      text procedure clé ; clé :-city ;
end ; $ ville $

objet class cde (sd, pd, qty) ;                 $ commandes $
      text sd, pd ; integer qty ;
begin
      text procedure clé ;
      begin text u ;
            u:-blanks (4) ;
            u.puttext (sd) ;
            u.puttext (pd) ;
            clé :- u
      end ; $ clé $
end ;  $ cde $

objet class scde (city, status, sname, sd, pd, qty) ; $ commandes
                  fournisseurs $
      text city, sname, sd, pd ; integer qty, status ;
begin
      text procedure clé ;
      begin text u ;
            u:-blanks (4) ;
            u.puttext (sd) ;
            u.puttext (pd) ;
            clé :- u
      end ; $ clé $
end ; $ scde $

objet class prod (pd) ; text pd ;               $ produits $
begin
      text procedure clé ; clé :- pd ;
end ; $ prod $
```



```
objet class cdek (sd, qty) ; text sd ; integer qty ;
                                                $ fournisseurs -
                                                  quantités $
begin
      text procedure clé ;
      begin text u ;
            u:-blanks (3) ;
            u.puttext (sd) ;
            u.sub (3,1).putint (qty) ;
            clé:-u
      end ; $ clé $
end ; $ cdek $
```

Toutes les relations considérées sont implantées en listes :

```
ref (liste) s, sp, sville, ssp, k, spk ;
```

s : relation fournisseurs,
sp : relation commandes,
sville : relation villes,
ssp : relation commandes-fournisseurs,
k : relation produits,
spk : relation fournisseurs-quantités.

On désire faire la projection de s sur svilles, la jonction de s et sp en ssp sur sd, la division de sp par k en spk, d'où les procédures :

```
ref (objet) procedure aff (x) ; ref (supplier) x ;
      aff :- new ville ( x.city ) ; $ aff $

ref (objet) procedure conc (x,y) ; ref (supplier) x ; ref (cde) y ;
      if x . sd = y . sd
      then  conc :- new scde (x.city, x.status, x.sname, x.sd,
                  y.pd, y.qty)
      else  conc :- none ;  $ conc $

procedure coup (x, y, z) ; name y, z ;
      ref (cde) x ; ref (cdek) y ; ref (prod) z ;
```

```
begin
      y :- new cdek  (x.sd, x.qty) ;
      z :- new prod  (x.pd)
end ;  $ coup $
```

Après création des relations, et remplissage de s, sp, k, il suffit, pour créer sville, ssp et spk, d'appeler :

```
      proj   (s, sville, aff) ;
      join   (s, sp, ssp, conc) ;
      divide (sp, k, spk, coup) ;
```

□

e) Prologue standard

On peut réunir les mécanismes précédents (classes et procédures abstraites et concrètes) dans une classe unique servant de prologue de programmathèque, Sa définition, sa mise en catalogue et son utilisation, sont en tous points semblables à ce que nous avons dit à propos de la classe MAESTRO (chapitre 5).

Sans doute la fusion de ces deux classes est-elle encore prématurée, mais on peut l'envisager à terme, après une plus grande expérience et une amélioration des différentes notions qui les composent.

f) Liaisons avec les types abstraits

La démarche précédente a évidemment un rapport avec celle que l'on adopte pour la définition de types abstraits. Il est sans doute utile d'en souligner les analogies et les différences.

Un type abstrait peut être considéré de deux façons : tantôt comme un ensemble d'objets, tantôt comme un ensemble d'opérations conformes à certains axiomes. Nous ne parlons ici que de la spécification algébrique des types abstraits et non de leur spécification axiomatique (voir par exemple DER 79).

(i) Déclaration d'un type abstrait algébrique

La spécification algébrique d'un type abstrait T, dépendant d'autres types $T_1, \ldots, T_n$, comprend deux parties (GUT 78) :

- l'une, syntaxique, où l'on définit les profils de certaines fonctions, qui sont à valeur dans T (constructeurs) ou à valeur dans l'un des T_j ($1 \leq j \leq n$) (sélecteurs ou discriminateurs (FIN 78)) ;

- l'autre, sémantique, où l'on spécifie par des équations certaines de ces fonctions à partir d'autres :

type T [$T_1, \ldots, T_n$]
syntaxe
  constructeurs
  cl : $T_{i_1} \times T_{i_2} \times \ldots \rightarrow T$
  ...
  sélecteurs
  sl : $T \times T_{i_1} \times \ldots \rightarrow T_j$
  ...
sémantique
  spécification équationnelle de certaines opérations à partir d'autres.

La formulation adoptée pour la partie sémantique est analogue à celle du calcul des prédicats du 1er ordre avec égalité dont nous avons parlé, à ceci près que les axiomes donnés dans la partie sémantique sont sans quantificateurs et sont uniquement des égalités entre fonctions.

On a souvent besoin d'une troisième partie où l'on restreint par des axiomes égalitaires de la même forme le domaine de définition de certaines fonctions.

Exemple 6.6

On reprend ici le type Stack (pile) écrit par J.V. GUTTAG dans (GUT 78). C'est en fait un modèle de type, ou type générique (JAC 78), parce que,

entre crochets, figure un paramètre elementtype qui peut être remplacé par un nom de type quelconque integer, real, ... , ce qui permet de définir des piles d'entiers, de réels, ... Ce paramètre est donc du type Type.

```
type Stack [ elementtype : Type ]
syntaxe
    constructeurs
    newstack :                      → Stack
    push     : Stack x elementtype  → Stack
    pop      : Stack                → Stack
    replace  : Stack x elementtype  → Stack
    sélecteurs
    top      : Stack                → elementtype ∪ {undefined}
    isnew    : Stack                → Boolean
Sémantique
    déclare  stk : Stack , elm : elementtype
    pop (newstack) = newstack
    pop (push (stk, elm)) = stk
    top (newstack) = undefined
    top (push (stk, elm)) = elm
    isnew (newstack) = true
    isnew (push (stk, elm)) = false
    replace (stk, elm) = push (pop (stk), elm)
```

La signification intuitive des constructeurs est la suivante : newstack est la création d'une pile, push l'empilement d'un élément de type elementtype, pop le dépilement, et replace le remplacement de l'élément au sommet d'une pile.

Celle des sélecteurs est la suivante : top est l'accès à l'élément du sommet d'une pile, et isnew est vrai si et seulement si on a la pile vide.

Les équations de la partie sémantique spécifient plus précisément ces fonctions. A titre d'exemple, la 7ème équation exprime que le remplacement de l'élément au sommet de la pile stk par elm conduit à la même pile qu'un dépilement de stk suivi d'un empilement de elm.

Pour plus de détails, on consultera par exemple (GUT 78). □

(iii) Problèmes posés dans ce cadre

Le cadre des types abstraits permet de s'interroger sur des questions fondamentales comme :

- la consistance, la complétude ou la minimalité des opérations, pour une définition rigoureuse de structures de données ;

- la représentation d'un type par un autre (notamment pour une bonne implantation de structures de données), avec la possibilité de faire la preuve que la représentation est correcte, c'est-à-dire qu'elle satisfait les spécifications sémantiques (GUT 78, GAU 78).

Un des buts est la fabrication d'outils semi-automatiques pour aider à définir et implanter des structures de données, et pour écrire plus facilement des programmes.

(iv) Comparaison avec les possibilités de SIMULA 67

Il est intéressant de voir en quoi les classes de SIMULA 67 diffèrent des types abstraits. Pour mieux le faire sentir, nous reprenons l'exemple des piles en écrivant la classe suivante :

Exemple 6.7

```
class stack ;
    virtual :
        ref (stack) procedure push, pop, replace ;
        ref (elementtype) procedure top ;
        boolean procedure isnew ;
null
```

□

On a bien, avec les procédures virtuelles, une description abstraite. Mais, d'un point de vue syntaxique, les spécifications des fonctions ne sont pas complètes : si l'on a le type du résultat, on n'a pas un profil complet, puisqu'il manque le nombre et le type des arguments.

L'aspect sémantique est totalement absent : il ne peut figurer que sous forme de commentaires inexploitables pour des preuves automatiques éventuelles.

On a de plus ici un point de vue plus dynamique des descriptions : les opérations sont liées aux objets et non pas aux types. Ainsi, la création d'une pile stk s'écrirait en SIMULA stk :- new stack, alors qu'elles s'exprimerait, avec les notations de CLU (LIS 77), stk := Stack $ create ; l'empilement d'un élément elm sur stk pour obtenir stk 1 s'écrirait en SIMULA stk1 :- stk.push (elm), alors qu'elle s'écrivait stk 1 := Stack $ push (stk, elm) en CLU.

Du fait du préfixage des classes, on n'a pas en SIMULA la possibilité de faire des vérifications statiques complètes de types lors d'appels de procédures, ce qui est une règle et un gage de sécurité avec les types abstraits. Mais, malgré les erreurs qu'elle risque de faire commettre, la hiérarchisation des classes est un élément de souplesse et de confort important pour la programmation en SIMULA. Elle permet d'ailleurs d'obtenir une sorte de généricité simple dans les classes.

g) Conclusion

On voit donc que la richesse du langage SIMULA 67 fournit des mécanismes d'abstraction, notamment à travers les classes hiérarchisées, les procédures virtuelles et la clause hidden protected. Les nouveaux systèmes SIMULA disponibles permettent de plus de faire de la compilation séparée de classes et de procédures (NCC 76).

La réalisation systématique d'un modèle relationnel par niveaux d'abstraction, telle que nous l'avons présentée, n'est bien sûr qu'une ébauche expérimentale et à échelle réduite. Son développement et son utilisation ne sont guère envisageables que dans notre cadre de maquettes de systèmes d'information.

Notons tout de même que SIMULA 67 est utilisé à l'heure actuelle pour implanter des systèmes de gestion de bases de données (KIR 77, MAK 77, BOU 77) ou des interfaces avec des systèmes existants (ROO 77).

Mais la réalisation en vraie grandeur d'un bon système nous semble nécessiter à l'heure actuelle :

. un système de conservation d'objets sur mémoires auxiliaires,
. un langage de manipulation de types abstraits,
. une compilation séparée de modules abstraits et concrets, avec un relieur dynamique évolué,
. des vérifications syntaxiques et sémantiques poussées.

C'est un peu dans cette optique qu'a été développé le système SOC de M. BANATRE (BAN 78), que s'orientent les projets ATM et TYP de J.C. DERNIAME et J.J. CHABRIER (MIN 79, CHA 79), et que sera implanté le langage GREEN-ADA (HON 79a, HON 79b).

1.2 Mécanismes de protection et de blocage

La protection de structures de données, le blocage et le déblocage de processus, sont réalisés exclusivement par le mécanisme de moniteur et de condition simple (sans priorité ni préemption), tel qu'il a été décrit dans le chapitre 4.

Toutefois, pour l'édition de statistiques dans le système MAESTRO, l'entête en a été complété, par un préfixe link (qui permet de mettre les moniteurs dans une liste) et par un paramètre nom ; le corps en est sans changement :

```
link class monitor (nom) ; text nom ;
begin ... end
```

De même, on a trouvé plus commode de complèter la classe condition par deux procédures sigleave et signalg, ce qui lui donne la structure suivante :

```
class condition (m) ; ref (monitor) m ;
begin
    ref (head) waitq ;
    procedure cwait ; ... ;
    procedure csignal ;... ;
    procedure sigleave ; ... ;
    procedure csignalg ;... ;
    boolean procedure cempty ;... ;
    waitq :- new head
end
```

En effet, C.A.R. HOARE a remarqué (HOA 74) que la procédure signal était souvent la dernière d'une procédure d'un moniteur, juste avant leave. Aussi a-t-il proposé de créer une procédure particulière sigleave concaténant les procédures signal (ici csignal) et leave.

L'appel de la procédure csignalg, quant à lui, a pour effet d'exécuter un signal (ici csignal) pour tous les processus en attente sur la condition : ceci a pour effet de les débloquer chacun à leur tour.

Nous nous sommes déjà longuement attardés sur ces outils. Aussi, nous ne donnons que deux exemples simples de composants qui nous servent souvent dans les maquettes : les files d'attente et les "moniteurs d'activation".

1.2.1 Files d'attente

Nous présentons ici une classe de structures de données où peuvent s'effectuer des blocages. Il s'agit de files d'attente, gérées

selon une politique quelconque, avec des possibilités multiples d'accès, de parcours et de modification : la classe tfile.

Les éléments rangés dans les files sont des liens, appartenant à la classe transaction ou à ses sous-classes. Cette classe transaction a été essentiellement créée à des fins de prise de mesure pour les statistiques (chapitre 9).

De la classe tfile, nous ne présentons que les éléments permettant de comprendre la gestion des files d'attente, avec le détail de quelques procédures seulement. On trouvera en annexe C le texte de la classe tfile, mais seulement avec les (fonctions-) procédures nécessaires à la résolution de l'application du chapitre 5.

```
monitor class tfile (mesure, équilibre, inter, dmes) ;
    boolean mesure, équilibre ; real inter, dmes ;
begin
    ref (head) q ;
    ref (condition) nonvide ;
    procedure entrer (t) ; ref (transaction) t ;
    begin
      this monitor . enter ;
      t . into (q) ;
      nonvide . sigleave
    end ; $ entrer $
    ref (transaction) procedure sortirtête ;
    begin
      this monitor . enter ;
      if q . empty then nonvide . cwait ;
      sortirtête :- q . first qua transaction ;
      sortirtête . out ;
      this monitor . leave
    end ; $ sortirtête $
```

```
    ref (transaction) procedure sortirqueue ; ... ;
    procedure enlever (t) : ref (transaction) t ; ... ;
    ref (transaction) procedure premier ; ... ;
    ref (transaction) procedure dernier ; ... ;
    ref (transaction) procedure suivant (t) ; ref (transaction) t ; ... ;
    ref (transaction) procedure précédent (t) ; ref (transaction) t ; ... ;
    procedure modifier (t, pds) ; ref (transaction) t ; real pds ; ... ;
    integer procedure longueur ; ... ;
    q :- new head ;
    nonvide :- new condition (this monitor)
    ...
end
```

La signification des paramètres de l'entête est la suivante :

mesure est true si et seulement si on désire des résultats statistiques généraux sur la file d'attente (chapitre 9) ;

équilibre est true si et seulement si on désire des résultats statistiques détaillés sur la file, c'est-à-dire la prise de mesures à partir de la date simulée dmes et à des intervalles de temps simulé inter (en heures).

Les éléments de type transaction sont rangés dans la liste q.

L'appel de la procédure entrer a pour effet de ranger la transaction repérée par t en queue de liste et d'émettre un signal sur la condition nonvide, qui débloque éventuellement un processus en attente sur cette condition.

L'appel de la fonction-procédure sortirtête a pour effet de délivrer la première transaction de q si elle existe, sinon de bloquer le processus appelant sur la condition nonvide.

La fonction-procédure sortirqueue a un rôle analogue pour la dernière transaction.

L'appel de la procédure enlever a pour effet de sortir de la liste q la transaction repérée par t.

Les appels des procédures premier, dernier, suivant, précédent ont pour effets respectifs l'accès à la transaction de tête, de queue, à celle qui suit la transaction t, à celle qui précède de la transaction t.

L'appel de la procédure modifier a pour effet de remplacer le poids (chapitre 9) de la transaction t de la file par pds.

L'appel de la fonction-procédure longueur délivre le nombre d'éléments dans la file.

C'est donc la gestion ordinaire d'un "tampon" de taille infinie.

L'exécution des appels des (fonctions-) procédures est réalisé en exclusion mutuelle. On peut noter que cette précaution n'est utile dans un contexte de quasi-parallélisme que quand la durée simulée d'exécution des (fonctions-) procédures n'est pas nulle.

Exemple 6.8

Dans l'application présentée au chapitre 5, on trouve les déclarations et instructions suivantes (annexe B) :

```
ref (tfile) courrier, ordres, bons1, bons2 ;
...
courrier :- new tfile ('COURRIER', true, true, 24, 6.5) ;
ordres   :- new tfile ('ORDRES', true, false, 24, 0) ;
bons1    :- new tfile ('BONS1', true, true, 24, 0) ;
bons2    :- new tfile ('BONS2', false, false, 0, 0) ;
```

Par exemple, courrier est réalisé comme une file d'attente de nom 'COURRIER', pour laquelle on désire des résultats statistiques généraux et détaillés, grâce à des mesures faites tous les jours à 6 heures 30 mn.

Dans les processus, on trouve des accès ou modifications à de telles structures. Ainsi, dans le processus enregistrement, on a la séquence suivante

(annexe B), où b est une transaction :

```
while  courrier . longueur > 0 do
begin
      b :- courrier . sortirtête
      ... traitement de b ...
end
```

□

1.2.2 Moniteurs d'activation

Ils sont utilisés de manière systématique pour lancer des processus, notamment quand on utilise des échéanciers secondaires. Le fonctionnement d'un moniteur d'activation peut être schématisé comme suit :

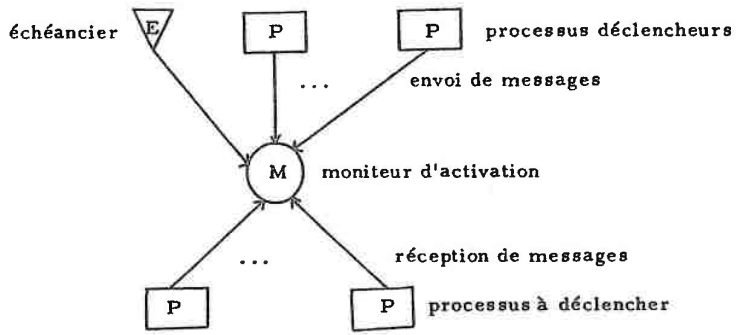


Figure 6.1 - Moniteur d'activation

Des messages sont envoyés par des processus déclencheurs (dont certains peuvent être contenus dans des échéanciers (cf. (1.4))) dans un moniteur d'activation, qui mémorise plus ou moins ces messages comme nous allons le voir.

Des processus à déclencher doivent chercher à les acquérir dans le moniteur pour poursuivre leur exécution.

Ces messages peuvent n'avoir pour les processus qu'une valeur de signal de déclenchement, ou, au contraire, contenir des informations utiles.

On a ici un mécanisme déclencheur standard, avec toutes les variantes que l'on désire. Dans les applications que nous avons réalisées, nous en avons retenu trois types.

- Le type 1 définit des moniteurs qui ne mémorisent qu'un message au maximum, et ce, seulement quand il y a au moins un processus en attente de message.

- Le type 2 est associé à des moniteurs qui mémorisent tous les messages envoyés par les processus déclencheurs.

- Le type 3 définit des moniteurs qui ne mémorisent qu'un message au maximum, qu'il y ait ou non un processus en attente de message.

On retrouve ici quelques éléments de la classification des modes de communication de (QUE 79).

Le choix du type de moniteurs d'activation dépend de la forme des processus et des structures de données sur lesquelles ils opèrent.

Les messages qui sont échangés entre les processus font partie d'une classe message, préfixée par link, et ayant comme seul paramètre un entier no :

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

Pour ne pas surcharger la description de la classe actmonitor des moniteurs d'activation, nous ne donnons que sa structure. Le texte complet est donné en annexe C.

```
monitor class actmonitor (type) ; irteger type ;
```

```
begin
    ref (head) actq ;
    ref (condition) déblocage ;
    procedure acquérir (m) ; name m ; ref (message) m ; ... ;
    procedure envoyer (m) ; ref (message) m ; ... ;
    actq :- new head ;
    déblocage :- new condition (this monitor)
end
```

type est le paramètre indiquant le type du moniteur : 1, 2 ou 3.

actq est la liste (cas 2) ou la boîte (cas 1 et 3) qui mémorise les messages.

La condition déblocage sert à bloquer les processus en attente d'un message.

L'appel de la procédure acquérir permet au processus invocateur d'accéder au premier message m de actq si cette liste n'est pas vide, sinon bloque ce processus.

L'appel de la procédure envoyer provoque l'envoi du message m dans actq : il est mémorisé ou non en fonction du type du moniteur d'activation et un signal est fait sur la condition déblocage.

Exemple 6.9

Dans l'application décrite dans le chapitre 5, les moniteurs activation1, activation2 et boite sont réalisés par des moniteurs d'activation, respectivement de type 3, 3 et 1 :

```
ref (actmonitor) activation1, activation2, boite ;
...
activation1 :- new actmonitor ('ENREGISTREMENT',3) ;
activation2 :- new actmonitor ('DECISION', 3) ;
boite       :- new actmonitor ('BOITE', 1) ;
```


Dans les processus, on retrouve des accès ou des modifications dans les moniteurs. Ainsi, le processus saisie, déclenché seulement après acquisition d'un message mess, a la structure suivante (annexe B) :

```
while true do
begin
    boite . acquérir (mess) ;
    ... traitement de mess ...
end
```

□

1.3 Processus ; durée de vie des processus

Un processus est un objet de la classe standard process ou de l'une de ses sous-classes.

Dans nos applications, nous utilisons souvent la classe processus :

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

nom est l'identification d'un processus, priorité sa priorité, et t un paramètre utilisable pour calculer une durée de travail.

Il est bien entendu qu'il est possible de créer d'autres sous-classes plus richement paramétrées.

Souvent les processus décrits sont cycliques, le début d'une nouvelle itération étant déterminé par la donnée enregistrée dans un moniteur.

Exemple 6.10

A titre d'illustration, nous donnons, la structure de la classe très simple de processus préparation, qui intervient dans l'application du chapitre 5 (annexe B) ;

```
processus class préparation ;
begin
      ref (bc) b, b1 ;
      while true do
      begin
          b :- ordres . sortir ;
          ...
          b1 :- new bc (...) ;
          bons1 . entrer (b) ;
          bons2 . entrer (b1) ;
          ...
      end
end
```

bc est une classe de bons de commande, sous-classe de la classe transaction. Le processus préparation se comporte comme un "consommateur" de bons de la file ordres et un "producteur" de bons dans les files bons1 et bons2, répétant indéfiniment le cycle consommation (avec risque de blocage) et production. D'autres contraintes liées au temps et à l'utilisation d'un processeur n'apparaissent pas ici.

□

Il est toujours possible de créer dynamiquement un processus nouveau grâce au générateur new, ou de supprimer dynamiquement un processus, grâce à la procédure cancel. Ceci est notamment nécessaire pour générer ou détruire des tâches parallèles en nombre indéterminé. Ainsi, un processus peut "naître" au départ ou au cours de la simulation, éventuellement créé par un autre processus. Il peut se détruire lui-même, quand il estime que son travail est terminé, ou être "tué" par un autre processus. Dans les applications simples que nous présentons ici, tous les processus existent et sont activés au début de la simulation. Il peuvent naturellement se trouver bloqués immédiatement dans un moniteur. Mais on trouve des cas où la génération et la destruction dynamiques de processus facilite bien les choses (chapitre 7).

1.4 Echéanciers secondaires

Nous avons vu, dans la présentation générale de SIMULA 67, que l'on y dispose d'un échéancier standard : 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 au même processus. Or, dans notre contexte, 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 notre système un mécanisme d'échéancier secondaire fonctionnant de concert avec l'échéancier standard de SIMULA.

Un échéancier contient une liste de couples (date d'activation, référence à un moniteur d'activation) de la classe evnotice :

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

Un processus scrutateur parcourt cette liste et se réveille à chaque date d'activation : alors, un message est généré et envoyé automatiquement au moniteur d'activation correspondant, ce qui peut avoir pour effet de déclencher un processus, comme nous l'avons vu en (1.2.2).

Les échéanciers sont conçus pour être cycliques : on donne les dates d'activation seulement sur une période de temps de longueur donnée, à partir d'une date déterminée : la génération et l'envoi de messages se reproduisent alors identiquement sur les périodes suivantes jusqu'à la fin de la simulation. Ceci correspond aux traitements répétitifs souvent observés en gestion.

La structure de la classe échéancier est la suivante :

```
class échéancier (per, hdp) : integer per, hdp ;
```

```
begin
    ref (head) echq ;
    ref (activation) scrutateur ;
    procedure vider (a) ; ref (actmonitor) a ; ... ;
    procedure lecturedates (a) ; ref (actmonitor) a ;... ;
    procedure insérer (a, d) ; ref (actmonitor) a ; real d ; ... ;
    ...
    echq :- new head ;
    scrutateur :- new activation (this échéancier)
end
```

per est la longueur entière d'une période (en heures) et hdp l'heure de début de la première période.

echq est la liste des couples de la classe evnotice rangés par dates croissantes, et scrutateur le processus d'exploration.

L'appel de la procédure vider supprime de echq les couples relatifs au moniteur a.

L'appel de la procédure lecturedates range dans echq des couples relatifs au moniteur a, avec des dates correspondant à la première période, lues en clair sous forme de triplets (jours, heures, minutes) et converties automatiquement en heures. Le nombre de triplets est quelconque et terminé par un astérisque *.

L'appel de la procédure insérer introduit dans echq le couple (a, d).

A chaque appel de vider, lecturedates ou insérer, le processus scrutateur est réordonnancé dans l'échéancier standard SQS, de manière à se réveiller à la prochaine date d'activation.

Le processus scrutateur appartient à la classe de processus activation que nous ne décrivons pas ici. On trouvera les textes de cette classe et de la classe échéancier en annexe C, avec quelques variantes pour les

procédures de mise à jour.

Exemple 6.11

Dans l'application du chapitre 5, on crée un échéancier éch, avec une période de 24 heures commençant à la date 0 par :

```
ref (échéancier) ech ;
...
éch :- new échéancier (24,0) ;
```

Si l'on désire qu'un message soit envoyé chaque jour à 10 heures dans activation1 et à 15 heures dans activation2, on peut écrire :

```
éch . lecturedates (activation1) ;
éch . lecturedates (activation2) ;
```

avec comme données :

```
0   10   0   *
0   15   0   *
```

Remarque : dans le programme présenté en annexe B, la procédure lecturedates s'appelle modifséch.

□

2 - POINT de VUE PHYSIQUE

Nous avons vu que les ressources physiques utilisées dans nos modèles se ramènent à un certain nombre de stations dont la gestion est indépendante , et dont les processus tentent d'obtenir les services. La liste des types de stations (ou ressources) que l'on peut envisager est infinie. Dans le projet MAESTRO, et pour les applications que nous avons étudiées, nous nous sommes limités à certains types de base que nous allons présenter.

Il nous a suffi d'utiliser des moniteurs et conditions simples pour réaliser certains d'entre eux. Mais il a fallu en revanche introduire de

nouvelles conditions, avec priorités et délais, pour réaliser certains autres.

Enfin, pour limiter l'usage des ressources à certaines périodes de disponibilité, nous avons construit un mécanisme de calendriers, qui enveloppe systématiquement les ressources, mais que l'on peut inhiber.

Nous présentons d'abord, comme compléments sur les moniteurs, les conditions avec priorités et délais (2.1), puis le mécanisme de calendrier (2.2), et, enfin, les différents types de ressources réalisés (2.3).

Les outils de base que nous avons défini dans la classe MAESTRO permettent de fabriquer une multitude d'autres types de ressources, au gré des applications.

2.1 Compléments sur les moniteurs ; conditions avec priorités et délais

Les moniteurs utilisés pour fabriquer des ressources sont identiques à ceux utilisés pour protéger les structures de données partageables. Cependant, pour des ressources préemptibles, il est nécessaire d'avoir des conditions avec priorités et délais.

Dans ce cas, un processus qui entre dans un moniteur pour demander une ressource doit pouvoir l'obtenir, même si elle est réservée pour un autre processus, à condition que sa priorité soit plus forte. Le deuxième processus est alors mis en attente derrière une condition avec priorité et le premier processus acquiert la ressource. Dans un contexte de quasi-parallélisme, on suppose, et ceci est une restriction importante, que :

Un processus qui tient une telle ressource se trouve toujours ordonnancé dans l'échéancier SQS.

La structure de la classe conditionp de conditions avec priorités et délais, dont on trouvera la description complète en annexe C, est la suivante :

```
class conditionp (m) ; ref (monitor) m ;
begin
    ref (head) waitq ;
    procedure cwait (pr, p) ; ref (process) pr ; real p ; ... ;
    procedure csignal ; ... ;
    boolean procedure cempty ; ... ;
    ref (process) procedure cfirst ; ... ;
    ...
    waitq :- new head
end
```

La liste waitq contient, rangés par priorités décroissantes, des éléments de la classe élément :

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

pr est une référence à 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 dans la SQS (politique "preemptive-resume").

L'appel de la procédure cwait permet de mettre l'élément (pr, p, dt) dans waitq à sa place : pr est une référence à un processus, p sa priorité, et dt est soit -1 pour le cas où pr est le processus courant, soit égal à pr . evtime-time (temps résiduel) pour les processus préemptés. Le processus pr est alors sorti de l'échéancier SQS, si besoin est.

L'appel de la procédure csignal permet de sortir le premier élément de waitq et d'ordonnancer 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 du moniteurs) que dans ce cas (HOA 74)).

L'appel de la fonction-procédure cempty permet de savoir si waitq est vide ou non.

L'appel de cfirst délivre le premier processus de waitq.

Nous donnerons, avec les ressources préemptibles, un exemple d'utilisation.

2.2 Calendriers de disponibilité de ressources

La nécessité de cette notion se fait sentir parce que certains processeurs ne sont pas toujours disponibles pour les processus d'un système, ou d'une partie de système, décrit par une maquette.

On peut définir, pour un processeur donné, des intervalles de temps pendant lesquels il est disponible. Souvent ces intervalles se reproduisent dans le temps de manière périodique, si bien qu'il suffit, comme pour les échéanciers, de décrire les débuts et fins de disponibilité sur la première période, après avoir fixé sa date de début et sa longueur.

En dehors d'un intervalle de disponibilité, il n'est pas possible d'acquérir une ressource protégée par un calendrier et les processus demandeurs sont bloqués. A la fin d'un tel intervalle, deux cas peuvent se produire, à la volonté du programmeur :

. tous les processus qui retiennent une ressouce protégée sont interrompus jusqu'à la prochaine disponibilité ;

. on les laisse terminer leur travail.

C'est à l'utilisateur de savoir quel cas correspond le mieux à la réalité qu'il veut décrire.

Deux processus internes scrutent la liste des bornes des intervalles pour lancer ou bloquer les processus.

Dans la version actuelle du système MAESTRO, la structure de la classe calendrier, dont on trouvera la description complète en annexe C, est la suivante :

```
ressource class calendrier (utilise, termine, per, hdp) ;
        boolean utilise, termine ; integer per, hdp ;
begin
        ref (head) noticeq, notcalq ;
        ref (condition) travail ;
        ...
        boolean procedure heuretravail ; ... ;
        procedure calconsulter ; ... ;
        procedure calsortir ; ... ;
        ...
end
```

La raison du préfixage de calendrier par ressource est expliqué en (2.3) ;

utilise est true si on utilise le mécanisme de calendrier pour les ressources protégées, false si on l'inhibe ;

termine est true si on laisse terminer les processus en cours lors d'une fin de disponibilité, false sinon ;

per et hdp donnent respectivement, comme pour les échéanciers, la longueur de la période et l'heure du début de la première période de description des horaires.

noticeq est la liste des (début, fin) des intervalles de disponibilité ; notcalq est la liste des références aux processus qui sont passés dans le calendrier (et qui utilisent, ou tentent d'utiliser une ressource protégée), avec leur durée résiduelle d'exécution (pour les processus interrompus en fin d'un intervalle de disponibilité).

travail est une condition qui bloque les processus en dehors des intervalles de disponibilité.

L'appel de la fonction-procédure heuretravail permet de savoir si l'on est, ou pas, dans une période de disponibilité.

L'appel de la procédure calconsulter par un processus a pour effet éventuel de le bloquer sur travail. Sinon, ou quand il sera débloqué (en début de disponibilité), un élément est créé pour lui dans notcalq.

L'appel de la procédure calsortir par un processus fait sortir l'élément de notcalq qui lui correspond.

Dans cette version, c'est à la création d'un calendrier par le générateur new que l'on crée définitivement la liste des bornes d'intervalles par lecture de couples de dates données sous la forme (jours, heures, minutes).

Dans une version ultérieure, il faudra donner plus de souplesse avec la mise à jour dynamique de noticeq et l'introduction des horaires variables (chapitre 7 (2.2)).

Des exemples d'utilisation sont donnés avec la présentation des types de ressources en (2.3).

2.3 Différents types de stations (ou ressources)

Ici encore, nous faisons abstraction des outils de mesure qui doivent compléter les descriptions pour permettre d'obtenir des résultats

statistiques. Il faut cependant dire que, pour collecter des mesures, les ressources utilisent des moyens analogues qui se trouvent rassemblés dans une classe unique préfixée par monitor, la classe ressource :

```
monitor class ressource ;
begin ... end
```

L'utilisateur n'a pas à connaître le contenu de cette classe sur lequel nous reviendrons au chapitre 9.

Nous parcourons à présent les différents types de ressources, en allant des plus simples aux plus compliquées. Nous ne les écrivons pas en détail, sauf exception, nous contentant en général de spécifier ce qui doit en être vu par un utilisateur (clause not hidden protected) avec quelques éléments supplémentaires nécessaires aux explications. Le texte complet en est donné en annexe C, sans la clause de protection qui n'existe pas dans la version actuelle du système SIMULA - IRIS 80.

Rappelons que toutes les ressources sont susceptibles d'être protégées, ou non, par un calendrier tel que nous venons de le décrire. Ceci constitue certainement une originalité de nos outils par rapport à d'autres que l'on rencontre dans les systèmes de simulation.

2.3.1 Ressources critiques simples : stations à un serveur avec une file PAPS

Une ressource critique peut être assimilée à un booléen occupé protégé par un moniteur avec une condition appelée libre.

Un processus demandant la ressource se bloque sur cette condition si la ressource est occupée.

Quand un processus libère la ressource, il signale aux éventuels processus en attente qu'il n'en a plus l'utilité, afin de débloquer l'un

d'entre eux : le premier dans la file d'attente associée à la condition libre (politique premier arrivé - premier servi).

La classe prédéfinie calrc de ressources critiques simples avec calendrier, dont on trouvera la description complète en annexe C, a la structure suivante :

```
calendrier class calrc ;
not hidden protected acquérir, libérer, occupé ;
begin
    boolean occupé ;
    ref (condition) libre ;
    procedure acquérir ; ... ;
    procedure libérer ; ... ;
    libre :- new condition (this monitor) ;
    ...
    occupé := false
end
```

Le booléen occupé est true si la ressource est utilisée par un processus, false sinon.

La condition libre sert à bloquer les processus demandeurs de la ressource occupée.

L'appel de la procédure acquérir par un processus permet de lui délivrer la ressource si elle est libre, sinon il le bloque sur la condition libre.

L'appel de la procédure libérer libère la ressource et émet un signal sur la condition libre, afin de débloquer éventuellement le premier processus en attente.

Au calendrier près, une telle ressource est identique aux "facilities" de GPSS (SCH 72, BRA 77).

Exemple d'utilisation 6.12

Dans le modèle décrit au chapitre 5, les ressources désignées par x, z et p sont considérées comme des ressources critiques simples protégées par un calendrier. Pour elles, on a les déclarations et instructions suivantes (annexe B).

```
ref (calrc) x, z, p ;
...
x :- new calrc ('X', true, true, 24, 0) ;
z :- new calrc ('Z', true, true, 24, 0) ;
p :- new calrc ('P', true, false, 24, 0) ;
```

Pour x, par exemple : le premier paramètre donne le nom 'X' de la ressource (classe monitor) ; le deuxième précise que l'on utilise effectivement le mécanisme de calendrier ; le troisième fait qu'en fin de disponibilité, on laisse terminer le processus en cours ; 24 donne la longueur de la période du calendrier et 0 le début de la première période (classe calendrier).

Si l'on désire que x travaille, chaque jour, de 10 à 12 heures et de 14 à 15 heures, z de 15 heures 30 à 17 heures, p de 8 à 12 heures et de 14 à 18 heures, on doit fournir les données suivantes, sous forme de couples (début, fin) de triplets (jours, heures, minutes) :

```
0  10  0   0  12  0   0  14  0   0  15  0   *
0  15  30  0  17  0   *
0  8   0   0  12  0   0  14  0   0  18  0   *
```

Dans le processus enregistrement, on remarque (annexe B) l'acquisition et la libération de x par x . acquérir et x . libérer.

□

2.3.2 Ressources critiques avec priorités : stations à un serveur avec une file à priorités

La description générale est sensiblement la même que celle des précédentes, mais on utilise une politique de gestion avec

priorités pour la file d'attente-condition libre.

La structure de la classe calrcp de ressources critiques avec priorités et calendrier, dont on trouvera la description complète en annexe C, est la suivante :

```
calendrier class calrcp ;
not hidden protected acquérir, libérer, occupé ;
begin
    boolean occupé ;
    ref (conditionp) libre ;
    procedure acquérir ; ... ;
    procedure libérer ; ... ;
    libre :- new conditionp (this monitor) ;
    ...
    occupé := false
end
```

La condition libre est cette fois une condition avec priorités et délais, utilisée dans un cas particulier.

L'appel de la procédure acquérir a le même effet que pour calrc, à ceci près que, dans libre, les processus sont rangés par priorités décroissantes.

L'appel de la procédure libérer a le même effet que dans calrc.

2.3.3 Ressources à accès multiples : stations à n serveurs avec une seule file

Le nombre total d'accès (ou de serveurs) est donné par le paramètre réel quantité et le nombre d'accès utilisés par l'entier occupés.

Un processus peut demander plusieurs accès à la fois. Si le nombre d'accès disponibles ne suffit pas à satisfaire la demande, le processus est bloqué sur une condition ressuffi, où les processus sont rangés par ordre d'arrivée.

Au moment de la libération de plusieurs accès, un signal général (csignalg) est fait sur la file ressuffi de manière à libérer le maximum de processus, dans un ordre qui n'est plus nécessairement premier arrivé - premier servi et favorise les "petits" demandeurs.

La classe calram de ressources à accès multiples avec calendrier, dont on trouvera le texte complet en annexe C, est la suivante :

```
calendrier class calram (quantité) ; real quantité ;
not hidden protected acquérir, libérer, occupés ;
begin
    real occupés ;
    ref (condition) ressuffi ;
    procedure acquérir (x) ; real x ; ... ;
    procedure libérer (x) ; real x ; ... ;
    ...
    ressuffi :- new condition (this monitor) ;
    occupés := 0
end
```

L'appel de la procédure acquérir par un processus a pour effet de lui délivrer x accès ou de le bloquer sur ressuffi s'il n'y a pas x accès disponibles.

L'appel de la procédure libérer provoque la libération de x accès et l'envoi d'un signal à tous les processus en attente sur ressuffi afin d'en débloquer le maximum.

Au calendrier près, une telle ressource est identique aux "storages" de GPSS (SCH 72, BRA 77).

Dans la version actuelle de notre système, seul ce type de ressources à accès multiples a été réalisé. On peut bien sûr en imaginer d'autres plus complexes. L'introduction d'une politique de file avec priorités ne pose guère de problème ; en revanche, la prise en compte de la préemption est beaucoup plus délicate et s'apparente aux problèmes d'ordonnancement (KLE 79).

2.3.4 Ressources à nombre infini de points d'accès : âtations à nombre de serveurs infini

Ce type de ressources a été introduit essentiellement à des fins de comptabilité et de détermination de temps de réponse.

C'est un cas particulier du précédent, où quantité serait infiniment grand et où chaque processsus ne demanderait qu'un accès à la fois. La structure de la classe calraminf de ressources à nombre d'accès infini avec calendrier, dont on trouvera la description complète en annexe C, est la suivante :

```
calendrier class calraminf ;
not hidden protected  acquérir, libérer, occupés ;
begin
      real occupés ;
      procedure acquérir ; ... ;
      procedure libérer ; ... ;
      occupés := 0
end
```

Le réel occupés comptabilise les accès utilisés.

L'appel de la procédure acquérir autorise systématiquement l'usage d'un accès, alors que celui de libérer le restitue. Ce mécanisme ne sert qu'à la prise de mesures et à la détermination de temps de réponse.

Exemple 6.13

Dans l'application du chapitre 5, la ressource ord est considérée comme une ressource à nombre infini d'accès. Sa déclaration et sa création s'écrivent :

```
ref (calraminf) ord ;
...
ord :- new calraminf ('ORDINATEUR', false, true, 0, 0) ;
```

Ici, ord a un nom qui est 'ORDINATEUR', et on inhibe le mécanisme de calendrier, ce qui signifie que l'on considère cette ressource comme toujours disponible.

On peut remarquer dans certains processus l'utilisation conjointe de ord avec d'autres processeurs (annexe B). Ainsi, dans décision, on a la séquence suivante :

```
z . acquérir ;
...
ord . acquérir ;
...
ord . libérer ;
z . libérer ;
```

□

2.3.5 Ressources critiques avec priorités préemptibles : stations à un serveur et à une file avec priorités et préemption

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 sur une file-condition avec priorités et délais. Lors de la libération de la ressource, celle-ci est accordée au premier processus de la file, parmi ceux qui ont la priorité la plus forte.

La classe calrcpp des ressources critiques, avec priorités, préemptibles, et avec calendrier, étant parmi les plus intéressantes, nous donnons son texte complet, à l'exclusion des mécanismes de mesure.

```
calendrier class calrcpp ;
not hidden protected  acquérir, libérer, occupant ;
begin
      ref (processus) occupant ;
      ref (conditionp) tour ;
      procedure acquérir ;
      begin
            this monitor . enter  ;
            calconsulter ;
            if occupant = / = none then
               begin
                     if occupant . priorité < current qua processus .
                                                              priorité
                     then tour . cwait (occupant, occupant . priorité)
                     else tour . cwait (current, current qua processus .
                                                              priorité)
               end ;
            occupant :- current ;
            this monitor . leave
      end ; $ acquérir $
```



```
      procedure libérer ;
      begin
            this monitor . enter ;
            if  tour . cempty then occupant :- none
            else begin
                  occupant :- tour . cfirst ;
                  tour . csignal
               end ;
            calsortir ;
            this monitor . leave
      end ; $ libérer $
      tour :- new conditionp (this monitor)
end
```

occupant est la référence au processus qui occupe la ressource critique.

tour est la condition avec priorités et délais permettant de bloquer les processus non prioritaires.

L'appel de la procédure acquérir donne la ressource critique au processus le plus prioritaire et fait attendre l'autre sur tour.

L'appel de la procédure libérer libère la ressource critique et la donne au premier processus de tour s'il existe.

Exemple 6.14

Dans l'application décrite au chapitre 5, la ressource y est considérée comme une telle ressource. On la déclare et on la crée par :

```
ref (calrcpp) y ;
...
y :- new calrcpp ('Y', true, true, 24, 0)
```

Le nom de y est 'Y' ; elle utilise le mécanisme de calendrier sans que celui-ci interrompe un travail en cours, la longueur de la période est 24 heures à partir de 0 heure.

Si l'on désire que y soit disponible chaque jour de 8 à 12 heures et de 14 à 18 heures, on lui associe les données suivantes :

*0 8 0 0 12 0 0 14 0 0 18 0 **

Dans les processus saisie et décision (annexe B), on remarque l'acquisition et la libération de y par y . acquérir et y . libérer.

□

2.3.6 Processeurs partagés : stations avec un serveur partagé selon la méthode du tourniquet

Nous avons cherché à définir un type de processeurs pouvant éventuellement être partagés entre plusieurs processus, chacun utilisant une telle ressource en exclusion mutuelle pendant un quantum de temps, selon la technique du "tourniquet" ("round-robin").

Un tel processeur est une ressource critique (booléen occupé) protégée par un moniteur, avec une condition libre.

La structure de la classe calcpp de processeurs partagés avec calendrier, dont on trouvera la description complète en annexe C, est la suivante :

```
calendrier class calcpp (k) ; real k ;
not hidden protected acquérir, libérer, n ;
begin
    boolean occupé ;
    integer n ;
    ref (condition) libre ;
    procedure acquérir (x, qu) ; name qu ;
        real x, qu ; ... ;
    procedure libérer (x, qu) ; name x ;
        real x, qu ; ... ;
```



```
    occupé := false ;
    n := 0 ;
    libre :- new condition (this monitor)
end
```

k est l'intervalle de temps à partager entre les processus ;

occupé indique si le processeur est occupé à un instant donné ;

n représente le nombre de processus qui se partagent l'intervalle k.

libre est la file-condition sur laquelle sont bloqués les processus demandeurs, si le processus n'est pas libre.

L'appel de la procédure acquérir provoque, si le processus 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.

L'appel de la procédure libérer provcque la libération de la ressource, la mise à jour du temps résiduel et un csignal sur la condition libre.

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

```
procedure utiliserpp (t, pp) ; real t ; ref (calcpp) pp ;
begin ... end
```

L'appel de cette procédure (détaillée dans (KLE 79)) par un processus lui permet d'utiliser le processeur partagé référencé par pp pendant un temps absolu de t, en concurrence avec d'autres processus.

Notons que la réalisation d'un processeur partagé avec la gestion d'une file d'attente avec priorités, qui n'est pas effective dans la version actuelle, ne pose aucun problème technique. Elle consiste simplement à fusionner dans la même classe, les classes calrcp et calcpp.

3 - STRUCTURE de la CLASSE MAESTRO

Dans ce paragraphe, nous faisons une synthèse de ce que nous venons de présenter, par un rappel des éléments standard principaux qui interviennent dans la classe MAESTRO. Nous ne faisons ressortir ici que l'enchaînement de leur construction, en ignorant les classes, procédures, objets secondaires, et tous les outils de mesure.

```
SIMULATION class MAESTRO ;
begin
    ...
    link class monitor (nom) ; text nom ; ... ;
    class condition (m) ; ref (monitor) m ; ...;
    class conditionp (m) ; ref (monitor) m ; ... ;
    ------------------------------------------------
    link class message (no) ; integer no ; null ;
    message class transaction (pds) ; real pds ; .... ;
    monitor class tfile (mesure, équilibre, inter, dmes) ;
        boolean mesure, équilibre ; real inter, dmes ;... ;
    monitor class actmonitor (type) ; integer type ; ... ;
    process class processus (nom, priorité, t) ;
        text nom ; real priorité, t ; null ;
    class échéancier (per, hdp) ; integer per, hdp ; ... ;
    ------------------------------------------------
    monitor class ressource ; ... ;
    ressource class calendrier (utilise, termine, per, hdp) ;
        boolean utilise, termine ; integer per, hdp ; ... ;
    calendrier class calrc ; ... ;
    calendrier class calrcp ; ... ;
    calendrier class calram (quantité) ; real quantité ; ... ;
    calendrier class calraminf ;... ;
```



```
    calendrier class calrcpp ; ... ;
    calendrier class calcpp (k) ; real k ; ... ;
    procedure utiliserpp (t, pp) ; real t ; ref (calcpp) pp ; ... ;
    ...
end $ MAESTRO $
```

Cette classe est précompilée et cataloguée sur l'IRIS-80 de l'IUCAL. Elle peut servir de prologue à des programmes SIMULA 67 quelconques. La dernière version de cette classe, qui a servi pour l'exemple du chapitre 5, est présentée complètement en annexe C.

Partie III

UTILISATION de MAQUETTES de SYSTEMES d'INFORMATION

Chapitres :

Partie III

UTILISATION de MAQUETTES de SYSTEMES d'INFORMATION

Après avoir dit comment on peut globalement décrire un système d'information par une maquette (chapitre 5), et montré la diversité des composants possibles (chapitre 6), nous traitons ici de l'utilisation de maquettes pour la description et l'évaluation des systèmes.

Il existe plusieurs types d'évaluations. Nous exposons les problèmes qu'elles posent et montrons en quoi des maquettes peuvent aider à les résoudre. Nous donnons quelques critères pour choisir des composants adéquats, nous précisons avec quels moyens et outils aborder ces questions.

Nous examinons successivement la description et l'évaluation structurale (chapitre 7), l'évaluation fonctionnelle qualitative (chapitre 8) et l'évaluation quantitative de comportement dynamique (chapitre 9).

Chapitre 7 - DESCRIPTION et EVALUATION STRUCTURALE d'un SYSTEME D'INFORMATION

"Tamino :

Dies Bildnis ist bezaubernd schön,
Wie noch kein Auge je gesehn ! "

(W.A. Mozart et E. Schikaneder, la Flûte Enchantée, acte 1, scène 4)

"Tamino (contemplant le portrait de Pamina) :

Cette image est merveilleusement belle,
Personne n'en a jamais vu de pareille ! "

Chapitre 7 - DESCRIPTION et EVALUATION STRUCTURALE d'un SYSTEME d'INFORMATION

1. QUELQUES PROBLEMES "SYNTAXIQUES"

   1.1 Point de vue logique.

   1.2 Point de vue physique.

2. SOLUTIONS APPORTEES PAR LES MAQUETTES

   2.1 Point de vue logique.

   2.2 Point de vue physique.

3. NIVEAU DE REDUCTION DANS UNE DESCRIPTION

   3.1 Système d'information logique.

      3.1.1 Structures de données.

      3.1.2 Processus.

      3.1.3 Protection, blocage.

      3.1.4 Echéanciers.

   3.2 Système d'information physique.

      3.2.1 Personnes.

      3.2.2 Services.

      3.2.3 Matériels.

      3.2.4 Horaires et calendriers.

La question est ici de savoir si la structure d'un système d'information (existant ou futur) est "satisfaisante".

Ce qui conduit à s'interroger sur les critères de satisfaction et à remettre immédiatement en cause la classification adoptée : un bon fonctionnement logique et un comportement dynamique correct ne sont-ils pas en effet garants d'une "bonne" structure ?

Cette discussion rejoint tout à fait celle que l'on pourrait avoir en informatique au sujet des "bons" programmes. Pour ceux-ci, on sait bien qu'il y a plusieurs critères : une syntaxe correcte et "agréable" (style de programmation), un algorithme correct (prouvé ou testé), une exploitation efficace (programme peu encombrant et rapide, ce qui est d'ailleurs souvent contradictoire) ...

Ici, nous nous préoccupons essentiellement du premier aspect : la correction de la syntaxe et le "style" de structure.

Il nous semble en effet qu'un certain nombre de problèmes simples posés dans ce cadre sont partiellement résolus par l'écriture de maquettes. Sans prétendre à l'exhaustivité, nous examinons successivement problèmes (1) et solutions (2). Puis nous confirmons que la structure d'un système peut être vue à différents niveaux de détail, et que la réduction apportée dans une maquette la fait parfois sensiblement différer de la réalité, ce qui doit inciter à la prudence (3).

1 - QUELQUES PROBLEMES "SYNTAXIQUES"

On peut bien sûr s'interroger sur les composants d'un système d'information et leurs liaisons, d'un point de vue logique et d'un point de vue physique, sous des aspects statiques et dynamiques.

1.1 Point de vue logique

Pour ce qui est des structures de données, se posent par exemple des questions de :

(1) duplication : existe-t-il des structures de données, ou même des données, identiques entretenues en des sites distincts de l'organisation ?

(2) mise à jour : chaque structure possède-t-elle bien des processus de création, suppression, modification ?

(3) protection, blocage : les accès ou modifications dans telle ou telle structure sont-ils réalisés en exclusion mutuelle ? Quels sont les états susceptibles de provoquer des blocages ?

Et, pour les processus :

(4) duplication : existe-t-il dans l'organisation des processus distincts qui accomplissent des tâches analogues ?

(5) déclenchement : les circonstances de déclenchement de chaque processus ont-elles été prévues ?

1.2 Point de vue physique

On peut se demander quel est l'usage qui est fait des ressources sous différents aspects :

(6) horaires : a-t-on défini les horaires de travail des différents services, des différentes personnes, ou les périodes d'utilisation des différents matériels ?

(7) lieux : n'a-t-on pas prévu, pour la même personne, différents

travaux trop rapprochés dans le temps, en des lieux trop éloignés les uns des autres ?

(8) affectation : chaque processus a-t-il au moins une ressource pour l'exécuter ?

2 - SOLUTIONS APPORTEES PAR LES MAQUETTES

Les maquettes sont écrites dans un langage de programmation (SIMULA 67) où l'on fabrique facilement de nouveaux types d'objets.

La construction de ces types et de ces objets aide à s'interroger sur la structure d'un système : elle facilite les comparaisons de composants, elle permet de mettre en évidence l'absence de l'un d'entre eux, ou celle d'une liaison indispensable, par des vérifications de nature essentiellement syntaxique.

A titre d'illustration, examinons les réponses aux questions posées précédemment.

2.1 Point de vue logique

(1) C'est en essayant de décrire des types de structures de données, peut-être avec un niveau de détail sémantique relativement fin, que l'on met en évidence des analogies qui permettent de répondre à la question (1).

(2) On peut évidemment commencer la réponse à la question (2) de la même manière. Mais, dans certains cas particuliers, une attitude plus radicale autorise des contrôles syntaxiques automatiques par le compilateur du langage de description.

Exemple 7.1

Si, dans un système avec des files d'attente, les éléments de chaque file sont produits par un seul processus et consommés par un seul processus, on peut toujours compléter la classe tfile (chapitre 6) par des attributs-paramètres, pour obtenir la classe tfile1, avec comme entête :

tfile classe tfile1 (producteur, consommateur) ;
ref (process) producteur, consommateur ;

Dans le corps de cette classe peuvent être activés les deux processus.

Ainsi la création de files de tfile1 s'accompagne d'une mise en relation nécessaire avec des processus. Le compilateur vérifie que pour toute génération d'objet de ce type, une liaison de cette forme a bien été établie : il suffit de contrôler le nombre et le type des paramètres effectifs.

□

(3) Pour ce qui concerne la protection et le blocage, c'est encore une fois à l'écriture de structures que se révèlent les difficultés.

Ce que nous envisageons dans les maquettes est parfois éloigné de la réalité, mais rien n'empêche de s'en rapprocher beaucoup plus : les moyens d'expression de SIMULA 67 et de la classe MAESTRO permettent de le faire facilement.

Exemple 7.2

Si l'on désire décrire finement les modifications sur un fichier, on peut envisager un verrouillage au niveau des blocs, ou même des enregistrements logiques.

Dans ce dernier cas, par exemple, il suffit d'associer à chaque enregistrement un moniteur réalisant le blocage désiré.

□

(4) Pour ce qui concerne les processus, c'est aussi en écrivant des modèles que l'on peut se rendre compte de leur similitude, et, éventuellement, de leur redondance.

(5) La certitude que l'on a réfléchi aux mécanismes de déclenchement peut être aussi obtenue par une vérification automatique pendant l'analyse syntaxique.

Exemple 7.3

Si la continuation de certains processus est soumise périodiquement à la consultation d'un moniteur d'activation (chapitre 6), on peut paramétrer une classe de processus comme suit :

process class processus1 (m) ; ref (actmonitor) m ;... ;

A chaque écriture d'un générateur de tels processus, il faut donner la référence du moniteur d'activation associé. Ceci permet de vérifier qu'il existe bien un lien entre tout processus et un moniteur d'activation.

□

2.2. Point de vue physique

Ici encore, l'effort demandé pour décrire dans une maquette les différents composants se révèle intéressant pour améliorer une analyse. La syntaxe retenue pour certains types d'objets apporte des contraintes, mais celles-ci permettent des vérifications par un compilateur du langage de description.

(6) En utilisant des ressources avec calendriers, l'utilisateur du système MAESTRO est conduit à s'interroger sur les horaires de travail ou de disponibilité des différents intervenants.

Rappelons qu'il a le choix entre utiliser le mécanisme de calendrier tel qu'il a été défini (avec blocage des travaux en fin de disponibilité ou

non), ou de l'inhiber et définir sa propre gestion temporelle.

Exemple 7.4

Dans la maquette du chapitre 5, nous avons vu que certaines personnes étaient représentées comme des ressources critiques, simples ou avec priorités et préemption. Dans les deux cas, nous leur avons associé un calendrier fixant leurs périodes de disponibilité pour le système considéré.

La ressource ord en revanche a été prise avec un nombre infini de points d'accès et sans calendrier, parce qu'on la considère toujours disponible. Elle est utilisée essentiellement pour faire des statistiques.

□

(7) Après que l'on ait fixé le nombre et le type des ressources, la détermination des données horaires (débuts et fins d'intervalles de disponibilité) doit être compatible avec le lieu de travail.

Exemple 7.5

Si dans un système, une personne est chargée de n tâches (processus) rigoureusement planifiés dans le temps, on la représente dans une maquette par n ressources critiques simples avec calendrier. Chaque calendrier contient les intervalles de disponibilité pour la tâche associée. Si les tâches sont exécutées à différents endroits assez éloignés, on prend garde aux temps de déplacement en fixant les horaires dans les calendriers.

□

(8) La mise en liaison d'un processus avec une ou plusieurs ressources peut être systématisée grâce à la déclaration de classes paramétrées.

Exemple 7.6

Si l'on décide d'associer aux processus une ressource et une seule, ils peuvent être définis comme des objets de la classe :

process class processus2 (r) ; ref (ressource) r ; ... ;

La vérification syntaxique est de même nature que les précédentes.

□

3 - NIVEAU DE REDUCTION DANS UNE DESCRIPTION

On peut s'intéresser à la structure d'un système à différents niveaux et points de vue. Après avoir vu ce qu'est une maquette, nous revenons ici sur les principes de réduction qui ont été émis au début de l'exposé (chapitre 3).

3.1 Système d'information logique

3.1.1 Structures de données

Nous avons vu dans l'exemple du chapitre 5 que la réduction sémantique des informations dans une maquette peut être importante. Bien que ce modèle soit plutôt destiné à l'évaluation quantitative, on peut cependant estimer qu'il est structuralement parlant à un certain niveau.

Exemple 7.7

Examinons les simplifications faites dans l'application du chapitre 5. Le cheminement des différentes transactions (bons de commandes, de livraison) y est assez fidèlement reproduit. Ces documents ne sont décrits que par quelques fonctions très synthétiques (notamment numéro et nombre de lignes commandées ou livrées), en négligeant toute référence aux clients et aux produits. Bien sûr, ces renseignements auraient pu être ajoutés, avec des structures clients et produits (comme dans l'exemple 1.1 du chapitre 1). Mais on n'y a aucun intérêt si l'on s'intéresse seulement à la

structure du chemin suivi par les transactions : alors, la description détaillée de toutes les informations ne ferait qu'obscurcir le modèle et le rendre inefficace.

□

3.1.2 Processus

Leur niveau de détail est nécessairement lié à celui des données, et l'on peut faire pour eux les mêmes remarques.

Exemple 7.8

Une partie du processus décision (chapitre 5) consiste dans la réalité en une confrontation des commandes avec le stock de produits pour une édition d'ordres de livraison, opération dont on sait qu'elle peut obéir à des règles d'affectation fort compliquées. Cette procédure comporte souvent une décision humaine pour favoriser ou retarder des commandes, selon des critères difficilement explicitables en totalité. Ces modifications sont ensuite enregistrées et de nouveaux ordres de livraison édités.

Le processus décision de la maquette décrit ceci à un niveau qui exclut les détails sémantiques. Par exemple, la proposition automatique de livraison consiste simplement à passer en revue les commandes et à appliquer une procédure aléatoire pour déterminer le nombre de lignes de commandes à livrer. On a encore ici favorisé la description du cheminement des transactions au détriment de la sémantique des traitements.

Ceci étant, la description d'une véritable gestion de stock est possible avec nos outils : tout dépend, encore une fois, des objectifs fixés.

□

3.1.3 Protection, blocage

Nous avons déjà fait remarquer que la description de la protection et du blocage est envisageable à différents niveaux. Il suffit

d'associer à chaque composant, que l'on veut ainsi protéger, un moniteur qui l'enveloppe ou non, selon le type de protection désiré (chapitre 5).

3.1.4 Echéanciers

Ce mécanisme est utilisable pour lancer des processus à dates déterminées. On peut lui reprocher un caractère un peu figé, mais il correspond à ce que l'on trouve souvent dans les organisations : des tâches planifiées à caractère répétitif.

Sans doute faut-il parfois assouplir ce déclenchement à dates fixes. Il suffit par exemple de faire attendre pendant une durée aléatoire un processus après son déblocage (à une date "au plus tôt"). De toutes façons, les temps de service aléatoires en atténuent toujours la rigidité.

3.2 Système d'information physique

Toutes les ressources sont modélisées par des stations de différents types, gérées indépendamment les unes des autres, éventuellement protégées par des calendriers fixant leurs périodes de disponibilité.

Nous proposons ici un essai de typologie des modes de travail dans une entreprise, pour aider au choix de ces stations. S'il s'agit de travaux manuels et intellectuels, on peut voir les ressources à deux niveaux : personne et service (groupe de personnes). S'il s'agit de travaux automatiques, on peut considérer globalement les matériels ou entrer dans le détail de leurs composants.

Nous nous interrogeons aussi sur la capacité des calendriers à décrire la gestion des horaires.

3.2.1 Personnes

- Dans un système, une personne peut exécuter ce que l'on considère comme un processus unique. On la représente alors par une ressource critique simple, éventuellement avec calendrier.

Exemple 7.9

C'est le cas d'un ouvrier travaillant à la chaîne ou d'un employé au guichet d'une banque (malgré la diversité des transactions, le niveau d'observation peut le faire considérer ainsi).

□

- Une personne peut exécuter plusieurs processus strictement planifiés dans le temps. Si, à chacun, on peut associer des intervalles de temps, on modèlise la personne par autant de ressources critiques avec calendrier qu'il y a de processus. Toutefois, il faut prendre garde que les intervalles de disponibilité de ces différentes ressources soient disjoints.

Exemple 7.10

Certains petits commerçants ont organisé de façon immuable leur journée par tranches horaires pour différentes tâches : réception de la marchandise, étiquetage, vente, passage des commandes, comptabilité...

□

- Une personne peut exécuter des processus multiples, à la demande, sans priorité. On utilise alors une ressource critique simple, éventuellement avec calendrier. Cette politique de gestion comporte évidemment le danger de monopolisation par un gros travail pour un temps assez long. Mais la réalité peut être conforme à ce modèle.

Exemple 7.11

Un cas très typique est celui d'une dactylo dans un pool, quand tout travail est banalisé.

□

- Une personne peut exécuter des processus multiples avec priorités, pour éviter l'écueil précédent. Eventuellement on peut y ajouter la préemption. Le modèle est évidemment ici une ressource critique avec priorités, et, éventuellement, préemption ou calendrier.

Dans notre système actuel, nous n'avons retenu pour de telles ressources que des priorités fixes pouvant être arbitraires ou fonctions de facteurs tels que : urgence, durée des travaux ... D'autres politiques fondées sur des priorités variables sont possibles : priorités augmentant avec les temps d'attente, notamment.

Exemple 7.12

Une standardiste qui s'occupe également du tri du courrier est susceptible d'un tel modèle : les appels téléphoniques sont prioritaires vis à vis du tri, et doivent le préempter quand ils interviennent.

□

- Quand le travail d'une personne est très diversifié et qu'il est difficile de fixer des règles de gestion précises, on peut éventuellement la modéliser par un processus partagé selon la règle du tourniquet.

Celle-ci peut être utilisée avec des priorités, mais la préemption est inutile si le quantième de temps est très petit.

Exemple 7.13

Dans un secrétariat, une dactylo, qui veut ne favoriser ni ne défavoriser personne, peut exécuter successivement un morceau de chaque travail de frappe.

□

En fait, ce modèle global n'est qu'une des innombrables manières de tenir compte de l'augmentation du temps de réponse quand le nombre des clients dans une file s'accroît.

- Enfin, si une personne travaille avec des méthodes très différentes pendant des périodes bien déterminées, on la représente par autant de ressources, avec des calendriers ayant des intervalles de disponibilité disjoints.

3.2.2 Services

Nous parlons ici d'un service comme d'un groupe de personnes interchangeables, c'est-à-dire capables d'effectuer les mêmes tâches. Un découpage judicieux d'un véritable service de l'organisation est parfois nécessaire pour satisfaire cette exigence d'homogénéité.

En général, un service doit partager son travail entre plusieurs processus et l'on sait que de multiples politiques de gestion sont possibles.

Dans notre système, nous avons pour l'instant retenu deux modes de représentation.

- Quand les travaux demandés par un service sont standard, sans que l'on puisse leur affecter de priorités, le modèle est une ressource à points d'accès multiples.

Le nombre d'accès est le nombre de personnes du service et plusieurs peuvent être demandées en même temps par un processus. Eventuellement, un calendrier est associé au service pour fixer ses périodes de disponibilité.

Exemple 7.14

Dans la partie du magasin d'une entreprise chargée de la réception et de l'expédition, tous les magasiniers peuvent constituer un service au sens ci-dessus, alors que réception et expédition sont des processus concurrents.

□

- Quand la diversité des travaux ou la variété des modes de gestion interne est telle qu'il est vain de vouloir trouver des règles fines, un modèle global est le processus partagé fonctionnant selon la règle du tourniquet, avec ou sans calendrier.

Eventuellement, la gestion de la file d'attente des processus peut être effectuée en respectant des priorités. Ce modèle n'est pas réalisé dans la version actuelle, mais il ne pose aucun problème technique.

Exemple 7.15

Un cas particulièrement bien connu des informaticiens est le pool de perforatrices ou encodeuses d'une entreprise. Si les travaux sont très bien planifiés, aucune priorité ne se fait sentir, sinon, il faut en tenir compte.

□

3.2.3 Matériels

Pour rendre compte dans une maquette de l'utilisation du matériel, la difficulté est encore de savoir à quel niveau se placer. Nous anticipons ici un peu sur l'évaluation quantitative.

On entre en effet dans le domaine de l'évaluation des systèmes informatiques par des modèles souvent fondés sur des réseaux de files d'attente. L'une des premières tentatives est celle du modèle à serveur central de BUZEN (BUZ 71). Il a donné lieu par la suite à des développement portants qui se poursuivent à l'heure actuelle par la modélisation de réseaux d'ordinateurs ou de micro-processeurs.

Dans un tel modèle, on trouve en général un certain nombre de stations : serveur central (unité centrale), processeurs d'entrées-sorties disques, processeur de pagination, consoles... avec des caractéristiques précises de mode d'allocation et de rapidité. On étudie alors le comportement du

modèle en fonction d'une charge de programmes avec des propriétés connues : loi de probabilité des temps d'utilisation de l'unité centrale, lois des entrées-sorties, des demandes de pages, éventuellement loi du comportement d'usagers à des consoles...

Ces études concernent des événements qui se situent à des échelles de temps microscopiques (autour de 10^{-4} seconde). Il est donc hors de question d'inclure dans nos maquettes de tels modèles et il nous faut considérer les choses bien plus globalement, en tirant profit des résultats obtenus à partir de ces études.

Pour l'instant, ordinateurs et réseaux d'ordinateurs sont modélisés de la façon suivante dans nos maquettes.

a) Ordinateurs

- En monoprogrammation, un ordinateur peut être considéré comme une ressource critique ininterruptible, ce qui ne pose guère de problème. C'est un mode d'exploitation qui tend à disparaître pour les ordinateurs, mais qui réapparait avec les micro-processeurs.

- En multiprogrammation, ou temps partagé, il faut considérer des modèles globaux, approximatifs, capables de fournir des temps de réponse convenables (c'est ce qui nous importe le plus en l'occurrence) :

. processeur partagé, avec ou sans priorités ;
. ressource à nombre d'accès infini.

Le temps de réponse obtenu par un processus dépend naturellement du temps absolu d'exécution (que l'on suppose calculable) et du degré de multiprogrammation (c'est-à-dire du nombre de processus qui sont en concurrence sur la ressource).

b) Réseaux d'ordinateurs

Nous ne parlons ici que de la technique de commutation de

paquets. Nous intéressant essentiellement au temps de réponse, nous retenons pour modèle une ressource à nombre infini de points d'accès. Mais, il y a plusieurs manière de considérer l'utilisation d'un réseau

- Une méthode est d'associer un processus à un transport déterminé de paquets :

. des processus émetteurs produisent des paquets dans une première file ;

. un processus transporteur consomme cycliquement un paquet de cette file, utilise la ressource réseau pendant une durée aléatoire (plus longue pour le 1er paquet), puis envoie le paquet dans une deuxième file ;

. des processus récepteurs consomment alors les paquets de cette file.

On respecte ici la contrainte de réception des paquets dans le même ordre qu'à l'émission.

- Une autre méthode consiste à associer à chaque paquet à transporter un processus :

. les processus émetteurs génèrent et activent périodiquement des processus-paquets ;

. chaque processus-paquet utilise la ressource réseau pendant une durée aléatoire, parallèlement aux autres, puis il se range dans une file d'attente où il se passive ;

. les processus récepteurs consomment alors les paquets de cette file.

- Si l'échelle de temps considérée est moins fine, on peut tout simplement admettre qu'on transporte d'un seul coup tout un fichier.

Dans ce cas, on a un processus transporteur qui se saisit de tout le fichier, utilise la ressource réseau, et transmet le fichier à un récepteur.

La durée d'utilisation de la ressource globale réseau risque de différer d'une méthode à l'autre, à moins que l'on utilise cette ressource

plusieurs fois en même temps, en fonction du nombre de paquets transmis. Le temps de réponse peut être aussi modulé en fonction de la charge du réseau, si on la connait avec son influence sur les performances.

Beaucoup d'études doivent encore être menées pour affiner ces modèles.

3.2.4 Horaires et calendriers

Dans sa version actuelle, le système MAESTRO ne permet pas de résoudre tous les problèmes qui se posent au sujet des horaires.

Nous avons déjà signalé qu'il y faudrait des procédures de modification des dates dynamiques : s'il n'y a pas de difficulté informatique importante, il existe un problème du côté de la validité des mesures que l'on peut effectuer (cf. chapitre 9).

De plus, rien n'y est encore défini pour un phénomène qui se développe dans les entreprises à la satisfaction des employés : celui des horaires variables, ou "à la carte". Pour le résoudre, il suffit certainement d'introduire des dates de début et fin d'intervalles de disponibilité aléatoires. Les difficultés informatiques, ici encore, sont négligeables, et le seul vrai problème est l'estimation statistique des aléas. Ceux-ci devraient ensuite se traduire, dans la classe calendrier, par l'adjonction de paramètres supplémentaires donnant quelques-uns de leurs moments.

Chapitre 8 - EVALUATION du FONCTIONNEMENT LOGIQUE d'un SYSTEME d'INFORMATION

"Don Juan :

Va bene in verita "

(W.A. Mozart et L. Da Ponte, Don Juan, acte 1, scène 14).

"Don Juan (au bal) :

Ça marche bien en vérité "

Chapitre 8 - EVALUATION du FONCTIONNEMENT LOGIQUE d'un SYSTEME d'INFORMATION

1. QUESTIONS sur le FONCTIONNEMENT LOGIQUE

2. REPONSES APPORTEES par les MAQUETTES

2.1 Correction individuelle.

2.1.1 Procédures et processus automatiques.

2.1.2 Procédures et processus non automatiques.

2.2 Correction globale.

2.2.1 Partie automatique.

a) Interblocage et famine.

b) Tests d'essai.

2.2.2 Partie non automatique.

a) Interblocage et famine.

b) Tests d'essai.

2.2.3 Correction d'une maquette.

2.2.4 Trace de l'exécution d'une maquette.

Nous nous intéressons ici à la correction du fonctionnement logique d'un système d'information.

De toute évidence, il faut faire une distinction entre la partie automatique d'un système, pour laquelle on peut, sinon résoudre les problèmes, du moins les poser, et la partie non automatique où les choses sont plus floues.

Comme précédemment, nous posons un certain nombre de questions (1) et montrons comment des maquettes peuvent y répondre, au moins partiellement (2).

Les techniques que nous développons ici sont propres à l'évaluation qualitative. Cependant, l'évaluation quantitative (chapitre 9) peut aussi être un moyen pour valider (ou plutôt invalider) le fonctionnement logique d'un système.

1 - QUESTIONS sur le FONCTIONNEMENT LOGIQUE

Nous étudions le fonctionnement logique d'un système réel, où sont essentiels les aspects liés aux structures de données, aux processus et aux structures de contrôle. Mais nous devons parler aussi du fonctionnement d'une maquette représentant un tel système, où l'aspect ressources physiques est aussi très important.

Deux grands groupes de questions nous semblent devoir être posées dans ce cadre :

(1) correction individuelle : les procédures et processus manuels et automatiques d'un système, pris individuellement, sont-ils corrects ?

(2) correction globale : le système fonctionne-t-il globalement convenablement ? Y a-t-il des risques d'interblocage ou de famine de processus ?

2 - REPONSES APPORTEES par les MAQUETTES

Un certain nombre de réponses sont celles que l'on pourrait donner pour tout système de processus coopérants et concurrents, par exemple pour les systèmes d'exploitation d'ordinateurs, mais d'autres sont plus spécifiques du domaine de l'organisation administrative.

2.1 Correction individuelle

On doit distinguer dans un système réel les procédures et processus automatiques des autres.

2.1.1 Procédures et processus automatiques

Les appels de procédures ou les processus sont des instances de programmes informatiques, pour lesquels on dispose, même s'ils sont limités, de certains moyens de validation. Ce sont notamment :

- les preuves de programmes, qui servent à en vérifier la correction totale (terminaison) ou partielle (conformité aux spécifications si terminaison) ; nous ne développons pas ce point qui sort du cadre de notre étude et renvoyons aux ouvrages spécialisés (par exemple C. LIVERCY (LIV 78)) ;

- les tests d'essai, exécution sur jeux d'essai systématiques qui servent à détecter des erreurs, en l'absence desquelles on induit la correction des programmes ; on peut leur rattacher des méthodes plus récentes, telles que l'évaluation symbolique des programmes.

Dans ces conditions, que peuvent apporter des maquettes ?

Si l'on veut vérifier la correction d'un programme, il semble nécessaire qu'il soit écrit complètement : ici, la réduction ne semble plus guère avoir de sens. Pourtant, on parle beacoup de programmation à plusieurs niveaux d'abstraction, ce qui est une notion très proche, dont nous avons montré qu'elle avait sa place dans notre cadre (chapitre 6). Alors, bien que manquant d'expérience sur le sujet, nous pensons que les concepts et méthodes pour construire proprement et efficacement des programmes avec plusieurs niveaux d'abstraction doivent pouvoir s'intégrer à terme dans notre approche.

2.1.2 Procédures et processus non automatiques

Une des hypothèses de base de notre travail est que les traitements de l'information manuels ou intellectuels peuvent être décrits par des programmes informatiques.

Dans ces conditions, rien n'interdit d'appliquer à ceux-ci les méthodes de validation de programmes décrites ci-dessus.

Mais, il faut être fermement convaincu de la similitude profonde entre les processus réels et leurs modèles, pour induire de la correction de ceux-ci la correction de ceux-là.

2.2 Correction globale

Tout ce qui vient d'être dit sur les processus pris individuellement se trouve amplifié ici, avec d'autres problèmes spécifiques.

A nouveau, nous distinguons les parties automatiques et non automatiques, puis nous examinons la correction d'une maquette.

2.2.1 Partie automatique

a) Interblocage et famine

Quand on construit la partie automatique d'un système, on ne se préoccupe que de sa correction d'un point de vue logique, puisque l'allocation des ressources informatiques est en principe réglée automatiquement, après l'implantation des processus sur des machines, par les systèmes d'exploitation.

Les incidents qui risquent d'affecter un tel système de processus logiques sont l'interblocage et la famine vis à vis de l'accès à des structures de données communes. Pour ces deux problèmes, des solutions de prévention et de guérison existent dans des contextes de contrôle centralisé (CRO 75, BRI 73), et même maintenant décentralisé, pour des systèmes répartis (LEL 78, DAR 79). Nous ne traitons pas ici de ces questions.

Des maquettes permettent de décrire le parallélisme et la synchronisation de processus grâce aux moniteurs et conditions, en respectant des contraintes pour prévenir l'interblocage. En effet, de tels mécanismes sont insuffisants en eux-mêmes pour l'éviter et l'emploi dans ce cadre d'un système de preuves formelles, tel celui donné par C.A.R. HOARE (HOA 74), ne suffit pas à prouver l'interblocage ou son absence.

Sur la question des preuves de systèmes de processus avec des moniteurs ou des classes, on lira le travail de S. OWICKI (OWI 78).

Des méthodes s'appuyant sur les réseaux de PETRI (chapitre 4) pourraient donner des résultats plus intéressants, quoiqu'encore limités (PET 77).

b) Tests d'essai

Un autre moyen permettant de détecter des erreurs dans des systèmes peut être utilisé sur des maquettes ; c'est encore le test d'essai, global ou par parties.

Il est souvent nécessaire de compléter la maquette du système que l'on étudie par des composants complémentaires qui en réalisent la clôture (chapitre 9). On adjoint notamment les processus d'alimentation en données d'essai ou de vidage de structures.

La maquette close est ensuite exécutée en simulation, grâce à une méthodes de quasi-parallélisme. Pendant cette exécution, il est possible d'éditer une trace et des analyses des données de certaines structures, ce qui permet de détecter des erreurs (2.2.4).

2.2.2 Partie non automatique

a) Interblocage et famine

Ces phénomènes peuvent naturellement se concevoir dans des procédures manuelles et intellectuelles. Aux risques d'interblocage et famine "logiques" comme dans la partie automatique, il faut ajouter ceux sur les ressources, puisqu'il n'y a pas de "système d'exploitation".

Voire !... En effet, si le fonctionnement de l'organisation n'est pas trop dégradé, les conflits sont résolus systématiquement, par un accord des personnes concernées ou un événement extérieur.

Aussi peut-on admettre que, dans cette partie d'un système, on ne doit pas avoir de phénomène d'interblocage ni, à un degré moindre, de famine.

On prend donc garde à les éviter dans une maquette de système d'information "normal".

b) Tests d'essai

De la même manière que pour la partie automatique d'un système, des tests d'essai peuvent être conduits sur une maquette de la partie non automatique sur des modèles recouvrant éventuellement les deux aspects.

2.2.3 Correction d'une maquette

On voit donc que, mis à part des cas pathologiques où les conditions d'interblocage seraient l'objet même de l'étude, on est conduit dans une maquette à les prévenir.

Naturellement, l'utilisateur est libre de la technique qu'il utilise, et même d'avoir des possibilités d'interblocage s'il le désire.

Nous recommandons cependant la méthode de prévention d'interblocage la plus simple qui soit, pour toutes les ressources confondues (structures de données bloquantes, processeurs) :

utiliser une seule ressource à la fois.

On entend ici de "vraies" ressources, susceptibles d'entraîner un blocage, et pas des ressources pour les statistiques ou le paramétrage, telles les ressources à nombre infini de points d'accès sans calendrier (calraminf dans MAESTRO). Celles-ci peuvent en effet être utilisées en même temps que d'autres sans problème (sinon celui de fausser des statistiques s'il y a blocage pendant leur utilisation).

Toute autre politique de prévention, statique du moins, risque d'être vouée à l'échec du fait des phénomènes de préemption ou d'interruption par les calendriers, puisque l'on a une gestion décentralisée (chapitre 1).

2.2.4 Trace de l'exécution d'une maquette

On peut introduire dans une maquette, à certains endroits "stratégiques", des appels de procédures d'édition de messages, ou de valeurs de certains éléments caractéristiques.

Ainsi, l'appel de la procédure suivante, incorporée au prologue MAESTRO (voir annexe C),

```
procedure passe (lib, b) ; text lib ; boolean b ; ... ;
```

a pour effet d'imprimer :

. la date simulée en clair (jour, heure, minute)

. le libellé "DEBUT DE TRAVAIL DE", ou "FIN DE TRAVAIL DE", suivant que le paramètre effectif correspondant à b est true ou false,

. le texte correspondant au paramètre lib, qui est souvent le nom d'un processus.

Compte-tenu de la limitation actuelle des méthodes de preuves de programmes, il semble bien qu'un moyen aussi rudimentaire qu'une trace reste encore le seul outil utilisable de manière générale.

Exemple 8.1

Nous reprenons ici la maquette du chapitre 5, close par des processus d'alimentation (générateur, génétel) et de vidage (videur1 et 2) (voir chapitre 9 et annexe B). L'introduction d'appels systématiques de la procédure passe et d'autres procédures particulières au modèle permet d'obtenir une trace dont nous extrayons simplement les premières 36 heures.

Elle commence par le rappel des bornes choisies initialement pour les intervalles de disponibilité des calendriers : on remarque que certaines dates sont différentes de celles données au chapitre 6, ceci pour amplifier certains phénomènes.

On y remarque le bon déclenchement des processus selon les conditions précisées aux chapitres 5 et 6, notamment en fonction des dates enregistrées dans l'échéancier éch. Sur la trace figurent les débuts et fins de travail des processus ou parties de processus. C'est ainsi que le processus décision a été découpé en 4 phases : décision1 correspond à l'édition automatique des propositions de livraison, décision2 à leur examen par y, décision3 à la saisie des modifications par z, et décision4 à l'édition des propositions définitives par ord.

On voit aussi la bonne allocation des ressources : par exemple, on peut suivre la préemption sur y et le fonctionnement des calendriers : à la fin des intervalles de disponibilité, on constate que, conformément à ce qui a été défini au chapitre 6, on laisse terminer les processus commencés. C'est pour mieux observer ce phénomène qu'ont été envisagés des horaires particuliers.

On voit aussi comment et quand le modèle est alimenté en commandes, et comment il est vidé des bons de livraison.

□

```
LE CALENDRIER ASSOCIE A LA RESSOURCE   X  EST LE SUIVANT :
 0 10  0   0 12  0   0 14  0   0 15  0
LE CALENDRIER ASSOCIE A LA RESSOURCE   Y  EST LE SUIVANT :
 0  8  0   0 12  0   0 14  0   0 18  0
LE CALENDRIER ASSOCIE A LA RESSOURCE   Z  EST LE SUIVANT :
 0 14  0   0 18  0
LE CALENDRIER ASSOCIE A LA RESSOURCE   U  EST LE SUIVANT :
 0  8  0   0 12  0   0 14  0   0 18  0
```

Exemple 8.1 - Trace - Paramétrage des Calendriers

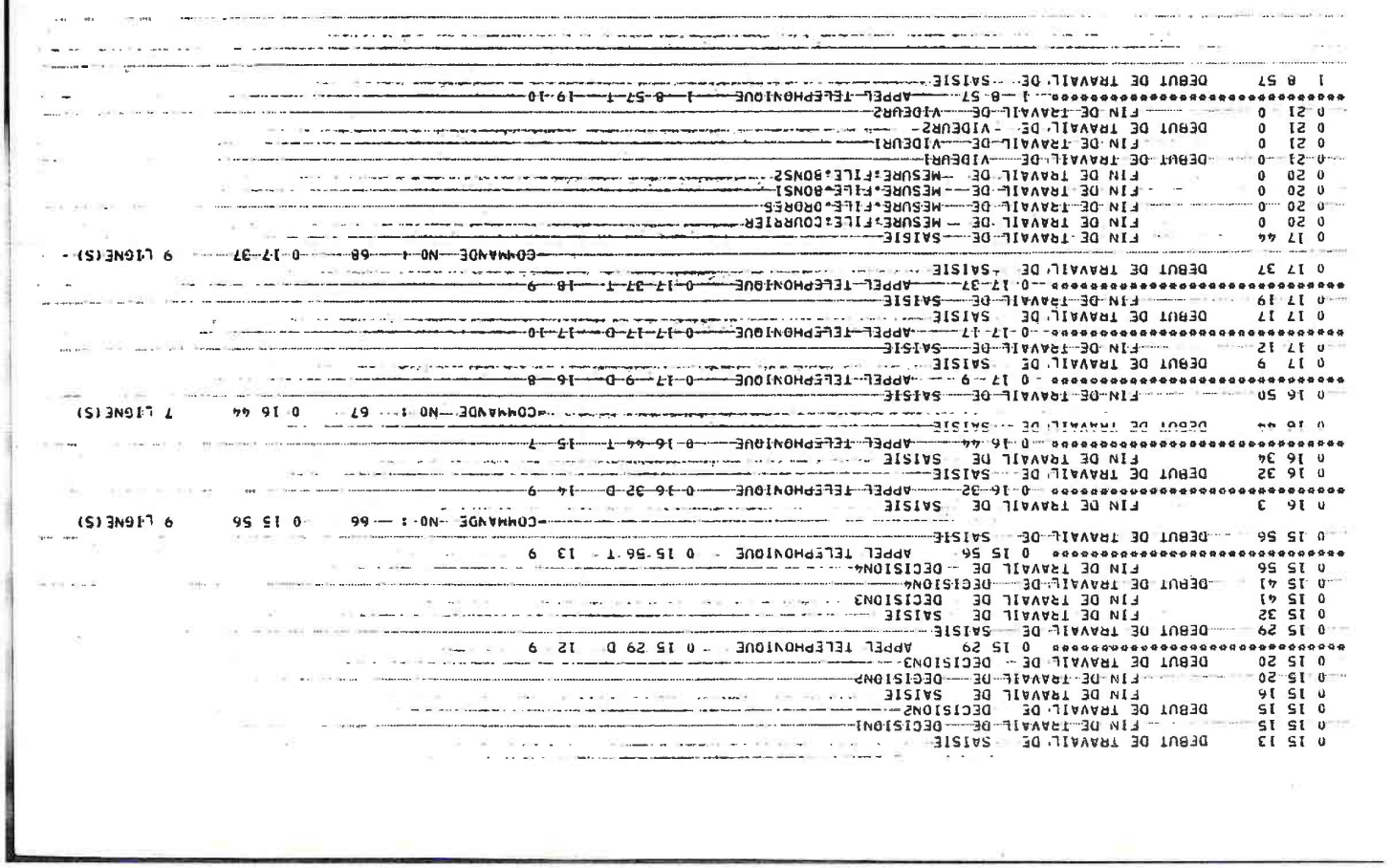
```
                                                                                               9
******************************  0  8 10   APPEL TELEPHONIQUE   0  8 10 T    1  9
 0  8 10     DEBUT DE TRAVAIL DE   SAISIE
                                                         -COMMANDE  NO :   51      0  8 10      9 LIGNE(S)
 0  8 17              FIN DE TRAVAIL DE    SAISIE
******************************  0  8 47   APPEL TELEPHONIQUE   0  8 47 D    2 11
 0  8 47     DEBUT DE TRAVAIL DE   SAISIE
 0  8 49              FIN DE TRAVAIL DE    SAISIE
******************************  0  8 59   APPEL TELEPHONIQUE   0  8 59 D    3  9
 0  8 59     DEBUT DE TRAVAIL DE   SAISIE
```

Exemple 8.1 - Trace

```
  0  9  0      DEBUT DE TRAVAIL DE   GENERATEUR
                                                                                GENERATION DE  11 COMMANDES ECRITES
                                                                              COMMANDE  NO :   52     0  9  0       1 LIGNE(S)
                                                                              COMMANDE  NO :   53     0  9  0       8 LIGNE(S)
                                                                              COMMANDE  NO :   54     0  9  0      13 LIGNE(S)
                                                                              COMMANDE  NO :   55     0  9  0      20 LIGNE(S)
                                                                              COMMANDE  NO :   56     0  9  0      34 LIGNE(S)
                                                                              COMMANDE  NO :   57     0  9  0       8 LIGNE(S)
                                                                              COMMANDE  NO :   58     0  9  0       1 LIGNE(S)
                                                                              COMMANDE  NO :   59     0  9  0      13 LIGNE(S)
                                                                              COMMANDE  NO :   60     0  9  0      18 LIGNE(S)
                                                                              COMMANDE  NO :   61     0  9  0       1 LIGNE(S)
                                                                              COMMANDE  NO :   62     0  9  0       1 LIGNE(S)
  0  9  0            FIN DE TRAVAIL DE   GENERATEUR
  0  9  1            FIN DE TRAVAIL DE   SAISIE
**********************************  0  9  2      APPEL TELEPHONIQUE     0  9  2  T     6  10
  0  9  2      DEBUT DE TRAVAIL DE   SAISIE
                                                                              COMMANDE  NO :   63     0  9  2      10 LIGNE(S)
  0  9  9            FIN DE TRAVAIL DE   SAISIE
**********************************  0  9 36      APPEL TELEPHONIQUE     0  9 36  D     5  6
  0  9 36      DEBUT DE TRAVAIL DE   SAISIE
  0  9 39            FIN DE TRAVAIL DE   SAISIE
  0 10  0      DEBUT DE TRAVAIL DE   ENREGISTREMENT1
**********************************  0 10 44      APPEL TELEPHONIQUE     0 10 44  D     6  10
  0 10 44      DEBUT DE TRAVAIL DE   SAISIE
  0 10 46            FIN DE TRAVAIL DE   SAISIE
  0 10 51            FIN DE TRAVAIL DE   ENREGISTREMENT1
  0 10 51      DEBUT DE TRAVAIL DE   ENREGISTREMENT2
**********************************  0 10 54      APPEL TELEPHONIQUE     0 10 54  T     7  6
  0 10 54      DEBUT DE TRAVAIL DE   SAISIE
                                                                              COMMANDE  NO :   64     0 10 54       6 LIGNE(S)
  0 11  0            FIN DE TRAVAIL DE   SAISIE
**********************************  0 11  5      APPEL TELEPHONIQUE     0 11  5  T     8  8
  0 11  5      DEBUT DE TRAVAIL DE   SAISIE
                                                                              COMMANDE  NO :   65     0 11  5       6 LIGNE(S)
  0 11 10            FIN DE TRAVAIL DE   SAISIE
**********************************  0 11 10      APPEL TELEPHONIQUE     0 11 10  D     9  6
  0 11 10      DEBUT DE TRAVAIL DE   SAISIE
  0 11 13            FIN DE TRAVAIL DE   SAISIE
  0 11 41            FIN DE TRAVAIL DE   ENREGISTREMENT2
**********************************  0 14 39      APPEL TELEPHONIQUE     0 14 39  D     10  7
  0 14 39      DEBUT DE TRAVAIL DE   SAISIE
  0 14 42            FIN DE TRAVAIL DE   SAISIE
  0 14 58            FIN DE TRAVAIL DE   MESURE.FILE.COMMANDES
  0 15  0      DEBUT DE TRAVAIL DE   DECISION1
**********************************  0 15 13      APPEL TELEPHONIQUE     0 15 13  D     11  7
```

Exemple 8.1 - Trace

Exemple 8.1 - Trace



Exemple 8.1 - Trace

```
 1 20  0          FIN DE TRAVAIL DE   MESURE.FILE.COURRIER -
 1 20  0        FIN DE TRAVAIL DE   MESURE.FILE.ORDRES
 1 20  0          FIN DE TRAVAIL DE   MESURE.FILE.BONS1
 1 20  0          FIN DE TRAVAIL DE   MESURE.FILE.BONS2
 1 21  0      DEBUT DE TRAVAIL DE   VIDEUR1 -
 1 21  0          FIN DE TRAVAIL DE   VIDEUR1
 1 21  0      DEBUT DE TRAVAIL DE   VIDEUR2
 1 21  0        FIN DE TRAVAIL DE   VIDEUR2
 2  8 11      DEBUT DE TRAVAIL DE   SAISIE      APPEL TELEPHONIQUE - 2  8 11 D-  39  8
 2  8 13        FIN DE TRAVAIL DE   SAISIE
 2  8 41      DEBUT DE TRAVAIL DE   SAISIE      APPEL TELEPHONIQUE - 2  8 41 D-  40 10
 2  8 43        FIN DE TRAVAIL DE   SAISIE
```

Exemple 8.1 - Trace

Chapitre 9 - EVALUATION QUANTITATIVE du COMPORTEMENT DYNAMIQUE d'un SYSTEME D'INFORMATION

"*Figaro :*

Cinque... dieci... venti... trenta...
Trentasei... quarantatre "

(*W. A. Mozart et L. Da Ponte, Les Noces de Figaro, acte 1, scène 1*).

"*Figaro (prenant des mesures) :*

Cinq... dix... vingt... trente...
Trente-six... quarante-trois "

Chapitre 9 - EVALUATION QUANTITATIVE du COMPORTEMENT DYNAMIQUE d'un SYSTEME d'INFORMATION

1. QUESTIONS sur le COMPORTEMENT DYNAMIQUE.

2. METHODOLOGIE de la SIMULATION du COMPORTEMENT d'un SYSTEME d'INFORMATION.

2.1 Formulation du problème.

2.2 Etablissement d'un modèle.

2.2.1 Clôture d'un modèle de simulation.

2.2.2 Paramétrage du modèle.

2.2.3 Collecte des résultats.

2.3 Processus stochastiques et séries chronologiques.

2.3.1 Processus stochastiques.

a) Définitions.
b) Processus (strictement) stationnaires.
c) Processus quasi-stationnaires.
d) Approche de BOX et JENKINS.

2.3.2 Séries chronologiques ; estimation des moments.

a) Processus quasi-stationnaires.
b) Processus non stationnaires.

2.3.3 Intervalle de confiance.

a) Estimation de $Var\hat{\mu}$ par la fonction d'auto-covariance.
b) Estimation de $Var\hat{\mu}$ par la méthode des blocs.
c) La méthode des points de régénération.

2.3.4 Identification et estimation de processus.

a) Echantillon de variables indépendantes.
b) Echantillon de variables corellées.

2.4 Analyse et génération des entrées.

2.4.1 Paramètres d'entrée.

2.4.2 Mesures des entrées.

2.4.3 Génération d'échantillons aléatoires.

2.5 Mesures et analyse des sorties.

2.5.1 Résultats à obtenir.

a) Volume de données.
b) Occupation de ressources.
c) Temps de réponse.

2.5.2 Mesures et analyses statistiques.

a) Volume de données.
b) Occupation de ressources.
c) Temps de réponse.

2.6 Validation et robustesse d'un modèle de simulation de système d'information.

2.6.1 Validation par rapport à la réalité ou à un autre modèle grâce à des échantillons de mesures.

a) Cas quasi-stationnaire.
b) Cas non quasi-stationnaire.

2.6.2 Validation par rapport à un modèle mathématique.

a) Résolution exacte des files d'attente.
b) Résolution exacte de réseaux de files d'attente.
c) Résolution approchée des files ou réseaux de files d'attente.
d) Application à la validation des modèles de systèmes d'information.

2.6.3 Robustesse d'un modèle.

2.7 Plan d'expérimentation.

2.7.1 Régime permanent.

2.7.2 Régime transitoire.

2.7.3 Autres problèmes.

a) Conditions initiales.
b) Conditions finales ; arrêt de la simulation.

3. OUTILS de MESURE et d'ANALYSE STATISTIQUE dans le SYSTEME MAESTRO.

3.1 Caractéristiques techniques générales.

3.2 Description des objets spécifiques aux mesures.

3.2.1 Les transactions.

3.2.2 Les files.

3.2.3 Les ressources.

3.2.4 Les régions.

3.3 Processus de prise de mesure.

3.4 Exécution d'une simulation.

3.5 Procédures pour obtenir les résultats détaillés.

3.6 Obtention des résultats.

On veut obtenir ici des renseignements quantitatifs sur l'évolution d'un système au cours du temps.

A nouveau, nous posons certaines questions (1) pour lesquelles la fabrication et l'utilisation systématiques de maquettes apportent des réponses plus ou moins complètes.

La méthode utilisée est la simulation à événements discrets. Nous rappelons en quoi elle consiste, le nombre considérable de difficultés qui lui sont liées, et nous dressons un panorama des concepts et méthodes statistiques indispensables à sa bonne utilisation (2).

Nous présentons enfin les méthodes et outils de collecte de mesures et d'analyse statistique que nous utilisons (3).

1 - QUESTIONS sur le COMPORTEMENT DYNAMIQUE

Dans les ouvrages d'analyse en informatique de gestion, ce sujet est abordé souvent dans les chapitres traitant d'"analyse de l'existant", parfois dans ceux parlant de "conception générale". L'approche en est généralement assez prudente, parce que c'est un domaine délicat dans lequel on est encore mal armé.

Certainement, des progrès doivent passer par une meilleure caractérisation des problèmes posés, par une définition plus précise des concepts en jeu, et par le développement de méthodes et d'outils mieux appropriés. Notre démarche veut aller dans ce sens.

On peut regrouper les préoccupations quantitatives sur le comportement dans le temps d'un système d'information autour de trois thèmes principaux :

(1) volume de données traitées ;

(2) charge des composants ;

(3) temps de réponse de certaines parties.

Pour chaque type de questions, nous tentons d'apporter des réponses, toutes fondées sur la simulation à événements discrets, dont nous rappelons et appliquons les principes méthodologiques aux systèmes d'information.

2 - METHODOLOGIE de la SIMULATION du COMPORTEMENT d'un SYSTEME d'INFORMATION

Comme toute étude de simulation, l'évaluation par une maquette du comportement d'un système d'information doit suivre un canevas précis (NAY 66) que nous rappelons (figure 9.1).

Chaque étape de ce schéma pose quantité de problèmes que nous évoquons dans les paragraphes qui suivent.

Le travail présenté ici n'est pas de nature statistique. Aussi, quand il en est question, nous nous contentons le plus souvent de rappeler les définitions et résultats fondamentaux. Nous renvoyons à des ouvrages généraux pour les résultats classiques, et à des thèses ou articles spécialisés pour les résultats récents.

Nous avons fait l'hypothèse ici que nous choisissions la simulation comme moyen de résolution. C'est lui qui nous semble le plus approprié, compte-tenu des contraintes sévères imposées généralement à nos modèles. Ce n'est pas nécessairement le plus simple, et dans certains cas très particuliers, d'autres méthodes exactes ou approchées, que nous évoquons, peuvent être envisagées.

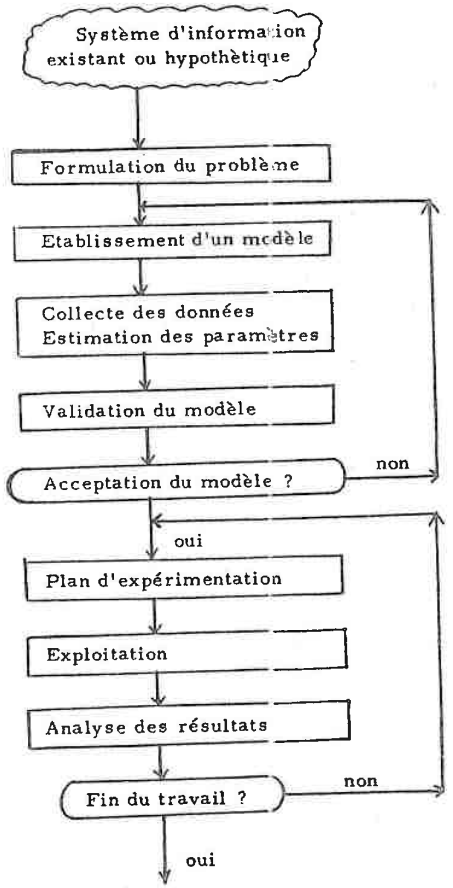


Figure 9.1 - Schéma d'une étude de simulation

2.1 Formulation du problème

Les objectifs d'une étude doivent être clairement définis. Nous en avons donné trois grands groupes parmi lesquels il importe de choisir celui ou ceux qui doivent être retenus.

Ceci est bien sûr une position de principe : la réalité est tout autre. C'est le plus souvent un processus itératif qui fait que, entre le début et la fin d'une étude, le problème a pu être reformulé plusieurs fois, par approximations successives. C'est d'ailleurs ce que l'on observe le plus couramment dans les recherches en sciences économiques ou sociales auxquelles s'apparente notre travail.

Ainsi, l'idéal du modèle unique, sur lequel on pourrait conduire toutes les études, est un leurre qu'il faut combattre vigoureusement.

2.2 Etablissement d'un modèle

Nous proposons bien entendu de bâtir une maquette de système d'information avec des composants pris parmi ceux que nous avons présentés.

Quand un programme-maquette a été écrit, il peut, moyennant quelques complèments, constituer un modèle de simulation programmé, compilable et exécutable sur une machine mono-processeur, selon une technique de quasi-parallélisme.

Nous développons ici les éléments qu'il est nécessaire d'adjoindre à une maquette.

2.2.1 Clôture d'un modèle de simulation

Souvent, une maquette de système d'information est un modèle ouvert, en ce sens qu'il a des relations avec ce qui n'a pas été écrit dans la maquette, et qu'on appelle extérieur ou environnement (chapitre 1).

Pour faire de la simulation, il est nécessaire de clore le modèle. Cette clôture peut être faite par adjonction de processus, et éventuellement de structures de données et de ressources, qui représentent tout ce qui, à l'extérieur, interagit avec la maquette (figure 9.2).

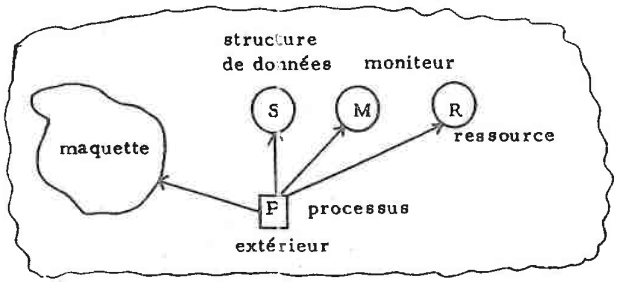


Figure 9.2 - Modèle de simulation clos

Exemple 9.1

Ainsi, dans l'exemple du chapitre 5, la maquette peut être close par l'adjonction de 4 processus.

Les processus cycliques générateur et génétel alimentent en messages les structures courrier et boîte. Un cycle du premier est déclenché par l'échéancier éch, à 9 heures tous les jours, grâce au moniteur d'activation activation5, alors qu'un cycle du deuxième est déclenché périodiquement de façon aléatoire (figure 9.3). Ils modèlisent respectivement l'arrivée du courrier et des appels téléphoniques.

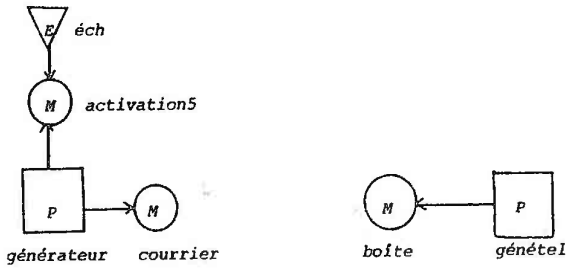


Figure 9.3 - Alimentation de la maquette du chapitre 5

Les processus cycliques videur1 et videur2 ont pour rôle de vider les files bons1 et bons2. Leurs cycles, qui vident ici instantanément les 2 files, sont déclenchés par l'échéancier éch, à 21 heures tous les jours, grâce aux moniteurs d'activation activation3 et activation4 (figure 9.4). Ils modèlisent la prise des bons de livraison avant 8 heures tous les jours pour le courrier et la comptabilité.

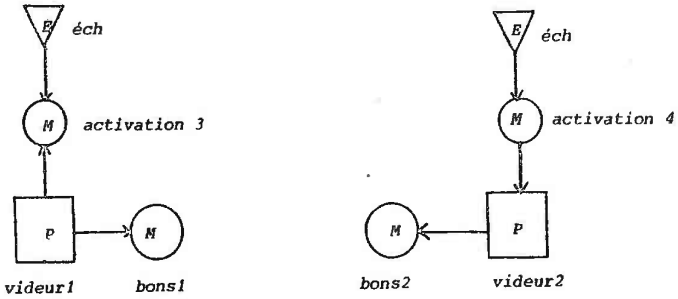


Figure 9.4 - Vidage de la maquette du chapitre 5

2.2.2 Paramétrage du modèle

En fonction des lois estimées pour les entrées, les transactions ou les temps de service (voir (2.4)), le modèle doit être complété.

Selon le niveau de paramétrage de la maquette close, il suffit de fixer des paramètres effectifs de générateurs ou d'appels de procédures, ou de compléter certaines instructions en suspens, telles que les instructions d'attente hold (...) ;

A ce propos, on peut signaler un inconvénient de SIMULA 67 qui complique un peu les choses : si le passage en paramètre d'une (fonction-) procédure est permis pour une (fonction-) procédure, il n'est pas permis pour une classe. Ainsi, deux objets identiques, mis à part un appel de procédure, ne peuvent pas être décrits par la même classe.

Exemple 9.2

Dans les processus, on rencontre souvent des suspensions pendant un temps dépendant d'une certaine loi. Si ce temps suit une loi Exponentielle négative pour l'un, et Normale pour l'autre, on doit écrire respectivement des instructions de la forme :

hold (negexp (a, u)) et hold (normal (a, b, u))

sans qu'il soit possible de paramétrer une classe de processus par une fonction-procédure à résultat réel.

□

2.2.3 Collecte des résultats

Pour obtenir de la simulation certains résultats statistiques, il est nécessaire de paramétrer convenablement le programme de simulation et, éventuellement, d'adjoindre quelques appels de procédures spécialisées.

Exemple 9.3

Nous avons déjà rencontré au chapitre 6 certaines classes qui sont paramétrées en fonction des résultats que l'on désire obtenir : c'est le cas notamment de la classe tfile, où deux paramètres booléens permettent

de préciser si l'on veut, ou non, des résultats globaux et détaillés sur les files. De telles classes contiennent de nombreux outils pour collecter des statistiques avec des appels à des procédures particulières.

Nous examinons en détail cette question en (3).

□

Avant de poursuivre avec les autres étapes de la simulation, il est nécessaire de faire ici un rappel sur les processus stochastiques et sur les séries chronologiques.

2.3 Processus stochastiques et séries chronologiques

Nous rappelons ici des définitions et résultats indispensables à la bonne compréhension du reste de l'exposé, concernant les processus stochastiques, les séries chronologiques et l'estimation de certaines caractéristiques. Cette présentation s'inspire de (BOX 76), (AND 71), (FIS 73), (AND 77), (FIC 76) et (LER 77).

2.3.1 Processus stochastiques

a) Définitions

On appelle processus stochastique une famille de variables aléatoires $\{X(t) ; t \in T\}$, où T est un ensemble d'index.

Dans la suite, T représente l'échelle de temps et est :

soit un intervalle de $\mathbb{R}$: le processus est alors dit continu,

soit une section de $\mathbb{N}$: le processus est alors dit discret.

Un processus est parfaitement défini si, pour tout n-uple d'instants $(t_1, \ldots, t_n) \in T^n$, la fonction de répartition conjointe suivante est connue :

$$F(t_1, \ldots, t_n ; x_1, \ldots, x_n) = \text{Prob}(X(t_1) < x_1, \ldots, X(t_n) < x_n)$$

Pratiquement, on se contente de la fonction de répartition absolue :

$$F(x ; t) = \text{Prob}(X(t) < x)$$

Comme pour des variables aléatoires simples, on peut définir des moments, qui dépendent en général du temps :

moyenne : $E(X(t)) = \mu(t)$

variance : $\text{Var}(X(t)) = E(X(t) - \mu(t)^2)$

écart-type : $\sigma(t) = \text{Var}(X(t))^{1/2}$

auto-covariance : $\gamma(t, s) = E((X(t) - \mu(t))(X(s) - \mu(s)))$

auto-corrélation : $\rho(t, s) = \gamma(t, s) / (\sigma(t) \times \sigma(s))$.

b) Processus (strictement) stationnaires

Ce sont des processus tels que, pour tout h et tout $(t_1, \ldots, t_n) \in T^n$ pour lesquels les fonctions de répartition conjointes sont définies :

$$F(x_1, \ldots, x_n ; t_1, \ldots, t_n) = F(x_1, \ldots, x_n ; t_1 + h, \ldots, t_n + h).$$

On a notamment

$$F(x ; t) = F(x ; t + h)$$

Ce qui a pour conséquence : pour tous t, s :

$$\mu(t) = \mu = \text{constante}$$

$$\text{Var}(X(t)) = \sigma^2(t) = \sigma^2 = \gamma(o) = \text{constante}$$

$\gamma(t, t+s) = \gamma(s)$ = fonction d'autocovariance (ne dépend que de s)

$\rho(t, t+s) = \rho(s)$ = fonction d'autocorrélation (ne dépend que de s).

Ces deux fonctions sont évidemment paires.

Théorème de DOOB-BIRKHOV :

Si $\{X(t)\}$ est un processus strictement stationnaire de moyenne $\mu \in \mathbb{R}$, de variance $\sigma^2 \in \mathbb{R}$, et tel que $\gamma(s)$ tend vers 0 quand s tend vers l'infini, alors les limites presque sûres des deux intégrales stochastiques suivantes sont réelles et :

$$\lim_{T \to +\infty} \text{p. s.} \quad \frac{1}{T} \int_0^T X^n(t)\, dt = E(X^n(t))$$

$$\lim_{T \to +\infty} \text{p. s.} \quad \frac{1}{T} \int_0^T X(t)\, X(t+s)\, dt = \gamma(s) + \mu^2$$

Ce théorème montre qu'il est possible d'estimer les moments et la fonction d'autocovariance en considérant des moyennes temporelles.

c) Processus quasi-stationnaires (ou faiblement stationnaires, ou stationnaires du 2ème ordre)

Il n'est souvent pas possible d'affirmer qu'un processus est strictement stationnaire. On se contente alors de processus satisfaisant deux hypothèses.

Un processus est quasi-stationnaire s'il satisfait pour tous t, s :

(1) $E(X(t)) = \mu =$ constante

(2) $\gamma(t, t+s) = \gamma(s)$: fonction d'autocovariance (ne dépend que de s).

Ceci entraîne :

$$\text{Var}(X(t)) = \gamma(0) = \sigma^2 = \text{constante.}$$

Les hypothèses (1) et (2), souvent supposées en pratique, peuvent être testées (PRI 69).

Pour des processus quasi-stationnaires, il est possible de définir

deux fonctions g et f, respectivement appelées spectre et fonction de densité spectrale, par des transformations de FOURIER dont nous ne donnons pas ici les conditions de validité :

Processus continu :

$$g(\lambda) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \gamma(s)\, e^{-i\lambda s} ds \qquad \text{pour} \qquad -\pi \leq \lambda \leq +\pi$$

$$\gamma(s) = \int_{-\pi}^{+\pi} g(\lambda)\, e^{i\lambda s}\, d\lambda \qquad \text{pour} \qquad s \in \mathbb{R}$$

$$f(\lambda) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \rho(s)\, e^{-i\lambda s} ds \qquad \text{pour} \qquad -\pi \leq \lambda \leq +\pi$$

$$\rho(s) = \int_{-\pi}^{+\pi} f(\lambda)\, e^{i\lambda s}\, d\lambda \qquad \text{pour} \qquad s \in \mathbb{R}.$$

Processus discret :

$$g(\lambda) = \frac{1}{2\pi} \sum_{s=-\infty}^{+\infty} \gamma(s)\, e^{-i\lambda s} \qquad \text{pour} \qquad -\pi \leq \lambda \leq +\pi$$

$$\gamma(s) = \int_{-\pi}^{+\pi} g(\lambda)\, e^{i\lambda s}\, d\lambda \qquad \text{pour} \qquad s \in \mathbb{N}$$

$$f(\lambda) = \frac{1}{2\pi} \sum_{s=-\infty}^{+\infty} \rho(s)\, e^{-i\lambda s} \qquad \text{pour} \qquad -\pi \leq \lambda \leq +\pi$$

$$\rho(s) = \int_{-\pi}^{+\pi} f(\lambda)\, e^{i\lambda s}\, d\lambda \qquad \text{pour} \qquad s \in \mathbb{N}$$

Ces deux fonctions sont paires et liées par la relation

$$g(\lambda) = \gamma(0)\, f(\lambda)$$

Nous en verrons l'usage ultérieurement.

d) Approche de BOX et JENKINS (BOX 70, AND 71, AND 77)

L'analyse de séries chronologiques a déjà été largement utilisée en informatique, par exemple pour l'analyse d'événements du type : défaut de page, accès à une base de données, etc ...

On considère ici un certain type de modèles généraux qui servent à construire, identifier, ajuster et valider des modèles de séries : l'approche globale de BOX et JENKINS qui commence seulement à être employé (ABA 79). Mais son intérêt dans un domaine comme le nôtre est tellement grand qu'il nous paraît indispensable de le mentionner. La présentation suivante est inspirée de (AND 77).

- Modèles simples

On s'intéresse à des processus linéaires discrets X (t), ou X_t, pour $t \in \mathbb{N}$, de la forme

$$X_t = \varphi_1 X_{t-1} + \ldots + \varphi_p X_{t-p} + A_t + \theta_1 A_{t-1} + \ldots + \theta_q A_{t-q} \qquad (1)$$

où les coefficients $\varphi_1, \ldots, \varphi_p, \theta_1, \ldots, \theta_q$ sont réels, et A_t un processus discret de variables aléatoires indépendantes, gaussiennes et de même loi $\mathcal{N}(0, \sigma_A)$ (bruits blancs). On peut choisir p et q tels que φ_p et θ_q soient différents de 0.

On introduit l'opérateur de décalage-arrière B, avec pour tout processus discret Z_t et toute date t :

$$B Z_t = Z_{t-1}$$

Alors (1) peut s'écrire :

$$(1 - \sum_{j=1}^{p} \varphi_j B^j) X_t = (1 + \sum_{j=1}^{q} \theta_j B^j) A_t$$

ou, en introduisant 2 polynômes notés φ_p et θ_q :

$$\varphi_p (B) \quad X_t = \theta_q (B) A_t \qquad (2)$$

Quand q = 0, (2) est l'équation d'un processus Auto-Régressif d'ordre p, ou AR (p).

Quand p = 0, c'est celle d'un processus à moyenne-mobile ("Moving-Average") d'ordre q, ou MA (q).

Pour des p et q différents de 0, c'est un processus ARMA (p, q). On démontre qu'un tel processus est stationnaire si le polynôme $\varphi_p (z)$ de la variable complexe z a tous ses zéros en dehors du disque unité. Il est dit inversible si la même condition est vérifiée pour $\theta_q (z)$.

- Modèles intégrés

Certains processus non stationnaires présentent une certaine homogénéité dont on peut tenir compte par une simple modification du modèle ARMA, le modèle Auto-Régressif à moyenne mobile Intégré. Ce modèle ARIMA (p, d, q) s'écrit :

$$\Phi (B) X_t = \theta_q (B) A_t$$

où $\Phi(B)$ est un polynôme de degré p + d, avec 1 comme zéro de multiplicité d, et tous les autres en dehors du disque unité :

$$\Phi (B) = \varphi_p (B) (1 - B)^d = \varphi_p (B) \nabla^d$$

où φ_p est un opérateur stationnaire autorégressif d'ordre p et ∇ l'opérateur effectuant la différence.

Si l'on remplace $\nabla^d X_t$ par Y_t, le processus ARIMA (p, d, q) X_t est réduit à un processus ARMA (p, q) Y_t.

On retrouve ici la notion de tendance (ou "trend") dans les séries chronologiques, approchée par des polynômes.

- Modèles saisonniers

Ces modèles complètent les précédents pour tenir compte de phénomènes saisonniers.

Ainsi, pour une période T, on est conduit à introduire l'opérateur

auto-régressif saisonnier :

$$\Phi_P(B^T) \equiv 1 - \Phi_1 B^T - \ldots - \Phi_P B^{TP}$$

l'opérateur de moyenne mobile saisonnier :

$$\Theta_Q(B^T) \equiv 1 + \Theta_1 B^T + \ldots + \Theta_Q B^{TQ}$$

et l'opérateur de différence saisonnier :

$$\nabla_T \equiv 1 - B^T$$

pour tenir compte des tendances entre les saisons. Un modèle de la forme :

$$\Phi_p(B^T)\ \nabla_T^D\ Y_t = \Theta_Q(B^T)\ A_t$$

est appelé modèle SARIMA d'ordre $(P, D, Q)_T$.

- Modèles généraux

Ainsi, un modèle général de BOX - JENKINS peut être écrit comme :

$$\varphi_p(B)\ \Phi_P(B^T)\ \nabla^d\ \nabla_T^D\ Y_t = \theta_q(B)\ \Theta_Q(B^T)\ A_t$$

qui est un modèle multiplicatif $(p, d, q) \times (P, D, Q)_T$.

2.3.2 Séries chronologiques ; estimation des moments

On appelle série chronologique (ou temporelle) une réalisation $x(t_1), \ldots, x(t_n)$ d'un processus stochastique observé aux instants $t_1, \ldots, t_n$ tels que $t_1 < \ldots < t_n$.

On peut disposer d'un ou plusieurs échantillons de ce type à partir desquels on cherche à inférer des propriétés du processus stochastique sous-jacent.

Par la suite, nous considérons des échantillonnages réguliers $t_i - t_{i-1} =$ constante.

nous notons x_i l'observation $x(t_i)$, et X_i la variable aléatoire $X(t_i)$ dont elle est issue.

Nous examinons successivement l'estimation des moments pour des processus quasi-stationnaires et non stationnaires.

a) Processus quasi-stationnaires

Dans le cas d'un processus strictement stationnaire, le théorème de DOOB-BIRKHOV montre qu'il est possible d'estimer les moments du processus à partir d'un seul échantillon.

On a les estimateurs non biaisés(*) suivants (notés avec un $\hat{}$) :

moyenne : $$\widehat{E(X(t))} = \hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

variance : $$\widehat{Var(X(t))} = \hat{\sigma}^2 = \frac{1}{n-1} \left(\sum_{i=1}^{n} X_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} X_i \right)^2 \right)$$

De plus, on a l'estimateur suivant pour l'auto-covariance d'ordre k :

$$\hat{\gamma}(k) = \frac{1}{n-k} \left(\sum_{i=1}^{n-k} X_i X_{i+k} - \frac{1}{n-k} \sum_{i=1}^{n-k} X_i \sum_{i=1}^{n-k} X_{i+k} \right)$$

On doit se limiter à des valeurs de k très inférieures à n, par exemple à n/4 (LER 77).

On en déduit un estimateur du coefficient d'auto-corrélation d'ordre k :

$$\hat{\rho}(k) = \frac{\hat{\gamma}(k)}{\hat{\gamma}(o)}$$

et, à partir de la formule donnant la fonction de densité spectracle dans le cas discret :

$$f(\lambda) = \frac{1}{2\pi} \sum_{s=-\infty}^{+\infty} \rho(s)\, e^{-i\lambda s} = \frac{1}{2\pi} \left(\rho(o) + 2 \sum_{s=1}^{\infty} \rho(s) \cos \lambda s \right)$$

(*) Si a est un paramètre à estimer, un estimateur non biaisé de a est une variable aléatoire, notée $\hat{a}$, telle que $E(\hat{a}) = a$.

on a un estimateur lissé (BOX 70) de la fonction de densité spectrale :

$$\hat{f}(\lambda) = \frac{1}{2\pi} \left(\hat{\rho}(o) + 2 \sum_{s=1}^{n-1} (1 - \frac{s}{n}) \hat{\rho}(k) \cos \lambda k \right)$$

Quand le processus est quasi-stationnaire, on utilise les mêmes estimateurs que ci-dessus.

b) Processus non stationnaires

Une méthode consiste à tenter de se ramener au cas précédent en décomposant le processus :

- soit sur des intervalles de temps où il peut être considéré comme (quasi-) stationnaire,

- soit en plusieurs composantes dont l'une est (quasi-) stationnaire. Sinon, on doit considérer une série de p réalisations du processus qui permettent d'utiliser les statistiques classiques pour une variable aléatoire à temps fixé.

2.3.3 Intervalles de confiance

Nous nous limitons ici au cas le plus courant de détermination d'intervalles de confiance pour la moyenne de processus quasi-stationnaire.

On a vu que $\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} X_i$ était un estimateur non biaisé de μ : $E(\hat{\mu}) = \mu$.

L'inégalité de BIENAYME-TCHEBYCHEFF fournit un intervalle de confiance dans le cas général :

$$\text{Prob}\left(|\hat{\mu} - \mu| < \alpha \sqrt{\text{Var}\,\hat{\mu}} \right) \geq 1 - \frac{1}{\alpha^2}$$

Si l'on peut considérer $\hat{\mu}$ comme suivant une loi Normale, on obtient un intervalle plus restreint par :

$$\text{Prob}\left(|\hat{\mu} - \mu| < \alpha \sqrt{\text{Var}\,\hat{\mu}} \right) = \frac{1}{\sqrt{2\pi}} \int_{-\alpha}^{+\alpha} \exp(-\frac{t^2}{2})\, dt = 2\,\Phi(\alpha) - 1$$

et, si au lieu de $\text{Var}\,\hat{\mu}$, on en a une estimation $\widehat{\text{Var}}\,\hat{\mu}$, alors $\frac{\hat{\mu} - \mu}{\widehat{\text{Var}}\,\hat{\mu}}$ suit une loi de STUDENT à p-1 degrés de liberté, p étant la taille de l'échantillon retenu.

Dans les deux cas, on est conduit à avoir un estimateur de la variance $\text{Var}\,\hat{\mu}$.

Nous nous limitons au cas où il n'y a pas qu'une seule réalisation du processus et examinons trois méthodes. Une autre utilisant l'analyse spectrale est donnée dans (FIS 73).

a) Estimation de $\text{Var}\hat{\mu}$ à partir de la fonction d'auto-covariance

Après quelques calculs, on obtient :

$$\text{Var}\hat{\mu} = E\left(\frac{1}{n} \sum_{i=1}^{n} (X_i - \mu) \right)^2 = \frac{1}{n^2} \sum_{i,j=1}^{n} \gamma(i-j)$$

d'où

$$\text{Var}\,\hat{\mu} = \frac{1}{n} \left(\gamma(o) + 2 \sum_{i=1}^{n} (1 - \frac{i}{n})\, \gamma(i) \right)$$

On obtient alors un estimateur de $\text{Var}\,\hat{\mu}$ à partir de ceux des coefficients d'auto-corrélation :

$$\widehat{\text{Var}}\hat{\mu} = \frac{1}{n} \left(\hat{\gamma}(o) + 2 \sum_{i=1}^{n} (1 - \frac{i}{n})\, \hat{\gamma}(i) \right)$$

Puisqu'on n'a pas une bonne estimation des $\gamma(i)$ pour les i grands, cette méthode n'est utilisée que si ces $\gamma(i)$ sont proches de 0. Des compléments à cette approche utilisant l'analyse spectrale sont donnés dans (FIS 73).

b) Estimation de $\text{Var}\,\hat{\mu}$ par la méthode des blocs

Cette méthode donnée par CONWAY (FIC 76, LER 77) permet de se débarasser de l'auto-covariance dans l'estimation.

L'ensemble des variables $X_1, \ldots, X_n$ est scindé en p blocs d'égale longueur ℓ (lorsque le nombre de mesures n'est pas un multiple de ℓ, on écarte des mesures centrales) :

$$\underbrace{|X_1 \ \ldots \ X_\ell|}_{\text{bloc 1}} \ldots \underbrace{|X_{(j-1)\ell+1} \ \ldots \ X_{(j-1)\ell+i} \ldots X_{j\ell}|}_{\text{bloc j}} \cdots \underbrace{|X_{(p-1)\ell+1} \ \ldots \ X_{p\ell}|}_{\text{bloc p}}$$

On calcule pour chaque bloc la moyenne empirique :

$$\overline{X}_j = \frac{1}{\ell} \sum_{i=1}^{\ell} X_{(j-1)\,\ell + i} \quad \text{de moyenne} \quad E(\overline{X}_j) = \mu$$

On fait alors l'hypothèse que les $\overline{X}_j$ sont des variables <u>indépendantes et de même loi</u>. Si cette hypothèse semble trop forte, on peut par exemple ne retenir qu'un bloc sur deux. Alors, la statistique classique indique que l'on peut estimer Var $\overline{X}_j$ par l'estimateur

$$\widehat{\text{Var}\ \overline{X}_j} = \frac{1}{p-1} \left(\sum_{j=1}^{p} \overline{X}_j^2 - \frac{1}{p} \left(\sum_{j=1}^{p} \overline{X}_j \right)^2 \right)$$

et, puisque $\hat{\mu} = \frac{1}{p} \sum_{j=1}^{p} \overline{X}_j$, on peut <u>estimer Var $\hat{\mu}$</u> par :

$$\widehat{\text{Var}\ \hat{\mu}} = \frac{1}{p\,(p-1)} \left(\sum_{j=1}^{p} \overline{X}_j^2 - \frac{1}{p} \left(\sum_{j=1}^{p} \overline{X}_j \right)^2 \right)$$

La longueur minimale ℓ_o d'un bloc pour assurer l'indépendance est obtenue par le calcul successif des estimations des coefficients d'auto-corrélation pour avoir :

$$| \widehat{\rho(\ell_o)} | << 1$$

La loi suivie par $\frac{\hat{\mu} - \mu}{\widehat{\text{Var}\ \hat{\mu}}}$ est considérée comme une <u>loi de STUDENT</u> à p-1 degrés de liberté.

c) <u>La méthode des points de régénération</u> (cf. par exemple FIS 73, KOB 78).

Cette méthode élimine le problème de la détermination de périodes transitoires qui est sous-jacent aux deux précédentes techniques. Bien que nous ne l'ayons pas encore expérimentée, nous pensons que nos systèmes présentent des aspects favorables à son utilisation, comme le caractère répétitif de beaucoup de procédures.

Un processus stochastique est dit <u>régénératif</u> s'il existe un état particulier (état régénératif) tel que, quand le processus retourne à cet état, l'histoire passée n'a pas d'influence sur le comportement futur du processus.

Les dates auxquelles le système retourne à un tel état sont appelées <u>points de régénération</u>, et le temps entre les k^e et $(k+1)^e$ points de régénération est appelé k^e cycle. Alors, pour un processus régénératif $\{ X(t) ; t \geq 0 \}$, la continuation du processus au-delà d'un certain point de régénération t_1 est une réplique probabiliste du processus commençant en 0, à condition que 0 soit un point de régénération.

Si le temps entre deux points de régénération successifs est fini, on peut penser que les observations faites dans un cycle sont indépendantes des observations faites dans les autres cycles. De plus, les échantillons relevés pendant de tels cycles sont identiquement distribués.

Soit X une variable aléatoire représentant X (t) à l'équilibre, c'est-à-dire, pour tout $x \in \mathbb{R}$:

$$\lim_{t \to \infty} \text{Prob}\,(X(t) \leq x) = \text{Prob}\,(X \leq x)$$

Nous cherchons à estimer $\mu = E(X)$.

Chaque cycle permet de générer un couple (Y_i, T_i) où Y_i est l'intégrale stochastique de X (t) sur le i^e cycle de durée T_i. Les couples

$(Y_i, T_i)_{i = 1, \ldots, n}$ sont supposés indépendants et identiquement distribués. On en déduit, d'après la loi forte des grands nombres :

$$\mu = \lim_{t \to \infty} \frac{1}{t} \int_0^t X(u)\, du = \frac{E(Y_i)}{E(T_i)}$$

Si l'on pose

$$\overline{Y} = \frac{1}{n} \sum_{i=1}^{n} Y_i \quad \text{et} \quad \overline{T} = \frac{1}{n} \sum_{i=1}^{n} T_i$$

on est amené à prendre comme estimateur de μ :

$$\hat{\mu} = \frac{\overline{Y}}{\overline{T}}$$

Le théorème de la limite centrale permet d'affirmer pour n grand :

$$\text{Prob}\left(\left| \overline{Y} - \mu \overline{T} - E(\overline{Y} - \mu \overline{T}) \right| < \frac{t\sigma}{\sqrt{n}} \right) = 2\,\Phi(t) - 1$$

où σ est l'écart-type de $Y_i - \mu T_i$. D'où l'intervalle de confiance donné par :

$$\text{Prob}\left(|\mu - \hat{\mu}| < \frac{t\sigma}{\overline{T} \times \sqrt{n}} \right) = 2\,\Phi(t) - 1$$

On peut remplacer σ par une estimation calculée grâce à l'estimateur de $\text{Var}(Y_i - \mu T_i)$:

$$\widehat{\text{Var}}(Y_i - \mu T_i) = \widehat{\text{Var}}(Y_i) - 2\hat{\mu}\,\widehat{\text{Cov}}(Y_i, T_i) + \hat{\mu}^2\,\widehat{\text{Var}}(T_i)$$

où

$\widehat{\text{Var}}(Y_i)$, $\widehat{\text{Var}}(T_i)$ et $\widehat{\text{Cov}}(Y_i, T_i)$ sont obtenus par les estimateurs non biaisés classiques à partir des (Y_i, T_i).

2.3.4 Identification et estimation de processus

On cherche ici à obtenir les lois suivies par des processus dont on possède une seule réalisation sous la forme d'une série chronologique $x_1, \ldots, x_n$.

Deux cas sont à considérer selon que les variables aléatoires sous-jacentes $X_1, \ldots, X_n$ peuvent être considérées comme indépendantes ou non.

a) Echantillon de variables indépendantes

. Il existe un certain nombre de tests d'indépendance des variables d'un échantillon.

Ils sont souvent fondés sur les statistiques non paramétriques, comme le "gap test", le "poker test" ou le "run test" (KNU 68, FIS 73).

On observe parfois aussi la forme de la fonction d'auto-corrélation estimée (voir (c)) qui doit, en cas d'indépendance, être nulle partout sauf en 0.

. Quand on peut considérer que les variables sont indépendantes et faire l'hypothèse qu'elles ont toutes la même loi de probabilité, on dispose alors de tous les outils statistiques classiques pour en faire l'estimation : méthode du maximum de vraisemblance, tests de KOLMOGOROFF, du Chi-deux, ...

b) Echantillon de variables corellées

Les choses sont ici singulièrement plus compliquées.

Dans le cas général, il s'agit de décomposer un processus en tendance, variations saisonnières et composante stationnaire. Nous avons montré que ceci correspondait aux objectifs des modèles de BOX et JENKINS. Les méthodes utilisées dans ce cadre sont affaire de spécialistes : elles sont d'ailleurs aujourd'hui l'objet d'un effort de recherche très important. Nous nous contentons de tracer l'esquisse de la méthode d'analyse préconisée, ou "cycle" de BOX et JENKINS parce que c'est une procédure itérative.

- Visualisation de la série et identification du modèle

La visualisation est importante pour mettre en évidence l'existence éventuelle de tendances ou composantes cycliques. Pour cela, on peut tracer :

. le graphe de la série, qui indique s'il est plausible d'accepter la stationnarité ;

. le corrélogramme : graphe de la fonction d'auto-corrélation, qui permet d'étudier l'intensité et les périodes des liaisons entre les variables ;

. le spectrogramme : graphe de la fonction de densité spectrale f estimée ; en général, on n'étudie pas toute la bande de fréquences, mais seulement celles des périodicités principales $\lambda_p = \frac{2\pi p}{n}$, $p = 1, \ldots, E(\frac{n-1}{2})$;

. des périodogrammes, de SHUSTER (BOX 70) ou de WHITTAKER et ROBINSON (AND 71), dans le même but ;

. des graphes de la série obtenus après filtrage de tendance et saisonnalité, pour voir s'il est possible d'accepter l'hypothèse de stationnarité.

L'identification d'un processus stationnaire ARMA est conduite par l'étude des propriétés des coefficients d'auto-corrélation et de leurs dérivées, avec un grand souci de "parcimonie", c'est-à-dire de minimisation du nombre des coefficients.

- Estimation

Il s'agit d'estimer les valeurs des coefficients du modèle. A cet effet, pour des processus stationnaires, on utilise une procédure d'approximation au sens des moindres carrés non linéaire pour minimiser :

$$S(\varphi_1, \ldots, \varphi_p, \theta_1, \ldots \theta_p) = \sum_{t=1}^{n} a_t^2$$

où $a_t = \theta^{-1}(B)\ \varphi(B)\ y_t$

Le résultat obtenu peut être modifié pour tenir compte de certaines particularités.

- Vérification

La dernière étape consiste à soumettre le modèle à des tests d'adéquation, qui sont utilisés lors de déviations d'ajustement, et qui mettent en évidence d'autres discordances.

On teste notamment que la série résiduelle :

$y_t - \hat{y}_t = \hat{a}_t$, où $\hat{y}_t$ désigne l'estimation obtenue, est un bruit blanc.

2.4 Analyse et génération des entrées

2.4.1 Paramètres d'entrée

Il est nécessaire d'estimer :

- certains paramètres du modèle : lois des transitions pour les chemins suivis par certaines transactions, lois des transformations pour d'autres, par exemple pour tenir compte de certains phénomènes de regroupement ou d'éclatement de données.

Exemple 9.4

Dans la maquette du chapitre 5, nous devons fixer la loi des livraisons en fonction des commandes. Le modèle simple qui a été retenu consiste à déterminer aléatoirement, pour chaque commande, et à chaque déclenchement du processus décision, le nombre de lignes à livrer.

□

- Les lois d'arrivée des entrées : on détermine si les transactions qui entrent dans le système sont groupées ou bien individualisées et les lois de leurs interarrivées dans le système.

Exemple 9.5

Dans la maquette du chapitre 5, nous avons fixé une date d'arrivée du courrier-commandes : ceci n'est pas très important parce que nous considérons un début du traitement de ce courrier bien ultérieur. En revanche, les interarrivées des coups de téléphones pour les commandes téléphonées sont aléatoires, mais de même loi.

□

- les caractéristiques de certaines données : quand la sémantique des données est relativement riche, il faut des lois de probabilité pour fixer les valeurs de leurs attributs.

Exemple 9.6

Dans la maquette du chapitre 5, nous avons fixé la probabilité pour qu'un coup de téléphone corresponde à une commande ou modification de commande téléphonée, le reste étant considéré comme uniquement des demandes de renseignements. Nous avons aussi donné la loi des nombres de lignes de commandes écrites et téléphonées.

□

- les temps de service : pour tous les traitements d'informations considérés comme élémentaires, il est nécessaire de déterminer des lois de durée des services.

Exemple 9.7

Dans la maquette du chapitre 5, nous avons fixé des durées pour tous les services, à des niveaux plus ou moins fins : temps de saisie d'une commande, temps d'élaboration des propositions automatiques de livraison, temps d'examen de ces propositions, etc...

□

2.4.2 Mesures des entrées

On peut fixer arbitrairement, par exemple en fonction de projections sur l'avenir, certains paramètres.

Il est néanmoins nécessaire d'en avoir d'autres qui correspondent à la réalité, et de les avoir tous, lorsqu'il s'agit de valider un modèle.

Cela conduit à faire de nombreuses mesures dans l'organisation. On doit souligner toute la difficulté d'obtenir des relevés significatifs dans un domaine aussi mouvant : si les statistiques sur les données et leurs flux sont relativement faciles à relever (à condition d'avoir des archives bien organisées), il n'en est pas de même d'éléments comme les temps de service. Pour ceux-ci se dressent des difficultés de tous ordres qui sont bien connues des spécialistes d'OST et qui s'exacerbent dans les bureaux : méfiance vis à vis du chronométrage, extrême variabilité des temps observés selon les personnes et les périodes, introduction d'un biais dû à l'observation elle-même, etc ..

Les procédés d'analyse statistique et d'estimation sur les séries chronologiques relevées peuvent être utilisés, mais il faut rester souple et admettre une marge d'erreur importante.

On remarque aussi, bien évidemment, que plus un modèle est détaillé, plus on a de mesures à faire et de lois à estimer : en plus de l'augmentation du travail à fournir, on risque aussi une accumulation d'erreurs faussant le modèle. Aussi, il est bon de reprendre ici la recommandation de "parcimonie" dans l'élaboration de modèles quantitatifs (BOX 76).

Exemple 9.8

Pour les paramètres de la maquette du chapitre 5, nous avons effectué un certain nombre de relevés et de mesures statistiques, pour l'obtention desquels nous avons rencontré les difficultés signalées ci-dessus. La

quantité d'information recueillie n'a pas été complètement analysée, et l'étude en est poursuivie actuellement par M. THAUREL pour sa thèse (THA 80).

Les paramètres que nous avons retenus pour obtenir les résultats que nous présentons en (2.5) ne sont que très approximatifs de la réalité. Cependant, il n'y a pas eu lieu de faire une "réduction quantitative" comme la possibilité en est signalée dans le chapitre 3.

Processus générateur

Un paquet de lettres de commande parvient chaque jour à 9 heures. Leur nombre est la partie entière d'une variable (pseudo-) aléatoire, générée selon une loi Normale $\mathcal{N}$ (5, 10) (de moyenne 5, d'écart-type 10), et tronquée, de manière à garder une valeur ≥ 0. Le nombre de lignes suit une loi analogue, basée sur $\mathcal{N}$(7, 10).

Processus génétel

Les coups de téléphone sont générés selon un processus de POISSON, c'est-à-dire que leurs interarrivées suivent une loi Exponentielle négative (KLE 75) Exp (15/7) (de moyenne 7/18 (d'heures)). Dans 8 cas sur 18, selon une loi Uniforme, ils donnent lieu à une commande effective, le reste étant des demandes de renseignements. Le nombre de lignes par commande est fondé sur une loi Normale $\mathcal{N}$(9,2), de manière identique à ci-dessus.

Processus enregistrement

La durée du codage des commandes écrites est calculée de manière déterministe en fonction du nombre de commandes et du nombre de lignes par commande. Il en est de même de l'enregistrement des commandes par saisie directe sur console de visualisation.

Processus saisie

La durée de la conversation est fixe dans le cas d'une demande de renseignement téléphonique et proportionnelle au nombre de lignes dans le cas d'une commande téléphonée. Dans ce dernier cas, une partie du temps est comptabilisée aussi par ord.

Processus décision

Il est déclenché tous les jours à 15 heures.

La durée d'obtention des propositions de livraison est fixée de manière déterministe à 15 mn.

Le nombre de lignes proposées par commande suit une loi Binômiale, dont les paramètres sont calculés de manière à maintenir constant en moyenne le nombre de lignes de commande en attente.

La durée de la prise de décision est déterministe et proportionnelle au nombre de commandes total.

La saisie des modifications des propositions a une durée déterministe proportionnelle au nombre total de lignes en attente. La durée de l'obtention des livraisons définitives est fixée à 15 mn.

Processus préparation

La durée de la préparation d'un envoi et de celle du remplissage des bons de livraison correspondants a été fixée selon une loi Normale tronquée dont les paramètres sont proportionnels au nombre de lignes figurant sur le bon de livraison de l'ordinateur.

Processus videur1 et videur2

Déclenchés tous les jours en fin de journée pour vider bons1 et bons2, ils sont considérés comme instantanés.

On trouve en annexe B le texte complet de ces processus avec le détail des paramètres utilisés.

□

2.4.3 Génération d'échantillons aléatoires

L'utilisation directe des échantillons relevés est une méthode possible pour alimenter un modèle, mais elle est très lourde.

On préfère générer des échantillons pseudo-aléatoires à partir des lois de probabilité que l'on a estimées.

Il n'y a pas de difficulté particulière dans nos modèles, que les processus soient stationnaires, ou qu'ils ne le soient pas, notamment dans le cas des processus de BOX et JENKINS.

L'essentiel est de possèder un bon générateur de nombres au hasard : nous renvoyons ici aux nombreuses études dans ce domaine, citées dans (KNU 68) et (LER 77).

Pour notre part, nous utilisons les procédures de la bibliothèque SIMULA 67 (CII 72), fondées sur l'utilisation d'un générateur congruentiel.

Pour chaque application particulière, il est possible de fabriquer ses propres générateurs d'échantillons pseudo-aléatoires.

Exemple 9.9

Dans le modèle du chapitre 5, plusieurs procédures de génération d'échantillons ont été écrites pour compléter celles de la bibliothèque de SIMULA.

La procédure

integer procedure normale (a, b, c) ; real a, b ; integer c ;...;

permet d'obtenir des échantillons parties entières de réels générés selon une loi Normale (a, b) (de moyenne a et d'écart-type b), tronquée à gauche, de manière que les nombres soient toujours ≥ c.

La procédure

real procedure réelnormale (a, b, c) ; real a, b, c ;...;

a le même rôle, mais pour des échantillons réels.

La procédure

integer procedure binômiale (nb, p) ; integer nb ; real p ;...;

permet de générer des échantillons distribués selon une loi Binômiale de paramètres nb et p.

On en trouvera les détails en annexe B.

□

2.5 Mesures et analyse des sorties

2.5.1 Résultats à obtenir

Nous nous intéressons à des résultats caractéristiques de l'évolution dans le temps des :

a) Volumes de données

Pour cela, on définit, pour certaines structures de données $\mathcal{S} = (\mathcal{J}, \mathcal{M})$, une fonction de valuation V à valeurs réelles ≥ 0 :

$V : \mathcal{J} \rightarrow \mathbb{R}_+$.

On suit alors l'évolution du processus $V_t = V(d_t)$, où $d_t \in \mathcal{J}$ est la donnée à l'instant t.

Exemple 9.10 - Files d'attente

Chaque élément d'une file f_t d'une structure de la classe tfile est une transaction affectée d'un attribut-poids x.pds (en SIMULA). On peut alors définir une fonction V telle que :

$$V(f_t) = \sum_{x \in f_t} x.pds$$

La valeur de cette fonction peut être recalculée sur demande, ou à chaque modification des données : c'est cette deuxième solution qui a été retenue.

□

b) Occupation de ressources

Pour chaque ressource à laquelle on s'intéresse, on étudie le processus stochastique :

N_t = nombre d'accès utilisés à l'instant t.

Une des difficultés majeures dans l'analyse de ce type de processus provient des calendriers, qui diminuent le temps pendant lequel la ressource est utilisable. Nous en verrons les conséquences sur les outils de mesure.

Exemple 9.11

Pour une ressource critique, le processus N_t a toujours comme valeurs 1 ou 0, selon que la ressource est utilisée ou pas.

Pour une ressource à n accès, $0 \leq N_t \leq n$, alors que N_t est illimité pour une ressource à nombre infini de points d'accès.

□

c) Temps de réponse

On étudie ici les temps qu'il faut pour exécuter certains traitements, que l'on ramène toujours aux durées nécessaires à des transactions pour effectuer certains parcours.

On est conduit à définir des régions dans lesquelles des transactions entrent et desquelles il en sort. Les transactions qui sortent peuvent être identiques à celles qui entrent, en même nombre ou pas (existence de "puits"), ou avoir été transformées, éventuellement par regroupement ou par éclatement de transactions entrantes.

L'essentiel pour le système est que toute transaction qui sort d'une région ait un numéro (numéro de la classe transaction) et une date d'entrée (interne à la classe transaction), qui permettent de la relier à une transaction entrante : à partir de la date actuelle, on détermine le temps de traversée de la région pour la transaction.

Ainsi, on est ramené à étudier les processus :

E_t = nombre de transactions entrées dans la région à l'instant t

S_t = nombre de transactions sorties de la région à l'instant t

et la série statistique :

T_n = temps passé dans la région par la n^e transaction sortante.

Exemple 9.12

Dans la maquette du chapitre 5, plusieurs régions peuvent être définies pour des mesures de temps de réponse. Par exemple :

- *la région écrite mesure la durée entre l'arrivée d'une lettre de commande et son enregistrement informatique ;*
- *la région administratif mesure le temps entre l'enregistrement d'une commande et l'émission du dernier bon de livraison qui la concerne (les commandes peuvent être livrées par fractions).*
- *la région physique mesure le temps entre l'émission d'un bon de livraison et la fin de la préparation de cette livraison.*

□

Cette notion de région est analogue à celle de "queue" de GPSS (IBM 70) et à celle de "région" de GPSSS (VAU 77).

2.5.2 Mesures et analyses statistiques

Pendant l'exécution d'une maquette en simulation, de nombreuses mesures peuvent être effectuées. Elles entrent immédiatement comme données dans des calculs ou, au contraire, elles sont enregistrées sous forme brute pour servir à des analysses statistiques ultérieures. Nos propos sont illustrés par des résultats obtenus par l'exécution de la maquette du chapitre 5.

a) Volumes de données

Pour suivre l'évolution du volume d'une structure de donnée, on fait des observations régulièrement espacées dans le temps, à

des instants $t_1, \ldots, t_n$ tels que

$$t_i - t_{i-1} = \underline{\text{constante}} \quad \text{pour} \quad i = 2, \ldots, n.$$

On obtient ainsi la série $v_1, \ldots, v_n$.

On en profite pour déterminer à chaque instant les volumes minimum, maximum et moyen (moyenne "temporelle").

Dans notre système, la série (v_i) peut être enregistrée intégralement dans un fichier. Elle contribue alors, après analyse statistique, à estimer l'équilibre ou l'évolution du système (tendance, variations saisonnières). Quand le processus stochastique sous-jacent aux (v_i) peut être considéré comme quasi-stationnaire, alors la moyenne temporelle obtenue est admise comme approchant la moyenne du processus, et l'on peut déterminer un intervalle de confiance, par exemple par la méthode des blocs (2.3.3).

Il y a cependant un danger à mesurer avec une période régulière un volume de données : c'est que la variation du volume soit également périodique et en phase avec le processus d'échantillonnage, et que l'on obtienne ainsi des résultats non significatifs.

C'est pourquoi nous utilisons un autre procédé, que nous n'avons encore mis en œuvre que pour les files d'attente : des relevés à chaque événement entraînant une variation de volume. On comptabilise alors les entrées, les sorties, les contenus maximum et minimum, et les retours à l'état vide des files d'attente.

Exemple 9.13

Nous présentons ci-après des résultats obtenus après exécution de la maquette du chapitre 5 sur 60 jours simulés, en régime stationnaire, après une période d'initialisation (2.7.3) de 5 jours. On constate une distorsion entre les résultats des figures 9.5 et 9.6 due à la différence du procédé de mesure signalé ci-dessus.

□

	MESURES		EVALUATION DES VOLUMES D INFORMATION : VOLUME			
FILE (S)	NOMBRE	INTERVALLE	MINIMAL	MAXIMAL	MOYEN	ACTUEL
COMMANDES	60	1 0 0	51	753	331	51
COURRIER	60	1 0 0	1	174	81	140
BONS!	60	1 0 0	1	263	149	252

Figure 9.5 - Récapitulatif volumes d'information

			CONTENU DE LA FILE			
FILE (S)	ENTREES	SORTIES	MAXIMUM	MOYEN	ACTUEL	RET A ZERO
COMMANDES	1114	1089	113	56.487	25	0
COURRIER	589	589	21	0.555	0	60
ORDRES	1530	1452	110	25.450	78	61
BONS!	1451	1451	68	7.484	0	60

Figure 9.6 - Récapitulatif files d'attente

b) Occupation des ressources

A chaque acquisition ou libération d'une ressource, ou d'un accès d'une ressource, il est possible d'enregistrer la date simulée courante, avec le nombre d'accès précédemment utilisés, et la durée de cette utilisation.

Ces mesures permettent d'apprécier l'évolution de la charge qui incombe aux ressources, par estimation des caractéristiques du processus N_t (nombre d'accès utilisés à l'instant t).

Ce processus peut être étudié après la simulation lors d'analyses statistiques complémentaires.

Si le système est dans un état d'équilibre, on obtient un taux d'utilisation τ, par la formule :

$$\tau = \lim_{T \to \infty} \frac{\int_0^T N_t \, dt}{\int_0^T D_t \, dt}$$

où D_t donne le nombre d'accès disponibles pour le système à l'instant t (y compris ceux qui sont utilisés).

L'appréciation de D_t est compliquée par l'usage des calendriers qui le réduisent temporairement à 0, pendant les périodes d'indisponibilité.

Comme nous l'avons déjà indiqué, il est indispensable, dans le cas (quasi-) stationnaire, de déterminer pour le taux d'utilisation un intervalle de confiance. Dans notre système, celui-ci est obtenu par la méthode des blocs (2.3.3).

Toujours dans les mêmes conditions, la présentation des résultats peut être complétée par des diagrammes récapitulant le nombre d'accès utilisés par périodes de temps en pourcentages.

Exemple 9.14

Nous présentons des résultats sur l'utilisation des ressources pour la maquette du chapitre 5, obtenus dans les conditions données précédemment.

Les figures 9.7, 9.8, et 9.9 synthétisent l'utilisation des différents types de ressources.

Les figures 9.10 et 9.11 donnent des exemples de diagrammes de répartition dans le temps de l'utilisation des accès.

La figure 9.12 présente trois intervalles de confiance à différents seuils pour la ressource z.

CALRC	PROCESSUS SERVIS	DUREE DISPONIBLE	TAUX UTILISATION	ETAT ACTUEL
X	1178	7 12 0	0.400	LIBRE
Z	60	3 18 0	0.186	LIBRE
P	1451	20 0 0	0.600	OCCUPE

Figure 9.7 - Récapitulatif de l'utilisation des ressources critiques avec calendriers

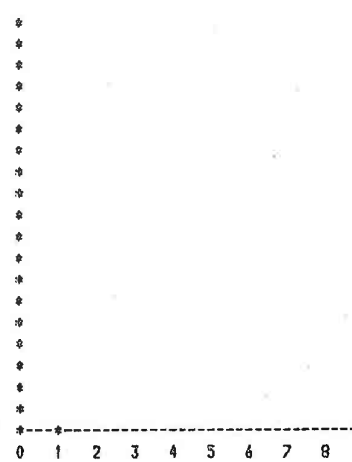
CALRCPP	PROCESSUS SERVIS	DUREE DISPONIBLE	TAUX UTILISATION	ETAT ACTUEL
Y	977	20 0 0	0.206	LIBRE

Figure 9.8 - Récapitulatif de l'utilisation des ressources critiques avec priorités, préemptibles et avec calendriers

RAMINF	NOMBRE D UTILISATEURS SERVIS	MOYEN	MAXI.	ACTUEL
ORDINATEUR	1725	0.04	2	0

Figure 9.9 - Récapitulatif de l'utilisation des ressources à nombre infini de points d'accès

DIAGRAMME D OCCUPATION DE LA RESSOURCE :ORDINATEUR
TYPE - RAMINF



NB ACCES	PROBA.UTILIS.
0	0.9670
1	0.0320
2	0.0010
3	0.0000
4	0.0000
5	0.0000
6	0.0000
7	0.0000
8	0.0000
9	0.0000
10	0.0000

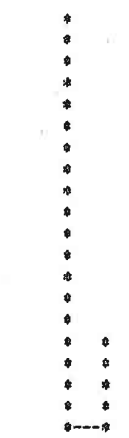
DUREE DE DISPONIBILITE : 60 0 0

DUREE DE RESERVATION : 2 3 7

REMARQUE : LA PROBABILITE DONNEE POUR 10 ACCES
EST LA PROBABILITE POUR 10 ACCES ET PLUS

Figure 9.10 - Diagramme d'utilisation des accès de la ressource ordinateur ord

```
DIAGRAMME D OCCUPATION DE LA RESSOURCE :Y
TYPE - CALRCPP    - NOMBRE D ACCES :  1
```




```
DUREE DE DISPONIBILITE :  20  0  0      DUREE DE RESERVATION :  4  3  0
```

NB ACCES	PROBA.UTILIS.
0	0.8070
1	0.1930

Figure 9.11 - Diagramme d'utilisation des accès de la ressource y (calrcpp)


```
*************************************
* RESSOURCE N° :  4  Z              *
*************************************

DUREE MINIMALE INTER-EVENEMENTS :     0.010      DUREE MINIMALE AUTORISEE     :     0.010

NOMBRE D INTERVALLES DE MESURE INFERIEUR A EPSI :       0

NOMBRE DE MESURES FAITES PENDANT LA SIMULATION :      181


RESSOURCE N° :   4    POUR RO =      0.0485    METHODE DES BLOCS  LO =    25

 NOMBRE DE BLOCS :  333
 TAILLE D UN BLOC:   27

INTERVALLE DE CONFIANCE DU TAUX D OCCUPATION MOYEN

        SEUIL ALPHA =   0.01      (  0.144  ,   0.229  )
        SEUIL ALPHA =   0.05      (  0.154  ,   0.219  )
        SEUIL ALPHA =   0.10      (  0.159  ,   0.214  )

 ESTIMATION DU TAUX D OCCUPATION MOYEN :  0.186

** PRECISION SUR LES TAUX D OCCUPATION **
```

RESSOURCE	TAUX MOYEN OCCUP	ECART SEUIL 1%	ECART SEUIL 5%	ECART SEUIL 10%
Z	0.1864	+0.0- 0.0025	+0.0- 0.0324	+0.0- 0.0272

Figure 9.12 - Intervalles de confiance aux seuils 1 %, 5 % 10 % pour le taux d'occupation moyen de la ressource z

□

c) Temps de réponse

Quand une transaction entre dans une région, on comptabilise une nouvelle entrée, tout en retenant la date courante (d'entrée).

Quand une transaction quitte une région, on comptabilise une sortie et on calcule le temps passé dans la région : date courante (de sortie) - date d'entrée.

Ceci permet d'obtenir les caractéristiques des processus E_t, S_t et la série des T_n.

Comme précédemment, ces mesures peuvent être enregistrées dans un fichier pendant la simulation, et analysées ultérieurement.

Dans le cas où le système est en équilibre, on détermine un temps de réponse moyen s par :

$$s = \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} T_i$$

où n est le nombre des transactions sorties de la région, et T_i le temps de réponse pour la i^e transaction sortie. L'étude doit être complétée par la détermination d'un intervalle de confiance. Il est obtenu dans notre système par la méthode des blocs (2.3.3).

La présentation des résultats est améliorée par la présentation d'histogrammes donnant la répartition des temps de réponse.

Exemple 9.15

Dans la maquette du chapitre 5, nous retenons deux régions , administratif et physique, permettant d'obtenir deux séries de caractéristiques sur les temps de réponse présentés à la figure 9.13.

Des intervalles de confiance pour les temps de réponse moyens sont donnés à la figure 9.14.

Enfin, un exemple d'histogramme sur la répartition des temps de réponse d'administratif est montré à la figure 9.15.

REGION (S)	ENTREES	SORTIES	CONTENU DE LA REGION			TEMPS DE SEJOUR		
			MAXIMUM	MOYEN	ACTUEL	MAXIMUM	MOYEN	MINIMUM
ADMINISTRATIF	1114	1089	113	56.490	25	8 5 49	3 5 4	0 1 25
PHYSIQUE	1530	1451	111	26.090	79	3 13 50	1 1 20	0 0 1

Figure 9.13 - Récapitulatif sur les temps de réponse des régions administratif et physique

```
REGION : ADMINISTRATIF    NO: 1    POUR RO =      0.0447    METHODE DES BLOCS LO =    13

NOMBRE DE BLOCS :   72
TAILLE D UN BLOC:   15

INTERVALLE  DE  CONFIANCE  DU TEMPS DE SEJOUR MOYEN

    SEUIL  ALPHA =      0.0100      (  2 18  4   ,    3 17 19   )
    SEUIL  ALPHA =      0.0500      (  2 20 49   ,    3 14 33   )
    SEUIL  ALPHA =      0.1000      (  2 22 15   ,    3 13  8   )

ESTIMATION DU TEMPS DE SEJOUR MOYEN :   3  5 41


REGION : PHYSIQUE         NO: 2    POUR RO =      0.0473    METHODE DES BLOCS LO =    26

NOMBRE DE BLOCS :   51
TAILLE D UN BLOC:   28

INTERVALLE  DE  CONFIANCE  DU TEMPS DE SEJOUR MOYEN

    SEUIL  ALPHA =      0.0100      (  0 19 26   ,    1  8 18   )
    SEUIL  ALPHA =      0.0500      (  0 20 58   ,    1  6 47   )
    SEUIL  ALPHA =      0.1000      (  0 21 45   ,    1  5 59   )

ESTIMATION DU TEMPS DE SEJOUR MOYEN :   1  1 52
```

** PRECISION SUR LES TEMPS DE SEJOUR MOYENS **

REGION	TPS MOYEN SEJOUR	ECART SEUIL 1%	ECART SEUIL 5%	ECART SEUIL 10%
ADMINISTRATIF	3 5 41	+OU- 0 11 37	+OU- 0 8 51	+OU- 0 7 26
PHYSIQUE	1 1 52	+OU- 0 6 26	+OU- 0 4 54	+OU- 0 4 7

Figure 9.14 - Intervalles de confiance aux seuils 1 %, 5 % et 10 % pour les temps de réponse moyens d'administratif et physique


```
**********************************
*** HISTOGRAMME *** ADMINISTRATIF
**********************************
```

Nombre de transactions

```
    I---------------------------------------->
 0  I
 1  I*******************************
 2  I*********************************
 3  I*****************************************
 4  I******************************************
 5  I*****************************************
 6  I***************************
 7  I**************
 8  I********
 9  I*
10  I
11  I
12  I
13  I
14  I
15  I
16  I
17  I
18  I
19  I
20  I
21  I
22  I
23  I
24  I
25  I
26  I
27  I
28  I
29  I
30  I
```

```
TOTAL ENTREES :      1089
VALEUR MINI.  :      1.42
VALEUR MAXI.  :    197.83

LARGEUR INTERVALLE :    24.00
INTERVALLE NO   1 :     1.00 A     25.00
INTERVALLE NO  65 :  1537.00 A   1561.00
```

Figure 9.15 - Exemple d'histogramme donnant la répartition des temps de réponse d'administratif

□

2.6 Validation et robustesse d'un modèle de simulation de système d'information

Pour pouvoir utiliser un modèle dans des expériences, il est préalablement nécessaire de le valider, c'est-à-dire de montrer qu'il reflète une réalité connue. Celle-ci peut être un autre modèle déjà validé ou un système réel connu par des mesures.

Comme dans (LER 77), nous examinons successivement deux cas :

- on dispose d'échantillons de mesures relevés sur un système réel ou pendant l'exploitation d'un autre modèle ;

- on ne dispose pas d'un tel échantillon et la validation est faite par rapport à un modèle mathématique.

N'ayant pas encore pu expérimenter convenablement ces méthodes dans le cadre de notre travail, nous n'en faisons qu'un exposé théorique rapide. Ce que nous dirons, notamment au sujet des files d'attente, a déjà été largement expérimenté dans le domaine de l'évaluation des systèmes informatiques. Pour celui des systèmes d'information, cela reste encore du ressort de la prospective.

2.6.1 Validation par rapport à la réalité ou à un autre modèle grâce à des échantillons de mesures

On suppose que l'on possède, pour un même processus réel X_t, deux échantillons de mesures :

. l'un, $X(t_i)$, $i = 1, \ldots, n$, provenant du modèle à valider ;

. l'autre, $X'(t_i)$, $i = 1, \ldots, n$, étant l'échantillon de référence.

a) Cas (quasi-) stationnaire

Dans le cas où le processus X_t est considéré comme (quasi-) stationnaire, on peut chercher à faire des comparaisons de grandeurs caractéristiques : on calcule par exemple les deux moyennes et leurs intervalles de confiance, et l'on vérifie s'il y a bien recouvrement.

Dans le cas où il y a indépendance entre les mesures, on peut procéder classiquement par analyse de variance pour tester si les échantillons proviennent de la même loi.

Une méthode utilisant une régression multilinéaire, et décrite dans (LER 77), permet de vérifier globalement l'accord entre les deux séries, moyennant l'usage de variables explicatives. S'il n'y a pas indépendance entre les mesures, on a recours à un modèle quadratique.

b) Cas non quasi-stationnaire

Dans ce cas, il faut chercher à montrer que tendances, mouvements saisonniers et résidus des deux séries chronologiques sont comparables.

On peut globalement avoir recours aux modèles de BOX et JENKINS (2.3). Chacune des deux séries donne lieu à l'identification et à l'estimation d'un modèle où les valeurs des paramètres varient dans des intervalles de confiance. Si les deux modèles sont identiques et si les intervalles de confiance se recouvrent, alors on peut considérer que les deux séries sont issues du même processus.

Cette méthode suppose bien entendu tout l'arsenal de l'étude des séries chronologiques (filtrage, analyse spectrale, régression...) que nous avons déjà évoqué.

2.6.2 Validation par rapport à un modèle mathématique

On cherche ici à se mettre dans des conditions telles que l'on puisse résoudre le modèle mathématiquement, de manière exacte ou approchée.

On doit alors se tourner vers la théorie des files d'attente et des réseaux de files d'attente.

Cette théorie fournit le plus souvent des résultats pour le régime stationnaire, et pratiquement pas pour le régime transitoire. Un des rares exemples connus dans ce dernier cas est celui de la résolution exacte d'une file M/M/1 (voir ci-après) à l'aide de fonctions de BESSEL (cf. par exemple (GEL 73), (BRA 75) ou (KLE 75)). Nous nous préoccupons ici seulement de la recherche de solutions stationnaires.

Nous examinons rapidement les différentes techniques utilisables, puis nous discutons de leur application à notre domaine.

a) Résolution exacte des files d'attente

D'une manière générale, le descripteur A/B/ m /K/M désigne une file d'attente à m serveurs, où A et B désignent les lois d'arrivée et de service, où K désigne la capacité maximale de la file (infinie s'il est omis), et M est la taille de la population de clients (illimitée s'il est omis). A et B prennent les valeurs :

M	pour Exponentielle
E_r	pour ERLANG à r étapes
H_r	pour Hyperexponentielle à r étapes
D	pour Déterministe
G	pour Générale.

La théorie classique des files d'attente concerne le plus souvent la

détermination de solutions stationnaires, sous la forme de densités-limites de probabilités.

Ainsi, si l'on note

p (k, t) la probabilité pour que le nombre de clients dans la file (y compris ceux en service) soit k à l'instant t, on recherche

$$p_k = \lim_{t \to +\infty} p(k, t)$$

On en déduit le nombre moyen de clients à l'état stationnaire, ainsi que le temps de réponse moyen.

Il existe un nombre considérable de résultats de cette nature pour des cas particuliers (cf. KLE 75)

M/M/1, M/M/m, M/M/∞, M/M/1/K, M/M/m/m, M/M/1//M, M/M/∞//M, M/M/m/K/M,

$M/E_r/1$, $E_r/M/1$, éventuellement avec arrivées ou services en groupes de tailles variables,

$M/H_r/1$, $H_r/M/1$, $H_{r_a}/H_{r_b}/1$, ...

Dans des cas plus généraux : M/G/m, G/M/m, G/G/1..., les solutions s'expriment par des transformées de LAPLACE, dont l'inversion est souvent impossible.

Cette théorie peut convenir pour des études ponctuelles bien localisées. Dans le cas d'un système complexe, il faut envisager des réseaux de files d'attente.

b) Résolution exacte des réseaux de files d'attente

Considérons un réseau de files d'attente tel que celui de la figure 9.16, ouvert ou fermé.

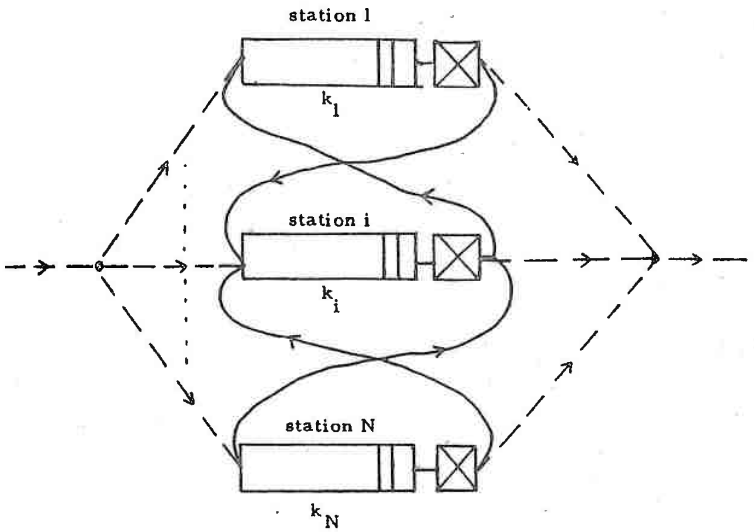


Figure 9.16 - Réseau de N files d'attente ouvert (avec tirets) ou fermé (sans tirets)

k_i désigne le nombre de clients à la station i (y compris ceux qui sont en service).

$K = \sum_{i=1}^{N} k_i$ désigne le nombre de clients dans le système (il est constant si le réseau est fermé).

$k = (k_1, \ldots, k_N)$ est le vecteur donnant le nombre de clients aux N stations, c'est-à-dire l'état du système.

$p(k,t)$ est la probabilité que le système se trouve dans l'état k à l'instant t.

Si, pour tout k, $\lim_{t \to \infty} p(k, t) = p(k)$ existe, on dit que le système possède une solution stationnaire, sous la forme de la distribution-limite $p(k)$.

La plupart des solutions connues font appel à des hypothèses restrictives que nous donnons en rappelant les principaux résultats exacts ou approchés.

- Théorème de J.R. JACKSON (JAC 63)

On fait les hypothèses suivantes :

. les probabilités de transition p_{ij} entre deux stations sont constantes :

p_{ij} = probabilité qu'un client sortant de la station i aille à la station j.

Pour un réseau ouvert, les probabilités d'entrée ou de sortie sont constantes :

p_{oi} = probabilité qu'un client entrant dans le système se dirige à la station i

$p_{i\,N+1}$ = probabilité qu'un client sortant de la station i se dirige vers l'extérieur.

. Pour un réseau ouvert, les clients parviennent dans le système selon un processus de POISSON de paramètre $\lambda(K)$.

. Les temps de service aux stations sont distribués selon une loi Exponentielle négative de paramètre $\mu(k_i)$.

. Dans les stations, la discipline de service est premier arrivé - premier servi.

On démontre alors que :

. pour un réseau fermé, la solution stationnaire existe toujours ;

. pour un réseau ouvert, la solution stationnaire existe sous réserve que deux conditions simples soient satisfaites ;

. sous réserve d'existence, la solution stationnaire est calculable.

Nous ne donnons pas la forme de cette solution, qui s'exprime simplement comme un produit. Pour plus de détails, on consultera par exemple (GEL 75) ou (KLE 76).

Les inconvénients du théorème de JACKSON sont évidemment contenus dans les hypothèses ci-dessus. D'autres résultats plus récents permettent de s'en libérer assez largement en généralisant les résultats ci-dessus.

- Théorème de BCMP (F. BASKETT, K.M. CHANDY, R.R. MUNTZ, et F.G. PALACIOS) (BAS 75).

Les hypothèses en sont les suivantes :

. On suppose qu'il y a R classes de clients et qu'un client peut changer de classe en accord avec des probabilités de transition constantes :

$p_{i,r\,;\,j,s}$: probabilité qu'un client de la classe r qui quitte la station i aille à la station j dans la classe s.

La matrice de transition $P = (p_{i,r\,;\,j,s})$ peut être considérée comme définissant une chaîne de MARKOV dont les états sont étiquetés par les couples (i, r). On suppose pour simplifier (sinon, voir (BAS 75)) que cette chaîne est ergodique et indécomposable.

. Si le réseau est ouvert, les clients parviennent dans le système selon un processus de POISSON de paramètre $\lambda(K)$, où K est le nombre de clients dans le système, et

q_{ir} = probabilité qu'un client entrant dans le système se dirige vers la station i dans la classe r.

Toujours quand le réseau est ouvert, un client de la classe r, qui quitte la station i, se dirige vers l'extérieur avec la probabilité

$$1 - \sum_{\substack{1 \le j \le N \\ 1 \le s \le R}} p_{i,r\,;\,j,s}$$

. Les stations sont de 4 types :

type 1 : la discipline de service est premier arrivé - premier servi ; tous les clients ont la même distribution de temps de service Exponentielle négative avec un taux de service $\mu(k_i)$ où k_i est le nombre de clients à la station i ;

type 2 : la station a un seul serveur qui est un processeur-partagé ; chaque classe de clients peut avoir une distribution de temps de service distincte, qui suit nécessairement une loi de COX (fonction à transformée de LAPLACE rationnelle) ;

type 3 : la station contient un nombre infini de serveurs ; chaque classe de clients peut avoir une distribution de temps de service distincte, qui suit une loi de COX ;

type 4 : la station possède un seul serveur et la discipline de service est dernier arrivé - premier servi avec préemption et reprise ("preemptive-resume") ; chaque classe de clients peut avoir une distribution de temps de service distincte, qui suit une loi de COX.

Une loi de COX peut être représentée par un réseau d'étapes Exponentielles de la forme de la figure 9.17.

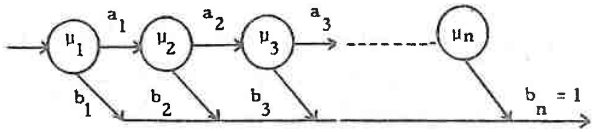


Figure 9.17 - Représentation des distributions de Cox par la méthode des étapes

Dans cette figure, b_i est la probabilité qu'un client quitte après la i^e étape, et $a_i = 1 - b_i$ est celle pour qu'il aille à l'étape suivante. La

distribution des temps de service à l'étape i suit une loi Exponentielle négative de moyenne $1/\mu_i$.

L'état du réseau est représenté par le vecteur $x = (x_1, \ldots, x_N)$, où x_i représente les conditions à la station i.

L'interprétation de x_i dépend du type de station :

type 1 : $x_i = (x_{i1}, \ldots, x_{ik_i})$, où k_i est le nombre de clients à la station i et x_{ij} est la classe du j^e client ;

type 2 ou 3 : $x_i = ((x_{i1}, \ell_{i1}), \ldots, (x_{ik_i}, \ell_{ik_i}))$, où ℓ_{ij} est l'étape atteinte dans la représentation de Cox de la distribution des services à la station i par le client j ;

type 4 : $x_i = ((x_{i1}, \ell_{i1}), \ldots, (x_{ik_i}, \ell_{ik_i}))$, où x_{ij} est la classe du j^e client se trouvant à la station i, dans l'ordre inverse d'arrivée, et ℓ_{ij} est l'étape qu'il occupe dans la représentation de COX.

On note :

$\mu_i(j)$: le taux de service Exponentiel dans une station i de type 1 contenant j clients ;

$\mu_{ir\ell}$: le taux de service Exponentiel dans une station i de type 2, 3 ou 4, pour la classe r, à l'étape ℓ de la loi de Cox ;

$A_{ir\ell} = \prod_{j=1}^{\ell} a_{irj}$: la probabilité qu'un client de la classe r à la station i atteigne la ℓ^e étape de son service (a_{irj} est le coefficient a_j de la figure 9.17) ;

$k_{ir\ell}$: le nombre de clients de la classe r à la station i à la ℓ^e étape de leur service ;

u_{ir} : le nombre d'étapes de la loi de COX à la i^e station pour la classe de clients r.

Soit $(e_{js} ; 1 \leq j \leq N, 1 \leq s \leq R)$ la solution du système d'équations :

$$e_{js} = \sum_{i=1}^{N} \sum_{r=1}^{R} e_{ir} \, p_{i,r\,;\,j,s} + q_{js}$$

Le théorème de BCMP s'énonce ainsi :

Pour un réseau indécomposable, ouvert ou fermé, de stations de types 1, 2, 3 ou 4, les probabilités stationnaires sont données par :

$$p(x) = C \, d(K) \prod_{i=1}^{N} f_i(x_i)$$

où C est une constante de normalisation (somme des probabilités égale à 1), d (K) est une fonction du nombre K de clients dans le système, et f_i une fonction qui dépend du type de la station i :

type 1 : $f_i(x_i) = \prod_{j=1}^{k_i} (e_{ix_{ij}} / \mu_i(j))$

type 2 : $f_i(x_i) = k_i ! \prod_{r=1}^{R} \prod_{\ell=1}^{u_{ir}} \{(e_{ir} A_{ir\ell} / \mu_{ir\ell})^{k_{ir\ell}} / k_{ir\ell} !\}$

type 3 : $f_i(x_i) = \prod_{r=1}^{R} \prod_{\ell=1}^{u_{ir}} \{(e_{ir} A_{ir\ell} / \mu_{ir\ell})^{k_{ir\ell}} / k_{ir\ell} !\}$

type 4 : $f_i(x_i) = \prod_{j=1}^{k_i} \{e_{ix_{ij}} A_{x_{ij}\ell_{ij}} / \mu_{ix_{ij}}\}$

$$d(K) = \begin{cases} \prod_{m=0}^{K-1} \lambda(m) & \text{si le réseau est ouvert} \\ 1 & \text{s'il est fermé.} \end{cases}$$

On obtient dans les mêmes conditions, en faisant des sommations marginales, des résultats plus synthétiques et, notamment, la probabilité -

limite p (k), où $k = (k_1, \ldots, k_N)$ donne le nombre de clients k_i à chaque station i (BAS 75).

Bien qu'il soit le résultat exact le plus fort sur les réseaux, ce théorème limite encore fortement les types de stations, des lois d'arrivée et de service. Pour des cas plus généraux, on peut se rabattre sur des techniques d'approximation.

c) Résolution approchée des files ou réseaux de files d'attente

- Approximation fluidique (KLE 76)

Plutôt que de vouloir des résultats analytiques très fins, on peut se contenter, éventuellement dans un premier temps, de résultats plus grossiers permettant de "dimensionner" le problème.

Dans le cas d'une validation, ceci doit permettre de vérifier sommairement le bien-fondé d'un modèle de simulation.

On traite alors les systèmes de files d'attente comme des flux continus plutôt que comme des flux discrets de clients : cette approximation est surtout valable quand on se trouve dans des conditions de trafic élevé, avec formation quasi-permanente d'une file d'attente, et des temps d'attente bien supérieurs aux temps de service.

On définit deux processus stochastiques d'arrivée et de départ :

a (t) = nombre des arrivées dans [o, t]

b (t) = nombre de départs dans [o, t].

Le nombre de clients présents dans le système à l'instant t est noté N (t). Pour N (o) = 0,

$$N(t) = a(t) - b(t).$$

Quand t devient grand on peut s'attendre à de faibles déviations relatives de a (t) par rapport à sa moyenne $E(a(t)) = \overline{a(t)}$; par la loi des grands nombres :

$$\lim_{t \to \infty} \text{p. s.} \quad \frac{a(t) - \overline{a(t)}}{\overline{a(t)}} = 0$$

Ceci suggère, pour une approximation du 1er ordre, de remplacer le processus a (t) par sa valeur moyenne fonction du temps : on appelle approximation fluidique le remplacement du processus stochastique discontinu a (t) pour le processus déterministe continu $\overline{a(t)}$.

De la même manière, b (t) est remplacé par $\overline{b(t)}$ et $N(t) = \overline{a(t)} - \overline{b(t)}$ est une fonction continue du temps.

On peut alors penser faire ce genre de transformation pour toutes les stations intervenant dans une maquette. Ceci nous rapproche un peu des conceptions de FORRESTER (FOR 61, FOR 68) et de ses flux.

Cependant, on montre facilement que cette approximation du 1er ordre, qui néglige l'influence de la variance des processus, peut conduire à des sous-estimations importantes des longueurs de files d'attente, et donc, des temps de réponse.

- Approximation par diffusion

Ces résultats s'appliquent aux files d'attentes G/G/1, avec une discipline premier arrivé - premier servi.

On considère toujours une approximation fluidique, mais du 2è ordre, en tenant compte des variations de a (t) et b (t) autour de leurs moyennes. Ces fluctuations sont représentées par des distributions Normales.

On se place ici dans le cas où N (t) est grand, de telle manière que l'on puisse considérer le processus de départ b (t) comme indépendant de celui d'arrivée a (t). Ainsi, puisque $N(t) = a(t) - b(t)$,

$$\overline{N(t)} = \overline{a(t)} - \overline{b(t)} \text{ et } \sigma^2_{N(t)} = \sigma^2_{a(t)} + \sigma^2_{b(t)}$$

On suppose que les durées interarrivées sont indépendantes et identiquement distribuées, avec une moyenne $1/\lambda$ et une variance V_a. Il en est de même des durées de service avec la moyenne $1/\mu$ et la variance V_b.

On démontre (KLE 76) que N (t) peut être approché par une variable aléatoire continue X (t) dont la densité de probabilité :

$$f(x, t)\,dx = Pr\ [\ x \leq X(t) < x + dx\ ]$$

satisfait, pour $X(t) > 1$, l'équation de diffusion :

$$-\frac{\partial f}{\partial t} - \beta \frac{\partial f}{\partial x} + \frac{1}{2}\,\alpha \frac{\partial^2 f}{\partial x^2} = 0$$

où $\beta = \lambda - \mu$ et $\alpha = \lambda^3 V_a + \mu^3 V_b$.

E. GELENBE a généralisé ce résultat pour X (t) dans $[\ o, \infty\ [$ à l'aide de barrières absorbantes à l'origine (GEL 75a).

Ces résultats peuvent être appliqués aux réseaux de files d'attente en régime stationnaire (voir par exemple (GEL 75b)).

d) Application à la validation des modèles de systèmes d'information

On reste perplexe devant le nombre et la diversité des modèles mathématiques exacts ou approchés, et l'on peut se demander dans quelles conditions ils sont bien adaptés à notre domaine.

Tout d'abord, on doit s'interroger sur la manière de concevoir nos maquettes en termes de réseaux de files d'attente. Tout naturellement, les processus apparaissent comme des clients pour les ressources qui sont les stations. Les processus étant souvent cycliques, le service d'un client correspond généralement à une partie d'un cycle destiné à traiter des données : les supprimer à tel endroit, les faire cheminer dans le système après une transformation plus ou moins importante, les stocker

à un autre endroit. Les véritables clients de stations parfois fort complexes sont donc tout de même les données et c'est sous cet angle que nous tentons d'appliquer les notions précédentes.

De nombreuses difficultés surgissent : elles sont dues aux transformations que subissent les données et surtout aux interruptions et blocages nombreux dus à l'état du système et au temps. Nous allons tenter de les sérier et de dire comment elles peuvent être aplanies pour une validation.

- Nous avons vu que les probabilités de transition dans les réseaux de files d'attente théoriques sont des constantes. Cette condition n'est pas extrêment contraignante, parce que les réseaux sous-jacents aux maquettes sont souvent simples, sans beaucoup de liaisons, et parce que l'on peut généralement se ramener à ce cas de manière approximative.

- Même si l'on fait abstraction des calendriers de disponibilité, la diversité des stations dans nos modèles semble un obstacle plus important. C'est le cas par exemple pour les ressources avec priorités et préemptibles : certes, ce cas est traité théoriquement dans la littérature (KLE 76) pour des files simples, mais il ne l'est pas pour des réseaux de files dont certaines ont ce comportement.

- Les lois d'arrivée ou de service efficaces dans les modèles résolus exactement sont des lois Exponentielles négatives, ou leurs dérivées : lois d'ERLANG, Hyperexponentielles, de COX. Nous manquons de résultats expérimentaux pour pouvoir affirmer que ces lois suffisent ou non dans la plupart des cas, pour des études en régime stationnaire. Et même dans le cas où ces lois ne s'appliquent pas exactement, on pourrait penser à les utiliser tout de même, moyennant une certaine "robustesse" des modèles par rapport à elles (cf. par exemple les études de D. POTIER (POT 77)). Dans le cas général, on doit sans doute se tourner vers les techniques d'approximation, comme la diffusion.

- L'obstacle qui nous parait déterminant est celui des contraintes de synchronisation très importantes imposées au traitement des données dans les maquettes.

En effet, un processus ne peut pas toujours être lancé dès qu'une donnée à traiter est présente : il doit souvent attendre qu'un signal lui parvienne. Dans notre système, ceci est par exemple réalisé par l'intermédiaire de moniteurs d'activation, éventuellement alimentés par des échéanciers à certaines dates.

De plus, un processus lancé doit parfois attendre la disponibilité de ressources pour continuer son exécution. Enfin, il peut être interrompu par un autre processus, par exemple appartenant à un calendrier pour lui faire attendre une nouvelle période de disponibilité.

Cette complexité des systèmes d'information, ces "à-coups" dans le traitement des données, anéantissent les chances d'une résolution analytique exacte ou approchée de beaucoup de maquettes construites. De plus, du fait des interruptions et blocages répétés, il n'est pas certain que l'on atteigne jamais un régime permanent, du point de vue de la théorie des files d'attente : on peut se trouver dans un perpétuel état transitoire.

- On doit alors se résigner à l'une ou l'autre des solutions suivantes :

. faire des validations partielles de maquettes, pour les stations que l'on sait résoudre exactement ou approximativement ;

. se placer dans des conditions simplificatrices : supprimer échéanciers, calendriers ... pour effacer des causes de blocage et avoir une chance de résolution mathématique ;

. voir le problème de plus haut, ce qui doit permettre de s'affranchir de certaines contraintes de détail, qui éloignent des modèles théoriques.

Quand on a pu déterminer les conditions de résolution mathématique, on exécute le modèle de simulation dans ces conditions, et l'on détermine des résultats avec des intervalles de confiance (des moyennes par exemple). La validation consiste à vérifier que les résultats théoriques tombent dans ces intervalles.

On peut espérer à l'avenir éviter d avoir à écrire soi-même les programmes nécessaires aux calculs de solutions exactes ou approchées de réseaux de files d'attente, grace à l'utilisation de produits-programmes paramétrés qui font la synthèse de différentes méthodes, tels QNAP : Queueing Network Analysis Package (MER 78).

2.6.3 Robustesse d'un modèle

On dit qu'un modèle est robuste par rapport à une hypothèse du modèle si le fait de s'écarter de cette hypothèse n'entraine pas de modification sensible des résultats.

Cette propriété est importante, parce que, si le modèle a été validé sous l'hypothèse en question, elle permet de l'utiliser dans d'autres conditions avec une certaine sécurité. Ceci est la base de toute prédiction.

Ainsi, pour un réseau de files d'attente, D. POTIER, dans (POT 77), traite de robustesse par rapport à :

. la nature de la source
. la distribution des temps de service
. la capacité des files d'attente.

Il est bien entendu que le fait d'écarter une hypothèse peut affecter certaines sorties et pas d'autres : D. POTIER constate parfois en même temps :

. une certaine stabilité des taux d'utilisation des serveurs
. un écart important entre les temps de réponse.

Malheureusement, il n'existe pas de résultats généraux sur ces questions, hormis ceux que nous venons de citer. On est donc souvent contraint à l'heure actuelle à supposer la robustesse d'un modèle que l'on veut utiliser hors de son domaine de validation.

2.7 Plan d'expérimentation

Quand un modèle a été validé, le plus souvent en régime permanent, on peut vouloir, sous réserve de sa robustesse, se livrer sur lui à de nombreuses expériences.

2.7.1 Régime permanent

Les unes sont réalisées en régime permanent : des modification de la structure du modèle, le changement de certains de ses composants, la modification de lois de probabilité, notamment d'entrée ou de service, ...

Exemple 9.16

Dans le modèle du chapitre 5, on peut être amené à tester des modifications diverses ; par exemple :

- *examiner l'influence qu'aurait la prise en compte des commandes téléphoniques par x au lieu de y ;*
- *changer les tranches d'heures des périodes de disponibilité des calendriers, et conjointement celles des échéanciers ;*
- *supprimer une personne ou, au contraire, en doubler une autre, en diminuant son temps de disponibilité ;*
- *augmenter le flux d'arrivée des commandes jusqu'à la perte de l'équilibre ;*

- *avoir des calendriers qui interrompent le travail au lieu de le laisser continuer en fin de période de disponibilité ;*

...

□

Dans ces conditions, on peut se préoccuper de l'influence de tels ou tels facteurs sur le comportement du modèle ; ce type d'études peut être mené systématiquement par essais successifs et analyse factorielle (KLEIJ 75).

2.7.2 Régime transitoire

D'autres expériences peuvent être faites en régime transitoire, avec les difficultés que nous avons déjà signalées, notamment la quasi-nécessité de faire des répliques nombreuses de la simulation, afin d'avoir plusieurs échantillons à analyser statistiquement.

Exemple 9.17

On peut tester le modèle du chapitre 5 en régime transitoire, par exemple :

- *examiner l'effet de l'arrivée brutale d'une grosse commande : ceci se produit parfois dans l'entreprise considérée et a un effet d'engorgement pour plusieurs jours, avec résorption progressive ;*
- *étudier l'effet d'arrivées à variations cycliques régulières ;*
- *tester le comportement du système soumis à une croissance régulière des commandes.*

...

□

Tous ces essais ont pour but de savoir comment réagirait le système, en supposant qu'il reste figé, face à des situations inhabituelles.

Une phase ultérieure serait sans doute de chercher à ce qu'il s'auto-adapte à de tels événements ; les techniques d'adaptation feraient alors à nouveau l'objet de tests par simulation sur ordinateur.

2.7.3 Autres problèmes

Signalons enfin deux facteurs auquels il faut prendre garde lors d'expérience de simulation : les conditions initiales et finales.

a) Conditions initiales

Considérons un processus stochastique quasi-stationnaire $X(t)$, dont on a recueilli par simulation une trajectoire depuis l'instant t_o. Une mesure faite à un instant $t > t_o$ dépend de la valeur initiale $X(t_o)$, et une estimation de la moyenne est en fait une estimation de la moyenne conditionnelle : $E(X(t)/X(t_o))$.

Dans certains types de simulations, c'est ce qui est souhaité, mais dans d'autres, on préfère s'affranchir des conditions initiales.

D'une façon générale, on cherche à diminuer le biais introduit par les conditions initiales :

- en choisissant un $X(t_o)$ pas trop éloigné de la moyenne pour raccourcir la phase transitoire ;
- en augmentant la taille des échantillons prélevés ;
- en supprimant les premières mesures : si (X_i), $i = 1, \ldots, n$, est l'échantillon relevé à intervalles constants à partir de t_o : au lieu d'estimer la moyenne par

$$\overline{X} = \frac{1}{n} \sum_{i=1}^{n} X_i, \text{ on l'estime par } \overline{X}_{n^*} = \frac{1}{n-n^*} \sum_{i=n^*+1}^{n} X_i,$$

par suppression des n^* premières mesures.

Toute la question est de choisir convenablement n^*. Il n'existe pas de réponse théorique, mais seulement des recettes empiriques (voir par exemple G. FISHMAN (FIS 73)).

Exemple 9.18

Dans notre système, nous avons donné à l'utilisateur la possibilité de faire tourner la simulation "à blanc", c'est-à-dire sans prendre en compte les mesures, pendant une période qu'il détermine.

□

b) Conditions finales ; arrêt de la simulation

Considérons le cas d'une file d'attente : si l'on désire obtenir la longueur moyenne de la file et le temps de réponse moyen, on est conduit à établir deux échantillons :

- longueurs de la file relevées à intervalles constants :

$$(X_i),\ i = 1, \ldots, n$$

- temps de réponse pour les N premiers clients sortis :

$$(Y_i),\ i = 1, \ldots, N.$$

Si, dans la même simulation, on veut calculer les deux résultats, on est placé dans l'alternative suivante :

- si n est fixé (≃ durée de la simulation fixée), alors N est une variable aléatoire et $\overline{Y} = \frac{1}{N} \sum_{i=1}^{N} Y_i$ est un estimateur biaisé ;
- si N est fixé, alors n est une variable aléatoire et $\overline{X} = \frac{1}{n} \sum_{i=1}^{n} X_i$ est un estimateur biaisé.

Ici encore, un remède est d'augmenter la taille des échantillons, ou de la durée de la simulation, de manière à atténuer les biais.

De plus, on cherche à arrêter la simulation quand la précision souhaitée sur les résultats est atteinte (LEG 68, FIS 73).

Exemple 9.19

Dans notre système, si l'on peut estimer que, sur de longues périodes, les biais sur les longueurs moyennes de files et les taux moyens d'occupation de ressources sont négligeables, il faut prendre garde à ceux sur les temps moyens de réponse. On se trouve en effet à peu de choses près dans la situation décrite ci-dessus.

□

3 - OUTILS de MESURE et d'ANALYSE STATISTIQUE dans le SYSTEME MAESTRO

Nous présentons ici l'état de ces questions dans la version actuelle de notre système. Elle est conforme à ce que nous avons dit précédemment et elle nous permet d'obtenir des résultats tels ceux donnés sur la maquette du chapitre 5.

Nous nous bornons ici aux caractéristiques techniques générales, à la forme syntaxique, et à l'utilisation des outils créés. Pour les détails de réalisation, nous renvoyons à l'annexe C et à (THA 80).

3.1 Caractéristiques techniques générales

Nous utilisons de bout en bout le langage SIMULA 67, aussi bien pour ce qui concerne la prise de mesures, l'édition et l'analyse statistique. Afin de ne pas trop alourdir les programmes de simulation, notamment par une occupation - mémoire excessive, nous avons décidé

de séparer nettement ces trois phases et, donc, de créer des fichiers séquentiels (qui sont les seuls autorisés dans la version actuelle sur IRIS 80) intermédiaires.

Ceux-ci sont au nombre de quatre :

- fichier FGEN : il contient les résultats généraux sur les files, les ressources et les régions ; il est créé à la fin de la simulation et est utilisé lors de la phase d'édition ;

- fichier FVOL : il contient les séries chronologiques donnant l'évolution des valuations des structures de données au cours du temps ; il est complété au fur et à mesure du déroulement de la simulation ;

- fichier FOCC : il contient les séries chronologiques concernant les nombres d'accès utilisés dans les ressources au cours du temps ; il est constitué pendant l'exécution du programme de simulation ;

- fichier FREP : il contient les mesures relatives aux temps de séjour des transactions dans les régions ; il est constitué pendant la simulation ;

Le fichier FGEN est utilisé pour éditer les résultats synthétiques généraux sur les volumes d'information, les ressources et les régions. Les trois autres sont utilisés dans des études statistiques ultérieures d'analyse de séries chronologiques. Dans les cas les plus simples (quasi-stationnarité), ils permettent d'obtenir des moyennes et des intervalles de confiance.

Nous ne détaillons pas la forme de leur contenu et renvoyons le lecteur que cela intéresse à (THA 80).

Pour obtenir des résultats dans ces fichiers, l'usager doit le prévoir dans le programme de simulation en utilisant des objets aux caractéristiques précises et des procédures que nous présentons maintenant.

3.2 Description des objets spécifiques aux mesures

3.2.1 Les transactions

Les objets qui entrent dans les files ou les régions appartiennent à la classe transaction, ou à une de ses sous-classes, ce qui permet de leur adjoindre tous les attributs nécessaires à une bonne description. Comme nous l'avons déjà dit, la classe transaction est elle-même sous-classe de la classe message :

```
link class message (no) ; integer no ; null ;
message class transaction (pds) ; real pds ;
begin
    real hdebf ; hdebrg
end
```

Le paramètre no permet de caractériser un message.

Le paramètre pds donne le "poids" de la transaction.

Les réels hdebf et hdebrg servent à noter les dates d'entrée dans une file et une région.

3.2.2 Les files

C'est un cas particulier de structure de données qui est très utilisé et a été systématisé. Nous avons déjà présenté (au chapitre 6) la classe tfile en tant que type de structures de données. Nous complétons sa description pour tenir compte des mesures, mais le texte en est complètement détaillé en annexe C.

```
monitor class tfile (mesure, équilibre, inter, dmes) ;
    boolean mesure, équilibre ; real inter, dmes ;
```



```
begin
    ... déjà vu précédemment ...
    if mesure or équilibre then into (fileq) ;
    if équilibre
    then begin
        ...
        activate new mesurevol (this tfile)
                qua process delay dmes
        end
end
```

La signification des paramètres formels a déjà été donnée (chapitre 6).

Dans le cas où l'on veut faire des mesures pour des résultats généraux ou détaillés, on met la file dans une liste de files générale fileq, qui permet de constituer le fichier FGEN en fin de simulation.

Dans le cas où l'on désire des résultats détaillés, un processus cyclique de sondage, de la classe mesurevol, est activé à la date dmes (début des mesures) ; il est ensuite réveillé après des périodes de inter heures.

Notons que, pour mesurer la file, ce processus ne respecte pas la contrainte d'exclusion mutuelle imposée par le mécanisme de moniteur. Ceci est naturel si l'on ne veut pas biaiser le recueil des échantillons, et est permis par le "laxisme" de SIMULA en matière de protection (bien commode parfois...).

3.2.3 Les ressources

Outre les mécanismes d'allocation déjà décrits (chapitre 6), les ressources contiennent des outils assez complexes pour le recueil des mesures, qui sont en majeure partie inclus dans la classe ressource détaillée complètement en annexe C. Nous en donnons ici seulement la structure :

```
monitor class ressource ;

begin
    ...
    real hdispo, hdeb, tpsdispo, tpsutil, tpsinutil, array util (1:10) ;
    real accès, rpmaxdispo, rpmaxutil ;
    integer rsort, nbutil, nbmaxutil ;
    procedure statacquérir (x) ; real x ; ... ;
    procedure statlibérer (x) ; real x ; ...;
    procedure statutilisateur (nb) ; integer nb ; ... ;
    procedure statservis ; ... ;
    procedure statinit (...) ; ... ;
    procedure statréinit ; ... ;
    ...
    new refres (this ressource) . into (ressourceq)
end
```

La signification des attributs locaux est la suivante :

hdispo : date de début de disponibilité de la ressource

hdeb : date de la dernière modification dans l'utilisation des accès

tpsdispo : cumul des durées de disponibilité

tpsutil : cumul des durées d'utilisation

tpsinutil : cumul des durées de non-utilisation

util (1:10) : tableau servant à cumuler les durées d'utilisation par nombre d'accès utilisés

accès : nombre d'accès utilisés

rpmaxdispo : nombre d'accès maximum disponibles

rpmaxutil : nombre d'accès maximum utilisés

rsort : nombre de processus ayant requis un ou plusieurs accès

nbutil : nombre d'utilisateurs de la ressource

nbmaxutil : nombre maximum d'utilisateurs de la ressource.

L'appel de la procédure statacquérir permet la mise à jour d'attributs locaux pendant une acquisition d'accès ; il permet aussi, si on l'a

demandé, l'enregistrement du nombre d'accès utilisés à cet instant dans le fichier FOCC.

L'appel de la procédure statlibérer a un effet similaire pendant une libération d'accès.

L'appel de la procédure statutilisateur met à jour le nombre d'utilisateurs communs de la ressource, dans le cas d'un processeur partagé.

L'appel de la procédure statservis provoque la mise à jour du nombre total des processus ayant acquis les accès voulus sur la ressource pour s'exécuter.

L'appel de la procédure statinit permet d'initialiser les attributs locaux et celui de la procédure statréinit de les réinitialiser, en cas de redémarrage d'une simulation.

Enfin, toutes les ressources sont associées à des étiquettes, de la classe refres, qui sont chaînées dans la liste de ressources ressourceq. Celle-ci permet de compléter le fichier FGEN à la fin de la simulation.

Notons que tous ces outils sont complètement transparents aux utilisateurs de composants standard du système MAESTRO.

Nous avons vu au chapitre 6 des exemples d'utilisation de ressources incluant ces mécanismes.

3.2.4 Les régions

Une région est essentiellement un compteur incrémenté (décrémenté) à chaque entrée (sortie) de transaction. Les valeurs courante, moyenne, maximale, minimale du contenu et des temps de réponse sont tenues à jour.

La classe trégion, dont le détail complet est donné en annexe C, a la structure suivante :

```
link class trégion (nom) ; text nom ;
begin
    integer rgent, rgsort, rgmax, rgmin, rgmoy ;
    real rgtpsmin, rgtpsmax, rgtpsat, rgtpsmoy ;
    procedure entrer (tr) ; ref (transaction) tr ;... ;
    procedure sortir (tr) ; ref (transaction) tr ; ... ;
    procedure réinit ; ... ;
    into (régionq)
end
```

Le paramètre nom permet d'identifier la région.

La signification des attributs est la suivante :

rgent	:	cumul du nombre d'entrées
rgsort	:	cumul du nombre de sorties
rgmax	:	contenu maximum
rgmin	:	contenu minimum
rgmoy	:	contenu temporel moyen
rgtpsmin	:	temps de réponse minimum
rgtpsmax	:	temps de réponse maximum
rgtpsat	:	cumul des temps de réponse des transactions sorties
rgtpsmoy	:	temps d'attente moyen temporel.

L'appel de la procédure entrer met à jour ces attributs locaux et permet de noter l'heure d'entrée dans la transaction concernée.

L'appel de la procédure sortir a un effet similaire. Elle permet aussi, si l'utilisateur l'a demandé, de constituer un enregistrement dans le fichier FREP, pour les analyses ultérieures.

L'appel de la procédure réinit a lieu lorsqu'une nouvelle simulation commence pour réinitialiser les attributs locaux.

Chaque région est chaînée dans une file de régions régionq, utilisée pour l'édition des résultats synthétiques.

Exemple 9.20

Dans l'application du chapitre 5, on déclare et créé les régions administratif et physique de la manière suivante :

```
ref (trégion) adm, phy ;
...
adm :- new trégion ('ADMINISTRATIF') ;
phy :- new trégion ('PHYSIQUE') ;
```

□

3.3 Processus de prise de mesure

A chaque file pour laquelle des résultats détaillés (fichier FVOL) ont été demandés, un processus de mesure cyclique est associé (3.2.2). Il appartient à la classe mesurevol, détaillée complètement en annexe C, dont la structure est :

```
process class mesurevol (fℓ) ; ref (tfile) fℓ;
begin
    while true do
    inspect fℓ do
    begin
        ...
        reactivate this process delay inter ;
        écrifvol (...)
    end
end
```

Le paramètre fℓ désigne la file associée au processus.

Ce processus est activé tous les inter unités de temps à partir de la date dmes (voir 3.2.2).

Le rôle de l'appel de la procédure écrifvol est de créer un enregistrement de mesure dans le fichier FVOL.

Cette classe fait partie de la classe MAESTRO (annexe C).

3.4 Exécution d'une simulation

Lorsque les objets décrits dans le programme de simulation sont créés, et les processus activés, on fixe les conditions du déroulement de la simulation par appel de la procédure simul appartenant à la classe MAESTRO (annexe C) :

```
procedure simul (duréesimul, resgen, équilibre, brep, bocc) ;
    real duréesimul ; boolean resgen, équilibre, brep, bocc ;...;
```

La signification des paramètres est la suivante :

duréesimul	:	fixe la durée simulée de l'exécution (en heures)
resgen	:	booléen true si et seulement si l'on veut des résultats généraux (utilisation du fichier FGEN)
équilibre	:	booléen true si et seulement si l'on veut des résultats détaillés sur certaines files (utilisation du fichier FVOL)
brep	:	booléen true si et seulement si l'on veut des résultats détaillés sur les temps de réponse (utilisation du fichier FREP)
bocc	:	booléen true si et seulement si l'on veut des résultats détaillés sur l'occupation des ressources (utilisation du fichier FOCC).

Plusieurs appels consécutifs de cette procédure peuvent être écrits pour effectuer plusieurs simulations successives. Ceci permet notamment de faire exécuter le modèle "à blanc" pour éliminer des périodes transitoires.

Exemple 9.21

Dans la maquette du chapitre 5 présentée en annexe B, deux appels successifs ont été faits :

```
simul (120, true, false, false, false)
```

permet de faire fonctionner le programme pendant 120 heures simulées (5 jours) de manière à laisser stabiliser le modèle ; on enregistre tout de même des résultats généraux sur la période transitoire ;

```
simul (1440, true, true, true, true)
```

permet d'exécuter le programme ensuite pendant 60 jours simulés, avec utilisation de toutes les options permises.

□

3.5 Procédures pour obtenir des résultats détaillés

D'autres procédures sont écrites qui permettent de préciser sur quels objets on désire obtenir des renseignements détaillés :

```
procedure préciseressource (res) ; ref (ressource) res ; ... ;

procedure préciserégion (rg) ; ref (trégion) rg ;... ;

procedure historégion (rg, n, inf, pas, larg, haut) ;
    ref (trégion) rg ; integer n larg, haut ; real inf, pas ;
    ... ;
```

L'appel de la procédure préciseressource permet de demander la prise de mesures détaillées sur l'occupation de la ressource res, et leur insertion dans le fichier FOCC.

L'appel de la procédure préciserégion a le même effet pour les temps de réponse dans la région rg et le fichier FREP.

Notons que la prise de mesures détaillées sur une file est réglée au moment de la création de cette file (chapitre 6). Cette différence de traitement est due au fait que les mesures sur les files sont réalisées à intervalles de temps constants, alors que celles sur ressources et régions sont réalisées quand se produisent des changements d'état, à cause des contraintes imposées par les calendriers.

La procédure historégion permet de calculer un histogramme des temps de réponse associés à la région rg.

Les paramètres ont la signification suivante :

n : fixe le nombre de classes indicées 0, 1, ..., n-1, moins les deux classes extrêmes, indicées -1 et n

inf : fixe la borne inférieure de la classe 0

pas : donne la taille d'un intervalle

long et haut : paramètres de mise en page en largeur et en hauteur

Il existe une classe d'histogrammes, et ceux qui sont créés lors d'une simulation sont chaînés dans une liste d'histogrammes histog utilisée pour l'édition (cf. annexe C et THA 80).

Exemple 9.22

Dans la maquette du chapitre 5, présentée à l'annexe 2, on demande des résultats détaillés sur les régions adm et phy et sur les ressources z et p par :

```
préciserégion (adm) ;
préciserégion (phy) ;
préciseressource (z) ;
préciseressource (p) ;
```

La construction d'un histogramme associé à la région adm est obtenue par :


```
historégion (adm, 65, 1, 24, 1, 40) ;
```

Les résultats correspondants ont été présentés précédemment. □

3.6 Obtention des résultats

Après leur constitution pendant l'exécution du modèle de simulation, les fichiers peuvent être exploités par des programmes.

Certains d'entre eux, qui nous ont servi à obtenir les résultats présentés précédemment (3.2.5), sont standard :

- le programme STATGEN permet d'obtenir les résultats synthétiques sur les volumes de données, les ressources et les régions contenues dans le fichier FGEN ;

- les programmes PREVOL, PREOCC, et PREREP utilisent la méthode des blocs (3.2.3) pour estimer les intervalles de confiance des moyennes des processus sous-jacents aux séries enregistrées dans les fichiers FVOL, FOCC et FREP.

On trouvera leur détail dans (THA 80).

Mais l'utilisateur a la possibilité d'exploiter les fichiers ainsi constitués comme il le désire et de faire des analyses de séries chronologiques plus fines. Certaines méthodes étudiées dans le cadre de ce projet (THA 80) pourront faire l'objet de nouveaux programmes standard.

CONCLUSION

"Walther :

Nicht Meister ! Nein !...

Sachs :

Was wollt Ihr von den Meistern mehr ? "

(R. Wagner, Les Maîtres-chanteurs de Nüremberg, acte 3, scène 5).

"Walther :

Pas maître ! Non !...

Sachs :

Que voulez-vous de plus des maîtres ? "

CONCLUSION

1. BILAN et PERSPECTIVES pour l'ANALYSE des SYSTEMES d'INFORMATION en ORGANISATION.

   1.1 Méthode.

   1.2 Modèle.

   1.3 Langage.

   1.4 Outils.

2. RELATIONS avec d'AUTRES DOMAINES.

   2.1 Conception des systèmes d'information.

   2.2 Relations avec la théorie des organisations.

CONCLUSION

A la fin de l'exposé de ce travail d'ordre méthodologique sur l'analyse en informatique d'organisation, il nous faut dresser un bilan et en tirer des perspectives d'avenir.

Nous le faisons d'abord pour notre domaine, puis nous élargissons nos vues en parlant des relations de cette approche avec d'autres études.

1 - BILAN et PERSPECTIVES pour l'ANALYSE des SYSTEMES d'INFORMATION en ORGANISATION

Notre objectif est d'apporter une aide à la description et à l'analyse de systèmes d'information existants ou en projet.

Pour ce faire, nous utilisons des modèles programmés (maquettes), plus ou moins détaillés, et différenciés selon le type d'analyse (ou évaluation) : structurale, qualitative ou quantitative.

Pour discuter de la réalisation ou non de l'objectif initial, nous examinons notre travail selon les éléments du quadruplet utilisé dans le groupe de travail INFORSID2 depuis quelques années : méthode, modèle, langage, outils (INF 76).

1.1 Méthode

- L'approche de ces questions par des maquettes est a priori très séduisante.

. Elle donne une vue globale d'un système plutôt que de privilégier tel ou tel aspect partiel.

. Elle permet d'inclure dans la même unité des aspects organisationnels connexes à l'informatique qui sont souvent négligés.

. Elle conduit à s'interroger non seulement sur la structure et le fonctionnement logique, mais aussi sur les performances d'un système.

. Elle donne la possibilité de conduire des expérimentations "en laboratoire" devant concourir à déterminer, sinon la meilleure, du moins une bonne architecture.

- Il ne faut cependant pas cacher que c'est une approche délicate.

. Les objectifs d'une telle étude ne sont pas toujours bien clairement explicités, et ce n'est qu'après quelques essais que l'on voit sur quels points concentrer les efforts.

. Le bon niveau de détail d'une maquette est difficile à déterminer et l'on doit souvent procéder par tâtonnements.

. Certains éléments d'un modèle sont délicats à obtenir : c'est le cas notamment de tout ce qui concerne les conditions et les rythmes de travail. Ce type d'études qui s'apparente à l'O.S.T (Organisation Scientifique du Travail) n'a pas bonne réputation parmi le personnel d'une entreprise et est à faire avec beaucoup de prudence. Les données que l'on recueille sont d'ailleurs soumises à quantité de facteurs, affectées d'une grande variance, et le plus souvent entachées d'erreurs.

. Nous avons vu que la validation d'une maquette est une opération difficile sur laquelle nous n'avons d'ailleurs pas suffisamment d'expérience.

. Enfin, malgré une apparente simplicité pour les non initiés, la résolution d'un modèle par simulation pose de nombreux problèmes, notamment pour une analyse statistique correcte des processus stochastiques qu'elle met en jeu.

- De plus cette approche est insuffisante pour diverses raisons.

. Dans une maquette d'un certain niveau, on gomme beaucoup de détails jugés embarassants. Or c'est justement l'accumulation de détails, de cas particuliers souvent infimes, qui fait toute la difficulté de l'analyse en informatique d'organisation où l'on en arrive souvent à dire : telle procédure automatisée est bonne dans x % des cas, sinon, on fait le travail "à la main" (plus ou moins). Il nous semble malgré tout que des maquettes doivent permettre de mettre en évidence de tels phénomènes

et aider à les formaliser convenablement, par des procédures secondaires bien définies à un certain niveau.

. Dans les maquettes, certains caractères qui font un système d'information ne sont pas pris en compte. Nous avons négligé les aspects psychologiques et sociologiques de la définition des procédures de traitement, dont on sait qu'ils peuvent jouer un rôle essentiel dans la réussite de l'exploitation. Les aspects économiques n'ont pas été directement pris en compte : ils doivent l'être par l'intérmédiaire des études sur l'architecture et les performances des systèmes, mais la liaison reste à faire.

- Il faudra à l'avenir se donner les moyens de ne pas aborder tous les problèmes à la fois et aboutir à une méthode de décomposition et de réduction des systèmes.

La décomposition peut être réalisée :

. structurellement : on a ici une superposition de processus asynchrones (logique indépendante du temps) et de phénomènes synchrones (avec des contraintes de temps) qu'il convient d'étudier séparément avant de les confondre dans les mêmes modèles ;

. horizontalement, par rapport à l'espace : on étudie alors chaque partie en tenant compte de ses interactions avec le reste du système modélisé globalement ;

. verticalement, selon différentes échelles de temps : on cherche le comportement global d'une partie du système modélisé à une échelle de temps assez fine ; on intègre ce résultat général dans un modèle du système pour une étude à échelle de temps de plus haut niveau.

Il nous semble possible de conduire des recherches en ce sens à partir des résultats mathématiques sur les réseaux de files d'attente (cf. théorème de NORTON et réseaux équivalents (CHA 75), méthodes de

décomposition (RIC 79)).

La réduction d'un système concerne :

. l'agrégation de ses données : celle-ci correspond souvent à un appauvrissement sémantique et à une réduction en volume ; des règles doivent être recherchées pour y procéder méthodiquement ;

sans doute les travaux sur les types abstraits de données (B. LISKOV, J.V. GUTTAG, C. PAIR, J.L. REMY...) doivent guider dans cette voie ;

. la modélisation, en stations indépendantes, de ressources réelles ; ici, ce sont les travaux sur la modélisation de systèmes (E. GELENBE, D. POTIER...) qui doivent aider.

1.2 Modèle

Le modèle que nous avons retenu est celui de processus

. coopérants par l'intermédiaire de données communes partagées ;

. concurrents pour l'obtention de ressources.

- La protection des données, la synchronisation des processus et le partage des ressources sont assurés par un mécanisme unique, avec quelques variantes : les moniteurs (au sens de P. BRINCH HANSEN et C.A.R. HOARE).

Cette position a été amplement justifiée dans le texte par un compromis entre la recherche d'un langage de haut niveau et celle de l'efficience.

Pour l'avenir, il faut s'attendre à un certain nombre de révisions allant dans le sens de plus de simplicité et d'élégance, grâce à des

mécanismes de niveau plus élévé. En ce sens, il faut porter la plus grande attention aux recherches récentes sur les processus séquentiels communicants (CSP) de C.A.R. HOARE (HOA 78), repris dans GREEN-ADA (HON 79a), et sur les processus distribués de P. BRINCH HANSEN (BRI 78), dont nous avons parlé au chapitre 4.

De la même manière, l'usage des idées de types abstraits nous semble être de nature à améliorer la structure et la compréhension des modèles.

- Pour ce qui est de l'évaluation fonctionnelle, il nous semble qu'un grand effort reste à faire du côté des preuves de programmes composés de processus coopérants : peut-être des modèles dérivés des réseaux de PETRI pourraient-ils aller dans ce sens. Mais cette question est loin de nous être spécifique : elle préoccupe également les concepteurs de systèmes d'exploitation.

- Pour l'évaluation du comportement dynamique, l'étude des réseaux de files d'attente doit être poursuivie, avec les méthodes de résolution exactes et approchées. Elle permettra d'épurer les maquettes destinées aux études de comportement et guidera leur validation.

- Enfin, une classification plus rigoureuse doit être faite des différents composants permettant de fabriquer des structures de données, processus et stations. Cette typologie doit tenir compte des objectifs et niveaux de détail fixés aux études.

1.3 Langage

Nous avons retenu SIMULA 67 en tant que langage de description de système et de simulation. Nous avons déjà longuement exposé ses avantages et inconvénients.

- Ce langage de haut niveau était révolutionnaire à sa sortie, grâce notamment au concept de classes hiérarchisées, qui lui conférait des qualités d'extension sans précédent. Il a su aussi s'adapter sous l'impulsion d'une active association d'utilisateurs et intégrer des notions nouvelles : protection, compilation séparée...

- L'implantation de SIMULA 67 sur IRIS 80 a été bien améliorée dans la dernière version de 1979, avec la possibilité d'interpréter ou de compiler les programmes. Mais dans les deux cas, l'exécution reste lente, avec aussi un temps prohibitif pour l'édition de liens quand on utilise le relieur standard après compilation.

- Aujourd'hui, les notions introduites dans SIMULA 67 ont été décantées et apparaissent modernisées dans de nombreux langages (voir chapitre 4 et (MEY 79)). On peut espérer dans l'avenir que certains donneront les mêmes facilités de description, avec plus de sécurité, de souplesse et d'efficacité : une version de GREEN-ADA avec quasi-parallélisme ?...

1.4 Outils

Nous avons présentés dans les chapitres précédents des outils de description et d'évaluation. La plupart sont réalisés et intégrés dans le système MAESTRO. D'autres pourront par la suite l'être facilement parce que nous bénéficions de l'extensibilité de SIMULA 67 et de sa possibilité de créer des catalogues.

- Bien qu'opérationnelle et d'usage assez facile pour ses concepteurs, la version actuelle du système MAESTRO est encore expérimentale et n'est pas à mettre entre toutes les mains. En effet, avant d'être transmissible, elle doit bénéficier :

. de certaines extensions et améliorations dont l'intérêt a été souligné dans les chapitres précédents ;

. d'une plus grande standardisation des composants ;

. d'une intégration plus poussée des différents outils ;

. enfin, et surtout, de nombreux "garde-fous" contre une programmation un peu hasardeuse, qui ont été délibérement sacrifiés dans cette version destinée au laboratoire.

- Dans le domaine des outils, on peut penser que des progrès viendront aussi des nouveaux concepts introduits pour la programmation parallèle. C'est ainsi que des essais sont faits à l'heure actuelle pour répartir la simulation de systèmes de processus séquentiels communicants du type C S P (HOA 79) sur des machines multiprocesseurs ou des réseaux de micro-ordinateurs (KAU 78, CHA 78), plutôt que sur un calculateur unique. Cette procédure pose des problèmes liés à la non-unicité du référentiel-temps un peu comparables à ceux que l'on rencontre dans les bases de données réparties (LEL 77), mais elle devrait à terme améliorer considérablement l'efficacité des simulations.

- De plus, il est bien certain que l'on devra éviter à terme que l'utilisateur du système ait à connaître SIMULA (ou son remplaçant). Cela signifie la définition d'outils standard paramétrés, dont l'assemblage pourra être exprimé dans un langage spécialisé, éventuellement de manière interactive. En ce sens, on doit suivre avec attention le développement de langages très synthétiques, cachant des outils efficaces, tels celui du progiciel QNAP, déjà cité (MER 78).

2 - RELATIONS avec d'AUTRES DOMAINES

Nous examinons successivement les possibilités entrouvertes par des maquettes dans deux domaines : celui de la conception des systèmes d'information et celui de l'organisation des entreprises.

2.1 Conception des systèmes d'information

- L'approche que nous avons présentée et qui sert à l'analyse générale de systèmes doit pouvoir être utilisée lors de la première phase de la conception d'un nouveau système. En effet, elle est envisageable au niveau d'une étude préalable, alors qu'il s'agit de "dimensionner" correctement l'architecture d'un système. Elle doit aider à mieux en apprécier les implications organisationnelles, humaines et financières.

La méthode est alors la suivante : après une étude d'existant, dont on trouve la démarche dans les ouvrages d'analyse (MAR 70, REI 71...), et à partir de l'idée que l'on se fait du nouveau système, on construit un modèle et l'on écrit une ou plusieurs maquettes. Celles-ci sont ensuite testées (chapitres 8 et 9) et modifiées jusqu'à ce qu'elles satisfassent les objectifs fixés.

Ces maquettes sont nécessairement des modèles simplifiés du nouveau système et la première phase doit être suivie d'une deuxième qui consiste à faire une conception plus détaillée du nouveau système. A ce niveau, on fait appel aux méthodes existantes ou encore à l'étude (chapitre 2), pour obtenir des spécifications complètes allant d'un niveau conceptuel à un niveau d'implantation physique. Mais on peut se demander si les maquettes que l'on a écrites dans la première phase ne pourraient pas, après quelques transformations mineures, être détaillées en d'autres

maquettes pour obtenir ces spécifications. Un tel procédé s'apparente de fait à la conception descendante ("top-down") d'un système, dont nous avons déjà parlé (chapitre 4). Peu à peu, les descriptions seraient "enrichies" et l'on songe ici aux possibilités de hiérarchisation des classes de SIMULA, ou à celles des clauses "utilise" de certains langages (CIVA, GREEN-ADA...).

Les maquettes peuvent être assez éloignées de la réalité, et le procédé à mettre en œuvre est sans doute plus complexe qu'un simple enrichissement. On sent bien pourtant que toutes les maquettes d'un système ont des liens entre elles, qu'elles pourraient notamment être issues d'une unique description de niveau élevé. Des études importantes sont à faire dans ce domaine, qui n'est en définitive rien d'autre que celui de la conception progressive d'un système en informatique, avec toutes les questions qu'elle pose...

En ce sens, notre travail est une contribution à ce vaste domaine d'étude.

- Pour s'y intégrer parfaitement, il nous semble que de nombreux progrès sont à faire dans la première phase de conception des maquettes elles-mêmes :

. avoir une expression plus statique :

de la conception des structures de données : en utilisant par exemple les spécifications statiques de types abstraits (GUT 78...) ;

du parallélisme et de la synchronisation : par l'usage d'expression de chemins (voir chapitre 4) ou de systèmes d'axiomes (PAI 78) ;

des algorithmes : par l'usage de méthodes comme la méthode déductive (PAI 79) ;

. bien séparer :

les parties asynchrones d'une maquette, qui doivent être testées pour elles-mêmes,

les parties synchrones, qui contraignent les autres et imposent le comportement dynamique.

Dans cet ordre d'idées, en plus des travaux dont nous avons déjà parlé, on peut citer une autre tentative de spécification très novatrice : celle de M.A. JACKSON (JACK 78) qui s'appuie sur CSP (HOA 78) pour considérer tout système d'information comme un ensemble de multiples processus communicants reflétant le mondre réel. Il rencontre des problèmes d'agrégation de processus qui rappellent les nôtres.

2.2 Relations avec la théorie des organisations

Cette étude se trouve à la frontière entre organisation et informatique : elle tente d'établir la jonction en considérant des systèmes globaux où les deux parties sont présentes. Elle souligne les interrelations entre différentes sortes d'activités dans une entreprise ou une administration : traitement d'information automatique, manuel et intellectuel, avec prise en compte éventuelle de processus décisionnels.

- Même si l'on doit se méfier des "serpents de mer", il nous semble que nous nous rapprochons d'un véritable langage d'analyse permettant d'exprimer globalement les projets communs en informatique et gestion.

- Enfin, il apparaît que les outils que nous utilisons peuvent servir à modéliser d'autres types de processus dans l'entreprise. On peut alors songer à des modèles globaux, où seraient considérés toute l'organisation et les flux qui la traversent. On rejoint ici les théories sur l'organisation (J. MELEZE (MEL 72), B. LUSSATO (LUS 72)...), ainsi que

les idées de J. W. FORRESTER (voir chapitre 3), mais avec des modèles différents, essentiellement par leur nature discrète, alors que les siens sont continus. Cette hypothèse de continuité est naturelle pour certains types de flux (exemple : matières combustibles) et elle peut être maintenue dans des modèles mixtes discrets-continus. On sait en effet que des langages permettent d'intégrer les deux aspects (MOD 77).

- Alors, l'ultime objectif apparaît comme la formulation d'un cadre conceptuel, la définition d'un langage général et la fabrication d'outils pour aider à décrire et évaluer la gestion d'une organisation. Mais, pour ne pas tomber dans l'utopie, l'approche de ce grand dessein doit s'appuyer sur des études précises avec des cas concrets mettant en évidence plus modestement :

. l'articulation du système d'information d'une entreprise avec le système décisionnel (MEL 72, LUS 72, LEM 73...) ;

. son couplage avec les systèmes régissant les autres "flux" ; pour reprendre la typologie de J.W. FORRESTER (FOR 61, FOR 68) : matières (gestion de la production), matériels (gestion des équipements), capitaux (gestion financière), personnes (gestion du personnel).

On doit considérer plusieurs niveaux et échelles de temps, comme pour les systèmes d'information (voir précédemment). De grandes difficultés sont bien sûr liées à la taille et à la diversité de ces systèmes. Mais le principal obstacle est certainement le caractère très ouvert et très évolutif des organisations, surtout en période d'instabilité économique et politique.

Ainsi, la conception, l'écriture et l'exploitation de maquettes de systèmes d'information paraissent s'intégrer dans les activités normales de formalisation et d'évaluation d'une organisation.

Sans doute n'obtient-on pas facilement une maquette conforme à tous ses désirs : la démarche est délicate et le processus d'élaboration itératif. Mais les avantages que l'on doit pouvoir retirer de sa définition, de sa construction et de son utilisation nous semblent à terme devoir l'emporter sur les inconvénients.

Le domaine que nous avons contribué à explorer est très vaste, riche et divers. Il est lié à de nombreuses questions ouvertes en informatique et en gestion. On peut prévoir que la poursuite de son investigation et l'approfondissement des notions présentées ici mobiliseront encore bien des efforts.

ANNEXES

A - MAQUETTE de MODELE RELATIONNEL ABSTRAIT.

B - EXEMPLE de PROGRAMME-MAQUETTE.

C - TEXTE COMPLET du PROLOGUE standard : la classe MAESTRO.

BIBLIOGRAPHIE

ANNEXE A

MAQUETTE de MODELE RELATIONNEL

ABSTRAIT programmée en SIMULA 67

EXEMPLE d'UTILISATION

Annexe A - MAQUETTE de MODELE RELATIONNEL ABSTRAIT

1 - **MODELE ABSTRAIT** (voir chapitre 6 (1.1.2.b)).

```
BEGIN
SIMSET BEGIN
LINK CLASS OBJET;
  VIRTUAL: TEXT PROCEDURE CLE; NULL;
BOOLEAN PROCEDURE EGAL(X,Y); REF(OBJET) X,Y;
  EGAL := X.CLE = Y.CLE;
CLASS RELATION;
  VIRTUAL: REF(OBJET) PROCEDURE ELEMENT,PREMIER,SUIVANT;
           PROCEDURE AJOUTER,SUPPRIMER,VIDER;
BEGIN
  INTEGER CARD ;
END;
LINK CLASS DR(B); BOOLEAN B; NULL;
PROCEDURE PROJ(R,S,AFF); NAME S;
  REF(RELATION) R,S;  REF(OBJET) PROCEDURE AFF;
BEGIN
REF(OBJET) X,Y;
  S.VIDER;
  X:- R.PREMIER;
  WHILE X=/= NONE DO
  BEGIN
    BOOLEAN EX;
    EX := FALSE;
    Y :- S.PREMIER;
    WHILE Y=/= NONE AND NOT EX DO
    BEGIN
      IF EGAL(X,Y) THEN EX:= TRUE;
      Y:- S.SUIVANT(Y);
    END;
    IF NOT EX THEN S.AJOUTER(AFF(X));
    X:- R.SUIVANT(X);
  END
END;
PROCEDURE JOIN(R,S,T,CONC); NAME T;
  REF(RELATION) R,S,T; REF(OBJET) PROCEDURE CONC;
BEGIN
  REF(OBJET) X,Y,Z;
  T.VIDER;
  X:- R.PREMIER;
  WHILE X=/= NONE DO
  BEGIN
    Y:- S.PREMIER;
    WHILE Y=/= NONE DO
    BEGIN
      Z:- CONC(X,Y);
      IF Z=/= NONE THEN T.AJOUTER(Z);
      Y :- S.SUIVANT(Y)
    END;
    X :- R.SUIVANT(X)
  END
END;
PROCEDURE DIVIDE(R,S,T,COUP); NAME T;
```

```
  REF(RELATION) R,S,T: PROCEDURE COUP;
BEGIN
  REF(OBJET) X;
  REF(HEAD) LDR;
  REF(DR) D;
  INTEGER I;
  BOOLEAN PROCEDURE BRECH(X2); REF(OBJET) X2;
  BEGIN
    REF(OBJET) Y;
    Y:- S.PREMIER;
    WHILE (IF Y == NONE THEN FALSE ELSE NOT EGAL (Y,X2))
    DO Y:- S.SUIVANT(Y);
    IF Y ==  NONE THEN BRECH := FALSE ELSE BRECH := TRUE
  END;
  LDR :- NEW HEAD;
  FOR I := 1 STEP 1 UNTIL     R.CARD  DO NEW DR(FALSE).INTO(LDR);
  X:- R.PREMIER;
  D:- LDR.FIRST;
  WHILE  X =/= NONE DO
  BEGIN
    IF NOT D.B
    THEN BEGIN
          INTEGER N;
          REF(OBJET) Z,X1,X2,U;  REF(DR) D1;
          COUP(X,Z,X2);
          U :- X;
          D1 :- D;
          WHILE U =/= NONE DO
          BEGIN
            IF NOT D1.B THEN
            BEGIN
            COUP(U,X1,X2);
            IF EGAL(X1,Z)
            THEN BEGIN  D1.B := TRUE;
                        IF BRECH(X2) THEN N:=N+1
                 END;
            END;
            U :- R.SUIVANT(U);
            D1 :- D1.SUC
          END;
          IF N = S.CARD THEN  T.AJOUTER(Z)
        END;
    D :- D.SUC;
    X :- R.SUIVANT(X)
  END
END;$DIVIDE$
```


2 - MODELE CONCRET (chapitre 6 2.1.2. c)

```
RELATION CLASS LISTE;
BEGIN
  REF(HEAD) L;
  REF(OBJET) PROCEDURE ELEMENT(C); TEXT C;
  BEGIN
    REF(OBJET) X;
    X :- L.FIRST;
    WHILE (IF X == NONE THEN FALSE ELSE X.CLE /= C)
    DO X :- X.SUC;
    ELEMENT :- X
  END;
  REF(OBJET) PROCEDURE PREMIER; PREMIER:- L.FIRST;
  REF(OBJET) PROCEDURE SUIVANT(X); REF(OBJET) X; SUIVANT:- X.SUC;
  PROCEDURE AJOUTER(X); REF(OBJET) X;
  BEGIN
    X.INTO(L);
    CARD:= CARD+1
  END;
  PROCEDURE SUPPRIMER(C); TEXT C;
  BEGIN
    ELEMENT(C).OUT;
    CARD:= CARD-1;
  END;
PROCEDURE MODIFIER(X); REF(OBJET) X;
  BEGIN
    SUPPRIMER(X.CLE);
    AJOUTER(X)
  END;
  PROCEDURE VIDER; L.CLEAR;
  L:- NEW HEAD
END;
PROCEDURE IMPRIMER(R,IMP); REF(RELATION) R; PROCEDURE IMP;
BEGIN
  REF(OBJET) X;
  X:- R.PREMIER;
  WHILE X =/= NONE DO
  BEGIN
    IMP(X); OUTIMAGE;
    X:- R.SUIVANT(X)
  END
END;
```

3 - EXEMPLE d'UTILISATION (chapitre 6 (2.1.2. d)

```
OBJET CLASS SUPPLIER(SD,SNAME,STATUS,CITY);
  TEXT SD,SNAME,CITY; INTEGER STATUS;
BEGIN
TEXT PROCEDURE CLE ; CLE :- SD;
END;
OBJET CLASS CDE(SD,PD,QTY);
  TEXT SD,PD; INTEGER QTY;
BEGIN
TEXT PROCEDURE CLE;
BEGIN
  TEXT U;
  U :- BLANKS(4);
  U.PUTTEXT(SD);
  U.PUTTEXT(PD);
  CLE :- U
END;
END;
OBJET CLASS VILLE(CITY); TEXT CITY;
BEGIN
  TEXT PROCEDURE CLE; CLE:- CITY;
END;
OBJET CLASS SCDE(CITY,STATUS,SNAME,SD,PD,QTY);
  TEXT CITY,SNAME,SD,PD; INTEGER QTY,STATUS;
BEGIN
TEXT PROCEDURE CLE;
BEGIN
  TEXT U;
  U :- BLANKS(4);
  U.PUTTEXT(SD);
  U.PUTTEXT(PD);
  CLE :- U;
END;
END;
OBJET CLASS PROD(PD); TEXT PD;
BEGIN
TEXT PROCEDURE CLE; CLE :- PD;
END;
OBJET CLASS CDEK(SD,QTY);TEXT SD; INTEGER QTY;
BEGIN
TEXT PROCEDURE CLE;
BEGIN
  TEXT U;
  U :- BLANKS(3);
  U.PUTTEXT(SD);
  U.SUB(3,1).PUTINT(QTY);
  CLE :- U
END;
END;
```



```
REF(OBJET) PROCEDURE AFF(X); REF(SUPPLIER) X;
  AFF:- NEW VILLE(X.CITY);
REF(OBJET) PROCEDURE CONC(X,Y); REF(SUPPLIER) X; REF(CDE) Y;
  IF Y.SD = Y.SD
  THEN CONC :- NEW SCDE(X.CITY,X.STATUS,X.SNAME,X.SD,Y.PD,Y.QTY)
  ELSE CONC :- NONE;
PROCEDURE COUP(X,Y,Z); NAME Y,Z;
  REF(CDE) Y; REF(CDEK) Y; REF(PROD) Z;
BEGIN
  Y :- NEW CDEK(X.SD,X.QTY);
  Z :- NEW PROD(X.PD)
END;
PROCEDURE IMP1(X); REF(SUPPLIER) X;
INSPECT X DO BEGIN
  OUTTEXT(SD); OUTTEXT(SNAME); OUTINT(STATUS,2); OUTTEXT(CITY)
END;
PROCEDURE IMP2(X); REF(CDE) X;
INSPECT X DO BEGIN
  OUTTEXT(SD); OUTTEXT(PD); OUTINT(QTY,1);
END;
PROCEDURE IMP3(X); REF(VILLE) X; OUTTEXT(X.CITY);
PROCEDURE IMP4(X); REF(SCDE) X;
INSPECT X DO BEGIN
  OUTTEXT(CITY); OUTINT(STATUS,2); OUTTEXT(SNAME);
  OUTTEXT(SD); OUTTEXT(PD); OUTINT(QTY,1)
END;
PROCEDURE IMP5(X);REF(CDEK) X;
BEGIN  OUTTEXT(X.SD); OUTINT(X.QTY,1) END;
REF(LISTE) S,SP,SVILLE,SSP;
REF(LISTE) K ,SPK ;
S       :- NEW LISTE;
SP      :- NEW LISTE;
SVILLE  :- NEW LISTE;

SSP     :- NEW LISTE;
K:- NEW LISTE;
 SPK:- NEW LISTE;
S.AJOUTER(NEW SUPPLIER('S1','SMITH',20,'LONDON'));
S.AJOUTER(NEW SUPPLIER('S2','JONES',10,'PARIS '));
S.AJOUTER(NEW SUPPLIER('S3','BLAKE',30,'PARIS '));
IMPRIMER(S,IMP1);OUTIMAGE;
SP.AJOUTER(NEW CDE('S1','P1',3));
SP.AJOUTER(NEW CDE('S1','P2',2));
SP.AJOUTER(NEW CDE('S2','P1',5));
IMPRIMER(SP,IMP2); OUTIMAGE;
PROJ(S,SVILLE,AFF);
IMPRIMER(SVILLE,IMP3); OUTIMAGE;
JOIN(S,SP,SSP,CONC);
IMPRIMER(SSP,IMP4); OUTIMAGE;
K.AJOUTER(NEW PROD('P1'));
DIVIDE(SP,K,SPK,COUP);
IMPRIMER(SPK,IMP5);
END
END.
```

4 - RESULTATS

```
S1SMITH20LONDON
S2JONES10PARIS
S3BLAKE30PARIS

S1P13
S1P22
S2P15

LONDON
PARIS
PARIS

LONDON20SMITHS1P13
LONDON20SMITHS1P22
PARIS 10JONESS2P15

S13
S25
EXECUTION TERMINEE
```

ANNEXE B

EXEMPLE de PROGRAMME-MAQUETTE

utilisant la CLASSE MAESTRO -

GESTION des COMMANDES-CLIENTS

d'une P M E

Annexe B - EXEMPLE de PROGRAMME-MAQUETTE

```
!V7SIMULA SI,INTR
BEGIN
    $*************************************************
    *********       MODELE REDUIT          *********
    *****************************************(*******$
    $*********
                    UTILISATION DE LA CLASSE MAESTRO
    *********$
    MAESTRO BEGIN
      TRANSACTION CLASS BC (DAT,TYPE,NL,NLL) ; REAL DAT ; CHARACTER TYPE ;
      INTEGER NL,NLL ; NULL ;
      INTEGER PROCEDURE NORMALE (A,B,C) ; VALUE A,B,C ;
      REAL A,B ;
      INTEGER C ;
      BEGIN
            REAL NOR ;
            NOR := NORMAL (A,B,U) + 0.5 ;
            NORMALE := IMAX ( ENTIER (NOR) , C) ;
      END ; $ NORMALE $
      REAL PROCEDURE REELNORMALE (A,B,C) ; VALUE A,B,C ;
      REAL A,B,C ;
      BEGIN
            REAL NOR ;
            NOR := NORMAL (A,B,U) ;
            IF NOR < 0 THEN REELNORMALE := 0 ELSE REELNORMALE := NOR ;
      END ; $ REELNORMALE $
      INTEGER PROCEDURE BINOMIALE (NB,P) ; VALUE NB , P ;
      INTEGER NB ; REAL P ;
      BEGIN
            INTEGER K,PR;
            PR := 0 ;
            FOR K := 1 STEP 1 UNTIL NB DO
                IF DRAW (P,U) THEN PR := PR + 1 ;
            BINOMIALE := PR ;
      END ; $BINOMIALE $
      TFILE CLASS CCOMMANDES ;
      BEGIN
            PROCEDURE MODIFIERCOM (COM,QTELIVREE);VALUE QTELIVREE;
            REF (BC) COM ;
            INTEGER QTELIVREE ;
            BEGIN
                  THIS MONITOR.ENTER ;
                  COM.NL := COM.NL - QTELIVREE ;
                  COM.NLL := QTELIVREE ;
                  THIS MONITOR.LEAVE ;
                  MODIFIER (COM,COM.NL) ;
            END ; $ MODIFIERCOM $
            PROCEDURE PROPOSERLIV (COM,QTEPROP) ; VALUE QTEPROP ;
            REF (BC) COM ;
            INTEGER QTEPROP ;
```

```
      BEGIN
            THIS MONITOR.ENTER ;
            COM.NLL := QTEPROP ;
            THIS MONITOR.LEAVE ;
      END ; $ PROPOSERLIV $
END ; $ CCOMMANDES $
PROCESSUS CLASS GENERATEUR ;
BEGIN
      WHILE TRUE DO
      BEGIN
            INTEGER I,NBC;   $ NOMBRE DE COMMANDES $
            REF (MESSAGE) MESS ;
            ACTIVATION5.ACQUERIR (MESS) ;
            MESS :- NONE ;
            NBC := NORMALE (10,5,1) ;
            FOR I := 1 STEP 1 UNTIL NBC DO
            BEGIN
                  REF (BC) B;
                  INTEGER NBL ;
                  NBL := NORMALE (7,10,1) ;
                  B :- NEW BC (NUMEROTATION,NBL,TIME,"E",NBL,0) ;
                  COURRIER.ENTRER (B) ;
            END ;
      HOLD (0)
      END ;
END ; $ GENERATEUR $
PROCESSUS CLASS GENETEL ;
BEGIN
      WHILE TRUE DO
      BEGIN
            CHARACTER C ;
            INTEGER NLL ;        $ NOMBRE DE LIGNES $
            REF (BC) B ;
            IF Y.HEURETRAVAIL
            THEN BEGIN
            IF DRAW (8/18,U) THEN C := "T"
                             ELSE C := "D" ;
                  NLL := NORMALE (9,2,1) ;
                  B :- NEW BC (NUMERO,NLL,TIME,C,NLL,0) ;
                  BOITE.ENVOYER (B)
            END ;
            HOLD (NEGEXP (18/7,U)) ;
      END ;
END ; $ GENETEL $
PROCESSUS CLASS ENREGISTREMENT ;
BEGIN
      WHILE TRUE DO
      BEGIN
            REF (HEAD) L ;   $ LISTE INTERMEDIAIRE $
            REF (MESSAGE) MESS ;
            REF ( BC ) B ;
            REAL DUREE ;
            TEXT LIB ;
            LIB :- 'ENREGISTREMENT1' ;
            ACTIVATION1.ACQUERIR (MESS) ;
            MESS :- NONE ;
            L :- NEW HEAD ;
            WHILE COURRIER.LONGUEUR > 0 DO
```



```
            BEGIN
                  B :- COURRIER.SORTIR ;
                  X.ACQUERIR ;
                  DUREE := 0.013 + ( B.NL * 0.006 ) ;
                  HOLD (DUREE) ;
                  X.LIBERER ;
                  B.INTO (L) ;
            END ;
            LIB :- 'ENREGISTREMENT2' ;
            WHILE L.CARDINAL > 0 DO
            BEGIN
                  B :- L.FIRST QUA BC ;
                  B.OUT ;
                  X.ACQUERIR ;
                  HOLD (0.006 + 0.006 * B.NL) ;
                  ORD.ACQUERIR ;
                  HOLD (0.001 + 0.0005 * B.NL) ;
                  ORD.LIBERER ;
                  X.LIBERER ;
                  COMMANDES.ENTRER (B) ;
                                                ADM.ENTRER (B) ;
            END ;
      END
END ; $ ENREGISTREMENT $
PROCESSUS CLASS SAISIE ;
BEGIN
      WHILE TRUE DO
      BEGIN
            REF (MESSAGE) MESS ;
            REAL DUREE ;
            BOITE.ACQUERIR (MESS) ;
            IF MESS QUA BC.TYPE = "D"
            THEN DUREE := 0.04
            ELSE BEGIN
                        DUREE := 0.04 + 0.008 + (MESS QUA BC.NL * 0.007) ;
                        MESS QUA BC.NO := NUMEROTATION ;
                  END ;
            Y.ACQUERIR ;
            HOLD (4 * DUREE / 5) ;
            ORD.ACQUERIR ;
            HOLD (DUREE / 5) ;
            ORD.LIBERER ;
            Y.LIBERER ;
            IF MESS QUA BC.TYPE = "T"
            THEN BEGIN
                        COMMANDES.ENTRER (MESS QUA BC) ;
                                          ADM.ENTRER(MESS QUA BC);
                  END ;
            MESS :- NONE ;
      END
END ; $ SAISIE $
```

```
PROCESSUS CLASS PREPARATION ;
BEGIN
      WHILE TRUE DO
      BEGIN
            REF (BC) B,B1 ;
            REAL T1,T2,T3 ;
            B :- ORDRES.SORTIR ;
            P.ACQUERIR ;
            T1 := B.NLL *0.032 ;
            T2 := B.NLL * 0.005 ;
            T3 := B.NLL * 0.01 ;
            HOLD (REELNORMALE (T1,T2,T3)) ;
            P.LIBERER ;
            BONS1.ENTRER (B) ;
            INSPECT B DO B1 :- NEW BC (NO,PDS,DAT,TYPE,NL,NLL);
            BONS2.ENTRER (B1) ;
                                                PHY.SORTIR (B);
       END ;
END ; $ PREPARATION $
PROCESSUS CLASS VIDEUR (BONS,ACT) ; REF (TFILE) BONS ;
REF (ACTMONITOR) ACT ;
BEGIN
      WHILE TRUE DO
      BEGIN
            REF (MESSAGE) MESS ;
            ACT.ACQUERIR (MESS) ;
            BONS.DETRUIRE ;
      END;
END ; $ VIDEUR $
PROCESSUS CLASS DECISION ;
BEGIN
      WHILE TRUE DO
      BEGIN
            TEXT LIB ;
            REF (MESSAGE) MESS ;
            REF (BC) B,B1 ;
            INTEGER I,NC,NLLT ; $ NC NB COMMANDES  NLLT NB TOT LIGNES $
            REAL LIV ; $ Z LIGNES A LIVRER $
            LIB :- 'DECISION1' ;   $ EDITION PROPOSITIONS LIVRAISON $
            ACTIVATION2.ACQUERIR (MESS) ;
            ORD.ACQUERIR ;
            NC := COMMANDES.LONGUEUR ;
            NLA := COMMANDES.VOLUME ;  $  NB LIGNES EN ATTENTE $
            NLLT := 0 ; $ NB LIGNES A LIVRER $
            IF NLA> 0 THEN LIV := (NLA - NLAP)/NLA ELSE LIV :=  0 ;
            B :- COMMANDES.PREMIER ;
            FOR I := 1 STEP 1 UNTIL NC DO
            BEGIN
                  INTEGER QTE;
                  QTE := BINOMIALE (B.NL,LIV) ;
                  NLLT := NLLT + QTE ;
                  COMMANDES.PROPOSERLIV (B,QTE) ;
                  B :- COMMANDES.SUIVANT (B) ;
            END ;
```



```
            HOLD (0.25) ;
            ORD.LIBERER ;
            LIB :- 'DECISION2' ;   $PRISE DE DECISION $
            Y.ACQUERIR ;
            HOLD (NC * 0.007) ;
            Y.LIBERER ;
            LIB :- 'DECISION3' ;   $ SAISIE DES MODIFICATIONS $
            Z.ACQUERIR ;
            HOLD (NLA * 0.004 /10) ;
            ORD.ACQUERIR ;
            HOLD (NLA * 0.002 / 10) ;
            ORD.LIBERER ;
            Z.LIBERER ;
            LIB :- 'DECISION4' ;
            $ EDITION DES BL ,MAJ DES COMMANDES $
            ORD.ACQUERIR ;
            B :- COMMANDES.PREMIER ;
            FOR I := 1 STEP 1 UNTIL NC DO
            IF B.NLL > 0 THEN
            BEGIN      $ LIGNES A LIVRER $
                  IF B.NL = B.NLL
                  THEN BEGIN     $ COMMANDE SOLDEE $
                              B1 :- B ;
                              B :- COMMANDES.SUIVANT (B1);
                              COMMANDES.ENLEVER (B1) ;
                              IF B1.DAT > 0 THEN
                              BEGIN
                                    REAL TPS ;
                                    TPS := TIME - B1.DAT ;
                                    ST.AJOU (TPS) ;
                              END ;
                                                ADM.SORTIR(B1) ;
                        END
                  ELSE BEGIN     $ LIVRAISON PARTIELLE $
                              INSPECT B DO
                              B1 :- NEW BC (NO,NLL,DAT,TYPE,NL,NLL);
                              COMMANDES.MODIFIERCOM(B,B.NLL);
                              B :- COMMANDES.SUIVANT (B) ;
                        END ;
                  ORDRES.ENTRER (B1) ;
                                                PHY.ENTRER (B1) ;
            END ;
            NLAP := NLA - NLLT ; $ NB LIGNES EN ATTENTE $
            HOLD (0.25) ;
            ORD.LIBERER ;
      END ;
END ; $ CDECISION $
INTEGER PROCEDURE NUMEROTATION ;
BEGIN
      ORDRE := ORDRE + 1 ;
      NUMEROTATION := ORDRE
END ; $ NUMEROTATION $
INTEGER PROCEDURE NUMERO ;
BEGIN
      NUMAPP := NUMAPP + 1 ;
      NUMERO := NUMAPP
END ; $ NUMERO $
```

```
INTEGER LIMCOMMANDES,NLA,NLAP,
        ORDRE , I , NUMAPP ;
REF (CCOMMANDES) COMMANDES ;
REF (TFILE) COURRIER,ORDRES,BONS1,BONS2 ;
REF (TREGION) ADM,PHY;
REF (CALRAMINF) ORD ;
REF (CALRC) X,Z,P ;
REF (CALRCPP) Y ;
REF (HISTOGRAMME) ST ;
REF (ACTMONITOR) ACTIVATION1 , ACTIVATION2 , ACTIVATION3,
                 ACTIVATION4 , ACTIVATION5 , BOITE ;
REF (ECHEANCIER) ECHE ;
LIMCOMMANDES := 25 ;
ORDRE := 0 ;
NUMAPP := 0 ;
ECHE :- NEW ECHEANCIER (24,0) ;
ADM :- NEW TREGION ('ADMINISTRATIF') ;
PHY :- NEW TREGION ('PHYSIQUE') ;
COMMANDES :- NEW CCOMMANDES ('COMMANDES',TRUE,TRUE,24,20) ;
COURRIER :- NEW TFILE ('COURRIER',TRUE,TRUE,24,9.50) ;
ORDRES :- NEW TFILE ('ORDRES',TRUE,FALSE,24,20) ;
BONS1 :- NEW TFILE ('BONS1',TRUE,TRUE,24,20) ;
BONS2 :- NEW TFILE ('BONS2',FALSE,FALSE,24,20) ;
ORD :- NEW CALRAMINF ('ORDINATEUR',FALSE,TRUE,0,0) ;
X :- NEW CALRC ('X',TRUE,TRUE,24,0) ;
Y :- NEW CALRCPP ('Y',TRUE,TRUE,24,0) ;
Z :- NEW CALRC ('Z',TRUE,TRUE,24,0) ;
P :- NEW CALRC ('P',TRUE,FALSE,24,0) ;
ACTIVATION5 :- NEW ACTMONITOR ('GENERATEUR',3) ;
ACTIVATION1 :- NEW ACTMONITOR ('ENREGISTREMENT',3) ;
ACTIVATION2 :- NEW ACTMONITOR ('DECISION',3) ;
ACTIVATION3 :- NEW ACTMONITOR ('VIDEUR1',3) ;
ACTIVATION4 :- NEW ACTMONITOR ('VIDEUR2',3) ;
BOITE :- NEW ACTMONITOR ('BOITE',1) ;
ECHE.MODIFECH (ACTIVATION1) ;
ECHE.MODIFECH (ACTIVATION2) ;
ECHE.MODIFECH (ACTIVATION3) ;
ECHE.MODIFECH (ACTIVATION4) ;
ECHE.MODIFECH (ACTIVATION5) ;
FOR I := 1 STEP 1 UNTIL LIMCOMMANDES DO
BEGIN
      REF (BC) B ;
      B :- NEW BC (NUMEROTATION,2,TIME,"A",2,2) ;
                                                  ADM.ENTRER(B);
```



```
      COMMANDES.ENTRER(B) ;
    END ;
    NLAP := COMMANDES.VOLUME ;
    ACTIVATE NEW GENERATEUR ('GENERATEUR',1,0) QUA PROCESS ;
    ACTIVATE NEW GENETEL ('GENETEL',1,0) QUA PROCESS ;
    ACTIVATE NEW ENREGISTREMENT ('ENREGISTREMENT',1,0) QUA PROCESS ;
    ACTIVATE NEW SAISIE ('SAISIE',1,0) QUA PROCESS ;
    ACTIVATE NEW DECISION ('DECISION',1,0) QUA PROCESS ;
    ACTIVATE NEW PREPARATION ('PREPARATION',1,0) QUA PROCESS ;
    ACTIVATE NEW VIDEUR ('VIDEUR1',1,0,BONS1,ACTIVATION3) QUA PROCESS ;
    ACTIVATE NEW VIDEUR ('VIDEUR2',1,0,BONS2,ACTIVATION4) QUA PROCESS ;
    ST :- NEW HISTOGRAMME ('SYSTEME',65,1,24,1,25) ;
    SIMUL (120,TRUE,FALSE,FALSE,FALSE) ;
    HISTOREGION (ADM,65,1,24,1,40) ;
    PRECISEREGION (ADM) ;
    PRECISEREGION (PHY) ;
    PRECISERESSOURCE (Z) ;
    PRECISERESSOURCE (P) ;
    SIMUL (1440,TRUE,TRUE,TRUE,TRUE) ;
  END $ BLOC MAESTRO $
  END  $  PROGRAMME  $ #
!EOD
 0 10 0 0 12 0 0 14 0 0 15 0**
 0 8 0 0 12 0 0 14 0 0 18 0**
 0 15 30 0 17 0**
 0 8 0 0 12 0 0 14 0 0 18 0**
 0 10 0*
 0 15 0*
 0 21 0*
 0 21 0*
 0 9 0*
!EOD
EOF HIT
*
```

ANNEXE C

TEXTE COMPLET du PROLOGUE standard :

la CLASSE MAESTRO

Cette version présente quelques différences minimes avec les spécifications données aux chapitres 6 et 8. Elle est la dernière qui ait été cataloguée et elle a servi d'environnement à l'écriture et à l'exploitation de la maquette du chapitre 5.

Annexe C - TEXTE COMPLET du PROLOGUE standard :

la CLASSE MAESTRO

```
/PROLOGUE
  BEGIN
   SIMULATION CLASS MAESTRO ;
     BEGIN
      LINK CLASS ELEMENT (PR,P,DT) ; REF (PROCESS) PR ; REAL P,DT ; NULL ;
      LINK CLASS MONITOR (NOM) ; TEXT NOM ;
      BEGIN
            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
      END ; $ MONITOR $
      CLASS CONDITION (M) ; REF (MONITOR) M ;
      BEGIN
            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 CURRENT ;
                              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 CURRENT ;
                        M.ENTERQ.FIRST.OUT
                  END
                  ELSE M.BUSY := FALSE ;
      END ; $ SIGLEAVE $
      BOOLEAN PROCEDURE EMPTY ;
      EMPTY := WAITQ.EMPTY ;
      WAITQ :- NEW HEAD
END ; $ CONDITION $
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 CURRENT;
                        M.URGENTQ.FIRST.OUT
                        END
                  ELSE IF NOT M.ENTERQ.EMPTY
                        THEN BEGIN
               ACTIVATE M.ENTERQ.FIRST QUA PROCESS AFTER CURRENT;
                           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
END ; $ CONDITIONP $
MONITOR CLASS ACTMONITOR (TYPE) ; INTEGER TYPE ;
BEGIN
      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 DEBLOCAGE.EMPTY AND ( TYPE = 1 OR ( TYPE = 3 AND NOT (
            ACTQ.EMPTY ) ) )
            THEN BEGIN
            ECRI (TIME) ;
                  OUTTEXT (' SIGNAL SUPPRIME ...') ;
                  OUTTEXT (NOM) ;
                  OUTIMAGE
                  END
            ELSE D.INTO (ACTQ) ;
            DEBLOCAGE.SIGLEAVE
      END ; $  ENVOYER  $
```

```
      ACTQ :- NEW HEAD ;
      DEBLOCAGE :- NEW CONDITION (THIS MONITOR)
END ; $  ACTMONITOR  $
CLASS ECHEANCIER (PER,HDP) ; INTEGER PER,HDP ;
BEGIN
      REF (HEAD) ECHQ ;
      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 $
```



```
MONITOR CLASS RESSOURCE ;
BEGIN
      BOOLEAN PRECISION ;
      REAL MDISPO,MDEB,TPSDISPO,TPSUTIL,TPSINUTIL,
           ACCES,RPMAXDISPO,RPMAXUTIL;
      REAL ARRAY UTIL (0:10) ;
      INTEGER RSORT,NBUTIL,NBMAXUTIL,TYPE,NORES,IND ;
      PROCEDURE STATACQUERIR (N) ; REAL N ;
      BEGIN
            STATMAJ (N) ;
            IF RPMAXUTIL < ACCES THEN RPMAXUTIL := ACCES ;
      END ;  $  STATACQUERIR  $
      PROCEDURE STATLIBERER (N) ; REAL N ;
      BEGIN
            STATMAJ (-N) ;
      END ;  $  STATLIBERER  $
      PROCEDURE STATMAJ (N) ;REAL N ;
      BEGIN
            REAL HMES ;
            IND := MIN (ENTIER (ACCES),10) ;
            UTIL (IND) := UTIL (IND) + TIME -MDEB ;
            TPSUTIL := TPSUTIL + (TIME - MDEB)*ACCES ;
            MDEB := TIME ;
            ACCES := ACCES + N ;
            HMES := TIME - TPSINUTIL ;
            IF PRECISION THEN ECRIFOCC (NORES,TYPE,ACCES,HMES) ;
      END ; $ STATMAJ $
      PROCEDURE STATUTILISATEUR (NB) ;  INTEGER NB ;
      BEGIN
            NBUTIL := NBUTIL + NB ;
            IF NBUTIL > NBMAXUTIL THEN NBMAXUTIL := NBUTIL ;
      END ;  $  STATUTILISATEUR  $
      PROCEDURE STATSERVIS ;
       RSORT := RSORT + 1 ;
      PROCEDURE STATINIT (N,TYP) ;  REAL N ; INTEGER TYP ;
      BEGIN
            INTEGER NBAC ;
            NBAC := MIN (ENTIER (N),10) ;
            TPSUTIL := TPSDISPO := TPSINUTIL := 0 ;
            RPMAXDISPO := N ;
            MDISPO := TIME ;
            ACCES := RPMAXUTIL := 0 ;
            NBUTIL := NBMAXUTIL := 0 ;
            TYPE := TYP ;
            FOR IND:=0 STEP 1 UNTIL NBAC DO UTIL(IND):=0;
            NORES := NUMRES ;
            PRECISION := FALSE ;
      END ;  $ STATINIT  $
      PROCEDURE STATREINIT ;
      BEGIN
            PRECISION := FALSE ;
            RSORT := 0 ;
            NBMAXUTIL := NBUTIL ;
            RPMAXUTIL := ACCES ;
            TPSUTIL := TPSDISPO := TPSINUTIL := 0 ;
            MDEB := MDISPO := TIME ;
            FOR IND := 1 STEP 1 UNTIL MIN (10,RPMAXDISPO) DO
                UTIL (IND ) := 0 ;
      END ; $STATREINIT $
      NEW REFRES (THIS RESSOURCE).INTO (RESSOURCEQ);
END ;  $  RESSOURCE  $
```

```
RESSOURCE CLASS CALENDRIER (B,TERMINE,PER,JDP) ; BOOLEAN B,TERMINE ;
INTEGER JDP , PER ;
BEGIN
      REF (HEAD) NOTICEQ,NOTCALQ ;
      REF (CONDITION) TRAVAIL ;
      INTEGER NOPER ;
      BOOLEAN HEURETRAVAIL ;
      PROCEDURE CALCONSULTER ;
      BEGIN
            IF NOT HEURETRAVAIL THEN TRAVAIL.CWAIT ;
            NEW NOTICE (CURRENT,-1).INTO (NOTICEQ)
      END ; $ CALCONSULTER $
      PROCEDURE CALSORTIR ;
      BEGIN
            REF (NOTICE) NOTI ;
            NOTI :- NOTICEQ.FIRST ;
            WHILE NOTI.PR =/= CURRENT DO NOTI :- NOTI.SUC ;
            NOTI.OUT
      END ; $ CALSORTIR $
      PROCEDURE CONSTCALENDRIER ;
      BEGIN
            REF (NOTCAL) Y ;
            REAL AD,AF ;
            BOOLEAN FIN ;
            LECTUREDATE (AD,FIN) ;
            OUTTEXT (' LE CALENDRIER ASSOCIE A LA RESSOURCE   ') ;
            OUTTEXT (NOM) ; OUTTEXT (' EST LE SUIVANT :'); OUTIMAGE ;
            WHILE NOT FIN DO
            BEGIN
             LECTUREDATE (AF,FIN) ;
            IF SYSOUT.POS >= 80 THEN OUTIMAGE ;
            ECRI (AD) ; ECRI (AF) ;
                  Y :- NEW NOTCAL (AD,AF) ;
                  Y.INTO (NOTCALQ) ;
                  LECTUREDATE (AD,FIN)
            END ;
            ININAGE ;
            OUTIMAGE ;
            Y :- NOTCALQ.FIRST ;
            HEURETRAVAIL := FALSE ;
            ACTIVATE NEW ACT (THIS CALENDRIER) AT Y.DEBUT + JDP ;
            ACTIVATE NEW DESACT (THIS CALENDRIER) AT Y.FIN + JDP
      END ; $ CONSTCALENDRIER $
      PROCEDURE STATACTIVATION ;
      BEGIN
            REAL HMES ;
            IF HDISPO < 0 THEN TPSINUTIL := TPSINUTIL + TIME + HDISPO ;
            HDISPO := TIME ;
            HDEB := TIME ;
            HMES := TIME - TPSINUTIL ;
            IF PRECISION THEN ECRIFOCC (NORES,TYPE,ACCES,HMES) ;
      END ;  $  STATACTIVATION  $
```



```
      PROCEDURE STATDESACTIVATION ;
      BEGIN
            TPSDISPO := TPSDISPO + (TIME - HDISPO) * RPMAXDISPO  ;
            STATMAJ (0) ;
            HDISPO := - TIME ;
      END ;  $  STATDESACTIVATION  $
      IF NOM.LENGTH > 15 THEN NOM :- NOM.SUB (1,15) ;
      NOPER := 0 ;
      IF B THEN BEGIN
            NOTICEQ :- NEW HEAD ;
            NOTCALQ :- NEW HEAD ;
            TRAVAIL :- NEW CONDITION (THIS MONITOR) ;
            CONSTCALENDRIER
      END
      ELSE HEURETRAVAIL := TRUE ;
END ; $ CALENDRIER $
CALENDRIER CLASS CALRC ;
BEGIN
      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 (1,1) ;
      INTO (CALRCQ) ;
      OCCUPE := FALSE
END ; $ CALRC $
CALENDRIER CLASS CALRCP ;
BEGIN
      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 (1,2) ;
      INTO (CALRCPQ) ;
      OCCUPE := FALSE ;
      LIBRE :- NEW CONDITIONP (THIS MONITOR)
END ; $ CALRCP $
CALENDRIER CLASS CALRAMINF ;
BEGIN
      PROCEDURE ACQUERIR ;
      STATACQUERIR (1) ;
      PROCEDURE LIBERER ;
      BEGIN
            STATLIBERER (1) ;
            STATSERVIS
      END ; $ LIBERER $
      STATINIT (11,8) ;
      INTO (RAMINFQ) ;
END ; $ CALRAMINF $
CALENDRIER CLASS CALRAM (QUANTITE) ; REAL QUANTITE ;
BEGIN
      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,5) ;
      INTO (CALRAMQ ) ;
      RESSUFFI :- NEW CONDITION (THIS MONITOR)
END ; $ CALRAM $
```



```
CALENDRIER CLASS CALCPP (K) ; REAL K ;
BEGIN
      BOOLEAN OCCUPE ; INTEGER N ;
      REF (CONDITION) LIBRE ;
      PROCEDURE ACQUERIR (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 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 B THEN CALSORTIR ;
      END ; $ LIBERER $
      OCCUPE := FALSE ;
      N := 0 ;
      LIBRE :- NEW CONDITION (THIS MONITOR) ;
      STATINIT (1,7) ;
END ; $ CPP $
CALENDRIER CLASS CALRCPP ;
BEGIN
      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.EMPTY
            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 (1,1) ;
      INTO (CALRCPP0) ;
      TOUR :- NEW CONDITIONP (THIS MONITOR)
END ; $ CALRCPF $
LINK CLASS TREGION (NOM) ; TEXT NOM ;
BEGIN
      BOOLEAN PRECIS,BHISTO ;
      REF (HISTOGRAMME) H ;
      INTEGER RGENT , RGSORT , RGMAX ,NOREG ;
      REAL RGTPSAT , RGTPSMAX , RGTPSMIN ,RGMOY,HMOY ;
      PROCEDURE ENTRER (TR) ; REF (TRANSACTION) TR ;
      BEGIN
            INSPECT TR DO HDEBRG := TIME ;
            MAJ ;
            RGENT := RGENT + 1 ;
            IF RGENT - RGSORT > RGMAX THEN RGMAX := RGENT - RGSORT
      END ; $ ENTRER $
      PROCEDURE SORTIR (TR) ; REF (TRANSACTION) TR ;
      BEGIN
            REAL TPSE ;
            INSPECT TR DO TPSE  := TIME - HDEBRG ;
            MAJ ;
            RGSORT := RGSORT + 1 ;
            IF PRECIS THEN ECRIFREP (NOREG,TPSE) ;
            IF BHISTO THEN H.AJOU (TPSE ) ;
            IF RGSORT = 1 THEN RGTPSMIN := RGTPSMAX := TPSE ;
            IF TPSE < RGTPSMIN THEN RGTPSMIN := TPSE ;
            IF TPSE > RGTPSMAX THEN RGTPSMAX := TPSE ;
            RGTPSAT := RGTPSAT + TPSE ;
      END ; $ SORTIR $
      PROCEDURE MAJ ;
      BEGIN
            RGMOY := RGMOY + (RGENT - RGSORT) * (TIME - HMOY) ;
            HMOY := TIME ;
      END ; $ MAJ $
```



```
      PROCEDURE REINIT ;
      BEGIN
            RGMAX := RGENT := RGENT - RGSORT ;
            RGSORT := 0 ;
            RGTPSMIN := RGTPSMAX := RGTPSAT := RGMOY := 0  ;
            HMOY := TIME ;
            PRECIS := FALSE ;
            BHISTO := FALSE ;
      END ; $ INIT $
      IF NOM.LENGTH > 15 THEN NOM :- NOM.SUB (1,15) ;
      NOREG := NUMREG ;
      HMOY := TIME ;
      PRECIS := FALSE ;
      INTO (REGIONQ )
END ; $ TREGION $
MONITOR CLASS TFILE (MESURE,EQUILIBRE,INTER,DMES)   ;
REAL INTER , DMES ;
BOOLEAN MESURE,EQUILIBRE ;
BEGIN
      REF (HEAD) Q ;
      REF (CONDITION) NONVIDE ;
      INTEGER FAPRE,FAMAX,FANUL,FASORT,NBMES,NOFILE ;
      REAL TVOL,FATPSAT,VOL,VOLMIN,VOLMAX ,HMOB,FAPDS,FAMOY;
      PROCEDURE ENTRER (T) ;REF (TRANSACTION) T ;
      BEGIN
            THIS MONITOR.ENTER ;
            T.INTO (Q) ;
            INSPECT T DO MAJ (1,PDS  ;
            IF FAPRE > FAMAX THEN FAMAX := FAPRE ;
            NONVIDE.SIGLEAVE
      END ;
      PROCEDURE MODIFIER (T,P) ;
      REF (TRANSACTION) T ;
      REAL P ;
      BEGIN
            THIS MONITOR.ENTER ;
            MAJ (0,P-T.PDS) ;
            T.MODIFIERPOIDS (P) ;
            THIS MONITOR.LEAVE ;
      END ; $ MODIFIER $
      REF (TRANSACTION) PROCEDURE SORTIR  ;
      BEGIN
            REF (TRANSACTION) T ;
            THIS MONITOR.ENTER ;
            IF Q.EMPTY THEN NONVIDE.CWAIT ;
            T :- Q.FIRST ;
            INSPECT T DO MAJ (-1,-PDS) ;
            T.OUT ;
            IF FAPRE = 0 THEN FANUL := FANUL + 1 ;
            FASORT := FASORT + 1 ;
            SORTIR :- T ;
            THIS MONITOR.LEAVE ;
      END ;
```

```
PROCEDURE MAJ (NB,PDSUNIT) ; VALUE PDSUNIT;
INTEGER NB ; REAL PDSUNIT ;
BEGIN
      FAPDS := FAPDS +  PDSUNIT ;
      FATPSAT := FATPSAT + (TIME - HMOD) * FAPRE ;
      FAPRE := FAPRE + NB ;
      HMOD := TIME ;
END ; $ MAJ $
REAL PROCEDURE VOLUME ;
VOLUME := FAPDS ;
PROCEDURE ENLEVER (T) ; REF (TRANSACTION) T ;
BEGIN
      THIS MONITOR.ENTER ;
      INSPECT T DO
      BEGIN
            MAJ (-1,- PDS) ;
            OUT
      END ;
      FASORT := FASORT + 1 ;
      IF FAPRE = 0 THEN FANUL := FANUL + 1 ;
      THIS MONITOR.LEAVE ;
END ; $ ENLEVER $
REF (TRANSACTION) PROCEDURE SUIVANT (T) ; REF (TRANSACTION) T ;
BEGIN
      THIS MONITOR.ENTER ;
      SUIVANT :- T.SUC QUA TRANSACTION ;
      THIS MONITOR . LEAVE ;
END ; $ SUIVANT $
INTEGER PROCEDURE LONGUEUR ;
LONGUEUR := Q.CARDINAL ;
REF (TRANSACTION) PROCEDURE PREMIER ;
BEGIN
      THIS MONITOR.ENTER ;
      PREMIER :- Q.FIRST QUA TRANSACTION ;
      THIS MONITOR.LEAVE ;
END ; $ PREMIER $
PROCEDURE DETRUIRE ;
BEGIN
      REF (TRANSACTION) T ;
      WHILE LONGUEUR > 0 DO
      BEGIN
            T :- SORTIR ;
            T :- NONE ;
      END ;
END ; $ DETRUIRE $
PROCEDURE MESUREFILE ;
BEGIN
      VOL := FAPDS ;
      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 ;
                  HMOD := TIME ;
            END ;
            Q :- NEW HEAD ;
             NONVIDE :- NEW CONDITION (THIS MONITOR) ;
            IF NOM.LENGTH > 15 THEN NOM :- NOM.SUB (1,15) ;
            HMOD := TIME ;
            IF EQUILIBRE OR MESURE  THEN  INTO (FILEQ) ;
            IF EQUILIBRE
            THEN BEGIN
                        NOFILE := NUMFILE ;
                        ACTIVATE NEW MESUREVOL(THIS TFILE) QUA PROCESS
                                          AT DMES ;
                  END;
      END ; $ TFILE $
      LINK CLASS REFRES  (PT) ;REF (RESSOURCE) PT ;NULL;
      LINK CLASS MESSAGE (NO) ; INTEGER NO ; NULL ;
      MESSAGE CLASS TRANSACTION (PDS) ; REAL PDS ;
      BEGIN
            REAL HDEDF , HDEBRG ;
            PROCEDURE MODIFIERPOIDS (P) ; REAL P ;
                  PDS := P ;
      END ; $ TRANSACTION $
LINK CLASS HISTOGRAMME (IDENT,N,INF,PAS,LARG,HAUT) ;
INTEGER N,LARG,HAUT;    TEXT IDENT ;
REAL INF,PAS ;
BEGIN
   INTEGER ARRAY A (-1 : N) ;
   INTEGER ENTREES ;
   REAL MINI,MAXI ;
   PROCEDURE  AJOU  (VAL) ;  REAL VAL ;
   BEGIN
      REAL J;
      INTEGER I;
      ENTREES := ENTREES + 1 ;
      IF ENTREES = 1 THEN MINI := MAXI := VAL ELSE
      IF VAL < MINI THEN MINI := VAL ELSE
      IF VAL > MAXI THEN MAXI := VAL ;
      I := ENTIER((VAL - INF )/ PAS ) ;
      IF I < 0 THEN  A(-1) := A(-1) + 1 ELSE
      IF I > N THEN  A (N) := A (N) + 1
               ELSE  A (I) := A (I) + 1 ;
   END ;
   PROCEDURE REINIT ;
   BEGIN
      INTEGER I ;
      ENTREES := 0 ;
      FOR I := -1 STEP 1 UNTIL N DO
            A(I) := 0 ;
   END ; $ REINIT $
```

```
   INTO (HISTOO) ;
END ; $ HISTOGRAMME $
      LINK CLASS EVNOTICE (DATACT,ACTN) ; REAL DATACT;REF (ACTMONITOR) ACTN ;
      NULL ;
      LINK CLASS DATE (JO,HE,MI) ; INTEGER JO,HE,MI ;
      NULL ;
      LINK CLASS NOTICE (PR,DT) ;  REAL DT ; REF (PROCESSUS) PR ;
      NULL ;
      LINK CLASS NOTCAL (DEBUT,FIN) ; REAL DEBUT,FIN ;
      NULL ;
      PROCESS CLASS PROCESSUS (NOM,PRIORITE,T) ; TEXT NOM ; REAL PRIORITE ;
                                                          REAL T ;
      NULL ;
      PROCESS CLASS ACTIVATION (EC) ; REF (ECHEANCIER) EC ;
      BEGIN
            REF (EVNOTICE) COUR ;
            COUR :- EC.ECHQ.LAST ;
            WHILE TRUE DO
            BEGIN
                  COUR :- EC.ECHQ.FIRST ;
                  WHILE COUR =/= NONE DO
                  BEGIN
                        REACTIVATE THIS PROCESS AT COUR.DATACT + EC.HDP ;
                        COUR.ACTN.ENVOYER (NEW MESSAGE (0)) ;
                        COUR :- COUR.SUC
                  END ;
                  EC.HDP := EC.HDP + EC.PER ;
            END
      END ; $ ACTIVATION $
      PROCESS CLASS ACT (CAL) ; REF ( CALENDRIER) CAL ;
      BEGIN
            REF (NOTCAL) X ;
            X :- CAL.NOTCALQ.FIRST ;
            WHILE TRUE DO
            BEGIN
                  REF (NOTICE) Y ;
                  REACTIVATE THIS PROCESS
                     AT X.DEBUT + CAL.PER * CAL.NOPER + CAL.JDP PRIOR ;
                  CAL.STATACTIVATION ;
                  CAL.HEURETRAVAIL := TRUE ;
                  Y :- CAL.NOTICEQ.FIRST ;
                  WHILE Y =/= NONE DO
                  BEGIN
                        IF Y.DT = 0
                        THEN REACTIVATE Y.PR QUA PROCESS AFTER CURRENT
                        ELSE IF NOT Y.DT = -1
                             THEN REACTIVATE Y.PR QUA PROCESS DELAY Y.DT PRIOR ;
                        Y :- Y.SUC
                  END ;
                  CAL.TRAVAIL.CSIGNALG ;
                  X :- X.SUC ;
                  IF X == NONE
                  THEN BEGIN
                        CAL.NOPER := CAL.NOPER + 1 ;
                        X :- CAL.NOTCALQ.FIRST
                        END ;
            END
      END ; $ ACT $
```



```
PROCESS CLASS DESACT (CAL) ; REF ( CALENDRIER) CAL ;
BEGIN
      REF (NOTCAL) X ;
      X :- CAL.NOTCALQ.FIRST ;
      WHILE TRUE DO
      BEGIN
            REF (NOTICE) Y ;
            REACTIVATE THIS PROCESS
                     AT X.FIN + CAL.PER * CAL.NOPER + CAL.JDP ;
            REACTIVATE THIS PROCESS DELAY 0 ;
            CAL.STATDESACTIVATION ;
            CAL.HEURETRAVAIL := FALSE ;
            Y :- CAL.NOTICEQ.FIRST ;
            WHILE Y =/= NONE DO
            BEGIN
                  IF NOT Y.PR.IDLE AND NOT CAL.TERMINE
                  THEN BEGIN
                        Y.DT := Y.PR.EVTIME - TIME ;
                        CANCEL (Y.PR QUA PROCESS )
                        END ;
                  Y :- Y.SUC
            END ;
            X :- X.SUC ;
            IF X == NONE THEN BEGIN
                              X :- CAL.NOTCALQ.FIRST ;
                              END ;
      END
END ; $ DESACT $
PROCESS CLASS MESUREVOL (FL) ; REF (TFILE) FL ;
BEGIN
      TEXT LIBN ;
      WHILE TRUE DO
      INSPECT FL DO
      BEGIN
            LIBN :- BLANKS(12+NOM.LENGTH) ;
            LIBN.PUTTEXT ('MESURE.FILE.') ;
            LIBN.PUTTEXT (NOM) ;

            PASSE (LIBN,TRUE) ;
            MESUREFILE ;
            PASSE (LIBN,FALSE) ;
            IF RECHEQUIL THEN
            ECRIFVOL (NBPAS,NOFILE,VOL) ;
            REACTIVATE THIS PROCESS DELAY INTER ;
      END ;
END ; $  MESUREVOL  $
PROCEDURE ECRI (T) ; REAL T ;
BEGIN
      REF (DATE) D ;
      D :- CONVNH (T) ;
      OUTINT (D.JO,3) ;
      OUTINT (D.HE,3) ;
      OUTINT (D.MI,3) ;
      OUTTEXT (' ')
END ; $ ECRI $
```

```
PROCEDURE LECTUREDATE (H,FIN) ; NAME H,FIN ; BOOLEAN FIN ; REAL H ;
BEGIN
      CHARACTER C ;
      C := INCHAR ;
      IF C = "*"
      THEN FIN := TRUE
      ELSE BEGIN
            DTRA.JO := ININT ;
            DTRA.HE := ININT ;
            DTRA.MI := ININT ;
            H := CONVJHM (DTRA) ;
           FIN := FALSE
            END
END ; $  LECTUREDATE  $
REF (DATE) PROCEDURE CONVNH (NH) ; REAL NH ;
BEGIN
      REAL XH ;
      DTRA.JO := ENTIER (NH/24) ;
      XH := NH - DTRA.JO * 24 ;
      DTRA.HE := ENTIER (XH) ;
      DTRA.MI := ENTIER ((XH - DTRA.HE) * 60) ;
      CONVNH :- DTRA
END ; $  CONVNH  $
REAL PROCEDURE CONVJHM (DA) ; REF (DATE) DA ;
BEGIN
      CONVJHM := (DA.JO * 24 ) + ( DA.HE ) + ( DA.MI / 60 )
END ; $  CONVJHM  $
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 (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.ACQUERIR (X,QU) ;
            ECRI (TIME) ; OUTTEXT (CURRENT QUA PROCESSUS.NOM) ;
            OUTTEXT ('    DEBUT') ; OUTIMAGE ;
            HOLD (QU) ;
            ECRI (TIME) ; OUTTEXT (CURRENT QUA PROCESSUS.NOM) ;
            OUTTEXT ('    FIN  ') ; OUTIMAGE ;
            PP.LIBERER (X,QU)
      END ;
      PP.STATUTILISATEUR (-1) ;
      PP.STATSERVIS ;
      END
END ; $ UTILISERPP $
```



```
PROCEDURE OUVRIRFREP ;
BEGIN
      REF(TREGION) RG ;
      TEXT LIBF,LIBFICH ;
      LIBFICH :- 'REP1REP2REP3REP4REP5REP6REP7REP8REP9' ;
      LIBF :- LIBFICH.SUB ((NBPAS-1)*4+1,4)  ;
      FREP :- NEW OUTFILE (LIBF) ;
      FREP.OPEN(BLANKS(17)) ;
      INSPECT FREP DO
      BEGIN
            OUTINT (NBREG,2) ; OUTINT (NBPAS,1) ;
            OUTIMAGE ;
            FOR RG :- REGIONQ.FIRST,RG.SUC WHILE RG =/= NONE DO
            BEGIN
                  OUTTEXT (RG.NOM) ; SETPOS (16) ;
                  OUTINT (RG.NOREG,2) ; OUTIMAGE ;
            END ;
      END ;
END ; $OUVRIRFREP $
PROCEDURE OUVRIRFOCC ;
BEGIN
      REF (REFRES) P ;
      TEXT LIBF,LIBFICH ;
      REF (RESSOURCE) RES ;
      LIBFICH :- 'OCC1OCC2OCC3OCC4OCC5OCC6OCC7OCC8OCC9' ;
      LIBF :- LIBFICH.SUB ((NBPAS-1)*4+1,4)  ;
      FOCC :- NEW OUTFILE (LIBF) ;
      FOCC.OPEN (BLANKS(18)) ;
      INSPECT FOCC DO
      BEGIN
            OUTINT (NBRES,2) ; OUTINT (NBPAS,1) ;
            OUTIMAGE ;
            FOR P :- RESSOURCEQ.FIRST,P.SUC WHILE P =/= NONE DO
            BEGIN
                  RES :- P.PT ;
                  INSPECT RES DO
                  BEGIN
                        OUTTEXT (NOM) ; SETPOS(16) ;
                        OUTINT (NORES,2) ;OUTINT (TYPE,1);
                        OUTIMAGE ;
                        OUTINT (PMAXDISPO*100,6);
                        OUTIMAGE ;
                  END;
            END ;
      END ;
END ; $ OUVRIRFOCC $
PROCEDURE OUVRIRFVOL ;
BEGIN
      REAL DUREE ;
      REF (TFILE) FL ;
      TEXT LIBF,LIBFICH ;
      LIBFICH :- 'VOL1VOL2VOL3VOL4VOL5VOL6VOL7VOL8VOL9' ;
      LIBF :- LIBFICH.SUB ( (NBPAS-1)*4+1,4) ;
      FVOL :- NEW OUTFILE (LIBF) ;
      FVOL.OPEN ( BLANKS (17)) ;
      INSPECT FVOL DO
```

```
      BEGIN
            OUTINT (NBFILE,2) ;
            OUTIMAGE ;
            FOR FL :- FILEQ.FIRST, FL.SUC WHILE FL =/= NONE DO
            INSPECT FL DO
            IF EQUILIBRE
            THEN BEGIN
                        OUTTEXT (NOM) ; SETPOS (16);
                        OUTINT  (NOFILE,2) ;
                        OUTIMAGE ;
                        OUTINT (INTER,10) ;
                        OUTINT (ENTIER (DURSIM/INTER + 0.5),7) ;
                        OUTIMAGE ;
                 END ;
      END ;
END ; $  OUVRIRFVOL  $
PROCEDURE ECRIFVOL (NBPAS,NOFILE,VOL);
INTEGER NBPAS,NOFILE ;
REAL VOL;
BEGIN
      INSPECT FVOL DO
      BEGIN
            OUTINT (NOFILE, 2) ;
            OUTINT (VOL * 100,15)  ;
            OUTIMAGE ;
      END ;
END ; $  ECRIFVOL  $
PROCEDURE ECRIFOCC (NORES,TYPE,ACCES,HMESURE) ;
INTEGER NORES,TYPE  ;
REAL ACCES,HMESURE ;
BEGIN
      INSPECT FOCC DO
      BEGIN
            OUTINT (TYPE,1) ;OUTINT (NORES,2) ;
            OUTINT (HMESURE*100,10) ; OUTINT (ACCES*100,5 );
            OUTIMAGE ;
      END ;
END ; $ ECRIFOCC $
PROCEDURE ECRIFREP (NOREG,TPSE) ; INTEGER NOREG ; REAL TPSE ;
BEGIN
      INSPECT FREP DO
      BEGIN
            OUTINT (NOREG,2) ;
            OUTINT (TIME*100,8) ;
            OUTINT (TPSE*100,7) ;
            OUTIMAGE ;
      END ;
END ; $ECRIFREP$
PROCEDURE CONSTFGEN ;
BEGIN
      REF (HISTOGRAMME) HST;
      INTEGER NB ;
      REF (TFILE) FL ;
      REF (RESSOURCE) RES;
      REF (TREGION) RG ;
      TEXT LIBF,LIBFICH ;
      PROCEDURE ENRTFILE ;
```



```
BEGIN
      INSPECT FGEN DO
      FOR FL :- FILEQ.FIRST ,FL.SUC WHILE FL =/= NONE DO
      INSPECT FL DO
      IF MESURE THEN
      BEGIN
            MAJ (0,0) ;
            OUTTEXT (NOM) ; SETPOS (16) ;
            OUTINT (FASORT,5) ;
            OUTINT (FAPRE,5) ;
            OUTINT (FAMAX,5) ;
            OUTINT (FANUL,2) ;
            OUTINT  (FATPSAT*100,15) ;
            OUTIMAGE ;
      END ;
END ; $ ENRTFILE $
PROCEDURE ENRTREGION ;
BEGIN
      INSPECT FGEN DO
      FOR RG :- REGIONQ.FIRST ,RG.SUC WHILE RG =/= NONE DO
      INSPECT RG DO
      BEGIN
            MAJ ;
            OUTTEXT (NOM) ; SETPOS (16) ;
            OUTINT (RGENT,4)  ;
            OUTINT (RGSORT,1) ;
            OUTINT (RGMAX,4)  ;
            OUTINT (RGMOY/((TIME-DEBSIMUL)*100,5) ;
            IF RGSORT > 0 THEN RGTPSAT := RGTPSAT/RGSORT;
            OUTINT (RGTPSAT*100,9) ;
            OUTINT (RGTPSMIN*100,9) ;
            OUTINT (RGTPSMAX*100,10) ;
            OUTIMAGE ;
      END ;
END ; $ ENRTREGION $
PROCEDURE ENRCALRAM (TYPER) ; INTEGER TYPER;
BEGIN
      IF TYPER = 1 THEN RES :- CALRAMQ.FIRST ELSE
      IF TYPER = 2 THEN RES :- CALRAMPQ.FIRST
                  ELSE RES :- CALRAMPPQ.FIRST ;
      INSPECT FGEN DO
      WHILE RES =/= NONE DO
      INSPECT RES DO
      BEGIN
            OUTTEXT (NOM) ; SETPOS (16) ;
            OUTINT (RPMAXDISPQ*100,7 ) ;
            OUTINT (RPMAXUTIL*100,7 ) ;
            OUTINT (ACCES*100,7 ) ;
            OUTINT (TPSUTIL*100,12) ;
            OUTINT (TPSDISPQ*100,12) ;
            OUTIMAGE ;
            RES :- RES.SUC ;
      END ;
END ; $ ENRCALRAM  $
```

```
PROCEDURE ENRCALRC  (TYPER ) ; INTEGER TYPER ;
BEGIN
      IF TYPER = 1 THEN RES :- CALRCQ.FIRST  ELSE
      IF TYPER = 2 THEN RES :- CALRCPQ.FIRST
                   ELSE RES :- CALRCPPQ.FIRST;
      INSPECT FGEN DO
      WHILE RES =/= NONE DO
      INSPECT RES DO
      BEGIN
            OUTTEXT (NOM) ; SETPOS (16);
            OUTINT (ACCES,1) ;
            OUTINT (RSORT,5) ;
            OUTINT (TPSUTIL*100,15) ;
            OUTINT (TPSDISPO*100,15) ;
            OUTIMAGE ;
            RES :- RES.SUC
      END ;
END ; $ ENRCALRC $
PROCEDURE ENRCALCPP ;
BEGIN
      INSPECT FGEN DO
      FOR RES :- CALCPPQ.FIRST ,RES.SUC WHILE RES =/= NONE DO
      INSPECT RES DO
      BEGIN
            OUTTEXT (NOM  ) ; SETPOS (16) ;
            OUTINT (RSORT,5) ;
            OUTINT (NBMAXUTIL,5) ;
            OUTINT (NBUTIL,5);
            OUTINT (TPSUTIL*100,15) ;
            OUTINT (TPSDISPO*100,15) ;
            OUTIMAGE ;
      END ;
END ; $ ENRCALCPP $
PROCEDURE ENRRAMINF  ;
BEGIN
      INSPECT FGEN DO
      FOR RES :- RAMINFQ.FIRST,RES.SUC WHILE  RES =/= NONE DO
      INSPECT RES DO
      BEGIN
            OUTTEXT (NOM) ; SETPOS (16) ;
            OUTINT (RSORT,5) ;
            OUTINT (RPMAXUTIL,5) ;
            OUTINT (ACCES,5) ;
            OUTINT (TPSUTIL*100,15) ;
            OUTINT (TPSDISPO*100,15) ;
            OUTIMAGE ;
      END ;
END ; $ ENRRAMINF $
```



```
PROCEDURE ENRFILEQUI;
BEGIN
      FL :- FILEQ.FIRST ;
      INSPECT FGEN DO
      WHILE FL =/= NONE DO
      INSPECT FL DO
      BEGIN
            IF EQUILIBRE
            THEN BEGIN
                      OUTTEXT (NOM) ; SETPOS (16) ;
                      OUTINT (NBMES,4) ;
                      OUTINT (INTER,5) ;
                      OUTINT (VOL,9);
                      OUTINT (VOLMAX,9) ;
                      OUTINT (VOLMIN,9);
                      OUTINT ( ENTIER(TVOL/NBMES + 0.5),9) ;
                      OUTIMAGE ;
                 END ;
            FL :- FL.SUC ;
      END ;
END ; $ ENRFILEQUI $
PROCEDURE ENRHISTO ;
BEGIN
      INTEGER P,I,J ;
      INTEGER BA ;
      HST :- HISTOQ.FIRST ;
      INSPECT FGEN DO
      WHILE HST =/= NONE DO
      INSPECT HST DO
      BEGIN
            OUTTEXT (IDENT) ; SETPOS (16) ;
            OUTINT (N,2);
            OUTINT (INF*100,10) ;
            OUTINT (PAS*100,9) ;
            OUTINT (MINI*100,9) ;
            OUTINT (MAXI*100,10) ;
            OUTINT (LARG,2) ;
            OUTINT (HAUT,3) ;
            OUTIMAGE ;
            P := (N+2)//15;
            IF P*15 < N + 2 THEN P := P + 1 ;
            FOR I := 1 STEP 1 UNTIL P DO
            BEGIN
                 BA := MIN (I*15-2,N) ;
                 FOR J:= (I-1)*15-1 STEP 1 UNTIL BA
                       DO  OUTINT ( A(J),4) ;
                  OUTIMAGE ;
            END ;
            HST :- HST.SUC ;
      END ;
END ; $ ENRHISTO $
```

```
PROCEDURE ENRDIAG ;
BEGIN
      INTEGER A,B ;
      REF (REFRES) P ;
      REAL TPOT ;
      INSPECT FGEN DO
      BEGIN
            FOR P :- RESSOURCEQ.FIRST,P.SUC WHILE P =/= NONE DO
            BEGIN
                  RES :- P.PT ;
                  INSPECT RES DO
                  BEGIN
                  B := MIN (RPMAXDISPO,10) ;
                  TPOT := UTIL (1) ;
                  FOR A := 2 STEP 1 UNTIL B DO
                      TPOT := TPOT + UTIL (A) ;
                  OUTTEXT (NOM) ; SETPOS (16) ;
                  OUTINT (TYPE,1) ;
                  OUTINT (RPMAXDISPO,2) ;
                  OUTINT (TPOT*100,15) ;
                  OUTINT (TPSDISPO/RPMAXDISPO*100,15) ;
                  OUTIMAGE ;
                  TPOT := TPOT + UTIL (0) ;
                  FOR A := 0 STEP 1 UNTIL B DO
                      OUTINT(UTIL(A)/TPOT*1000,4);
                  OUTIMAGE ;
                  END ;
            END ;
    END ;
END ; $ENRDIAG $

LIBFICH :- 'GEN1GEN2GEN3GEN4GEN5GEN6GEN7GEN8GEN9' ;
LIBF :- LIBFICH.SUB ((NBPAS-1)*4+1,4)  ;
FGEN :- NEW OUTFILE (LIBF) ;
FGEN.OPEN (BLANKS (60)) ;
INSPECT FGEN DO
BEGIN
      INTEGER NA ;
      OUTINT (NBPAS,2) ;
      OUTINT (DEBSIMUL*100,15) ;
      OUTINT (TIME * 100,15) ;
      OUTINT (CALRCQ.CARDINAL,2) ;
      OUTINT (CALRCPQ.CARDINAL,2) ;
      OUTINT (CALRCPPQ.CARDINAL,2) ;
      OUTINT (CALRAMQ.CARDINAL,2) ;
      OUTINT (CALRAMPQ.CARDINAL,2) ;
      OUTINT (CALRAMPPQ.CARDINAL,2) ;
      OUTINT (CALCPPQ.CARDINAL,2) ;
      OUTINT (RAMINFQ.CARDINAL,2) ;
      NB := NA := 0 ; ;
      FL :- FILEQ.FIRST ;
      WHILE FL =/= NONE DO
      BEGIN
            IF FL.EQUILIBRE THEN NB := NB + 1 ;
            IF FL.MESURE THEN NA := NA + 1 ;
            FL :- FL.SUC ;
      END ;
```



```
            OUTINT (NB,2) ;
            OUTINT (NA,2) ;
            OUTINT (REGIONQ.CARDINAL,2) ;
            OUTINT (HISTOQ.CARDINAL,2) ;
            OUTIMAGE ;
      END ;
      IF NOT ( CALRCQ.EMPTY)    THEN ENRCALRC (1) ;
      IF NOT ( CALRCPQ.EMPTY)   THEN ENRCALRC (2) ;
      IF NOT ( CALRCPPQ.EMPTY ) THEN ENRCALRC (3) ;
      IF NOT ( CALRAMQ.EMPTY  ) THEN ENRCALRAM (1) ;
      IF NOT ( CALRAMPQ.EMPTY ) THEN ENRCALRAM (2) ;
      IF NOT ( CALRAMPPQ.EMPTY ) THEN ENRCALRAM (3);
      IF NOT ( CALCPPQ.EMPTY ) THEN ENRCALCPP ;
      IF NOT ( RAMINFQ.EMPTY ) THEN ENRRAMINF ;
      IF NB > 0 THEN ENRFILEQUI ;
      IF NOT (FILEQ.EMPTY) THEN ENRTFILE ;
      IF NOT ( REGIONQ.EMPTY ) THEN ENRTREGION ;
      IF NOT ( HISTOQ.EMPTY )  THEN ENRHISTO ;
      ENRDIAG ;
      FGEN.CLOSE ;
END ; $  CONSTFGEN  $
PROCEDURE REINITIALISATION ;
BEGIN
      REF (RESSOURCE) RES ;
      REF (TREGION) RG ;
      REF (TFILE) FL ;
      REF (HISTOGRAMME) H  ;
      REF (REFRES) P ;
      FOR P :- RESSOURCEQ.FIRST,P.SUC WHILE P =/= NONE DO
      BEGIN
            RES :- P.PT ;
            RES.STATREINIT ;
      END ;
      FOR RG :-  REGIONQ.FIRST , RG.SUC  WHILE RG  =/= NONE
            DO RG.REINIT ;
      FOR FL :-  FILEQ.FIRST , FL.SUC WHILE FL =/= NONE
            DO  FL.REINIT ;
      FOR H  :-  HISTOQ.FIRST, H.SUC WHILE H =/= NONE
            DO H.REINIT ;
END ; $  REINITIALISATION  $
PROCEDURE HISTOAJOU (RG,VAL);
REF (TREGION) RG ;
REAL VAL ;
INSPECT RG WHEN TREGION DO
IF BHISTO THEN H.AJOU (VAL) ;
PROCEDURE PRECISEREGION (RG) ; REF (TREGION) RG ;
RG.PRECIS := TRUE ;
PROCEDURE PRECISERESSOURCE (RES) ; REF (RESSOURCE) RES;
RES.PRECISION := TRUE ;
PROCEDURE HISTOREGION(RG,N,INF,PAS,LARG,HAUT) ;
REF (TREGION) RG ;
INTEGER N,LARG,HAUT;
REAL INF,PAS ;
```

```
BEGIN
      INSPECT RG WHEN TREGION DO
      BEGIN
            BHISTO := TRUE ;
            H :- NEW HISTOGRAMME (NOM,N,INF,PAS,LARG,HAUT);
      END ;
END ; $ HISTOREGION $
PROCEDURE SIMUL (DUREESIMUL,RESGEN,EQUILIBRE,BREP,BOCC) ;
REAL DUREESIMUL ;
BOOLEAN RESGEN, EQUILIBRE, BOCC,BREP ;
BEGIN
      RECHEQUIL := EQUILIBRE ;
      DURSIM := DUREESIMUL ;
      DEBSIMUL := TIME ;
      PAGE ;
      U := U + 9741 ;
      NBPAS := NBPAS + 1 ;
      IF EQUILIBRE THEN OUVRIRFVOL ;
      IF BREP THEN OUVRIRFREP ;
      IF BOCC THEN OUVRIRFOCC ;
      HOLD (DUREESIMUL) ;
      IF RESGEN THEN CONSTFGEN ;
      IF EQUILIBRE THEN FVOL.CLOSE ;
      IF BOCC THEN FOCC.CLOSE ;
      IF BREP THEN FREP.CLOSE ;
      REINITIALISATION ;
 END ;  $  SIMUL  $
INTEGER PROCEDURE NUMRES ;
BEGIN
      NBRES := NBRES + 1 ;
      NUMRES := NBRES ;
END ; $NUMRES$
INTEGER PROCEDURE NUMFILE ;
BEGIN
      NBFILE := NBFILE + 1 ;
      NUMFILE := NBFILE ;
END ; $ NUMFILE $
INTEGER PROCEDURE NUMREG ;
BEGIN
      NBREG := NBREG + 1 ;
      NUMREG := NBREG ;
END ; $ NUMREG $
```



```
      BOOLEAN RECHEQUIL ;
      REF (HEAD) CALRAMQ,CALRCQ,CALRCPQ,FILEQ,REGIONQ,CALRAMPQ,CALRAMPPQ;
      REF (HEAD) RAMINFQ,RESSOURCEQ,CALCPPQ,CALRCPPQ,HISTOQ ;
      REF (OUTFILE) FVOL,FOCC,FREP,FGEN ;
      INTEGER NBFILE,NBREG,NBRES ;
      INTEGER U,NBPAS,PER,HDP ;
      REF (DATE) DTRA ;
      REAL DURSIM,DEBSIMUL ;
      U := 987654321 ;
      DTRA :- NEW DATE (0,0,0) ;
      NBPAS := 0 ;
      HISTOQ :- NEW HEAD ;
      CALRAMPQ :- NEW HEAD ;
      CALRAMPPQ :- NEW HEAD ;
      CALRAMQ :- NEW HEAD ;
      CALRCQ :- NEW HEAD ;
      CALRCPPQ :- NEW HEAD ;
      CALRCPQ :- NEW HEAD ;
      RAMINFQ :- NEW HEAD ;
      CALCPPQ :- NEW HEAD ;
      FILEQ :- NEW HEAD ;
      REGIONQ :- NEW HEAD ;
      RESSOURCEQ :- NEW HEAD ;
      INNER ;
   END ; $ MAESTRO $
END $
EOF HIT
*
```

REFERENCES BIBLIOGRAPHIQUES

ABA 79 — ABADIE J.et MESLIER F., Etude de l'utilisation des modèles ARIMA pour la prévision à très court terme de l'énergie journalière produite par EDF, RAIRO, vol. 13, n° 1, février 1979.

ADI 78 — ADIBA M., Un modèle relationnel et une architecture pour les systèmes de bases de données réparties - Application au projet POLYPHEME, thèse d'Etat, Grenoble, 1978.

AND 71 — ANDERSON T.W., The Statistical Analysis of Time Series, éd. Wiley and Sons, 1971.

AND 77 — ANDERSON O.D., The Box-Jenkins Approach to Time Series Analysis, RAIRO, Recherche Opérationnelle, vol. 11, n° 1, février 1977.

AST 76 — ASTRAHAN M.M. et al., System R : Relational Approach to Database Management, ACM Transactions on Database Systems, vol. 1, n° 2, 1976.

BAN 78 — BANCILHON F., On the completeness of query languages for relational data bases, rapport IRIA-Laboria, mai 1978.

BAS 75 — BASKETT F., CHANDY K.M., MUNTZ R.R. and PALACIOS F. G., Open, Closed, and Mixed Networks of Queues with Different Classes of Customers, JACM, vol. 22, n° 2, 1975.

BET 77 — BETOURNE C. et al., Le langage LEST, rapport de recherche SESORI n° 76 032, Toulouse, 1977.

BEZ 75 — BEZIVIN J., Concepts et méthodes pour la production de systèmes opératoires, thèse de spécialité, Rennes, 1975.

BEZ 76 BEZIVIN J. et al., SIMONE : Manuel de référence, Pub. interne n° 54, IRISA, Rennes, novembre 1976.

BEZ 78 BEZIVIN J. et al., Présentation du projet XIMONE, Actes du Congrès de l'AFCET-TTI, Gif-sur-Yvette, novembre 1978.

BOD 78 BODARD F. et PIGNEUR Y., Spécifications dynamiques d'un système d'information, Actes du Séminaire INFORSID gr. 2, Cergy-Pontoise, octobre 1978.

BOU 77 BOUILLE F., Un modèle universel de banque de données, simultanément partageable, portable et réparti, thèse d'Etat, Paris, avril 1977.

BOX 76 BOX G.E.P. and JENKINS G.M., Time Series Analysis : Forecasting and Control, éd. Holden-Day, édition révisée, 1976.

BRA 75 BRANDWAJN A., Equivalence et décomposition dans les modèles à files d'attente et leur application à l'évaluation des performances de systèmes d'exploitation, thèse d'Etat, Paris, janvier 1975.

BRA 77 BRATLEY P. et VAUCHER J., Cours de Langages de Simulation, Ecole d'été internationale de l'AFCET, Montréal, 1977.

BRI 73 BRINCH HANSEN P., Operating System Principles, éd. Prentice Hall, 1973.

BRI 75 BRINCH HANSEN P., The programming language Concurrent PASCAL, International Summer School, Munich, 1975.

BRI 76 BRINCH HANSEN P., The Solo Operating System, Software-practice and experience, vol. 6, 1976.

BRI 78 BRINCH HANSEN P., Distributed Processes : A Concurrent Programming Concept, CACM, vol. 21, n° 11, novembre 1978.

BUX 66 BUXTON J.N., Writing simulations in CSL, Computing Journal, vol. 9, 1966.

BUZ 71 BUZEN J., Queueing Networks Models for Multiprogramming, thèse Ph. D., Harward, 1971.

CAM 74 CAMPBELL R.H. and HABERMANN N., The Specification of Process Synchronization by Path Expressions, Lecture Notes in Computer Science, vol. 16, éd. Springer-Verlag, 1974.

CAV 79 CAVARERO J.L., LAPAGE : Un modèle et un outil d'aide à la conception de systèmes d'information, thèse d'Etat, Nice, juin 1979.

CHA 75 CHANDY K.M., HERZOG U. and WOO L., Parametric Analysis of Queueing Networks, IBM J. Res. Develop., janvier 1975.

CHA 78 CHANDY K.M. and MISRA J., Specification, Synthesis, Verification and Performance Analysis of Distributed Programs, pub. TR-86, University of Texas, Austin, novembre 1978.

CHA 79 CHABRIER J.J., Projet SPHINX : Construction structurée de système d'information d'organisation, Actes des Journées BIGRE, Nancy, janvier 1979.

CHE 76 CHEN P.P., The Entity-Relationship Model, ACM Transactions on Database Systems, vol. 1, n° 1, 1976.

CHEC 76 CHECROUN A. et FRESNAIS P., Modèle d'aide à la mise en place d'une structure administrative adaptée à une organisation, Actes du Colloque IRIA Informatique, Automatique et Sciences des organisations, Rocquencourt, Janvier 1976.

CHEV 76 CHEVAL J.L. et al., Un système d'aide à l'écriture de systèmes d'exploitation, Actes du Congrès AFCET-TTI, Gif-sur-Yvette, novembre 1976.

CII 72 CII, SIMULA sous SIRIS 7/8, Manuels de présentation et d'utilisation, CII-CIDOC, Louveciennes, 1972.

COD 70 CODD E.F., A Relational Model of Data for Large Shared Data Banks, CACM, vol. 13, n° 6, juin 1970.

COD 72 CODD E.F., Relational completness of data base sublanga-ges, Data base systems, Current Computer Science, Symposia Series, vol. 6, éd. Prentice Hall, 1972.

CON 63 CONWAY M.E., Design of a Separable Transition-Diagram Compiler, CACM, vol. 6, n° 7, 1963.

COR 69 CORBATO F.J., PL/1 as a tool for system programming, Datamation, 1969.

COU 73 COUGER D., Evolution of business system analysis techni-ques, Computing Surveys, vol. 5, n° 3, 1973.

COU 77 COURTOIS P.J., Decomposability, Queueing and Computer System Application, ACM Monograph Series, éd. Academic Press, 1977.

CRE 75 CREHANGE M., Description formelle, représentation, in-terrogation des informations complexes - Système PIVOINES, thèse d'Etat, Nancy, décembre 1975.

CRO 75 CROCUS, Systèmes d'exploitation des ordinateurs, éd. Dunod, 1975.

CUN 79 CUNIN P.Y., Fiabilité et Sécurité des Programmes-Pro-positions autour d'un langage d'essai, thèse d'Etat, Nancy, mars 1979.

DAH 66 DAHL O.J. and NYGAARD K., SIMULA - an ALGOL - based simulation language, CACM, vol. 9, n° 9, 1966.

DAH 68 DAHL O.J., Discrete event simulation languages, Program-ming Languages, éd. F. Genuys, Academic Press, 1968.

DAH 70 DAHL O.J., MYHRHAUG B., and NYGAARD K., SIMULA 67 common base language, pub. S-22, Norwegian Comp. Center, 1970.

DAR 79 DARGENT L. et MEYER D., Un mécanisme de contrôle ré-parti pour un système d'accès à des données réparties, Actes du Séminaire IRIA-IGDD Distributed Data Sharing Systems, Aix en Provence, mai 1979.

DAT 75 DATE C.J., An Introduction to Database Systems, éd. Addison-Wesley, 1975.

DDH 72 DAHL O.J., DIJKSTRA E.W. and HOARE C.A.R., Structured Programming, Academic Press, 1972.

DER 74 DERNIAME J.C., Le projet CIVA - Un système de program-mation modulaire, thèse d'Etat, Nancy, janvier 1974.

DER 78 DERNIAME J.C. et MINOT R., Réalisation d'un mécanisme d'abstraction appliqué aux bases de données, BIGRE n° 8, janvier 1978.

DER 79 DERNIAME J.C. et FINANCE J.P., Types abstraits de don-nées : spécification, utilisation, réalisation, Cours Ecole d'été d'informatique, Monastir, juillet 1979.

DIJ 67 DIJKSTRA E.W., Cooperating sequential processes, Programming Languages, éd. F. Genuys, Academic Press, 1967.

DIJ 75 DIJKSTRA E.W., Guarded Commands, Non determinacy and Formal Derivation of Programs, CACM, vol. 18, n° 8, août 1975.

DUF 76 DUFOURD J.F., Modèle dynamique de système de traitement d'information en organisation administrative, Actes du Séminaire INFORSID gr.2, Saint-Pierre de Chartreuse, octobre 1976.

FIC 76 FICHEFET J., Introduction à la simulation, Ecole d'été d'informatique internationale de l'AFCET, Lannion, juillet 1976.

FIN 74 FINANCE J.P., Contribution à la formulation de la sémantique d'un langage de programmation - Application à ALGOL 68, thèse de spécialité, Nancy, juin 1974.

FIN 78 FINANCE J.P., De la spécification abstraite d'une donnée à sa représentation en mémoire : les états successifs d'une information, Actes du Congrès AFCET-TTI, Gif-sur-Yvette, novembre 1978.

FIS 73 FISHMAN G., Concepts and methods in discrete event digital simulation, éd. Wiley and Sons, 1973.

FLO 77 FLORY A., Un modèle et une méthode pour la conception logique d'une base de données, thèse d'Etat, Lyon, 1977.

FOR 61 FORRESTER J.W., Industrial Dynamics, éd. MIT Press, 1961.

FOR 68 FORRESTER J.W., Industrial Dynamics : after the first decade, Management Science, vol. 4, n° 7, mars 1968.

FOU 78 FOUCAUT O. and ROLLAND C., Concepts for Design of an Information System Conceptual Schema and its Utilization in the REMORA Project, Actes de la Conference on Very Large Data Base, Berlin, septembre 1978.

GAA 72 Groupe ALGOL de l'AFCET, Définition du langage algorithmique ALGOL 68, éd. Hermann, 1972.

GAA 73 Groupe ALGOL de l'AFCET, Manuel du Langage ALGOL 68, éd. Hermann, 1973.

GAU 78 GAUDEL M.C. et TERRINE G., Synthèse de la représentation d'un type abstrait par des types concrets, Actes du Congrès AFCET-TTI, Gif-sur-Yvette, novembre 1978.

GEL 73 GELENBE E., Modèles de systèmes informatiques, thèse d'Etat, Paris, 1973.

GEL 75a GELENBE E., On Approximate Computer System Models, JACM, vol. 22, n° 2, 1975.

GEL 75b GELENBE E. and PUJOLLE G., Probabilistic models of computer systems, part II rapport IRIA-Laboria n° 147, décembre 1975.

GEL 75c GELENBE E. et LEROUDIER J., Cours de modèles de systèmes informatiques, Ecole internationale d'été d'informatique AFCET, Rabat, juillet 1975.

GELI 69 GELINIER O., Fonctions et tâches de direction générale, éd. Hommes et Techniques, 4è éd., 1969.

GES 77 GESCHKE C.M., MORRIS Jr. J.H. and SATTERTHWAITE H., Early Experience with Mesa, CACM, vol. 20, n° 8, août 1977.

GIR 76 GIRAUDIN J.P., Le projet MACSI - P - Méthode d'aide à la conception de systèmes d'information : construction de prototypes, thèse de spécialité, Grenoble, 1975.

GIR 78 GIRAULT C., Réseaux de PETRI et Synchronisation de processus, Actes des Journées Informatiques, Nice, juin 1978.

GUT 77 GUTTAG J.V., Abstract Data Types and the Development of Data Structures, CACM, vol. 20, n° 6, juin 1977.

GRE 75 GREIBACH S., Theory of Programs Structures : Schemes, Semantics, Verification, Lecture Notes in Computer Science, n° 36, éd. Springer, 1975.

GUT 78 GUTTAG J.V., HOROWITZ E. and MUSSER D.R., Abstract Data Types and Software Validation, CACM, vol. 21, n° 12, décembre 1978.

HAB 69 HABERMANN A.N., Prevention of System Deadlocks, CACM, vol. 12, n° 7, juillet 1969.

HAB 77 HABRIAS H., Les graphiques de l'analyste, 01 Informatique, n° 111, 1977.

HEW 77 HEWITT C. and ATKINSON R., Parallelism and synchronization in actor systems, 4 th ACM symposium on Principles of Programming Language, Los Angelès, janvier 1977.

HIL 67 HILLS P.R., SIMON - A computer simulation language in ALGOL, Digital simulation in operational research, éd. Hollingdall, 1967.

HOA 72 HOARE C.A.R. and WIRTH N., An axiomatic definition of the programming language PASCAL, Acta Informatica, vol. 2, éd. Springer-Verlag, 1972.

HOA 74 HOARE C.A.R., Monitors : An Operating System Structuring Concept, CACM, vol. 17, n° 10, octobre 1974.

HOA 75 HOARE C.A.R., The structure of an operating system, International Summer School, Munich, juillet 1975.

HOA 78 HOARE C.A.R., Communicating Sequential Processes, CACM, vol. 21, n° 8, août 1978.

HON 79a HONEYWELL Inc. and CII HONEYWELL BULL, Reference manual for the GREEN programming language, mars 1979.

HON 79b HONEYWELL Inc. and CII HONEYWELL BULL, Rational for the design of the GREEN programming language, mars 1979.

HUR 76 HURTUBISE R., Informatique et Information, éd. Agence d'Arc, Montréal et Editions d'Organisation, Paris, 1976.

IBM 70 IBM, General Purpose Simulation System/360 User's Manual, GH 20 - 0326, 1970.

ICH 72 ICHBIAH J.D. et al., Spécification de définition de LIS, rapport CII STG - 0 - 59 T, 1972.

INF 76 INFORSID, Colloque sur la Recherche en Informatique des Organisations, Caen, 1976.

JAC 63 JACKSON J.R., Jobshop - Like Queueing Systems, Management Science, vol. 10, n° 1 1963.

JACK 78 JACKSON M.A., Information Systems : Modelling, Sequencing and Transformations, Actes de la 3th Conference on Software Engineering, 1978.

JAC 78 JACQUET P., La généricité comme outil d'abstraction dans les langages de programmation, Actes du Congrès AFCET - TTI, Gif-sur-Yvette, novembre 1978.

KAR 65 KARR H.W. and al., SIMSCRIPT I.5, Consolidated Analysis Center, CACJ 65 - INT - 1, Santa Monica, 1965.

KAU 76 KAUBISCH W.H, PERROTT R.H. and HOARE C.A.R., Quasiparallel Programming, Software-practice and experience, éd. Wiley and Sons, vol. 1, 1976.

KAU 78 KAUBISCH W.H. and HOARE C.A.R., Discrete Event Simulation Based on Communicating Sequential Processes, pub. Dep. of Comp. Science, The Queen's University, Belfast, 1978.

KIR 77 KIRKERUD B., SIMBAS, Actes du Workshop SIMULA and Data Bases IP/ASU, Paris, mai 1977.

KIV 69 KIVIAT P.J. and al., The SIMSCRIPT II Programming Language, éd. Prentice Hall, 1969.

KLE 75 KLEINROCK L., Queueing Systems, vol. 1 : theory, éd. Wiley and Sons, 1975.

KLE 76 KLEINROCK L., Queueing Systems, vol. 2 : computer applications, éd. Wiley and Sons, 1976.

KLEIJ 75 KLEIJNEN J.P.C., Statistical Techniques in Simulation, vol. 1 et 2, éd. Dekker, 1975.

KLE 79 KLEIN P., Concepts et Outils pour l'Ecriture de Maquettes dans le Projet MAESTRO, thèse de spécialité, Nancy, mai 1979.

KNU 68 KNUTH D.E., The Art of Ccmputer Programming, éd. Addison-Wesley, 1968.

KOB 78 KOBAYASHI H., Modeling and Analysis : An Introduction to System Performance Evaluation Methodology, Addison - Wesley, 1978.

KRE 67 KREISEL G. et KRIVINE J.L., Eléments de logique mathématique, éd. Dunod, Paris, 1967.

LAN 73 LANGEFORS B., Theorical Analysis of Information Systems, éd. Auerbach, 1973.

LAR 75 Petit Larousse illustré, éd. Larousse, 1975.

LAU 75 LAUER P.E. and CAMPBELL R.H., Formal Semantics of a Class of High-Level Primitives for Coordinating Concurrent Processes, Acta Informatica, n° 5, éd. Springer-Verlag, 1975.

LED 75 LEDGARD H., and MARCOTTY M., A Genealogy of Control Structures, CACM, vol. 18, n°11, novembre 1977.

LEG 68 LE GALL P., Convergence des Simulations et Application aux Réseaux Téléphoniques, Les méthodes de simulation, pub. AFIRO, éd. Dunod, 1968.

LEL 77 LE LANN G., Introduction à l'analyse de systèmes multi-référenciels, thèse d'Etat, Rennes, mai 1977.

LEM 73 LEMOIGNE J.L., Les systèmes d'information dans les organisations, éd. PUF, 1973.

LER 77 LEROUDIER J., Systèmes adaptatifs à mémoire virtuelle, thèse d'Etat, Grenoble, mai 1977.

LIS 77 LISKOV B. and al., Abstraction Mechanisms in CLU, CACM, vol. 20, n° 8, août 1977.

LIV 78 LIVERCY C., Théorie des programmes, éd. Dunod, 1978.

LUG 75 LUGUET J., SCAPFACE - Système de conception automatique et progressive d'un système de base de données fondé sur l'analyse conversationnelle des états de sortie, thèse d'Etat, Toulouse, 1975.

LUS 72 LUSSATO B., Introduction critique aux théories des organisations, éd. Dunod, 1972.

MAK 77 MAKILA K., Data base handling in SIMULA of CODASYL-type, Actes du Workshop SIMULA and Data Bases IP/ASU, Paris, mai 1977.

MAL 71 MALLET R.A., La Méthode Informatique, éd. Hermann, 1971.

MAR 70 MARTZLOFF C., Les ordinateurs, l'analyse et l'organisation, éd. Dunod, 1970.

MEA 72 MEADOWS and al., The limits to growth, éd. Universe Books, 1972.

MEL 72 MELEZE J., Analyse Modulaire de Systèmes, éd. Hommes et Techniques, 1972.

MER 78 MERLE D., POTIER D. and VERAN M., A tool for computer system performance analysis, Actes de Intern. Conf. on Performance of Computer Installation, Gardone, juin 1978.

MEY 79 MEYER B., Quelques concepts importants des langages de programmation modernes et leur expression en SIMULA 67, Actes des Journées du gr. GROPLAN, Cargèse, mai 1979.

MIL 78 MILNER R., Algebras for communicating systems, CSR - 25 - 28, Dep. of Computer Science, Univ. of Edinburgh, juin 1978.

MIN 79 MINOT R., ATM : un système de fabrication de programmes basé sur les concepts de modularité et de type abstrait, thèse de spécialité, Nancy, mars 1979.

MOO 77 MOORE L.J. and al., Combined continuous-discrete system simulation with GASP IV, Comput. and Ops Res., vol. 4, éd. Pergamon Press, 1977.

NAY 66 NAYLOR T.H. and al., Computer Simulation Techniques, éd. Wiley and Sons, 1966.

NCC 76 Norwegian Computing Center, pub. S - 79, Oslo, mai 1976.

OPE 72 Operating systems techniques, éd. HOARE C.A.R. and PERROTT R.H., Academic Press, 1972.

OWI 78 OWICKI S., Verifying Concurrent Programs with Shared Data Classes, Formal Description of Programming Concepts, éd. Neuhold, North-Holland 1978.

PAI 71 PAIR C., Cours de Structures de données, Ecole d'été d'informatique AFCET, Alès, juillet 1971.

PAI 74 PAIR C., Formalization of the notions of data, information and information structure, Data Base Management Systems, éd. Klimbie-Koffman, North-Holland, 1974.

PAI 78 PAIR C., Some Theorical Aspects of Program Construction, International Summer School on Program Construction, Munich, 1978.

PAI 79 PAIR C., La Construction des Programmes, RAIRO, vol. 13, n° 2, 1979.

PAL 76 PALME J., A New Feature for Module Protection in SIMULA, SIMULA Newsletter, vol. 4, n° 1, février 1976.

PAM 69 Société des Fonderies de Pont-à-Mousson, Analyse fonctionelle et organique, Nancy, 1969.

PEA 77 PEAUCELLE J.L., Les besoins en informatique dans les systèmes informatiques de gestion, thèse d'Etat, Paris, 1977.

PEC 75 PECCOUD F., Le projet MACSI - Méthode d'aide à la conception de systèmes d'information, thèse d'Etat, Grenoble, 1975.

PEL 71 PELLAUMAIL P., Conception de système DB/DC, GUIDE, Actes de la 19è conférence, Bologne, 1978.

PET 62 PETRI C.A., "Kommunikation mit Automaten", Schriften des Rheinisch - Westfalischen Institutes für Instrumentelle Mathematik, 2è cahier, Univ. de Bonn, 1962.

PET 77 PETERSON J.L., Petri Nets, Computing Surveys, vol. 9, n° 3, septembre 1977.

POT 77 POTIER D., Modèles à files d'attente et gestion des ressources dans les systèmes informatiques, thèse d'Etat, Grenoble, janvier 1977.

PRI 69 PRIESLEY M.B. and SUBBARAO T., A test for non stationarity of time series, JRSS, vol. B31, n° 1, 1969.

PRIT 69 PRITSKER A.A., Simulation with GASP II, éd. Prentice Hall, 1969.

PUG 63 PUG A.L., DYNAMO User's manual, MIT Press, 1963.

QUE 79 Mac QUEEN D.B., Models for distributed computing, rapport n° 351, IRIA - Laboria, Rocquencourt, avril 1979.

RAY 78 RAYNAL M., Une expression de la synchronisation pour les types abstraits, RAIRO, vol. 12, n° 4, décembre 1978.

REI 71 REIX C., L'analyse en informatique de gestion, tomes 1 et 2, éd. Dunod, Paris, 1971.

REM 74 REMY J.L., Structures d'information - Formalisation des notions d'accès et de modification d'une donnée, thèse de spécialité, Nancy, 1974.

REM 79 REMY J.L., Enrichissement des types abstraits, Séminaire d'informatique du CRIN, Nancy, avril 1979.

RIC 79 RICHETIN M. et MILGRAM M., Analyse structurale et partition des systèmes complexes par les graphes, Analyse et commande des systèmes complexes, éd. CEPADUES, 1979.

ROB 73 Petit ROBERT, éd. Robert, 1973.

ROL 74 ROLLAND et al., Séminaires, conférences et thèses sur le projet REMORA, 1974 à 1979.

ROO 77 ROOS, Connection SIMULA - IMS, Actes du Workshop SIMULA and Data Bases, IP/ASU, Paris, mai 1977.

ROU 78 ROUCAYROL G.P., Mots de synchronisation, RAIRO, vol. 12, n° 4, décembre 1978.

SAA 70 SAAL H.J. and RIDDLE W.E., Communicating Semaphores, SLAC, CGTM 117, Stanford, 1970.

SCH 65 SCHWARTZ L., Méthodes mathématiques pour les sciences physiques, éd. Hermann, 1965.

SCH 72 SCHRIBER T., A GPSS Primer, Ulrich's Books, Ann Arbor, 1972.

SCH 73 SCHOENFIELD J.R., Mathematical Logic, éd. Addison-Wesley, 1973.

SCH 78 SCHOLL P.C., CUNIN P.Y. et GRIFFITHS M., Construction méthodique et vérification systématique de programmes : éléments d'un langage, Actes du Congrès AFCET - TTI, Gif-sur-Yvette, novembre 1978.

SCO 73 SCOM, Simplification du travail administratif - L'étude de processus, éd. SCOM, Paris, 1973.

SHA 77 SHAW M. and WULF W.A., Abstraction and Verification in Alphard : Defining and Specifying Iteration and Generators, CACM, vol. 20, n° 8, août 1977.

SIN 77 SINTZOFF M., Séminaires sur la synthèse de programmes, Nancy, 1977.

SIR 78 SIRIUS, Documentation sur le projet-pilote de l'IRIA, Le Chesnay, 1978.

TAR 79 TARDIEU H. et al., Conception d'un système d'information, Les Editions d'Organisation, Paris, 1979.

TEI 72 TEICHROEW D. and al., An introduction to computer based information processing system, ISDOS working paper n° 72, 1972.

THA 80 THAUREL M., Concepts et outils de mesure et d'analyse statistiques dans le projet MAESTRO, thèse de spécialité, Nancy, 1980 (à paraître).

VAU 71 VAUCHER J., Programmation de Simulation, Univ. de Montréal, Montréal, 1971.

VAU 73 VAUCHER J., A wait until algorithm for general purpose simulation languages, Actes de la Winter Simulation Conference, San Francisco, janvier 1973.

VAU 78 VAUCHER J. et DAVEY D., Accélération du wait until pour l'ordonnancement conditionnel en simulation, Actes du Congrès AFCET - TTI, Gif-sur-Yvette, novembre 1978.

VAU 79 VAUCHER J., BETA : Un successeur de SIMULA pour la programmation de systèmes, BIGRE, n° 15, juin 1979.

VEI 72 VEILLON et CAGNAT J.M., Cours de Programmation en Langage PL/1, 3 tomes, éd. A. Colin, 1972.

VER 78 VERJUS J.P., Expression de la synchronisation par invariants, Actes des journées du gr. GROPLAN de l'AFCET, La Bresse, mars 1978.

WAE 67 Van der WAERDEN B.L., Statistique mathématique, éd. Dunod, 1967.

WIJ 68 Van WIJNGAARDEN A., MAILLOUX B.J., PECK J.E.L. and KOSTER C.H.A., Report on the Algorithmic Language Algol 68, pub. Mathematisch Centrum, Amsterdam, 1968.

WIR 71 WIRTH N., The programming language PASCAL, Acta Informatica, vol. 1, n° 1, éd. Springer-Verlag, 1971.

WIR 76 WIRTH N., MODULA : A language for modular multiprogramming, Berichte des Instituts für Informatique, n° 18, 1976.

ZEI 76 ZEIGLER B.P., Theory of modelling and simulation, éd. Wiley and Sons, 1976.

REFERENCES BIBLIOGRAPHIQUES relatives au

PROJET MAESTRO

DUFOURD J.F.

Modèle dynamique de système d'information en organisation, actes du séminaire gr. 2 INFORSID, IRIA/SESORI, Grenoble, 1976 (CRIN 76-R-006).

DUFOURD J.F.

Quelques réflexions sur une méthode de conception de systèmes d'information, pub. dans journal CRIN, Nancy, 1976, (CRIN 76-R-061).

DUFOURD J.F. et KLEIN P.

Ordonnancement des processus dans le projet MAESTRO, pub. dans BIGRE n° 9, IRISA, Rennes, 1978 (CRIN 78-P-026).

DUFOURD J.F., HERTSCHUH N. et KLEIN P.

Le projet MAESTRO : Présentation générale, pub. dans actes du colloque CNRS/IRIA bilan ATP, Rennes, 1978 (CRIN 78-P-028).

DUFOURD J.F. et KLEIN P.

Les outils d'écriture de maquettes dans le projet MAESTRO, 1er rapport partiel ATP, Nancy, 1978 (CRIN 78-R-029).

DUFOURD J.F., HERTSCHUH N. et KLEIN P.

Le projet MAESTRO - Maquettes pour l'évaluation de systèmes d'information en organisation, actes du congrès de l'AFCET - TTI, Gif-sur-Yvette, 1978 (CRIN 78-P-056).

KLEIN P. et DUFOURD J.F.,

Projet MAESTRO - Exemple de maquette, 2è rapport partiel ATP, 1978 (CRIN 78-R-077).

KLEIN P.

Concepts et outils pour l'écriture de maquettes dans le projet MAESTRO, thèse de 3ème cycle, Nancy, mai 1979.

DUFOURD J.F.

Modélisation des systèmes d'information répartis dans les organisations, pub. dans BIGRE n° 13, et actes du Seminar on Distributed Data Sharing Systems IRIA/SIRIUS/IGDD, Aix-en-Provence, mai 1979 (CRIN 79-P-033).

DUFOURD J.F.

Types abstraits, modèle relationnel et langage SIMULA, Colloque Bases de Données, chap. français ACM/EDF/ Un. Paris VI, Paris, juin 1979 (CRIN 78-R-027).

DUFOURD J.F.

Maquettes de systèmes d'information, Actes du Congrès CIPS/DPMA/FIQ, Québec (Canada), juin 1979 (CRIN 79-P-034).

DUFOURD J.F. et THAUREL M.

The MAESTRO project - Models for evaluating information systems in organizations, actes de la 7th SIMULA User's Conference, Cernobbio (Italie), Septembre 1979 (CRIN 79-P-035).

DUFOURD J.F. et HERTSCHUH N.

Maquettes pour évaluer les systèmes d'information d'organisations, actes de la Convention Informatique, Paris, septembre 1979 (CRIN 79-P-036).

THAUREL M. et DUFOURD J.F.

Evaluation et outils statistiques dans le projet MAESTRO, 3è rapport partiel ATP, octobre 1979.

DUFOURD J.F.

Introduction de la dynamique dans des maquettes de systèmes d'information, actes du séminaire gr. 2 INFORSID, Aix-en-Provence, janvier 1980.

NOM DE L'ETUDIANT : Monsieur DUFOURD Jean François

NATURE DE LA THESE : DOCTORAT ès SCIENCES

VU, APPROUVE

et PERMIS D'IMPRIMER

NANCY LE 28 DEC 1979 11404

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

UNIVERSITE NANCY I
LE PRESIDENT
M. BOULANGE