UNIVERSITE DE MONTREAL

UNE INTERFACE ORIENTEE-OBJET POUR LA CONSTRUCTION MODULAIRE DE MODELES EN SIMULATION MANUFACTURIERE

par

Michel FABRE

DEPARTEMENT DE GENIE INDUSTRIEL

ECOLE POLYTECHNIQUE

MEMOIRE PRESENTE EN VUE DE L'OBTENTION

DU GRADE DE MAITRE ES SCIENCES APPLIQUEES (M.Sc.A.)

Août 1990

c Michel FABRE 1990

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-58937-X

Canada

UNIVERSITE DE MONTREAL

ECOLE POLYTECHNIQUE

Ce mémoire intitulé:

UNE INTERFACE ORIENTEE-OBJET POUR LA CONSTRUCTION MODULAIRE DE MODELES EN SIMULATION MANUFACTURIERE

présenté par Michel FABRE

en vue de l'obtention du grade de maître es sciences appliquées (M.Sc.A.),

a été dûment accepté par le jury d'examen constitué de:

M. Daniel LEBLANC, Ph.D., président

M. Jean-Marc ROBERT, Doctorat

M. Laurent VILLENEUVE, M. Eng.

LISTE DES FICHIERS ET DES REPERTOIRES DE LA DISQUETTE EN ANNEXE F

Le logiciel développé porte le nom de map1X. Les noms entre parenthèses sont les noms officiels des fichiers utilisés dans le logiciel sous UNIX; les autres sont ceux qui ont été attribués par le système d'exploitation DOS au moment de leur sauvegarde sur disquette. Pour plus de détails, se reporter aux fichiers "README.FIR" et "HOW_USE" qui sont des fichiers textes présentant le logiciel.

can_tilb (can_tile_b)

can_tile

const_de.h (const_def.h)

icone_gr (icone_graph)

intrinsi.h (intrinsic.h)

libmap.c

map1cpp.h (map1CPP.h)

map1cx.h (map1CX.h)

map1x.c (map1X.c)

max

min

HOW_USE

README.FIR

APPDEFAU ---- map1x (APPDEFAULTS, Map1X)

```
                x.log    (.login)
DEFAULTS ----   frm0_c    (defaults)
                ...
                frm19_c
                frm0_v
                ...
                frm19_v
ELEMENT -----   CONDINIT -----  inc_2      (element, condinit)
                                incndt_1   (INCNDT_1)
                GOAL ---------  goal_1     (goal)
                                goal_2
                RESOURCE -----  res_1      (resource)
                                res_2
                PRODUCT ------  prdc_2     (product)
                                prdc_3
                                prdc_4
                                GRAPH -----  prdc_2_lk  (.graph)
                                             prdc_3_lk
                                             prdc_4_lk
PROJECT -----   pjt1   (project)
                pjt2
REPORT_T ----   bottlene   (report_type, BOTTLENECK)
                capacity   (CAPACITY_PLANNING)
                design     (DESIGN)
                downtime   (DOWNTIME)
                inventol   (INVENTORY_SPACE)
```

```
                inventor   (INVENTORY)
                lead_tim   (LEAD_TIME)
                material   (MATERIAL_HANDL.)
                overtime   (OVERTIME)
                personne   (PERSONNEL)
                product1   (PRODUCTION_MIX)
                product2   (PRODUCTIVITY)
                producti   (PRODUCTION)
                reliabil   (RELIABILITY)
                rep_std    (.rep_std)
                rep_type   (.rep_type)
                scheduli   (SCHEDULING)
                status     (STATUS)
                summary    (SUMMARY)
                supply     (SUPPLY)
                throughp   (THROUGHPUT)
                time       (TIME)
                trace      (TRACE)
                utilizat   (UTILIZATION)
                warm_up_   (WARM_UP_PERIOD)
RES_KERN ----   conv   (res_kern)
                fixt
                pers
                stat
                stoc
                tran
```

SOMMAIRE

En raison de la mondialisation des échanges, les entreprises et leurs gestionnaires ont, encore moins qu'avant, le droit à l'erreur quant à la prise de décision. Dans le contexte des entreprises manufacturières auquel nous nous intéressons, il existe des outils pour mettre en valeur les informations en provenance du système de manière à prendre de bonnes décisions. Au rang de ces outils, la simulation est sans nul doute le plus performant en raison de son caractère dynamique mais reste encore trop peu utilisé en regard de ses possibilités.

Cela est dû au fait que le bon usage de logiciels de simulation réclame des compétences nombreuses et variées (informatique, statistiques, milieu étudié...) qui ne sont pas nécessairement celles de l'utilisateur potentiel, c'est-à-dire de l'ingénieur ou du gestionnaire qui a un problème à résoudre: sa connaissance porte sur le système industriel, pas sur l'outil.

Pour qu'il puisse accéder plus facilement à la simulation, en l'occurrence au logiciel MAP/1, nous avons conçu une interface dirigée par une souris et multi-fenêtrée faisant appel au système X Window. Le logiciel développé faci-

lite la phase de modélisation (30% à 40% de l'effort total dans un projet de simulation) et celles de mise à jour et de maintenance des modèles en supprimant tout codage.

Il permet ainsi de faire évoluer le modèle après sa validation et son implantation, au gré des changements du système étudié. Il encourage la réutilisation des éléments qui composent un modèle en le découpant en quatre parties (description des conditions initiales, des objectifs de la simulation, des moyens industriels et des produits) qui sont stockées indépendamment dans la base de données.

Ce logiciel sert également de prototype pour valider le principe de la programmation orientée objet (POO) en simulation et constitue le point de départ pour le développement d'un simulateur orienté objet proprement dit qui représente à notre avis la meilleure solution pour construire un système à la fois plus convivial et plus performant que les outils actuels.

Mots clés: Simulation, interface homme-machine, Programmation orientée objet, modélisation

ABSTRACT

Due to the increasing world-wide competition, manufacturing managers must take the right decision at the right time to keep their firm in the race. In such a context, an inappropriate decision could involve desastrous consequences. Such situation can be avoided, by the use of decision support systems and other tools like simulation systems which provide major help. Simulation is not only under-used but also misused with respect to its high dynamic capacity for system analysis.

In fact, using simulation asks major training in lots of different specialities (software development, statistics, modeling, manufacturing system studied...) and so can't be performed quickly by the engineer or the manager who has a problem to solve: he has knowledge about the industrial system, but not about the tool.

In order to help the decision maker, we have developed an user-friendly interface for the simulator MAP/1, using X window programming system. Our front-end makes the modeling task (which generaly takes about 40% of the total simulation effort) and the maintenance task easier to perform while removing all the coding task.

So, the model can evolve following the developments and the changes in the system itself. Reusing parts of simulation models is supported and encouraged by our system, dividing the modeling task in four parts (initial conditions, goals of study, industrial resources and products description) which are stored independently in the database.

This software is also a prototype developed to test and validate object-oriented programming in simulation and can be used as the starting point to develop a fully object-oriented simulator which seems to be the most appropriate way to make simulation an easier and more efficient tool to use.

Key words: User-friendly Simulation, Object-oriented, Modelling

REMERCIEMENTS

Je tiens tout d'abord à remercier le professeur Laurent VILLENEUVE pour avoir accepté de diriger cette recherche et pour l'aide qu'il m'a apportée durant la réalisation de ce travail.

Qu'il me soit ensuite permis de remercier le professeur Daniel LEBLANC pour son soutien, pour la confiance dont il m'a honoré et surtout pour les discussions que nous avons eues et qui ont toujours été pour moi source d'enrichissement.

Je remercie Denis A. MARTIN, ingénieur système au laboratoire de VLSI, pour son aide ô combien précieuse lors de la phase informatisée de ce projet, ses connaissances m'ont été fort utiles.

Je tiens aussi à remercier les étudiants de génie industriel que j'ai eu beaucoup de plaisir à côtoyer au cours de ces deux années et particulièrement ceux de maîtrise et de doctorat qui ont travaillé autour de moi pour la bonne ambiance qui a toujours prévalu dans notre

"bocal". Merci aussi à ceux que je n'ai fait que croiser pendant ce séjour au Québec, simplement pour avoir été là.

Pour finir, il me reste à faire à la fois les plus simples et les plus grands remerciements qu'il puisse se faire: à mes parents, pour leur présence à mes côtés même par delà les océans d'embruns ou d'éther, pour leur amour leur amitié et leur confiance de chaque instant.

TABLE DES MATIERES

LISTE DES FICHIERS ET DES REPERTOIRES DE LA DISQUETTE EN ANNEXE F iv

SOMMAIRE..... vii

ABSTRACT..... ix

REMERCIEMENTS..... xi

LISTE DES TABLEAUX..... xvi

LISTE DES FIGURES..... xvii

INTRODUCTION..... 1

1. LA SIMULATION DES SYSTEMES MANUFACTURIERS..... 3

1.1 Les systèmes manufacturiers..... 3

1.2 L'apport de la simulation à la gestion des systèmes manufacturiers..... 4

1.2.1 Vers une gestion en temps réel des systèmes..... 4

1.2.2 Les éléments d'un projet de simulation.. 15

1.3 Les principes de la simulation..... 18

1.3.1 Les différents types de simulation..... 18

1.3.2 Les langages généraux et les simulateurs..... 19

1.3.3 Les directions de recherche en simulation..... 21

2. PRESENTATION DU PROJET..... 31

2.1 Les buts recherchés..... 31

2.2 Les moyens..... 33

2.2.1 Le matériel et l'environnement informatique..... 34

2.2.2 Le logiciel MAP/1..... 36

2.3 La méthodologie utilisée..... 40

2.3.1 Les étapes du développement..... 40

2.3.2 Les étapes de la Conception Orientée Objet..... 42

3. LE DEVELOPPEMENT DE L'INTERFACE..... 45

3.1 Les principaux éléments du système..... 45

3.1.1 Présentation générale..... 45

3.1.2 Considérations ergonomiques..... 52

3.1.3 Modifications apportées à la modélisation MAP/1, hypothèses et limites..... 58

3.2 Conception préliminaire du système..... 66

3.2.1 Décomposition du problème..... 68

3.2.2 La base de données et les objets..... 75

4. COMMENTAIRES ET PROPOSITIONS D'EVOLUTION..... 78

4.1 Validation de l'approche informatique..... 78

4.2 Critique de l'interface..... 80

4.3 Axes d'évolutions futures..... 82

CONCLUSION..... 86

BIBLIOGRAPHIE..... 88

ANNEXE A: PRESENTATION DE FICHES GEREES PAR L'INTERFACE..... 94

ANNEXE B: DESCRIPTION DES FENETRES..... 97

ANNEXE C: LES INTERFACES DES OBJETS ET LE CONTROLEUR DE LA BASE DE DONNEES..... 104

ANNEXE D: LE CONTROLEUR D'ECRAN..... 122

ANNEXE E: MODE D'EMPLOI DU LOGICIEL..... 131

ANNEXE F: (Pochette) DISQUETTE CONTENANT LE CODE SOURCE DE L'INTERFACE ET LES BASES DE DONNEES

LISTE DES TABLEAUX

1.1 Les différents types de production..... 5

1.2 Comportements industriels et commerciaux traditionnels et nouveaux..... 7

1.3 Architecture d'un système intégré..... 7

3.1 Les combinaisons Etudes / Rapports MAP/1..... 49

A.1 Contenu des fiches de l'interface..... 96

LISTE DES FIGURES

1.1 Les étapes de la conception d'un système..... 9

1.2 Etapes types de l'implantation d'un nouveau système dans l'entreprise..... 10

1.3 Méthodologie d'innovation et de conception de produits..... 11

1.4 Intégration d'un projet de simulation..... 24

1.5 (a) à (c) Modélisation par objets en simulation.. 28

3.1 Symboles utilisés pour les graphes de produits... 51

3.2 Organisation de l'interface..... 53

3.3 Exemple d'utilisation des énoncés CONVEYOR, MERGE et DIVERGE..... 60

3.4 Hiérarchie des classes..... 67

3.5 Formalisation des classes principales..... 69

3.5 Formalisation des classes principales (fin)..... 71

3.6 Organisation de la base de données..... 76

A.1 Exemples de présentation de fiches..... 95

B.1 à B.6 Les fenêtres du logiciel..... 98 à 103

INTRODUCTION

La mondialisation des marchés et les bouleversements récents en Europe de l'Est mettent l'accent, si c'était encore nécessaire, sur l'apreté de la bataille que vont se livrer les grands blocs économiques par industriels interposés.

Dans un tel contexte de concurrence généralisée marqué par l'incertitude et par la disparition progressive de la "chasse gardée" que constituait souvent le marché national, les entreprises sont confrontées au défi de l'efficacité généralisée en termes de qualité, coûts et délais qui sous-entend une connaissance et un contrôle parfait de l'outil de production. Or, aujourd'hui encore, le rôle joué par les systèmes d'informations et les outils d'aide à la décision demeure faible et on remarque souvent l'inadéquation des moyens utilisés aux systèmes étudiés.

Au rang des outils de gestion et d'aide à la décision, il en est un dont l'usage reste paradoxalement limité malgré sa capacité de comparer des scénarios et donc de prédire l'avenir: la simulation. Pourquoi ce manque de popularité? c'est ce que nous allons tenter d'expliquer dans le premier chapitre de ce mémoire en nous intéressant tout

d'abord aux entreprises concernées et à leurs caractéristiques avant d'étudier plus en détail par la suite la simulation en elle-même.

Le deuxième chapitre présente la solution que nous avons envisagée pour favoriser l'usage de la simulation en milieu industriel ainsi que les moyens utilisés pour l'atteindre. Le résultat prend les traits d'une interface facilitant l'accès à un logiciel de simulation.

Le troisième chapitre, quant à lui, décrit les caractéristiques de l'interface, leurs justifications et présente l'organisation générale du logiciel.

Le quatrième et dernier chapitre réalise une critique a posteriori des moyens utilisés et de l'interface réalisée pour proposer finalement de nouvelles directions de recherche et de développement à partir du résultat de la présente recherche.

1. LA SIMULATION DES SYSTEMES MANUFACTURIERS: GENERALITES

1.1 Les systèmes manufacturiers

Le contexte de notre étude est celui des entreprises industrielles à vocation manufacturière. Ce type de société a pour finalité de produire des biens et des services et de les vendre avec le maximum de profit. Elles disposent pour cela de moyens d'étude, de production, de gestion, de commercialisation...

Nous nous intéressons plus particulièrement dans ce qui suit au système de production ou système manufacturier qui, comme son nom l'indique, est chargé de fabriquer les biens qui seront vendus. Les produits sont obtenus par la transformation des matières premières à l'aide de ressources (hommes et machines), suivant une séquence prédéterminée (ou gamme de fabrication). Cette décomposition nous permet donc d'isoler les composants fondamentaux d'un tel système, à savoir:

- les moyens de production (hommes et machines),
- les matières ou composants traités et produits,
- les informations relatives à l'état du système.

Il existe plusieurs types de production classés dans trois catégories (production continue, par lots, production discontinue). Une récapitulation des différents paramètres qui les caractérisent apparaît au tableau 1.1. Dans cette étude nous laisserons de côté les processus continus de type PROCESS (industrie chimique, raffinage, sidérurgie...) pour nous concentrer sur la production de biens dénombrables et plus particulièrement la production par lots qui concerne le plus grand nombre d'entreprises (70% des industries manufacturières en France [VOI 85]).

Avant d'introduire la notion de simulation pour de tels systèmes, il convient de s'intéresser aux contraintes et aux objectifs des gens qui en ont la responsabilité.

1.2 L'apport de la simulation à la gestion des systèmes

1.2.1 Vers une gestion en temps réel des systèmes

Dans la compétition internationale qui constitue désormais leur environnement quotidien, les dirigeants d'entreprises se doivent de modifier leur comportement et leur mentalité face à l'évolution du marché, des produits et des moyens industriels (voir tableau 1.2). Pro-

PRODUCTION	VALEUR AJOUTEE	VARIETE DE PRODUITS	TYPE DE GESTION DE PRODUCTION
PROCESS	FAIBLE	NULLE	CONTINUE; SPECIFIQUE
GRANDE SERIE (FLOW SHOP)	FAIBLE A GRANDE	TRES FAIBLE	CONTINUE; SPECIFIQUE A STOCK NUL
MOY. SERIE INTERMITT.	FAIBLE A GRANDE	FAIBLE A MOYENNE	SUR STOCK, A STOCK NUL, SUR PROGRAMME (GEST. PREVISION.)
PET. ET MOY. SERIE REPET. (JOB SHOP)	GRANDE	GRANDE	SUR PROGRAMME, A LA COMMANDE; GESTION PREVISIONNELLE
UNITAIRE OU PET. SERIE UNIQUE	TRES GRANDE	TRES GRANDE	SUR COMMANDE; GESTION D'ATELIER GESTION DE PROJET

Tableau 1.1: Les différents types de production

ductivité, fiabilité, performance et flexibilité de l'outil de production d'une part, innovation, modularité et évolutivité des produits d'autre part, sont des notions qui font dorénavant partie de l'univers quotidien des gestionnaires et qu'ils ne peuvent pas négliger sous peine de conséquences graves pour la survie de leur entreprise.

L'usine de demain, telle qu'on la décrit aujourd'hui, n'a plus pour seul rôle de produire des biens et des services, elle doit en plus assurer une certaine cohérence entre les différents objectifs et ce, malgré l'instabilité de son environnement. Pour la décrire, VOISIN [VOI 85] la présente comme une

> "coordination de styles, d'informations, de technologies, de structures physiques, humaines, organisationnelles, répondant à des objectifs variés, parfois contraires, face à des environnements incertains."

Relever le défi que pose ce concept d'usine modèle passe, entre autres, par l'intégration des moyens informatiques à tous les niveaux de l'entreprise (voir tableau 1.3). Ceci sous-entend aussi l'automatisation totale ou partielle du système de production mais aussi son optimisation en terme de délais, d'efficacité, de coûts et de qualité de produit. Le système doit aussi être capable de réagir rapidement, ce qui suppose que son pilotage se fait en temps réel.

COMPORTEMENTS COMMERCIAUX		MARCHE / PRODUIT	COMPORTEMENTS INDUSTRIELS	
NOUVEAU	ACTUEL		ACTUEL	NOUVEAU
Flexib. Extension du marché	Qualité/ prix Pénétrat. du marché	ACTUEL	Qualité/ coût Exploit.	Innovation Invest de productiv.
Flexib. + Innov. Diversif.	Innovation Développt du produit	NOUVEAU	Flexib. Adaptat.	Flexib. Innovation (re)conver.

Tableau 1.2: Comportements industriels et commerciaux traditionnels et nouveaux

	FLUX D' INFORMATIONS	FLUX DE MATIERES
ADMINISTRATION FINANCES (1)	ORDINATEURS CENTRAUX	MAITRISE DES COUTS NIVEAU DE SERVICE
GESTION INDUSTRIELLE (2)	ORDINATEURS CENTRAUX MINI ORDINATEURS	GESTION DES FLUX
C.F.A.O. (3)	MICRO-ORDINATEURS STATIONS DE TRAV.	MAITRISE ET OPTIMISATION DES FLUX PHYSIQUES
RESEAUX (4)	MICRO-PROCESSEURS TELECOMMUNICATION RESEAUX LOCAUX	PILOTAGE DU SYSTEME SYST. DE MANUTENTION CONDITIONNEMENT
INSTRUMENTATION (5)	CAPTEURS AUTOMATES PROG.	MAGASIN DISTRIBUTION PHYSIQUE

Tableau 1.3: Architecture d'un système industriel intégré

Outils de simulation: (1) <-> (5)

Organisation Flexible: (2) <-> (4)

Condition de travail et ergonomie: (2) <-> (5)

Dans cet ordre d'idées, il est souvent question d'atelier flexible. Dans la plupart de cas, sa conception ne part pas de zéro, mais réutilise en l'optimisant le patrimoine actuel: les figures 1.1 et 1.2 récapitulent brièvement les étapes de cette conception. Cependant, cette solution de l'atelier flexible, quand elle est adoptée, exige un lourd investissement et doit être précédée ou accompagnée de mesures de rationalisation (comme celles qui ont lieu lors de l'implantation de systèmes de production "juste à temps") de l'ensemble de l'appareil de production pour pouvoir en retirer tous les bénéfices. Cette rationalisation passe aussi par une définition de produits qui tient compte des facteurs énoncés en début de paragraphe (voir figure 1.3).

Ainsi, dans un tel contexte, les investissements concernant l'appareil de production font intervenir de nombreux critères simultanément et mettent en jeu des sommes importantes. En cas de mauvais choix, tant au plan tactique que stratégique, la survie même de l'entreprise peut être compromise, surtout celle des PME et des PMI.

On peut regrouper les contraintes et les objectifs qui influencent les décisions concernant l'appareil de production dans les catégories suivantes:

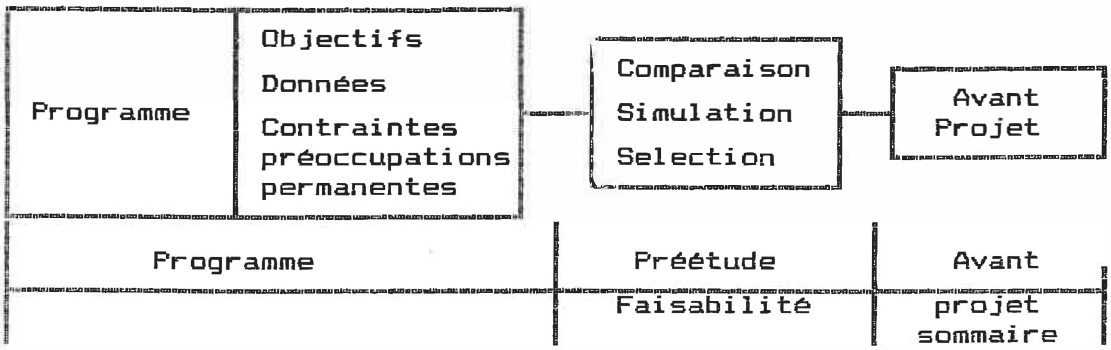
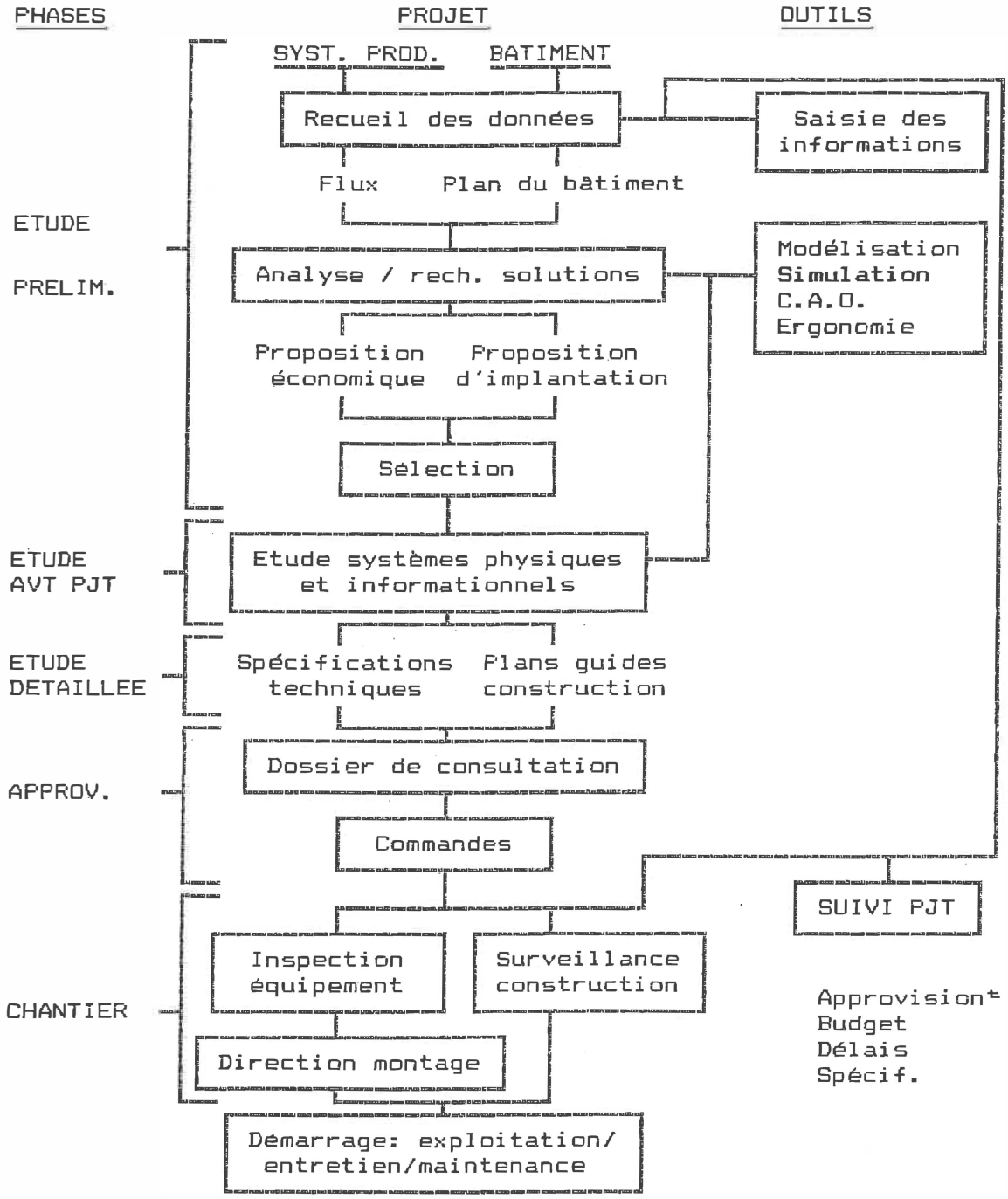


fig 1.1: Les étapes de la conception d'un système



PHASES 1 et 2 : CONCEPTION PHASES 3, 4 et 5: REALISATION

fig 1.2: Etapes types de l'implantation d'un nouveau système dans l'entreprise (d'après [VOI 85])

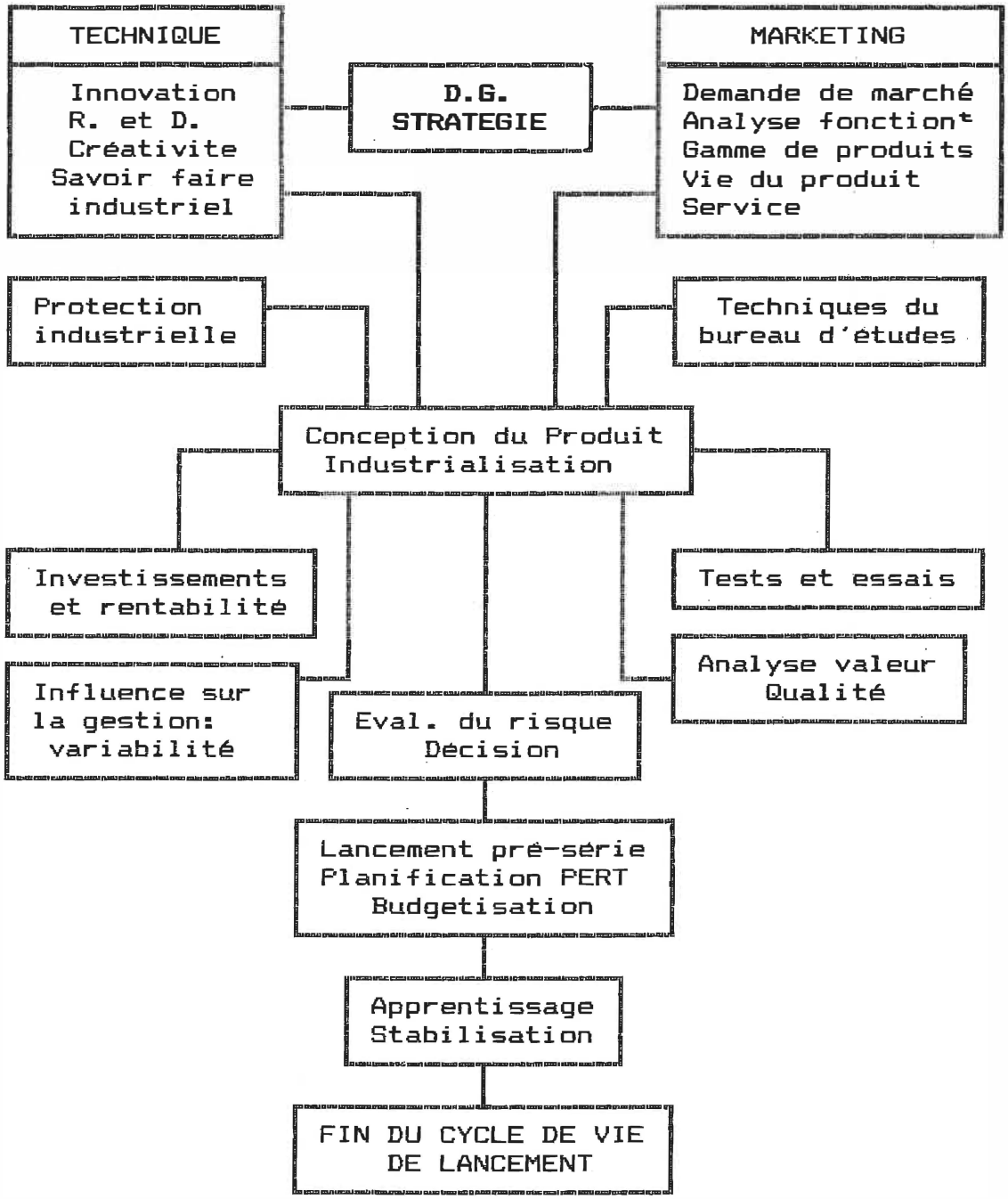


fig 1.3: Méthodologie de conception et d'innovation de produits (d'après [VOI 85])

- les contraintes techniques (nature des moyens, quantité de travail nécessaire par produit, caractéristiques des machines, liens entre les divers flux de fabrication...),
- les contraintes commerciales (caractéristiques du marché...),
- les contraintes financières (taux d'endettement...),
- les objectifs (optimiser les bénéfices en minimisant les coûts de production par augmentation de la productivité, diminution des en-cours, diminution des cycles de production, respect des délais contractuels...) [BEN 85].

Les dirigeants d'entreprises désireux d'implanter un nouveau système doivent donc réunir suffisamment d'informations pertinentes pour expliquer leurs choix aux dirigeants et pour assurer ensuite la bonne marche du système. Pour cela, ils doivent avant tout comprendre le mécanisme qu'ils sont censés contrôler. Il ne doit plus leur apparaître comme un enchevêtrement obscur d'éléments aux contours vagues mais comme un ensemble de composantes interagissantes et bien identifiées. Cette compréhension permet d'intervenir plus efficacement sur le système et de rencontrer plus aisément les objectifs de qualité définis en début de paragraphe. Il est donc primordial qu'ils disposent d'outils et

de méthodes permettant des réactions rapides pour analyser les informations sur le comportement dynamique du système et, si possible, capables également de prévoir l'avenir!

Parmi les techniques destinées à modéliser les différents niveaux d'un système manufacturier qui sont le plus fréquemment utilisées, il n'est certes pas question de boule de cristal et on peut citer brièvement:

- la modélisation mathématique,
- les réseaux de files d'attente,
- les graphes potentiel / tâches,
- les réseaux de Pétri,
- le GRAFCET,
- la simulation...

Les cinq premières constituent des approches analytiques, leur but est de définir un point optimum de fonctionnement, tandis que la simulation s'intéresse à l'évolution du système au cours du temps et non à un éventuel optimum.

Si l'approche analytique s'avère intéressante pour l'étude de processus continus et pour l'analyse grossière des systèmes manufacturiers (prédimentionnement), elle se révèle lourde, voire génératrice d'erreurs (dans le cas de la fabrication par lots, pour la détection des goulots

d'étranglement, pour tenir compte de règles d'ordonnancement et de pilotage basées sur l'observation de l'état du système) et devient vite insuffisante à un stade plus avancé de conception [BEL 85]. La simulation s'avère un outil plus général et plus flexible que les autres par ses capacités dynamiques qui éliminent ces lacunes [SUR 85]. Elle se prête ainsi aisément à l'étude de différents scénarios ("what if") et de leur impact respectif sur le système modélisé.

Dans le cadre des usines de demain, comme on a déjà pu le voir sur les figures 1.1, 1.2 et dans le tableau 1.3, la simulation s'impose comme un outil pratique et performant. Elle offre la possibilité de considérer tous les paramètres influents, y compris ceux que les méthodes analytiques laissent de côté.

On peut donc s'attendre à ce qu'elle prenne de plus en plus d'importance comme outil d'aide à la décision, tant au niveau de la conception d'un système de production que de sa gestion quotidienne.

Elle peut en effet être utilisée comme:

- un outil d'exploration pour définir un système,
- un outil d'analyse des composants critiques,

- un outil de conception pour synthétiser et comparer différentes solutions,
- un outil de prévision et de planification pour des développements futurs [PRI 86].

On peut ainsi l'employer avec bénéfice à divers stades de la vie du système:

- au moment du dimensionnement du système (étude de configurations d'atelier, évaluation de capacité des éléments du système...)
- lors de son optimisation et pour l'ordonnancement (choix du nombre de palettes et de chariots, test et validation de l'ordonnancement...)
- pour son contrôle et sa gestion en temps réel (étude de l'impact de modifications dans les plans de travail...) [BEN 85], [SHM 86], [LAW 89a].

1.2.2 Les éléments d'un projet de simulation

La simulation d'un système manufacturier s'opère par l'entremise d'un modèle chargé de représenter le dit système sous forme informatisée. Ce modèle se prête à des manipulations qu'il serait délicat voire même dangereux de réaliser sur le système réel (risques techniques ou

coûts encourus trop élevés, dangers pour les ouvriers, comportement inconnu...). L'objectif recherché lors de la réalisation d'une simulation peut être de différents types (évaluation globale du système, comparaison d'alternatives, étude de faisabilité, prévision, analyse de sensibilité, recherche des goulots, problème de remplacement de matériel, ordonnancement, étude du régime transitoire, optimisation [SHM 86], [PRI 86]...) et l'on peut ainsi connaître des propriétés concernant le système et son comportement sans avoir à le construire, le modifier ou le détruire. A la fin d'une simulation, qui peut comporter plusieurs exécutions du modèle (voir plus bas les étapes d'une expérience de simulation), l'utilisateur dispose de rapports sur les différents composants du modèle (donc sur ceux du système) durant le laps de temps simulé. L'interprétation de ces rapports permet de tirer les conclusions ou, pour le moins, des indications sur la validité des choix testés.

La simulation, certes efficace, n'en reste pas moins aujourd'hui encore une technique coûteuse en temps et en argent. Aussi, son utilisation doit être justifiée et sa mise en oeuvre doit être soigneusement planifiée. En général, on décompose une expérience de simulation en 10 étapes qui englobent la simulation proprement dite dans un processus de collecte (en amont) et d'analyse (en aval) d'informations pour aboutir à la prise de décision. Ces étapes

sont les suivantes:

- la formulation du problème et des objectifs,
- la construction du modèle,
- la collecte des données et l'estimation des paramètres,
- l'évaluation du modèle,
- la réalisation du programme,
- la validation du modèle par rapport au système réel,
- l'expérimentation (ou simulation) du modèle,
- la définition (planification) de l'expérience,
- l'analyse des résultats,
- l'implantation et la documentation.

Ces étapes sont répétées toutes ou en partie seulement jusqu'à obtention d'un résultat satisfaisant [PRI 86]. Les étapes les plus importantes, pour autant que l'on puisse établir une hiérarchie entre elles, sont celles de création du modèle (30 à 40% de l'effort total) et de définition de l'expérience (à rapprocher des objectifs définis au début).

LAW [LAW 89b] indique les principales erreurs à éviter et montre combien il faut être prudent lors de la réalisation d'une simulation. Cela nécessite une bonne connaissance des techniques utilisées et en premier lieu, bien entendu, des principes directeurs de la simulation.

1.3 Les principes de la simulation

1.3.1 Les différents types de simulation

Dans la suite, nous nous limitons à la simulation d'évènements discrets (discrete-event simulation) qui correspond à l'étude des systèmes de production de biens dénombrables.

Pour présenter en quelques mots la théorie de la simulation, il faut introduire les concepts d'évènement et d'activité. Le premier, le plus utilisé des deux, consiste à considérer par ordre chronologique les changements d'état du système, l'horloge de simulation permettant de passer d'un changement au suivant. Parmi les langages de simulation utilisant cette approche, on peut citer GPSS-FORTRAN, SIMSCRIPT, GASP IV.

L'approche par activité est la duale de la précédente et consiste à répertorier les différents types d'activités que l'on peut rencontrer dans le système. La modélisation du changement d'état se fait alors en spécifiant les conditions sur l'état du système nécessaires à la réalisation du début ou de la fin de chacune des activités. Le mécanisme

d'avance du pas d'horloge est dans ce cas fixe, ou variable comme pour le langage de simulation ECSL.

Si ces deux approches ont dominé les premiers travaux menés en simulation, ce n'est plus le cas aujourd'hui.

1.3.2 Les langages généraux et les simulateurs

En effet, avec l'apparition de nouveaux logiciels dans le courant des années 70, l'approche par processus a été mise en valeur. Elle consiste à regrouper la logique des changements d'état au sein de certaines séquences prédéterminées d'évènements (les processus). On choisit bien sûr des séquences qui sont répétées souvent au cours d'une simulation (files d'attente, postes de service...).

Cette façon de procéder favorise le rapprochement avec la structure du système réel. On peut citer parmi les langages utilisant cette notion SLAM II, SIMSCRIPT II.5, QNAP 2, SIMAN, SIMULA. L'approche processus facilite grandement la tâche de modélisation grâce à l'utilisation de réseaux (SLAM II, GPSS/H), néanmoins cette tâche reste ardue et, souvent, le résultat obtenu est difficile à appréhender par un non "initié" en simulation. Cela tient en grande partie au caractère général de ces langages qui sont utilisables

pour modéliser des systèmes de tout type [PRI 86; p68] et non pas uniquement les systèmes manufacturiers.

Pour tenter de pallier cet inconvénient et pour répondre à la demande des usagers, des langages de simulation à usage restreint, appelés aussi simulateurs, ont vu le jour. Ils sont conçus pour étudier et modéliser certains types de problèmes dans des secteurs particuliers. Ainsi, on peut citer FACTOR pour l'ordonnancement et le contrôle d'opérations, NETWORK II-5 pour les réseaux d'ordinateurs, SAINT, HOS et MOPADS pour les tâches d'opérateurs, MAST, GALS et MAP/1 pour la modélisation de systèmes manufacturiers [PRI 86], [LAW 89a]. Typiquement, les simulateurs sont basés sur des langages de simulation généraux qu'ils viennent parfois compléter (comme MAP/1 et SLAM II), langages pour lesquels l'orientation processus a été poussée plus loin encore pour rendre leur apprentissage et leur utilisation plus facile.

Cependant, ces logiciels ne sont pas suffisamment développés au niveau des interfaces avec l'usager et en ce qui concerne les moyens mis à sa disposition pour faciliter les étapes qui précèdent et qui suivent la simulation proprement dite qui sans cela peuvent se révéler fastidieuses. Leur utilisation réclame des connaissances dans de nombreux domaines comme la modélisation, la planification d'expérience, l'informatique, les probabilités, les statistiques,

le milieu que l'on veut étudier... Mis à part ce dernier élément, SHANNON [SHA 86] estime à 720 heures de cours et pas loin de 1400 heures d'apprentissage le temps nécessaire à l'étude et à la maîtrise des notions de base de la simulation. On voit là un inconvénient majeur qui limite l'utilisation de la simulation en milieu industriel.

1.3.3 Les nouvelles directions de recherche

Actuellement, les produits disponibles sur le marché ne rencontrent en effet qu'un succès mitigé. On leur reproche essentiellement leur abord difficile et, comme on vient de le voir, le manque d'intégration des différentes étapes de la simulation. De plus, leur prix relativement élevé (à l'achat et à l'usage) limite leur utilisation aux grandes entreprises ou aux sociétés de services en ingénierie et, en général, pour des projets de grande envergure.

Pour développer une utilisation plus massive de cet outil, on cherche à réaliser actuellement les simulateurs de la cinquième génération qui sont censés combler ces lacunes (du moins celle de l'intégration des étapes de l'expérience) grâce à l'apport de techniques comme l'intelligence artificielle (notamment les systèmes experts, les bases de règles, l'orientation objectif et l'orientation

objet), les interfaces graphiques, la communication en langage naturel, les outils de gestion de bases de données [SHM 86]...

R.E. SHANNON [SHA 86] les présente de la façon suivante:

> "l'usager introduit la connaissance sur le système (spécialement la description des objets), définit les objectifs et laisse travailler l'ordinateur pour trouver la solution...".

L'utilisateur définit le modèle dans un langage qui lui est propre, interagit avec le système et choisit le type et la forme des résultats désirés. Ainsi, plus besoin d'être un expert en simulation pour avoir accès à ce type de logiciel; un responsable de la gestion du système de production qui a un problème à résoudre peut directement l'utiliser sans avoir recours à un intermédiaire comme c'est le cas actuellement [TRE 88].

Ainsi, on s'oriente vers une solution où l'utilisateur reproduit élément par élément le système à modéliser, chaque objet physique correspondant à un objet informatique [MIZ 89] et [ROB 88], contredisant en cela l'approche traditionnelle de la simulation avec les langages d'aujourd'hui [LAW 86] où la phase de modélisation, dépendante du type de codage employé dans le logiciel, conduit à une

représentation "différente" du système étudié où cette correspondance n'existe que rarement. On met ainsi encore en évidence la condition sine qua non de la mise en oeuvre de la simulation qui est la bonne connaissance des composantes du système physique.

Pour atteindre le but fixé par SHANNON, deux orientations sont généralement choisies par les équipes de recherche:

- l'habillage des logiciels de simulation (fig 1.4),
- le développement de simulateurs plus performants grâce aux nouveaux outils informatiques.

Dans la première catégorie on trouve les interfaces graphiques comme TESS pour MAP/1, GPSS et SLAM II [STA 87], CINEMA pour SIMAN, BEAM pour MAST, Autogram pour GPSS/H; ces produits sont disponibles sur le marché. On peut aussi noter des études dans le domaine des générateurs de programmes de simulation (SPG) [HAD 87] qui font souvent appel à des notions d'intelligence artificielle et de gestion de bases de données [KET 89] pour assurer la réutilisation des modèles générés.

Ces travaux conduisent à des résultats intéressants, mais les produits demeurent liés aux caractéristiques des

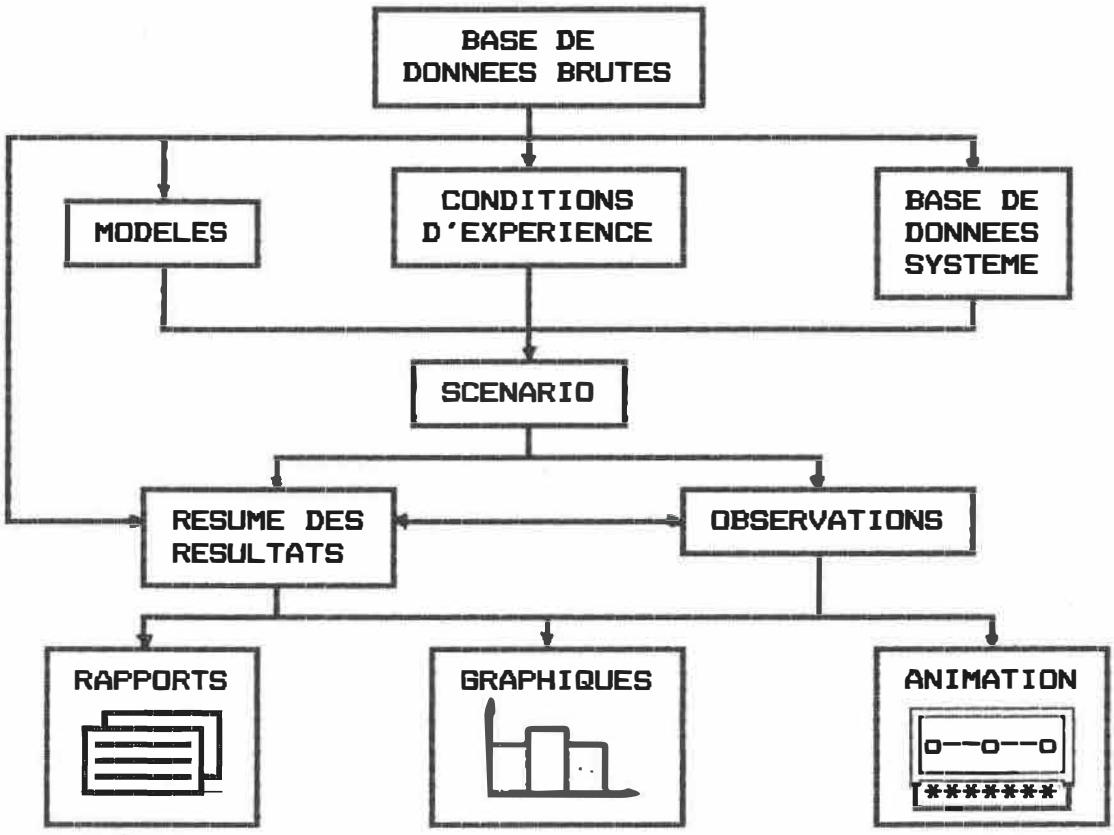


fig 1.4: Intégration d'un projet de simulation

langages de simulation qu'ils utilisent. Ainsi, la souplesse dont l'usager dispose est toute relative, notamment en ce qui concerne les aspects de réutilisation et de modularité des modèles.

La deuxième catégorie fait intervenir la Conception Orientée Objet (COO) ou Programmation Orientée Objet (POO). Cette notion -à la mode dans les milieux informatiques depuis quelques années- est héritée d'un logiciel... de simulation (!), SIMULA. Elle voit son utilisation se généraliser avec l'apparition d'environnements de travail associés aux langages de COO (EIFFEL, C++, SCOOPS, SMALLTALK 80). C'est, de l'avis de Judith JEFFCOATE [CAN 89], la voie à suivre pour l'avenir quel que soit le domaine de l'application à réaliser. La création de l'"Object Management Group" qui regroupe 10 des plus gros fabricants mondiaux d'ordinateurs (IBM, HP, SUN, PHILIPS...) tend à confirmer cet avis.

Son application à la simulation semble naturelle: les logiciels sont développés à partir de blocs de code transparents, organisés hiérarchiquement et réutilisables. Cette organisation facilite ainsi la modularité et la "cannibalisation" des modèles [SHM 86]. Bien sûr, ce concept de bloc n'est pas nouveau, c'est même là-dessus que reposent les logiciels de simulation aujourd'hui disponibles. La nou-

veauté réside dans le fait que l'usager peut créer et organiser ses propres blocs en utilisant, notamment, la notion d'héritage qui permet de conserver les propriétés de blocs situés plus haut dans la hiérarchie sans avoir à réécrire le code [BON 88]. Ainsi, il peut enrichir et construire sur mesure son logiciel de simulation. Cette technique offre en plus l'opportunité de simuler un modèle à différents niveaux de détails, selon les informations dont on dispose ou les résultats que l'on cherche à obtenir [MIZ 89] et [BON 88] et ce, à partir d'un même noyau d'objets. Les objets que manipulent ces systèmes sont de deux principaux types:

- ceux destinés à mettre en route le processus de simulation (calendrier, générateur de nombres aléatoires...),
- ceux qui sont particuliers au domaine étudié (machine, produit, matériel de manutention...).

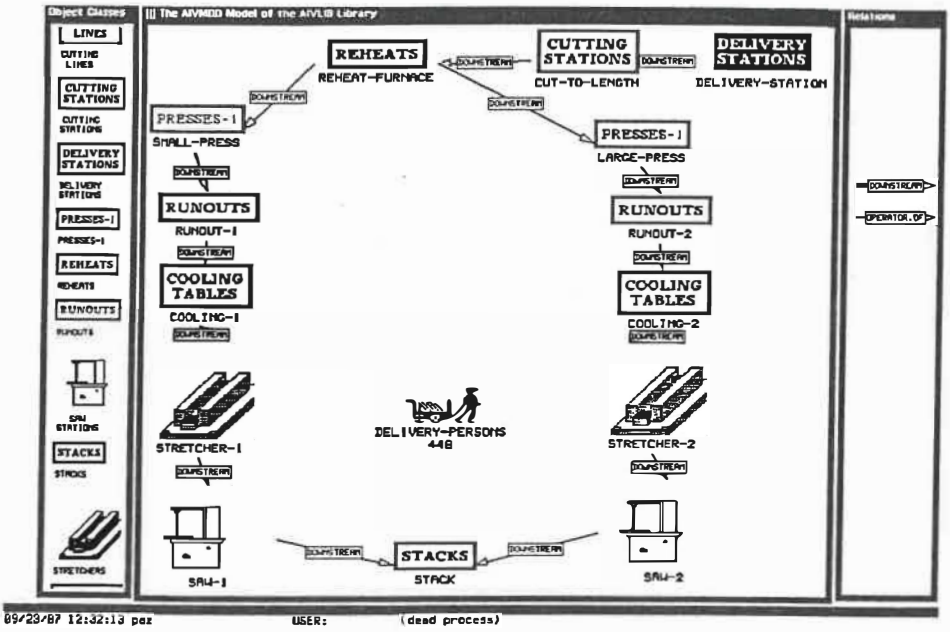
La librairie ainsi peuplée fournit les éléments pour construire des applications à architecture souple, décentralisée et modulaire (on peut rapprocher cette façon de procéder du concept de la brique LEGO). Elle autorise l'identification individuelle, si nécessaire, des objets manipulés: elle n'oblige pas une conception du modèle orientée vers les ressources ou bien orientée vers les pièces comme avec SLAM [KIN 86]. Du point de vue programma-

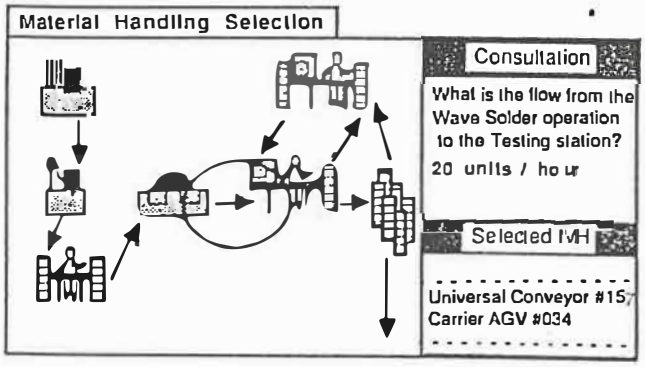
tion, une telle organisation facilite les mises à jour du logiciel et l'ajout de nouveaux éléments (classes).

Cette direction semble plus prometteuse que l'habillage de logiciels de simulation en raison de la souplesse et de la puissance qu'elle sous-entend. Elle est surtout intéressante lors de la phase de modélisation qu'elle simplifie en associant objet réel et objet informatique. Le dialogue avec l'usager devient plus convivial et plus direct, comme le montrent les figures 1.5 (a) [ZAL 88], (b) [KIN 86] et (c) [THO 88], où les objets apparaissent sous forme d'icônes à l'écran. D'un point de vue maintenance et mise à niveau du logiciel de simulation, cette conception modulaire évite de remettre en cause l'intégralité du système à chaque modification.

A l'heure actuelle, de tels systèmes ne sont pas encore commercialisés et tournent sur de "petites" applications en laboratoire. Ils souffrent comparativement aux logiciels traditionnels de simulation d'une tare assez fréquente dans les applications informatiques à usage industriel: une trop grande gourmandise en termes de temps d'exécution et d'espace mémoire.

A titre d'exemple, le simulateur développé par THOMAS-MA [THO 89] en SMALLTALK-V ne peut simuler en 30 minutes





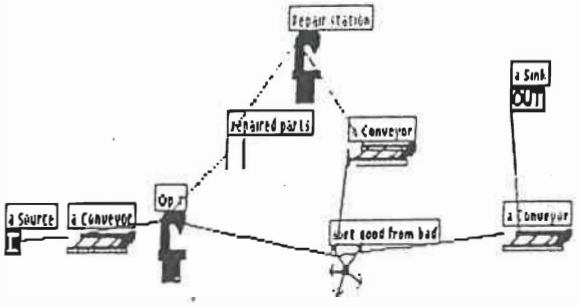


fig 1.5 (a) [ZAL 88], (b) [KIN 86] et (c) [THO 88]:

modélisation par objets en simulation

d'exécution que le tiers du temps étudié avec SIMAN sur la même application (12h33 contre 36h41). Cependant, cet écart n'est pas tout à fait significatif, la version de SMALLTALK utilisée ici est vieille (il en existe de plus puissantes) et le prototype n'est pas encore parfaitement optimisé du point de vue des structures de données manipulées.

Il faut aussi noter que les ordinateurs dont on dispose actuellement sont appelés à être supplantés, dans un avenir plus ou moins proche, par ceux de cinquième génération traitant l'information en parallèle (concept des réseaux de neuronnes). On peut prévoir une réduction notable du temps d'exécution des logiciels OO dont l'organisation non procédurale se prête à ce genre de manipulation.

Dans un futur immédiat, THOMASMA [THO 89] suggère d'utiliser des post-processeurs traduisant les applications orientées objet dans un langage de simulation tel SLAM II, SIMAN... Il s'agit en quelque sorte d'une voie intermédiaire entre les deux orientations présentées ci-dessus (l'habillage des logiciels existants et le développement de nouveaux simulateurs) qui permet de profiter en partie des avantages de la représentation par objets tout en diminuant ses inconvénients. C'est cette voie que nous avons choisie d'explorer.

En effet, elle nous semble préférable à l'orientation orientée objet "totale" car on peut ainsi réutiliser les composantes qui gèrent le processus de simulation dans les systèmes actuels et dont la qualité est assurée. Ceci représente une économie en termes de développement qui nous permet d'apprécier plus rapidement le logiciel ainsi conçu et qui nous permet aussi de mettre l'accent sur le côté convivial de la communication avec l'usager. Elle nous permet enfin de bâtir les bases d'un système qui, si les résultats sont satisfaisants, serviront au développement progressif d'un simulateur orienté objet à partir de cet acquis.

2. PRESENTATION DU PROJET

2.1 Les buts recherchés

Compte tenu des remarques précédentes, le premier but de ce travail est de rendre plus "convivial" un logiciel de simulation afin de faciliter son usage par des personnes peu sensibilisées à l'informatique et dont la connaissance porte sur l'environnement industriel dans lequel elles évoluent et pour lequel elles utilisent la simulation. En d'autres termes, l'objectif recherché est de réduire le temps d'apprentissage du logiciel et les connaissances requises pour la réalisation de la simulation, particulièrement pour la phase de modélisation.

Il existe une autre étape qui n'apparaît pas dans la liste présentée au chapitre précédent et qui est néanmoins très importante: la maintenance et la mise à niveau du modèle. En effet, si la dixième étape ressemble au nec plus ultra du développement du logiciel ce n'est pourtant pas le cas. Au delà, toute modification apportée à la structure du système modélisé (moyens de production ou produits) demande souvent d'intervenir de façon importante sur le modèle qui existe déjà ou même d'en développer un nouveau.

Il est facile de modifier certaines données comme les temps d'opération et plus généralement les données chiffrées, il est cependant plus délicat d'intégrer une modification dans la description structurale du processus ou du matériel. Pour cela, le système développé doit faciliter la mise à jour rapide des composantes d'un modèle dans la base de données.

L'utilisateur visé est un ingénieur qui utilise le système (ou "hands-on user" [PRI 86]) ou encore un preneur de décision ayant des connaissances techniques (ou "renaissance decision maker" [PRI 86]) auquel on veut offrir une plus grande interaction avec le logiciel de simulation en lui permettant de réaliser lui-même ses modèles.

Pour atteindre cet objectif on peut isoler les points suivants:

- simplifier la communication homme / ordinateur,
- réduire la phase de codage en favorisant la manipulation d'icônes ou d'objets graphiques,
- améliorer la documentation de la simulation,
- améliorer la gestion des données (librairies...),
- utiliser au maximum la dialectique industrielle de préférence à celle des informaticiens...

En parallèle et conformément à ce qui a été dit au paragraphe précédent, il apparaît intéressant de valider l'approche orientée objet en simulation. Pour cela, on souhaite constituer un noyau d'objets standard à partir duquel il sera possible de faire évoluer le logiciel en rajoutant au fur et à mesure de leur développement de nouveaux éléments.

2.2 Les moyens

Nous avons choisi les éléments suivants:

- le logiciel MAP/1 [MIN 86] dans sa version opérationnelle sur station de travail SUN,
- le système d'exploitation UNIX [CHR 88] qui est automatiquement associée aux stations SUN,
- le système de construction de fenêtres X Windows,
- le langage de POO C++ [WIE 88].

Tous ces éléments sont développés à partir du langage C à l'exception de MAP/1 qui a été conçu en FORTRAN. Les paragraphes suivants présentent brièvement ces produits ainsi que les raisons qui ont entrainé leur sélection.

2.2.1 Le matériel et l'environnement informatique

Les stations de travail (workstations) du type des stations SUN voient leur utilisation se généraliser dans les milieux industriels. Elles offrent plus de souplesse et de rapidité que les gros ordinateurs (mainframe) et plus de puissance que les micro-ordinateurs de type PC. Le développement sur ce type de matériel garantit un bon niveau de diffusion et suffisamment d'espace pour réaliser des simulations de taille respectable.

Pour la partie graphique de l'interface (fenêtres, icônes...), X (X window systems programming, développé conjointement par HP et le MIT) s'impose par rapport aux outils propres aux stations SUN (librairie SUNVIEW) en raison de sa portabilité et des éléments qu'il propose pour définir un environnement multi-fenêtré de façon simple.

Xlib [NYE 88] et Xt Intrinsics [YOU 89] sont les deux librairies de composants et de fonctions au standard X Windows que nous utilisons pour développer le logiciel. La librairie Xt Intrinsics est conçue suivant les principes de la COO et propose des éléments simples, facilement modifiables et réutilisables pour notre application. Tous ces éléments sont écrits en langage C, ce qui garantit l'homogénéité du logiciel.

UNIX est le système d'exploitation associé notamment aux stations SUN et à X. Il est largement diffusé (90% des universités américaines en ont une licence) et reconnu pour sa qualité. Il donne accès à des moyens puissants (outils, bases de données et librairies) pour gérer une application et organiser les fichiers qui s'y rapportent. Il permet aussi de demander et de diffuser aisément des informations grâce à son système de courrier électronique. On pourrait toutefois lui reprocher le manque de clarté des instructions, mais l'usager n'a théoriquement pas besoin de les utiliser, notre logiciel assurant la gestion de l'espace de travail.

UNIX et C sont des produits du même laboratoire (ATT-Bell) et sont parfaitement compatibles: leur association est naturelle car UNIX est écrit en C.

Ce même laboratoire est en partie à l'origine de la COO, ses produits sont réalisés en tenant compte de ce mode de programmation. C++ est une version optimisée pour la COO de C dont il conserve la puissance et la flexibilité auxquelles vient s'adjoindre une plate-forme supportant un haut niveau d'abstraction. C++ peut être utilisé comme un langage orienté objet pur ou comme un langage procédural (un C amélioré en quelque sorte!).

2.2.2 Le logiciel MAP/1

Le choix du logiciel de simulation devant servir de base à notre système s'est effectué entre les langages SLAM II et MAP/1 de la firme PRITSKER & ASSOCIATE. Le premier de par son caractère général et le type d'instructions manipulés est certes plus flexible que MAP/1 pour la diversité des systèmes qu'il permet de simuler, mais d'un accès plus complexe. Aussi, dans le cadre restreint de la simulation des systèmes manufacturiers et parce que notre but est de réaliser rapidement un prototype, notre choix s'est porté sur MAP/1.

Dans la suite, il sera également fait référence à AutoMod II [AUT 88] à titre de comparaison critique avec SLAM II associé à TESS et avec la version de MAP/1 agrémenté de l'interface développée. AutoMod II n'a pas été pris en considération dans le choix initial du logiciel car il n'était pas disponible sur le site de développement à ce moment là.

L.J. ROLSTON [ROL 85] présente MAP/1 comme un simulateur

"conçu pour être utilisé par des ingénieurs et par le personnel technique impliqué dans la conception et l'usage de systèmes manufacturiers de production discrète. Il peut être utilisé tant pour le développement de nouveaux systèmes que pour évaluer des changements apportés à des systèmes existants (...).

Le simulateur MAP/1 permet d'effectuer différents types d'analyses, entre autres:

- identification de goulots,
- planification de capacité,
- ordonnancement,
- mesure d'inventaire".

A l'origine, MAP/1 est destiné à des usagers qui n'ont pas forcément d'expérience en informatique ou en simulation. Il est relativement facile à utiliser par rapport à un langage de simulation général en raison justement de sa spécialisation. Toutefois, son apprentissage pour en retirer tous les bénéfices réclame un certain temps. La difficulté provient de la façon dont sont écrites les instructions avec MAP/1: chaque commande se présente sous la forme d'une séquence commençant par un mot clé qui identifie le type d'élément décrit, suivi de champs contenant les informations sur cet élément et qu'il faut remplir dans un ordre rigoureux. Il est difficile de connaître tous ces champs, ce qui complique la phase d'apprentissage et entraîne souvent des erreurs de codage ou de modélisation (informations mal placées ou manquantes...). En cela, il se rapproche de SLAM II.

Pour faciliter la tâche de codage, PRITSKER & ASSOCIATE propose une interface appelée IIS (Interactive Input System) qui présente à l'usager des menus et des feuilles standard d'instructions où il remplit des champs laissés blancs ou occupés par une valeur par défaut. Bien que cette interface facilite la tâche de modélisation, surtout au niveau de la mémorisation des instructions de MAP/1, l'ensemble reste toutefois assez fastidieux à utiliser -il est facile de se perdre dans les commandes de gestion d'IIS et dans les fiches car l'on n'a pas une vue d'ensemble bien claire du modèle et la progression ou la régression dans la hiérarchie s'avère vite lourde. Cette interface laisse donc la place pour des améliorations notamment au niveau de la convivialité.

Il est également possible d'utiliser TESS, une autre interface développée par cette même firme pour tenir compte des problèmes de communication avec l'usager et pour intégrer les concepts énoncés au paragraphes 1.3.2 et 2.1, mais elle ne supprime rien à la phase de codage et oblige au contraire à rajouter des nouvelles instructions qui sont elles aussi peu explicites pour un usager non initié. Cette interface se rapproche toutefois plus de celle que nous réalisons car elle est définie sur le principe de la gestion d'une base de données composée de modèles.

Il manque aussi à MAP/1 un support pour documenter un projet de simulation, pour en présenter les résultats ainsi que pour proposer le modèle autrement que sous sa forme codée en MAP/1. Cette forme-là est le plus souvent incompréhensible pour les gens intéressés d'abord par les résultats de l'application. Souvent, la solution de ce problème consiste à faire développer le modèle par un "spécialiste maison" de la simulation. Le modèle est ensuite validé par les intéressés en fonction des résultats qu'il fournit sur un cas test, puis stocké en mémoire. Il peut s'avérer difficile à modifier ou à maintenir par quelqu'un d'autre que le concepteur et devenir à la longue un objet statique ne correspondant plus à la réalité du système.

Ces remarques s'appliquent, on l'a vu, à la plupart des logiciels de simulation et pas spécifiquement à MAP/1. Il est bien évident que ce langage ne comporte pas que des inconvénients. Au nombre de ses avantages citons d'abord le faible nombre d'instructions, dont les noms se rapportent à ceux connus dans les milieux industriels (PART, STATION, CONVEYOR, FIXTURE, OPERATOR...). Ces instructions se rapprochent ainsi des objets qui sont manipulés en COO. Vient ensuite son caractère non procédural qui autorise la création non séquentielle du code (excepté quelques instructions qui doivent apparaître avant d'autres, voir [VIL 89]) ce qui laisse une grande liberté à l'usager et favorise la

modularité du modèle suivant des critères propres au système étudié et non pas fonction du support informatique.

Pour finir, l'usager a la possibilité de créer ses propres fonctions et ses propres rapports en générant des routines FORTRAN. On peut penser notamment à des fonctions de calculs économiques et financiers (basées sur les renseignements obtenus au cours et à la fin de la simulation) et qui font actuellement l'objet d'un projet dans notre équipe [TEL 89]. Cette option permet d'obtenir directement les indicateurs pertinents pour prendre une décision.

2.3 La méthodologie utilisée

2.3.1 Les étapes du développement

La méthode de développement choisie s'inspire de la décomposition en cascade qui est fréquemment utilisée en génie logiciel mais qui a été simplifiée pour répondre aux besoins de la réalisation d'un prototype. Les principales étapes que l'on rencontre sont donc les suivantes:

- définition du projet et spécifications,
- conception préliminaire et détaillée,
- codage et tests,

- validation,
- conclusion.

Les deux premières étapes sont celles où se définissent et se raffinent progressivement les recettes qui vont constituer le système final. Elles nécessitent une très bonne connaissance des objectifs du projet, de MAP/1 et une bonne connaissance des outils informatiques disponibles.

Elles sont réalisées "manuellement", le passage à l'informatique se fait au niveau de la conception détaillée qui génère les en-têtes des différents modules du logiciel et qui sert aussi à documenter le code qui vient à sa suite.

Au niveau de la troisième étape, il n'est plus nécessaire de connaître la finalité du logiciel, on s'intéresse uniquement à la solution informatique qui donnera la bonne fonctionnalité à chaque module, conformément aux résultats des phases précédentes.

L'avant-dernière étape confronte le produit obtenu et les spécifications du début. Une fois le produit validé, la dernière étape oriente les recherches à venir.

2.3.2 Les étapes de la conception orientée objet

La définition et la spécification se font en langage naturel à partir des buts vu au paragraphe 2.1.

La conception préliminaire consiste à définir les objets qui vont intervenir dans le logiciel ainsi que leur organisation, leur fonctionnalité et leurs interfaces. La décomposition est descendante et s'arrête à un niveau "raisonnable" de détail. A la fin de cette étape et au cours de la suivante et suivant les besoins, certains objets peuvent être rajoutés par rapport à la configuration originelle.

La conception détaillée reprend les objets de la conception préliminaire et en précise le contenu, les redécoupe (si besoin est) et regroupe certains des modules élémentaires qui apparaissent, s'ils remplissent la même fonction. Chaque objet est décrit à ce niveau par sa fonctionalité, ses interfaces et par les algorithmes de ses méthodes.

La phase de codage consiste à traduire en langage informatique les algorithmes écrits précédemment. Quand commence cette phase, il n'est idéalement plus nécessaire de modifier la fonctionalité des objets qui ont été définis, tout comme il n'est pas souhaitable d'en rajouter.

Chaque module est testé individuellement puis le système est testé dans son ensemble afin de vérifier la valeur du code. Si tout va bien, après cette étape le logiciel est opérationnel. Néanmoins, il est rare qu'il en soit ainsi dans la pratique dès le premier essai, ce qui signifie qu'il est raisonnable d'envisager à ce niveau et lors de l'étape suivante des modifications mineures.

Arrivé à la phase de validation, on est en présence d'un logiciel qui, informatiquement parlant, fonctionne correctement, il convient alors de vérifier sa capacité à atteindre les objectifs établis lors de la phase de définition et de spécification et d'apporter éventuellement les corrections qui s'imposent.

Au moment de sa création, chaque module est décomposé de la façon suivante:

<u>Définition du problème</u>

- énoncer le problème en une ou plusieurs phrases,
- analyser et clarifier les données.

Cette étape fait référence à la spécification et permet d'écrire et de justifier les choix de conception.

Stratégie informelle

On décompose le problème que doit résoudre le module en sous-problèmes décrits sous forme de phrases simples où les verbes en caractères gras correspondent aux actions à prendre sur les objets soulignés.

Formalisation du problème

A partir de la stratégie informelle, on schématise le problème en représentant sous forme de boîtes les différents objets. Les verbes correspondant aux actions (ou méthodes en COO) apparaissent comme les interfaces des dits objets.

Après cela, pour chaque objet, on fait la description sommaire des interfaces, des variables qu'elles font intervenir et de leur rôle. Cette decription est informatisée et constitue l'armature de départ pour la conception détaillée et le codage qui apparaissent à la suite.

3. LE DEVELOPPEMENT DE L'INTERFACE

3.1 Les principaux éléments du système

3.1.1 Présentation générale

Cette interface au logiciel de simulation MAP/1 est destinée à en faciliter l'accès et s'ajoute aux moyens de programmation MAP/1 déjà existants (les fiches de IIS, l'interface TESS ou bien directement le langage MAP/1) en offrant une nouvelle alternative de saisie et de gestion des informations. Pour rester homogène avec ces produits, elle est développée en anglais.

La décomposition de la modélisation en quatre étapes distinctes en est le principe de base. Elle repose sur la constatation faite au paragraphe 1.1 concernant les composantes d'un système manufacturier: celui-ci se décompose en une partie ressources, une partie produit, une partie information; la partie ressources comprend les entités permanentes du système alors que la partie produit décrit les entités temporaires [KET 89]. Pour que le modèle développé tienne compte de cette réalité, le logiciel doit pouvoir faire apparaître directement chacun de ces éléments.

Les instructions standard de MAP/1 se prêtent à une telle partition, la partie information se "réduisant" dans le cadre d'une simulation à la définition des objectifs de l'étude lors de la phase de modélisation et aux rapports générés par le simulateur MAP/1. Finalement, on rajoute une quatrième partie servant à identifier l'expérience et les conditions initiales. Ces quatre phases indépendantes (voir exceptions au paragraphe 3.1.3) sont donc les suivantes:

- description des conditions initiales de la simulation (correspondant aux énoncés BEGIN, CLEAR...),
- description des objectifs de la simulation et choix des rapports à générer en fonction du type d'étude désiré, du régime permanent du système et du laps de temps simulé (énoncés REPORT, DEFINE REPORT...),
- description des moyens industriels disponibles et si besoin est de leur disposition (énoncés STATION, TRANSPORTER, CONVEYOR, MERGE...),
- description des produits fabriqués dans le système en utilisant comme support des graphes de déroulement matière ou "process charts" (énoncés PART, SCHEDULE, ROUTE...),

A l'intérieur des quatre étapes, l'usager dispose d'une grande autonomie pour faire la description de son modèle, tout en étant guidé dans ses choix par des rensei-

gnements contenus dans les différentes fenêtres. La description d'un modèle se fait à l'aide de fiches comme c'est le cas avec IIS. Cependant, elles sont différentes de ces dernières et correspondent souvent à des regroupements d'instructions de MAP/1. Elles s'accordent au schéma de description en quatre phases et à l'utilisation de graphes produits. L'annexe A fait la présentation de quelques unes de ces fiches.

Le concept de description modulaire permet la réutilisation séparée des composantes du modèle et aussi d'étudier la fabrication de différents produits (pour une configuration d'atelier donnée) ou pour un même produit d'évaluer plusieurs ateliers envisageables ou encore, d'évaluer une usine suivant plusieurs points de vue (ou objectifs)... Elle facilite aussi la maintenance du modèle en donnant un accès individuel et direct à toute l'information concernant un élément du modèle.

Cette modularité se retrouve sous une forme différente dans un logiciel comme AutoMod II qui est lui aussi destiné à modéliser des systèmes manufacturiers. Là, elle porte essentiellement sur la description des ressources et sépare par exemple, le système de manutention du reste des moyens industriels en décomposant la modélisation en deux étapes distinctes. S'il autorise dès le début la spécification des

différents objets qui composent le système, tant les moyens industriels que les produits, AutoMod II n'intégre pas directement les gammes de fabrication des pièces par le biais des graphes de déroulement matière, la description se fait de façon fragmentée et codée dans des objets appelés "process". On ne peut pas de ce fait panacher des modèles directement, il faudrait avoir recours aux manipulations de fichiers sous UNIX. Cela peut s'avérer compliqué et hasardeux pour une personne peu familière avec le système UNIX et, a fortiori, avec ce logiciel de simulation et la manière dont il organise ses fichiers en base de données.

Le choix d'un type d'étude se conforme aux possibilités de MAP/1 telles qu'elles ont été énoncées au paragraphe 2.2.2. Pour chacune d'elles, nous avons défini un choix type de rapports générés par MAP/1 permettant d'obtenir les renseignements adéquats. Le tableau 3.1 présente les différents types d'études et les rapports de MAP/1 qu'elles incluent. Cette façon de procéder donne accès rapidement aux renseignements qui seront utiles à l'usager.

Il peut également créer de nouveaux types d'études s'il en éprouve le besoin. Ces derniers sont rajoutés à la liste des rapports standard mais, contrairement à ceux-ci qui sont inamovibles, il est possible de les effacer.

	PRD	CST	THP	UTI	OVT	DWT	INV	TIM
CONCEPTION				*			*	*
PERSONNEL				&	*			
ORDONNANC^{t}	*			*			*	*
CAPACITE				*	*		*	
PROD. MIX			*					*
APPROVIS^{t}			*					*
CHOIX DE MANUTENTION			&	&			*	*
GOULOTS	*	*	*		*			
ESPACE DE STOCKAGE				&			*	
FIABILITE				*		*		
PRODUCTIVITE			*		*			*
DELAIS	*							*
REGIME PERM.			*	*				

Tableau 3.1: Les combinaisons Etudes / Rapports MAP/1

*: rapport complet

&: rapport partiel (i.e. avec un champ de type INCLUDE ou EXCLUDE)

La description des produits fait appel à des icônes pour représenter les différents éléments. Ces icônes s'inspirent de celles définies dans le standard ASME 101 [ASM 47]. Toutefois certaines ont été rajoutées, comme le montre la figure 3.1, pour tenir compte de la spécificité de MAP/1 et pour rationaliser les séquences d'icônes (par exemple, deux opérations distinctes sont toujours séparées par un transport ou un retournement, ce qui n'est pas obligatoire dans le standard ASME). L'idée qui motive ce type de représentation est de rendre le modèle visible et compréhensible "d'un seul coup d'oeil" tout en utilisant un symbolisme largement reconnu. Le nombre d'icônes est très réduit ce qui facilite leur mémorisation.

Les différentes parties d'un modèle sont stockées dans une base de données indépendamment du modèle pour lequel elles ont été créées. Cela facilite leur réutilisation de façon séparée conformément à ce que propose PRITSKER [PRI 86; chap. 17] dans sa description du logiciel de simulation de nouvelle génération. L'usager peut ainsi constituer de nouveaux modèles en combïnant différents éléments créés auparavant pour d'autres études sans pour cela avoir besoin de recommencer la modélisation.

L'exécution de la simulation et la production des rapports sont laissées sous la responsabilité de MAP/1. L'usa-

COMPOSANT :



PRODUIT :



OPERATION :



OPER. D'ASSEMBLAGE :



OPER. DE PRODUCTION :



STOCK :



TRANSPORT :



RETOURNEMENT :



FIN DE PROCESSUS :



fig 3.1: Symboles utilisés pour les graphes de produits

ger peut ensuite faire imprimer ces rapports d'exécution qui apparaissent dans un fichier portant le nom du projet suffixé par ".out".

La figure 3.2 enfin, présente grossièrement les différents composants de l'interface.

3.1.2 Considérations ergonomiques

Il existe un nombre considérable de moyens de communication entre un usager et un logiciel. Pour éviter un foisonnement qui serait nuisible à la rétention de l'interface, les moyens choisis sont volontairement limités et prennent en considération les possibilités offertes par la station SUN, les éléments gérés par X (pointeur, multi-fenêtrage...) et les remarques de B. SHNEIDERMAN [SHN 87] concernant la conception d'interfaces et les avantages et inconvénients de ces moyens. Le choix d'un médium s'est fait en fonction des critères suivants:

- facilité d'accès, de compréhension et de mémorisation (si besoin est),
- rapidité d'invocation d'une action (nombre de paramètres limités à un ou deux maximum).

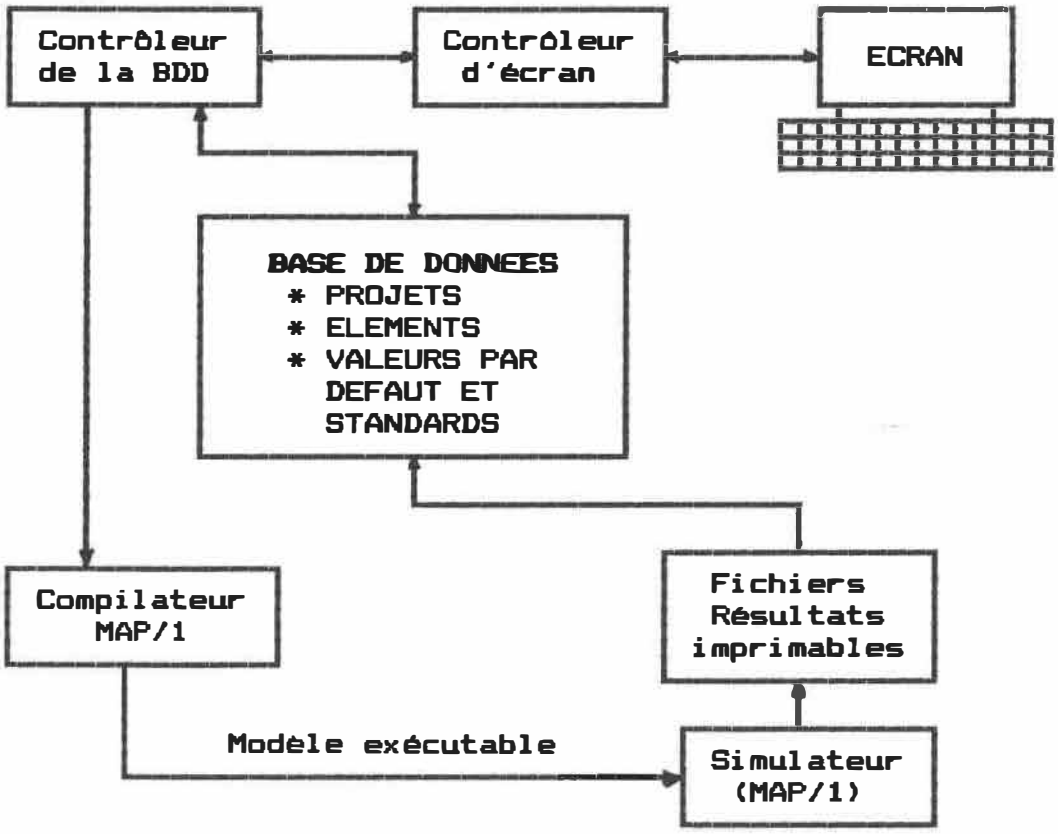


fig 3.2: Organisation de l'interface

L'ensemble de moyens retenus est donc le suivant:

- des **fenêtres** destinées à donner accès au logiciel et aux éléments correspondant à chaque phase de modélisation,
- des **boutons** affichés dans un panneau en haut des fenetres pour les commandes de base et affichés dans un panneau à gauche pour les composants,
- le **pointeur** (ou souris) pour "cliquer" les boutons, positionner le curseur ou choisir un élément dans une liste,
- le **clavier** enfin, pour entrer les données.

La majorité des actions se font grâce au pointeur et ne nécessitent pas l'utilisation du clavier dont l'usage est limité à la saisie des paramètres alphanumériques du modèle (c'est-à-dire au remplissage des fiches et, de manière alternative, au choix d'un élément lors de la construction d'un projet) tandis que la gestion du système se fait grâce aux boutons des fenêtres qui sont accessibles par le biais du pointeur et de ses boutons.

Cette organisation avec des boutons fixes entraîne un surplus de mouvements du pointeur par rapport à celle faisant intervenir des menus sous forme ponctuelle ("popup

menus") et peut occasionner une certaine fatigue à l'usager dans le cas d'une application de grande taille ou bien, comme c'est le cas avec AutoMod II mais pas dans nôtre interface, quand une action répétitive demande de répéter à chaque fois la même (longue) séquence de sélections à l'aide du pointeur.

Cette hypothétique lassitude est compensée par le fait que toutes les informations sur les possibilités offertes sont en permanence affichées à l'écran, ce qui est très pratique pour un usager familier ou non, alors qu'elle ne le sont que de façon fugitive dans le cas des menus (et parfois ne correspondent pas à celles désirées suivant l'endroit où se trouve le pointeur lors de l'invocation du menu). L'utilisateur peut ainsi se concentrer uniquement sur l'objet de sa simulation: il n'a pas besoin de retenir par coeur les commandes puisqu'elles sont affichées en permanence dans son environnement de travail. Dès lors, un soin particulier est apporté au choix du nom de ces commandes pour empêcher les confusions toujours possibles et faire en sorte qu'à une action donnée corresponde le même nom de commande quelle que soit la fenêtre active.

Toutefois, l'utilisation de menus ponctuels est requise quand il convient de spécifier une option pour une commande donnée. C'est le cas quand l'usager désire obtenir la

liste des moyens industriels d'un type donné; la sélection du bouton "#RESOURCES#" provoque alors l'apparition des différents types de ressources (STATION, TRANSPORTER, CONVEYOR, STOCK, PERSONNEL) sous forme de menu. Nous reviendrons plus en détail sur le rôle particulier de ce bouton.

Conformément à ce qui se fait habituellement, les commandes de destruction ou d'effacement d'un élément provoquent l'apparition d'un panneau de mise en garde demandant la confirmation ou l'infirmation de l'action par le "cliquage" d'un bouton sur l'écran. L'apparition d'une telle fenêtre s'accompagne d'un "bip". Un "bip" est également émis quand l'usager invoque une commande invalidée pour le type d'élément qu'il a choisi, l'invitant à reformuler son choix. Dans ce cas-là, puisqu'il ne s'agit pas d'une action "dangereuse", aucun message n'apparaît pour ne pas ralentir l'utilisateur dans son travail.

Pour ce qui est de la configuration de l'écran, on utilise une fenêtre différente pour chacune des 4 parties du modèle, plus une cinquième, dite générale, activée lors de l'invocation de l'interface et permettant de débuter le dialogue et de le clore à la fin d'une session de travail. Il n'y a au maximum que 3 fenêtres ouvertes simultanément à l'écran: la fenêtre générale qui est active en tout temps, éventuellement celle qui correspond à la partie du modèle

décrite et celle où se fait le remplissage d'une fiche. Par instant, l'emploi de certaines commandes provoque comme on vient de le voir l'apparition de sous-fenêtres de mise en garde ou d'assistance qui disparaissent automatiquement après la réponse de l'usager.

Cette limitation du nombre de fenêtres se justifie car on remarque souvent une certaine lourdeur dans la manipulation d'un environnement multi-fenêtré comme celui que propose par exemple AutoMod II: une trop grande prolifération de fenêtres et une organisation de l'écran un peu anarchique entraînent de nombreux déplacements inutiles du pointeur et, d'un autre côté, tendent à égarer l'usager dans la hiérarchie et à l'intérieur même des fenêtres. Aussi, les quatre fenêtres de description du modèle, ont une allure et une organisation qui restent identiques (dans la mesure du possible), il en est ainsi également pour les fenêtres temporaires qu'elles génèrent le cas échéant (l'annexe B décrit plus en détail toutes ces fenêtres). Les constantes de cette organisation sont les suivantes:

- une sous-fenêtre occupant la partie haute du cadre avec les boutons de commande,
- une sous-fenêtre à gauche pour disposer (quand c'est nécessaire) les icônes ou les types d'éléments de base, ou les composants sélectionnés pour le modèle

en cours,

- une sous-fenêtre occupant le reste de l'espace pour l'affichage de listes de composants d'un type donné ou, dans le cas des produits, pour la description du graphe de déroulement matière.

Dans le cas de la fenêtre des produits, une dernière sous-fenêtre vient prendre place sur la gauche du cadre et propose des commandes pour faciliter la construction du graphe (affichage des coordonnées du pointeur lors du positionnement d'une icône, bouton de mise en place d'un quadrillage pour faciliter l'alignement des icônes).

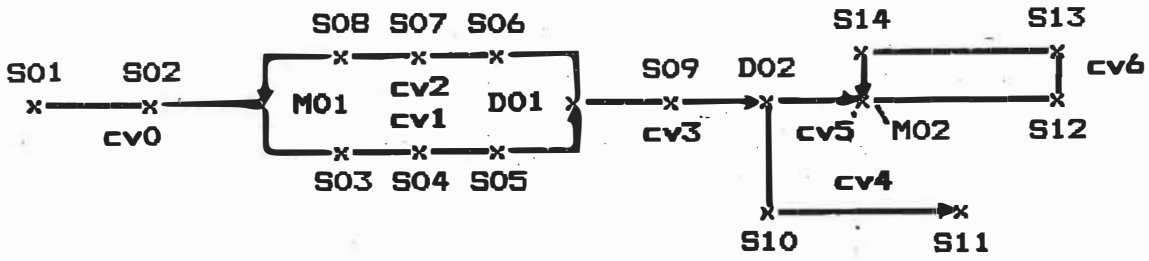
3.1.3 Modifications apportées à la modélisation MAP/1, hypothèses et limites

Les quatre parties d'un modèle sont indépendantes sauf pour la description des postes d'assemblage (ASSEMBLY STATION) et de production (PRODUCE STATION). En effet, les informations concernant les produits assemblés sont entrées à la suite de l'énoncé STATION avec MAP/1 ce qui ne permet pas de faire une description du parc machine de façon vraiment séparée des produits. Ce type de description est contraignant car il fige la description d'une telle STATION et nuit à la flexibilité et à la généralité d'un

modèle. Dans notre interface, les informations concernant l'assemblage et la production sont donc prises en compte dans le graphe de déroulement matière. L'usager doit faire attention de ne pas définir ces ressources autrement ailleurs dans le graphe et ce, même si à un moment donné elles effectuent des opérations de nature différente.

De même, la description du trajet d'un convoyeur a été modifiée de manière à rationaliser la présentation de cette information et à réduire sa répétition. En effet, la syntaxe MAP/1 demande d'introduire pour chaque énoncé CONVEYOR la liste des points qu'il dessert ainsi que la longueur et la capacité des segments reliant deux points consécutifs. Mais si le réseau de convoyeurs comporte un point de divergence d'où émanent plusieurs convoyeurs, il est demandé à l'usager de répéter dans l'énoncé DIVERGE tous les postes de type STATION qui sont accessibles à partir de ce point en empruntant chacun des convoyeurs qui en sortent, que ces postes soient situés sur ces convoyeurs ou sur leurs suivants.

Comme on peut le voir dans la figure 3.3, cela entraîne la répétition de la même information à plusieurs reprises, alors que le logiciel devrait se charger de réaliser les connections à partir des renseignements qui ont été introduits dans les énoncés CONVEYOR.



- S**: station **
- M**: point de convergence **
- D**: point de divergence **
- cv*: convoyeur *

Les énoncés MAP/1 pour décrire le trajet des convoyeurs ci-dessus ont la forme suivante:

```
CONVEYOR, cv0,.., S01,.., S02,.., M01,..;
CONVEYOR, cv1,.., M01,.., S03,.., S04,.., S05,.., D01,..;
CONVEYOR, cv2,.., D01,.., S06,.., S07,.., S08,.., M01,..;
CONVEYOR, cv3,.., D01,.., S09,.., D02,..;
CONVEYOR, cv4,.., D02,.., S10,.., S11,..;
CONVEYOR, cv5,.., D02,.., M02,..;
CONVEYOR, cv6,.., M02,.., S12,.., S13,.., S14;
MERGE, M01, cv1;
MERGE, M02, cv6;
DIVERGE, D01, 2, cv3 (S09, S10, S11, S12,  S13, S14), cv2
(S06, S07,  S08, S03,  S04, S05, S09, S10, S11, S12, S13,
S14);
DIVERGE, D02, 2, cv4(S12, S13, S14), cv6(S10, S11);
```

fig 3.3: Exemple d'utilisation des énoncés CONVEYOR, MERGE et DIVERGE

Un nouvel énoncé appelé SEGMENT a donc été créé pour remédier à cet inconvénient alors que, dans le même temps, la description du trajet d'un convoyeur disparaît de la fiche CONVEYOR. Il n'existe pas non plus de fiches pour les points de divergence et de convergence (énoncés MAP/1 DIVERGE et MERGE). Cette représentation facilite pour l'usager la modélisation car elle se rapproche plus de la représentation mentale qu'il se fait du convoyeur et peut aussi permettre, le cas échéant, de réaliser plus facilement une modélisation graphique des moyens industriels.

Pour développer le prototype, certaines limites ont par ailleurs été définies. Même si l'on souhaite fournir à l'usager un outil peu contraignant, quelques points nécessitent cependant une approche rigoureuse. La plupart de ces contraintes pourront être enlevées au fur et à mesure de l'évolution du logiciel.

La division de la tâche de modélisation par rapport à l'écriture "d'un trait" d'un programme MAP/1, renforce le risque d'erreurs pour l'appellation des composants du modèle, essentiellement des moyens industriels (STATION, TRANSPORTER...). La réutilisation et l'assemblage de différents modules provenant de plusieurs projets favorise aussi ce genre d'erreurs. Il est donc conseillé d'attribuer des noms

de façon standardisée (en adoptant, par exemple, une norme suivant les concepts de la technologie de groupe pour les pièces et une représentation similaire pour les machines).

Pour minimiser cet inconvénient, le bouton nommé "#RESOURCES#", dont il a déjà été question plus haut, permet d'obtenir la liste de toutes les ressources d'un type donné qui ont déjà été créées par l'utilisateur et ce, quel que soit le modèle dans lequel elles interviennent.

Dans un premier temps, il n'est pas prévu de pourvoir l'interface d'un système d'aide interactif, qui réclame un trop grand investissement au niveau du temps et qui n'apporte rien de plus pour valider l'interface. Néanmoins, les premiers composants de ce système ont été définis pour les champs des fiches où l'utilisateur doit entrer une information et où il peut rencontrer des problèmes sur la façon de présenter cette information. Un "cliquage" du commentaire précédant le champ provoque l'apparition d'une fenêtre d'assistance destinée à renseigner l'usager sur la signification du champ, sur son format et sur ses limites éventuelles. Cette fenêtre ne contient pour l'instant que le nom du champ pour lequel elle a été invoquée. Cette organisation décentralisée facilite l'accès ponctuel et rapide aux informations dont l'usager a besoin sans pour cela passer au travers de plusieurs pages d'informations ou de

menus comme c'est le cas traditionnellement avec un système d'aide centralisé.

D'autre part, si l'utilisateur a introduit des paramètres erronés, ces erreurs lui sont signalées par MAP/1 avant de commencer la simulation. Il lui "suffit" alors de se référer au fichier ".out" généré par le traducteur où MAP/1 lui signale la présence de ces erreurs au niveau des lignes de code où elles apparaissent: on ne développe pas non plus de nouvel outil pour cette tâche, pour la même raison que pour le système d'aide.

(Dans les deux cas ci-dessus, cette façon de faire va à l'encontre d'une meilleure convivialité, mais développer un nouveau système d'aide et un détecteur d'erreurs est en soi un projet à part entière.)

D'autres caractéristiques de MAP/1 n'ont pas été prises en compte. Ainsi, les variables mises à la disposition de l'usager pour définir des conditions ou des durées (ex: USTIM, LSTSTATION, MXTTIME...) et dont les noms sont souvent peu explicites n'ont pas été renommées. Pour pouvoir les utiliser, l'usager doit savoir qu'il en existe une qui correspond à ses besoins et pour cela doit accéder à la documentation MAP/1. Ainsi, une modification de leur nom ne produit pas un bénéfice appréciable. Cette amélioration

peut se faire conjointement à l'élaboration du système d'aide.

L'interface n'intègre pas non plus la création des fonctions FORTRAN que l'usager pourrait vouloir adjoindre à son modèle ni l'écriture des appels de sous-programmes dans le programme principal FORTRAN prévu à cet effet. Cela se justifie par le fait que ce type de fichiers est en marge du processus de modélisation MAP/1 proprement dit: il est possible de les créer et d'y accéder en tout temps grâce aux fenêtres "TEXTEDIT" des stations SUN ou tout autre éditeur de texte sous UNIX. Bien entendu, notre interface tient compte de leur existence au moment d'exécuter la simulation en demandant à l'usager de spécifier le nom du fichier FORTRAN (s'il existe) associé au modèle.

Un graphe de produit commence obligatoirement par (au moins) une icône de type composant ou produit, la première positionnée donnant le nom au produit qui est ainsi décrit. Cette contrainte est justifiée par la façon dont le traducteur compile le graphe produit. De manière à tenir compte de la contrainte imposée par MAP/1 concernant l'ordre d'un énoncé PART et des énoncés ROUTE qui s'y rapportent (ces derniers doivent se trouver immédiatement après l'énoncé PART), le traducteur réalise une recherche récursive "en profondeur d'abord" sur l'ensemble du graphe pour chacun

des composants ou produits intervenant afin de définir tous les énoncés ROUTE qui le concernent: ainsi, le premier composant ou produit positionné (par ordre chronologique) sera considéré implicitement comme le composant majeur et donnera, par exemple, son nom au produit résultant d'une opération d'assemblage.

Une opération d'assemblage ne doit donc pas forcément être suivie d'une icône de produit, au contraire d'une opération de production qui doit être suivie d'une icône de ce type pour chacun des éléments générés. Une icône de produit débute ou clot un graphe ou une branche d'un graphe mais peut aussi se rencontrer entre deux icônes d'opération (au sens large, i.e. normale, d'assemblage...), ceci afin de favoriser la modularité de la description des produits. Ainsi, tout graphe décrivant un élément qui entre dans la composition d'un autre se termine par une icône produit qui met en relation les deux graphes lors de la compilation. Ceci sous-entend que ces graphes apparaissent tous les deux dans la liste des éléments du modèle. Un modèle se compose obligatoirement d'un seul élément de chacun des types "objectif", "condition initiale", "moyen", mais peut donc inclure plus d'un "produit".

Le graphe de déroulement matière n'est pour l'instant pas imprimable (sa méthode d'impression ne provoque pas

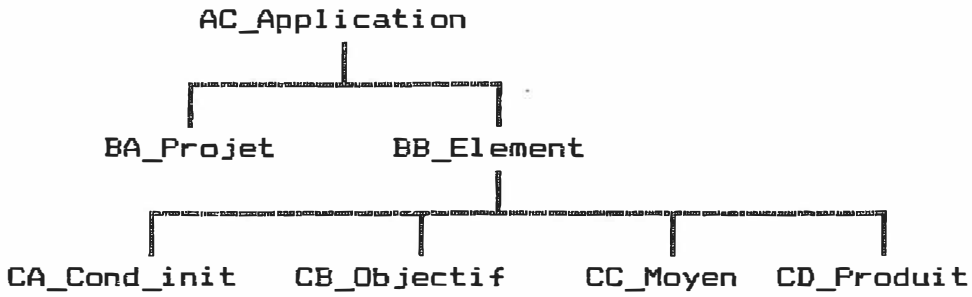
d'action) car la réalisation de ce module fait appel à des connaissances spécifiques sur le type de traceur utilisé. Aussi, en raison de sa complexité et de sa longueur due à la diversité des cas, cette fonction reste à réaliser.

3.2 Conception préliminaire du système

Cette étape voit la création de 4 "super-classes" dont les noms ont pour préfixe A*_. La première "classe" porte ce nom par abus car elle correspond en fait au programme principal dont le rôle est de donner accès aux autres classes. Avec la classe AB_Fenêtre, qui est chargée de gérer l'organisation de l'écran, cette première classe constitue le contrôleur d'écran décrit à la figure 3.2. Les deux classes suivantes correspondent quant à elles au contrôleur de la base de données présenté dans cette même figure. Les classes de niveau inférieur sont repérées par des préfixes B*_ et C*_ suivant leur niveau dans la hiérarchie. On obtient finalement la décomposition présentée sur la figure 3.4.

Le paragraphe suivant présente dans ses grandes lignes le mode opératoire de l'interface. La formalisation des problèmes, telle qu'annoncée dans la méthodologie de la POO (paragraphe 2.3.2), est faite dans la figure 3.5. Le niveau

AA_Main AB_Fenetre



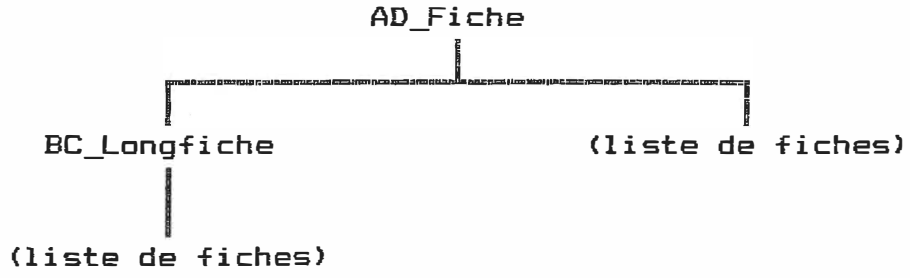


fig 3.4: Hiérarchie des classes

indiqué avant la définition des problèmes correspond au niveau de la classe dans la hiérarchie.

3.2.1 Décomposition du problème

*** NIVEAU 0 ***

PROBLEME

Réaliser un système convivial et interactif qui facilite l'apprentissage et l'usage de MAP/1

STRATEGIE INFORMELLE

Après la mise en route du logiciel, le système ouvre la fenêtre principale où l'usager choisit un projet dans la liste des projets déjà créés et que lui propose le système. Il choisit ensuite le type d'élément qu'il désire décrire pour le projet en cours et le système affiche la liste de tous les éléments de ce type. L'usager choisit ensuite de construire un élément qu'il décrit puis sauvegarde. Un modèle est complet une fois qu'un élément de chaque type (ou plus pour les produits) a été sauvegardé.

L'usager peut à tout moment sauvegarder le projet. Dès que celui-ci est complet, il peut demander au système de compiler le projet puis de l'exécuter. L'utilisateur peut faire imprimer les éléments de son modèle (conditions initiales, objectifs, moyens, produits et résultats). A tout

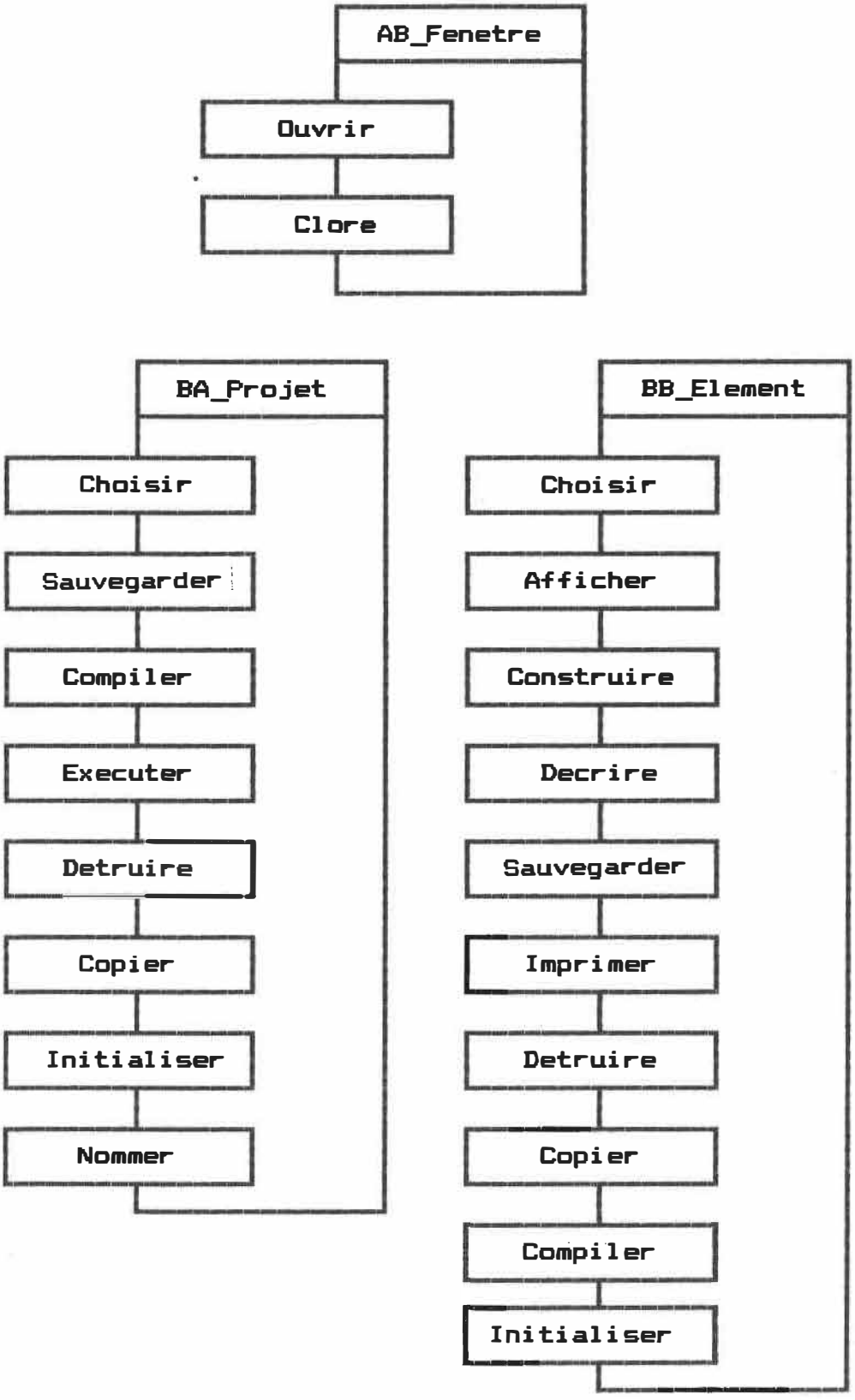


fig 3.5: Formalisation des classes

moment, il peut modifier un élément (objectif, moyen...), en recommençant la procédure ci-dessus ou en le détruisant pour le projet en cours ou de façon définitive. Il peut encore copier un élément ou un projet.

*** NIVEAU 1.1 ***

PROBLEME

Décrire la fonctionalité de AC_Application.

STRATEGIE INFORMELLE

*1 Pour décrire une application, on utilise différentes méthodes suivant le type de l'application: voir les éléments du niveau 2.

*2 Pour sauvegarder une application, on utilise différentes méthodes suivant le type de l'application: voir les éléments du niveau 2.

*3 Pour terminer une application, le système clos la fenêtre de l'application en sauvegardant ou non l'application concernée.

*** NIVEAU 2.1 ***

PROBLEME

Décrire la fonctionalité de BA_Projet.

STRATEGIE INFORMELLE

*1 Pour choisir un projet, l'usager le nomme et le système l'initialise.

*2 Pour sauvegarder un projet, le système sauvegarde la

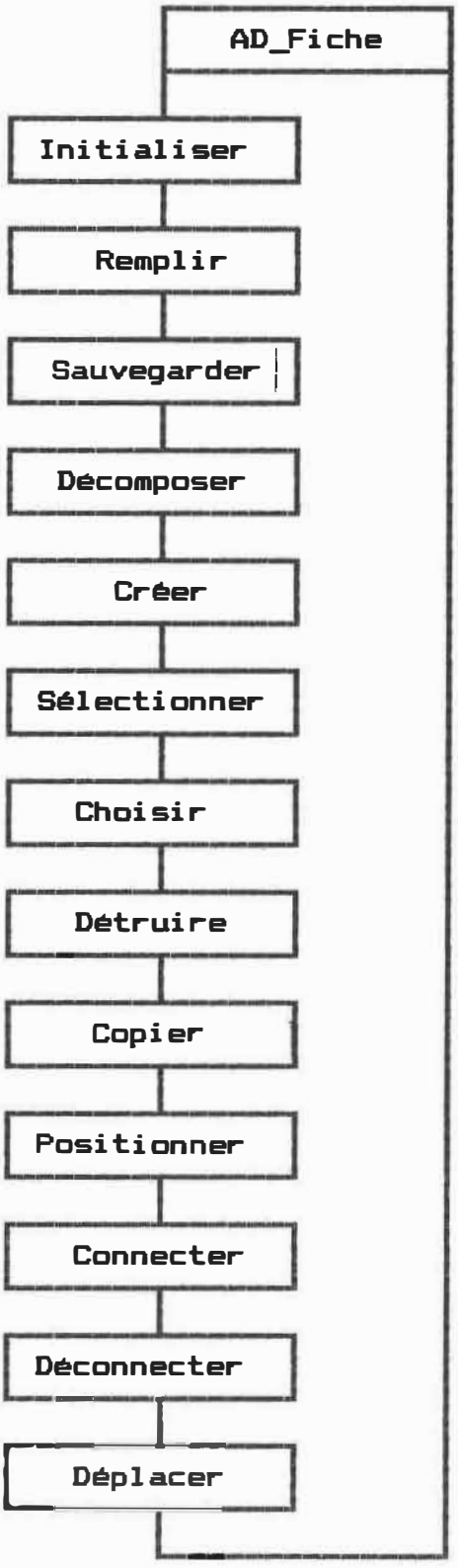


fig 3.5: Formalisation des classes (fin)

liste des éléments qui le composent dans son fichier.

*3 Pour compiler un projet, le système sauvegarde le projet puis compile chacun de ses éléments.

*** NIVEAU 2.2 ***

PROBLEME

Décrire la fonctionalité de BB_Elément.

STRATEGIE INFORMELLE

*1 Pour construire un élément, le système ouvre sa fenêtre et initialise (si nécessaire) les renseignements sauvegardés préalablement.

*2 Pour sauvegarder un élément, le système sauvegarde la liste de fiches qui le composent dans son fichier.

*3 Pour compiler un élément, le système décompose suivant la syntaxe MAP/1 les renseignements contenus dans chaque fiche.

*** NIVEAU 3.1 ***

PROBLEME

Décrire la fonctionalité de CA_Condinit.

STRATEGIE INFORMELLE

Pour décrire les conditions initiales, le système ouvre leur fenêtre où il initialise la fiche présentée. L'usager remplit et sauvegarde la fiche que le système lui propose. Pour modifier sa description, l'usager peut recommencer à sa guise le même processus.

*** NIVEAU 3.2 ***

PROBLEME

Décrire la fonctionalité de CB_Objectifs.

STRATEGIE INFORMELLE

Pour décrire les objectifs, l'usager choisit la fiche de l'étude qu'il désire réaliser ou bien, s'il n'en trouve pas à sa convenance, il en crée et en sauvegarde une nouvelle. Le système ouvre une fenêtre où il initialise la fiche choisie que l'usager remplit puis sauvegarde. Le système présente dans la partie gauche de la fenêtre la liste des études choisies par l'usager.

*** NIVEAU 3.3 ***

PROBLEME

Décrire la fonctionalité de CC_Moyen.

STRATEGIE INFORMELLE

Pour décrire les moyens industriels, l'usager sélectionne un type de fiche dans le menu et choisit ensuite une de celles qui lui sont proposées (ancienne ou nouvelle). Le système ouvre une fenêtre pour disposer cette fiche qu'il initialise et que l'usager remplit et sauvegarde. Il peut aussi détruire une fiche ou modifier les renseignements qu'elle contient. Pour cela, il la remplit et la sauvegarde de nouveau. Il peut encore la copier.

*** NIVEAU 3.4 ***

Remarque préliminaire: dans la décomposition qui suit le terme icône est employé pour désigner le symbole (la fenêtre) représentant une fiche, il ne représente donc pas un nouveau type d'objet mais correspond dans ce contexte à une fiche.

PROBLEME

Décrire la fonctionalité de CD_Produit.

STRATEGIE INFORMELLE

Pour décrire un produit, l'usager construit le graphe de déroulement matière. Il **sélectionne** un type d'icône dans le menu situé à gauche dans la fenêtre. Il **positionne** ensuite l'icône dans la fenêtre. Le système **ouvre** alors une fenêtre et initialise la fiche choisie que l'usager **remplit** et **sauvegarde**. Il **connecte** l'icône aux autres éléments du graphe conformément au plan de déroulement matière. Il peut aussi détruire une icône du graphe ou la **dupliquer** ou la **déconnecter** si besoin est. Dans le premier cas, le système se charge de **déconnecter** l'icône. L'usager peut encore modifier une fiche: pour cela, il **ouvre** l'icône, remplit puis sauvegarde la fiche modifiée. L'usager peut enfin **déplacer** une icône mal positionnée dans la fenêtre.

La description des autres objets n'est pas faite en raison de leur trop grand niveau de détail et n'apparaît

qu'au niveau de la conception détaillée. L'annexe C décrit le fonctionnement des interfaces des objets et leurs paramètres.

3.2.2 La base de données et les objets

Les données nécessaires au fonctionnement du logiciel entrent dans trois catégories:

- les objets standard,
- les objets créés par les usagers,
- les fichiers de données standard (fig 3.6).

Les deux contrôleurs font l'objet d'un codage séparé afin d'assurer une grande indépendance entre eux [DRA 85]. Pour la communication de ces données entre les deux contrôleurs et afin de les rendre indépendants, on a recours à des fichiers temporaires où chacun d'eux vient lire ou écrire les renseignements demandés.

De même, lors de la compilation, de nouveaux fichiers temporaires sont créés pour assurer la bonne répartition des informations dans les champs d'instruction de MAP/1. Ces fichiers sont détruits une fois la compilation terminée. Les annexes C et D présentent les fichiers d'en-tête

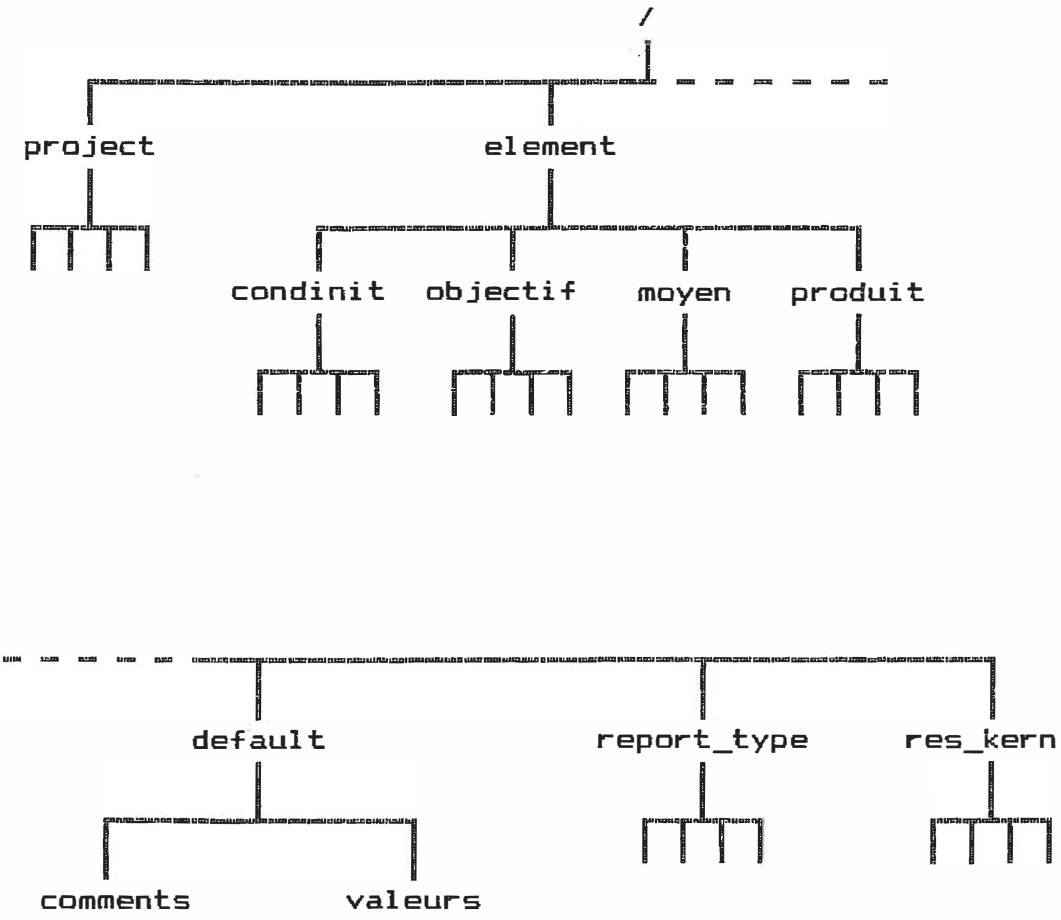


fig 3.6: Organisation de la base de données

du contrôleur de la base de données et du contrôleur d'écran, fichiers qui découlent de la phase de conception détaillée.

La classe BA_Projet gère les renseignements concernant les noms des éléments qui composent un projet et assure leur regroupement lors de la compilation.

La classe BB_Elément gère les fiches qui composent un élément donné et les renseignements qui y sont associés (nombre de fiches, rang...).

La classe AD_Fiche gère les informations rentrées par l'usager lors du remplissage d'une fiche ou par le système s'il s'agit de valeurs par défaut.

En ce qui concerne le contrôleur d'écran, on essaye, dans la mesure du possible, de réutiliser des objets déjà définis dans les librairies Xt et Xlib; de ce fait, le contrôleur d'écran est donc réalisé séparément des autres objets et suivant la syntaxe du langage C qui est celle des objets de Xt et Xlib. On peut le comparer à une boîte noire qui assure la connection entre les boutons que l'usager voit à l'écran et les méthodes des objets manipulés en arrière.

4. COMMENTAIRES ET PROPOSITIONS D'EVOLUTION

4.1 Validation de l'approche informatique

En dépit du fait que le langage C soit à la base des librairies X et du langage C++, quelques problèmes de compatibilité mineurs sont apparus lors du développement du logiciel dûs en majorité à des erreurs de codage dans certaines librairies X. Ces quelques erreurs ont pu être corrigées sans problèmes et sont somme toute mineures, en comparaison de l'énorme potentialité des éléments de ces librairies et de leur fiabilité. Leur usage ne réclame aucune connaissance spécifique de la façon dont ils sont définis (ils se comportent comme des boîtes noire dont on connaît le rôle et les interfaces) et ils sont facilement exploitables et combinables entre eux.

D'un point de vue programmation orientée objet proprement dite, cette approche s'avère très intéressante en raison de la clarté du découpage qu'elle sous-entend et de la très grande indépendance entre les divers objets. Elle n'est pas contraignante et se révèle en même temps rigoureuse. Elle laisse une grande liberté à l'intérieur d'un cadre de programmation bien défini et assure une meilleure clarté (association objet informatique / objet réel) tant

pour celui qui développe que pour les personnes qui assurent la mise à jour du logiciel. Ainsi on peut facilement envisager une évolution du logiciel sans remaniements profonds ni transformations, simplement en rajoutant aux classes de nouvelles méthodes ou de nouvelles classes suivant les besoins.

Le seul inconvénient notable tient au fait que la version du C++ que nous utilisons ne supporte pas l'héritage multiple qui consiste à rendre une classe fille de plusieurs autres. Si pour le moment cette lacune n'a pas causé de problèmes ce pourrait être le cas lors de futurs développements. Cette caractéristique (l'héritage multiple) est toutefois censée apparaître dans la version 2.0 prévue pour l'automne 90.

L'approche choisie se révèle donc a posteriori aussi intéressante qu'elle semblait l'être a priori et ce, pour les raisons suivantes:

- réutilisations de composants puissants et très fiables pour la plupart par le biais des librairies X (Xt Toolkit et XLib),
- assurance d'un grand niveau de compatibilité et de portabilité par l'usage de ces librairies d'objets standards et du C++,

- grand niveau d'indépendance entre les éléments de base du logiciel (les deux contrôleurs),
- facilité de maintenance et de mise à jour des modules orientés objet,
- facilité d'ajout et d'intégration de nouvelles classes d'objets.

Finalement, la taille du code produit approche les 6000 lignes qui sont réparties à peu près équitablement entre d'une part, le contrôleur d'écran (UNIX, C, Xt et Xlib) et d'autre part, le contrôleur de la base de données et le compilateur (UNIX et C++).

4.2 Critique de l'interface

Du fait des qualités énoncées ci-dessus, l'interface se révèle facile d'accès et de manipulation pour l'usager. Son usage ne réclame pas un grand entraînement, sauf peut-être pour le format des valeurs introduites dans les champs des fiches et les règles de successions dans un graphe (l'annexe E présente le mode d'emploi de l'interface). Cet inconvénient pourrait être encore réduit par la réalisation complète du système d'aide qui favorise l'auto-apprentissage ("learning by doing").

Les temps d'accès aux fenêtres et d'affichage des informations sont rapides grâce aux capacités de traitement des stations de travail et en dépit d'un codage pas toujours optimisé du point de vue vitesse de traitement.

La modularité de la modélisation s'avère avantageuse pour l'usager à deux points de vue: tout d'abord elle fixe et ordonne dans son esprit la vision du modèle (un projet se compose d'un élément de chacun des types condition initiale, objectif, moyen et éventuellement plusieurs de type produit et chacun d'eux n'est rien d'autre qu'un ensemble de fiches) et répartit par catégories logiques (moyens, produits...) les différentes informations dont il dispose; ensuite, elle lui permet de bâtir de nouveaux modèles ou de faire des modifications à partir des éléments créés antérieurement et sauvegardés en base de données lui épargnant ainsi tout nouvel effort de modélisation.

Par rapport à IIS, qui fait également intervenir des fiches, les niveaux hiérarchiques sont ici à la fois moins nombreux mieux documentés et plus accessibles. Le multifenêtrage (voir annexes A et B) montre en tout temps le niveau hiérarchique auquel on se trouve, ainsi que le chemin qui a permis d'y accéder, tandis que la disposition standardisée des commandes en haut de chaque fenêtre permet à l'usager de se concentrer sur le problème industriel qu'il

cherche à résoudre sans avoir en plus à mémoriser les dites commandes, comme c'est le cas avec IIS.

L'utilisation des graphes simplifie également la tâche de modélisation en faisant apparaître le minimum d'information nécessaire à la compréhension du processus industriel et en rejetant au niveau inférieur (celui des fiches) les détails de ce processus. L'usager peut maintenant valider cette partie du modèle au moment où il la crée. Cette représentation rend l'apprentissage et l'utilisation du logiciel plus aisés qu'une présentation par tableaux et fiches.

De plus, on peut envisager de rendre l'interface utilisable avec un autre simulateur que MAP/1, au prix de modifications mineures, ou pour le moins locales, essentiellement au niveau du contrôleur de BDD, du compilateur (méthodes "split" pour les objets de la classe "AD_Fiche" et ses héritiers et "compile" de BA_Projet et de BB_Element) et des éléments standard de la base de données (les fiches propres à MAP/1).

4.3 Axes d'évolutions futures

On peut d'ores et déjà isoler trois grandes directions de développement (qui ne sont pas incompatibles

entre elles d'ailleurs) à partir de l'interface réalisée:

- l'intégration des outils manquants (aide interactive, dévermineur...) et de nouveaux outils (aide à la réalisation des autres étapes de la simulation...),
- le substitution de MAP/1 par un autre logiciel en raison de la polyvalence de l'interface (voir paragraphe précédent),
- le développement de nouveaux objets destinés à supplanter MAP/1 et à obtenir un simulateur OO.

Dans la première direction, les travaux peuvent porter sur l'intégration d'autres outils d'aide à la réalisation de la simulation:

- génération de scénarios,
- analyse des résultats,
- analyses économiques plus poussées,
- recommandations résultant des analyses...

outils qui pourraient prendre la forme de systèmes experts -par exemple pour la génération de scénarios, le choix des conditions de simulation en fonction des objectifs recherchés...- intégrés à la classe CB_Objectif en complément des études standard proposées actuellement, ou accessibles par une nouvelle classe.

La troisième étape de la modélisation, celle de description des moyens industriels, peut facilement donner lieu à des améliorations au niveau de la présentation. On peut par exemple, envisager de réaliser une interface avec un système de CAO ou avec TESS où serait définie la configuration de l'atelier ou de l'usine étudiés.

On peut aussi plus simplement modifier la fenêtre des moyens de façon à intégrer des icônes standard destinées à modéliser les différentes ressources comme cela a été fait pour les produits, en supprimant par là-même les fiches DISTANCE et SEGMENT pour les moyens de transport, les renseignements qu'elles contiennent étant directement retrouvables dans le dessin par le positionnement relatif des autres icônes. Cette représentation symbolique pourrait aussi permettre, comme c'est le cas avec AutoMod II, de réaliser des animations.

On peut enfin envisager de concevoir des modèles encore plus modulaires dans leur architecture en réalisant l'assemblage non pas au niveau des éléments (moyens, produits...) mais directement au niveau des fiches. Cependant, une telle configuration est difficilement réalisable, excepté pour les moyens industriels qui peuvent se prêter à un tel degré de réutilisation. D'ailleurs, une amorce est

déjà effectuée avec les fenêtres temporaires qui affichent la liste de toutes les resources d'un type donné. On pourrait ainsi construire un élément de type moyens comme on construit actuellement un projet, en réutilisant des composants déjà créés et sauvés en base de données.

Pour développer le simulateur OO dont il est question dans la troisième avenue, l'organisation des objets induite par MAP/1 peut être conservée, chacun recevant en plus des méthodes destinées à décrire son comportement au cours de la simulation, tandis que de nouveaux objets destinés à contrôler le mécanisme de simulation (horloge, générateur de nombres aléatoires...) doivent être ajoutés. Dans cette optique, il est possible d'utiliser des librairies publiques d'objets C++ (voir [ELD 90]) qui tendent à simplifier ce développement, sous réserve de pouvoir utiliser le concept d'héritage multiple.

Si MAP/1 sert de point de départ et de modèle pour développer ce nouveau simulateur, on peut également envisager d'étendre sa portée en rajoutant de nouveaux objets destinés, par exemple, à raffiner la description du système industriel, notamment en ce qui concerne les moyens de manutention et de stockage (intégration de nouveaux types comme AGVS, ASRS...).

CONCLUSION

La simulation est un outil puissant permettant de prendre en compte la forte incertitude qui prévaut tant en phase de conception et d'évaluation d'un projet, qu'en phase d'opération d'un système manufacturier. Elle s'avère encore plus précieuse dans le cadre de l'intégration de nouvelles technologies (de type automatisation, atelier flexible...), où les investissements sont souvent très lourds, en permettant de diminuer les risques en générant plusieurs scénarios. Toutefois, son usage réclame des connaissances pluri-disciplinaires et présente un coût en terme de temps relativement élevé ce qui limite sa diffusion dans l'industrie. Il y a donc grand besoin d'améliorer la convivialité de cet outil afin de permettre à l'usager de se concentrer uniquement sur le problème pratique qu'il désire résoudre.

Aussi, en réalisant ce projet, nous poursuivions un double objectif: d'une part donc, simplifier l'usage de cet outil et le rendre utilisable sans expérience préalable importante en simulation et d'autre part, valider la programmation orientée objet pour développer des logiciels de simulation, option que nous avons préférée au simple habillage d'un simulateur en raison des évolutions qu'elle permet d'envisager.

L'interface réalisée se révèle facile à manipuler et ne s'avère pas "impressionnante" pour l'usager en raison de sa présentation simple qui fait néanmoins apparaître tous les éléments nécessaires à la conduite de chaque tâche, quelle que soit la partie du modèle où l'on travaille. Les caractéristiques de l'interface correspondent aux standards qui prévalent dans le domaine tout en intégrant des concepts nouveaux tant au niveau utilisation (aide décentralisée...) qu'au niveau modélisation (graphe de produits, modularité...) du projet de simulation. Ces caractéristiques font qu'elle encourage la réutilisation des composants d'un modèle et leur maintenance. Son mode de gestion des informations (graphes et fiches) favorise ces deux aspects et peut aussi s'adapter sans trop de difficultés à d'autres logiciels de simulation.

En ce qui concerne le second objectif de ce projet, nous avons énoncé au chapitre 4 les avantages du mode de programmation par objet. Il représente à notre avis la meilleure avenue pour le développement de simulateurs capables de répondre aux besoins des usagers tant du point de vue puissance (capacité de fournir des informations très précises...) que du point de vue simplicité (modélisation, réutilisation...) et évolutivité du logiciel.

BIBLIOGRAPHIE

[ASM 47] -, "Operation and Flow Process Charts, ASME Standard 101", American Society of Mechanical Engineers, New York, (1947)

[AUT 88] -, "AUTOMOD II rev 1.1: Users Manual", AUTOSIMULATION INC, (1988)

[BAL 86] O. BALCI, "Requirements For Model Development Environments", COMPUTER & OPERATIONS RESEARCH 13:1, 53-67, (1986)

[BEL 85] G. BEL, D. DUBOIS, "Modélisation et simulation de systèmes automatisés de production", RAIRO APII vol 19, nº 1, (1985)

[BEN 85] G. BENCHIMOL coordinateur, "La Conception des Usines de Demain", HERMES PUBLISHING, (1985)

[BON 88] A.H. BOND & B. SOETARMAN, "Multiple Abstraction in Knowledge-based Simulation", AI AND SIMULATION: THE DIVERSITY OF APPLICATIONS, T. HENSON éd., SCS, (1988)

[CAN 89] A. CANE, "'Oop' Could Revolutionize Software Programming", FINANCIAL POST 11/09/89, 43, (1989)

[CHR 88] K. CHRISTIAN, "The UNIX Operating System", 2e éd., JOHN WILEY & SONS, (1988)

[DRA 85] S.W. DRAPER & D.A. NORMAN, "Software Engineering for User Interfaces", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING SE-11 no 3, 252-258 (mars 1985)

[ELD 90] D.E. ELDREDGE, J.D. McGREGOR & M.K. SUMMERS, "Applying the Object-Oriented Paradigm to Discrete Event Simulations Using the C++ Language", SIMULATION 54:2, 83-91, (1990)

[FOR 87] D.R. FORD & B.J. SCHROER, "An Expert Manufaturing Simulation System", SIMULATION 48:5, 193-200, (1987)

[HAD 87] J. HADDOCK, "An Expert System Framework Based on a Simulation Generator", SIMULATION 48:2, 45-53, (1987)

[KET 89] M.G. KETCHAM, R.E. SHANNON & G,L, HOGG, "Information Structures for Simulation Modeling of Manufacturing Systems", SIMULATION 52:2, 59-67, (1989)

[KIN 86] C.U. KING & E.L. FISHER, "Object-Oriented Shop-Floor Design, Simulation and Evaluation", PROCEEDINGS OF THE FALL INDUSTRIAL ENGINEERING CONFERENCE, IIE, (1986)

[LAW 86] A.M. LAW, "Simulation of Manufacturing Systems", INDUSTRIAL ENGINEERING, (mai 1986)

[LAW 89a] A.M. LAW & S.W. HAIDER, "Selecting Simulation Software for Manufacturing Applications: Practical Guidelines & Software Survey", INDUSTRIAL ENGINEERING 31:5, 33-46, (mai 89)

[LAW 89b] A. M. LAW & M.G. McCOMAS, "Pitfalls to Avoid in the Simulation of Manufacturing Systems", INDUSTRIAL ENGINEERING 31:5, 28-31, (mai 89)

[MEL 89] J.M. MELLICHAMP & Y.H. PARK, "A Statistical Expert System For Simulation Analysis", SIMULATION 52:4, 134-

139, (1989)

[MIN 86] R.J. MINER & L.J. ROLSTON, "MAP/1 User's Manual, Version 3.0", PRITSCKER AND ASSOCIATES, (1986)

[MYZ 89] J.H. MIZE, T. BEAUMARIAGE & C. KARACAL, "System Modeling Using Object-Oriented Programming", PROCEEDINGS OF THE 1989 INTERNATIONAL INDUSTRIAL ENGINEERING CONFERENCE & SOCIETIES' MANUFACTURING AND PRODUCTIVITY SYMPOSIUM", IIE, 13-17, (1989)

[NYE 88] A. NYE, "XLib Reference Manual", Vol. 1 et 2, O'REILLY & ASSOCIATES Inc., (1988)

[OKE 86] R. O'KEEFE, "Simulation and Expert Systems: a Taxonomy and Some Examples", SIMULATION 46:1, 10-16, (1986)

[ORE 86] T. OREN, "Knowledge Bases for Advanced Simulation Environment", PROCEEDINGS OF THE CONFERENCE ON INTELLIGENT SIMULATION ENVIRONMENTS, P.A. LUKER & H.H. ADELSBERGER eds., SCS, 16-22, (1986)

[PRI 86] A.A.B. PRITSKER, "Introduction to Simulation and SLAM II", SYSTEM PUBLISHING CORP., 3e éd., (1986)

[RED 87] R. REDDY, "Epistemology of Knowledge Based Simulation", SIMULATION 48:4, 162-166, (1987)

[RIB 86] A. RIBOUD, "Modernisation, mode d'emploi; rapport au premier ministre", UNION GENERALE D'EDITION, (1986)

[ROB 88] S.D. ROBERTS & J. HEIM, "A Perspective on Object-Oriented Simulation", PROCEEDINGS OF THE 1988 WINTER SIMULATION CONFERENCE, M.ABRAMS, P.HAIGH & J. COMFORT

eds., SCS, (1988)

[ROL 85] L.J. ROLSTON, "Modeling Flexible Manufacturing Systems with MAP/1", ANNALS OF OPERATIONS RESEACH 3, 189-204, (1985)

[RUI 87] S. RUIZ-MIER & J. TALAVAGE, "A Hybrid Paradigm For Modeling of Complex Systems", SIMULATION 48:4, 135-141, (1987)

[SHA 86] R.E. SHANNON, "Intelligent Simulation Environments", PROCEEDINGS OF THE CONFERENCE ON INTELLIGENT SIMULATION ENVIRONMENTS, P.A. LUKER & H.H. ADELSBERGER eds., SCS, 150-156, (1986)

[SHM 86] R.E. SHANNON, R.J. MAYER & D.T. PHILLIPS, "Knowledge Based Simulation Techniques for Manufacturing", Technical paper; Conference Ultratech - Artificial Intelligence, (1986)

[SHN 87] B. SCHNEIDERMAN, "Designing the User Interface: Strategies for Effective Human-Computer Interaction", ADDISON-WESLEY PUBLISHING COMPANY, (1987)

[SUR 85] R. SURI, "An Overview of Evaluative Models for Flexible Manufacturing Systems", ANNALS OF OPERATIONS RESEARCH 3, (1985)

[STA 87] C.R. STANDRIDGE & A.A.B. PRITSKER, "TESS The Extended Simulation Support System", JOHN WILEY, (1987)

[TEL 90] A. TELLIER, "Une interface calculant le prix de revient à partir des résultats d'une simulation manufacturière", Mémoire de maîtrise à l'Ecole Polytech-

nique de Montréal (non publié), (1990)

[THO 86] T. THOMASMA & O.M. ULGEN, "Simulation Modeling in an Object-Oriented Environment using SMALLTALK-80", PROCEEDINGS OF THE 1986 WINTER SIMULATION CONFERENCE, J. WILSON, J.HENRIKSEN, S. ROBERTS eds., SCS, (1986)

[THO 88] T. THOMASMA & O.M. ULGEN, "Hierarchical, Modular Simulation Modeling in Icon-based Simulation Program Generators for Manufacturing", PROCEEDINGS OF 1988 WINTER SIMULATION CONFERENCE, M. ABRAMS, P. HAIGH & J. COMFORT eds., (1988)

[TRE 88] S. TREU, "Designing a 'Cognizant Interface' Between the User and the Simulation Software", SIMULATION 51:6, 227-234, (1988)

[VIL 89] L. VILLENEUVE, "Introduction à la Simulation des Systèmes Manufacturiers", (Traduit de PRITSKER & Ass., Inc), Ecole Polytechnique de Montréal, (1989)

[VOI 85] W. VOISIN, "Les Usines de Demain", HERMES PUBLISHING, (1985)

[WIE 88] R. WIENER & L. J. PINSON, "An Introduction to Object-Oriented Programming and C++", ADDISON-WESLEY Inc., (1988)

[YOU 89] D.A. YOUNG, "X Window System Programming and Applications with Xt", PRENTICE HALL Inc., (1989)

[ZAL 88] P. A. ZALEVSKY, "Knowledge-based Simulation of Manufacturing Facilities", AI AND SIMULATION: THE DIVERSITY OF APPLICATIONS, T. HENSON éd., SCS, (1988)

ANNEXES

ANNEXE A: PRESENTATION DE FICHES GEREES PAR L'INTERFACE

Les fiches gérées par notre interface sont au nombre de 20 et, pour la plupart, regroupent plusieurs énoncés MAP/1. La figure A.1 de la page suivante présente différentes fiches dans leur fenêtre.

Les champs de commentaires ainsi que les valeurs par défaut de ces fiches se trouvent dans le répertoire "default", dans des fichiers appelés frm*_c et frm*_v respectivement où * correspond au rang de la fiche (de 0 pour BEGIN à 19 pour END PROCESS). La répartition des énoncés MAP/1 dans ces fiches est présentée dans le tableau A.1.

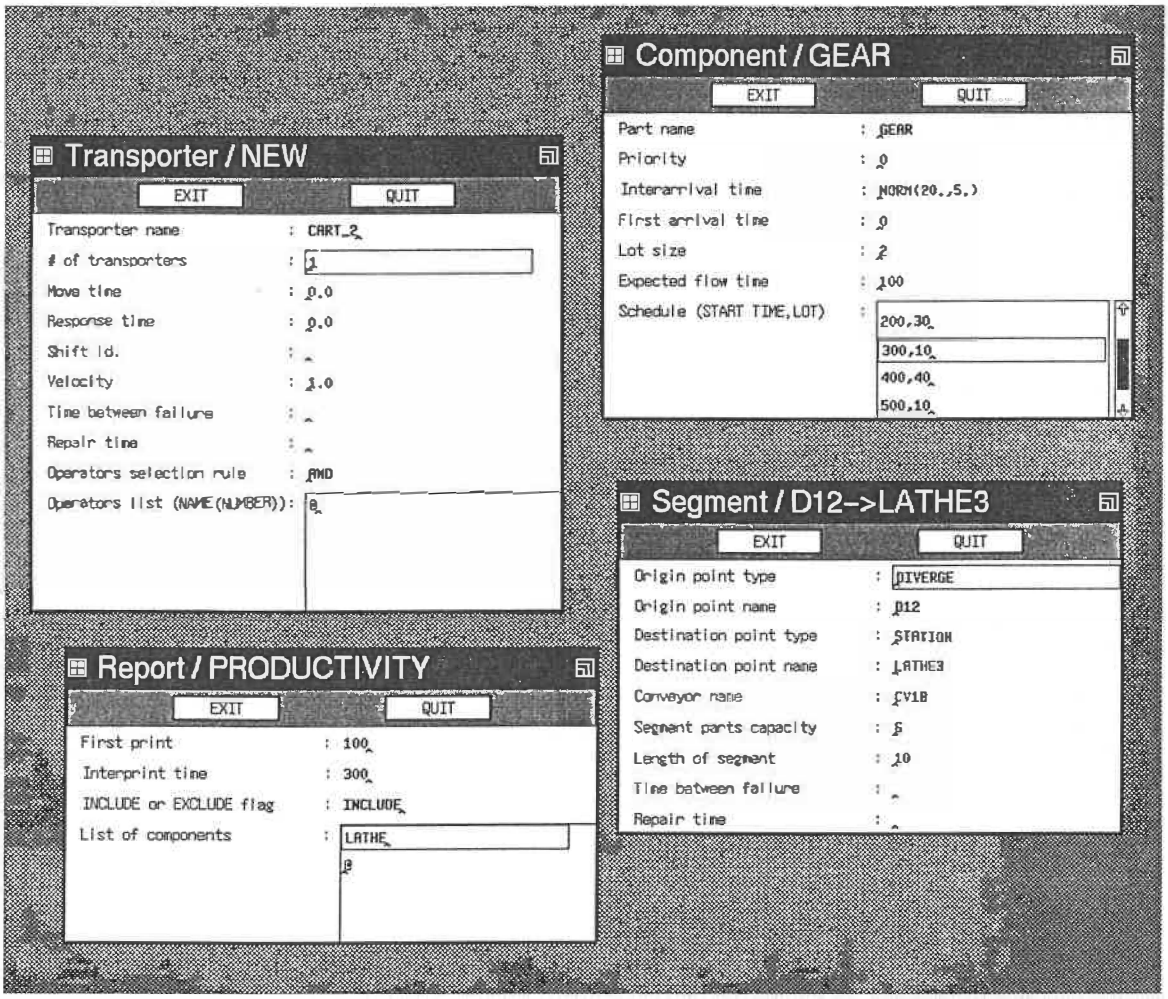


fig A.1: Exemples de fiches

Fiche	Enoncés MAP/1
BEGIN	BEGIN, SEED, CLEAR, SHIFT
REPORT	REPORT, DEFINE REPORT (partiel)
DEFINE REPORT	DEFINE REPORT
TRANSFORM. STATION	STATION (partiel), RANKING, FAILURE, MAINTENANCE, CHANGETOOL, OPERATOR
STOCK	STATION, RANKING, OPERATOR
TRANSPORTER	TRANSPORTER, BREAKDOWN, OPERATOR
DISTANCE	DISTANCE
CONVEYOR	CONVEYOR (partiel), BREAKDOWN
SEGMENT	MERGE, DIVERGE, CONVEYOR BREAKDOWN (partiels)
FIXTURE	FIXTURE (partiel)
PERSONNEL	PERSONNEL
COMPONENT	PART, SCHEDULE
PRODUCT	PART, ROUTE (partiel)
OPERATION	ROUTE (partiel)
ASSEMBLY STATION	ROUTE (partiel), STATION (partiel)
PRODUCTION STATION	ROUTE (partiel), STATION (partiel)
INVENTORY	ROUTE (partiel)
TRANSPORT	ROUTE (partiel)
RETURNMENT	ROUTE (partiel)
END PROCESS	-

Tableau A.1: Contenu des fiches de l'interface

ANNEXE B: DESCRIPTION DES FENETRES

Les figures B.1 à B.5 présentent l'aspect des différentes fenêtres (principale et de description des éléments), tandis que la figure B.6 à présente l'aspect des fenêtres temporaires (avertissements, liste de ressources, aide...).

Les noms des boutons ainsi que les commentaires affichés dans les fenêtres se trouvent dans le fichier d'entête "map1CX.h" qui contient aussi la déclaration des fonctions du contrôleur d'écran (voir annexe D).

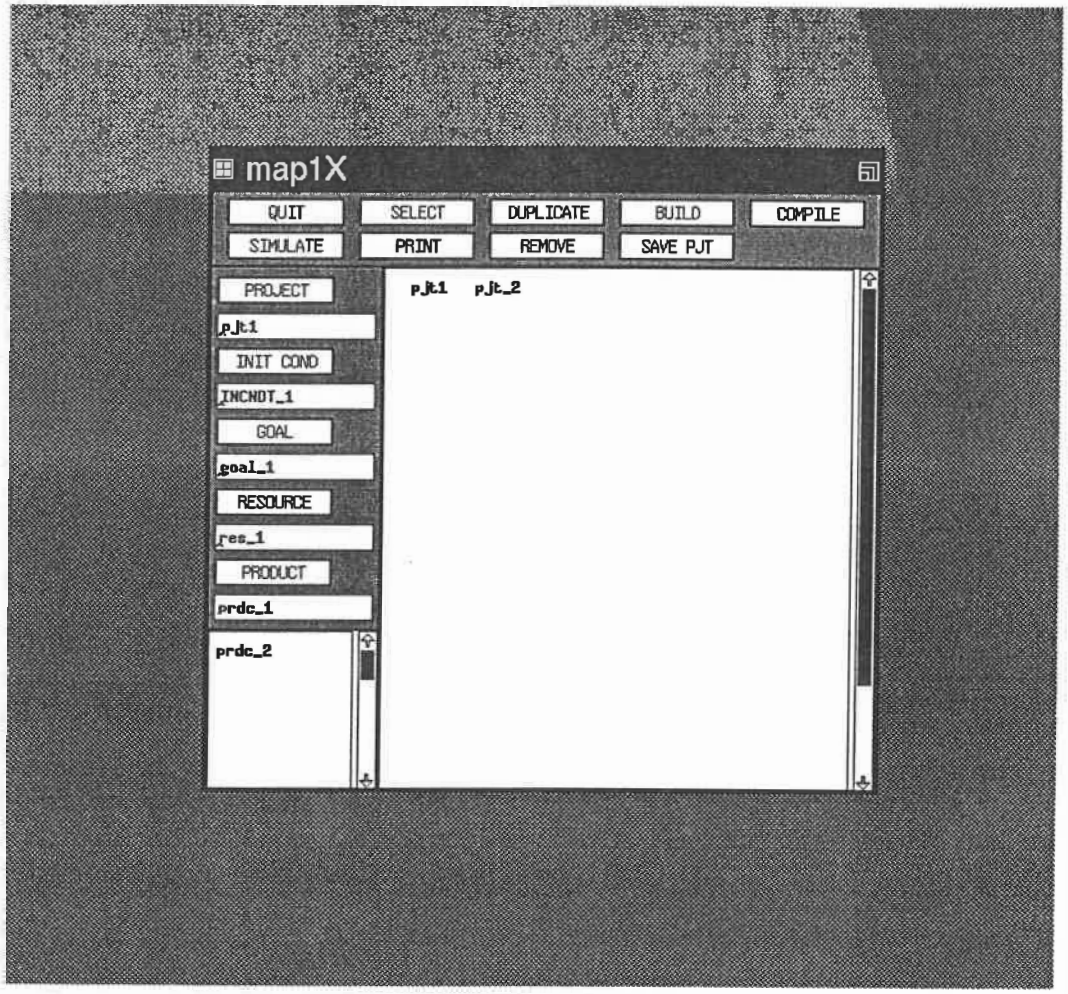


fig B.1: Fenêtre principale

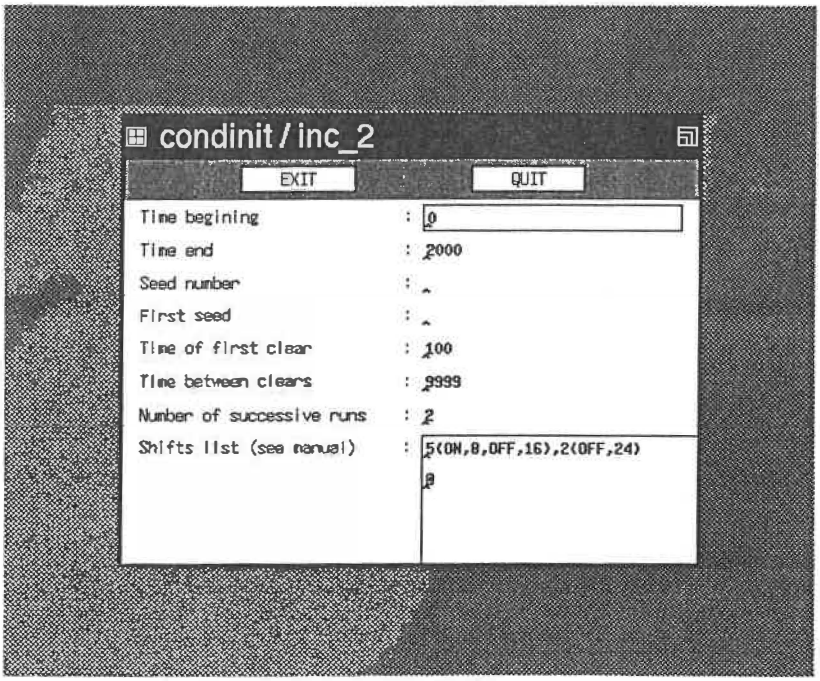


fig B.2: Fenêtre des conditions initiales

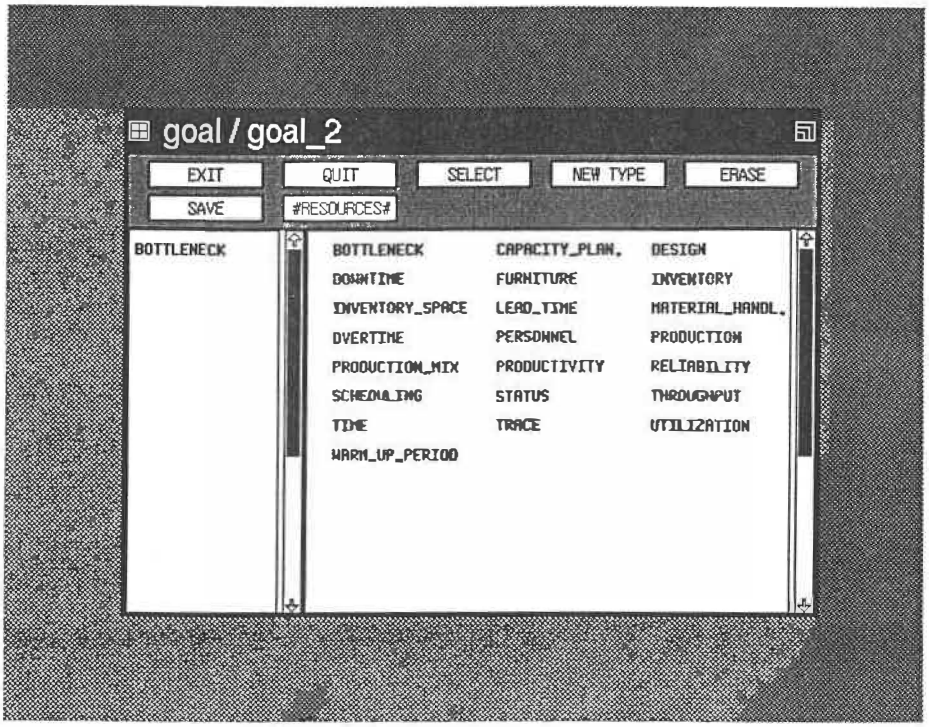


fig B.3: Fenêtre des objectifs

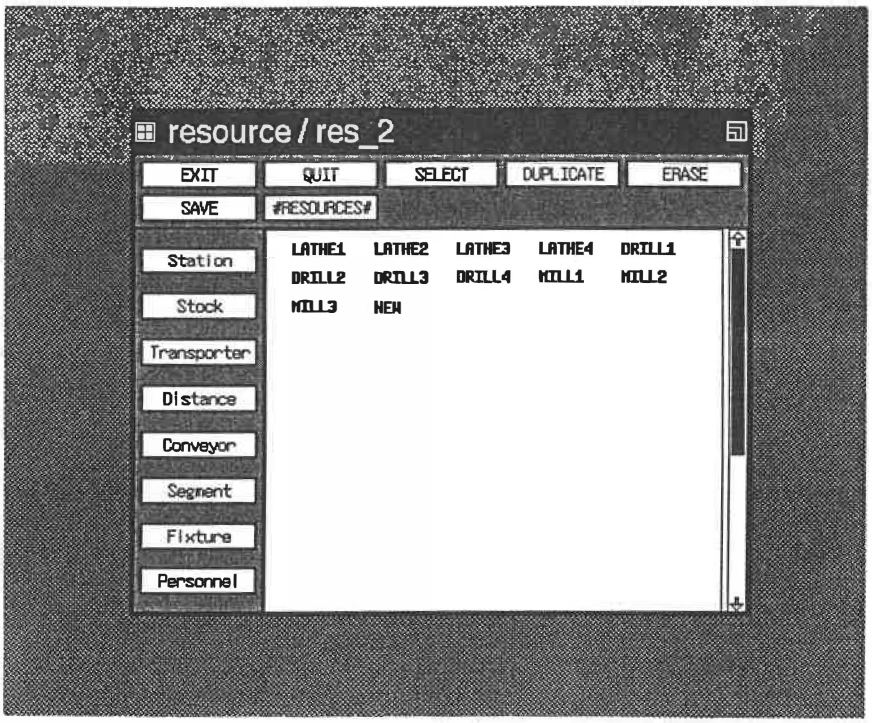


fig B.4: Fenêtre des moyens

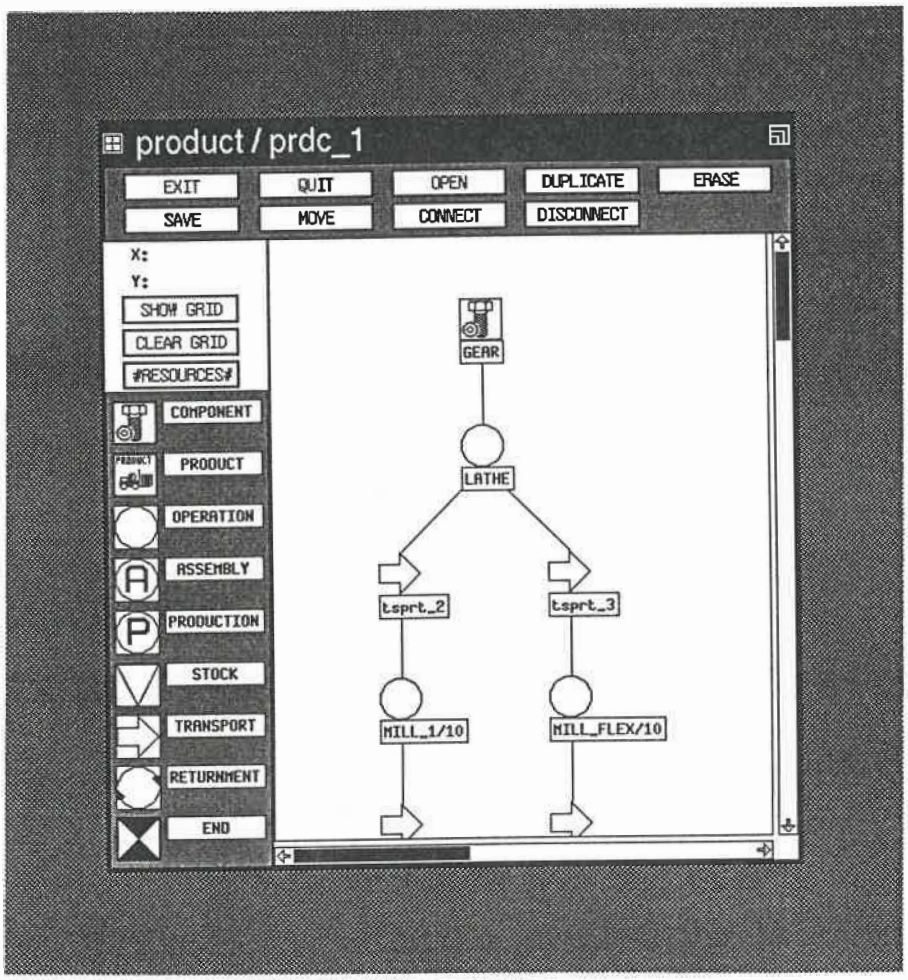


fig B.5: Fenêtre des produits

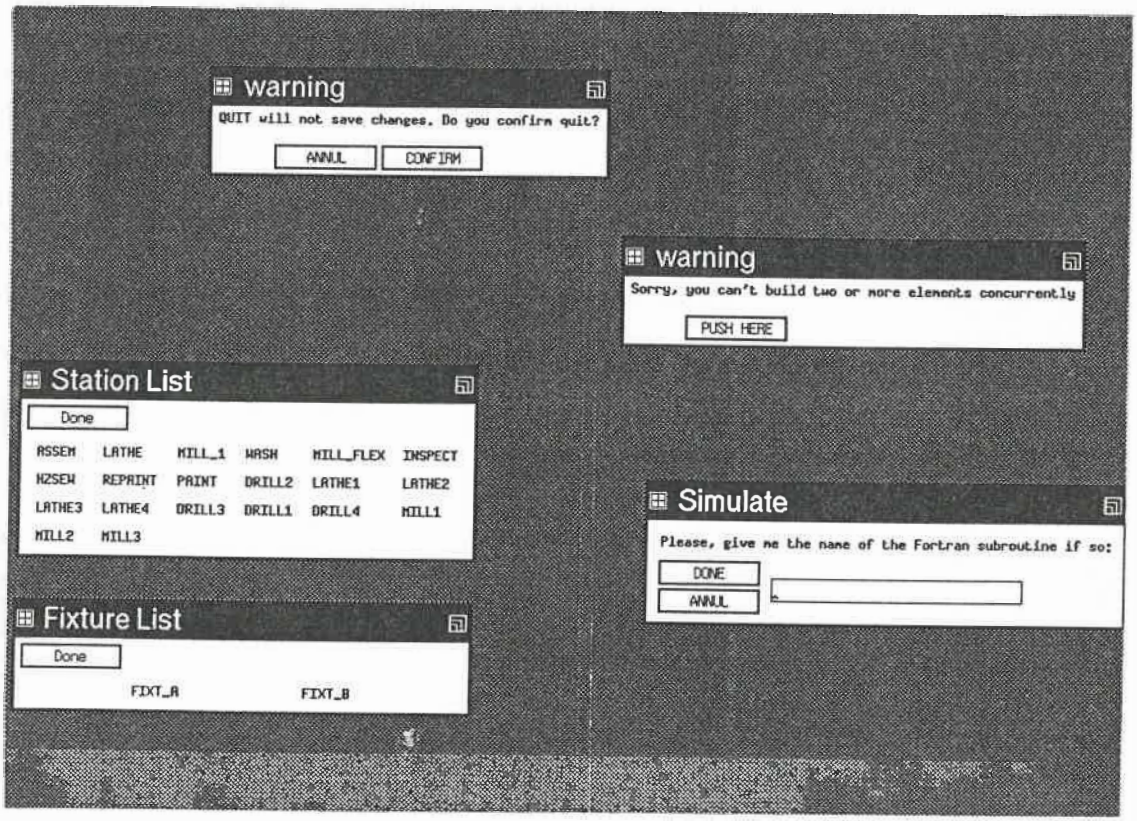


fig B.6: Exemples de fenetres temporaires

ANNEXE C: LES INTERFACES DES OBJETS ET LE CONTROLEUR DE LA BASE DE DONNEES

Les pages suivantes reprennent le fichier d'entête de la partie du logiciel écrite en C++ (fichier "map1CPP.h") et présentent succintement la foncionnalité des principales interfaces des objets.

```
class ROUTE
{
  public:
    char *get_end_rout(char *routnm);
```

permet d'indiquer le nom de la STATION suivante à la fin d'un énoncé ROUTE

```
};
/*************************************************************/
class AD_Fiche
{
  protected:
    int is_long, is_split, nsuc;
```

indiquent respectivement si la fiche est de type BC_Longfiche, si elle a déjà été compilée et le nombre de ses successeurs (cas des graphes de déroulement matière)

```
    int type, nfield, ncomt, nlstfld;
```

type de fiche, nombre de champs, de commentaires et de champs dans la liste du dernier champ

```
    char name[MAXCHAR], project[MAXCHAR];
```

nom de la fiche et nom du projet pour l'énoncé BEGIN

```
    char comment[MAXCOMT][MAXCHARCOM];
    char field[MAXFILD][MAXCHAR];
    char last_field[MAXFILD][MAXCHAR];
```

commentaires, champs et liste du dernier champ

```
    int mode;
```

indique le type de STATION (REGULAR, ASSEMBLY...)

```
    char succ[MAXNAME][MAXCHAR];
liste de successeurs pour une icône d'opération
    char pred[MAXCHAR];
    char last_node[MAXCHAR];
dernier noeud atteint par un convoyeur
    ROUTE end_route;
    int x_icon, y_icon, rank;
coordonnées de l'icône dans le graphe et numero d'ordre
    virtual int is_standard(char *nm) {}
détermine s'il s'agit d'un rapport standard de MAP/1 ou non
    virtual char  *cat(char *nm, int d, int f, char *delim,
                       int nd);
ajoute à la suite de nm les champs field[d] à [f-1] séparés
par les caractères délim[nd] et renvoie la nouvelle chaîne
    virtual char *find_day(char *date) {}
trouve la date du jour pour l'énoncé BEGIN
  public:
    AD_Fiche(int tp);
initialise la fiche de type tp
    AD_Fiche(char *nm) {}
initilalise une fiche DEFINE REPORT
    int get_type() { return type; }
    char *get_name() { return name; }
    char *get_succ(int i) { return succ[i]; }
    char *get_pred() { return pred; }
    int get_ncomt() { return ncomt; }
```

```
    int get_nfield() { return nfield; }
    int get_size() { return is_long; }
    int get_split() { return is_split; }
    char *get_field(int i) { return field[i]; }
    char *get_comment(int i) { return comment[i]; }
    virtual void init_values();
```

charge les valeurs par défaut dans field et last_field pour une nouvelle fiche

```
    virtual void get_all_fields(FILE *fptr);
```

charge les valeurs déjà sauvée pour une fiche qui existe

```
    virtual char *get_report_name() {}
    virtual char *get_last() {}
    virtual char *get_last_field(int i) {}
    int get_rank() { return rank; }
    int get_x() { return x_icon; }
    int get_y() { return y_icon; }
    virtual int get_nlstfld() { return 0; }
    virtual void set_name();
    virtual void set_report_name(char *nm) {}
    void make_name(char *delim, int k0, intt k1);
```

construit le nom de la fiche en regroupant field[k0] et [k1] séparés par delim

```
    void set_pjct(char *nm) { sprintf(project, "%s", nm); }
    virtual void set_last(char *nm) {}
    void set_rank(int i) { rank = i; }
    void set_pos(int x, int y);
```

```
    virtual void set_mode(int mode) {}
    void set_pred(char *nm) { sprintf(pred, "%s", nm); }
    void set_succ(char *nm);
    virtual void fill();
remplit les  champs de la fiche avec les valeurs retrouvées
dans un fichier temporaire
    virtual void save(FILE *fptr);
    virtual void save() {}
    virtual void erase() {}
    virtual void split(FILE *fptr) {}
répartit les champs de la fiche dans les énoncés MAP/1
    void duplic_name(int i)
                    { sprintf(name, "%s%d", name, i); }
    virtual void cut_field();
enlève le caractère '\n' en fin de champ
};
/*************************************************************/
class BC_Longfiche: public AD_Fiche
{
  protected:
    char *catlast(char *nm);
  public:
    BC_Longfiche(int tp) : (tp) { nlstfld = 1; }
    BC_Longfiche(char *nm) : (nm) {}
    void init_values();
    void get_all_fields(FILE *fptr);
```

```
    char *get_last_field(int i) { return last_field[i]; }
    int get_nlstfld() { return nlstfld; }
    void fill();
    virtual void save(FILE *fptr);
    void cut_field();
};
/*************************************************************/
class BEGIN: public BC_Longfiche
{
  protected:
    char *find_day(char *date);
  public:
    BEGIN(int tp) : (tp) {}
    void set_name() { sprintf(name, "begin"); }
    void split(FILE *fptr);
};
/*************************************************************/
class DFRPT: public BC_Longfiche
{
  protected:
    int is_standard(char *nm);
  public:
    DFRPT(int tp) : (tp) {}
    DFRPT(char *name);
    void save();
    void erase();
```

```
  void split(FILE *fptr);
};
/*************************************************************/
class RPORT: public BC_Longfiche
{
  public:
    RPORT(int tp) : (tp) {}
    void set_name() {}
    char *get_report_name() { return name; }
    void set_report_name(char *nm)
                    { sprintf(name, "%s", nm); }
    void split(FILE *fptr);
};
/*************************************************************/
class TRFRM: public BC_Longfiche
{
  public:
    TRFRM(int tp) : (tp) {}
    void set_mode(int i) { mode = i; }
    void split(FILE *fptr);
};
/*************************************************************/
class STOCK: public BC_Longfiche
{
  public:
    STOCK(int tp) : (tp) {}
```

```
    void split(FILE *fptr);
};
/*************************************************************/
class FIXTR: public AD_Fiche
{
  public:
    FIXTR(int tp) : (tp) {}
    void split(FILE *fptr);
};
/*************************************************************/
class TRNSP: public BC_Longfiche
{
  public:
    TRNSP(int tp) : (tp) {}
    void split(FILE *fptr);
};
/*************************************************************/
class DISTN: public BC_Longfiche
{
  public:
    DISTN(int tp) : (tp) {}
    void set_name();
    void split(FILE *fptr);
};
/*************************************************************/
class CONVR: public AD_Fiche
```

```
{
  public:
    CONVR(int tp) : (tp) {}
    void set_last(char *nm)
                    { sprintf(last_node, "%s", nm); }
    char *get_last(char *nm) { return last_node; }
    void split(FILE *fptr);
};
/*************************************************************/
class SEGMT: public AD_Fiche
{
  public:
    SEGMT(int tp) : (tp) {}
    void set_name() { AD_Fiche::make_name("->", 1, 3); }
    void split(FILE *fptr);
};
/*************************************************************/
class PERSL: public BC_Longfiche
{
  public:
    PERSL(int tp) : (tp) {}
    void split(FILE *fptr);
};
/*************************************************************/
class COMPT: public BC_Longfiche
{
```

```
  public:
    COMPT(int tp) : (tp) {}
    void split(FILE *fptr);
};
/*************************************************************/
class PRUCT: public AD_Fiche
{
  public:
    PRUCT(int tp) : (tp) {}
    void split(FILE *fptr);
};
/*************************************************************/
class OPERN: public AD_Fiche
{
  public:
    OPERN(int tp) : (tp) {}
    void set_name() { AD_Fiche::make_name("/", 0, 1); }
    void split(FILE *fptr);
};
/*************************************************************/
class ASSEM: public BC_Longfiche
{
  public:
    ASSEM(int tp) : (tp) {}
    void set_name() { AD_Fiche::make_name("/", 0, 1); }
    void split(FILE *fptr);
```

```
};
/*************************************************************/
class PRODN: public BC_Longfiche
{
  public:
    PRODN(int tp) : (tp) {}
    void set_name() { AD_Fiche::make_name("/", 0, 1); }
    void split(FILE *fptr);
};
/*************************************************************/
class INVTY: public AD_Fiche
{
  public:
    INVTY(int tp) : (tp) {}
    void set_name() { AD_Fiche::make_name("/", 0, 1); }
    void split(FILE *fptr);
};
/*************************************************************/
class TSPRT: public AD_Fiche
{
  public:
    TSPRT(int tp) : (tp) {}
    void set_name() { sprintf(name, "tsprt_%d", rank); }
    void split(FILE *fptr);
};
/*************************************************************/
```

```
class RETRN: public AD_Fiche
{
  public:
    RETRN(int tp) : (tp) {}
    void set_name() { sprintf(name, "retrn_%d", rank); }
    void split(FILE *fptr);
};
/*************************************************************/
class ENDPC: public AD_Fiche
{
  public:
    ENDPC(int tp) : (tp) {}
    void split(FILE *fptr);
};
/*************************************************************/
class AC_Application
{
  protected:
    char name[MAXCHAR];
    char path[MAXCAR];
chemin  d'accès  pour  arriver  jusqu'à l'élément depuis le
répertoire /
    FILE *fptr_AC;
    int status;
public:
    AC_Application(char *nm, int order);
```

```
initialise l'application de type order et de name nm
    void set_name(char *nm) { sprintf(name, "%s", nm); }
    char *get_name() { return name; }
    int get_status { return status; }
    virtual void remove();
    virtual void print() {}
    virtual void copy(char *nm);
};
/*************************************************************/
class BA_Projet: public AC_Application
{
  private:
    char ci[MAXCHAR];
    char gl[MAXCHAR];
    char rs[MAXCHAR];
    char pc[MAXPDCT][MAXCHAR];
    int npdct;
    void format();
s'assure après la compilation de tous les éléments du
projet que la longueur des lignes ne dépasse pas 80
colonnes
  public:
    BA_Projet(char *nm, int order);
    void set_ci(char *nm) { sprintf(ci ,"%s", nm); }
    void set_gl(char *nm) { sprintf(gl ,"%s", nm); }
    void set_rs(char *nm) { sprintf(rs ,"%s", nm); }
```

```
    void set_pc(int i, char *nm)
                    { sprintf(pc[i] ,"%s", nm); }
    void set_npdct(int m) { npdct = m; }
    char *get_ci() { return ci; }
    char *get_gl() { return gl; }
    char *get_rs() { return rs; }
    char *get_pc(int i) { return pc[i]; }
    int get_npdct() { return npdct; }
    void save();
    void print();
    void compile();
    void simulate();
};
/*************************************************************/
class BB_Element: public AC_Application
{
  protected:
    AD_Fiche *fiche[MAXFORM];
    int nfch, nlinks, rank_max;
    int link[MAXLINK][2];
    AD_Fiche *conv[MAXCONV];
  public:
    BB_Element(char *nm, int order);
    virtual void set_project(char *nm) {}
    virtual AD_Fiche *get_from_wdw(int x, int y) {}
```

retrouve une fiche à partir des coordonnées de l'icône

```
    virtual AD_Fiche *get_from_rank(int rk) {}
```

renvoie un pointeur sur la fiche de rang k

```
    virtual void redraw() {}
```

constitue un fichier avec le nombre d'icônes du graphe, leur type leur nom et leur position

```
    virtual void refresh() {}
```

constitue un fichier avec le nombre de liens entre les icônes et pour chaque lien le rank des deux icônes

```
    virtual int get_connect_o(AD_Fiche *frm) {}
```

renvoie le nombre d'icônes auxquelles frm est connectée en amont et met leur liste dans un fichier temporaire

```
    virtual int get_connect_d(AD_Fiche *frm) {}
```

comme ci-dessus mais pour les connections en aval

```
    virtual int get_nlinks() {}
    virtual  int  connect(AD_Fiche *frm_o, AD_Fiche *frm_d)
                                    {}
```

crée un lien entre frm_o et frm_d

```
    virtual int disconnect(AD_Fiche *frm_o,
                           AD_Fiche *frm_d) {}
```

efface le lien entre frm_o et frm_d s'il existe

```
    virtual int disconnect_all(AD_Fiche *frm) {}
```

efface tous les liens entre frm_o et les autres icônes

```
    void set_form(AD_Fiche *form) { fiche[nfch] = form; }
    virtual void add() { nfch++; }
    void substract() { nfch--; }
    AD_Fiche *get_form(int i) { return fiche[i]; }
```

```
    int get_nfch() { return nfch; }
    int retrieve(char *nm);
retrouve le numéro de la fiche de nom nm
    void retrieve_all(int m);
retrouve toutes les fiches de type m et  met leur  nom dans
un fichier temporaire
    virtual void save();
    AD_Fiche *duplicate(int j);
    void erase(char *nm);
    void erase_last();
    virtual void compile();
};
/*************************************************************/
class CA_Condinit: public BB_Element
{
  public:
    CA_Condinit(char *nm, int order);
    void set_project(char *nm) { fiche[0]->set_pjct(nm); }
    void save();
};
/*************************************************************/
class CB_Objectif: public BB_Element
{
  public:
    CB_Objectif(char *nm, int order);
    void save();
```

```
    void compile();
};
/*************************************************************/
class CC_Moyen: public BB_Element
{
  protected:
    int non_yet_succ(int sp, char *command);
indique si  le convoyeur  # sp  est déjà  dans la liste des
successeurs du convoyeur associé au fichier "command"
    void get_all_succ(int base, int  i, char  *command, int
                      ncv);
reconstitue récursivement  la liste des STATIONs atteint en
emrpuntant un convoyeur donné après une divergence
  public:
    CC_Moyen(char *nm, int order);
    void compile();
};
/*************************************************************/
class CD_Produit: public BB_Element
{
  protected:
    int rk_part;
    void set_rk_part(int i) { rk_part = i; }
    int get_rk_part() { return rk_part; }
    void recur_split(int rk, int cpt, int i, FILE *ptr);
reconstitue de façon récursive lesénoncés ROUTE pour chaque
```

```
produit (COMPONENT et PRODUCT) d'un graphe
  public:
    void remove();
    void copy(char *nm);
    void save();
    void print() {}
    AD_Fiche *get_from_wdw(int x, int y);
    AD_Fiche *get_from_rank(int rk);
    void redraw();
    void refresh();
    int get_connect_o(AD_Fiche *frm);
    int get_connect_d(AD_Fiche *frm);
    int get_nlinks();
    int connect(AD_Fiche *frm_o, AD_Fiche *frm_d);
    int disconnect(AD_Fiche *frm_o, AD_Fiche *frm_d);
    int disconnect_all(AD_Fiche *frm);
    void add();
    void compile();
};
/*************************************************************/
```

ANNEXE D: LE CONTROLEUR D'ECRAN

Les pages suivantes présentent la partie du fichier d'entête du contrôleur d'écran (fichier "map1CX.h") où sont définies ses fonctions. Les fonctions invoquées par le cliquage d'un bouton et donnant accès au contrôleur de BDD sont généralement suffixées par CK ou par EH pour faire référence au type de méthode Xt utilisé (CK pour "callback" et EH pour "event handler"). Ce fichier d'entête définit les fonctions dont le code apparaît dans les fichiers "map1X.c" et "libmap.c". Pour plus de détail concernant l'usage des fonctions CK et EH, voir l'annexe E.

```
/*************************************************************/
Routines de création des fenêtres
/*************************************************************/
   void make_form_envt(Widget parent, char *nm);
   void make_goal_envt(char *nm);
   void make_res_envt(char *nm);
   void make_prdc_envt(char *nm);
/*************************************************************/
Routines de création du panneau supérieur et de ses boutons
et affectation des boutons aux fonctions CK
/*************************************************************/
   Widget create_cmd_panel(Widget parent, int type);
   void add_clbk(int type);
/*************************************************************/
Fonctions connectées aux boutons du panneau supérieur de la
fenêtre principale
/*************************************************************/
   void selectCK(Widget w, caddr_t dt, caddr_t cl);
   void buildCK(Widget w, caddr_t dt, caddr_t cl);
   void copyCK(Widget w, caddr_t dt, caddr_t cl);
   void send_to_copyCK(Widget w, Widget dest, caddr_t cl);
       retrouve le nom du nouveau fichier après une copie
   void removeCK(Widget w, caddr_t dt, caddr_t cl);
   void printCK(Widget w, caddr_t dt, caddr_t cl);
   void compileCK(Widget w, caddr_t dt, caddr_t cl);
   void simulateCK(Widget w, caddr_t dt, caddr_t cl);
```

```
    void send_to_srunCK(Widget w, Widget dest, caddr_t cl);
        envoie le nom du  fichier FORTRAN  associé au modèle
        lors de l'éxécution
    void pop_ask_name(Widget w, int mode);
        choisi la fenêtre à faire apparaître après copyCK ou
        simulateCK
    void terminateCK(Widget w, caddr_t dt, caddr_t cl);
    void saveCK(Widget w, int subwin, caddr_t cl);
        invoquée par SAVE pour toutes les fenêtres
/*************************************************************/
Fonctions connectées aux boutons du panneau  supérieur pour
les fenêtres de description des éléments
/*************************************************************/
    void chooseCK(Widget w, caddr_t dt, caddr_t cl);
    void describeCK(Widget w, caddr_t dt, caddr_t cl);
    void createCK(Widget w, caddr_t dt, caddr_t cl);
    void eraseCK(Widget w, int is_icon, caddr_t cl);
    void duplicateCK(Widget w, int is_icon, caddr_t cl);
    void moveCK(Widget w, caddr_t dt, caddr_t cl);
    void connectCK(Widget w, caddr_t dt, caddr_t cl);
    void disconnectCK(Widget w, caddr_t dt, caddr_t cl);
    void exitCK(Widget w, int type, caddr_t cl);
    void add_reportCK(Widget w, caddr_t dt, caddr_t cl);
        rajoute le rapport créé avec le bouton NEW TYPE à la
        liste des rapports standards
    void quitCK(Widget w, int type, caddr_t cl);
```

```
    void quit_done(Widget w, int type);
        libère les objets affectés lors de la création de la
        fenêtre et initialise les indicateurs
/*************************************************************/
Fonctions appelées  lors du choix d'un bouton du panneau de
gauche
/*************************************************************/
    void request_listCK(Widget w, Widget part, caddr_t cl);
        fait  apparaître  la  liste  des  éléments  du  type
        correspondant au bouton w dans la fenêtre part
    void choose_iconCK(Widget w, int num, caddr_t cl);
        identique à  chooseCK mais en plus crée une nouvelle
        icône de type num
/*************************************************************/
Fonctions de sélection des items pour toute opération CK
/*************************************************************/
    void select_eltEH(Widget w, caddr_t dt, XEvent *evt);
    void select_1st_eltEH(Widget w, Widget parent,
                                    XEvent *evt);
    void select_stc_productEH(Widget w, caddr_t dt,
                                XEvent *evt);
    void select_iconCK(Widget w, Widget icon, caddr_t cl);
/*************************************************************/
Routines de création de champs éditables (sauf stc_prdc) et
redéfinition des touches de retour et de ses semblables
/*************************************************************/
```

```
    Widget make_stc_prdct(char *message);
    Widget gobot_text_W(char *message, int i);
    Widget last_gobot_text_W(char *message, int i);
    Widget create_one_line_text_widget(char *name,
                                       Widget parent);
    Widget create_gobot_text_widget(char *name,
                                    Widget parent);
/*************************************************************/
Routines de gestion des champs éditables de la fenêtre
principale et des fiches
/*************************************************************/
    void gobotEH(Widget w, int i, XEvent *evt);
    void gobotlastEH(Widget w, int i, XEvent *evt);
        ces deux fonctions permettent de passer au champ
        suivant dans une fiche
    void changefocusEH(Widget w, caddr_t dt, XEvent *evt);
    void disting_widget(Widget new_focus, int fin);
        gèrent le champ actif et le cadre qui l'entoure
    extern "C" void copy_buffer();
    extern "C" void XwTextClearBuffer(Widget w);
/*************************************************************/
Fonctions d'affichage et de sauvegarde des champs d'une
fiche
/*************************************************************/
    void show_form();
    void set_values();
```

```
/*************************************************************/
Routines utilitaires  poour  gérer  l'espace  de  dessin du
graphe
/*************************************************************/
   void track_mouseEH(Widget w, caddr_t dt, XEvent *evt);
   void clear_trackEH(Widget w, caddr_t dt, XEvent *evt);
       ces deux  fontions gèrent l'affichage de la position
       courante du pointeur
   void show_gridCK(Widget w, caddr_t dt, caddr_t cl);
   void clear_gridCK(Widget w, caddr_t dt, caddr_t cl);
       ces deux fonctions gèrent l'affichage du quadrillage
   void refreshEH(Widget w, caddr_t dt, XEvent *evt);
   void display_graph(Widget w);
       ces deux fonctions redessinent le graphe
/*************************************************************/
Routines utilitaires de création  et de  gestion des icônes
et du curseur dans l'espace de dessin
/*************************************************************/
   Widget create_icon(Widget w, int type, int x, int y,
                              char *name);
   void connect(Widget wo, Widget wd, int disc);
       permet de connecter ou de déconnecter wo et wd
   void connect_all((Widget_w, AD_Fiche *frm);
   void set_cursor(Window w, unsigned int mode);
   void reset_cursor(Window w);
/*************************************************************/
```

```
Routines d'appel des fenêtres temporaires
/*************************************************************/
   void warning(Widget parent, int text, int nbut,
                              char *name);
   void end_warningCK(Widget w, int rep, caddr_t cl);
   void helpCK(Widget w, int hlp, caddr_t cl);
   void end_popCK(Widget w, Widget pop, caddr_t cl);
/*************************************************************/
Routines d'appel des fenêtres pour les listes de ressources
/*************************************************************/
   void popupmenuCK(Widget w);
   void update_res_list(int type, char *nm, int eras);
   void show_res_list(Widget w, int type, caddr_t cl);
/*************************************************************/
Routines utilitaires
/*************************************************************/
   AC_Application *make_object(char *name, int type);
   AD_Fiche *make_form(int i);
   int can_connect(int or, int de);
       indique si  les icônes de type od et de peuvent être
       connectées
   int confirm_oldnew(Widget w, char *name, int type);
       vérifie si une application est nouvelle ou ancienne
   int convert_dest_Wi(Widget w);
       transforme le widget w en un  entier représentant un
       type d'élément ou de ressource
```

```
    Widget convert_dest_Iw(int dest);
        pratique l'opération inverse du précédent
    void create_atom(Widget w);
        crée les  atomes où  sont stockés les renseignements
        échangés  entre  les  fenêtres  par  les  fonctions
        change_* et retrieve_*
    void init_pixmap();
        crée les pixmap associés aux icônes
    void beep(Widget w);
        emet un bip
    void wait_pop_down();
        boucle infini pour attendre la réponse à un warning
    void null() {}
    void nullEH(Widget w, caddr_t dt, XEvent *evt);
    void invert_widget(Widget w);
        inverse la couleur du widget w sélectionné
/*************************************************************/
Fonctions  de  stockage  et  de récupération d'informations
dans des atomes
/*************************************************************/
    extern "C" void change_origine(Widget w, char *orig);
    extern "C" void change_name(Widget w, char *name);
    extern "C" void change_dest(Widget w, int n);
    extern "C" void change_rep(Widget w, int rep);
    extern "C" void change_window(Widget w, Window wdw);
    extern "C" int retrieve_dest(Widget w);
```

```
extern "C" int retrieve_rep(Widget w);
extern "C" char *retrieve_origine(Widget w);
extern "C" char *retrieve_name(Widget w);
extern "C" Window retrieve_window(Widget w);
```

ANNEXE E: MODE D'EMPLOI DU LOGICIEL

Généralités et installation

Le matériel requis pour utiliser le logiciel map1X est le suivant:

- une station de travail SUN,
- l'environnement graphique X11R4 ou suivant,
- le compilateur GNU g++, version 1.37 ou suivantes.
- le logiciel de simulation MAP/1.

Le logiciel et la base de données se composent des fichiers suivants:

dans le répertoire de travail:

- map1X: fichier éxécutable;
- icone_graph: fichier contenant le vecteur de bitmaps pour les icônes;
- can_tile: fichier contenant la trame du fond vide de la fenêtre de dessin des graphes;
- can_tile_b: fichier contenant la trame du fond quadrillée de la fenêtre de dessin des graphes;
- map1X.c: code source du contrôleur d'écran;
- map1CPP.h: code source du contrôleur de base de

données et des fiches;

- **libmap.c**: code source des fonctions utilitaires du contrôleur d'écran écrites en C;
- **map1CX.h**: fichier d'entête du contrôleur d'écran contenant la définition des variables globales;
- **const_def.h**: fichier contenant la liste des inclusions, la définition des constantes et des chemins d'accès aux répertoires de la base de données.
- **intrinsic.h** et **xos.h**: fichiers contenant une copie modifiée pour les rendre compatibles avec C++ des fichiers standards de X11 Intrinsic.h et Xos.h

dans le répertoire de départ (celui du "login"):

- **AppDefaults**: répertoire contenant le fichier de définition des caractéristiques des fenêtres;
- **res_kern**: répertoire contenant la liste des ressources rangées par types (utilisé pour "#RESOURCES#");
- **report_type**: répertoire contenant les fichiers de rapports types et le fichier **.report_std**;
- **default**: répertoire contenant les fichiers de définition des champs de commentaires et des valeurs par défaut pour chaque type de fiche;
- **project**: répertoire destiné à recevoir les fichiers de projet;
- **element**: répertoire regroupant les répertoires suivants: - **condinit**,

- goal,
- resource,
- product qui contient à son tour le répertoire ".graph".

Lors de l'installation du logiciel, il faut aller indiquer à la fin du fichier const_def.h les chemins d'accès absolus aux différentes parties de la base de données en remplaçant "/usr/users/map1" par le chemin du répertoire de "login" (obtenu en invoquant la commande UNIX "pwd" une fois entré sur le système), puis recompiler le logiciel en utilisant les commandes "make -f min" puis "make -f max".

Mise en route

Avant de mettre en route le programme, il convient d'indiquer au système l'endroit où il pourra retrouver les caractéristiques physiques des fenêtres. Elles sont dans le répertoire "AppDefaults", dans le fichier "Map1X". La commande pour réaliser cette initialisation est la suivante:

"setenv XAPPLRESDIR <path>/AppDefaults"

où <path> représente le chemin d'accès absolu menant jusqu'au répertoire. On peut faire exécuter automatiquement cette instruction en l'incluant dans le fichier ".login".

Il convient ensuite de se placer dans l'environnement X par la commande "x11". Une fois l'écran reconfiguré, on peut mettre en route le logiciel par la commande "map1X" qui provoque l'apparition de la fenêtre principale (figure I.1). Pour terminer une séquence de travail, l'usager doit sélectionner avec le pointeur le bouton "QUIT" de cette fenêtre (Il est possible de faire exécuter le programme plusieurs fois simultanément, mais cela est fortement déconseillé en raison du risque de travail parallèle sur le même fichier).

Les opérations de manipulation des composants d'un modèle se font généralement en deux étapes, quelle que soit la fenêtre dans laquelle on travaille:

- choix de l'élément sur lequel on désire effectuer un traitement,
- choix du traitement qu'on désire lui apporter et qui est matérialisé par un des boutons du panneau supérieur.

La fenêtre principale

A la mise en route, le logiciel fait apparaître la liste des projets existant en mémoire.

* "QUIT" : permet de sortir du logiciel sans sauver le projet courant.
* "SELECT" : sélectionne un élément affiché dans la liste de droite et le fait apparaître dans la colonne de gauche, dans la partie correspondant à son type; permet aussi quand il est choisi après avoir sélectionné l'élément de type produit situé sous le bouton "PRODUCT" de libérer cette place pour ajouter un nouveau produit au projet en cours tout en conservant également le premier.
* "DUPLICATE" : réalise une copie de l'élément choisi et la sauve sous le nom indiqué par l'usager.
* "BUILD" : accessible uniquement aux éléments situés dans le panneau de gauche à l'exception du projet courant et provoque l'apparition de la fenêtre associée à ce type d'élément.
* "COMPILE" : n'est accessible que pour le projet courant et permet de générer un fichier exécutable, à condition que le projet soit complet; réalise avant de compiler une sauvegarde en base de donnée du projet courant.
* "SIMULATE" : exécute la simulation et génére le fichier résultat; fait apparaître une sous-fenêtre où l'on peut indiquer le nom de la routine

FORTRAN associée au projet. Cette fenêtre disparaît une fois la simulation terminée.

* "PRINT" : imprime le fichier résultat (il vaut mieux vérifier au préalable son contenu et son volume).
* "REMOVE" : efface un élément définitivement dans le cas d'un élément apparaissant à droite ou uniquement pour le projet courant dans le cas d'un élément du menu de gauche.
* "SAVE PJT" : permet de sauver le projet courant en base de données.

Les boutons disposés dans la fenêtre de gauche font apparaître la liste des éléments d'un type donné qui ont été sauvés en base de données.

Les produits du projet courant qui sont disposés dans la fenêtre en bas à gauche sont inactifs (seul celui apparaissant sous le bouton "PRODUCT" est actif). Pour avoir accès à l'un d'eux, on le sélectionne avec le pointeur ce qui provoque son transfert dans la case située sous le bouton "PRODUCT" et rend inactif le précédent produit qui s'y trouvait.

La fenêtre des conditions initiales et des fiches

Cette fenêtre sert à présenter une fiche. Comme toutes les fenêtres de ce type, elle ne contient que 2 boutons: "EXIT" qui sauve la fiche et sort de la fenêtre et "QUIT" qui sort sans sauver.

Pour accéder aux champs lors de l'ouverture de la fenêtre, il faut placer le pointeur à l'intérieur du cadre qui entoure le premier pour le rendre actif ou bien déplacer le pointeur sur n'importe lequel puis le cliquer. Les données sont entrées par le clavier et le passage d'un champ à un autre peut s'effectuer de différentes façons:

- par sélection du nouveau champ avec le pointeur,
- par une des touches "enter" ou "return" ou "line feed" qui provoquent le passage au champ suivant,
- par les flèches du clavier qui provoquent le passage au champ suivant ou au champ précédent (une fois revenu en haut de la fiche choisir de passer au champ précédent entraine un "bip" sonore).

Quand le champ actif (matérialisé par un cadre) est le dernier de la fiche, passer au champ suivant provoque le retour au premier champ.

Les champs doivent être remplis en **MAJUSCULES.**

Certaines fiches, comme celle des conditions initiales, ont un dernier champ de longueur variable, c'est à dire qui peut à lui seul contenir plusieurs champs. Pour de telles fiches, cette liste de champs est encadrée en permanence. Le caractère '@' dans une chaîne de caractères indique au logiciel que le champ sur lequel on se trouve est le dernier de la liste; tous les champs suivants celui-là sont alors effacés.

Quand cette liste de champs devient trop longue pour l'espace qui lui est réservé dans la fiche, un curseur apparaît à la droite de cet espace et permet de se déplacer sur toute la longueur de la liste sans pour autant changer le champ actif.

Pour obtenir des renseignements sur la signification d'un champ, on sélectionne avec le pointeur le commentaire qui apparaît en regard du champ. Une fenêtre d'assistance est alors affichée.

La fenêtre des objectifs

Les rapports apparaissant dans la fenêtre de gauche sont ceux qui ont déjà été choisis pour le modèle, tandis

que ceux apparaissant à droite présentent les différents types disponibles.

Le rôle des boutons qui n'ont pas été préalablement définis est le suivant:

* "SELECT" : s'emploie aussi pour choisir un rapport apparaissant dans la fenêtre de gauche.
* "NEW TYPE" : crée un nouveau type de rapport à partir des rapports standards suivants: DOWNTIME, OVERTIME, TIME, STATUS, INVENTORY, PRODUCTION, THROUGHPUT et UTILIZATION.
* "ERASE" : efface un des rapports apparaissant à gauche ou un de ceux créés avec "NEW TYPE".
* "#RESOURCES#": fait apparaître un menu proposant les différents types de ressources puis, après le choix de l'une d'elles, crée une fenêtre présentant les noms des ressources de ce type existant dans l'ensemble des modèles.

NB: les boutons "SAVE", "EXIT" et "QUIT" s'utilisent sans choisir auparavant un élément d'une des fenêtres.

La fenêtre des moyens

* "DUPLICATE" : copie dans une nouvelle fiche le contenu des champs d'une fiche qui existe déjà.

Les boutons de gauche font apparaître la liste des éléments d'un type donné pour le modèle en cours.

Il faut aussi noter que le trajet d'un convoyeur ne peut admettre un point de divergence (séparation en plusieurs convoyeurs) qu'au départ ou à l'arrivée de son parcours.

La fenêtre des produits

* "OPEN" : identique à "SELECT" mais ne s'applique qu'aux icônes appartenant au graphe.
* "DUPLICATE" : positionne en plus la nouvelle icône créée.
* "MOVE" : déplace une icône du graphe.
* "CONNECT" : relie l'icône sélectionnée avant l'invocation de ce bouton à celle choisie après.
* "DISCONNECT": efface le lien existant entre l'icône sélectionnée avant ce bouton et celle choisie après; efface tous les liens d'une icône si on choisi la même avant et après.

Les boutons "SHOW GRID" et "CLEAR GRID" permettent respectivement d'afficher et d'effacer une grille destinée à faciliter la construction du graphe. En mode positionnement d'icônes, les coordonnées courantes du pointeur sont disposées en face de "X:" et de "Y:".

Les icônes présentées dans le menu de gauche servent à choisir l'élément à rajouter au graphe. Après en avoir sélectionné une, il faut emmener le pointeur à l'endroit du graphe où elle doit apparaître puis cliquer à cet endroit pour la faire apparaître.

Pour construire un graphe, on doit respecter les règles suivantes:

- faire débuter chaque racine du graphe par une icône "component" ou "product",
- positionner en premier le composant majeur du produit final, c'est à dire, celui sur lequel tous les autres viennent se rapporter,
- séparer toujours deux opérations (normale, d'assemblage, de production ou de stockage) par un transport ou un retournement (ou un produit dans le deuxième et le troisième cas),
- faire se terminer les branches du graphe par une icône de type "product" ou "end pc",
- les icônes de type "component" ne peuvent apparaître qu'en début de graphe,
- les icônes de type "product", si elles apparaissent au milieu du graphe, jouent le même rôle qu'un retournement ou un transport.

ÉCOLE POLYTECHNIQUE DE MONTRÉAL
3 9334 00290750 7