MASTER EN SCIENCES INFORMATIQUES

Une approche déductive à la spécification et au développement d'une application de gestion en temps réel

Dubois, Eric

Award date:
1981

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES

N.D. DE LA PAIX. NAMUR.

Institut d'Informatique

ANNEE ACADEMIQUE 1980-1981

UNE APPROCHE DEDUCTIVE A LA SPECIFICATION ET AU DEVELOPPEMENT D'UNE APPLICATION DE GESTION EN TEMPS REEL.

Eric DUBOIS.

Mémoire présenté en vue de l'obtention du grade de Licencié et Maître en Informatique.

Avant l'exposé de ce mémoire, je tiens à remercier tous ceux qui, à titre divers, ont contribué à la réalisation de ce travail.

Ma gratitude va tout d'abord à Monsieur A. VAN LAMSWEERDE, Professeur à l'Institut d'Informatique de Namur, qui a bien voulu accepté d'être le promoteur de ce travail. Pendant toute l'élaboration de ce dernier, j'ai bénéficié de sa haute compétence scientifique ainsi que de son soutien permanent. Qu'il trouve ici l'assurance de mes remerciements très sincères.

Qu'il me soit également permis de remercier Monsieur J-P. FINANCE, Professeur à l'Université de Nancy. Pendant les quelques mois que j'ai passés au sein de son équipe, il m'a initié, en matière de spécification, à la recherche constante d'une meilleure définition des objectifs ainsi qu'à la nécessité d'une démarche déductive et rigoureuse.

Je tiens aussi à remercier Monsieur J-F. DUFOURD, également Professeur à l'Université de Nancy, qui, lors de mon séjour, m'a beaucoup aidé de ses conseils et de ses remarques sur diverses versions de mes essais de spécification.

Ces remerciements s'adressent également à Monsieur J-L. HAINAUT, Professeur à l'Institut d'Informatique de Namur, pour l'aide accordée lors de la modélisation des données. La plupart des idées développées dans ce travail n'auraient certainement pas vu le jour sans ses recherches dans ce domaine.

Monsieur F. BODART, Professeur à l'Institut d'Informatique de Namur, a bien voulu s'intéresser à ce travail; je tiens à le remercier pour l'ensemble des conseils et remarques qu'il a bien voulu formuler lors d'une version antérieure de ce mémoire.

Enfin, ces remerciements ne seraient pas complets si je n'évoquais pas ici toutes les personnes qui, d'une façon quelconque, ont participé à la réalisation de ce mémoire.

° °
°

TABLE DES MATIERES.

- - - - - - - - - - - -

Introduction.

L'évolution dans la gestion des organisations et sa complexité croissante ont rendu nécessaires, ces dernières années, la conception et la réalisation d'outils permettant aux gestionnaires de disposer d'informations utiles à leur tâche d'analyse et de décision. Dans la gestion d'une organisation, les systèmes d'information et plus particulièrement ceux qui sont automatisés peuvent être considérés comme indispensables dans la mesure où ils assurent la communication entre le système opératoire (correspondant aux interactions de flux de biens et de services), le système de gestion (associé aux comportements décisionnels) et l'environnement. Nous retiendrons comme définition d'un système d'information celle de N. C. CHURCHILL, C.H. KRIEBEL et A.C. STREDY citée dans (LEM,73) :

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

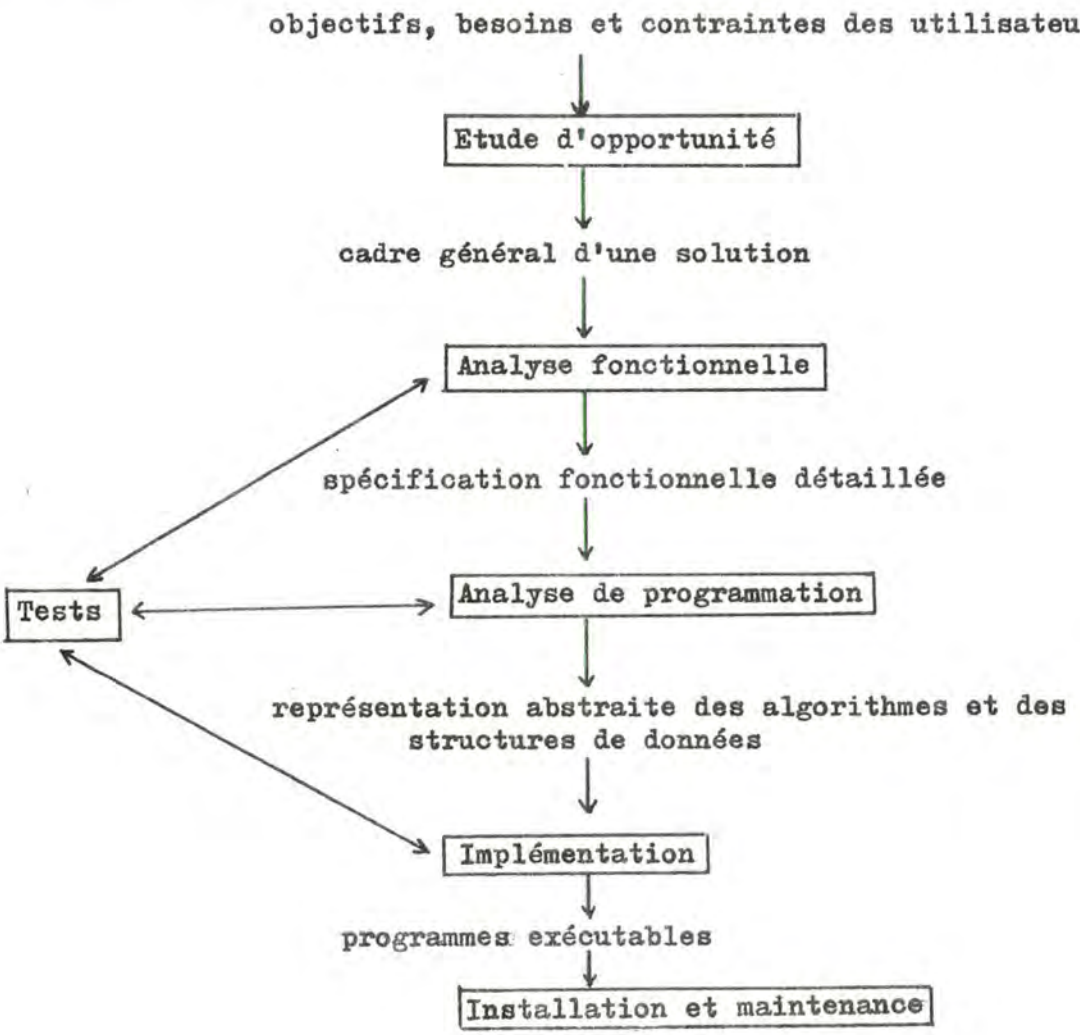
En dépit de nombreuses tentatives, il existe très peu de méthodes assurant une conception, une réalisation et une exploitation effectives d'un système d'information. La difficulté réside dans le fait qu'il faut appréhender un sujet qui relève à la fois des sciences exactes, des sciences humaines (en ce compris des facteurs socio-économiques et organisationnels), et qui en outre, est tributaire de l'évolution extrêmement rapide des moyens techniques disponibles.

Les facteurs d'échec que l'on peut rencontrer reflètent bien cette variété; nous nous contenterons d'en énumérer quelques-uns :

- la conception du système ne répond pas aux besoins en informations des utilisateurs de manière satisfaisante;
- la conception de système ne tient pas compte de certaines contraintes dictées par l'existant;

- la mise en place du système reflète une mauvaise compréhension des desiderata des utilisateurs;
- la partie automatique du système offre des temps de réponse trop longs (en raison par exemple d'une mauvaise architecture des traitements ou des données);
- les délais de mise en oeuvre du système automatique sont trop longs (en raison par exemple de trop nombreuses erreurs de programmation);
- la maintenance du système face à une évolution de la technologie ou des besoins des utilisateurs est trop coûteuse.

Ces différents facteurs d'échec proviennent de différents types d'erreurs commises lors des différentes étapes constituant le cycle de vie d'un projet informatique. Une classification de ces étapes pourrait être la suivante (voir, par exemple, (BOD,8I)) :



Notons en passant que les deux premières étapes fournissent comme résultats des descriptions à la fois des parties automatiques et manuelles du système d'information. Les étapes ultérieures, quant à elles, ne concernent plus que la partie automatique du système. Cette volonté, au départ, de décrire le système d'information sans le limiter à sa partie automatique vise à éviter le danger décrit par (LEM,73), à savoir la tendance dans le chef de responsables d'organisations à réduire le système d'information au système informatique et ainsi à démissionner devant les informaticiens.

Dans le cadre de la démarche d'analyse décrite ci-dessus, le présent travail portera sur les étapes d'analyse fonctionnelle, d'analyse de programmation et d'implémentation; l'effort majeur s'est cependant centré sur l'analyse fonctionnelle. En effet, il est indispensable de disposer d'une description du système d'information qui soit précise, cohérente et complète avant de passer à sa réalisation. Une mauvaise analyse fonctionnelle se répercute sur chacune des étapes ultérieures :

- dans la plupart des cas, les utilisateurs sont mécontents du fonctionnement du système car celui-ci ne répond pas à leur attente;
- lors de l'analyse de programmation et lors de l'implémentation, des difficultés surgissent et des choix sont à faire, lesquels auraient dû être réglés lors de l'analyse fonctionnelle;
- il devient illusoire de concevoir des plans de test significatifs en vue de valider le système;
- les coûts et délais de maintenance du système deviennent prohibitifs.

Si, dans de nombreuses organisations, l'analyse fonctionnelle est sommaire, voire inexistante, cela peut s'expliquer :

- par le fossé qui existe entre, d'une part, les demandeurs qui désirent un "bon" système mais souvent sont incapables de définir ce qu'ils entendent par "bon" et, d'autre part, les responsables

de la réalisation qui se contentent de ce flou et se préoccupent avant tout de réaliser un système qui "tourne".

- par le manque de méthodes et d'outils permettant de poser le problème de manière systématique tout en aidant à le comprendre et à le maîtriser. Dans le cas particulier de l'informatique de gestion, la difficulté de maîtriser le problème ne provient pas tellement des éléments du système pris séparément (ceux-ci sont en général relativement faciles à appréhender), mais plutôt du nombre de ces éléments et des interrelations qui existent entre eux.

Ce mémoire a pour but :

- de proposer une démarche permettant de spécifier un système d'information de manière méthodique, compréhensible et maîtrisable;
- de suggérer un cadre formel sous-jacent;
- de montrer comment une telle démarche peut être structurée avec les étapes d'analyse de programmation et d'implémentation;
- de montrer l'applicabilité de cette démarche et de son articulation par leur mise en oeuvre sur une application non triviale de gestion de ventes par correspondance.

La base de notre travail est constituée par la "méthode déductive" développée à Nancy par C. Pair et son équipe (PAI,79), et utilisée par J-P. Finance comme méthode de spécification dans le cadre de la construction du premier énoncé d'un problème purement informatique et libre de toute contrainte organisationnelle. Les idées de base de cette approche sont les suivantes (FIN,79) :

- commencer par spécifier le résultat qui se trouve généralement décrit dans l'énoncé initial;
- essayer d'exprimer le résultat en fonction de données de "départ", en introduisant éventuellement des données "intermédiaires";

- décomposer la spécification du problème en isolant des sous-problèmes indépendants : l'introduction d'un intermédiaire constituant un nouveau sous-problème;
- structurer la spécification en utilisant la séparation entre les concepts d'"univers" (ensemble de données et de leurs propriétés) et d'"énoncé" (ensemble des fonctions d'accès définies en utilisant les propriétés des données du problème).

Pour étendre la méthode déductive de la spécification d'un programme à l'analyse fonctionnelle d'un système d'information, nous avons décidé de travailler au niveau de l'"application" au sens de (BOD,8I) : il s'agit d'un problème ayant une existence "quasi-autonome" (au sens de (SIM,74)), un flux homogène d'informations, une certaine permanence dans le temps, et communiquant avec d'autres applications par échange d'agrégats (informations représentant des propriétés d'état du système d'information). Cette application représente l'unité de base de conception et de mise en oeuvre d'un projet informatique; en particulier, elle devrait constituer l'unité de base minimale d'une analyse fonctionnelle (BOD,8I).

Pour faire de la méthode déductive, une méthode apte à décrire une application, nous avons dû :

- enrichir la méthode en prenant en compte l'environnement organisationnel. En effet, la méthode déductive permet de présenter une sorte de noyau du système, reprenant la logique inhérente à l'application à traiter, mais ne permet pas nécessairement d'appréhender un certain nombre des caractéristiques propres à l'environnement organisationnel considéré (certains traitements propres à l'organisation étudiée, des enchaînements particuliers entre traitements, des interfaces avec l'environnement du système automatique).
- assurer la prise en compte des contraintes globales de performance attendue par le demandeur;
- développer un cadre formel pour exprimer la description des différents éléments constituant l'application. Ce cadre permet de présenter, en paral-

lèle, une description en langue naturelle et une description formalisée du problème et des sous-problèmes traités;

- nous donner un outil de représentation graphique des relations entre les différents sous-problèmes traités.

Nous avons voulu juger de l'effectivité de la démarche proposée en montrant comment nous pouvions l'articuler sur les étapes ultérieures d'analyse de programmation et d'implémentation : d'une part, à partir de la description de la structure des données, nous pouvons dériver le modèles des accès nécessaires, et d'autre part, à partir de la description de la structure des traitements, nous pouvons définir une architecture des programmes basée sur une modularisation de type suivant :

- modules fonctionnels. Ils concernent les différents traitements associés aux sous-problèmes.
- modules interactifs. Ils gérent les interactions avec l'extérieur du système informatique (saisie et diffusion de données).
- modules gérant la concurrence éventuelle. Ils règlent les interférences entre différents modules.
- modules d'entrée-sortie et d'accès. Ils cachent les particularités du matériel et du logiciel de communication et de stockage des données.

Enfin, nous avons brièvement décrit les liens assurant le passage de l'analyse de programmation à l'implémentation des programmes et du modèle d'accès.

Ce travail est organisé de la manière suivante.

Un premier chapitre présente un état de l'art sur les méthodes de spécification. Après avoir défini l'activité de spécifier et argumenter sur sa nécessité, nous passons en revue quelques méthodes existantes.

Le deuxième chapitre constitue le coeur du travail, il expose la démarche :

- au niveau fonctionnel, en décrivant la spécification selon une analyse déductive (§1), en adaptant celle-ci à l'environnement organisationnel (§2) et en mentionnant les contraintes globales de performance (§3);

- au niveau organique, en définissant une architecture des traitements et des données (§4) puis en passant à la programmation et à la dérivation d'un modèle d'accès aux données (§5);
- au niveau de l'implémentation des programmes et du modèle d'accès (§6).

Un troisième chapitre est consacré au développement d'une application de gestion de vente par correspondance suivant la démarche décrite.

Chapitre I.

ETAT DE L'ART SUR LES METHODES DE SPECIFICATION.

Dans ce premier chapitre, nous passerons en revue quelques considérations concernant l'activité qui consiste à spécifier un système d'information.

Dans un premier paragraphe, nous nous préoccuperons d'une définition de cette tâche.

Le second paragraphe explicitera la nécessité et les raisons d'une "bonne" spécification.

Dans le troisième paragraphe, nous verrons, pour une spécification, les contraintes liées à son audience variée.

Au cours du quatrième paragraphe, nous essaierons de mettre en évidence un code minimal de bonne conduite concernant le contenu d'une spécification.

Le paragraphe cinq passera en revue respectivement les défauts et les qualités caractérisant les spécifications trouvées.

Enfin, le paragraphe six, à travers l'étude de trois méthodes de spécification, mettra l'accent sur les attributs du langage de description employé mais aussi sur les critères qui guident le spécificateur dans son approche du problème.

I.I. Définition de l'activité de spécifier.

Avant de nous plonger dans l'exposé des problèmes posés par la spécification d'un système d'information, il est fondamental de savoir ce que l'on entend exactement par cette activité. Pour essayer de mieux s'en rendre compte, nous allons présenter quelques définitions parmi tant d'autres et souligner leurs caractéristiques.

Commençons par celle donnée par le Larousse.

> "Spécifier quelque chose, c'est le déterminer, l'exprimer d'une manière précise". (LAR, 7I)

Complétons-la par quelques définitions données par des "hommes du métier".

> "Spécifier un problème, c'est fournir un cadre précis pour constituer sous une forme fiable et rigoureuse, pouvant éventuellement être traitée automatiquement, l'exposé du problème que l'on envisage de résoudre". (DEM, 79)
>
> "Une spécification est l'exposé précis des exigences auxquelles un produit doit satisfaire". (BAR, 77)
>
> "Spécifier un problème informatique, c'est le poser aussi précisément que possible en s'interdisant de penser prématurément à sa solution". (MEY, 80)

Il est assez remarquable de trouver dans ces trois définitions le mot "précis". Cela met bien l'accent sur la différence qui existe entre un problème et sa spécification ainsi que sur la difficulté de passer d'une description imprécise à une autre bien précise. Cependant, quant à la manière de spécifier, on peut constater une signification distincte entre le Larousse et les informaticiens. En effet, si pour le premier, la représentation binaire d'un programme constitue une expression précise du problème, pour les seconds, celle-ci ne peut convenir car elle constitue déjà une solution du problème.

Cette distinction problème-spécification-solution est importante. D'une part, a priori, passer du problème à une solution est difficile car c'est à la fois définir d'une manière non-ambiguë et rigoureuse le problème à traiter (quoi faire ?) et prendre en compte des idées de sa réalisation et de son exécution. D'autre part, comme le souligne la troisième définition, comment pourra-t-on rendre compte que le produit auquel on a abouti satisfait entièrement aux exigences requises par l'utilisateur si celles-ci ne sont pas précises et explicites au préalable ?

I.2. Pourquoi spécifier ?

Spécification, développement et maintenance.

La spécification d'un système d'information ne servira pas seulement a posteriori pour s'assurer que l'implémentation a bien répondu aux besoins, mais, a priori, constituera un document qui sera à la base du contrat liant le demandeur et les réalisateurs du système d'information. Pour cela, il faudra donc qu'il soit clair, non-ambigu et compréhensible pour les deux parties. De plus, parce que les besoins évolueront dans le temps, la spécification devra permettre de refléter facilement des modifications et, en tout cas, beaucoup plus clairement que les programmes eux-mêmes qui eux, présenteront des complexités supplémentaires dues aux optimisations réalisées (BAL, 79).

Spécification et validation.

Si, dans ces dix dernières années, nous avons vu naître un ensemble de termes tels que "programmation structurée", "programmation modulaire", "analyse descendante", "analyse ascendante", ce n'est sûrement pas par hasard, et cela reflète les préoccupations du monde de l'informatique concernant la conception, la réalisation et la maintenance de logiciels. Ces méthodes, cependant, ne font que résoudre une partie du problème car elles n'interviennent qu'au niveau de l'analyse organique et de la programmation.

Or, si on se limite à prendre uniquement en compte ces étapes dans le développement d'un logiciel, il va être très difficile, voire impossible, de vérifier l'adéquation entre le programme et le problème qu'il est censé résoudre. En effet, il s'agira de donner la preuve de la validité du programme par rapport à des concepts et des propriétés qui ne sont clairs, dans le meilleur des cas, que dans l'esprit d'une ou plusieurs personnes. Il va de soi qu'une telle validation ne peut être elle-même que peu fiable. De plus, cette validation sera compliquée :

- par la présence d'un trop grand niveau de détails (par exemple la définition de fichiers, messages, transactions obscurcit la notion d'objet "logique" dans l'esprit du demandeur) ,

-par l'aspect dynamique (procédural) de l'enchaînement des tâches.

Une telle attitude est donc à rejeter, excepté peut-être dans le cas de la réalisation personnelle de petits logiciels à son propre usage comme l'indique la figure I. On peut, cependant, objecter à une telle présentation le pourquoi de spécifications inutiles dans le cadre de petits logiciels et le manque d'erreurs de programmation pourtant fréquentes lors de la réalisation de gros logiciels.

utilisateur	= réalisateur	≠ réalisateur
documentation	minimale	importante
validation	minimale	indispensable
coût de modification	faible	élevé
spécifications	inutiles	nécessaires
information	tout dans la tête	très répartie
erreurs	de programmation	de spécification
	PETITS LOGICIELS PERSONNELS	GROS LOGICIELS

figure I. Différences entre petits et gros logiciels (d'après (GER,80)).

Par contre, en informatique de gestion, où la masse de données à traiter est très importante, la nécessité de validations plus systématiques est indispensable. Une possibilité consiste à introduire une spécification intermédiaire entre le problème et sa résolution (LIS,75a). Si cette spécification possède tout un ensemble de propriétés que nous décrirons ultérieurement dans le paragraphe I.5 , dont la principale est une définition précise du problème, on peut se donner des moyens systématiques de vérification de la cohérence des programmes par rapport à leur spécification. La difficulté consiste à se donner une méthodologie de spécification permettant de décrire totalement le problème. A nouveau, se convaincre de cette complétude ne peut se faire que d'une manière informelle mais sera plus facile si la spécification est aisément compréhensible.

Impact des spécifications dans le développement d'un projet (BOE,76).

Sur la figure 2, on constate qu'à travers les années,si les coûts du matériel ont continuellement baissé,les coûts du logiciel ont suivi exactement la courbe inverse; c'est ainsi qu'en 1975, 250 milliards de francs français ont été investis en informatique dont à peu près 200 sont des coûts de logiciel.

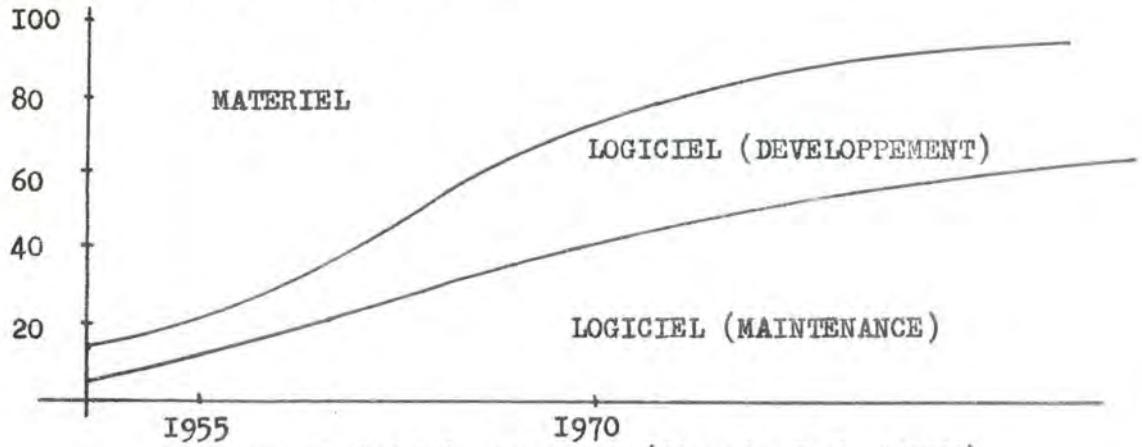


Figure 2. Coût du logiciel (d'après B.W. BOEHM) .

Quand on sait que 50 % des coûts sont des coûts de correction d'erreurs, il convient d'en voir la répartition comme indiquée dans la figure 3.

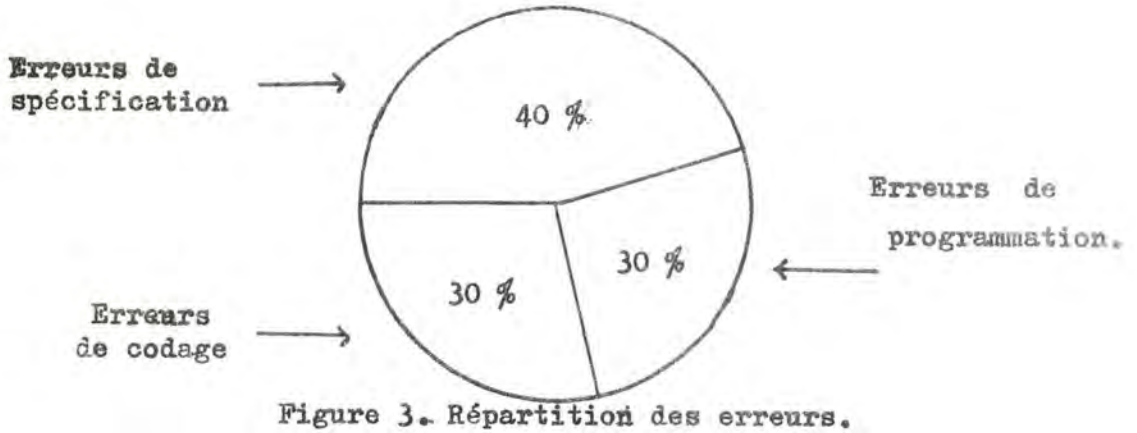


Figure 3. Répartition des erreurs.

On constate que les erreurs de spécification et de programmation représentent 70 % du nombre total d'erreurs, mais ce qui est plus grave, c'est le coût des erreurs comme l'indique la figure 4.

Type	% du nombre total d'erreurs	Coût de correction unitaire relatif	% du coût de correction total
Spécification	40 %	5	66 %
Programmation	30 %	2,5	25 %
Codage	30 %	I	9 %

Figure 4. Coût des erreurs.

Ce tableau montre clairement que le coût des erreurs de spécification représente en moyenne 2/3 du coût total de correction, soit le tiers du coût total du logiciel (270 milliards de francs français en I975). Le coût relatif des erreurs de spécification est très important car ces erreurs sont les plus tenaces parce que la plupart sont détectées au moment ou après la livraison du logiciel, ce qui signifie un retour en arrière important et par conséquent très coûteux.

Nous avons montré jusqu'ici le rôle nécessaire et critique de la spécification dans la réalisation d'un logiciel; nous avons également esquissé quelques critères caractérisant une bonne définition (précision, rigueur, non-ambiguïté, compréhensible et complète). Pour approcher d'autres qualités d'une spécification, il faut au préalable nous poser les questions

de savoir à qui elle est destinée et ce que l'on va précisément y mettre.

I.3. Pour qui spécifier ?

L'audience d'une spécification dépasse largement celle de ceux qui les écrivent pour s'adresser à tout un ensemble de personnes dont les attentes et les nécessités sont bien différentes. Ainsi :

- le demandeur devra pouvoir décider si les spécifications correspondent à ses besoins. Il lui faut donc un document qui expliquera clairement ce que sera et ce que fera le système décrit, éventuellement ce qu'il ne fera pas;
- les utilisateurs non-informaticiens devront, à partir des spécifications, trouver une définition non-ambiguë des tâches qu'ils doivent réaliser ainsi que des interfaces qui existeront entre eux et la machine;
- le chef de projet devra disposer d'une définition aussi complète que possible des attributions et des interfaces du système sous étude;
- le responsable de la réalisation du logiciel devra, à partir des spécifications, pouvoir évaluer les coûts, les délais d'implémentation et éventuellement désapprouver certaines spécifications;
- l'équipe de la réalisation devra mettre en oeuvre une structure du système permettant de réaliser les fonctions spécifiées et faire le design de la structure de données décrite. Elle assurera ensuite la programmation et la constitution des fichiers nécessaires tout en vérifiant, à tout moment, que ce qui a été réalisé répond exactement à ce qui a été spécifié;
- les responsables de la maintenance, lors d'erreurs ou d'évolution dans les spécifications, devront pouvoir juger de l'importance et des conséquences de telles modifications par rapport aux spécifications existantes.

Il apparaît donc que, vu le nombre important de personnes qui ont besoin des spécifications et vu leurs attentes parfois fortement différentes, la spécification devra posséder de nombreuses qualités. Cependant, il n'est pas évident que toutes ces qualités soient compatibles entre elles.

Ainsi la lisibilité des spécifications attendue par les utilisateurs non-informaticiens et les demandeurs peut être un obstacle face à l'utilisation d'un langage formel (par exemple, le langage mathématique) permettant d'exprimer les choses. De telles incompatibilités vont nous amener à juger les qualités des spécifications à leurs possibilités, telles quelles, vues sous différents angles (en effet,le niveau de détail requis n'est pas le même suivant les différentes personnes concernées), de satisfaire les besoins des différents individus; si ce n'est pas le cas, à la facilité, à partir des descriptions, de rédiger les documents nécessaires et répondant aux exigences des différentes personnes.

I.4. Que spécifier ?

Comme nous l'avons déjà signalé dans le paragraphe I.I., spécifier, au sens informatique du terme, c'est décrire un problème en répondant à la question: "que faut-il réaliser ?", cette question s'opposant à celle de "comment le réaliser ?" . Souvent, en effet,l'analyste du système produit une conception approximative (une idée de réalisation du système). Cette distinction étant faite, il s'agit de savoir quel sera le contenu d'une spécification. A ce sujet, plutôt que de relever toutes les alternatives que l'on peut répertorier dans la volumineuse littérature concernant le " Software Engineering ", nous avons opté pour la présentation et la comparaison du contenu des spécifications trouvées dans trois méthodologies différentes. Celles-ci donnent un assez bon aperçu de tout ce qu'on peut trouver dans d'autres techniques.

Commençons par la description des spécifications que l'on trouve dans les travaux de F.BODART et de son équipe de Namur (BOD, 79a).

- Spécification de la structure conceptuelle des données.

Elle se fera à l'aide du modèle entité-association dont les concepts de base sont les entités, les associations, les propriétés et les contraintes d'intégrité.

- Spécification de la structure de traitements.

Décomposition des traitements sur base d'une nomenclature hiérarchique (système,sous-système,application,phase,fonction).

Chacun des traitements sera décrit:

+ d'une manière statique: entrées,sorties (messages),collections d'informations et spécification pour les traitements d'une fonction sous forme de règles à respecter.

+ d'une manière dynamique: le modèle dynamique des traitements est basé sur les concepts de

- processus (attaché à chaque exécution d'une procédure).
- évènement (changement d'état du système connu par un message).
- point de synchronisation.

Parnas et Heninger (HEN, 80b), lors de la réécriture des spécifications du logiciel utilisé par l'avion A-7, ont donné la description suivante.

- Pour chaque input et output du système, on donne les caractéristiques suivantes:

+ attribution d'un nom symbolique: nom mnémotechnique identifiant la donnée.

+ les valeurs pouvant être prises.

+ la séquence d'instruction permettant d'acquérir la donnée.

+ les valeurs d'encodage montrant comment les valeurs prises par les variables ayant un nom mnémotechnique peuvent être transformées en des représentations binaires spécifiques.

+ la représentation des données montrant la location d'une valeur dans un mot de I6 bits.

En outre, pour chaque donnée-résultat, on décrit les fonctions portant sur elle.

- Chaque fonction est spécifiée par:

+ ses effets externes visibles.

+ une indication indiquant si la fonction est modifiable ou pas

et les éventuels changements auxquels elle devra pouvoir faire face.

+ des contraintes de temps.

+ la description des évènements associés à l'exécution d'une fonction.

+ les données de sortie.

+ l'expression des conditions caractérisant les valeurs des outputs.

+ les réponses face à des évènements non désirés.

Chez SADT (Structured Analysis and Design Technique) (ROS,77), on trouve:

- les actigrammes.

  Ils sont composés:

  + de boîtes où l'on décrit les traitements de manière informelle (texte),les inputs, les outputs et les fonctions de contrôle définissant les conditions d'activation des traitements. On indique également le processeur exécutant cette tâche.

  + de flèches reliant les boîtes et exprimant les contraintes relationnelles en termes de flux de données entre les boîtes.

- les datagrammes.

  Ils sont composés:

  + de boîtes où l'on décrit les données d'une manière informelle ainsi que les fonctions de création, d'utilisation et de contrôle de ces données. On indique également le support de mémorisation.

  + des flèches indiquant les différents traitements utilisant des données pour en créer d'autres.

Au vu de ces différentes techniques, nous pouvons définir un code minimal en matière de spécifications qui sera :

- une définition et les propriétés caractérisant l'ensemble des concepts liés au problème.

- une description de la structure des traitements.

Cette définition devrait se présenter sous la forme d'une hiérarchie de fonctions pour lesquelles on donnerait les données d'entrées, de sortie et les collections d'informations (informations présentant un caractère permanent et nécessitant une mémorisation) manipulées.On spécifierait également l'effet de la fonction; cependant, on se demande si une description externe, comme le fait Parnas, suffit. En effet, une description uniquement en termes de données d'entrées et de sorties accompagnée d'une brève explication concernant les traitements réalisés suffit-elle pour pouvoir identifier de manière non-ambiguë plusieurs fonctions ayant un effet équivalent et pouvant être paramétrisée? Pour chacune des fonctions , il faudra en outre, décrire les interfaces ainsi que les conditions d'activation des traitements. La réponse face à des évènements exceptionnels et le type des modifications probables seraient également des éléments intéressants à prendre en compte.

3/ - une description de la structure des données.

Le système devra être décrit non seulement sous la forme de traitements mais également sous la forme des données manipulées. En effet, celles-ci présentent des propriétés, des interrelations qu'il est important de définir en vue de choix de représentations ultérieures. En outre,comme la possibilité est offerte chez SADT, la présentation des fonctions travaillant sur chacune des données permet d'avoir une vue synthétique et complète (notion de type abstrait au sens de (LIS,75a) et (GUT,77)).

4/- une description des contraintes matérielles et logicielles de réalisation ainsi que des performances requises du point de vue de l'utilisation et de l'environnement externe.

Remarque.

On peut s'étonner, au niveau de la spécification de A-7, du grand nombre de détails hardware trouvés. Cependant,ceux-ci ont leur place au niveau fonctionnel car pour ce type de problème d'avionique, les caracté-

ristiques matérielles sont connues dès le départ et ne peuvent donc être considérées comme des choix de réalisation. Cela montre, en particulier, qu'au niveau des spécifications, d'après le type de problème à traiter, les besoins et les attentes peuvent être différents.

I.5. Qualités et défauts d'une spécification.

Dans les paragraphes précédents, nous avons soulevé quelques problèmes liés aux spécifications, et de là, donné de premières idées concernant les qualités d'une définition telles que la précision, la non-ambiguïté, la complétude quant à son contenu, la compréhensibilité pour différentes catégories de personnes. Avant d'essayer d'en formuler d'autres caractérisant un "bon" énoncé initial, il serait intéressant de présenter une classification des écueils couramment rencontrés dans un cahier de charge et de l'illustrer par quelques exemples.

Pour B.Meyer, (MEY,80), on peut détecter, chez le spécificateur, six " péchés capitaux ".

I°) Le silence.

Il s'agit d'éléments non spécifiés du problème. Ces erreurs sont très graves car si elles ne sont pas découvertes, cela signifie simplement qu'au stade de la réalisation, on devra faire des choix qui auraient dû être réglés au cours de la définition; de plus, il est à peu près sûr que les solutions retenues ne correspondront pas à l'attente du demandeur.

Exemple.

Une spécification trouvée dans (CLA,79) pour un problème est la suivante. " Compter combien de fois apparaît un mot dans un texte (on fait l'hypothèse qu'un mot n'est jamais coupé sur deux lignes) ".

Les silences proviennent souvent du fait que pour le spécificateur, quand il est proche du demandeur, certaines caractéristiques paraissent évidentes et ne doivent pas être explicitées. Ainsi, on n'a pas défini précisément ce qu'étaient un texte, une ligne, un mot, c'est-à-dire les concepts

de base liés au problème. Tout au plus, peut-on comprendre qu'un texte ainsi qu'une ligne sont formés de mots.

2°) Le bruit.

Il s'agit d'éléments de la spécification n'apportant pas de nouvelles informations sur le problème. En particulier, des variantes de ce bruit sont la redondance et le " repentir " (des caractéristiques propres à un élément n'apparaissant pas lors de sa définition mais bien lors de l'une de ses utilisations).

Exemple.

La spécification d'un problème de traitement de texte donné par (GOD,76) dit, entre autre:

"...Un mot est une suite non vide de caractères non de saut . Un saut est une suite de un ou plusieurs caractères de saut... La sortie du programme doit être... et au début du texte de sortie, s'il existe ... ".

Les bruits ne sont pas forcément gênants dans une spécification à condition qu'ils ne servent pas à redire la même chose dans des termes différents, ce qui amène alors le lecteur à se poser des questions. Ainsi, dans les deux premières lignes de l'énoncé, après réflexion, on se rend compte qu'une " suite non vide "de caractères est en fait "une suite d'un ou plusieurs caractères".

Un exemple de repentir se trouve dans les deux dernières lignes où l'on définit ce qui est la sortie du programme, tout en signalant plus loin la possibilité qu'il n'y ait pas de texte de sortie; cette caractéristique ayant dû apparaître beaucoup plus tôt car elle constitue un attribut essentiel de le définition de la sortie.

3°) La surspécification.

Il s'agit d'éléments de la spécification qui sont des caractéristiques d'une solution possible plutôt que des caractéristiques du problème.Typiquement,il s'agit d'erreurs reflétant une confusion entre le "quoi" et le"comment".

Exemple.

La spécification de la "Daisy Chain" donnée par (JAC,75) dit:"...La carte contient trois champs-FI,F2 et F3. FI doit être numérique et compris entre I et 99; si c'est le cas, il peut être utilisé pour retrouver une entrée à partir de la table. L'entrée contient deux valeurs limites et un multiplicateur. Si F2 est à l'intérieur des limites, il peut être multiplié par le multiplicateur pour donner l'adresse sur le disque. L'article à l'adresse du disque contient un string de caractères. F3 doit être égal à un sous-string de ce string de caractères. Si c'est le cas, ce préfixe dans le string doit être imprimé... Il y a 3 processus qui prennent en charge respectivement FI,F2 et F3:

PI retrouve l'entrée dans la table,

P2 accède à l'article sur le disque,

P3 imprime le préfixe de F3 dans le string des caractères ..."

La surspécification provient ici du fait que l'on ne se contente pas de définir le problème, mais, en outre, on dit comment le résoudre en préconisant un enchaînement des tâches beaucoup trop algorithmiques pour ce niveau, et de plus, en introduisant les notions de processus nécessaires à la réalisation du système.

4°) La contradiction.

Il s 'agit d'éléments de la spécification qui définissent de façon incompatible une même caractéristique du problème.

Exemples.

Toujours dans la spécification de Goodenough et de Gerhart (GOO,76),on trouve deux définitions distinctes de la donnée d'entrée. En effet,d'une part, elle est définie comme "un flot de caractères" et d'autre part,comme "une suite de mots et de sauts".

On passe ainsi de la notion d'une suite de caractères à celle d'une suite de suite de caractères.

Un autre exemple est celui de la fonction "mise-à-jour-plus-du-stock"

dans la spécification du cas Petitpas (BOD,79b).

"...le message d'entrée est Produit-réapprovisionné..." et la règle 2 dit "...pour chaque produit pour lequel on vient d'enregistrer une livraison..." On est amené ici à se demander si le message d'entrée concerne bien un produit réapprovisionné ou plusieurs.

5°) L'ambiguïté.

Il s'agit d'éléments de la spécification qui autorisent deux interprétations ou plus.

Exemple.

Dans la spécification du cas Petitpas (BOD,79b), on trouve comme première règle pour la fonction de "vérification pour l'identité du client" : "...tout client est identifié soit par son numéro de client, soit par ses nom et prénom ...". Cette spécification aurait été ambiguë, car on ne sait quelle priorité donner si on possède à la fois les deux identifiants,si elle n'avait été complétée plus loin par la troisième règle disant: "... les nom,prénom, adresse inscrits par le client sur le bon de commande ont toujours priorité sur le numéro inscrit par le client sur son bon de commande lors d'un essai d'identification...".

De manière générale, l'emploi des opérateurs logiques tels que "et","ou", "soit ...,soit ..." entraîne fréquemment une certaine ambiguïté.

6°) La référence en avant.

Il s'agit d'éléments de la spécification utilisant des caractéristiques du problème définies plus loin dans le texte.

Exemple.

Dans la spécification donnée par Balzer,Goldman et Wile (BAL,77) d'un système automatique de messages dans des bureaux, on trouve dans le premier paragraphe: "Les messages reçus du "AUTODIN-ASC" sont traités pour être assignés par une distribution automatique". Et c'est seulement plus loin, à partir du sixième paragraphe que l'on définira la technique

d'assignation. De manière générale, ce sont ces références qui empêcheront une lecture séquentielle de la spécification.

A côté de ces différents défauts qui caractérisent souvent une spécification, celle-ci peut être également jugée sur un ensemble de critères représentatifs d'une bonne définition du problème. La liste de ceux-ci n'est pas exhaustive mais nous avons choisi de retenir ceux que l'on retrouve le plus souvent chez la plupart des auteurs tels que (PAR,72), (LIS,75a),(DEM,79),(FIN,79).

La première qualité d'une spécification est qu'on n'y trouve pas trace des six "péchés capitaux" qui viennent d'être définis. En particulier, on veillera à ce qu'il n'y ait pas de contradictions, de silences ni de surspécifications, ce qui assurera respectivement la cohérence, la complétude et la minimalité de la spécification.

Cohérence (consistance).

Un énoncé sera cohérent s'il ne contient pas de contradictions au niveau des définitions des objets manipulés.

Il faut remarquer que fréquemment un énoncé informel est incohérent, notamment lorsqu'il comporte de nombreuses définitions conditionnelles (comment s'assurer que plusieurs cas identiques de combinaison de conditions n'ont pas entraîné des traitements différents ?) .

La complétude.

Une spécification doit décrire complètement le problème, c'est-à-dire définir précisément toutes les relations possibles et nécessaires entre les résultats et les données du départ.

La minimalité.

Il s'agit de construire des spécifications qui présentent les propriétés intéressantes des concepts rencontrés et rien d'autre. En particulier,on décrira ce que doivent faire les programmes sans dire comment ils seront réalisés. Si l'énoncé était surspécifié,cela restreindrait les possibilités

de résolution tout en augmentant la complexité de la vérification du programme en accroissant le nombre de propriétés à prouver.

D'autres qualités pouvant caractériser une bonne spécification seront:

la constructibilité.

Il doit être possible de construire les spécifications d'un problème sans grande difficulté. Pour cela, il s'agit d'avoir une méthode aidant le spécificateur dans sa tâche. S'il existe un langage particulier de description, il faut que celui-ci soit facilement assimilable, ne conduise pas à un surcroît de travail fastidieux et puisse s'appliquer à la description d'une large classe de concepts.

la fidélité.

Il s'agit d'une qualité non formalisable car elle exprime l'adéquation de l'énoncé initial au problème donné. Comme le souligne (FIN,79), on ne pourra se convaincre qu'intuitivement de cette adéquation et la formalisation de l'énoncé sera d'autant plus convaincante qu'elle sera proche de l'énoncé informel. Cela constitue une raison supplémentaire de non-surspécification de l'énoncé.

la compréhensibilité.

L'énoncé devra être compréhensible pour l'informaticien ainsi que pour la personne qui pose le problème. La spécification devra être claire et pas de trop grande taille pour rester facile à comprendre. A la limite, comme le signale (LIS,75), il pourrait être souhaitable d'avoir un énoncé de dimension plus petite que le programme qu'il spécifie.

l'extensibilité.

Il est désirable qu'un changement minimum dans le problème résulte en un changement minimum dans la spécification.

Remarque : de la formalisation des spécifications.

Pour plusieurs auteurs, la seule manière d'échapper à beaucoup de défauts d'une spécification est de l'exprimer à l'aide d'un langage formel qui

permettra d'expliciter les concepts développés d'une manière précise et non-ambiguë. En particulier, la formalisation s'avère nécessaire lorsqu'on veut utiliser les spécifications en conjonction avec des outils de preuve de programme, de vérification d'équivalence de spécification,etc...
Par contre, comme le souligne (BAL,77) , une description informelle présente, aussi, bien des avantages: la concision (car seulement une partie de la spécification est explicite), la communicabilité, une maintenance plus aisée (moins de complexité que la spécification formelle). En outre, la non-ambiguïté peut être également assurée si la spécification informelle est faite avec suffisamment de rigueur et de précision.
Pour concilier les représentations informelles mieux adaptées aux demandeurs et les représentations formelles plus recherchées par les programmeurs, il faudrait la possibilité de techniques capables de transformer l'une en l'autre grâce à la connaissance du contexte.

A côté de tous ces critères caractérisant une "bonne" spécification, les différents formalismes employés peuvent encore être comparés au point de vue:
- des environnements qu'il offrent, en particulier,concernant des fins de documentation automatique ou de contrôles élémentaires de cohérence;
- des supports graphiques possibles;
- de la compétence requise de la part des personnes qui spécifient (non-informaticiens ou informaticiens);
- de la généralité du domaine d'application de la technique de spécification.

Enfin, dans la description des problèmes de taille importante (spécification de système d'information), il est fondamental de disposer d'une démarche méthodologique qui nous guidera dans la rédaction des spécifications. C'est sur l'importance d'une telle démarche que nous insisterons dans le paragraphe suivant.

I.6. Quelques méthodologies de spécifications et quelques considérations sur les critères de décomposition d'un système d'information.

On peut constater que la spécification de problèmes de taille relativement modeste, tel que la gestion d'une pile, a été particulièrement réussi par plusieurs spécificateurs et présente les qualités requises. Cependant, si pour un problème bien délimité, l'application directe de tel ou tel langage de spécification est immédiate et ne pose pas de difficultés, par contre, face à des problèmes de dimensions importantes tels que la description de systèmes d'information, le langage de spécification ne suffit plus et il faut se définir une démarche méthodologique permettant de structurer l'approche du problème et par conséquent sa spécification. En effet, spécifier un problème, ce n'est pas seulement l'expliciter, mais c'est encore le structurer.

Dans cette partie, nous allons passer en revue quelques moyens récents d'aide à la spécification et nous jugerons chacun d'entre eux sur :

- la qualité de la spécification obtenue,
- la démarche méthodologique suivie.

On peut classifier les systèmes existants, comme le fait (GER,80), en deux catégories :

-les systèmes passifs. Il s'agit d'un ensemble d'outils intégrés autour d'une base de données et gérés par un système exécutif.

- les systèmes actifs. Ils reposent sur l'utilisation d'une base de connaissances permettant, par un dialogue interactif avec le demandeur, d'aboutir à une spécification formelle tout en étant parti d'une définition informelle. Un tel scénario présuppose une certaine intelligence du système au niveau des questions posées à l'utilisateur.

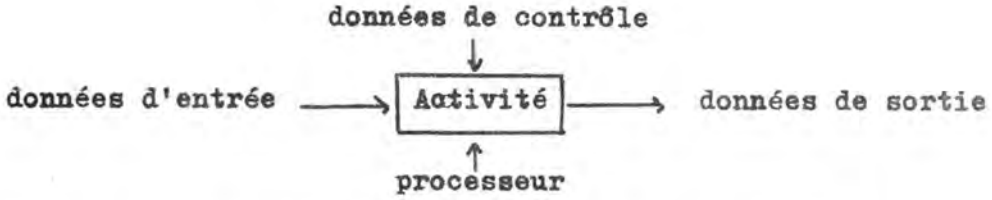
Les systèmes du second type sont apparemment plus sophistiqués, ce qui explique que leur réalisation complète ne sera pas assurée avant plusieurs années. Quant à ceux du premier type, il s'agit de méthodes que l'on

trouve depuis quelques années sur le marché et pour lesquels on dispose de données concernant les expériences connues par les utilisateurs lors de leur emploi. Parmi les systèmes dits "passifs", nous décrirons celui qui est commercialisé par Softech appelé S.A.D.T. et celui étudié par F. Bodart et son équipe à l'Institut d'Informatique de Namur. Comme exemple de système actif, nous prendrons le système français Zaide.

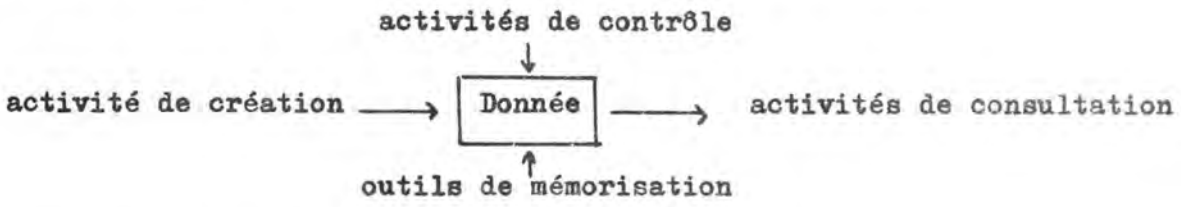
I.6.I. S.A.D.T. (Structured Analysis and Design Technique)(ROS,77),(SOF,76).

Cette méthodologie prend en compte le modèle fonctionnel des traitements que le système doit réaliser aussi bien que le modèle organique concernant l'implémentation de ces fonctions. Dans l'analyse qui suit, nous nous intéresserons uniquement à la vue fonctionnelle.
Comme nous l'avons vu dans le chapitre I.4., SADT repose sur une décomposition structurée d'éléments complexes en ses parties constituantes. Au départ, on considère le problème comme un seul "module" représenté par une boîte. Celle-ci pourra être décomposée en un certain nombre de boîtes plus détaillées (au moins trois, pas plus de six), chacune d'entre elles représentant une fonction contenue dans la boîte initiale. Ensuite, chacune des boîtes introduites est à son tour décomposée pour mieux décrire l'information impliquée dans les boîtes jusque là décrites. De plus,pour chaque "module", on définit ses relations avec d'autres "modules" au moyen de flèches les interconnectant. Une démarche "top-down" ainsi appliquée permet de ne pas devoir appréhender un trop grand nombre de détails trop vite.
Il est important de signaler la dualité qui existe, à chaque niveau de la décomposition, entre les diagrammes de traitements appelés "actigrammes" et les diagrammes de données appelés "datagrammes".

Chaque boîte de traitement est décrite de la manière suivante :



Chaque boîte de donnée est décrite de la manière suivante :



Les étapes de l'analyse fonctionnelle seront les suivantes :

. établir les diagrammes des activités et des données caractérisant le système,

. vérifier la dualité données/traitements,

. compléter éventuellement les activités ou données manquantes,

. analyser les séquences dans lesquelles l'activité peut survenir,

.identifier les mécanismes qui permettront d'implémenter les fonctions et ce qui servira de point de départ pour passer à l'analyse organique.

Par rapport aux critères énoncés dans le chapitre I.5. caractérisant une bonne spécification, on peut dire que le langage utilisé par SADT présente les originalités suivantes :

- la constructibilité. Il existe une méthode permettant une décomposition descendante basée sur une structure hiérarchique de fonctions ou d'objets.
- la consistance. Le contrôle de la cohérence de la description repose sur celui des interfaces à chaque niveau de décomposition. Dans les variantes du système telles que celles développées par I.T.T. (BAS,78),il existe des possibilités de vérifications automatiques.
- la compréhensibilité. La représentation graphique très claire et facilement communicable constitue le point fort de la méthode.
-l'aspect formel. Au niveau de l'expression de l'organisation entre les différents modules, on a la possibilité d'une description formelle grâce

aux possibilités graphiques qui présentent quarante enchaînements possibles. Malheureusement, ce formalisme n'intervient que pour décrire la structure du système et les relations entre les éléments constitutifs (traitements ou données). Au niveau du rôle et de la signification de ces éléments, on doit se contenter de l'exprimer à l'aide d'un texte rédigé dans un langage naturel.

-l'extensibilité. Du fait de la décomposition arborescente et de la description des interfaces, il doit être possible de pouvoir localiser et voir les conséquences d'un changement.

Au niveau de la démarche qui va nous permettre de construire un système structuré de spécifications qui reflète bien la réalité du problème, la difficulté est de savoir exactement ce que c'est qu'une boîte (traitement ou donnée). A ce sujet, les auteurs donnent relativement peu de renseignements précis. Tout au plus, signalent-ils qu'il s'agit de la phase la plus critique et la plus coûteuse en temps et qu'il faudra interviewer de nombreux experts (personnes qui ont la compréhension de la réalité), cette démarche étant de plus itérative.

Une telle approche signifie que le spécificateur ne disposera a priori d'aucun moyen pour identifier les différentes fonctions et, d'autre part, comme il s'adressera à plusieurs personnes différentes, il se peut que des fonctions qui lui paraissent distinctes soient en fait les mêmes. N'y aurait-il pas intérêt,dès le départ, à établir un consensus entre les différents utilisateurs afin d'éviter des retours en arrière éventuels ?

Un autre point intéressant est de savoir à quel moment arrêter la décomposition des boîtes. Il n'est rien dit à ce propos, excepté que cela doit convenir au demandeur, ce qui est dangereux car celui-ci connaît très bien le contexte du problème et ne jugera peut-être pas utile de décomposer certaines fonctions qui lui paraissent suffisamment explicites;

en conséquence, par la suite, les fonctions obtenues seront-elles également compréhensibles pour les informaticiens ?

En conclusion, nous pouvons dire qu'un tel système présente l'avantage de manipuler des concepts apparemment simples mais, malheureusement, dont l'utilisation paraît moins évidente.

I.6.2. PSL/PSA et l'approche de Namur (TEI,77), (BOD,8I).

Cette approche est celle proposée aux étudiants en informatique des F.N.D.P. de Namur. Elle repose sur une série de travaux réalisés par F. Bodart, P. Chen et D. Teichroew. Nous mettrons ici l'accent sur la partie concernant l'élaboration des spécifications fonctionnelles.

Cette démarche, comme nous l'avons déjà mentionné dans le chapitre I.4., repose sur l'élaboration

- d'une spécification du modèle des données,
- d'une spécification du modèle des traitements.

La représentation du modèle conceptuel des données sera exprimé à l'aide du modèle entité-association. On en trouvera une définition dans (BEN,76) et (BOD,8I).

La modélisation des traitements se fera en trois étapes :

- décomposition des traitements en fonction d'une nomenclature hiérarchique donnée (Système, Sous-système, Application, Phase, Fonction). Une explicitation de ces différents composants se trouve dans (BOD,8I).
- description statique des traitements.
  Pour chaque fonction, on décrit les inputs (messages en entrée), les outputs (messages en sortie), les collections d'information manipulées et le traitement proprement dit à l'aide de règles exprimées en langage naturel.
- description dynamique des traitements.
  Le système est modélisé comme un ensemble de processus (une

occurence de processus est créée à chaque exécution d'un traitement correspondant à une fonction); ces processus interagissent par des messages. Tout événement (correspondant à la production d'un message) modifie l'état du système. La description dynamique est assurée par la spécification des enchaînements des processus et des synchronisations entre les processus (via les points de synchronisation).

La démarche méthodique proposée est la suivante :

I. Décomposition fonctionnelle des traitements.

2. Pour chaque phase,

- spécification du sous-schéma conceptuel et constitution du dictionnaire de données,
- définition des messages d'entrée-sortie.

3. Intégration des sous-schémas en un schéma conceptuel et complètage du dictionnaire de données.

4. Pour chaque niveau de la structure des traitements, description de la statique des traitements.

5. Description de la dynamique des traitements.

Ces différentes spécifications seront exprimées à l'aide du langage PSL développé à l'université de Michigan dans le cadre du projet Isdos (TEI,77) et fondé sur les concepts de types d'objets, de relation entre types d'objets et de propriété associée à un type d'objet ou à une relation.

Confrontons cette approche avec les qualités énumérées ci-dessus.

- constructibilité. Au niveau de la spécification au sens large ("spécification in the large"), la description des traitements et des données peut être facilitée par la modélisation du système d'information qui est proposée (sous-système, application, phase). Par contre, au niveau des critères permettant d'identifier une fonction ainsi qu'au niveau de la description d'une fonction ("specification in the small"), il n'existe pas de guide méthodologique véritable.

- cohérence. Certains contrôles de cohérence de type syntaxique sont effectués : correspondance entre les objets déclarés, vérification de la définition d'un objet avant son utilisation, ...De plus, la cohérence des règles au niveau de la spécification statique d'une fonction peut être contrôlée manuellement à l'aide d'une table de décision. Par contre, rien n'assure la cohérence entre les règles se trouvant dans différentes fonctions.
- compréhensibilité. Au niveau de la structure globale du système d'information, le nombre de concepts proposés par le modèle n'est pas très élevé, il est dès lors facile de communiquer aisément avec le spécificateur Par contre, au niveau local de la spécification d'une fonction (en particulier, si celle-ci a été mal définie), le nombre et la complexité des règles peut entraîner une mauvaise compréhension de l'effet du traitement.
- fidélité. Il est envisagé actuellement de développer un générateur de maquette de système permettant à partir de la spécification d'évaluer le caractère effectif des règles de traitement.
- extensibilité. Du fait de la décomposition hiérarchique en système, sous-système, application, phase basée sur des critères précis et rigoureux ainsi que de la spécification non-ambiguë des interfaces, il est possible de localiser facilement les modifications. Il est à remarquer que cela serait moins le cas pour les fonctions où là, les critères d'identification sont moins précis (sémantique simple, objectif élémentaire, résultat constituant un message significatif).

Remarque au sujet de l'aspect formel de la spécification.

Au niveau des interfaces et des communications entre les différents objets introduits, la syntaxe est formalisée ce qui permet un certain nombre de contrôles automatiques et d'outils permettant de tester la faisabilité des spécifications (simulateur). Par contre, au niveau de l'expression du contenu des traitements, le principe des règles exprimées en langue naturelle, s'il n'est pas appliqué avec précision, rigueur et minimalité peut conduire

à une description fastidieuse et ambiguë.

Dans l'élaboration des spécifications, la méthode de Namur présente un guide méthodologique dans la décomposition des traitements au niveau du sous-système, de l'application et de la phase. C'est en effet à partir de celle-ci qu'on élaborera la structure conceptuelle des données et qu'on décrira les différentes fonctions utilisées. La phase est définie dans le contexte d'une cellule d'activité, centre d'activité homogène dans l'espace et dans le temps, doté de ressources et pourvu de règles de comportement nécessaires à son fonctionnement. Le critère d'identification d'une application repose quant à lui sur le principe de quasi-décomposabilité introduit par (SIM,74). Le sous-système ,lui, regroupe des applications non-indépendantes dans le cadre des interactions d'exécution (flux physiques) et de gestion (flux de décisions). Ces notions sont fondamentales car elles assurent la liaison entre le système informatique, le système d'allocation de ressources et le système d'organisation.

I.6.3. Le Système Zaide (ABR,78), (DEM,79), (DEM,80).

Le langage Z est un formalisme utilisé pour la spécification, la conception et l'écriture de programmes. Plus précisément, il existe deux niveaux de langage : ZO "orienté spécification" et ZI "orienté programmation, ainsi qu'un catalogue de règles de transformations permettant le passage de ZO à ZI. C'est sur l'aspect spécification, qui a d'ailleurs été le plus développé jusqu'à maintenant, que nous allons porter notre attention.
La spécification obtenue est caractérisée par :

- une description totalement indépendante d'une implémentation éventuelle;
- par un développement par raffinements successifs;
- par l'aspect statique (on se contente de décrire les relations existant entre les données et les résultats);
- par l'utilisation d'un formalisme mathématique basé sur les

concepts et les notations de la théorie des ensembles.

La structure d'une spécification en Z est la suivante :

- à partir du cahier de charges, formalisation du document par l'écriture de "clauses" Z;
- ces clauses peuvent être de trois sortes :
  + les clauses de "type" introduisant différentes sortes d'objets.
  + les clauses de "relation" qui définissent des fonctions entre les types définis (syntaxe du système).
  + les clauses d'"assertion" qui expriment les lois logiques du du système (sémantique du système).
- quand on a fini d'exploiter le cahier des charges, on dispose d'une première version de la spécification. Celle-ci sera améliorée par un ensemble de transformations parmi lesquelles on trouve :
  + groupement de relations en une relation dont le domaine est le produit cartésien des domaines initiaux ("merge").
  + l'élimination éventuelle de la récursivité.
  + le choix d'une méthode d'accès particulière pour les éléments d'un certain type.
  + etc...

  Après chaque transformation, on obtient une nouvelle version de la spécification formelle.

La démarche proposée pour aboutir à une telle spécification est la suivante:

- dans une première étape, lecture des différents éléments du cahier de charges, ce qui permet d'aboutir à l'identification des types de base du problème, de quelques relations et de quelques assertions.
- dans une deuxième étape, relecture du cahier des charges qui permettra d'obtenir presque tous les types et quelques nouvelles relations et assertions.

- nouvelles lectures permettant d'affiner les types d'objets, les relations et les assertions.

L'arrêt de ce processus itératif correspond, du moins s'il converge, à l'obtention d'une spécification complète du problème. De plus, au cours de cette démarche, des questions peuvent se poser à chacune des étapes et doivent recevoir une réponse au moment où elles sont apparues afin de s'assurer une compréhension complète du problème.

Un premier prototype du système Zaide d'aide à la construction de spécifications écrites en langage Z existe sous forme interactive; il permet de réaliser les fonctions d'analyse, de vérification, d'archivage, d'édition et de visualisation des spécifications.

Au niveau de la qualité des spécifications obtenues, il faut d'abord mettre l'accent sur le fait que la démarche nous a permis de décrire des types abstraits de données (LIS,75a), (GUT,77). Ceux-ci représentent des outils d'écriture de grands logiciels (DER,79) caractérisés par :

- une description d'un objet reposant sur la séparation entre, d'une part, les propriétés intrinsèques de cet objet et les opérations abstraites définies dessus, et d'autre part, les détails de représentation de l'objet. Cette insensibilité vis-à-vis de représentations particulières permet une amélioration sensible lors de la maintenance des programmes.
- le cadre des types abstraits permet, en principe, de s'assurer plus facilement que la description d'une structure de données est cohérente, complète et minimale (voir (GUT,78) pour de telles techniques de vérification).
- les types abstraits se prêtent bien à une utilisation dans n'importe quel genre de problème.
- enfin, ils constituent des aides au processus de construction d'un programme correct en introduisant différents niveaux de décisions de design (explicitation en un algorithme en introdui-

sant de nouvelles fonctions de base et transformation de l'algorithme abstrait en un algorithme concret).

- pour un certain nombre de langages, l'utilisation de types abstraits est essentielle (voir, par exemple, CLU (LIS,75b), ADA (BAR,80), ALPHARD (WUL,75)).

Confrontons le langage Z aux critères qualitatifs repris ci-dessus :

- constructibilité. Une fois bien assimilés les concepts du langage, ce qui n'est peut-être pas évident pour une personne habituée à un tel formalisme, il est possible de construire la spécification d'une manière répétitive et en se posant toujours le même genre de questions pour chacune des clauses définies.
- cohérence et complétude. Des contrôles tels que la détection d'incohérence entre types, des références manquantes et des erreurs syntaxiques sont possibles et automatisés dans Zaide.
- compréhensibilité. Dans le formalisme tel qu'il est, les spécifications sont communicables à très peu de personnes. Une possibilité envisagée est d'introduire un "médiateur" entre le spécificateur et les demandeurs qui assurerait la communicabilité.
- aspect formel de la spécification. Z repose sur une mixture de notations mathématiques de type ALGOL et LISP et relationnelles à la Codd; les propriétés de telles notations assurent une définition rigoureuse tant sur le plan de la syntaxe que sur celui de la sémantique.
- minimalité. L'utilisation de notions proches des types abstraits assure une description assez indépendante de n'importe quelle idée de réalisation.
- extensibilité. Un changement dans les spécifications est facilement localisable en fonction de l'objet sur lequel il porte.

Au niveau de la méthode employée, il est difficile de porter sur elle un jugement valable; en effet, Z a été uniquement employé à la spécification de systèmes informatiques de gestion sur une échelle relativement modeste. En particulier, si cette méthode permet de structurer les objets rencontrés

en présentant différents types, il serait également intéressant de fournir une vue structurée des traitements à effectuer (cfr. dualité traitements/ données chez SADT) qui est loin d'être obtenue par la démarche actuelle consistant à écrire les spécifications directement à partir du cahier de charges en le suivant ligne par ligne. En outre, dans la description de systèmes d'information relatifs à des organisations, on peut se demander comment rendre compatibles les fonctions avec leur environnement organisationnel, comment décrire les interfaces entre le monde extérieur et le système informatique, ... ?

Chapitre 2.

PRINCIPES DE LA DEMARCHE.

Dans le premier chapitre, après avoir suggéré certaines qualités d'une spécification, nous avons étudié quelques méthodes existantes. Si la plupart de celles-ci présentent un effort au niveau du langage de description employé, seule l'une d'entre elles (l'approche de Namur) présente un certain guide méthodologique permettant au spécificateur d'appréhender le problème posé, du moins, au niveau de la structure globale du système d'information. En effet, au niveau de chacune des fonctions, nous avons souligné le manque de rigueur au niveau de sa spécification.
Dans ce chapitre, nous allons présenter une démarche alternative s'attachant d'abord à une description logique du système sous étude par raffinements successifs, avant de prendre en compte les contraintes de l'environnement organisationnel. De plus, afin de juger le caractère effectif de la méthode proposée, nous passerons à la réalisation et à l'implémentation de la partie informatique du système ainsi spécifié.

Ce deuxième chapitre se composera de cinq paragraphes.

Dans le premier paragraphe, nous verrons comment utiliser la "méthode déductive", développée au CRIN de Nancy, comme outil de spécification permettant d'obtenir une vue logique des traitements et données nécessaires, abstraction faite des contraintes organisationnelles.

Le deuxième paragraphe montrera comment compléter la première spécification en prenant en compte les différentes contraintes liées à l'environnement organisationnel dans lequel le système se trouve.

Le troisième paragraphe sera consacré à l'expression des performances globales du système attendues par le demandeur.

Dans le quatrième paragraphe, nous traiterons de la réalisation de la partie informatique du système en définissant une architecture de programmes pour les traitements et un premier modèle d'accès pour les données.

Le cinquième paragraphe présentera la manière dont les programmes seront réalisés ainsi que le modèle des accès effectivement nécessaires pour les données.

Enfin, le sixième paragraphe montrera comment implémenter les programmes et la structure de données.

2.I. Spécification descendante d'un système d'information.

2.I.I. La construction de programmes par méthode déductive.

Depuis la fin des années 60, des auteurs, de plus en plus nombreux, se sont préoccupés des problèmes de la programmation. Jusqu'à cette époque, l'empirisme qui était de règle dans la résolution de ces problèmes avait eu comme résultats des programmes non conformes aux spécifications, contenant de nombreuses erreurs, mal documentés et illisibles. Cette anarchie intolérable a conduit petit à petit à l'avènement d'un certain nombre de méthodes de programmation, (DAH,72), (WIR,73), (JAC,75), (DIJ,76), visant la construction de "bons programmes". La réflexion menée à Nancy, depuis I973, par C. Pair et son équipe (PAI,79) n'est certainement pas indépendante de ces travaux regroupés fréquemment sous le vocable de "programmation structurée". L'approche de Nancy, baptisée "programmation déductive", est basée sur la décomposition de l'activité de construction d'un programme en un certain nombre d'étapes permettant de réduire ainsi le nombre de questions à se poser à tout moment.

a) la définition du problème.

Cette première étape consiste à définir le plus précisément possible le problème à traiter. En effet, de manière générale, un problème n'est jamais tout-à-fait défini et il s'agit d'en donner une explicitation claire, correcte et complète que l'on appellera l'énoncé.

b) la construction du programme.

Cette seconde étape sera assurée par le choix de structures de données adéquates et par le passage d'un énoncé non-procédural à une forme procédurale par des compositions itératives, conditionnelles et séquentielles d'actions de base.

c) le codage du programme.

Il s'agit enfin d'exprimer l'algorithme dans la forme requise par le langage de programmation retenu.

Dans le cadre de notre travail, nous allons nous intéresser plus spécialement à la construction de l'énoncé, c'est-à-dire à la première spécification du problème.

Cette analyse se fera d'une manière déductive en définissant, d'abord, les données "résultats" puis en caractérisant celles-ci en fonction de données "intermédiaires" considérées comme résultats de sous-problèmes et en appliquant itérativement cette démarche jusqu'à ce qu'on n'ait plus comme données "intermédiaires" que des données "de départ" du problème.

Les idées essentielles sous-jacentes à cette démarche sont les suivantes (BEL,78), (FIN,79) :

A. On part de la définition des données "résultats". Dans toute proposition de problème, elles sont, en effet, généralement décrites assez précisément, on peut donc partir de cette base comme spécification initiale.

B. Il est rarement possible d'exprimer directement les résultats à partir des données "de départ" (que nous appelerons dorénavant "arguments"), on est donc conduit à introduire et à définir des données "intermédiaires". Cette approche permet de se poser les sous-problèmes (liés à la définition des intermédiaires) aux bons moments. D'autre part, elle permet de dresser petit à petit un inventaire de toutes les données qui sont amenées à jouer un rôle dans la spécification du problème.

C. Pour chacune des données intermédiaires introduites, on réitère cette démarche jusqu'à ne plus avoir comme données que les arguments du problème initial. Cette méthode est structurée car elle décompose à tout instant le problème en sous-problèmes correspondant chacun à un des intermédiaires introduits.

D. La description du problème et des sous-problèmes sera à tout moment formelle et informelle. Ces définitions sont ainsi compatibles avec une spécification de bonne qualité (cfr. la discussion sur le caractère formel ou informel d'une spécification dans le paragraphe cinq du premier chapitre).

Remarque.

Lorsqu'on applique une méthode dite "déductive", il ne s'agit certainement pas d'un raisonnement purement déductif (qui, en particulier, ne tiendrait pas compte des arguments du problème initial) mais plutôt d'une indication de la direction dans laquelle le raisonnement doit avoir lieu (PAI,79).

Cette méthode est donc caractérisée par :

- une démarche induite par l'objectif à réaliser, c'est-à-dire allant des résultats vers les arguments. En effet, en général, on connait mieux les résultats que l'on cherche à obtenir que les données "de départ" (DIJ,76).
- un ordre d'étude totalement indépendant de l'ordre d'exécution (cfr. les langages fonctionnels (HEN,80a).
- une méthode induite par certains langages de très haut niveau voisins de cette approche (cfr. exemple de BDL (HAM,77)).

On trouvera, à titre d'exemple, la spécification d'un problème à la figure 5. Il est à remarquer que cette description n'est en rien algorithmique car elle se contente de présenter une définition pour chacune des données rencontrées sans définir un ordre de prise en compte de ces définitions.

s : salaire net d'un ouvrier	s = sb - rss
sb : salaire brut	sb = si nh > 40
rss : retenue sécurité sociale	alors sbs40
sbs40 : salaire brut pour plus de 40 heures de travail	sinon sbm40
	rss = sb x 0,06
sbm40 : salaire brut pour 40 heures de travail ou moins	sbs40 = nh x shs40
	sbm40 = nh x shm40
nh : nombre d'heures de travail	nh : argument
shs40 : salaire horaire pour plus de 4o heures de travail	shs40 : argument
	shm40 : argument
shm40 : salaire horaire pour 40 heures de travail ou moins	

où :

- la partie gauche de la table constitue la description informelle appelée lexique;
- la partie droite de la table constitue la description formelle;
- le résultat est s;
- les arguments sont nh, shs40, shm40;
- les intermédiaires sont sb, sbs40, sbm40, rss.

figure 5. Spécification d'un calcul de salaire.

La méthode déductive utilisée dans la construction de programmes semble donc posséder l'ensemble des qualités requises pour en faire une méthode de construction d'un énoncé de système d'information, excepté peut-être le manque de description précise des données (résultats, arguments, intermédiaires). En effet, si on a bien mis en évidence l'ensemble des données nécessaires, la plupart des propriétés de ces données ont été sous-entendues.

Exemple.

Dans la figure 5, il n'est pas dit si s, sb, sh sont des entiers, des réels, ..., d'autre part, nous ne savons pas si les valeurs prises peuvent être strictement négatives, nulles, ...

Or, par la suite, la connaissance de ces propriétés sera indispensable dans la résolution du problème, il convient donc de les mettre en évidence dans la spécification.

Nous verrons dans le paragraphe 2.I.2. en quoi la méthode de construction d'énoncé que nous venons de présenter va pouvoir nous servir, en l'adaptant, dans la spécification de systèmes d'information.

2.I.2. La construction des spécifications d'un système d'information par méthode déductive.

Comme nous l'avons vu dans l'introduction, le système d'information est composé d'un ensemble de données et de traitements. Notre but va être, à l'aide de la méthode déductive décrite précédemment, de fournir des

spécifications les "meilleures possibles" (au sens du paragraphe I.5) de chacun des composants du système d'information. Cependant, comme nous le verrons par la suite, la méthode utilisée ne nous permettra pas de donner une description complète du système et il faudra enrichir celle-ci dans une deuxième étape en prenant davantage en compte les contraintes dictées par l'environnement organisationnel.

§I. Présentation générale de la méthode suivie.

Si la construction systématique et méthodique de programme à partir d'un énoncé est une tâche abordée depuis plusieurs années déjà, il n'en va pas de même pour la construction du premier énoncé qui reste un travail relativement empirique, comme nous l'avons vu lors de l'étude de quelques méthodes de spécification (cfr. paragraphe I.6.). Comme l'indique (FIN,79), le problème de la spécification est en fait le suivant :

" Etant donné un problème énoncé de manière imprécise, en donner une spécification précise à partir de laquelle on puisse en dériver une solution".

Dans le cadre particulier de la spécification de systèmes d'information, " le problème énoncé de manière imprécise " sera le document résultant d'une analyse d'opportunité et donnant le cadre général d'une solution. A partir de là, il conviendra de donner une description rigoureuse et détaillée du système d'information à réaliser. Celle-ci devra prendre en compte aussi bien les traitements que les données et devra satisfaire aux critères d'une " bonne " spécification. Le problème est de nous donner une démarche

- permettant d'aborder la description du problème d'une manière progressive, descendante et structurée,
- présentant une rigueur telle que nous soyons amenés à nous poser les questions aux bons moments,
- prenant en compte aussi bien la description de la structure de traitement que celle des données,

. nous permettant d'écrire à tout moment de "bonnes" spécifications (au sens du chapitre I.5.).

Sur base de ces constatations, nous aboutissons à une méthode de spécification basée en partie sur le "méta-algorithme de spécification" de (FIN,79) qui consistera à décrire le système d'information à l'aide d'un ensemble de pavés.

I. Concepts et définitions.

Les pavés seront de deux types : ceux décrivant la structure de traitements et ceux décrivant la structure de données.

- Les pavés relatifs à la spécification des traitements seront structurés sous forme d'une arborescence que l'on appelera "l'arbre déductif".

Au sommet de cet arbre, on trouvera le pavé "principal" qui décrira le rapport qui existe entre les arguments (données de "départ") et les résultats (données "résultats") à l'aide de fonctions et éventuellement en introduisant des données intermédiaires. Ces fonctions (appelées ainsi par abus de langage car il s'agit plutôt de relations au sens mathématique du terme) auront comme domaine des données qui pourront être des arguments ou des intermédiaires, et comme codomaines, des données qui pourront être des intermédiaires ou des résultats.

Les différents noeuds de l'arbre déductif seront constitués par des pavés développant les fonctions introduites dans les pavés du niveau supérieur et qui correspondent à différents sous-problèmes dont les résultats et les arguments sont respectivement les codomaines et domaines des fonctions explicitées . D'une manière itérative, ces nouveaux sous-problèmes seront décrits par l'introduction de nouvelles fonctions et données intermédiaires.

Les feuilles de l'arborescence déductive de description des traitements seront constituées par des fonctions correspondants à des sous-problèmes pour lesquels les relations entre résultats, arguments et éventuels intermédiaires introduits pourront s'exprimer à l'aide de fonctions terminales,

c'est-à-dire de fonctions que nous ne trouvons plus nécessaire et pertinent d'expliciter dans un nouveau pavé de traitement.

Par raffinements successifs, on est donc amené progressivement à introduire de nouveaux pavés pour chacune des fonctions non-terminales à remplir; on arrive ainsi à organiser la spécification des traitements en une série de niveaux distincts et ordonnés en introduisant une hiérarchie de type "utilise" entre les différents pavés (nous dirons que le pavé A "utilise" le pavé B si le fonctionnement correct de A dépend de la disponibilité d'une implémentation correcte de B, (PAR,74)).

- D'autre part, parallèlement à la description de la structure des traitements, on spécificiera des pavés de description des données (arguments et résultats du problème de départ, intermédiaires); cette spécification inclura la définition des fonctions terminales qui y sont associées.

2. Représentation.

La représentation d'un pavé est constituée de trois parties :

- le lexique qui décrit informellement les différents identificateurs (données ou fonctions).
- le profil qui définit le type des données manipulées dans le pavé ainsi que les domaines et codomaines de chaque fonction associée;

le type d'une donnée est l'ensemble des données caractérisées par les même propriétés.

Les types du domaine et codomaine d'une fonction sont respectivement les types des arguments et des résultats de cette fonction.

- la description formelle qui caractérise

+ pour les pavés de traitement: les arguments, les résultats, les éventuels intermédiaires et les fonctions portant sur ces différents objets.

+ pour les pavés de données: les différents types de données, les invariants qui les caractérisent et les fonctions de base portant sur eux.

Nous reviendrons dans les paragraphes qui suivent sur une description un

peu plus précise et un peu plus riche des pavés de traitements et de données, mais dès à présent, nous pouvons relever quelques caractéristiques inhérentes à cette démarche (FIN,79).

- Structurations conjointes des pavés de traitements et de données.
- Présence d'un guide méthodologique (le profil) qui, à tout instant, nous signale les intermédiaires et les fonctions non-terminales qu'il reste à développer.
- Regroupement de toutes les fonctions terminales portant sur les types de données (notion de "types abstraits" (LIS,75a),(GUT,77)).
- Possibilité de se convaincre de l'adéquation de la spécification au problème par des justifications pas à pas.
-Possibilités de discussion avec le demandeur grâce à la présence de descriptions informelles en langue naturelle (lexique). En particulier,il est possible, à l'aide de spécifications formelles et du lexique, de revenir, en langue naturelle mais structurée, à une version du problème.

Remarque I.

Comme nous venons de le voir, le principe qui nous guidera dans la démarche de structuration des traitements sera l'approche déductive à partir des résultats. Cette approche est sensiblement différente de celles étudiées dans le premier chapitre (cf.par. I.6.) et assez similaire à l'approche induite par BDL (HAM,77). En effet, nous nous éloignons d'une démarche en terme de "flow control" (flux de contrôle) basée sur une découpe selon les fonctions à effectuer successivement sur les processeurs et les ressources disponibles, discutées dans le paragraphe I.6., pour nous orienter vers un modèle "data flow" (flux des données), basée sur une découpe des traitements dont le description est non-procédurale et où tout séquencement non dicté par la dépendance entre les données du problème a été éliminé. Certains argumentent que cette optique se rapproche davantage du penchant des utilisateurs qui préfèrent exprimer leurs procédures en termes de flux de données plutôt que de flux de contrôle (MIL,74).

Ainsi l'exemple donné par (GOL,75) et inspiré par l'analyse de Miller concernant la difficulté pour les non-programmeurs de spécifier le flux de contrôle à l'intérieur d'une fonction est significatif: le le non-programmeur préférera spécifier " mettre les choses rouges dans la boîte" plutôt que "pour chaque chose, si la chose est rouge,la mettre dans la boîte".

Cependant, il est assuré que, tôt ou tard, nous devrons tenir compte de ce flux de contrôle, ne serait-ce que pour distinguer la partie automatique de la partie manuelle du système d'information, mais à ce stade, nous préférons nous limiter uniquement à une description de la découpe des traitements dictée par le flux de données du problème et ainsi, dans un premier temps, conserver une spécification la plus indépendante possible des choix inhérents à l'environnement.

Remarque 2.

En outre, cette méthode de spécification pourrait être à la base de la construction d'un certain nombre d'outils de spécification que nous décrirons brièvement dans le quatrième paragraphe. Auparavant, nous allons présenter de manière plus détaillée les pavés de traitements et de données constituant la définition du système d'information.

§2. La structure des traitements.

Nous allons maintenant, sur base de l'exemple de la figure 6, introduire, de manière précise, les différents concepts que l'on peut trouver dans un pavé de traitement.

Remarque préalable sur le langage de description.

Tous les mots soulignés font partie du langage de base que nous nous donnons pour la description des spécifications et ne nécessitent pas d'explicitations. Dans le langage de base, nous trouverons également un certain nombre de notations mathématiques et informatiques habituelles :

- les types simples : entier, booléen, réel ... ainsi que les opérateurs arithmétiques et logiques.

- les fonctions d'un domaine E dans un codomaine F que nous indiquerons : E $\longrightarrow$ F .
- des symboles tels que celui de l'ensemble: $\{\}$

du produit cartésien: X .

a) Le titre.

Nous indiquerons le nom de la fonction décrite par le pavé (dans l'exemple, NOM-FONCTION).

b) La description formelle.

bI. Les arguments et les résultats .

On définit les données en entrée et en sortie de pavé.

On distinguera les données internes et externes.

+ les données externes d'un pavé sont en provenance (arguments externes) ou à destination (résultats externes) d'un pavé de niveau égal ou inférieur dans la hiérarchie "utilise".

Ces données externes correspondent dans l'approche de Namur (BOD,80) aux messages d'input et d'output et chez BDL (NAM,77) aux documents d'input et d'output.

+ les données internes d'un pavé correspondent à un ensemble d'informations présentant un caractère permanent dû au fait qu'elles sont utilisées dans différents pavés n'ayant pas de liens hiérarchiques entre eux ou qu'elles doivent persister entre différentes activations du pavé .

Ces données internes correspondent dans l'approche de Namur aux collections d'information et dans BDL aux fichiers.

b2. Les fonctions.

Dans le cadre de la spécification du lien entre les résultats et les arguments, on peut être amené à introduire différentes fonctions (cfr "fonction I", "fonction 2",...) ainsi que différentes données intermédiaires (cfr "int").

La réalisation de la fonction décrite peut être conditionnelle et soumise alors à l'évaluation d'un prédicat (fonction booléenne).

Spécification du problème : NOM-FONCTION

Lexique	Profil	Description formelle
résultat (commentaire libre) types prédicats	types. result1 : RESULT1 result2 : RESULT2 doc.result, doc.arg : DOC arg : ARGUM int : INT prédicats prédicat : ARGUM x DOC ⟶ booléen prédicat1 : ARGUM x DOC ⟶ booléen prédicat2 : ARGUM x DOC ⟶ booléen fonctions fonction1 : ARGUM x DOC ⟶ RESULT1 x DOC fonction2 : INT ⟶ RESULT2 fonction3 : ARGUM ⟶ INT de base dans INT	résultats externes : result1, result2 interne : doc.result arguments externes : arg tq prédicat (arg, doc.arg) interne : doc.arg Spécification de la relation résultats-arguments. predicat1 (arg, doc.arg) et (result1, doc.result) = fonction1 (arg, doc.arg) ou$_{ex}$ predicat2 (arg, doc.arg) et (result2) = fonction2 (int) et (int) = fonction3 (arg) pour arg il existe {result} tq il existe 0, 1 ou plusieurs result ∈ RESULT1 ou$_{ex}$ il existe 0, 1 ou plusieurs result ∈ INT Structure des gardes predicat (arg, doc.arg) ≡ predicat1 (arg, doc.arg) ou$_{ex}$ predicat2 (arg, doc.arg)

figure 6. Exemple de spécification d'un pavé de traitement.

Chaque fonction sera définie par un domaine (partie droite) et un codomaine (partie gauche) :

+ dans le domaine, peuvent figurer des arguments ou des intermédiaires (à condition que ces derniers se trouvent également dans les codomaines d'autres fonctions).

+ dans le codomaine, peuvent se présenter des résultats ou des intermédiaires .

Lorsque la fonction (rappelons que la notion de fonction introduite ne correspond pas toujours à la définition mathématique du terme) entre le dbmaine et le codomaine n'est pas bijective, nous spécifierons la relation existante entre les arguments et les résultats (cf. clauses de connextivité dans la spécification du schéma conceptuel donné par (BOD,80)).

Exemple.

A partir de la spécification de la figure 6, on peut déduire que "fonction 2" est bijective; par contre, on a :

$\forall$ int $\in$ INT, $\exists!$ arg $\in$ ARG tq

(int) = fonction 3 (arg)

mais

$\forall$ arg $\in$ ARG, $\exists$ int $\in$ INT tq

(int) = fonction 3 (arg)

($\forall$: pour tout,
$\exists!$: il existe exactement un,
$\exists$: il existe au moins un).

La fonction 3 dite "de base" correspond à une fonction terminale dans l'arborescence déductive des traitements.

b3. Les gardes.

Nous pouvons définir une condition ou un ensemble de conditions que doivent remplir les arguments à l'entrée du pavé; nous introduisons ainsi la notion de fonction gardée par un prédicat (au sens d'une commande gardée (DIJ,75)). Dans l'exemple de la figure 6, la garde "prédicat" est associée à la fonction " nom-fonction" et se décompose en 2 gardes "prédicat I" et

"prédicat 2" qui sont associées respectivement aux fonctions :"fonction I" et "fonction 2","fonction 3" au niveau du raffinement suivant.

b4. Structuration des fonctions et des prédicats.

Pour l'ensemble des fonctions définies dans un pavé, nous pouvons définir une structure à l'aide des opérateurs logiques ou, ou_{ex}, et et pour tout connectant leurs définitions. Ainsi, les opérateurs ou et ou_{ex} indiqueront que les réalisations des fonctions pourront s'exécuter respectivement d'une manière sélective non-déterministe ou déterministe, tandis que l'opérateur et permettra une exécution de ces réalisations dans un ordre quelconque ou non suivant la disponibilité des arguments.

Exemple.

et "déterminé" :

a = f(b) et b = f(c).

et "indéterminé" :

a = f(b) et c = f(d).

Quant à l'opérateur pour tout, il caractérisera une fonction dont la réalisation se déroulera d'une manière répétitive sur chaque élément d'une suite.

Il est à remarquer la grande similitude qui existe entre ce type de structuration de fonctions et celui introduit par Jackson (JAC,75) pour ses composants. En effet, Jackson décrit trois types de structure qui sont celles de la programmation structurée (DAH,72) :

- la structure séquentielle,
- la structure conditionnelle,
- la structure itérative.

Nous retrouvons, dans notre démarche, ces trois techniques de décomposition accompagnées, cependant, dans le cas des deux premières d'une notion de déterminisme ou de non-déterminisme dans le choix et dans la séquence. Cette possibilité de non-déterminisme est exclue dans une structuration à la "Jackson", alors que Dijkstra (DIJ,75) a montré

l'intérêt d'une telle possibilité au niveau de la structuration des traitements.

Nous pouvons également structurer les prédicats en les décomposant à l'aide des connecteurs logiques $\underline{ou}_{ex}$, $\underline{ou}$ et $\underline{et}$ en des prédicats plus élémentaires qui interviendront à leur tour comme gardes de nouvelles fonctions. Cette décomposition prendra fin lorsque les fonctions "gardées" seront terminales; les prédicats pourront alors être explicités à l'aide de fonctions "terminales" (de base) à résultats booléens qui seront spécifiées dans les pavés de données relatifs à leurs arguments.

Remarque. Cette structuration des prédicats revient à définir un arbre des prédicats. L'utilisation de celui-ci pourra servir lors de la vérification de la complétude.

b5. Revenons sur la notion de fonction "terminale" qui va nous guider dans notre choix pour arrêter ou développer davantage l'arborescence des fonctions et des prédicats.

Une fonction terminale (de base) est une fonction que nous ne trouvons plus nécessaire ni pertinent de décomposer et que nous décidons de décrire dans les pavés de la structure de données correspondant aux domaines et codomaines (cf. la structuration des données au §3).

Exemple.

Soit (a) = fonction (b),une fonction terminale. Cela veut dire que l'on trouvera la description de cette fonction dans le pavé se rapportant à la donnée a (en termes de fonction de création) et dans le pavé se rapportant à la donnée b (en termes de fonction de consultation).

Le problème est définir ce qu'on entend par "non-nécessaire" et "non-pertinent" de décomposer. A ce sujet, la littérature est très abondante mais malheureusement, on ne peut pas dire qu'elle offre des solutions universelles en la matière.

Nous allons, quant à nous, nous baser sur quelques critères que nous illustrerons sur l'exemple suivant.

Exemple.

Les résultats et les arguments d'un problème de facturation sont :

résultats: externe. Une facture dont la structure hiérarchique des données est la suivante:

facture = (date-facture,identification-client-facture (nom-client-facture, adresse-client-facture), ligne-de-facture (numéro-produit-facturé, nom-produit-facturé, quantité-facturée), renseignements-facture (...)).

interne. Une documentation produit (doc.prod') telle que pour chaque produit, on trouve les renseignements suivants :

produit = (nom-produit, numéro-produit, quantité-stock-produit, prix-unitaire-produit).

arguments externes:

a) une commande telle que :

commande = (identification-client-commande (nom-client-commande, adresse-client-commande), ligne-de-commande (numéro-produit-commandé, quantité-produit-commandé)).

b) une date-du-jour.

interne : doc.prod: la documentation produit décrite telle que ci-dessus.

Si on suppose que la commande est correctement libellée, il s'agit de calculer la quantité de produit livrable, de mettre à jour la quantité de produit en stock et de préparer la facture.

Les critères retenus pour la mise en évidence des fonctions terminales seront les suivants.

(i) Ne pas définir de fonctions terminales ayant à la fois des effets (création,mise-à-jour,...) sur plusieurs objets (résultats externes,internes; arguments externes,internes) simultanément, c'est-à-dire faire en sorte que le codomaine d'une fonction terminale porte sur un seul objet.

Exemple.

Dans notre problème, les objets sont la facture, la documentation

produit, la commande et la date du jour.

Une fonction qui ne sera pas terminale est, par exemple :

(facture, doc.produit') = construction-facture (commande, date--du-jour, doc.produit) (I)

Une fonction qui sera terminale est, par exemple :

(doc.produit') = mise-à-jour-stock (doc.produit, ligne-de-commande)

Ce critère assure que tous les effets d'une fonction terminale seront limités à un objet (argument, résultat, intermédiaires).

(ii) Ne pas définir des fonctions terminales dont le nombre d'objets impliqués dans le domaine de la fonction soit trop grand.

Exemple.

La fonction définie en (I) possède déjà un domaine de taille raisonnable (trois objets différents), il ne faudrait pas qu'il soit plus grand.

Ce critère assure aux fonctions terminales une certaine lisibilité et une certaine transparence, c'est-à-dire qu'il est relativement aisé de comprendre leurs effets rien qu'en regardant leur codomaine et domaine vu la taille réduite de celui-ci.

Cependant, il faut remarquer que ce critère n'est pas toujours applicable; ainsi, il se pourrait que la relation entre les données soit complexe et ne permette pas une "factorisation" dans les arguments.

(iii) Raffiner successivement les fonctions et leurs prédicats jusqu'au moment où les données du domaine de la fonction correpondent aux données sur lesquelles porte la garde.

Exemple.

La fonction suivante ne sera pas terminale :

ligne-livrable-totalement (ligne-de-commande, doc.produit)

et (identification-client-facture, numero-produit-facture,

quantité-facturée) = construction-facture-pour-commande-en--tièrement-satisfaite (identification-client-commande, numero--produit-commande, quantité-produit-commandée).

La fonction suivante pourra être terminale :

ligne-livrable-totalement (ligne-de-commande, doc.produit) et (numéro-produit-facturé, quantité-facturée) = construction--ligne-facture-pour-ligne-commande-entièrement-satisfaite (numero-produit-commande, quantité-produit-commandée).

Pour la première fonction, seule une partie du traitement est concernée par le prédicat (en effet, la construction de l'identification sur la facture n'est pas concernée par le fait que le produit soit livrable totalement ou partiellement).

Grâce à ce troisième critère, nous isolons mieux ainsi les parties critiques (soumises à la vérification d'une ou de plusieurs conditions) de la spécification.

Remarque.

Il va de soi que ces quelques critères ne doivent pas être suivis d'une manière totalement rigoureuse et ne constituent pas un référentiel pour tous les cas. Ils constituent uniquement un guide pour le spécificateur qui connaît bien sa tâche et les objectifs qu'il veut atteindre.

c) le profil.

cI. les types.

Pour chaque donnée manipulée dans la description formelle (résultats, arguments, intermédiaires), nous en donnerons le type, c'est-à-dire le nom attribué à l'ensemble des données (auquel celle considérée appartient) caractérisées par les mêmes propriétés.

c2. les prédicats.

Il s'agit de fonctions (domaine $\longrightarrow$ booléen) dont le domaine est constitué par le produit cartésien (X) des types des données auxquels les arguments appartiennent.

c3. les fonctions.

Nous décrirons les domaines et codomaines des fonctions (domaine → codomaine) à l'aide des produits cartésiens des types de données auxquels les arguments et les résultats appartiennent. Rappelons que les fonctions terminales ("de base") sont décrites dans les pavés de donnée.

d) le lexique.

dI. les résultats.

Nous définirons en langue naturelle les résultats du pavé et les liens qui existent entre ceux-ci et les arguments.

d2. les types.

Pour tous les types introduits dans le profil, nous en donnerons une brève description en langue naturelle.

d3. les prédicats.

Nous préciserons, de manière informelle, la fonction remplie par chacun des prédicats rencontrés dans le profil.

Remarque.

Les fonctions sont décrites par la connaissance des types des arguments et des résultats; il s'agit donc d'une description externe en termes d'entrées et de sorties.

Représentation graphique de l'arbre déductif.

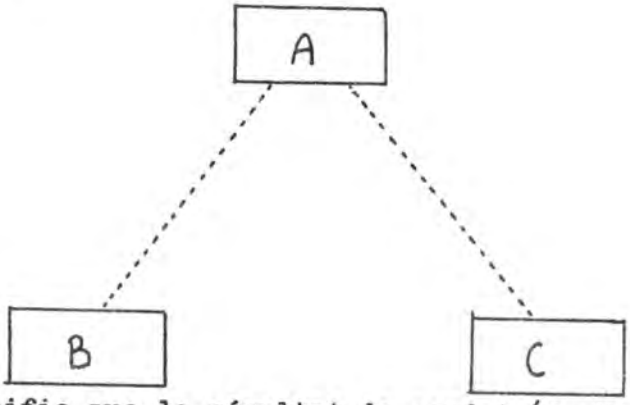
Au niveau de la présentation des différents pavés de traitement obtenus par raffinements successifs, nous avons retenu le formalisme employé dans figure 6. Le développement en parallèle du lexique, du profil et de la description formelle permet une description précise , sans ambiguïté, et compréhensible par une personne non complètement familiarisée à l'aspect formel de la spécification; d'autre part, le fait que les différentes colonnes se trouvent côte à côte permet à tout moment de vérifier l'adéquation de l'énoncé formel à l'énoncé informel.

Concernant la représentation de l'arbre déductif des pavés de traitements, nous avons opté pour une description graphique basée directement sur les

primitives introduites pour structurer les fonctions (cf.b4 ci-dessus). C'est ainsi que nous trouverons les présentations suivantes.

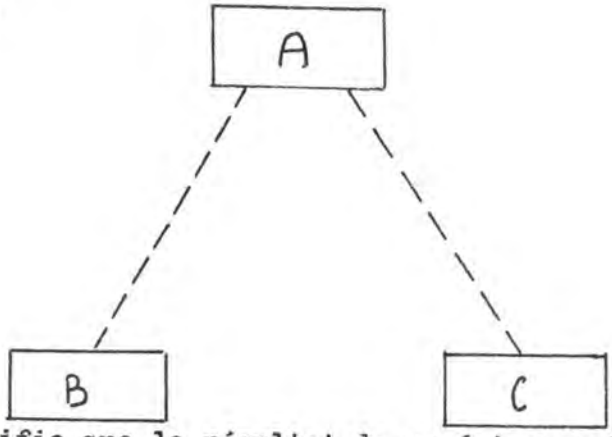
Soient A, B, C, D des noms de pavés de traitements, on peut avoir :

- pour le ou$_{ex}$,



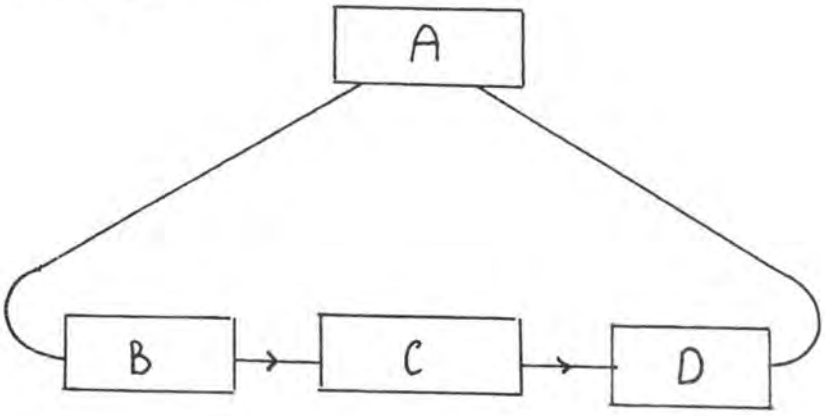
ce qui signifie que le résultat de pavé A (de la fonction A) sera soit le résultat du pavé B ou soit le résultat du pavé C.

- pour le ou ,



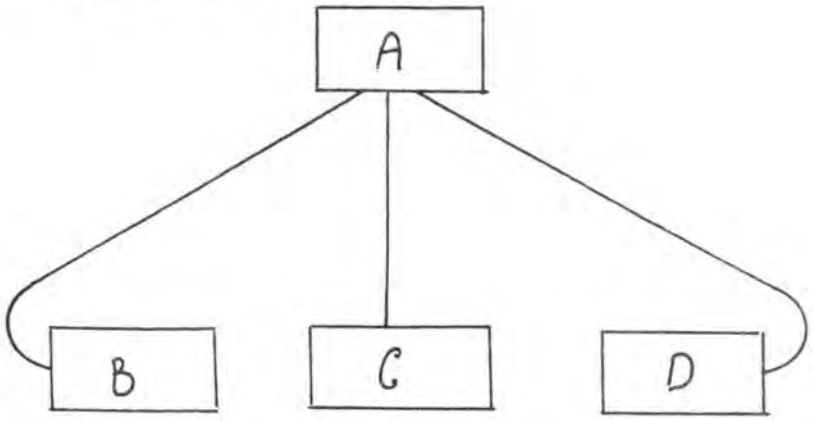
ce qui signifie que le résultat du pavé A sera le résultat du pavé B ou (non-exclusif) le résultat du pavé C.

- pour le et avec ordre,



ce qui signifie que le résultat du pavé A sera obtenu par le résultat du pavé B, du pavé C et du pavé D, la réalisation du pavé B se faisant avant celle du pavé C qui se fait elle-même avant celle de D.

- pour le <u>et</u> sans ordre,



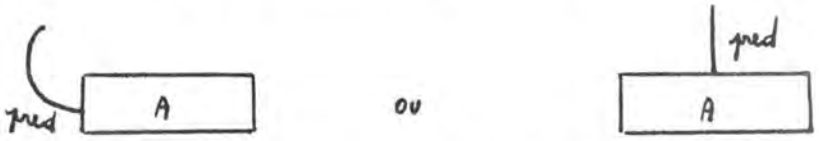
ce qui signifie que le résultat du pavé A sera obtenu par le résultat du pavé B, par le résultat du pavé C et par le résultat du pavé D, la réalisation des pavés B, C, D se faisant sans contraintes de séquence.

- pour le <u>pour tout</u>,

ce qui signifie que le résultat du pavé A sera obtenu par l'application itérative du pavé B.

Remarque.

Si nécessaire, on pourra expliciter dans l'arbre déductif qu'un prédicat "pred" garde un pavé A en utilisant, par exemple, la notation :



La garde est attachée à l'arc du graphe joignant A au pavé qui l'utilise (au sens de la hiérarchie "utilise" (PAR,74)).

§3. La structure des données.

Jusqu'ici, nous avons décrit la structure des pavés prenant en compte les traitements, mais conjointement, nous avons aussi introduit progressivement un certain nombre de données et de fonctions portant sur celles-ci. Il s'agit maintenant de décrire ces données en terme de leurs propriétés dont la plupart seront celles des fonctions terminales introduites.
Notre approche déductive du problème nous a permis de mettre en évidence trois sortes d'objets :

- les arguments du problème;
- les résultats du problème;
- les intermédiaires permettant la description du problème.

A chacun de ces objets, nous allons faire correspondre un type (nom attribué à l'ensemble des données caractérisées par les mêmes propriétés) et un pavé de description. Toutes les fonctions terminales que nous avons décrites ne manipulent pas toujours ce type mais bien parfois, des parties de celui-ci; nous sommes donc amenés à construire pour chaque type, si c'est nécessaire, une structure de sous-types.
Les différents constituants d'un pavé de données seront les suivants :

(on trouvera un exemple d'un tel pavé à la figure 7)

a) le titre.

S.I. indiquera qu'il s'agit d'un pavé de donnée jouant un rôle dans le système d'information.

" TYPE " est l'identificateur du type décrit dans le pavé.

b) la description formelle.

" TYPE " : il s'agit du nom du type décrit dans ce pavé. Cette donnée pourra être un des arguments du problème, un des résultats du problème (éventuellement les deux à la fois, s'il existe un résultat et un argument du même type), ou un des intermédiaires du problème.

" TYPE I,TYPE 2, ..." : le type décrit ci-dessus peut être décomposé en un ensemble hiérarchique de sous-types formant une arborescence. Sous les

feuilles de celle-ci (sous-types non décomposés),nous en donnerons le format.

La description de la structure sera faite à l'aide d'une structure de parenthèse.

les invariants: il s'agit de propriétés qui doivent être vérifiées avant et après toute opération définie sur toute donnée d'un type ou sous-type concerné.

Exemples .

- si un type est une suite d'éléments t, on peut donner des contraintes de cardinalité sur cette suite (le nombre d'occurrence des éléments t dans la suite).
- on peut donner pour les types élémentaires (non-décomposés) des contraintes d'intégrité sur le (ou les) domaine(s) de valeurs qui peuvent être prises par les occurrences de ces types.

* Les fonctions : ces opérations correspondent à des fonctions terminales ("de base") introduites dans les pavés de traitement. Elles auront la forme générale :

fonction (argument) résultat

et la description de la fonction se fera en caractérisant :

- la pré-condition associée à l'argument;
- la post-condition associée au résultat.

Ce genre de caractérisation axiomatique est utilisée par (HOA,69) depuis les travaux de (FLO,67).

On pourra distinguer différentes sortes de fonctions de base :

- fonctions de création et de mise à jour.

  Les résultats de ces fonctions seront des occurrences des types définis dans ce pavé de données.

- fonctions de consultation.

  Ces fonctions auront :

S.I. TYPE

Lexique	Profil	Description formelle
types.	types donnée3 : TYPE3 : S.I. donnée1 : TYPE1 donnée2 : TYPE2	TYPE (TYPE1 : (TYPE2 : chaîne alphabétique [30]; TYPE3 : chaîne alphanumérique [10]; TYPE4 : chaîne numérique [6])); TYPE5 : chaîne alphabétique [15]) invariants
invariants		
fonctions	fonctions fonct1 : TYPE3 → TYPE2 fonct2 : TYPE2 → booléen accèstype2 : → TYPE2	fonctions de mise à jour et de création fonct1 (donnée3) donnée2 pré : ... post : ... de consultation fonct2 (donnée1) booléen pré : ... post : ... accèstype2 ( ) donnée2 pré : / post : ...

figure 7. Exemple de spécification d'un pavé de donnée.

+ soit un résultat réduit au couple de valeurs (faux,vrai). Il s'agit alors de fonctions qui interviennent dans l'évaluation d'un prédicat (décrit dans le § 2).
+ soit pas d'argument et un résultat qui est une occurrence du type défini dans le même pavé de données.

Cette fonction joue alors le rôle d'une fonction d'extraction de l'information. Le nom de cette fonction sera obtenu en juxtaposant le mot "accès" et le nom du type de donnée auquel on veut accéder.

Exemple.

Soit une fonction de base décrite dans un pavé de traitement et ayant la forme suivante : (a) = fonction (b)

dont le profil est: fonction : B ⟶ A .

Au niveau des pavés de données, nous trouverons les spécifications suivantes :

- pour le pavé où se trouve le type A, on aura une fonction de "création et de mise à jour" de la forme: fonction (b)a

avec le profil : fonction :B ⟶ A

-pour le pavé où se trouve le type B, on aura une fonction de "consultation" de la forme : accès B ( ) B

avec le profil : accès B : ⟶ B.

c) le profil.

Pour toutes les données utilisées dans l'expression des invariants ou des fonctions, nous en donnerons le type. Si ce type n'est pas un de ceux définis dans le même pavé, nous le ferons suivre de S.I., ce qui signifiera que sa description se trouvera dans un autre pavé de données.

Pour les fonctions, nous définirons leurs domaines et codomaines à l'aide de produits cartésiens des types de données respectivement arguments et résultats. Il est à remarquer la redondance de cette description qui pouvait être déduite de la spécification des types.

d) le lexique.

Nous donnerons une description en langue naturelle

- des types de données manipulés,
- des invariants et des fonctions définies.

La représentation de la spécification d'un pavé se fera de la même manière qu'à la figure 3 (§2), c'est-à-dire en présentant la description formelle, le profil et le lexique en parallèle. Ce graphisme aura les mêmes avantages que dans le cas du pavé de traitement (cf. §2,la structure des traitements).

L'introduction de ces pavés de données permettent :

- le regroupement en un point de la spécification des fonctions portant sur un même type et dispersées dans plusieurs pavés de traitements (fonctions de base et fonctions d'accès déduites).
- une description des propriétés des types qui, excepté les invariants, sont les propriétés des fonctions portant sur ces types. Une telle démarche permet de conserver un maximum d'abstraction en ne prenant en compte aucun détail concernant la représentation des données et l'implémentation des opérations associées.

Une telle approche nous a permis de définir ce qu'on appelle communément, dans la littérature, des types abstraits (LIS,75a). On trouvera dans le premier chapitre (cf.I.6., description du langage "Z") une discussion des nombreux avantages procurés par des pavés de données reposant sur un tel concept.

§ 4. Conclusions sur cette méthode.

La méthode déductive de spécification que nous avons employée nous a permis de décrire les différents traitements d'une manière arborecente et hiérarchisée et de construire conjointement la structure des données sous la forme de " types abstraits " incluant la description des traitements terminaux dans l'arborescence. Nous allons maintenant tenter d'éva-

luer une telle approche en considérant une "bonne" spécification (cf. chapitre I.5.).

a) Constructibilité.

Par une démarche descendante, par une découpe en différents pavés hiérarchisés, une meilleure maîtrise de la complexité dans l'élaboration de la spécification d'un système d'information devrait être favorisée.

b) Cohérence et complétude.

Il est possible de faire un certain nombre de contrôles tels que:

-s'assurer si toutes les données sont définies en termes de types,

- vérifier si toutes les fonctions (de base ou non) ont été explicitées,
- contrôler si tous les prédicats sont exprimés à l'aide de fonctions de base à résultats booléens,
- s'assurer que l'ensemble des cas a bien été traité en vérifiant que toutes les combinaisons de conditions ont été prévues,
- vérifier la cohérence des traitements et des prédicats à l'aide de techniques du genre de celles que l'on utilise dans le cadre des tables de décision.

On verra des exemples de ces contrôles dans le troisième chapitre (cf. 3.I.)

c) Aspect formel.

La description formelle de la spécification par l'utilisation d'un langage proche du langage mathématique devrait encourager une certaine rigueur dans l'élaboration des spécifications.

d) Compréhensibilité.

Grâce à la description informelle, on pourrait très bien reconstruire toute la spécification d'une manière structurée et en langue naturelle.

e) Fidélité.

L'adéquation de la spécification au problème est toujours difficile à prouver. Cependant, le développement structuré et progressif de la spécification au niveau de chaque pavé devrait favoriser, et donc faciliter,

le travail de vérification d'une telle adéquation. Au niveau de l'ensemble des pavés, une technique consisterait à vérifier a posteriori que les solutions correspondant à certaines valeurs de données sont bien celles que l'on attendait (FIN,79).

Il faut souligner que la rigueur de la méthode peut bien souvent soulever des points mal définis ou indéfinis dans le problème initial. Ainsi, dans l'application traitée dans le troisième chapitre, une analyse en profondeur de la proposition d'automatisation nous a convaincu, par exemple, de la mauvaise définition de la notion d'expédition et du manque de définition de la notion de frais de transport.

f) Minimalité.

De telles spécifications s'attachent à décrire le "quoi" d'une fonction plutôt que son "comment" sans introduire de détails d'implémentation ou de représentations concrètes pour les données (cfr. types abstraits).

g) Extensibilité.

Comme toutes les décisions concernant les fonctions et les données sont enfermées dans des pavés définis à l'aide d'une décomposition hiérarchique et descendante du problème, des changements ultérieurs dans les spécifications doivent pouvoir être facilement localisés aux pavés concernés.

Il semble donc que la méthode définie présente un certain nombre des qualités requises; il faut cependant nuancer ce jugement, car au lieu de parler de la fidélité de la spécification au problème, il eut été plus adéquat de parler de fidélité à des parties du problème. En effet, comme nous allons le voir dans le paragraphe suivant, un certain nombre de sous-problèmes (données et traitements manquants, enchaînements dictés par l'environnement organisationnel, ...) n'ont pu être pris en compte dans notre spécification déductive; il s'agit, dès lors, d'introduire de nouveaux outils permettant de caractériser complètement la définition du problème.

2.2. Adaptation de la spécification du système d'information à l'environnement organisationnel.

Dans le premier paragraphe de ce second chapitre, nous nous sommes donnés une méthode permettant d'aboutir à une première définition du système d'information en présentant, d'une part, les fonctions et leurs interrelations (arborescence déductive) et d'autre part, les données manipulées et leurs propriétés (types abstraits). Ce premier schéma général de spécification a permis de présenter une sorte de noyau du système, reprenant la logique inhérente à l'application à traiter, telle qu'on a pu la dégager d'un processus déductif.

Il reste cependant un certain nombre de caractéristiques qui sont propres à un environnement organisationnel particulier et qui ne peuvent être dégagées d'un processus déductif; celles-ci doivent en quelque sorte se "greffer" sur le noyau logique. C'est à l'approche et à l'intégration de ces contraintes organisationnelles que nous allons d'abord nous consacrer. C'est ainsi que nous essaierons de traduire ces contraintes organisationnelles sous forme de traitements et de données manquants, sous forme d'enchaînements entre traitements et sous forme de traitements d'interface entre le système d'information et l'environnement organisationnel.

Ensuite, nous nous poserons pour la première fois la question du "comment" afin de pouvoir distinguer les parties manuelles et automatiques du système d'information en vue des étapes ultérieures de développement du système.

Enfin, nous tirerons les conclusions et quelques perspectives d'une telle démarche.

2.2.I. Traitements et données organisationnelles.

§I. Spécification des traitements et données manquants.

a) Au départ de notre raisonnement, nous avons considéré le système d'information à décrire comme une "boîte noire" produisant un certain nombre de données de sortie (résultats) à partir d'un certain nombre de données

d'entrée (arguments). Si nous avons différencié certains arguments et certains résultats, c'est parceque les éléments d'information correspondants requéraient une telle différentiation.

Exemple.

Si la sortie d'un S.I. de gestion de commandes est une facture de la forme suivante :

- un ensemble de lignes de commande livrables et un montant total;
- un ensemble de lignes non livrables.

On définira par la méthode, non pas un, mais bien deux résultats :

- la partie livrable de la commande;
- la partie non-livrable.

Si cette approche permet de mettre en évidence la logique sous-jacente au problème, elle ne convient cependant pas nécessairement à l'utilisateur qui peut vouloir appréhender ces résultats et ces arguments sous une autre forme (par exemple, dans ce cas-ci, la facture). Nous allons décrire dès lors un certain nombre de traitements et de données additionnels qui permettront de faire le lien entre les objets logiques considérés jusqu'ici et ceux attendus par l'utilisateur. En l'occurrence, dans l'exemple, l'obtention d'une facture sera assurée par un traitement de regroupement des parties livrable et non-livrable.

b) Ces pavés ne sont malheureusement pas les seuls qui manquaient dans notre spécification. En effet, il se peut que l'arborescence déductive des traitements et des données n'ait pas permis de décrire toutes les opérations et toutes les informations nécessaires à une organisation particulière. C'est ainsi, qu'à côté du noyau logique qui a été obtenu, il existe des décisions purement organisationnelles propres au système étudié, qui ont été introduites pour des raisons particulières d'efficacité ou bien qui sont la conséquence d'une structure d'organisation bien spécifique qu'il ne nous appartient pas de remettre en cause.

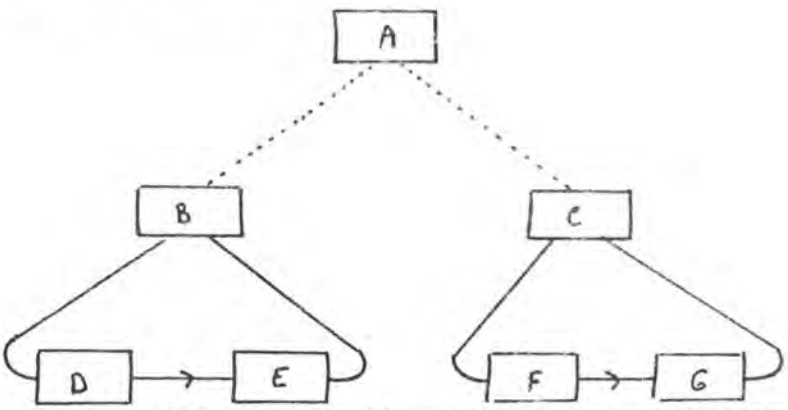
Exemples. Une procédure d'ordonnancement spécifique à un agencement en

magasin donné, un traitement de contrôle et de recouvrement d'erreur, ... Pour pouvoir les mettre en évidence, nous allons relire le texte constituant le cadre de la solution fournie par l'analyse d'opportunité et nous allons vérifier si chacune des informations de ce texte a sa contrepartie dans la spécification; si ce n'est pas le cas, on introduira de nouveaux pavés de traitements et de données associés à ces informations dans un cadre d'expression analogue à celui utilisé jusqu'ici. Une fois construits ces nouveaux pavés, il restera à les "greffer" sur l'arbre déductif en modifiant éventuellement certains arguments et résultats.

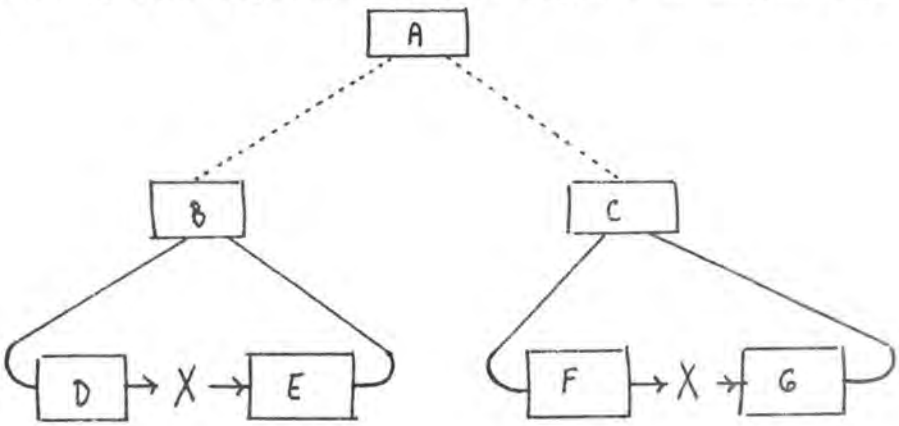
Au niveau de la représentation graphique de la nouvelle structure, la rupture d'une relation logique entre deux pavés de l'arbre due à l'introduction de traitements dits "organisationnels" sera indiqué par une croix (X).

Exemple.

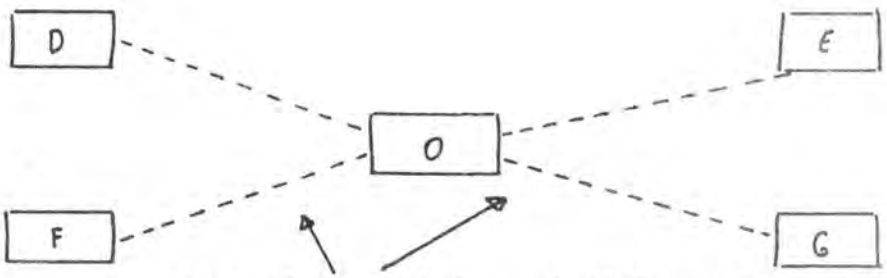
Soit l'arbre déductif suivant :



et soit un pavé "O" de traitement "organisationnel" dont les arguments sont les résultats de D et de F et dont les résultats sont les arguments de E et G. Nous représenterons la rupture de séquence par :



et



ces relations indiquent d'où proviennent les arguments et où vont les résultats du pavé "O".

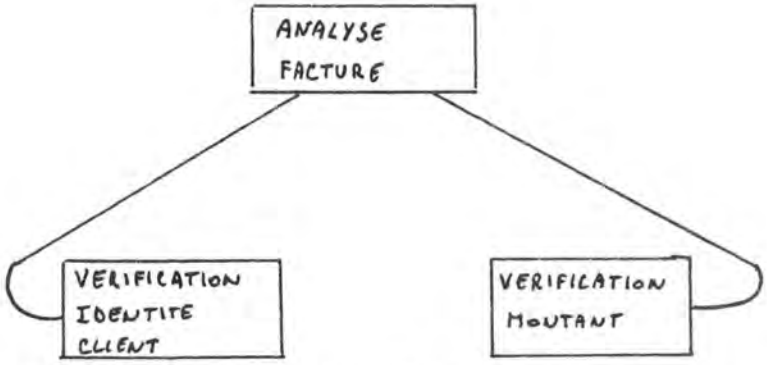
On trouvera des exemples de cette rupture de séquence dans le paragraphe 2.I. du chapitre 3.

§2. Spécification des interrelations "organisationnelles" entre traitements.

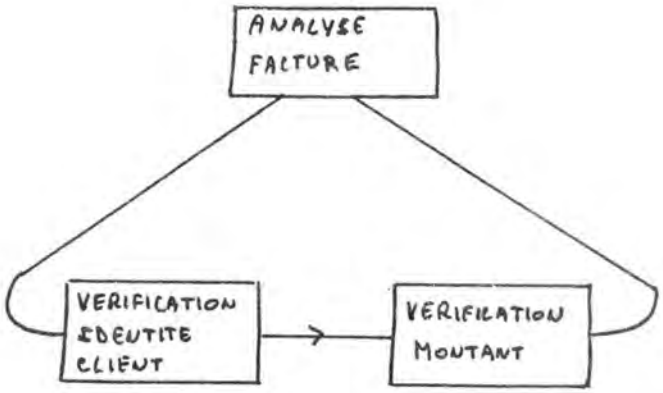
Jusqu'à maintenant, les interrelations que nous avons construites entre les différents pavés de traitement étaient uniquement dictées par une dépendance entre les données. Cependant, pour rendre notre spécification compatible avec les contraintes organisationnelles, il faut tenir compte également d'enchaînements rendus nécessaires par celle-ci. C'est ainsi qu'au niveau de la spécification et de la représentation graphique de l'arbre des fonctions, nous ajouterons de nouveaux séquencements.

Exemple.

Soit une fonction d'analyse d'une facture se subdivisant en une fonction de vérification de l'identité du client et une autre de vérification du montant de la facture. Ces deux dernières fonctions pourraient s'exécuter dans un ordre quelconque, ce qui d'une manière graphique se représenterait par :



Si une contrainte organisationnelle stipule que chaque facture est prise en charge par un employé qui vérifiera d'abord l'identité avant le montant, on aura la modification graphique suivante :



§3. Spécification des traitements et pavés d'interface.

Chaque pavé de traitement décrit est caractérisé par un ensemble de données en entrée (les arguments) et un ensemble de données en sortie (les résultats). Pour chacun de ces éléments, il s'agit d'examiner s'il n'intervient pas à la fois dans le système d'information et dans l'environnement organisationnel, auquel cas un pavé d'interface sera associé à cet élément.

C'est ainsi que l'on sera amené à distinguer les pavés d'interface de saisie de données pour les arguments et les pavés d'interface de diffusion de données pour les résultats. Dans ces pavés, outre la description des procédures d'acquisition et de diffusion de données, la définition du format des informations manipulées (layout) et la spécification de différents contrôles de validation des informations, on peut associer à chaque pavé d'interface des traitements additionnels spécifiques auxquels les données doivent souscrire.

Les traitements d'interface seront spécifiés dans le formalisme habituel rencontré jusqu'ici; quant aux autres éléments, ils seront spécifiés le plus précisément possible à l'aide de la langue naturelle et éventuellement, pour éviter la lourdeur de la spécification de formats de documents, on pourra recourir à une description graphique à l'aide de dessins de bordereaux, de grilles d'écran, ...

Exemple.

Soit un pavé où l'argument est un "texte" et un sous-arbre à partir de ce pavé où l'on trouve à un niveau i un pavé de traitement avec une

"phrase" comme argument et à un niveau j (avec i<j) un pavé dont l'argument est "mot". S'il est dit dans le cahier de charges issu de l'étude d'opportunité que le texte sera introduit, par exemple, phrase par phrase au terminal par un opérateur, alors on décrira un pavé d'interface qui sera adjoint au pavé de traitement de la phrase. Dans ce pavé d'interface, nous mentionnerons la manière de rentrer une phrase au terminal, différents contrôles de validité quant aux caractères rentrés et éventuellement un format dans lequel la phrase doit être rentrée.

Remarques.

En général, ces interfaces assurent le contact entre l'utilisateur et le système d'information, mais il se peut que pour certains problèmes où, dès la phase de conception, on annonce l'utilisation d'un certain "hardware" (par exemple, deux ordinateurs), on soit amené à spécifier des interfaces entre le système d'information et le matériel existant (voir, par exemple, les spécifications en avionique de (HEN,80)).

Lorsque le pavé de traitement et la garde associée (cfr. définition chapitre 2.I.2., §2) sont réalisés à l'aide de processeurs différents (hommes ou machines, nous reviendrons sur cette notion dans le chapitre 2.2.2.), nous pouvons être amenés pour les arguments intervenant à la fois dans les fonctions de la garde et dans celles du pavé de traitement à décrire des pavés d'interface de saisie de données distincts pour la garde et le pavé de traitement car les procédures d'acquisition de ces informations peuvent être différentes.

2.2.2. Distinction entre les systèmes d'information manuels et automatiques.

Jusqu'ici, la spécification que nous avons donnée s'est toujours efforcée de prendre en compte la description du "quoi" avant celle du "comment". Toutefois, en introduisant les pavés d'interface entre le système d'information et l'environnement organisationnel, nous avons déjà pu être amenés à recourir à la notion de moyens nécessaires au traitement de l'information.

Comme le fait (DUF,80), nous pourrions, en fait, classifier ces différents moyens comme étant :

a) les moyens actifs de traitement (ou processeurs).

Un traitement ne peut être réalisé sans l'usage d'au moins un de ces moyens. Ces processeurs peuvent être :

- matériels. Il s'agit d'organes capables de fonctionner de manière autonome, au moins pendant un certain temps pour effectuer un traitement.
- humains. Il s'agit des personnes pouvant prendre en charge des traitements.

b) les moyens passifs de traitement de l'information.

Il s'agit d'éléments indispensables au traitement de l'information mais qui ne peuvent être jamais utilisés sans un moyen actif de traitement dont ils sont souvent considérés comme des constituants (par exemple, une machine à écrire, une ligne téléphonique, un terminal à écran, une imprimante, ...).

Pour compléter notre description du système d'information et pour commencer à préciser certains aspects du "comment" de sa réalisation, nous allons affecter les traitements aux différents processeurs disponibles. Cette affectation s'avère d'autant plus nécessaire que, dans la suite de notre travail, il va falloir distinguer le système informatique du système d'information et par là même, cerner la tâche de l'informaticien. Il aurait été malsain d'introduire une telle séparation dès l'étude d'opportunité (cfr. étapes de l'analyse dans l'introduction) car elle aurait contribué au "mythe" du S.I. réduit au système informatique, ce qui, comme le souligne (LEM,73), est une des confusions qui "obèrent la conception d'un système d'information" en risquant de ne plus prêter d'attention aux autres éléments du système (réseaux de communication, hommes, ...).

La prise en compte des différents processeurs se fera de la manière suivante. Pour chaque pavé de traitement constituant une feuille de l'arbre construit par la méthode déductive, pour chaque pavé "organisationnel",

nous allons mentionner le ou les processeurs nécessaires à la réalisation des fonctions de base (aussi bien ceux intervenant dans l'évaluation de la garde que ceux intervenant dans le traitement proprement dit). C'est ainsi que l'on mentionnera l'utilisation d'un ordinateur, d'une ou plusieurs personnes appartenant à des services déterminés, etc.

2.2.3. Discussion de cette approche.

Dans les paragraphes I et 2 de ce second chapitre, nous nous sommes efforcés de présenter une méthode de construction des spécifications d'un système d'information. Cette approche a été caractérisée par deux grandes étapes :

a) Utilisation d'une démarche descendante et déductive permettant de structurer le problème à l'aide de fonctions logiquement découpées et résultant en un arbre déductif des traitements; expression de la structure des données sous forme de "types abstraits".

b) Enrichissement de la structure des traitements et des données par la prise en compte des contraintes organisationnelles particulières : introduction de nouveaux pavés propres à l'organisation, adaptation des pavés existants, description des interfaces.

Ayant déjà mis l'accent dans le paragraphe 2.I.2. sur les avantages procurés par la démarche déductive, nous allons maintenant signaler quelques caractéristiques inhérentes à la méthode dans son ensemble et envisager quelques possibilités d'extension.

a) Tout d'abord, cette approche en deux étapes a permis de réduire la complexité de l'analyse du système sous étude en nous permettant de construire en premier lieu un noyau (cfr. supra) de la spécification du S.I. avant de nous attaquer à la spécification des aspects plus particulièrement organisationnels du S.I.. En revanche, il n'est peut-être pas toujours aisé de percevoir, en l'absence de critères précis, ce qui est, d'une part, proprement organisationnel et, d'autre part, logique intrinsèque de l'application considérée.

b) D'autre part, outre le fait qu'elle permet de structurer notre description du système, cette méthode va plus loin en nous permettant de distinguer et d'isoler des contraintes propres à l'organisation considérée. Une telle localisation assure une maintenance plus aisée du système face aux fréquentes modifications de l'environnement qui ne manqueront certainement pas de survenir.

c) En outre, cette approche, par sa rigueur et par son caractère méthodique, permet d'amener le spécificateur à poser au demandeur de bonnes questions aux bons moments concernant différents choix rendus possibles par le "flou" entourant parfois le cadre général de la solution retenue à l'issue de l'étude d'opportunité, voire même certaines contradictions ou incomplétudes. A titre d'exemple, dans le traitement de l'application proposée dans le troisième chapitre, nous avons dû définir avec précision certaines données et traitements (frais de transport, numéro-commande, constitution d'une expédition, ...) et prêter beaucoup d'attention à la spécification des pavés d'interface.

d) Enfin, la spécification du système d'information obtenue de la sorte peut être à la base du dialogue avec toutes les personnes concernées dans la réalisation d'un projet (cfr. paragraphe I.3. du premier chapitre) :

- avec le demandeur. On pourra lui remettre l'ensemble des documents obtenus spécifiant clairement ce que fera le S.I.. Eventuellement, s'il n'est pas habitué au formalisme employé (en particulier, la présentation en trois colonnes), un texte rédigé entièrement en langue naturelle pourra être obtenu d'une manière systématique. On pourrait également concevoir des outils qui, à partir de cette spécification, permettraient de construire une maquette du S.I. et par là juger de sa faisabilité en terme de disponibilité de ressources, d'efficacité des règles de décisions et de performances. Il faudrait cependant, dans ce cas, pouvoir fournir davantage de renseignements concernant les ressources (processeurs automatiques ou humains, données, quantifications, ...) et se donner un modèle précis d'ex-

pression de la dynamique des traitements (BOD,79c), (DUF,80).

- avec les utilisateurs. Un document indispensable, que nous pourrions construire à partir de la spécification et remettre à ceux-ci, est un graphe de circulation. En effet, celui-ci a comme mérite de donner une vue globale du S.I., de mettre en évidence les processeurs, les tâches à accomplir, les liaisons entre elles et les informations qui circulent (données temporaires ou permanentes). Nous verrons ultérieurement, à partir d'un exemple de spécification (cfr. paragraphe 3.2. du troisième chapitre) comment construire un graphe de circulation de type CORIG (MAL,7I).
Un autre document facilement réalisable à partir de la description des pavés d'interface sera celui spécifiant clairement les interfaces entre le système informatique et les utilisateurs.
- avec les rédacteurs des spécifications. Ils auront à leur disposition des documents faciles à contrôler et à modifier. En particulier, la représentation graphique de l'arbre déductif facilite grandement la localisation d'un traitement, son but et ses interrelations avec les autres traitements.
- avec les responsables de l'analyse de programmation. Ils auront des spécifications qu'on espère compréhensibles et non-ambiguës de la partie du S.I. qu'ils ont à automatiser.

Avant de porter notre attention sur le travail d'analyse de programmation et en particulier sur le prolongement de la méthode en vue d'exploiter au mieux les spécifications dans le développement du système informatique, il convient au préalable d'aborder un dernier ensemble de spécifications relatives aux performances globales du système.

2.3. Spécification des contraintes globales de performance.

Jusqu'ici, dans les deux premiers paragraphes de ce second chapitre, après avoir dégagé par un processus déductif un noyau constituant une première définition du système d'information, nous avons appréhendé un certain nombre de caractéristiques propres à l'environnement organisationnel, qui sont venues se "greffer" sur le noyau logique de base. Cependant, ces deux premières étapes ne permettent pas de définir tous les aspects d'une spécification; en effet, le demandeur, après que son étude ait été réalisée lors de l'analyse conceptuelle, émet souvent certaines considérations quant aux performances qu'il attend de son système.

Exemples.

- le temps de réponse et la durée du cycle de traitement de l'information dans des circonstances normales et dans les périodes de pointe,
- le temps de réponse pour une consultation,
- la sécurité du système face à des incidents,
- la confidentialité de certaines informations.

Ces contraintes sont importantes à mentionner car elles auront de sérieuses répercussions au niveau de l'analyse de programmation où l'architecture logicielle conçue devra respecter ces contraintes.

Dans le cadre des applications que nous envisageons dans ce travail, nous sommes amenés à décrire des systèmes multiutilisateurs pour lesquels les temps de réponse doivent être les plus brefs possibles.

L'étude de tels systèmes entraîne l'introduction de différentes notions dont celle d'interférence entre pavés de traitement. Nous dirons que deux pavés (ou deux parties de pavé) interfèrent si le premier pavé (ou une partie de ce pavé) ne préserve pas la sémantique de l'autre et vice versa.

A partir de là, nous pouvons définir les notions de parallélisme entre pavés de traitement, c'est-à-dire l'exécution possible de pavés (ou parties

de pavé) qui n'interfèrent pas et de mutuelles exclusions entre pavés de traitement lorsque ceux-ci (ou des parties de ceux-ci) interfèrent.
Pour obtenir un degré de parallélisme satisfaisant, il y a lieu d'assurer un grain d'atomicité (traitements mutuellement exclusifs) le plus fin possible.
De plus, des règles de concurrence additionnelles peuvent être ajoutées (priorités, préemptions, ...).

Exemple.

Soient une fonction A et une fonction B s'excluant mutuellement et à réaliser par un processeur X déterminé. Une règle souhaitée peut être la suivante : "donner la priorité à la réalisation de la fonction B par rapport à la fonction A". Cela signifie que le processeur X ne pourra jamais se consacrer à l'exécution de la tâche A tant qu'il restera une tâche B à accomplir.

Au niveau de la spécification, il s'agira donc d'exprimer des relations d'exclusion mutuelle, de parallélisme, de priorité, ... entre les pavés (ou parties de pavé) de traitement.
De telles relations pourront s'exprimer dans notre modèle de deux manières:

. si la relation porte sur des fonctions qui ne sont pas dites de "base" ou dont les codomaines portent sur des types d'objets différents, on exprimera le comportement désiré au niveau des différents pavés de traitement correspondant à la description des fonctions.

. si la relation porte sur des fonctions qui ont été décrites comme étant de "base" et dont les codomaines portent sur le même type d'objets, alors on exprimera le comportement désiré au niveau du pavé de donnée correspondant à l'objet considéré (type abstrait).

La spécification de ces relations se fera à l'aide de clauses exprimant :

- par une notation, la possibilité de parallélisme ou d'interférence.

Exemple.

Soient les fonctions de "base" de réservation de places d'avion et de libération de places s'excluant mutuellement et à réaliser par un processeur X déterminé.

Au niveau du type "place-avion", nous indiquerons pour les fonctions concernées : RESERVATION $\not\!\!/\!\!/$ LIBERATION .

Par contre, si le parallélisme avait été possible, nous aurions indiqué : RESERVATION // LIBERATION.

- d'une manière informelle, en langue naturelle, les clauses additionnelles de concurrence.

Remarque.

En général, lorsque nous n'indiquerons pas de clauses de concurrence pour une fonction, c'est qu'il n'y a pas d'interférence possible entre cette fonction et d'autres.

Nous sommes conscients que le traitement de la concurrence au sein de notre spécification est loin d'être totalement satisfaisant et qu'il devrait, par la suite, être traité avec beaucoup plus de profondeur et avec une plus grande rigueur. Cependant, si nous avons introduit dans ce paragraphe ces quelques éléments, c'est parce que nous les considérons comme devant jouer un rôle fondamental dans les choix qui guideront l'analyse de programmation.

2.4. Définition d'une architecture des traitements et des données.

A partir de l'analyse fonctionnelle concernant la partie automatique du système d'information, il s'agit de réaliser l'analyse de programmation suivant deux axes :

(i). pour les traitements : dans le cadre d'un système où, pour obtenir des temps de réponses les plus brefs possibles, on a introduit la notion de parallélisme entre les traitements (cf.2.3.), il s'agit

de s'orienter vers un système de gestion en temps réel, composé d'une architecture de traitements sous forme de processus communicants : il s'agit d'un ensemble de processus (correspondant chacun à une exécution particulière d'un programme) dont certains peuvent être simultanément actifs et qui communiquent et se synchronisent par échanges de messages. Ces messages sont des représentations des données du système (HOA,78).

(ii). pour les données : on suppose disposer d'un système de gestion de fichier ou de base de données.

Au niveau de la spécification des traitements, on a procédé de manière déductive, des résultats vers les arguments. Au niveau de l'implémentation, il s'agit de revenir à une démarche inductive, des arguments vers les résultats; en effet, les concepteurs de compilateurs pour langages déductifs (voir, par exemple, BDL (HAM,77)) se butent encore actuellement à des problèmes majeurs d'optimisation de code. Ces problèmes sont essentiellement liés à de trop grandes redondances tant dans les données que dans les traitements. Citons, entre autres, les problèmes suivants (LAM,79a) :

- des bouclages trop nombreux sur les arguments. En effet, avec un langage inductif "classique", une seule boucle est exécutée sur les données d'entrée, ce qui n'est pas le cas d'un langage déductif où les arguments seront considérés plusieurs fois.
- tous les arguments externes d'entrée d'un "pavé" cessent d'exister après l'exécution de ce pavé, ce qui, dans une implémentation grossière, entraînerait des recopies fréquentes pour la création de résultats différant peu de données contenues dans les arguments.
- absence de concept de procédure, absence de contraintes de séquencement (excepté celles requises par les données).

Au niveau de la structure des données, la spécification a été faite en terme d'objets logiques connus uniquement par l'intermédiaire

des fonctions qui les manipulent (types abstraits). Une partie de ces objets constitueront des messages entre les différents pavés, d'autres serviront de documentations et seront stockés dans des fichiers ou dans la base de données. Cependant, autant la notion de type abstrait a été utilisée pour la spécification de problèmes, autant on trouve peu de traces,dans la littérature, de son utilisation lors de la conception de bases de données.

En ce qui concerne la structure de données, nous passerons de la description par types abstraits à une description à l'aide du modèle relationnel binaire. Cette structure intermédiaire nous permettra de construire facilement un modèle d'accès aux données.

Pour établir une architecture des traitements, avant de passer à un choix d'une décomposition physique des traitements en unités d'exécution (programmes), il s'agit de structurer le système en un ensemble de composants appelés modules.

2.4.I. Structuration des traitements et dérivation d'un modèle relationnel binaire.

2.4.I.I. Dérivation d'un modèle relationnel binaire.

En ce qui concerne la structure de données, nous nous en sommes tenus jusqu'à maintenant à une représentation en termes de types abstraits. Cette description, correspondant au niveau conceptuel figurant dans le rapport ANSI-SPARC (ANS,75), permet d'exprimer d'une manière rigoureuse les propriétés des objets manipulés dans le système d'information; elle se prête néanmoins mal à une description logique (en termes d'accès logiques) de l'information à mémoriser et à son implémentation (description des accès physiques).

En effet, la description sous forme de types abstraits n'exprime pas les liens logiques entre les informations (ces liens se trouvent décrits au niveau des traitements) et n'offre pas d'outils (opérateurs) permettant

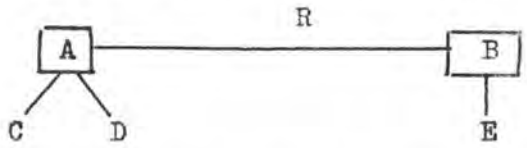
de travailler sur ces données.

Pour l'ensemble des informations, nous passerons de la description par types abstraits à une description à l'aide du modèle relationnel binaire (ABR,74). On trouvera une description détaillée du modèle relationnel binaire dans (HAI,81a); nous en reprenons ici les principales caractéristiques.

Ce modèle est une restriction du modèle relationnel généralisé (HAI,80) où I) il n'y a que des relations binaires (associations biunivoques).

2) les relations sont définies sur un domaine entité au moins.

La représentation graphique est la suivante :



où un domaine simple (c'est-à-dire un domaine (ensemble d'information) désignant des propriétés) est représenté par une suite de caractères (C, D, E) indiquant son nom;

un domaine entité (ou complexe,c'est-à-dire un domaine désignant un objet autonome du monde réel) est représenté par une suite de caractères (A, B) indiquant son nom et figurant dans un rectangle;

une relation (ensemble de toutes les associations de même nature) est indiquée par un arc entre deux domaines, cette relation porte éventuellement un nom (R).

Pour une relation, nous pouvons indiquer :

a) sa connectivité (expression de la dépendance fonctionnelle contenue dans la relation).

relation	connectivité	représentation
R (A,B)	n-n	A ⟶ R ⟵ B
R (A,B)	I-n	A ⟵ R ⟵ B
R (A,B)	n-I	A ⟶ R ⟶ B
R (A,B)	I-I	A ——— B

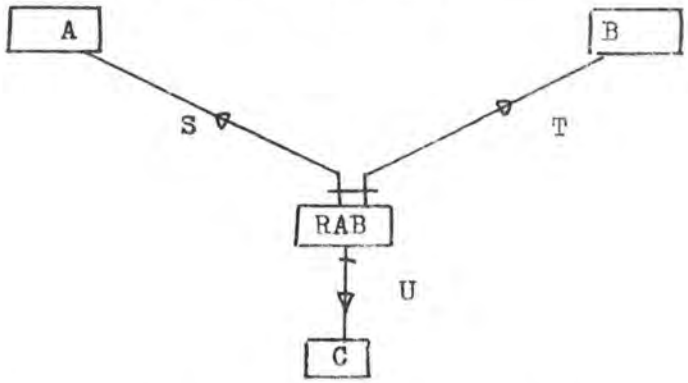
b) une contrainte d'existence d'un domaine entité.

Si la relation R(A,B) est telle que R[A]= A (projection de la relation R sur A, c'est-à-dire la relation obtenue en ne conservant de la relation initiale que les valeurs de A), cela signifie que chaque occurrence "a" du domaine entité A intervient dans un "tuple" de la relation R.

De manière graphique, cette relation dite "forte" pour A s'exprimera :

c) un identifiant multi-domaine.

Une relation appartenant au modèle relationnel généralisé de la forme R(A,B,C), où A, B, C sont des domaines entités, s'exprimera dans le formalisme binaire de la manière suivante :



tels que : à chaque tuple de la relation R correspond une occurrence du domaine RAB et R = S*T*U[A,B,C].

Le nouveau domaine RAB permet d'exprimer que les domaines A et B constituent un identifiant du domaine C.

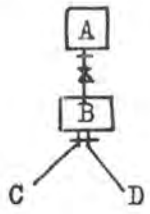
Nous pouvons aussi exprimer des contraintes au niveau d'une relation ou d'un ensemble de relations à l'aide d'opérateurs classiques tels que la jointure, la composition, la projection, ...

Au niveau des types abstraits, nous avions une représentation pour les données de chaque pavé de la forme suivante :

A (B: (C: chaîne numérique[IO];
D: chaîne alphabétique[20]))*

Cette structure, s'il n'y a pas d'autres propriétés exprimées par des invariants, pourra s'exprimer sous la forme d'un modèle binaire de la manière

suivante :



où les domaines entités correspondent aux données décomposables et où les données simples correspondent aux données élémentaires.

2.4.I.2. Structuration des traitements.

Comme le préconise (LAM,8I), on peut structurer les traitements d'un système d'information par niveaux hiérarchiques d'abstraction; à chaque niveau, les détails d'implémentation des niveaux inférieurs sont non pertinents et cachés. La relation définie entre deux niveaux hiérarchiques est la relation "utilise" (PAR,74) : un composant A de niveau supérieur utilise un composant B de niveau inférieur si le fonctionnement correct de A dépend de la disponibilité d'une implémentation correcte de B.
En suivant une telle démarche, nous pouvons structurer les traitements en cinq niveaux hiérarchiques du type "utilise".

Niveau cinq . Nous trouvons ici des composants que nous appelerons 'modules fonctionnels" et "modules inetractifs".
Chaque module fonctionnel est dérivé soit d'une fonction (correspondant à un pavé de traitement), soit d'une composition de plusieurs fonctions (correspondant à plusieurs pavés de traitement), mais jamais d'une décomposition plus fine d'une fonction dite de "base", ce qui ferait perdre tout aspect de fonctionnalité. Chaque module interactif est dérivé soit d'un pavé d'interface, soit d'une composition de plusieurs pavés d'interface.

Niveau quatre . Nous plaçons ici les modules d'entrée-sortie et d'accès.
Chaque module d'entrée-sortie peut être utilisé, au sens défini ci-dessus, par les modules interactifs afin d'assurer des opérations telles que saisie de données, validation de données, édition de données, gestion d'écrans, ...
Chaque module d'accès peut être utilisé par un module fonction-

nel ou interactif afin d'assurer des opérations relatives à la structure d'accès aux fichiers ou à la base de données.

Niveau trois . Nous trouvons ici les éventuels modules de gestion globale de la concurrence. Ces modules s'avèrent parfois nécessaires pour garantir au système un comportement satisfaisant aux contraintes globales de performance spécifiées (cfr. 2.3.).

Niveau deux . Nous plaçons ici certains modules du système d'information (par exemple, les logiciels de communication entre programmes) et du système de gestion de fichiers ou base de données; ces modules ne sont visibles aux niveaux supérieurs que par des interfaces bien définis.

Niveau un . Nous trouvons ici les modules du système d'exploitation (allocation de ressources, gestion des programmes, ...), ceux-ci ne nous concernent pas directement ici.

Le programmeur d'application n'est concerné que par les trois niveaux supérieurs de la structure et n'a connaissance des modules de niveau inférieur que par le biais de procédures avec des paramètres.

C'est à l'étude détaillée de ces modules supérieurs que nous allons maintenant nous consacrer.

§I. Identification des modules fonctionnels dans l'arbre déductif.

Au niveau des traitements, nous avons suivi une démarche descendante dans l'analyse du problème. C'est ainsi que pour expliciter le problème initial, nous avons été amenés à introduire des sous-problèmes à résoudre et ainsi de suite jusqu'au moment où les sous-problèmes correspondaient à des fonctions de "base" sur les données. On a ainsi construit un arbre des traitements à réaliser (appelé arbre déductif) sur lequel se sont greffés des traitements propres à l'organisation.

En vue d'identifier un certain nombre de modules fonctionnels, on pourrait choisir la solution immédiate consistant à faire correspondre un tel module à chacune des fonctions de base. Ce choix risque en général d'être

inapproprié car on aurait alors :

- une taille de module très réduite, donc une faible cohésion interne (degré d'interdépendance faible au sein du module).
- un couplage important entre modules,c'est-à-dire un degré d'interdépendance fort entre traitements de modules différents.
- de la redondance dans l'exécution de certains traitements et dans l'évaluation des prédicats.

Les critères qui nous guideront dans cette modularisation seront les suivants :

I) Ces modules doivent refléter, à des fins de transparence et d'adaptibilité, les pavés de traitement décrits. C'est pour cela que nous opterons pour une structuration des modules fonctionnels sous forme de processus communicants (cf. (HOA,78)).

2) Chaque module "cachera " un maximum d'information aux autres modules (mode de réalisation d'un traitement,...).

3) Chaque module devra être caractérisé par un faible couplage et une forte cohésion interne (cf. définitions ci-dessus).

4) Dans le cadre d'une application temps réel, afin d'obtenir un degré de parallélisme satisfaisant, il y a lieu d'assurer un grain d'atomicité le plus fin possible en définissant des modules contenant le moins de traitements possibles hormis les sections réellement critiques.

Pour assurer le respect des conditions énoncées ci-dessus, il s'agira de tenir compte des éléments suivants dans notre identification des modules au sein de l'arborescence déductive :

I) Fusion des pavés de traitements identiques (ou très semblables) portant sur des objets différents.

Il est à remarquer que la fusion de tels traitements aurait déjà pu se faire dès la spécification en introduisant des "pavés paramétrés". L'élimination prématurée des redondances aurait cependant conduit à

une compréhension moins claire et à une maintenance plus difficile (il suffit par exemple de considérer le cas où le traitement sur un des objets paramétrés change).

2) Ne pas choisir comme module fonctionnel un pavé de traitement A de niveau i s'il existe un "sous-pavé" B de niveau j ($j > i$), appartenant à l'arborescence dont A est la racine, qui présente une interaction avec l'environnement manuel du système automatique;

par interaction, on entend :

- soit que le traitement inhérent au pavé B est réalisé par un processeur humain;
- soit que la garde du pavé B est évaluée manuellement;
- soit que le pavé B est un pavé sur lequel est greffé un pavé d'interface (saisie ou diffusion de données).

3) Regrouper en un module les pavés AI, A2, ... ,An se trouvant dans différents chemins de l'arborescence et présentant les caractéristiques suivantes :

a) arguments (AI) = arguments (A2)= ... = arguments (An).

b) garde (AI) U garde (A2) U ... U garde (An) = vrai avec garde (Aj) $\neq$ vrai $\forall$ j (garde étant une fonction booléenne)

Exemple.

Soit une fonction de construction d'une ligne de commande correcte dont la garde est ligne-correcte et une fonction de construction d'une ligne de commande incorrecte dont la garde est non ligne-correcte.

Ces deux traitements peuvent être fusionnés car :

- les arguments des deux pavés sont une ligne de commande;
- la réunion des gardes donne assurément un résultat vrai car une des gardes est la négation de l'autre.

En fait, ce critère a pour but d'éviter les redondances dans l'évaluation des gardes en plusieurs endroits différents ou encore d'éviter l'introduction d'un module d'aiguillage vers les différents pavés Aj suivant

évaluation des gardes (Aj) respectifs. Nous pourrions définir de la sorte un module à bonne cohésion interne, dans la mesure où le degré d'interdépendance entre conditions et actions au sein du module est élevé.

Dans d'autres cas, ce critère pourrait nous conduire à un module de taille trop élevée; nous ne l'appliquerons dès lors que dans les situations où il produit un module à bonne cohésion interne et à faible dimension après d'éventuelles optimisations (cf. 2.4.2).

4) Mise en évidence de pavés dont les fonctions présentent des interférences mutuelles décrites dans les spécifications des performances globales et qui requièrent dès lors la mise en oeuvre de mécanismes d'exclusion mutuelle et de synchronisation. De tels modules pourraient être des parties d'autres modules (A est partie de B si la description de A figure dans la description de B). Cette séparation permet de mettre en évidence les parties critiques du point de vue de la concurrence.

5) Dans la mesure du possible, afin de limiter le niveau d'imbrication de modules, nous regrouperons des pavés de manière à n'avoir qu'une entrée et qu'une sortie par module : celles-ci correspondant respectivement à des arguments et résultats externes n'appartenant chacun qu'à un type d'objet (arguments du problème, intermédiaires introduits, résultats du problème).

Pour chacun de ces modules, nous exprimerons leur caractérisation à l'aide de spécifications qui prendront la forme d'assertions dans le style de (HOA,69) et (FLO,67).

La spécification d'un module est ainsi donnée par une paire de prédicats :

- une précondition exprimant la condition d'applicabilité du module;
- une postcondition caractérisant l'effet d'une application du module.

Ces pré/post conditions d'un module fonctionnel seront facilement constructibles à partir des pré/post caractérisant chacune des fonctions de base regroupées dans ce module fonctionnel. Rappelons que les pré/post

de chacune des fonctions de base sont données au niveau de la description du type d'objet sur lequel elle porte (cf. description du pavé de données dans le chapitre 2.I.2, §3).

En particulier, pour les modules qui présentent des interférences mutuelles, la spécification de cette concurrence peut se faire à l'aide de la notation employée par (LAM,78b). Nous allons ainsi exprimer que deux modules A et B sont atomiques et mutuellement exclusifs de la manière suivante : ⟨A⟩ ⟨B⟩

Le fait que ces modules peuvent être cycliques par :

REPEAT ⟨A⟩ , REPEAT ⟨B⟩

La concurrence entre modules cycliques est exprimée par une synchronisation au moyen de conditions:

REPEAT ⟨ B1 ----→ A ⟩ , REPEAT ⟨ B2 -----→ B ⟩

où les transitions ⟨ condition ----→ module ⟩ peuvent être interprétés comme étant " ⟨ attendez jusqu'au moment où la condition est vraie ⟩ "

De telles spécifications présentent l'avantage de caractériser un module sans donner d'algorithme, donc, de bien mettre l'accent sur le "quoi" avant le "comment"; en outre, elles portent sur les objets formels manipulés par l'algorithme à développer, se prêtant ainsi à d'éventuelles techniques de dérivation de programmes corrects par construction (DIJ,76). Cette spécification des modules fonctionnels devra éventuellement être transformée en cas d'optimisations ultérieures tenant compte de l'élimination de la redondance et de l'utilisation de moyens techniques de réalisation particuliers (ordinateur centralisé, réseau de micro-ordinateurs, gestionnaire de S.G.B.D., ...).

§2. Identification des modules interactifs.

Dans le chapitre 2.2.I. (§3), nous avons décrit des pavés "d'interface" gérant la saisie et la diffusion de données, éventuellement, à l'aide

de traitements additionnels spécifiques. Lors de l'identification des modules fonctionnels par regroupement de pavés, un de nos critères a été d'écarter les pavés de traitements sur lesquels étaient greffés des pavés d'interface.

Pour chacun de ces pavés d'interface, nous donnerons une description :

- des formats des données à saisir ou à éditer (cette description pourra éventuellement être graphique),
- des différents contrôles de validation auxquels les données doivent souscrire,
- sous forme pré/post, des différents traitements spécifiques. Cette spécification, comme celle des modules fonctionnels, pourra être transformée ultérieurement lors des optimisations réalisables.

§ 3. Identification des modules d'entrée-sortie et d'accès.

Les modules décrits jusqu'ici reflètent des décisions prises au niveau fonctionnel. A côté de celles-ci, il y a des décisions concernant la nature du matériel et du logiciel permettant la saisie ou la diffusion de données ainsi que l'accès aux données permanentes (arguments ou résultats internes).

Pour bien mettre en évidence et localiser de tels choix, la meilleure solution est d' "enfermer" ces décisions dans des modules spécifiques qui sont de deux types :

- modules d'entrée-sortie. Ces modules vont gérer l'acquisition et la diffusion de données en provenance ou à destination de l'environnement du système automatique (par exemple, gestion des écrans, édition,...); ils peuvent en outre assurer un ensemble de contrôles élémentaires de validation des données traitées.
- modules d'accès. Ces modules vont gérer tous les accès (consultation, création, mise à jour, ...) à des données permanentes figurant dans des documentations (fichiers, base de données, ...).

Le fait d'isoler de tels choix dans des modules distincts permet d'assurer une plus grande portabilité au système automatique, ainsi qu'une maintenance plus aisée, en localisant au maximum les dépendances par rapport aux supports logiciels et matériels existants.
Concernant les modules d'accès, il faut remarquer cependant que leur identificateur dans une structuration du système n'est pas toujours rendue possible. En effet, certains logiciels de communication peu performants (lenteur, taille réduite des messages, non-structuration des données du message) rendent impossible l'utilisation de tels modules.

§ 4. Identification des modules de gestion de la concurrence.

Lors des spécifications des contraintes globales de performance du système, nous avons été amenés à formuler les possibilités de parallélisme, d'exclusion mutuelle entre les différents pavés de traitement ainsi que d'éventuelles règles de concurrence additionnelles (priorité, préemption, ...). L'implémentation de ces contraintes peut se faire :

- au sein des modules correspondant aux pavés de traitement (cfr.§2.);
- à l'aide des utilitaires existants (S.G.B.D., ...) qui correspondent aux modules de niveau 2;
- au sein des modules d'accès;
- à l'aide de modules spécifiques garantissant au système un comportement dynamique qui est celui attendu.

C'est cette dernière catégorie de modules que nous appelons "de gestion de la concurrence". Un exemple d'un module pouvant appartenir à cette catégorie est constitué par le moniteur (au sens de (HOA,74)).
Un moniteur est l'association d'une structure locale de données, de procédures et fonctions d'accès ou de modifications de ces données, qui sont appelées de l'extérieur par des programmes. Les (fonctions) procédures ont pour but d'accèder ou de modifier les données locales depuis l'extérieur. L'exécution des appels de ces (fonctions) procédures est faite en exclusion

mutuelle. Les moniteurs sont utilisés pour décrire des allocateurs de ressources et pour exprimer la synchronisation de processus parallèles. Pour ce faire, ils utilisent un mécanisme de condition. Une condition est une file d'attente déclarée à l'intérieur d'un moniteur dans laquelle on peut bloquer des processus. Nous verrons dans le troisième chapitre (cfr. 3.4. I.2.) un exemple d'utilisation du moniteur.

2.4.2. Simplification des données et des modules de traitement par unification.

Jusqu'ici, les spécifications et l'identification des modules fonctionnels et interactifs ainsi que la dérivation du modèle binaire ont reflété uniquement des choix de nature fonctionnelle concernant la structure des traitements et des données. A ce stade, on peut constater qu'il existe au niveau des données et des traitements une certaine redondance. Les sources de celle-ci peuvent être :

- au niveau fonctionnel. En effet, l'introduction d'objets équivalents redondants peut être inhérente à la méthode déductive;
- au niveau organique. Il arrive un moment où la connaissance des outils techniques disponibles permet de faire des choix en vue d'éliminer certaines redondances et de réduire la complexité des modules.

L'élimination de cette redondance se fera grâce à un processus d'unification qui sera guidé essentiellement par l'analyse de la structure des données.

2.4.2.I. Unification des données.

Concernant la structure de données, nous avons, dans le paragraphe 2.4.I.I., assuré le passage entre la description des types abstraits et une description équivalente à l'aide du formalisme du modèle relationnel binaire.
Si nous avions n structures de types abstraits (correspondant aux structures des arguments et résultats du problème, des intermédiaires et des documentations internes), nous avons, après la dérivation du modèle, n structures binaires.

A ce stade, il faut remarquer que certaines de ces structures binaires ne sont plus pertinentes à conserver comme telles et qu'elles peuvent être remplacées par d'autres jugées sémantiquement et syntaxiquement équivalentes.

Exemple. On peut ainsi "unifier" des structures de données caractérisées par les mêmes fonctions de base, les mêmes invariants, les mêmes clauses d'interférence et des fonctions regroupées dans les mêmes modules et fournissant les mêmes résultats.

Ce processus d'unfication de données, comme nous le verrons plus tard (cfr. 2.4.2.2.) est fortement imbriqué avec celui d'unification des traitements.

Cette première unification visant à éliminer des structures de données sémantiquement et syntaxiquement équivalentes (redondantes) est faite d'une manière totalement indépendante des outils logiciels à notre disposition. De ce fait, l'analyse faite jusqu'à présent est valable quelque soit le type de configuration, aussi bien centralisée que répartie. Cependant, il arrive un moment où la connaissance des outils techniques disponibles permet de faire de nouveaux choix en vue d'éliminer certaines redondances et de réduire la complexité des modules. Ainsi, en ce qui concerne les structures de données et plus particulièrement les documentations, des unifications peuvent se faire d'après les outils de support d'information disponibles.

Exemple.

Supposons que nous ayons une documentation sur les commandes dont les lignes portent sur des produits épuisés et une documentation sur les commandes dont les lignes portent sur des produits en différé telles qu'on trouve, dans les deux documentations, des renseignements communs; supposons que l'on dispose d'un outil de gestion intégrée et centralisée de fichiers (base de données). On pourrait alors, par des transformations sur les structures de données des documentations, éliminer toute redon-

dance en unifiant certaines données sémantiquement et syntaxiquement équivalentes.

Remarque. Lorsqu'on parle ici de documentations, il s'agit d'une part des documentations internes identifiées lors de l'analyse fonctionnelle (arguments et résultats internes) et d'autre part de messages abstraits (arguments et résultats externes) que nous sommes amenés à mémoriser. Nous reviendrons sur cette notion dans le paragraphe qui suit (cfr. 2.4.2.2.).

Nous n'allons évidemment pas entrer ici dans le détail des unifications possibles selon un contexte matériel et logiciel particulier; nous en verrons un certain nombre dans l'exemple traité dans la troisième partie. Contentons-nous ici d'en montrer la logique ainsi que les transformations qui les rendront possibles.

Entre les différentes structures de données qu'il nous faut mémoriser, on pourra établir certaines relations entre elles d'après les traitements qui leur sont associés.

Exemple.

Si nous avons dans un pavé de fonction la spécification suivante (dont la sémantique est expliquée dans le paragraphe 2.I.2.) :

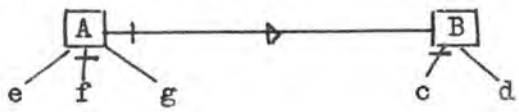
prédicat (b) et a = fonct (b)

pour b ∈ B

il existe un ou plusieurs a∈A

si les structures de données de A et de B sont les suivantes :

Alors nous pourrons créer une relation "fonct" entre les données A et B :



où

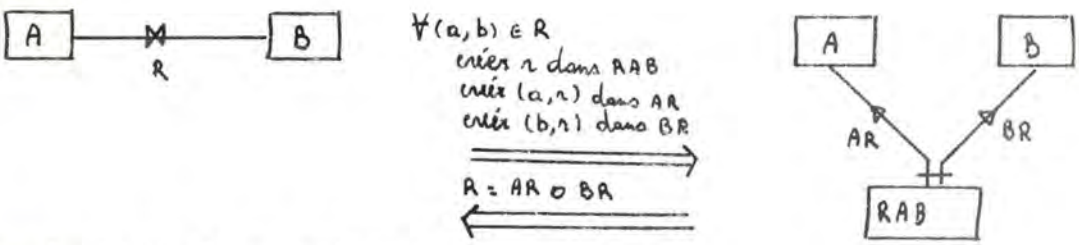
- la relation "fonct" est forte pour le domaine entité A car chaque entité A intervient dans un tuple de la relation "fonct";

- la relation "fonct" est faible pour le domaine entité B car chaque entité B n'intervient pas nécessairement dans un "tuple" de la relation "fonct" (car la fonction "fonct" est soumise à la réalisation d'un prédicat);
- la connectivité de la relation "fonct" reflète le fait que, pour une occurrence du domaine entité B, peut correspondre plusieurs occurrences du domaine entité A).

A partir des différentes relations que nous aurons pu ainsi dégager entre les différents types, nous allons appliquer différentes transformations rendues possibles par l'utilisation du modèle relationnel binaire, dans le but d'éliminer un certain nombre de redondances existantes tout en rendant notre modèle binaire compatible avec le système de gestion des données à notre disposition. On trouvera pour ces transformations une description précise dans (HAI,8Ia); mentionnons-en cependant deux que nous utiliserons couramment :

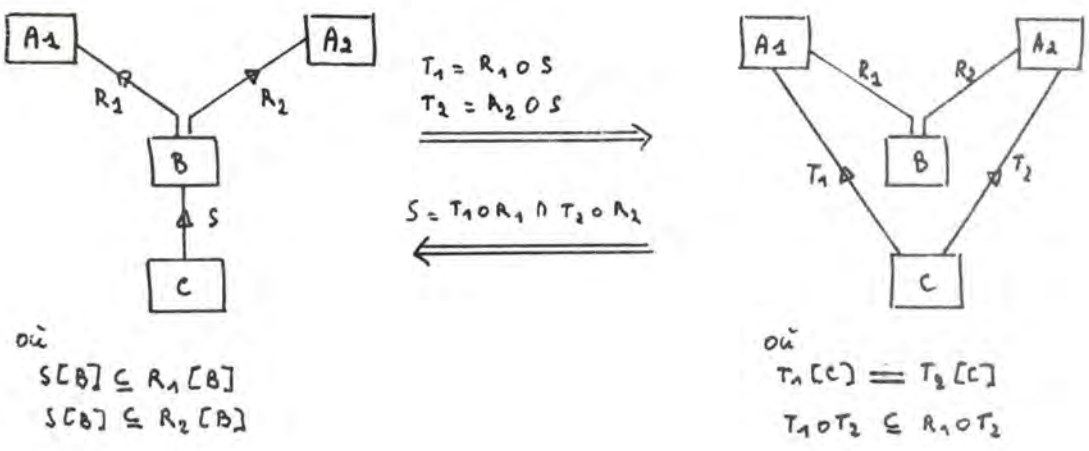
a) l'introduction ou la suppression d'un domaine entité.

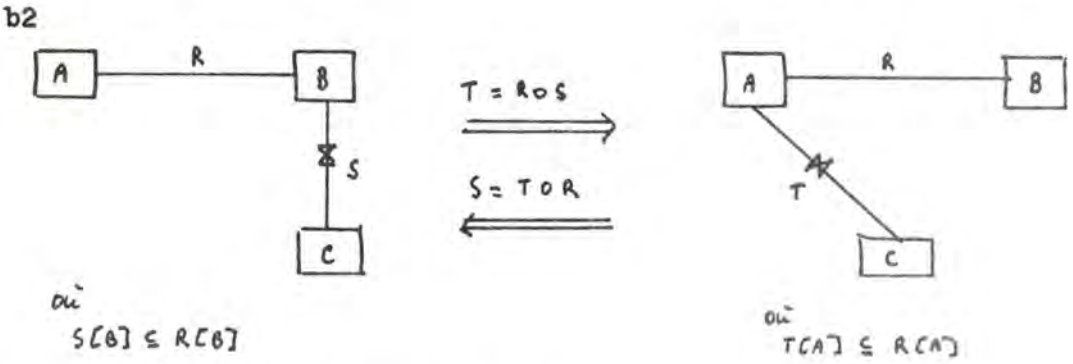
Exemple.



b) la migration d'une relation

bI





2.4.2.2. Unification des traitements.

Jusqu'ici, les modules fonctionnels et d'interfaces ont été structurés sous la forme de processus communicants (HOA,78). Leurs interfaces étaient faits de messages implémentant les messages abstraits de l'analyse fonctionnelle (arguments et résultats externes). Par la suite et parallèlement à l'unification des données, les modifications de ce modèle vont être les suivantes :

- après la première unification concernant l'élimination de structures de données redondantes inhérentes à la spécification déductive, nous sommes amenés à une unification des traitements portant sur les structures de données unifiées. La conséquence est donc la disparition, dans notre modèle initial,de processus communicants, de modèles fonctionnels et d'interface dont les traitements sont redondants avec d'autres.
- la seconde unification de données concernait les structures de données qui nécessitaient une mémorisation. Ces structures sont

+ celles correspondant, dans l'analyse fonctionnelle, aux documentations internes (arguments et résultats internes).

+ celles correspondant, dans l'analyse fonctionnelle, à certains messages abstraits (arguments et résultats externes). En effet, de tels messages (que nous avions considérés jusqu'ici comme étant les messages circulant entre les processus communicants) ne sont pas toujours compatibles avec des logiciels de communication particuliers (nous en verrons un exemple dans l'application traitée au troisième chapitre) et il est préférable que ces messages se restreignent,dans la mesure du possible,

à des données élémentaires identifiantes des messages abstraits qui, eux, seront mémorisés temporairement.

La nature de ces nouvelles informations à mémoriser est différente de celles déjà décrites (documentations internes: arguments et résultats internes). En effet, ces dernières doivent persister entre l'exécution de plusieurs modules ou entre plusieurs activations du même, tandis que les messages abstraits mémorisés ont une durée de vie limitée au moment où un module les crée et un autre les exploite; chacun est donc écrit et lu une seule fois.

Du fait des transformations de la notion de message entre modules et de l'élimination de la redondance au niveau de l'information mémorisée, nous sommes amenés à réécrire les spécifications de nos différents modules, cette fois dans l'optique de l'utilisation de logiciels bien spécifiques.

2.4.3. Choix d'une décomposition physique des traitements.

Une fois élaborée notre structuration et notre spécification des différents modules, il s'agit de définir une identité "physique" pour chaque module : unité d'exécution indépendante (programme), unité de compilation indépendante (sous-programme), ... Ces choix qui doivent satisfaire aux exigences de performance de l'utlisateur (en particulier dans les sytèmes temps réel) sont fortement liés à l'environnement logiciel disponible.

A titre d'exemple, nous pouvons citer les critères suivants :

- faire en sorte qu'il y ait un maximum de parallélisme possible, ce qui est fondamental dans une application temps réel;
- ne pas regrouper des modules dont l'un interfère avec d'autres dans d'autres programmes, on risquerait alors de bloquer le déroulement de tout un programme uniquement à cause d'un module;
- ne pas séparer un module en plusieurs programmes ou ne pas regrouper des modules effectuant des tâches tout-à-fait différentes ou asynchrones aux autres, cela ferait perdre l'interdépendance fonctionnelle au sein

d'un programme;

- permettre à l'utilisateur de contrôler sa transaction jusqu'au moment où elle est validée, c'est-à-dire isoler les modules de saisie et de contrôle de validité des données.

2.4.4. Dérivation d'un premier modèle d'accès aux données.

A partir du modèle relationnel binaire représentant l'information à mémoriser, nous pouvons dériver d'abord une première structure d'accès possible. Cette première structure vise la clarté et la simplicité tant du schéma que des algorithmes que nous décrirons dans le chapitre 2.5.. Elle ne prendra, par conséquent, pas en compte des critères tels que la performance, la sécurité, la concurrence, ...

Le modèle d'accès va permettre d'exprimer les structures de données dans des termes familiers pour les programmeurs de l'application. Les données seront décrites en termes de base de données, fichiers, articles, valeurs d'item, identifiant d'un type d'article, clé d'accès (d'item vers articles), chemin d'accès (d'article à articles) ... . On trouvera la spécification exacte de tous ces concepts dans (HAI,79a).

La transformation du modèle conceptuel relationnel binaire en modèle d'accès se fera aisément en suivant les règles suivantes (HAI,79b).

I. A tout type de domaine entité, est associé un type d'article. Chaque entité étant représentée par un article.
2. A tout domaine simple, est associé un item.
3. A toute relation sémantique, sont associées deux relations d'accès équivalentes à la première, inverses l'une de l'autre.
4. Un accès séquentiel est prévu pour tout type d'article (relation du Système vers le type d'article).

Il va de soi que cette première structure d'accès n'est pas définitive. L'élaboration d'une structure des accès utilisés ne pourra se faire qu'après avoir analysé et quantifié les accès requis par les traitements (analyse algorithmique des opérations sur la structure des données).

2.5. Programmation et dérivation du modèle des accès utilisés.

2.5.I. Programmation.

Jusqu'ici, la description des différents modules intervenant dans les programmes s'est faite uniquement en termes non-algorithmiques. Nous nous sommes contentés de poser les sous-problèmes en termes de pré/post conditions. En effet, il était déjà assez difficile de formuler ce que chaque module doit faire que pour encore nous préoccuper de détailler, à l'aide d'un algorithme , comment il le fait. De plus, donner a priori un algorithme pourrait conduire à des spécifications implicites ou entraîner des choix d'implémentation prématurés.

Il s'agit donc maintenant de construire les différents algorithmes réalisant les pré/post conditions que l'on a spécifiées. Nous ne suivrons pas de méthode particulière pour les construire; nous pensons que cet aspect sort du cadre du présent travail.

Pour exprimer les algorithmes, nous utilisons un pseudo-langage reprenant les éléments suivants :

a) les structures algorithmiques classiques de type Algol.

Les constructions de base sont l'affectation, la composition séquentielle, la conditionnelle, l'itération et l'appel de procédure.

On peut donner une définition axiomatique de la sémantique de chacune de ces constructions de base en termes de pré/post conditions (HOA,68); de tels axiomes s'avèrent très utiles si l'on veut valider l'algorithme eu égard à ses spécifications.

b) les structures de données.

On peut utiliser des types simples et des types structurés, par exemple de type Pascal.

c) les opérations sur la base de données.

Il s'agit des opérations d'accès (simple, sélectif, itératif, à partir d'une donnée "origine" vers une donnée "cible" via un certain chemin) et d'opérations de modification (création,suppression, mise à jour).

d) les structures de synchronisation (attendre, signaler la validité d'une condition ou la survenance d'un évènement).

e) les structures de communication (envoi, réception de messages entre programmes).

f) les opérateurs d'entrées/sorties (saisir, éditer des données).

On peut trouver une description de ce langage dans (LAM,80).

L'approche "pseudo-code",que nous suivons ainsi, a comme principaux avantages :

- d'exprimer les traitements à réaliser et leur composition uniquement en termes de structures algorithmiques sans devoir se soucier des caractéristiques particulières d'un langage de programmation : on distingue donc le travail inventif de programmation du travail clérical de codage.

2.5.2. Modèle des accès utilisés.

Avec les algorithmes décrits, nous avons pu mettre en évidence les différents accès et modifications de la base de données qui sont nécessaires dans notre application. D'autre part, grâce à des quantifications (élaborées au stade conceptuel) de l'application, nous pouvons calculer une estimation du nombre d'activations de ces différentes opérations. Nous pouvons dès lors choisir un schéma des accès nécessaires (sous-ensemble du schéma des accès possibles) et présenter un document spécifiant les types d'opérations sur la base de données et leurs quantifications.

2.6. Implémentation.

2.6.1. Implémentation de la structure de données.

A partir du document produit lors de l'étape précédente et des caractéristiques propres au logiciel disponible, nous pouvons proposer quelques implémentations possibles de la structure de données.
Ces différentes implémentations devront être jugées en termes de

- volumes (longueurs et nombre d'occurrences de chaque type de données et de leurs relations),
- temps d'accès,
- degré de concurrence possible entre les différents programmes travaillant sur les mêmes données,
- sécurité et capacité de reprise en cas d'incidents légers ou graves,
- adaptabilité face à des modifications ou des extensions.

Pour l'implémentation retenue, nous passerons au réglage des paramètres caractéristiques des mémoires (taille des pages ,...) et du SGBD (taille des tampons, partage de tampons,...) puis nous évaluerons une dernière fois cette implémentation, quitte à revenir éventuellement à une autre parmi les implémentations possibles si la première s'avère peu performante au niveau "physique".

2.6.2. Implémentation de la structure des traitements.

Il n'est guère difficile de coder de manière systématique, dans le langage de programmation retenu pour l'application, les parties de programme n'ayant pas trait aux opérations sur la base de données; on peut pour cela appliquer des règles de transcription. Il reste donc à assurer la traduction des opérations d'accès et de modification de la base de données en ordres de langage de manipulation de données imposé par le logiciel disponible, puis à intégrer ces ordres dans le code déjà obtenu. De tels ordres peuvent également être obtenus en appliquant des règles systématiques de traduction; un exemple de ces règles, dans le cas

d'un SGBD répondant aux normes CODASYL, est donné dans (HAI,8Ib). Nous verrons ultérieurement, dans l'étude de cas, quelques exemples de codages basés sur de telles règles (cf. chapitre 3.6.).

Chapitre 3.

MISE EN OEUVRE DE LA DEMARCHE SUR UNE APPLICATION DE GESTION DE VENTES PAR CORRESPONDANCE.

Le second chapitre proposait une démarche intégrant la spécification et la réalisation d'un système d'information; nous voulons maintenant juger du caractère effectif de cette démarche en l'utilisant pour développer une application de traitement des commandes clients dans une société de vente par correspondance. Ce troisième chapitre se composera de six paragraphes correspondant aux six étapes de la démarche proposée. Chacun des paragraphes sera illustré par des exemples en provenance de la documentation que nous avons réalisée sur l'application.

Le premier paragraphe est consacré à la spécification descendante de l'application.

Dans le deuxième paragraphe, nous introduisons la description des contraintes particulières liées à l'environnement organisationnel.

Le troisième paragraphe est consacré à la description des contraintes globales de performance attendue par le demandeur.

Le quatrième paragraphe présente une architecture des traitements et un premier modèle d'accès possibles aux données.

Dans le cinquième paragraphe sont présentés les programmes et le modèle des accès nécessaires aux données.

Enfin, le cinquième paragraphe montre comment coder les programmes et la structure de données en fonction des logiciels disponibles.

3.I. Spécification descendante de l'application Petitpas.

Le point de départ de la spécification est constitué par la proposition d'automatisation qui résulte de la phase d'analyse d'opportunité. Un exemplaire de cette proposition se trouve en annexe I. Nous en retiendrons l'application relative à la gestion des commandes clients. Il est à remarquer que la première description fournie par ce genre de proposition est souvent fort incomplète; nous trouvant dans ce cas, nous avons dû faire des choix supplémentaires lors de notre spécification. Un certain nombre de ces choix sont ceux retenus dans les spécifications réalisées à l'Institut d'Informatique de Namur par (BOD,79c). Dans une situation réelle, on aurait dû procéder à un inventaire des choix à opérer et soumettre celui-ci au demandeur.

Au niveau des traitements relatifs aux commandes clients et aux commandes fournisseurs, on peut distinguer deux problèmes ayant chacun une existence quasi-autonome (au sens de (SIM,74)), un flux homogène d'informations, une certaine permanence dans le temps et communiquant entre eux par l'échange d'agrégats : il s'agit d'une part du traitement des commandes clients et d'autre part du traitement des commandes fournisseurs. La nature des interactions entre les deux applications est la suivante :

- du traitement des commandes clients au traitement des commandes fournisseurs. On signale qu'il y a un produit pour lequel la quantité commandée sur une ligne du bon de commande du client est insatisfaite (partiellement ou complètement).
- du traitement des commandes fournisseurs au traitement des commandes clients. On signale, par le passage d'un agrégat, qu'il y a un produit pour lequel on vient d'enregistrer un réapprovisionnement en stock ou qui est décrété "épuisé".

Dans l'application qui nous intéresse (celle du traitement des commandes clients), on peut distinguer des inputs et des outputs bien définis :

- en entrée : le bon de commande du client

- en sortie : - le bon de commande du client jugé erroné ainsi qu'un nouveau bon de commande vierge et un document d'explication;
  - une facture et/ou un justificatif de non-livraison (pour produits différés ou épuisés), figurant tous deux sur le même bordereau.

Au niveau de notre première spécification, nous ne retiendrons, pour pouvoir appliquer la méthode déductive, comme données "résultats" que les outputs qui ont un lien direct, par l'information contenue, avec le bon de commande du client. C'est ainsi que nous distinguerons le bon de commande annoté, la facture et le justificatif de non-livraison.

De la proposition d'automatisation, il ressort directement qu'on doit tenir compte

- d'une part, de la distinction qui existe entre la première livraison et les livraisons différées;
- d'autre part, parmi les produits non-livrés, de la distinction entre ceux qui sont épuisés et ceux qui sont différés.

Pour la donnée "de départ" constituée par un bon de commande du client (notée COM-CLI), on considèrera dès lors comme données "résultats", les objets logiques suivants :

- une facture lors d'une première livraison (BON-FACTURATION);
- une facture lors d'une livraison différée (BON-FACTURATION-DIFFEREE);
- un bon de commande non-valide (COM-CLI-REFUSEE);
- un justificatif de non-première livraison pour produits épuisés (JUSTIFI--CATIF-NON-PREMIERE-LIVRAISON-PRODUITS-EPUISES);
- un justificatif de non-première livraison pour des produits dont la livraison est différée partiellement ou complètement (JUSTIFICATIF-NON-PREMIERE-LIVRAISON-PRODUITS-DIFFERES);
- un justificatif de non-livraison différée pour les produits différés qui restent différés après le réapprovisionnement ou la déclaration "épuisé" d'un produit manquant (JUSTIFICATIF-NON-LIVRAISON-DIFFEREE-

-PRODUITS-DIFFERES);

- un justificatif de non-livraison différée pour un produit différé qui est devenu épuisé (JUSTIFICATIF-NON-LIVRAISON-DIFFEREE-PRODUIT-EPUISE).

Remarque. D'après la politique retenue par l'entreprise, il y a au plus une première livraison tandis qu'un nombre quelconque de livraisons différées est permis. D'autre part, du fait que le message de réapprovisionnement ou de non-réapprovisionnement (épuisement) d'un produit en provenance de l'application de gestion des commandes fournisseurs ne concerne qu'un seul produit à la fois, le BON-FACTURATION-DIFFEREE et le JUSTIFICATIF--NON-LIVRAISON-DIFFEREE-PRODUIT-EPUISE ne porteront jamais que sur un seul produit.

Il reste maintenant à mettre en évidence les données "intermédiaires" entre les données "résultats" et la donnée de "départ". Par une approche déductive, nous pouvons mettre en évidence :

- un bon de commande client qui a été vérifié, éventuellement complété et corrigé, et accepté (COM-CLI-VALIDE) auquel, lors de l'enregistrement, on aura affecté la date du jour et un numéro de commande (COM-CLI-COMPLETEE);
- une partie du bon de commande complétée dont les lignes de commande sont livrables (partiellement ou totalement) dès la première livraison (COM--CLI-COMPLETEE-LIVRABLE);
- une partie du bon de commande complétée dont les lignes de commande portent sur des produits épuisés lors de la première livraison (COM-CLI-COMPLETEE--EPUISEE);
- une partie du bon de commande complétée dont les lignes de commande sont différées (partiellement ou totalement) lors de la première livraison (COM-CLI-COMPLETEE-DIFFEREE);
- une partie d'une commande différée dont une ligne est livrable (partiellement ou totalement) après la rentrée en stock du produit (COM-CLI-COM--PLETEE-DIFFEREE-LIVRABLE);
- une partie d'une commande différée dont une ligne de commande n'est pas

livrable à la suite de l'épuisement d'un produit (COM-CLI-COMPLETEE-
-DIFFEREE-EPUISEE);

- une partie d'une commande différée qui reste différée suite à la production d'une COM-CLI-COMPLETEE-DIFFEREE-LIVRABLE ou d'une COM-CLI-COM-
-PLETEE-DIFFEREE-EPUISEE (COM-CLI-COMPLETEE-DIFFEREE-DIFFEREE).

Remarque. Dans le cas de la gestion des commandes différées, l'objet " commande différée" (COM-CLI-DIFFEREE) représente en fait des lignes de la commande qui sont encore différées après

- une première livraison, ce qui correspond alors à des COM-CLI-COM-
-PLETEE-DIFFEREE;
- une livraison différée, ce qui correspond alors à des COM-CLI-COMPLETEE-
-DIFFEREE.

Comme nous le verrons plus tard, cette commande différée appartiendra en fait à une documentation concernant toutes les commandes se trouvant en différé à un moment donné.

Les liens existant entre les données "résultats", "de départ" et "intermédiaires" sont schématisés dans la figure 8.

Ayant fait un inventaire des données qui constitueront les arguments, les résultats et les intermédiaires externes de chacun des traitements, il s'agit de mettre en évidence les documentations (arguments et résultats internes) regroupant des informations qui doivent persister entre différents traitements non directement consécutifs dans le temps ou entre différentes activations d'un même traitement; nous aurons ainsi :

- la documentation client regroupant tous les renseignements concernant les clients de la firme (DOC.CLIENTS);
- la documentation produit regroupant tous les renseignements concernant les produits commercialisés par la firme (DOC.PRODUITS);
- la documentation comptable regroupant l'ensemble des mouvements comptables inhérents aux différentes factures (DOC.COMPTABLE);
- la documentation sur les parties de commandes clients en attente de

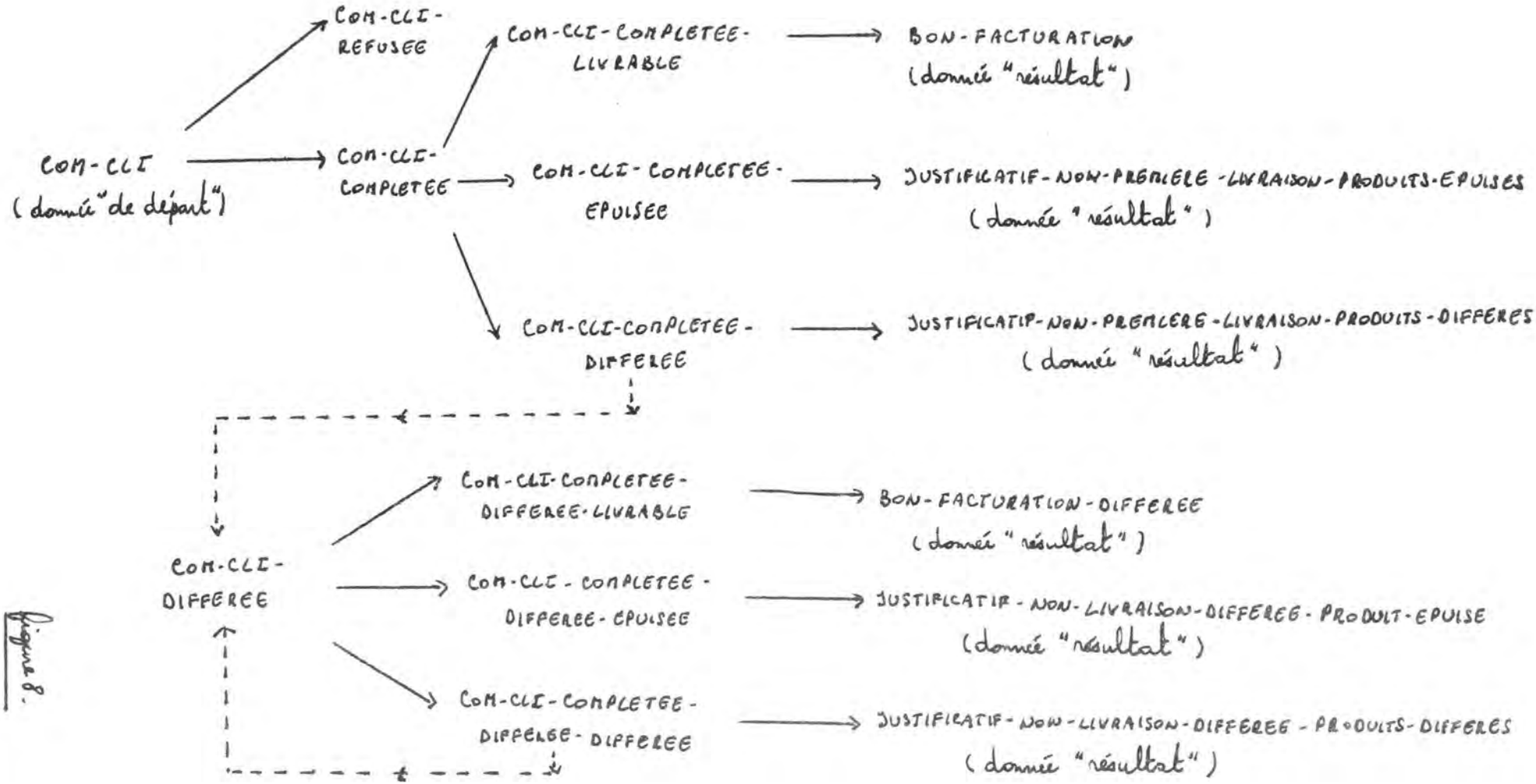

COM-CLI
(donnée "de départ")
COM-CLI-REFUSEE
COM-CLI-COMPLETEE
COM-CLI-COMPLETEE-LIVRABLE
BON-FACTURATION
(donnée "résultat")
COM-CLI-COMPLETEE-EPUISEE
JUSTIFICATIF-NON-PREMIERE-LIVRAISON-PRODUITS-EPUISES
(donnée "résultat")
COM-CLI-COMPLETEE-DIFFEREE
JUSTIFICATIF-NON-PREMIERE-LIVRAISON-PRODUITS-DIFFERES
(donnée "résultat")
COM-CLI-DIFFEREE
COM-CLI-COMPLETEE-DIFFEREE-LIVRABLE
BON-FACTURATION-DIFFEREE
(donnée "résultat")
COM-CLI-COMPLETEE-DIFFEREE-EPUISEE
JUSTIFICATIF-NON-LIVRAISON-DIFFEREE-PRODUIT-EPUISE
(donnée "résultat")
COM-CLI-COMPLETEE-DIFFEREE-DIFFEREE
JUSTIFICATIF-NON-LIVRAISON-DIFFEREE-PRODUITS-DIFFERES
(donnée "résultat")
t

figure 8.

livraisons différées (DOC.COMMANDES-CLIENTS-DIFFEREES);

- la documentation sur la date du jour (DATE).

Ayant fait l'inventaire des différentes données du problèmes que constitueront les résultats, les arguments et les intermédiaires, on peut passer à la spécification des traitements.

Pour cela, nous construisons d'abord l'arbre déductif des différents pavés de traitement obtenus par raffinements successifs ainsi que les relations entre ces différents pavés. Nous utilisons à cet effet le graphisme introduit à la page 59 et qui permet d'exprimer clairement cette spécification.

Après avoir présenté la structure de l'arbre déductif, nous allons donner la spécification de chaque fonction (c'est-à-dire de noeuds dans cet arbre). Parallèlement, nous construisons la spécification de chaque pavé de donnée.

On trouvera, en annexe 2, la représentation de l'arbre déductif de l'application traitée ainsi que des exemples de spécifications de pavés de traitement et de donnée.

3.2. Adaptation de la spécification descendante du cas Petitpas à l'environnement organisationnel.

Nous nous attacherons dans ce paragraphe à complèter la spécification obtenue jusqu'ici, en prenant en compte les différents traitements et données "organisationnelles", puis en considérant pour une première fois les moyens dont nous disposons pour assurer la réalisation des traitements (processeurs "humains" et "matériels").

3.2.I. Traitements et données "organisationnelles".

§I. Spécification des traitements et données manquants.

Une première catégorie de traitements et de données qui nous manquent proviennent de fonctions spécifiques permettant de faire le lien entre les objets logiques considérés jusqu'à maintenant (résultats et arguments du pavé constituant la racine de l'arborescence déductive) et ceux attendus par les demandeurs. Sur la figure 9, nous rappelons l'entrée et les sorties du pavé initial et, pour être complets, nous mentionnons dans un cercle les documentations utilisées (résultats et arguments internes).

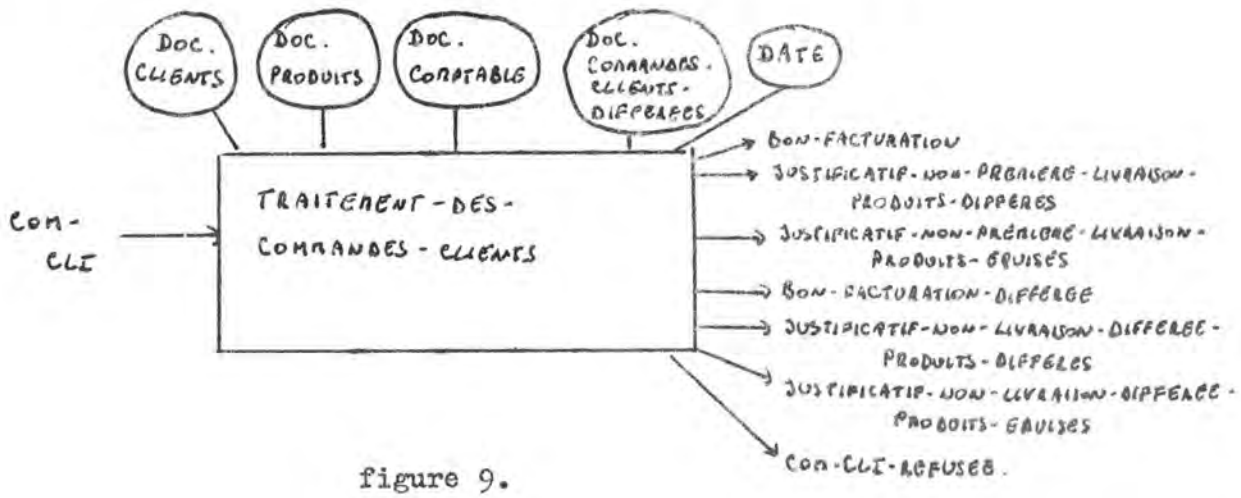


figure 9.

Si on relit la proposition d'automatisation (cfr. annexe I), on se rend compte que :

- en ce qui concerne les commandes erronées (COM-CLI-REFUSEE), il existe un traitement des cas litigieux qui tente de les recycler en COM-CLI et qui, s'il ne le peut, renvoie la commande au client en l'accompagnant d'un

bon de commande vierge et d'une lettre d'explication;

- les justificatifs de produits différés ou épuisés (construits lors d'une première livraison ou d'une livraison différée) ne sont pas envoyés immédiatement mais sont expédiés lors de l'envoi d'une facture (différée ou pas); ceci requiert
  + des traitements d'archivage des justificatifs de produits différés ou épuisés;
  + un traitement postérieur à l'établissement de la facture (différée ou pas), qui associe à ce document les informations relatives aux produits différés ou épuisés.

En tenant compte des nouveaux traitements ainsi que des données (externes ou internes) ainsi introduites, la figure précédente peut être complétée de la manière suivante :

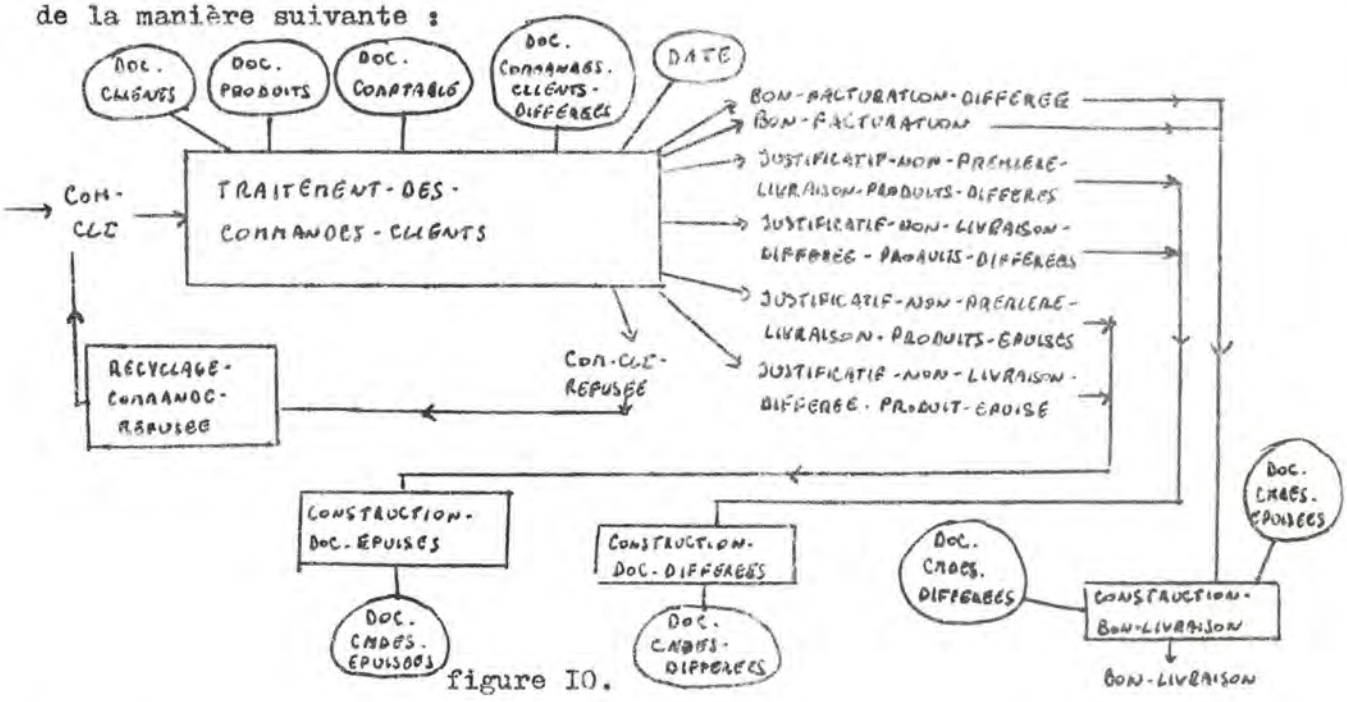


figure IO.

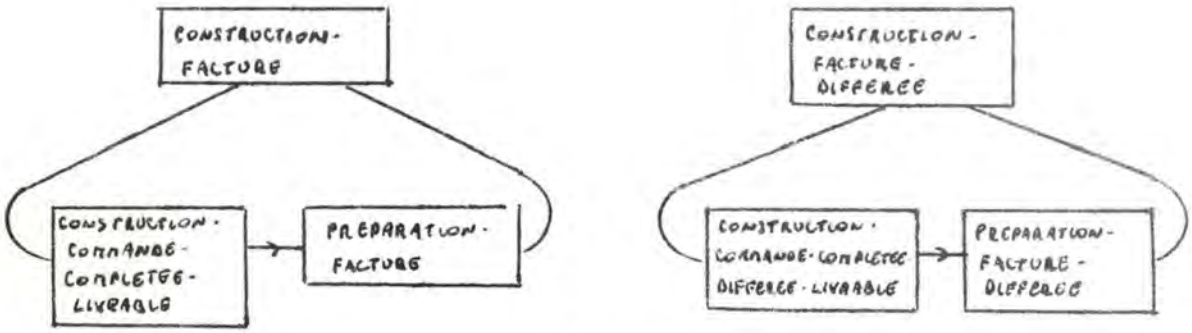
Il est bien entendu que ces nouveaux pavés de traitement et de donnée devront être également spécifiés.

A côté de ces traitements se situant en dehors de l'arbre déductif, il en existe d'autres qui eux sont internes à celui-ci. En effet, il y a des décisions purement organisationnelles propres au système étudié et qui ont été introduites pour des raisons particulières d'efficacité, ou bien qui sont la conséquence d'une structure d'organisation bien spécifique.

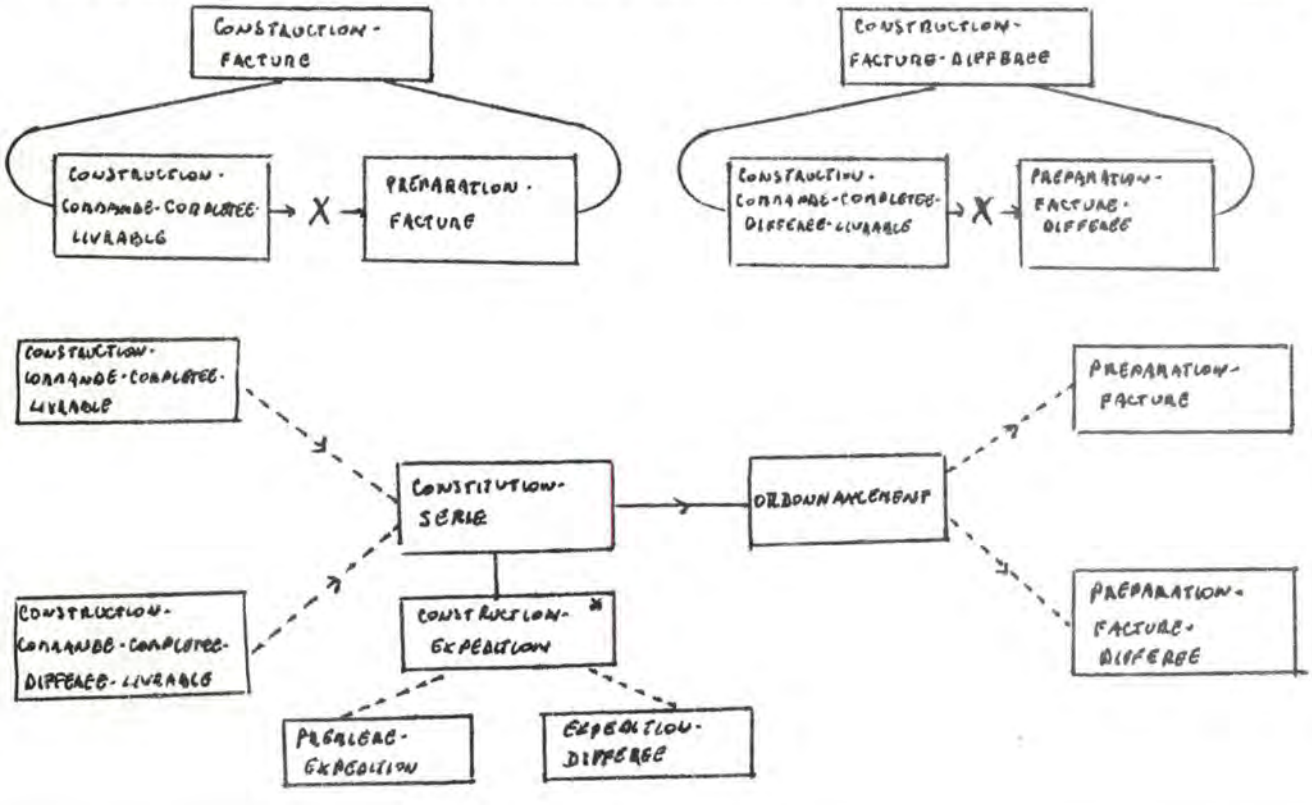
C'est ainsi que dans la spécification du problème traité, il faut introduire les notions relatives :

- à l'ordonnancement. Ce traitement, propre à chaque organisation, consiste à regrouper plusieurs commandes livrables (différées ou pas) et à spécifier le parcours en magasin le plus adéquat pour aller chercher les produits livrables.
- au prélèvement des commandes en magasin. Ce traitement a pour but de tenir compte du fait que le stock physique d'un produit peut être inférieur au stock informatique; si c'est le cas, il s'agit de réajuster la quantité du produit expédié au client, à en différer la différence et à rectifier le stock informatique.

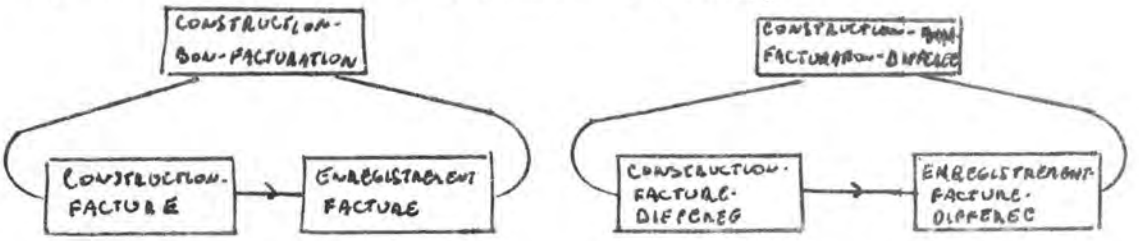
Pour tenir compte des traitements relatifs à l'ordonnancement, l'arborescence déductive qui se présentait de la manière suivante:



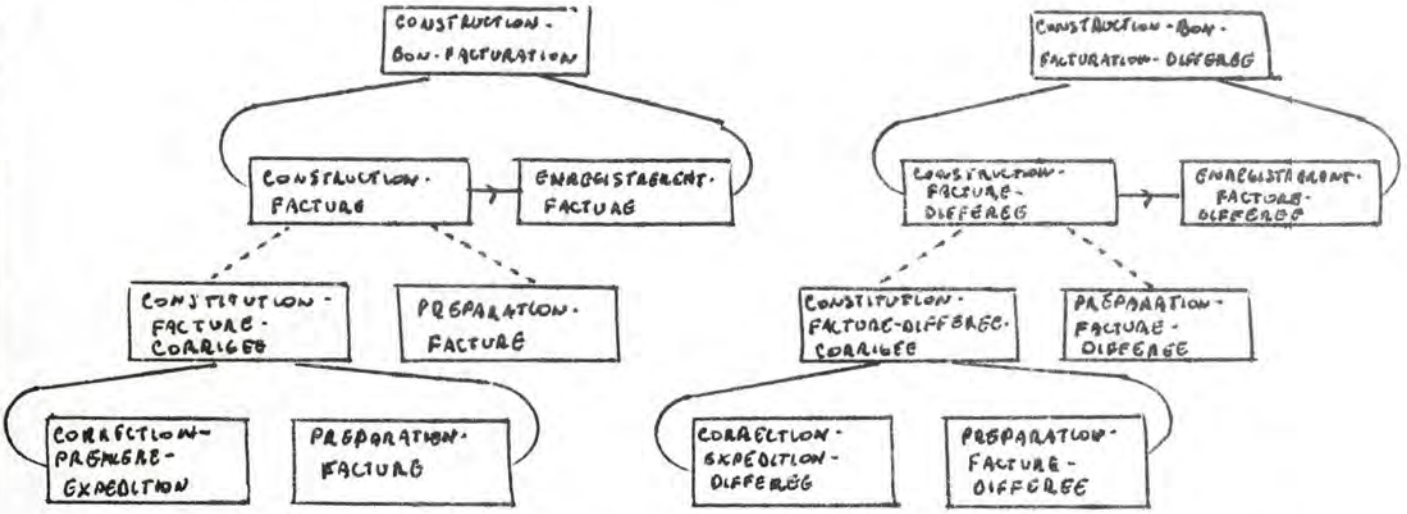
devient selon les transformations expliquées dans le paragraphe 2.2.I.§2 :



Remarquons la différence que nous avons faite concernant la première expédition et l'expédition différée, elle suit la logique qui nous a amenés jusqu'ici à différencier ce qui est différé de ce qui ne l'est pas.
Concernant les traitements relatifs au prélèvement des commandes en magasin, nous modifierons l'arborescenee suivante,



pour en faire,



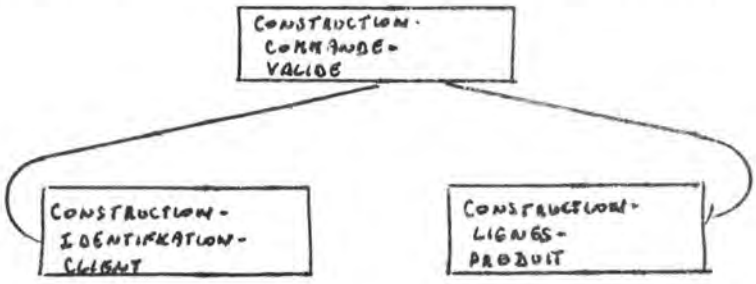
ce qui indique bien que dans certains cas nous pourrons passer à une facturation immédiate, tandis que dans d'autres, il faudra au préalable rectifier la quantité de produit expédiée.
Il va de soi qu'outre la spécification des nouveaux pavés de traitement et de donnée ainsi introduite, il s'agira d'adapter la spécification obtenue de manière déductive en modifiant les relations entre pavés, les arguments et les résultats de certains pavés en amont et en aval de la "greffe" organisationnelle, ...

Remarque. Les nouveaux types de données introduits dans ce complètage de la spécification sont les suivants : LISTE-ORDONNANCEMENT, BONS-CASIERS, BON-PAR-SERIE, DOC-EXPEDITIONS, EXPEDITION-PREMIERE, EXPEDITION-DIFFEREE.

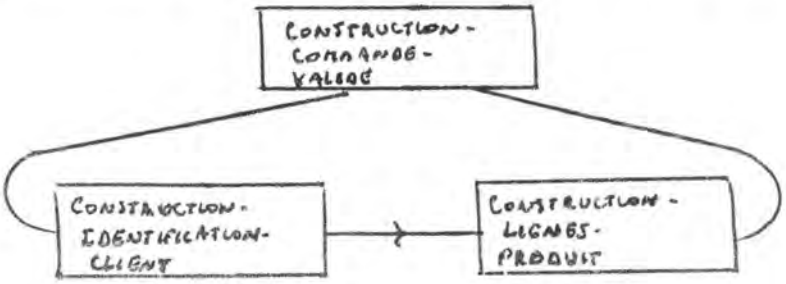
§2. Spécification des interrelations organisationnelles.

Jusqu'ici, les interrelations décrites entre les pavés de traitement et de donnée étaient uniquement dictées par des dépendances entre les données. Cependant, pour rendre la spécification cohérente avec l'organisation, il faut éventuellement tenir compte d'enchaînements requis par celle-ci. Reprenons, à titre d'exemple, les traitements inhérents à la vérification du bon de commande tels qu'ils sont décrits dans la proposition d'automatisation : on se rend compte que les traitements liés à la vérification de la signature, de l'identification du client et du corps de la commande sont exécutés séquentiellement dans cet ordre et que, dès qu'une vérification ne peut se faire, on abandonne les contrôles suivants. Par exemple, si le bon de commande n'est pas signé, il n'est pas nécessaire de faire les autres vérifications. De telles contraintes de séquencement dues au fait que ces contrôles sont faits par le même individu entraînent les surspécifications suivantes au niveau des enchaînements relatifs à la construction de la commande valide.

A la place d'un indéterminisme entre les traitements suivants:



on aura un enchaînement reflétant la décision organisationnelle de vérifier d'abord l'identification du client avant le corps de la commande :



De même, pour la "construction-commande-client-refusée", il s'agit de remplacer l'indéterminisme par une sélection déterministe pour prévenir l'abandon du traitement de contrôle dès la découverte d'une erreur.

§3. Spécification des traitements et pavés d'interface.

Il s'agit maintenant de mettre en évidence les pavés d'interface et les éventuels traitements qui assureront le contact entre l'environnement organisationnel et la machine ou entre les différentes parties de l'environnement organisationnel.

Ces pavés d'interface seront relatifs :

- à la saisie de données constituant les arguments d'un pavé existant;
- à la diffussion de données constituées par les résultats d'un pavé existant.

Par exemple, toujours dans le cadre de la vérification de la commande d'un client, l'opératrice est amenée à introduire les informations d'un client au terminal en vue de leur acquisition et de leur contrôle. Ces informations constituent l'argument externe des pavés de traitement "construction-identification-client" et de "construction-erreur-identification-client". Il nous faudra par conséquent décrire, pour ces deux pavés de traitement, un pavé d'interface de saisie de données dont la description informelle sera :

Pavé d'acquisition des données relatives au client :

"L'opératrice introduit au terminal l'identification du client (nom, prénom, adresse et numéro) figurant sur le bon de commande qu'elle traite."

Toujours en rapport à ces deux pavés, nous aurons deux pavés distincts de diffusion de données. Ainsi, pour le pavé de "construction-erreur-idenfication-client", on aura :

Pavé de diffusion de "erreur-identification-client" :

"L'opératrice inscrit, sur le bon de commande qu'elle traite, le motif d'erreur concernant l'identification du client apparue au terminal."

Quant au pavé de diffusion de données associé au pavé "construction-identification-client, il ne décrit pas uniquement la procédure de diffusion liée à l'"identification-client" mais il doit spécifierune fonction supplémentaire. En effet, le résultat "identification-client" se compose du nom, prénom, adresse du client identifié; ces renseignements suffisent dans le cadre des bordereaux que l'on enverra au client, mais par contre, ils ne seront pas suffisants pour l'opératrice car celle-ci désire en plus connaître le numéro identifiant du client. Il faudra donc décrire une fonction supplémentaire qui fournira le numéro du client.

Pavé de diffusion de "identification-client" :

i) "L'opératrice vérifiera la concordance entre les nom, prénom, adresse et numéro apparus au terminal et ceux figurant sur le bon de commande. En cas de non-concordance, elle barrera les zones non-identiques sur le bon de commande et les remplacera par celles apparues au terminal."

ii) fonction : recherche-numero-client.

fonction associant à l'identification d'un client comme son numéro figurant dans la documentation clients.	RECHERCHE-NUMERO-CLIENT : NOM-CLI-CON-VALIDE x PRENOM-CLI-CON-VALIDE x ADRESSE-CLI-CON-VALIDE x DOC.CLIENTS → NUMERO-CLIENT	(numéro-client) = RECHERCHE-NUMERO-CLIENT (nom-client-valide, prénom-client-valide, adresse-client-valide, doc-cli)

Nous trouverons de semblables pavés d'interface de saisie de données pour les pavés de traitement suivants : construction-erreur-signature et signature et montant, construction-ligne-produit et construction-erreur-ligne, recyclage-commande-refusée et construction-explicatif-refus, correction-expédition-différée et preparation-facture-différée, correction-première-expédition et préparation-facture.

Une interface de diffusion de données de même type sera nécessaire pour chacun des pavés de traitement suivants : construction-erreur-signature,

signature, construction-erreur-ligne et construction-ligne-produit (avec ici une fonction supplémentaire relative au calcul du montant pour une ligne de commande), préparation-facture et préparation-facture-différée, ordonnancement.

Cette approche qui nous permet de mettre en évidence les pavés d'interface nous paraît très importante. En effet, d'une part, elle permet de mieux structurer les traitements en évitant de mêler des descriptions qui ne doivent pas se trouver sur le même pied et, d'autre part, elle amène le spécificateur à consacrer plus d'attention à la description des interactions entre le système et son environnement organisationnel, ce qui souvent n'est pas spécifié très clairement.

3.I.2. Distinction entre les systèmes d'information manuels et automatiques.

Pour compléter la description du S.I., il s'agit maintenant de décrire les différents processeurs (moyens actifs de traitement) et de les affecter aux différentes tâches à réaliser. Cette prise en compte des moyens capables d'effectuer les traitements et fournis par l'organisation va nous permettre :

- de distinguer la partie automatique de la partie manuelle du S.I., ce qui est fondamental par la suite pour délimiter la tâche de l'informaticien;
- de distinguer, dans la partie manuelle, les différents services et catégories de personnel affectés aux différentes tâches et ainsi de faire le lien avec les différents composants de l'organisation.

Cette présentation des différents processeurs sera faite :

- pour chaque feuille de l'arbre ainsi que pour les traitements introduits lors de la spécification des éléments organisationnels manquants;
- pour chaque feuille de l'arbre des prédicats;
- pour chaque pavé d'interface.

Exemples. Pour le pavé "construction-identification-client", nous indiquerons que :

- le traitement est assuré automatiquement;
- le prédicat est évalué automatiquement.

Pour les pavés d'interface de saisie et diffusion de données relatifs à ce pavé de traitement, nous indiquerons que :

- le pavé d'acquisition des données relatives au client est assuré par l'opératrice du service d'enregistrement et de contrôle des bons de commande des clients;
- pour le pavé de diffusion de "identification-client", la fonction de "recherche-numero-client" est automatique tandis que le reste du traitement est assuré par l'opératrice du service d'enregistrement et de contrôle des bons de commande des clients.

Remarque.

A partir de l'ensemble des documents que nous avons réalisés jusqu'ici, il est possible de constituer un graphe de circulation. Si celui-ci ne permet pas d'exprimer tous les détails, il présente cependant l'avantage de fournir une vue synthétique et claire aux utilisateurs.

Ainsi, la méthode CORIG (MAL,7I) permet de décrire la circulation d'informations par des "procédures" définies comme des réponses de l'organisation à des événements. Une procédure est représentée par un schéma qui met en évidence des "interventions" (rectangles) accomplies par différents "agents d'activité", elles sont liées par un cheminement d'information (flèches) ou par l'intermédiaire de "documentations" manuelles (boîtes) ou automatiques (cercles).

A partir des spécifications, nous pourrons identifier :

- les "interventions" qui sont constituées par les pavés de traitement et d'interface;
- les"agents d'activités" qui sont les processeurs décrits ci-dessus;
- le cheminement d'information qui correspond à la transmission des arguments et résultats externes;
- les documentations constituées des arguments et résultats internes.

3.3. Spécification des contraintes globales de performance.

L'étude d'opportunité réalisée propose, pour résoudre le problème de l'entreprise Petitpas, une solution informatisée basée sur un système centralisé, à savoir un mini-ordinateur central auquel seront reliés des terminaux répartis en différents endroits pour permettre une informatique transactionnelle.

Un tel système temps réel implique, pour garder des performances acceptables, la possibilité d'exécution des traitements en parallèle.

Ce parallélisme est cependant limité en ce qui concerne la gestion des stocks; en effet, les traitements de mise à jour des stocks sont mutuellement exclusifs et il y a, en outre, des règles de priorité à respecter.

La mutuelle exclusion entre les fonctions de mise à jour des produits peut être spécifiée au niveau du pavé de donnée DOC.PRODUITS. Elle s'exprime au niveau des fonctions de base de la manière suivante :

- MAJ-QUANTITE-PRODUIT-LIVRAISON (MAJ moins première livraison)
- MAJ-QUANTITE-PRODUIT-LIVRAISON-DIFFEREE (MAJ moins $_{\text{differee}}^{\text{livraison}}$)
- MAJ-RECTIFICATIVE-QUANTITE-PRODUIT-EXPEDIEE (MAJ moins rectificative)

La règle de concurrence additionnelle est la suivante :

la fonction MAJ-RECTIFICATIVE-QUANTITE-PRODUIT-EXPEDIEE est prioritaire sur la fonction MAJ-QUANTITE-PRODUIT-LIVRAISON-DIFFEREE qui est elle-même prioritaire sur la fonction MAJ-QUANTITE-PRODUIT-LIVRAISON.

3.4. Définition d'une architecture des traitements et des données pour l'application Petitpas.

A partir de l'analyse fonctionnelle faite dans les trois premiers paragraphes de ce chapitre, nous allons passer à l'analyse de programmation relative à la partie automatique du système d'information. Nous commencerons, pour les traitements, à établir une architecture des programmes se présentant sous la forme d'un réseau de processus communicants puis, pour les données, à dériver un premier modèle d'accès.

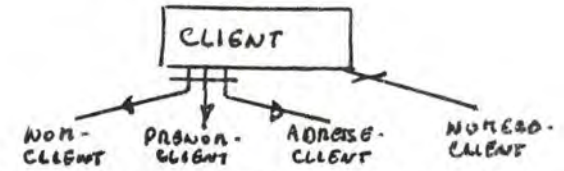
3.4.I. Structuration des traitements et dérivation d'un modèle relationnel binaire.

3.4.I.I. Dérivation d'un modèle relationnel binaire.

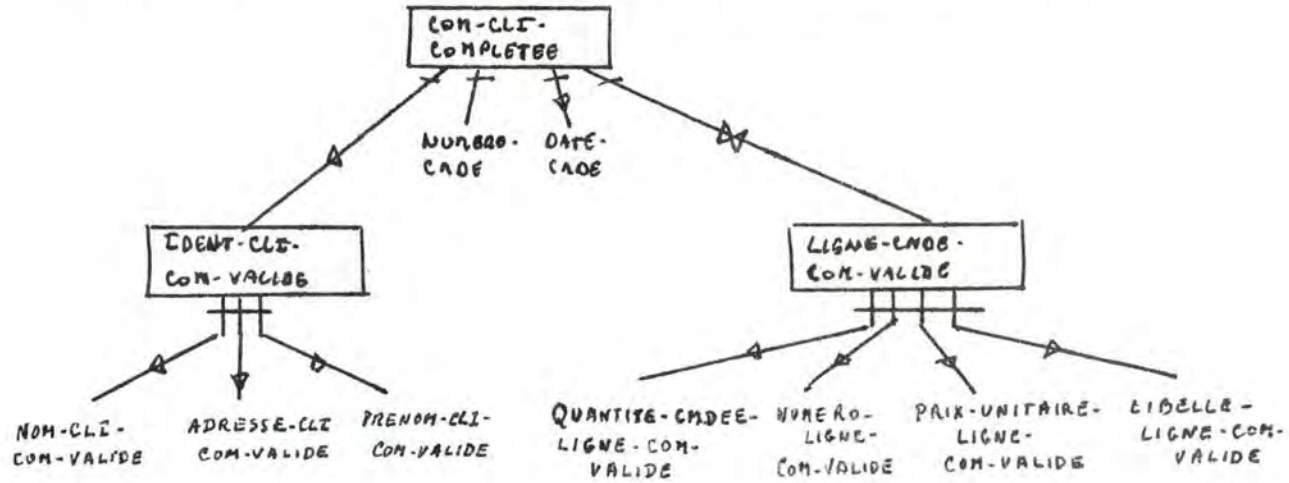
Pour l'ensemble des informations contenues dans les pavés de donnée, nous passons, au niveau de chaque pavé, de la description du type abstrait à une description à l'aide du modèle relationnel binaire. On trouve les règles de cette transformation dans le deuxième chapitre (cfr. 2.4.I.I.).

Exemples.

Pour la DOC.CLIENTS, on obtient le modèle suivant :



Pour la COM-CLI-COMPLETEE, on obtient le modèle suivant :



3.4.I.2. Structuration des traitements.

§I. Identification des modules fonctionnels dans l'arbre déductif.

En appliquant les critères énoncés aux pages 88, 89 et 90 dans le paragraphe 2.4.I., nous avons identifié dans la spécification des traitements (incluant l'arbre déductif et les traitements greffés ainsi que les fonctions intervenant dans l'évaluation des gardes) les modules fonctionnels suivants :

- "Identification du client" qui regroupe les pavés "construction-identification-client-connu" et "construction-erreur-identification-client (critères 2 et 3).
- "Construction d'une ligne de commande" qui regroupe les pavés "construction-ligne-produit" et "construction-erreur-ligne" (critères 2 et 3).
- "Calcul montant total" qui correspond à la fonction "montant-calculé" intervenant dans le prédicat "montant-correct" lié à l'acceptation d'une commande (critère 2).
- "Complétage d'une commande correcte" qui regroupe les pavés de "adjonction-date-commande" et "adjonction-numéro-commande" (critère 2).
- "Analyse d'une commande complétée" qui regroupe les pavés de "préparation-commande-complétée-livrable", "préparation-commande-complétée-épuisée", "préparation-commande-complétée-différée" et "construction-doc-épuisés" (critères 2 et 3).
- "Maj-stock" dont la description fait partie du module précédent. Il correspond à la fonction "maj-quantité-produit-livraison" qui est isolée du fait de son interférence possible avec d'autres fonctions portant sur le stock (critère 4).
- "Sélection commande complétée différée" qui correspond au pavé de "sélection-commande-différée" (critères 2 et 5).
- "Analyse commande complétée différée" qui correspond aux pavés de "préparation-commande-complétée-différée-livrable", "préparation-commande--complétée-différée-épuisée", "préparation-commande-complétée-différée-

-différée" et "construction-doc-épuisés" (critères 2 et 3).

- "Maj-différée" dont la description fait partie du module ci-dessus. Il correspond à la fonction "maj-quantité-produit-livraison-différée" qui est isolée du fait de son interférence possible avec d'autres fonctions portant sur le stock (critère 4).
- "Constitution série" qui correspond au pavé "constitution-série" (critère 5).
- "Ordonnancement" qui correspond au pavé "ordonnancement" (critères 2 et 5).
- "Correction expédition" correspondant aux pavés "correction-première-expédition" et "correction-expédition-différée" (critères I et 2).
- "Maj-rectificative" qui correspond à la fonction "rectification-quantité-produit-expédiée". La description de ce module fait partie de celui qui précède; ce module a été isolé du fait de son interférence avec d'autres fonctions portant sur le stock (critère 4).
- "Facturation" qui regroupe les pavés de "construction-facture", "construction-facture-différée", "enregistrement-facture", "enregistrement--facture-différée" et "construction-bon-livraison" (critères I et 5).
- "Construction documentation sur les différés" qui correspond au pavé "construction-doc-différés" (critères 2 et 5).

Les spécifications par assertions de ces différents modules seront facilement constructibles à partir de pré/post caractérisant chacune des fonctions de base intervenant dans la description du type d'objet sur lequel elle porte. A titre d'exemple, voici la pré/post qu'on obtient ainsi pour le module relatif à l'analyse d'une commande complétée :

précondition.

commande-complétée = (ident-cli-com-valide (nom-cli-com-valide, prénom-cli-com-valide, adresse-cli-com-valide), (ligne-cmde-com-valide (libelle-ligne-com-valide, numéro-ligne-com-valide, prix-unitaire-ligne-com-valide, quantité-cmdée-ligne-com-valide))*, date-cmde, numéro-cmde).

postcondition

[∃ { ligne-cmde-com-valide$_j$ }* ≠ ∅ tq ∀ ligne-cmde-com-valide$_j$:
ligne-livrable-totalement (ligne-cmde-com-valide$_j$, doc.prod) ∨
ligne-livrable-partiellement (ligne-cmde-com-valide$_j$, doc.prod)
⇒
ident-cli-com-livrable = ident-cli-com-valide
∧ date-cmde-com-livrable = date-cmde
∧ numéro-cmde-com-livrable = numéro-cmde
∧ (∀ ligne-cmde-com-valide$_j$:
libelle-ligne-com-liv = libelle-ligne-cmde-valide
∧ numéro-ligne-cmde-liv = numéro-ligne-cmde-valide
∧ prix-unitaire-ligne-cmde-liv = prix-unitaire-ligne-cmde-valide
∧ quantité-cmde-ligne-cmde-liv =
si ligne-livrable-totalement (ligne-cmde-com-valide$_j$, doc.prod)
alors quantité-cmdée-ligne-com-valide
sinon quantité-produit-disponible' (produit (: numéro-ligne-cmde-valide))
∧ quantité-produit-disponible (produit (: numéro-ligne-cmde-valide)) =
si quantité-cmdée-ligne-com-valide ≯ quantité-produit-disponible' (produit (: numéro-ligne-cmde-valide))
alors ∅
sinon (quantité-produit-disponible' (produit (: numéro-ligne-cmde-valide)) − quantité-cmdée-ligne-com-valide)
∧ prix-ligne-cmde-liv = prix-unitaire-cmde-liv ×
quantité-cmdée-ligne-cmde-liv)
∧ commande-complétée-livrable (ident-cli-com-livrable , date-cmde-com-livrable , numéro-cmde-com-livrable , (ligne-cmde-com-livrable)*) envoyé à "Constitution série"]

∨

[∃ { ligne-cmde-com-valide$_j$ } ≠ ∅ tq ∀ ligne-cmde-com-valide$_j$:
ligne-différable (ligne-cmde-com-valide$_j$, doc.prod)
⇒
ident-cli-com-diff = ident-cli-com-valide
∧ date-cmde-com-diff = date-cmde
∧ numéro-cmde-com-diff = numéro-cmde

$\wedge$ ($\forall$ ligne-cmde-valide$_j$:
libelle-ligne-com-diff = libelle-ligne-com-valide
$\wedge$ numero-ligne-com-diff = numero-ligne-com-valide
$\wedge$ quantite-cmdee-ligne-com-diff = quantite-cmdee-ligne-com-valide
- quantite-produit-disponible' (produit (: numero-ligne-com-valide))]
$\wedge$ doc.commande-completee-differable = doc.commande-complétée-différable'
$\cup$ commande-completee-différée (ident-cli-com-diff, date-cmde-com-diff,
numero-cmde-com-diff, (ligne-cmde-com-diff)*)]

$\vee$

[$\exists$ { ligne-cmde-valide$_j$ }* $\neq \emptyset$ tq $\forall$ ligne-cmde-valide$_j$:
non (ligne-livrable (ligne-cmde-valide, doc.prod) $\vee$
ligne-différable (ligne-cmde-valide, doc.prod))
$\Rightarrow$
ident-cli-com-épuisé = ident-cli-com-valide
$\wedge$ date-cmde-com-épuisée = date-cmde
$\wedge$ numero-cmde-com-épuisée = numero-cmde
$\wedge$ ($\forall$ ligne-cmde-valide$_j$:
ligne-cmde-com-épuisée : ligne-cmde-valide)
$\wedge$ doc.commande-complétée-épuisé = doc.commande-complétée-épuisée'
$\cup$ commande-complétée-épuisée (ident-cli-com-épuisée, date-cmde-
com-épuisée, numero-ligne-com-épuisée, (ligne-cmde-com-épuisée)*)]

§2. Identification des modules intéractifs.

A partir des pavés de saisie et de diffusion de données décrits dans le paragraphe 3.2.2., nous pouvons identifier des modules interactifs qui assureront la liaison entre les modules fonctionnels et les utilisateurs via les périphériques.

Nous pouvons obtenir de cette manière un module gérant le dialogue avec l'opératrice dans le service d'enregistrement des commandes et un autre gérant le dialogue avec le magasinier dans le service d'emballage et de facturation. Comme exemple, voici le module interactif gérant le dialogue avec l'opératrice lors de la tentative d'enregistrement d'une commande :

précondition

postcondition

(ident-cli-com-cli) reçu

∧ si saisie-ok (ident-cli-com-cli) ∧ client-identifiable (ident-cli-com-cli, doc-clients)

⇒

élu numéro-client tq (numéro-client, ident-cli-com-valide) ∈ doc-clients

∧ sorti-terminal (numéro-client, ident-cli-com-valide)

∧ ∀ ligne-cmde-com-cli ∈ lignes-cmde-com-cli :

(ligne-cmde-com-cli) reçu

∧ si saisie-ok (ligne-cmde-com-cli) ∧ ligne-produit-identifiable (ligne-cmde-com-cli, doc-produits)

⇒

[prix-ligne = prix-unitaire-valide * quantité-commandée-valide

∧ sorti-terminal (ligne-cmde-commande, prix-ligne)]

sinon sorti-terminal (erreur-ligne-com-cli-refusée)

∧ si ∄ erreur-ligne-com-cli-refusée

⇒

sorti-terminal (montant-total-calculé)

∧ si accord-montant (montant-total-calculé)

⇒

envoyé (ident-cli-com-valide, lignes-cmde-com-valide)

à "Complétage-commande-correcte"

sinon sorti-terminal (erreur-ident-cli-com-cli-refusée)

Remarques.

- saisie-ok est une condition qui peut être évaluée à la suite d'un certain nombre de contrôles élémentaires effectués par des modules d'entrée-sortie sur les données. Ces contrôles portent sur les formats, la syntaxe, les différentes valeurs acceptables;

- client-identifiable, ligne-produit-identifiable et ident-cli-com-valide, ligne-cmde-com-valide, montant-total-calculé, erreur-ident-cli-com-cli--refusée, erreur-ligne-com-cli-refusée sont respectivement les gardes et les résultats des modules fonctionnels "identification-client", "construction-ligne-commande" et "calcul-montant-total".
- accord-montant est la condition dont la valeur est définie par la réponse de l'opératrice lorsqu'elle a pu comparer le montant calculé à partir des lignes acceptées et celui figurant sur le bon de commande.

§3. Identification des modules d'entrée-sortie et d'accès.

Nous pouvons identifier comme module d'entrée-sortie les modules relatifs :

- à la saisie de l'information au service d'enregistrement et de contrôle de la commande client et au service d'emballage et de facturation;
- à l'édition des documents d'ordonnancement et des bons de livraison.

Nous aurions pu également identifier et isoler des modules d'accès aux informations stockées dans différentes documentations (modules accès doc. clients, doc.produits, ...). Cependant, une telle configuration aurait accru considérablement le nombre de communications entre les différents modules, ce qui n'est pas acceptable, comme nous le verrons plus tard, en raison de l'inefficacité du logiciel dont nous disposons pour la gestion des messages.

§4. Identification du module de gestion de la concurrence.

Nous avons exprimé dans le paragraphe 3.3. l'interférence qui existe entre les fonctions de mise à jour du stock. Cette interférence nous a guidés dans l'identification de trois modules correspondant à ces fonctions. L'exclusion mutuelle entre la réalisation de ces modules pourrait être assurée au niveau du gestionnaire de la base de données. Ce n'est cependant pas suffisant car il faut encore respecter la règle de priorité qui existe entre ces modules.

La solution que nous avons retenue est l'introduction d'un module de gestion de la concurrence : le moniteur produit.

A chacun des modules correspondant aux fonctions de mise-à-jour du stock, nous associons un compteur qui, manipulé dans le moniteur, permettra de vérifier la règle de priorité.

3.4.2. Simplification des données et des modules de traitement par unification.

2.4.2.I. Unification des données.

Une première unification possible, indépendante du contexte matériel et logiciel, est celle qui consiste à éliminer les structures de données sémantiquement et syntaxiquement équivalentes avec d'autres. Ainsi, dans l'application traitée, on remarque la redondance qui existe entre "doc. commandes-clients-différées" et "doc.cmdes-différées". On gardera dorénavant un seul de ces objets ("doc.commandes-clients-différées") et les traitements portant sur celui éliminé se reporteront sur celui conservé. Par le même raisonnement, disparaîtront également :

- le "bon-facturation-différée" dont le traitement est pratiquement identique (excepté le calcul des frais de transport) à celui portant sur le "bon-facturation";
- l'"expédition-différée" dont le traitement est identique à celui portant sur l'"expédition-première".

La seconde unification possible peut avoir lieu lors de la prise en compte des moyens techniques à notre disposition. Ainsi, pour l'application traitée, nous utiliserons :

- le logiciel Traffic-20 de gestion d'écran et de communication entre programmes (DIG,79);
- le logiciel DBMS-20 de gestion de base de données (DIG,77).

Le logiciel de communication de messages est caractérisé par

- sa lenteur;
- la taille réduite des messages admis;
- la non-structuration des données du message.

L'utilisation de la base de données, dans une configuration centralisée,

va nous permettre de réduire la taille et la complexité de certains des messages en ne passant, dans la mesure du possible, que des identifiants de ces messages, lesquels permettront d'accéder aux messages stockés dans la base de données. D'autre part, nous pourrons profiter des informations déjà mémorisées et donc ne plus devoir mémoriser de nouvelles informations qui seraient redondantes.

A titre d'exemple, on trouvera une application de la technique employée, dans l'annexe 3, lors de la mémorisation d'une "com-cli-complétée".

2.4.2.I. Unification des traitements.

A la suite de la première unification de données, on peut :

- supprimer le module "construction-documentation-sur-les-différés" car le traitement réalisé est le même que celui que subit "doc.commandes--clients-différées" dans d'autres modules;
- simplifier les pré/post des modules suivants : "constitution-série", "ordonnancement", "correction-expédition", "maj-rectificative" et "facturation".

De la même manière, après la seconde unification, nous simplifierons la spécification de chaque traitement décrit dans les modules fonctionnels et d'interface, ce qui nous amènera à profiter pleinement de la propriété de non-redondance offerte par la base de données tout en réduisant au maximum la complexité des messages.

On trouvera ci-dessous, à titre d'exemple, la nouvelle spécification du module "analyse-commande-complétée"; il est intéressant de remarquer sa simplification par rapport à celle décrite précédemment (cfr. 3.4.I.2.,§I).

précondition

numéro-cmde

postcondition

$[\exists \{ \text{ligne-cmde/com-validée} \}^{*} \subseteq \text{com-cli-complétée}$ et $\neq \emptyset$ tq $\forall$ ligne-cmde/com-validée :

ligne-livrable-totalement (ligne-cmde/com-validée, doc.prod) $\vee$

ligne-livrable-partiellement (ligne-cmde/com-validée, doc.prod)

$\Rightarrow$

($\forall$ ligne-cmde/com-valide$_j$: $\exists$ ligne-cmde/com-livrable$_i$ ((: com-cli-complétée), (: produit)) tq

quantité-cmdée-com-livrable =

si ligne-livrable-totalement (ligne-cmde/com-valide$_j$, doc-prod)

alors quantité-cmdée-com-valide

sinon quantité-disponible'(: produit)

$\wedge$ quantité-disponible (: produit) =

si quantité-cmdée-com-valide $\geq$ quantité-disponible'(: produit)

alors $\emptyset$

sinon quantité-produit-disponible'(: produit) - quantité-cmdée-com-valide)

$\wedge$ (numéro-cmde) envoyé à "Constitution série"]

$\vee$

[$\exists$ { ligne-cmde/com-valide }$^* \leq$ com-cli-complétée et $\neq \emptyset$ tq $\forall$ ligne-cmde/com-valide$_j$:

ligne-différable (ligne-cmde/com-valide$_j$, doc-prod)

$\Rightarrow$

($\forall$ ligne-cmde/com-valide$_j$: $\exists$ ligne-cmde/com-différée$_i$ ((: com-cli-complétée), (: produit)) tq

quantité-cmdée-com-différée = quantité-cmdée-com-valide -

quantité-disponible'(: produit))]

$\vee$

[$\exists$ { ligne-cmde/com-valide }$^* \leq$ com-cli-complétée et $\neq \emptyset$ tq $\forall$ ligne-cmde/com-valide$_j$:

non ((ligne-livrable (ligne-cmde/com-valide$_j$, doc-prod) $\vee$

(ligne-différable (ligne-cmde/com-valide$_j$, doc-prod))

$\Rightarrow$

($\forall$ ligne-cmde/com-valide$_j$: $\exists$ ligne-cmde/com-épuisée$_i$ ((: com-cli-complétée), (: produits)) tq

quantité-cmdée-com-épuisée =

quantité-cmdée-com-valide)]

$\wedge$

{ ligne-cmde/com-valide }$^* =$ { ligne-cmde/com-valide }$^{*'} \setminus$

{ ligne-cmde/com-valide }$^* \leq$ com-cli-complétée

3.4.3. Choix d'une décomposition physique des traitements.

A partir des modules décrits jusque maintenant, nous pouvons opérer un regroupement de plusieurs de ceux-ci de la manière la plus performante possible. Les critères qui nous guident sont ceux énoncés dans le paragraphe 2.4.3.. A ceux-ci, il faut ajouter notre désir de contrecarrer la lenteur du logiciel de gestion des communications en minimisant le nombre des messages par le regroupement de plusieurs modules en une seule unité d'exécution.

On trouvera à la figure II l'architecture des programmes obtenus ainsi que l'indication de la nature des communications entre les différents programmes.

3.4.4. Dérivation d'un premier modèle d'accès aux données.

A partir du modèle relationnel binaire représentant l'information à mémoriser, nous pouvons dériver une première structure d'accès possible. Les règles présidant à cette transformation sont celles décrites dans le paragraphe 2.4.4..

La structure d'accès obtenue sera celle présentée à la figure I2.

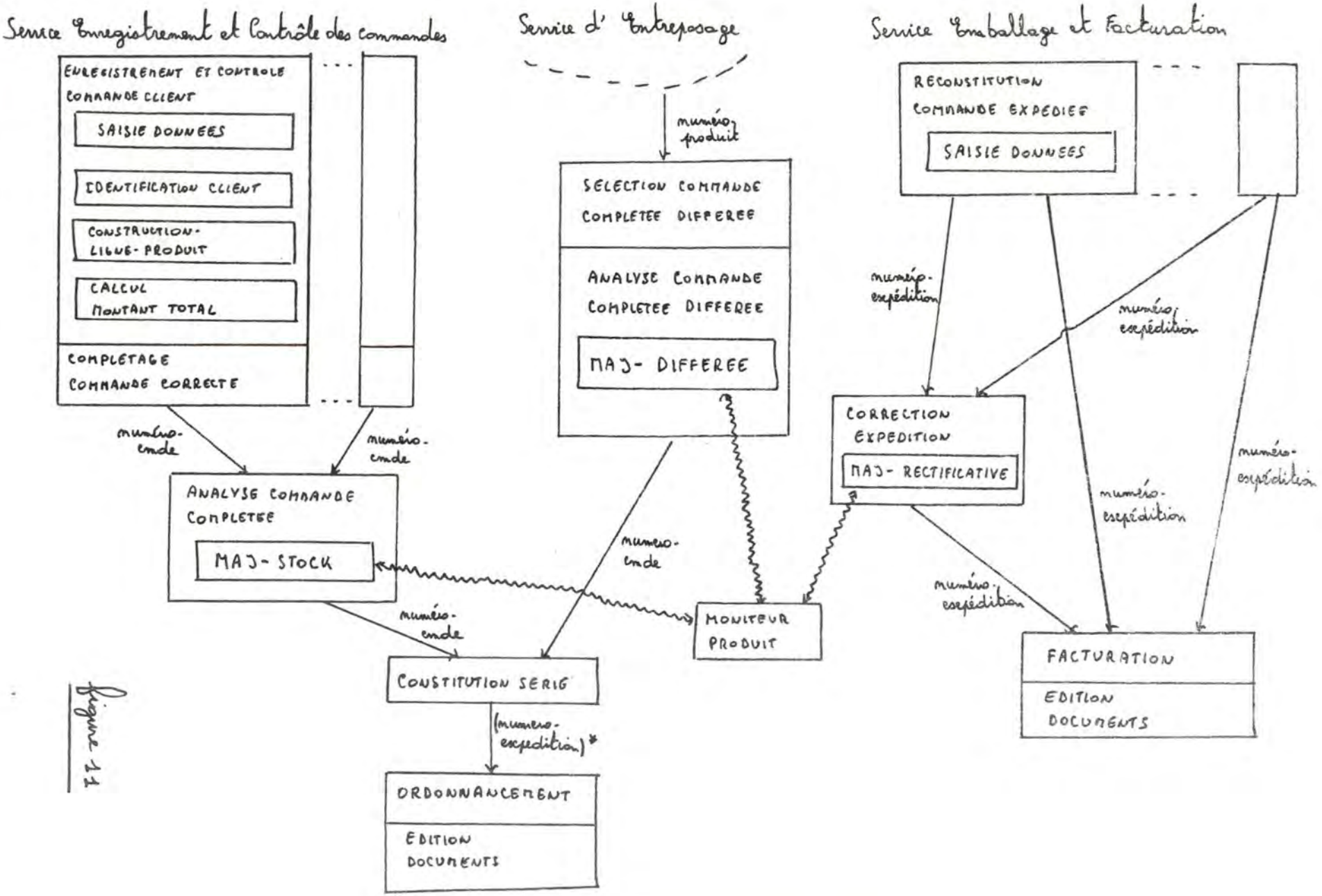

Service Enregistrement et Contrôle des commandes
Service d' Entreposage
Service Emballage et Facturation
ENREGISTREMENT ET CONTROLE COMMANDE CLIENT
SAISIE DONNEES
IDENTIFICATION CLIENT
CONSTRUCTION-LIGNE-PRODUIT
CALCUL MONTANT TOTAL
COMPLETAGE COMMANDE CORRECTE
numéro-cmde
numéro-cmde
numéro produit
SELECTION COMMANDE COMPLETEE DIFFEREE
ANALYSE COMMANDE COMPLETEE DIFFEREE
MAJ- DIFFEREE
RECONSTITUTION COMMANDE EXPEDIEE
SAISIE DONNEES
numéro-expédition
numéro-expédition
numéro-expédition
numéro-expédition
numéro-expédition
CORRECTION EXPEDITION
MAJ- RECTIFICATIVE
ANALYSE COMMANDE COMPLETEE
MAJ- STOCK
numéro-cmde
numéro-cmde
numéro-cmde
MONITEUR PRODUIT
CONSTITUTION SERIG
(numéro-expédition)*
ORDONNANCEMENT
EDITION DOCUMENTS
FACTURATION
EDITION DOCUMENTS

figure 11

Premier modèle des accès possibles

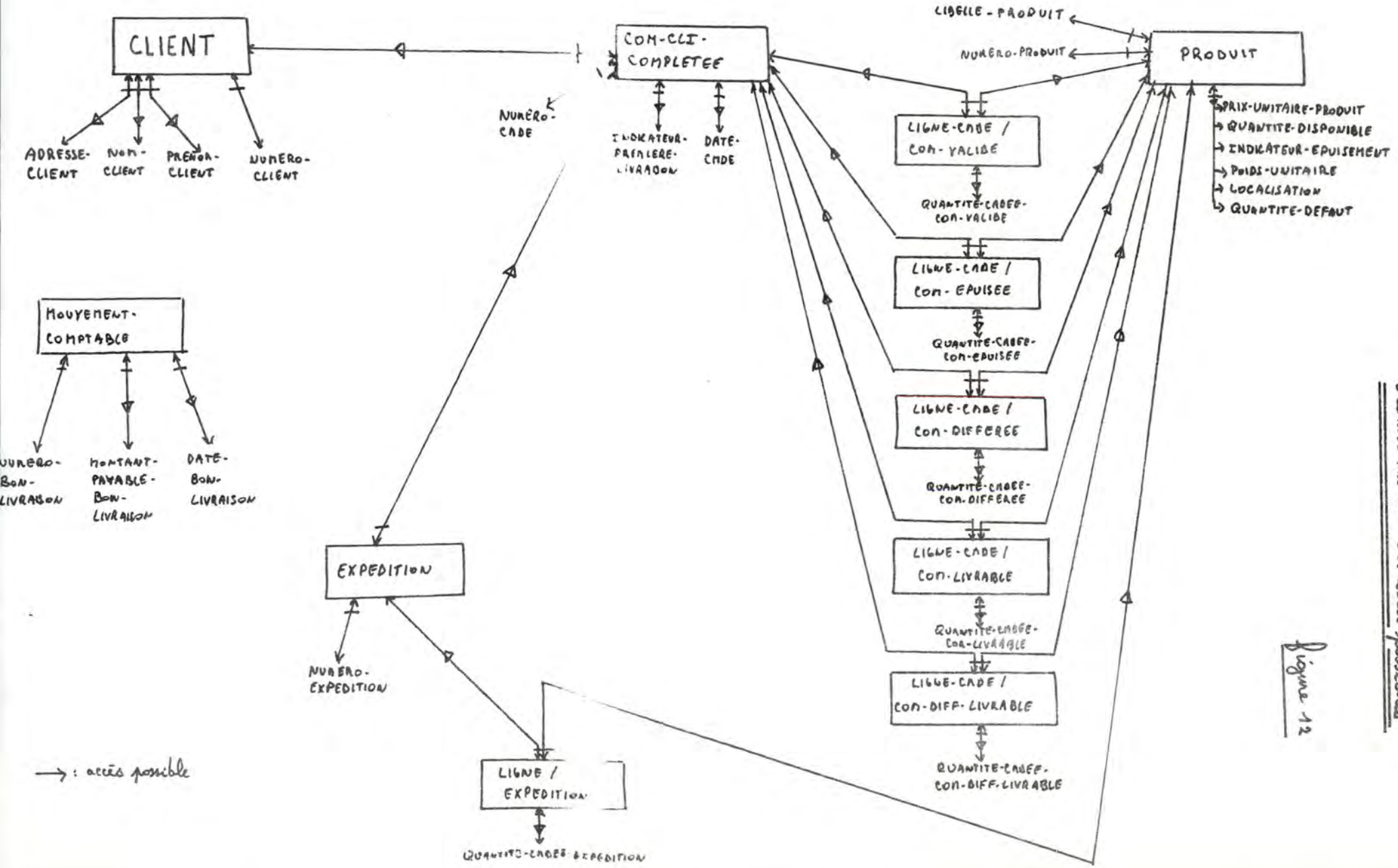


figure 12

3.5. Programmation et dérivation du modèle des accès utilisés.

3.5.I. Programmation.

Jusqu'ici, la description des différents modules intervenant dans les programmes s'est faite uniquement en termes non-algorithmiques par des spécifications pré/post.

Nous allons maintenant construire les différents algorithmes et les exprimer dans un pseudo-langage (une brève description de celui-ci se trouve page I0I). Concernant les opérations sur la base de données, elles portent sur le schéma des accès possibles décrit à la figure I2.

On trouvera divers exemples de l'utilisation du pseudo-code dans l'annexe 4.

3.5.2. Modèle des accès utilisés.

A partir des algorithmes décrits ci-dessus et du schéma des accès possibles, nous pouvons mettre en évidence les différents accès et mises à jour nécessaires à l'application. On pourra alors déduire le schéma des accès utilisés avec les opérations requises tel qu'il est présenté à la à la figure I3.

Remarque.

Pour être complet, il aurait encore fallu spécifier un ensemble de quantifications, comme le fait (HAI,79b), concernant les types d'informations et les opérations portant sur eux.

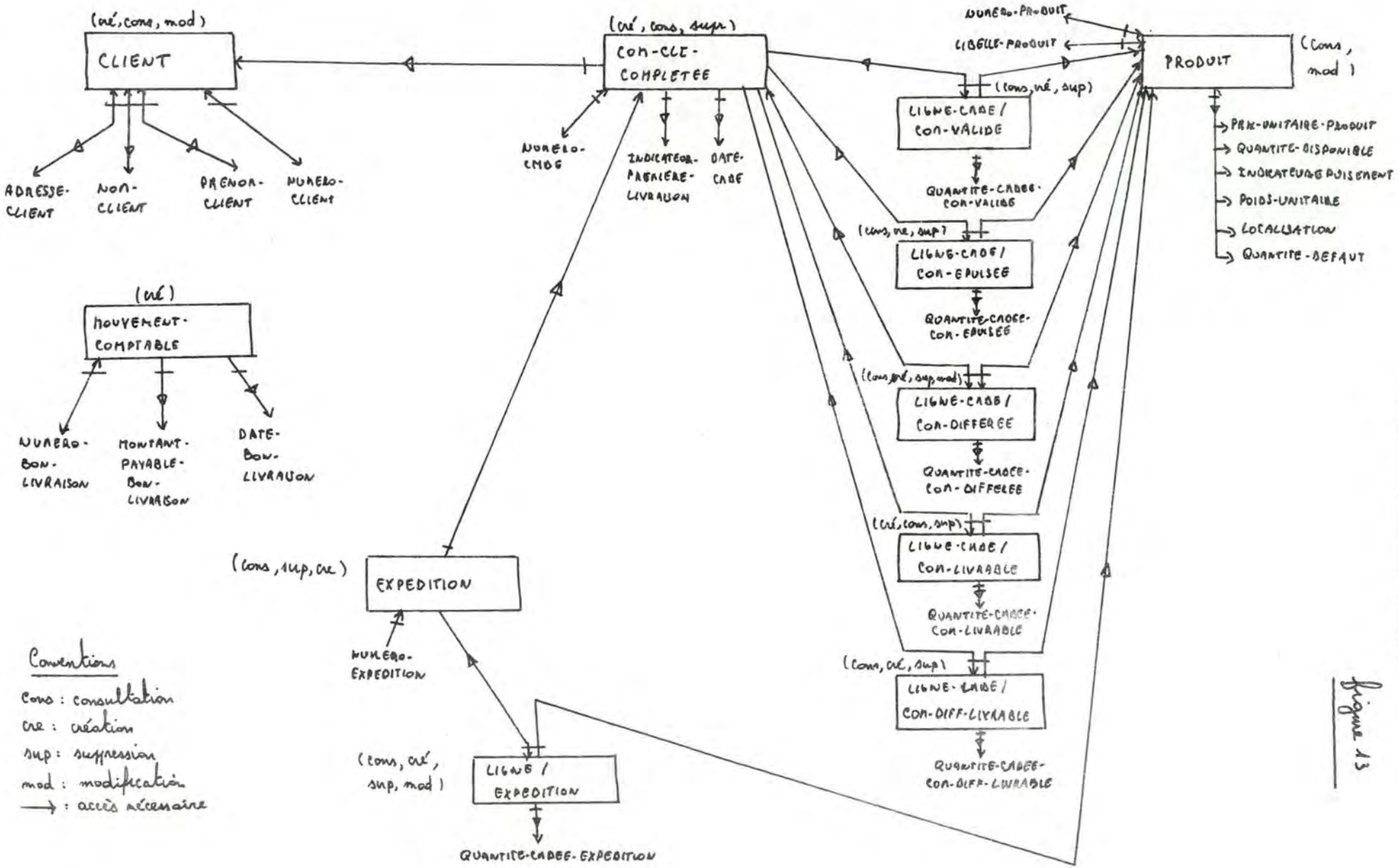


Schéma des accès utilisés

figure 13

3.6. Implémentation.

3.6.I. Implémentation de la structure de données.

A partir du schéma des accès utilisés et des quantifications, il est possible de proposer quelques implémentations de la structure de données en DBMS-20 (DIG,77).

L'implémentation choisie a été retenue en fonction de divers critères mis en évidence par (HAI,79b).

Ce choix est provisoire et devrait peut-être être modifié en fonction d'une étude plus approfondie portant sur les aspects de sécurité et de concurrence sur le logiciel DBMS-20; ces aspects n'ont pas été traités plus en détail dans ce travail car ils nous semblent sortir de l'objectif que nous nous sommes assignés, c'est-à-dire de montrer la faisabilité de notre démarche d'analyse fonctionnelle et d'analyse de programmation. On trouvera, en annexe 5, le schéma de la base de données retenu.

3.6.2. Implémentation de la structure des traitements.

Par des règles de transcription, on peut coder de manière systématique en COBOL (langage de programmation retenu pour l'application), les parties du programme n'ayant pas trait aux opérations sur la base de données.

De plus, il est également possible d'assurer la traduction des opérations d'accès et de modification de données en appliquant les règles de traduction données par (HAI,8Ib).

Un exemple de codage systématique relatif au programme d'"analyse d'une commande complétée" se trouve en annexe 6.

Conclusion.

La conception, la réalisation et l'exploitation des systèmes d'information sont devenues des tâches nécessaires pour assurer au sein des organisations un fonctionnement cohérent, respecter les objectifs assignés et permettre de maîtriser la complexité croissante des interactions possibles. Cependant, force nous est de constater qu'il existe peu de méthodes permettant de décrire et de développer ces systèmes d'information de manière effective. Une étape cruciale concerne la description des besoins exprimés par les demandeurs. En effet, le manque de spécifications précises entraîne indubitablement la réalisation d'un système ne répondant pas exactement aux attentes des utilisateurs, ainsi des coûts de développement et de maintenance prohibitifs. On trouve des cahiers de charge souvent énormes, complexes, imprécis, incomplets et difficiles à contrôler; il existe en effet peu de démarches permettant de répondre à la multiplication des questions posées par une description :

- comment comprendre et décomposer le problème ?
- que faut-il mettre dans une spécification ?
- pour qui faut-il spécifier ?
- quelles sont les qualités et les défauts d'une spécification ?

Pour tenter de répondre à ces questions, nous avons proposé une démarche permettant de décrire un système d'informations par raffinements successifs, tout en tenant compte des contraintes de l'environnemnt organisationnel. Résumons brièvement cette démarche.

Nous étendons la "méthode déductive" développée au CRIN de Nancy (PAI,79) et utilisé comme outil de spécification de programmes (FIN,79) à la spécification d'un système d'information.

Cette approche déductive à partir des résultats permet, d'une part, de présenter une découpe en traitements dont la description est non-procédurale, tout séquencement étant dicté par la dépendance entre données,

et, d'autre part, une décomposition de la structure des données utilisées. Ce premier schéma général de spécification s'est attaché à caractériser un "noyau logique" inhérent au type d'application envisagé; il ne permet donc pas d'appréhender un certain nombre de concepts et de contraintes propres à l'environnement organisationnel considéré. L'approche et l'intégration de ceux-ci dans la première spécification conduisent à une spécification complète du système en étude.

Celle-ci se présente sous forme d'un ensemble de pavés :

- certains assurent une description des traitements correspondant aux fonctions du système à réaliser;
- d'autres assurent une description des données sous forme de types abstraits;
- d'autres encore assurent une description des interfaces entre le système d'information et l'environnement organisationnel.

Ces descriptions sont données à la fois sous forme de texte en langue naturelle, en vue de vérifications possibles par le demandeur, et sous forme de propriétés fonctionnelles formalisées, en vue des étapes ultérieures de réalisation.

Nous avons ensuite proposé un mode d'articulation entre cette analyse fonctionnelle et les étapes ultérieures de réalisation. Dans un premier temps, on identifie et on spécifie un certain nombre de modules de différents types: modules reflétant directement l'analyse fonctionnelle (nous montrons que leur spécification se déduit de la spécification fonctionnelle), modules enfermant des décisions de programmation, et modules enfermant des caractéristiques propres au matériel et au logiciel utilisés. Parallèlement à la description de ces modules, on établit, à partir des pavés de spécification des données et à l'aide du formalisme binaire, la structure de l'information qui devra être mémorisée. Dans un second temps, on procède à une découpe physique des traitements en programmes et on dérive un premier modèle d'accès aux données.

On passe alors à la programmation dans un pseudo-langage et à la détermination des accès nécessaires. Il reste enfin à traduire ces programmes et ce modèle d'accès en code, et ce , en fonction des caractéristiques de logiciels et matériels disponibles.

Tâchons maintenant de voir en quoi la démarche proposée permet de répondre à certaines des questions qui se posent dans le travail de spécification et qu'on a rappelée en début de chapitre.
La démarche proposée a les caractéristiques suivantes :

- analyse descendante et déductive à partir des résultats;
- décomposition en structures de données et en structures de traitements;
- description non-procédurale des traitements;
- distinction en "specification in the large" et "specification in the small";
- enrichissement progressif de la spécification "logique" par la prise en compte de l'environnement organisationnel;
- utilisation de types abstraits dans le cadre de la spécification d'un problème de taille relativement importante.

Voyons maintenant les avantages de ces caractéristiques:

- la démarche descendante et déductive permet de structurer le problème à l'aide de fonctions logiquement découpées et amenant un arbre déductif de traitements; de plus, on part de la description des résultats, qui sont généralement mieux connus;
- l'utilisation des types abstraits permet de décrire les objets rencontrés par leurs propriétés et les opérations définies à leur sujet sans introduire de détails de représentation;
- la découpe des traitements est basée, dans un premier temps, uniquement sur le séquencement entre les données du problème;
- la démarche proposée semble pouvoir guider le spécificateur dans son activité en lui permettant de ne devoir se préoccuper que d'un nombre restreint d'éléments constitutifs de la spécification;

- une maintenance plus facile par la localisation des traitements dans l'arbre déductif et dans les "greffes".

Comme nous l'avons montré, un autre avantage de la démarche est qu'elle se prête de manière assez systématique à un prolongement au niveau de la réalisation informatique; la vérification qu'un tel prolongement conserve la sémantique des spécifications nous paraît plus aisée que dans bon nombre d'autres méthodes. On peut, en effet, se convaincre facilement que la sémantique des spécifications des pavés de traitement, de données, d'interface et d'expression de l'interférence est conservée dans les pré/post conditions des modules fonctionnels, interactifs et de gestion de la concurrence. De même, la sémantique des relations entre pavés de traitement et spécification de propriétés de données se retrouvent dans le modèle binaire. Ensuite, les pré/post conditions des modules sont transformées en algorithmes de traitement tandis la modèle binaire l'est en un modèle d'accès. La vérification de l'équivalence sémantique entre pré/post conditions et algorithmes n'est évidemment pas chose aisée; cet aspect constitue un domaine très actif de recherches ((FLO,67), (HOA,68), (DIJ,76), etc...). Au niveau de l'implémentation enfin, toujours selon le principe d'équivalence sémantique, le schéma des accès logiques est transformé en base de données tandis que les algorithmes de traitement sont transformés en code.

Nous pensons dès lors qu'une démarche comme celle décrite ci-dessus peut assurer la réalisation et l'exploitation d'un système informatique répondant scrupuleusement aux attentes de l'utilisateur, en permettant la vérification de l'exactitude de ce qui a été implémenté par rapport à ce qui a été spécifié (SPI,78).

Mentionnons, pour terminer, qu'au niveau de la démarche dans son ensemble, l'effort apporté au niveau des spécifications, et, dans une moindre mesure, à celui des premières étapes de l'analyse de programmation, a permis

un développement plus rapide des algorithmes et de leur codage. Ainsi, dans le cadre de l'application traitée, la programmation de modules fonctionnels en pseudo-code a pris un jour et demi et lors du codage de quelques-uns de ces modules, les seuls obstacles qui sont apparus ont relevé des caractéristiques (ou des anomalies?) des logiciels employés.

Le présent travail comporte bien entendu un certain nombre de limites. Parmi les aspects qui nous semblent requérir une étude plus approfondie, citons les points suivants :

a) Au niveau des fonctions terminales, nous avons dû nous contenter de donner quelques "recettes" pour déterminer la "profondeur" de décomposition des fonctions; il serait intéressant d'investiguer davantage cette notion en tâchant d'énoncer des critères plus précis pour raffiner ou s'arrêter.

b) Au niveau des traitements, nous avons distingué le noyau de traitements "logiques" et l'ensemble des traitements "organisationnels"; existe-t-il certaines caractéristiques dans les traitements qui permettraient leur identification dans une classe plutôt que dans l'autre ?

c) Au niveau de la structure des données, nous n'avons éliminé la redondance des informations qu'au niveau de l'analyse de programmation; on aurait pu procéder plus tôt à cette élimination, en utilisant, au niveau des types abstraits, les notions de "constructeurs" et d' "extension" ainsi que les possibilités offertes par les types paramétrés (DER,79).

d) Plutôt que d'enrichir progressivement la structure des données au fur et à mesure de la construction de la spécification, ne serait-il pas préférable, dès le départ, de prendre en compte, comme le suggèrent par exemple (JAC,78) et (BOD,8I), la réalité à modéliser et ensuite

de superposer les fonctions sur ce modèle, ou encore de s'orienter vers une approche unifiée combinant ces deux points de vue.

e) Dans le contexte d'une application temps-réel, nous avons déjà mentionné (cf.2.3.) le traitement peu satisfaisant des aspects de concurrence: comment spécifier ces aspects, à quel niveau ?

f) Dans le cadre de la description d'un système de taille raisonnable, la tâche du spécificateur devient rapidement fastidieuse du fait de la multiplication des pavés de données et de traitement; des aides automatiques pour maîtriser cette complexité nous paraissent indispensables.

L'applicabilité de la démarche à un problème " en vraie grandeur " n'a été jugée que sur base du problème traite dans le troisième chapitre; une expérience plus étendue dans le traitement d'autres problèmes s'avère nécessaire pour tester le caractère effectif de la méthode et pour proposer un certain nombre de raffinements et d'ajustements.

Annexe I

PROPOSITION D'AUTOMATISATION DE L'ENTREPRISE PETITPAS.

1.0 MODIFICATION DE LA POLITIQUE DE LIVRAISON.

Dans le cadre de la politique de limitation des ruptures de stocks et de l'amelioration du service a la clientele, la firme a decide de ne plus limiter le nombre de differes par commande.

En effet, l'analyse de la nouvelle solution a permis de mettre en evidence que, compte tenu de l'amelioration attendue de la gestion des stocks et des clauses de contrats de transport avec la SNCB et la Regie des Postes, il serait commercialement avantageux pour la firme d'admettre a priori pour une commande un nombre illimite de differes. Les livraisons s'effectueront au fur et a mesure des reapprovisionnements; la commande ne sera archivee qu'apres son apurement complet ou epuisement complet des articles restant a livrer.

Le paiement par cheque entrainant des couts administratifs eleves, la firme a decide de le supprimer et de generaliser le systeme d' expedition contre remboursement

2.0 NOUVELLE ORGANISATION DE LA FIRME.

La gestion en temps reel des stocks (enregistrement des commandes client et des livraisons fournisseurs) a provoque la reorganisation suivante de la firme:

1. La Direction Generale et les departements Commercial, Financier, et du Personnel restent inchanges.
2. Le departement de Production sera organise commu suit:

   2.1 Service d'enregistrement de commandes procedera a l'aide de terminaux

   - au controle des commandes
   - a la mise a jour du "fichier" des Adresses-client
   - a l'enregistrement de commandes

   2.2 Le service de preparation des commandes procedera a la recherche et a l'enlevement des produits par X series de Y commandes.

2.3 Le service emballage et expedition procedera a l'aide de terminaux

- a la reconstitution de la commande
- au lancement de la facturation
- a l'emballage et a l'expedition du colis

2.4 Inventaire permanent

Tous les vendredis midi, les services de preparation des commandes, d'emballage et d'expedition cessent leur travail habituel et procedent a un inventaire permanent. Ils rangent les rayons, comptabilisent le nombre d'articles de chaque rayon et signalent l'etat des stocks a un gestionnaire des stocks qui corrige eventuellement la quantite en stock dans le "fichier d'Etat des stocks.

3. Departement Achats et Fournisseurs.

Ce departement n'est plus constitue que du seul service des achats qui se charge d'effectuer la gestion des fournisseurs: choix des fournisseurs, commandes aux fournisseurs, suivi des commandes et des livraisons, analyse de l'Etat des stocks et des commandes client. Ce service est egalement dote de terminaux.

4. Departement d'entreposage.

Ce departement s'occupe de la reception des livraisons fournisseurs, de leur controle et de leur enregistrement par terminal.
Il est aussi responable du rangement dans les magasins des marchandises recues.

5. Un systeme informatique, base sur un mini-ordinateur ou un reseau de micro-ordinateurs executera:

1- en liaison temps reel avec le service d'enregistrement des commandes client, le controles et l'enregistrement de ces commandes

2- en liaison temps reel avec le service d'entreposage, le controle et l'enregistrement des livraisons fournisseur

3- en liaison temps reel avec le service d'emballage et d'expedition, l'edition de tous les documents d'expedition en ce compris la facture

4- en liaison temps reel avec le service des achats, une aide a la preparation des commandes fournisseur

Compte tenu des liaisons avec les autres services, il assumera egalement les traitements suivants:

1- mise a jour de l'etat des stocks et des ressources

2- gestion des commandes differees

3- optimisation des recherches en magasin (ceci afin d'optimiser les parcours).

3.0 LES NOUVELLES PROCEDURES

3.1 ENREGISTREMENT ET CONTROLE DES COMMANDES CLIENTS.

3.1.1 PREPARATION DU BON DE COMMANDE -

A chaque arrivee de courrier, ce poste recoit les bons de commande (inchanges par rapport a la solution manuelle). Il decachete les enveloppes et s'assure que les bons sont signes.
Les bons de commande non signes, sont transmis a la cellule de traitement des cas litigieux, les autres sont diriges vers les operatrices d'enregistrement des commandes.

3.1.2 ENREGISTREMENT ET CONTROLE DES BONS DE COMMANDE CLIENTS.

Ce poste se compose de quelques operatrices qui sont en liaison directe avec l'ordinateur grace a un terminal a ecran. Le principe est de composer a l'ecran le bon de commande controle et eventuellement corrige. C'est seulement si cette operation a reussi que le bon de commande est enregistre et que s'effectueront les mises a jour du "fichier" Adresse-client.

En cas d'echec, le bon de commande client est transmis a la cellule de traitement des cas litigieux.

L'operatrice effectue dans l'ordre les operations suivantes:

A- IDENTIFICATION DU CLIENT.

On distingue deux cas, selon que le bon de commande possede un numero de client non prospect preimprime ou selon que ce numero n'existe pas.

Dans le premier cas, l'operatrice procedera a une recherche dans le "fichier" Adresse-client pour verifier l'existence de ce numero et la concordance des adresses si ce numero existe.
Si le numero n'existe pas, elle effectuera une recherche sur les nom, prenom et adresse afin de verifier une eventuelle concordance.

Dans le second cas, l'operatrice procedera a une recherche sur nom prenom et adresse.

B- CONSTITUTION DU CORPS DE LA COMMANDE

L'operatrice constitue la commande ligne par ligne. Elle a pour objectif d'interpreter au mieux le bon de commande du client afin de perdre le moins de commande possible. Pourtant si elle ne peut interpreter une ligne, elle abandonne entierement la commande. La regle d'interpretation est d'accorder la priorite au libelle du produit par rapport au numero de reference.
On distingue deux cas, selon que la ligne possede ou non un numero de produit. L'operatrice procede a l'identification d'une ligne par une analyse de concordance des libelles.
Pour un produit identifie, en l'absence d'une quantite commandee mentionnee par le client, l'ordinateur affichera la quantite par defaut.

Si la commande ne peut etre acceptee, l'operatrice marque la ou les ligne(s) litigieuse(s) sur le bon de commande et expedie celui-ci a la cellule de traitement des cas litigieux.

Si toutes les lignes du bon de commande ont pu etre interpretees, l'ordinateur affiche alors le prix total a payer par le client.
L'ecart entre le total calcule et celui du bon de commande est apprecie par l'operatrice, qui decide d'enregistrer ou de refuser la commande auquel cas cette commande est expediee au service des cas litigieux.

3.1.3 TRAITEMENT DES CAS LITIGIEUX.

Ce bureau examine consciencieusement les bons de commande refuses et tente de les recycler. A defaut, il envoye au client son bon de commande annote, un justificatif de refus et un nouveau bon de commande vierge.

3.2 PREPARATION ET ENREGISTREMENT DES COMMANDES AUX FOURNISSEURS.

Quotidiennement, le service des achats aux fournisseurs consulte le "fichier" d'Etat des stocks. Cette interrogation fournit une liste des stocks a reapprovisionner sur base des calculs suivants:

Niveau des ressources = Quantite en stock +
Quantite en commande chez les fournisseurs -
Quantite due pour les commandes differees.

Si le niveau des ressources est < ou = au Point de Commande, il y a decision de reapprovisionnment. La quantite a commander doit etre le plus petit multiple de la Quantite economique de commande superieur a la difference entre le point de commande et le niveau des ressources. Connaissant ces informations,

le service des achats consulte eventuellement le "fichier" des Fournisseurs et celui des Commandes Fournisseurs en cours, contacte les fournisseurs et etablit un bon de commande au fournisseur selectionne.

Par fournisseur, le gestionnaire des achats possede notamment les renseignements suivants relatifs aux produits:

- prix unitaire
- delai de livraison
- capacite de livraison
- pourcentage de remise
- couts de transport.

Chaque commande adressee a un fournisseur, possede un numero de commande attribue par compostage.

Ce bon de commande est alors enregistre dans le "fichier" des Commandes fournisseur en cours, et pour chaque produit commande la quantite commandee est ajoutee a la Quantite en commande du "fichier" Etat des stocks.

3.3 RECEPTION DE LA COMMANDE FOURNISSEUR.

Le service d'entreposage receptionne les livraisons en provenance des differents fournisseurs. Il effectue le controle de la marchandise:

A- En consultant en temps reel le "fichier" des Commandes fournisseur en cours, il verifie que la livraison correspond en tout ou en partie a une commande chez ce fournisseur.

B- En s'assurant que la marchandise est conforme aux normes de qualite exigees.

Au fur et a mesure des manipulations de la marchandise, celle-ci est acheminee vers les rayonnages dans le magasin. Ensuite, il met a jour le "fichier" des Commandes fournisseurs en cours (en apurant les commandes les plus anciennes).

3.4 MISE A JOUR DE L'ETAT DES STOCKS.

3.4.1 TRAITEMENT DES ENTREES EN STOCK <MAJ+>.

Ce traitement est active chaque fois qu'une transaction "Produit reapprovisionne" est creee par le traitement de reception des commandes fournisseur.

La quantite emmagasinee (Quantite livree - Quantite refusee) est ajoutee a la quantite disponible en stock et soustraite de la quantite en commande aupres du fournisseur.

Apres la mise a jour du stock, l'ordinateur procede a la selection des commandes differees dont une ligne au moins est relative a un produit reapprovisionne.

3.4.2 TRAITEMENT DES SORTIES DE STOCK <MAJ-> ET ORDONNANCEMENT.

Ce traitement est active par deux types de transactions:

soit une commande enregistree par le service d'enregistrement des commandes
soit une commande differee selectionnee.

Les commandes differees selectionnees sont traitees en priorite par rapport aux commandes normales.
Ce traitement effectue les operations decrites dans les tables de decisions qui suivent, ligne de commande par ligne de commande.

A- pour une commande enregistree

	cmde differee	cmde a facturer	etat du stock
stk epuise	0-> qte due	E-> indic. epuise 0-> qte livree qte cmdee-> qte restant due	
stk#epuise et qte en stk > ou = qte cmdee	0-> qte due	0-> qte restant due qte cmdee-> qte livree	qte en stk=qte en stk - qte livree
stk#epuise et qte en stk <qte cmdee	qte due= qte cmdee - qte en stk	qte livree = qte en stk qte restant due = qte cmdee - qte livree	0-> qte en stk qte diff=qte diff +(qte cmdee - qte livree)

Pour chaque produit commande, l'existence d'une quantite a livrer engendre une REQUISITION. Les requisitions relatives a une commande sont regroupees en une transaction appelee "bon de requisition". La date du jour figure dans un bon de requisition. On notera que si un produit est epuise, un message est enregistre dans le systeme.

B- pour une commande differee selectionnee.

	cmde differee	cmde a facturer	etat du stock
stk epuise	0-> qte due	E-> indic. epuise 0-> qte livree qte due -> qte restant due	
stk#epuise et qte en stk > ou = qte due	0-> qte due	0-> qte restant due qte due -> qte livree	qte en stk=qte en stk - qte livree qte diff=qte diff - qte livree
stk#epuise et qte en stk <qte due	qte due= qte due - qte en stk	qte livree = qte en stk qte restant due = qte due - qte livree	0-> qte en stk qte diff=qte diff - qte livree (qte diff vaut 0 si elle est < 0

Les regles de requisition, ainsi que le traitement des produits epuises sont analogues au cas de l'enregistrement d'une commande client. On notera que pour la mise a jour du stock:

- la <MAJ+> a priorite sur la <MAJ->
- les commandes differees ont priorite sur les commandes normales.

En puisant dans le casier et en suivant a l'ecran, il reconstitue le colis. S'il peut reconstituer ce colis entierement, il enclenche la facturation c'est-a-dire:

- l'impression d'une facture et d'un justificatif de non livraison
- l'impression d'une etiquette de colisage
- l'impression d'un bordereau SNCB ou Poste suivant le poids total du colis.

Cette facturation genere un mouvement comptable qui ira s'archiver sur le "fichier" des mouvements comptables.

S'il ne peut reconstituer le colis entierement, il met momentanement cette commande en suspend et poursuit avec les autres commandes du casier.

Quand toutes les commandes du casier ont ete passees en revue, il prend en charge la ou les commandes restees en suspend. Un magasinier, travaillant sous ses ordres, replace dans les rayons les marchandises restees dans le casier et, profitant de ce parcours en magasin, essaye de completer la ou les commandes incompletes.

Si le completage reussit totalement, le prepose a l'emballage se retrouve dans le cas normal et enclenche la facturation.

Si malgre ces recherches en rayons, le colis reste incomplet, il decide de l'envoyer tel quel au client mais de corriger a l'ecran les quantites reellement livrees.

Apres cette correction, il peut enclencher la facturation mais doit poursuivre par une serie d'operations rectificatrices:

- effectuer une mise a jour du stock pour le produit incompletement livre, c'est-a-dire:

  mettre a 0 la Quantite en stock
  ajouter a la Quantite differee, la difference entre la Quantite a livrer et la Quantite reellement livree.

- effectuer une mise a jour du "fichier" des Commandes differees par ajout d'un du
- avertir, par telephone, le gestionnaire des stocks, que certains stocks etaient inconsistants.

De toute facon, une transaction explicitant la modification effectuee est generee par l'ordinateur a destination du gestionnaire des stocks.

Annexe 2.

SPECIFICATION DES TRAITEMENTS ET DES DONNEES.

Avant de passer à la spécification globale du système, nous allons présenter la spécification détaillée des traitements et des données relatives à un sous-problème important : la construction d'une commande complétée.

A la page 181, on trouve l'arbre déductif inhérent à ce sous-problème.

Attirons l'attention sur les points suivants :

- on construit une commande complétée en ajoutant un numéro de commande et une date de commande (date à laquelle on a reçu le bon de commande) à la commande du client qui a été acceptée (commande valide);
- la commande du client est acceptée si elle est signée et si on a pu construire à partir des renseignements mentionnées :
  - + l'identification du client;
  - + toutes les lignes de commande.

A titre d'exemple, dans les pages qui suivent, nous allons présenter la spécification des traitements suivants (noeuds de l'arbre déductif) :

- construction-commande-valide;
- construction-identification-client;
- construction-lignes-produits et tous ses descendants.

Spécification du problème : CONSTRUCTION - COMMANDE - VALIDE (page 181)

résultat

Création d'une commande valide et d'une documentation client mise à jour à partir d'un bon de commande client et des documentations clients et produits.

types.

COM-CLI : bon de commande client

COM-VALIDE : bon de commande contenant des informations se trouvant sur le bon de commande client complétées, libellées correctement et nécessaires aux factures, justificatifs de différés et justificatifs d'épuisés.

IDENT-CLI-COM-CLI : identification du client figurant sur le bon de commande client.

IDENT-CLI-COM-VALIDE : identification du client figurant sur la commande valide.

LIGNES-CMDE-COM-CLI : lignes de commande figurant sur la commande valide.

LIGNES-CMDE-COM-VALIDE : lignes de commande figurant sur la commande valide.

SIGNATURE-COM-CLI : signature figurant sur le bon de commande client.

MONTANT-COM-CLI : montant figurant sur le bon de commande client.

DOC-CLIENTS : documentation possédée sur les clients

DOC-PRODUITS : documentation possédée sur les produits.

types

bc_j : COM-CLI

$bcval_k$: COM-VALIDE

identification-client-valide : IDENT-CLI-COM-VALIDE

lignes-cmde-client-valides : LIGNES-CMDE-COM-VALIDE

identification-client-commande : IDENT-CLI-COM-CLI

lignes-cmde-client-commande : LIGNES-CMDE-COM-CLI

montant-commande-client : MONTANT-COM-CLI

signature-commande-client : SIGNATURE-COM-CLI

doc.cli, doc.cli' : DOC. CLIENTS

doc.prod : DOC.PRODUITS

prédicats $\forall$: entier

commande-correcte : COM-CLI x DOC. CLIENTS x DOC. PRODUITS → booléen

client-identifiable : IDENT-CLI-COM-CLI x DOC. CLIENTS → booléen

lignes-produits-identifiables : LIGNES-CMDE-COM-CLI x DOC. PRODUITS → booléen

signature-identifiée : SIGNATURE-COM-CLI → booléen

résultats : externe : $bcval_k$ = (identification-client-valide, lignes-cmde-client-valides)

interne : doc.cli'

arguments : externe : bc_j = (identification-client-commande, lignes-cmde-client-commande, montant-commande-client, signature-commande-client)

tq

Commande correcte (bc_j, doc.cli, doc.prod)

interne : doc.cli, doc.prod.

Spécification de la relation résultats - arguments.

client-identifiable (bc_j, doc.cli) et (identification-client-valide, doc.cli') = CONSTRUCTION-IDENTIFICATION-CLIENT (identification-client-commande, doc.cli)

et

lignes-produits-identifiables (bc_j, doc.prod)

et (lignes-cmde-client-valides) = CONSTRUCTION-LIGNES-PRODUITS (lignes-cmde-client-commande, doc.prod) (*)

Structure des gardes.

Commande-correcte (bc_j, doc.cli, doc.prod) ≡ client identifiable (identification-client-commande, doc.cli)

Spécification du problème : CONSTRUCTION-COMMANDE-VALIDE (suite)

prédicats.

commande-correcte : prédicat précisant que le bon de commande est libellé d'une manière telle qu'il puisse être identifié.

client-identifiable : prédicat précisant que l'identification du client du bon de commande est libellée d'une manière telle que le client puisse être identifié.

lignes-produits-identifiables : prédicat précisant que toutes les lignes de commande sont libellées d'une manière telle qu'elles puissent être identifiées.

signature-identifiée : prédicat précisant que le bon de commande est signé.

montant-correct : prédicat précisant que la différence entre le montant se trouvant sur la commande du client et le montant calculé à partir des lignes de commande figurant sur la commande valide ne dépasse pas un certain seuil.

montant-correct : MONTANT-COM-CLI x LIGNES-CMDE-COM-VALIDE → booléen.

fonctions.

CONSTRUCTION-IDENTIFICATION-CLIENT : IDENT-CLI-COM-CLI x DOC.CLIENTS → IDENT-CLI-COM-VALIDE x DOC. CLIENTS

CONSTRUCTION-LIGNES-PRODUITS : LIGNES-CMDE-COM-CLI x DOC. PRODUITS → LIGNES-CMDE-COM-VALIDE

SIGNATURE : SIGNATURE-COM-CLI → booléen
de base dans COM-CLI

MONTANT : MONTANT-COM-CLI → entier
de base dans COM-CLI

MONTANT-CALCULE : LIGNES-CMDE-COM-VALIDE → entier
de base dans COM-VALIDE

et
lignes-produits-identifiables (lignes-cmde-client-commande, doc. prod)
et
signature-identifiée (signature-commande-client)
et
montant-correct (montant-commande-client, lignes-cmde-client-valides);
signature-identifiée (signature-(commande-client) ≡ SIGNATURE (signature-commande-client) = vrai;
montant-correct (montant-commande-client, lignes-cmde-client-valides) ≡
| MONTANT (montant-commande-client) - MONTANT CALCULE (lignes-cmde-client-valides) | < τ .

(*)
pour bc_j
il existe $\{bcval_k\}$ tq
il existe au plus 1
$bcval_k \in$ COM-VALIDE

Spécification du problème : CONSTRUCTION - IDENTIFICATION - CLIENT (page 181)

résultat

Création de la zone identification client sur la commande validée et mise à jour de la documentation client à partir de la zone identification client sur le bon de commande client et des documentations clients.

types

IDENT-CLI-COM-VALIDE : identification du client complète et correcte, contenant des informations nécessaires au bon de facturation, au bon de facturation différé et au justificatif de livraison différé.

IDENT-CLI-COM-CLI : identification du client figurant sur le bon de commande client.

DOC.CLIENTS : documentation possédée sur les clients.

prédicats.

client-identifiable : prédicat précisant que l'identification du client sur le bon de commande est libellé d'une manière telle que le client soit identifiable.

ancien-client : prédicat précisant que l'identification du client sur le bon de commande est libellé d'une manière telle que le client soit identifiable comme un client connu.

types.

identification-client-valide : IDENT-CLI-COM-VALIDE

identification-client-commande : IDENT-CLI-COM-CLI

doc.cli : DOC.CLIENTS

doc.cli : DOC.CLIENTS

prédicats

client-identifiable : IDENT-CLI-COM-CLI X DOC.CLIENTS → booléen

ancien-client : IDENT-CLI-COM-CLI X DOC.CLIENTS → booléen.

nouveau-client : IDENT-CLI-COM-CLI X DOC.CLIENTS → booléen

fonctions.

CONSTRUCTION-IDENTIFICATION-CLIENT-CONNU:
IDENT-CLI-COM-CLI X DOC.CLIENTS
→ IDENT-CLI-COM-VALIDE x DOC.CLIENTS

CONSTRUCTION-IDENTIFICATION-NOUVEAU-CLIENT:
IDENT-CLI-COM-CLI X DOC.CLIENTS
→ IDENT-CLI-COM-VALIDE x DOC.CLIENTS

résultats externe : identification-client-valide
interne : doc.cli'

arguments externe : identification-client-commande tq client-identifiable (identification-client-commande-doc.cli)
interne : doc.cli

Spécification de la relation résultats-arguments.

ancien-client (identification-client-commande-doc.cli)
et (identification-client-valide, doc.cli) =
CONSTRUCTION-IDENTIFICATION-CLIENT-CONNU (identification-client-commande-doc.cli)
ou ex
nouveau-client (identification-client-commande-doc.cli)
et (identification-client-valide, doc.cli) =
CONSTRUCTION-IDENTIFICATION-NOUVEAU-CLIENT (identification-client-commande-doc.cli)

Structure des gardes.

client-identifiable (identification-client-commande-doc.cli)
≡
ancien client (identification-client-commande-doc.cli)
ou ex
nouveau client (identification-client-commande-doc.cli)

Spécification du problème : CONSTRUCTION-IDENTIFICATION-CLIENT (suite)

nouveau-client : prédicat précisant que l'identification du client sur le bon de commande est libellé d'une manière telle que le client soit identifiable comme un nouveau client.		

Spécification du problème : CONSTRUCTION - LIGNES - PRODUITS

résultat.

Création des lignes de commande sur la commande validée à partir des lignes de commande du bon de commande client et de la documentation produits.

types.

LIGNES-CMDE-COM-VALIDES : lignes de commande figurant sur la commande validée.

LIGNE-CMDE-COM-VALIDE : une ligne de commande figurant sur la commande validée.

LIGNES-CMDE-COM-CLI : lignes de commande figurant sur le bon de commande du client.

LIGNE-CMDE-COM-CLI : ligne de commande figurant sur le bon de commande du client.

DOC. PRODUITS : documentation possédée sur les produits.

prédicats.

lignes-produits-identifiables : prédicat précisant que toutes les lignes du bon de commande sont libellées d'une manière telle qu'elles puissent être identifiées

ligne-produit-identifiable : prédicat précisant que la ligne du bon de commande est libellée de manière telle qu'elle puisse être identifiée.

types.

lignes-commande-client-validée :
LIGNES-CMDE-COM-VALIDES

ligne-commande-client-validée :
LIGNE-CMDE-COM-VALIDE

lignes-commande-client-commande : LIGNES-CMDE-COM-CLI

ligne-commande-client-commande :
LIGNE-CMDE-COM-CLI

doc. prod : DOC. PRODUITS

prédicats.

lignes-produits-identifiables :
LIGNES-CMDE-COM-CLI x DOC. PRODUITS
→ booléen

ligne-produit-identifiable :
LIGNE-CMDE-COM-CLI x DOC. PRODUITS
→ booléen

fonction.

CONSTRUCTION-LIGNE-PRODUIT :
LIGNE-CMDE-COM-CLI x DOC. PRODUITS
→ LIGNE-CMDE-COM-CLI

résultats externes : lignes-commandes-client-validées

interne : /

arguments externes : lignes-commande-client-commande = { ligne-commande-client-commande }

tq

lignes-produits-identifiables (lignes-commande-client-commande, doc. prod)

internes : doc. prod

Spécification de la relation résultats-arguments.

lignes-commande-client-validée =
{ ligne-commande-client-validée }*

tq

pour toute ligne-commande-client-validée **il existe une unique** ligne commande-client-commande

tq

ligne-produit-identifiable (ligne-commande-client-commande, doc. prod)

et

(ligne-commande-client-validée) =
CONSTRUCTION-LIGNE-PRODUIT (ligne-commande-client-commande, doc. prod)

Spécification du problème : CONSTRUCTION - LIGNES - PRODUITS (suite)

Structure des gardes.

lignes - produits - identifiables (lignes - commande - client - commande, doc. prod) ≡
∀ ligne - commande - client - commande :
ligne - produit - identifiable (ligne - commande - client. commande, doc. prod)

Spécification du problème : CONSTRUCTION - LIGNE - PRODUIT.

résultat.

Création d'une ligne de commande dont la commande valide à partir de la ligne de commande du bon de commande du client et de la documentation produits.

types.

LIGNE - CMDE - COM - VALIDE : ligne de commande figurant sur la commande valide.

LIBELLE - LIGNE - COM - VALIDE : libellé du produit de la ligne de commande figurant sur la commande valide.

NUMERO - LIGNE - COM - VALIDE : numéro du produit de la ligne de commande figurant sur la commande valide.

PRIX - UNITAIRE - LIGNE - COM - VALIDE : prix unitaire du produit de la ligne de commande figurant sur la commande valide

QUANTITE - CMDE - LIGNE - COM - VALIDE : quantité commandée du produit de la ligne de commande figurant sur la commande valide.

LIGNE - CMDE - COM - CLI : ligne de commande figurant sur le bon de commande.

LIBELLE - LIGNE - COM - CLI : libellé du produit de la ligne de commande du bon de commande.

NUMERO - LIGNE - COM - CLI : numéro du produit de la ligne de commande du bon de commande.

PRIX - UNITAIRE - LIGNE - COM - CLI : prix unitaire du produit de la ligne de commande figurant sur le bon de commande.

QUANTITE - CMDE - LIGNE - COM - CLI : quantité commandée du produit sur la ligne de commande figurant sur le bon de commande.

types.

ligne - commande - client - valide : LIGNE - CMDE - COM - VALIDE

libelle - produit - valide : LIBELLE - LIGNE - COM - VALIDE

numero - produit - valide : NUMERO - LIGNE - COM - VALIDE

prix-unitaire - valide : PRIX - UNITAIRE - LIGNE - COM - VALIDE

quantité-commande - valide : QUANTITE - CMDE - LIGNE - COM - VALIDE

ligne - commande - client - commande : LIGNE - CMDE - COM - CLI

libelle - produit - commande : LIBELLE - LIGNE - COM - CLI

numero - produit - commande : NUMERO - LIGNE - COM - CLI

prix-unitaire - produit - commande : PRIX - UNITAIRE - LIGNE - COM - CLI

quantité - commandée - produit - commande : QUANTITE - CMDE - LIGNE - COM - CLI

montant - ligne - commande : MONTANT - LIGNE - COM - CLI

doc. produits : DOC. PRODUITS

prédicats

ligne - produit - identifiable : LIGNE - CMDE - COM - CLI × DOC. PRODUITS → booléen

résultat externe : ligne - commande - client - valide

interne : /

arguments externes : ligne - commande - client - commande = (libelle - produit - commande, numero - produit - commande, prix - unitaire - produit - commande, quantité - commandée, produit - commande, montant - ligne - commande)

tq

ligne - produit identifiable (ligne - commande - client - commande, doc. prod)

internes : doc. produits

Spécification de la relation résultats - arguments.

ligne - commande - client - valide = (libelle - produit - valide, numero - produit - valide, prix-unitaire-valide, quantité - commandée - valide)

tq

libelle identifiable (libelle - produit - commande, doc. produits)

et

(libelle - produit - valide, numero - produit - valide, prix - unitaire - valide, quantité - commandée - valide) = CREATION - LIGNE - PRODUIT - LIBELLE - CONNU (libelle - produit - commande, doc. produits, quantité - commandée - commande)

ou ex

numero identifiable (libelle - produit - commande, numero - produit - commande, doc. produits)

Spécification du problème : CONSTRUCTION - LIGNE - PRODUIT (suite)

MONTANT-LIGNE-COM-CLI : montant de la ligne sur une ligne de commande figurant sur le bon de commande.
DOC. PRODUITS : documentation possédée sur les produits.

prédicats.

ligne-produit-identifiable : prédicat précisant que la ligne du bon de commande est libellée d'une manière telle qu'elle puisse être identifiée.
libelle-identifiable : prédicat précisant que la ligne du bon de commande est identifiable par la connaissance du libellé du produit sur lequel elle porte.
numero-identifiable : prédicat précisant que la ligne du bon de commande est identifiable par la connaissance du numéro du produit sur lequel elle porte.

libelle-identifiable : LIBELLE-LIGNE-COM-CLI x DOC. PRODUITS → booléen

numero-identifiable : LIBELLE-LIGNE-COM-CLI x NUMERO-LIGNE-COM-CLI x DOC. PRODUITS → booléen

fonctions.

CONNU:
CREATION-LIGNE-PRODUIT-LIBELLE-LIBELLE-LIGNE-COM-CLI x QUANTITE-CMDE-LIGNE-COM-CLI x DOC. PRODUITS → voir (*)

CREATION-LIGNE-PRODUIT-NUMERO-CONNU : NUMERO-LIGNE-COM-CLI x QUANTITE-CMDE-LIGNE-COM-CLI x DOC. PRODUITS → (*) LIBELLE-LIGNE-COM-VALIDE x NUMERO-LIGNE-COM-VALIDE x PRIX UNITAIRE-LIGNE-COM-VALIDE x QUANTITE-CMDE-LIGNE-COM-VALIDE

QUANTITE-COMMANDEE-EXISTANTE-LIGNE-COM-CLI : QUANTITE-CMDE-LIGNE-COM-CLI → booléen
de base dans QUANTITE-CMDE-LIGNE-COM-CLI

et
(libelle-produit-valide, numero-produit-valide, prix-unitaire-valide, quantité-commandée-valide) = CREATION-LIGNE-PRODUIT-NUMERO-CONNU (numéro-produit-commande, doc. produits, quantité-commandée-commande)

Structure des gardes.

ligne-produit-identifiable (ligne-commande-client-commande, doc. prod)

≡

[libelle-identifiable (libelle-produit-commande, numero-produit-commande, doc. produits)

∨ numero-identifiable (libelle-produit-commande, numero-produit-commande, doc. produits)

]

∧ QUANTITE-COMMANDEE-EXISTANTE-LIGNE-COM-CLI (quantité commandée-commande)

Spécification du problème : CREATION - LIGNE - PRODUIT - LIBELLE - CONNU .

résultat.

Création du libellé - produit, du numéro - produit, du prix - unitaire du produit et de la quantité commandée du produit sur la commande valide à partir du libellé du produit et de la quantité commandée figurant sur le bon de commande client et de la documentation produits.

types -

LIBELLE - LIGNE - COM - VALIDE : libellé du produit de la ligne de commande figurant sur la commande valide.

NUMERO - LIGNE - COM - VALIDE : numéro du produit de la ligne de commande figurant sur la commande valide.

PRIX - UNITAIRE - LIGNE - COM - VALIDE : prix unitaire du produit de la ligne de commande figurant sur la commande valide

QUANTITE - CMDE - LIGNE - COM - VALIDE : quantité commandée du produit de la ligne de commande figurant sur la commande valide

LIBELLE - LIGNE - COM - CLI : libellé du produit de la ligne de commande figurant sur le bon de commande.

QUANTITE - CMDE - LIGNE - COM - CLI : quantité commandée du produit de la ligne de commande figurant sur le bon de commande.

types.

libelle - produit - valide : LIBELLE - LIGNE - COM - VALIDE

numero - produit - valide : NUMERO - LIGNE - COM - VALIDE

prix - unitaire - valide : PRIX - UNITAIRE - LIGNE - COM - VALIDE

quantité - commandée - valide : QUANTITE - CMDE - LIGNE - COM - VALIDE

libelle - produit - commandé : LIBELLE - LIGNE - COM - CLI

quantité - commandée - commande : QUANTITE - CMDE - LIGNE - COM - CLI

doc. produits : DOC. PRODUITS

prédicat.

libelle - identifiable : LIBELLE - LIGNE - COM - CLI x DOC. PRODUITS → booléen.

fonctions.

CREATION - LIBELLE - VALIDE - LIBELLE - CONNU : LIBELLE - LIGNE - COM - CLI → LIBELLE - LIGNE - COM - VALIDE

de base dans LIBELLE - LIGNE - COM - VALIDE

CREATION - NUMERO - VALIDE - LIBELLE - CONNU : LIBELLE - LIGNE - COM - CLI x DOC. PRODUITS → NUMERO - LIGNE - COM - VALIDE

de base dans NUMERO - LIGNE - COM - VALIDE

résultats externes : libelle - produit - valide, numero - produit - valide, prix unitaire - valide, quantité - commandée - valide.

interne : /

arguments externes : libelle - produit - commande, quantité - commandée - commande

Kq

libelle identifiable (libelle - produit - commande, doc. produits)

interne : doc. produits

Spécification de la relation résultats - arguments

(libelle - produit - valide) = CREATION - LIBELLE - VALIDE - LIBELLE - CONNU (libelle - produit - commande)

et

(numéro - produit - valide) = CREATION - NUMERO - VALIDE - LIBELLE - CONNU (libelle - produit - commande, doc. produits)

et (prix - unitaire - valide) = CREATION - PRIX - UNITAIRE - VALIDE - LIBELLE - CONNU (libelle - produit - commande, doc. produits)

et (quantité commandée valide) = CREATION - QUANTITE - COMMANDEE - VALIDE - LIBELLE - CONNU (libelle - produit - commande, doc. produits, quantité commandée - commande)

Structure des gardes.

libelle - identifiable (libelle - produit - commande, doc. prod)

Spécification du problème : CREATION-LIGNE-PRODUIT-LIBELLE-CONNU (suite)

prédicat. libelle-identifiable : prédicat précisant que la ligne du bon de commande est identifiable par la connaissance du libellé du produit sur lequel elle porte.	CREATION-PRIX-UNITAIRE-VALIDE-LIBELLE-CONNU : LIBELLE-LIGNE-COM-CLI x DOC.PRODUITS → PRIX-UNITAIRE-LIGNE-COM-VALIDE de base dans PRIX-UNITAIRE-LIGNE-COM-VALIDE CREATION-QUANTITE-COMMANDE-VALIDE-LIBELLE-CONNU : LIBELLE-LIGNE-COM-CLI x DOC.PRODUITS x QUANTITE-CMDE-LIGNE-COM-CLI → QUANTITE-CMDE-LIGNE-COM-VALIDE de base dans QUANTITE-LIGNE-COM-VALIDE LIBELLE-PRODUIT-EXISTANT-LIGNE-COM-CLI : LIBELLE-LIGNE-COM-CLI → booléen de base dans LIBELLE-LIGNE-COM-CLI LIBELLE-PRODUIT-EXISTANT-DOC-PROD : DOC.PRODUITS x LIBELLE-LIGNE-COM-CLI → booléen de base dans DOC.PRODUITS	≡ [LIBELLE-PRODUIT-EXISTANT-LIGNE-COM-CLI (libelle-produit-commande) ∧ LIBELLE-PRODUIT-EXISTANT-DOC-PROD (libelle-produit-commande, doc.produits)]

Spécification du problème : CREATION - LIGNE - PRODUIT - NUMERO - CONNU.

résultat

Création du libelle - produit, du numero - produit, du prix - unitaire - produit, de la quantité - commandée produit sur la commande valide à partir du numero - produit et de la quantité - commandée figurant sur le bon de commande client et de la documentation produits.

types

LIBELLE-LIGNE-COM-VALIDE : libellé du produit figurant sur la ligne de commande de la commande valide.

NUMERO-LIGNE-COM-VALIDE : numéro du produit figurant sur la ligne de commande de la commande valide.

PRIX-UNITAIRE-LIGNE-COM-VALIDE : prix unitaire du produit figurant sur la ligne de commande de la commande valide.

QUANTITE-CMDE-LIGNE-COM-VALIDE : quantité commandée du produit figurant sur la ligne de commande de la commande valide.

NUMERO-LIGNE-COM-CLI : numéro du produit figurant sur la ligne de commande du bon de commande.

LIBELLE-LIGNE-COM-CLI : libellé du produit figurant sur la ligne de commande du bon de commande.

QUANTITE-CMDE-LIGNE-COM-CLI : quantité commandée du produit figurant sur la ligne de commande du bon de commande.

types.

libelle - produit - valide : LIBELLE - LIGNE - COM - VALIDE

numero - produit - valide : NUMERO - LIGNE - COM - VALIDE

prix - unitaire - valide : PRIX - UNITAIRE - LIGNE - COM - VALIDE

quantité commandée - valide : QUANTITE-CMDE-LIGNE-COM-VALIDE

numero - produit - commande NUMERO-LIGNE - COM-CLI

libelle - produit - commande : LIBELLE-LIGNE - COM-CLI

quantité-commandée-commande : QUANTITE - CMDE-LIGNE - COM-CLI

doc. produits : DOC. PRODUITS

prédicat.

numero - identifiable : NUMERO-LIGNE - COM-CLI x LIBELLE-LIGNE-COM - VALIDE x DOC. PRODUITS → booléen

fonctions.

CREATION - LIBELLE-VALIDE - NUMERO - CONNU : NUMERO-LIGNE-COM - CLI x DOC. PRODUITS → LIBELLE-LIGNE-COM - VALIDE

de base dans LIBELLE_LIGNE-COM-VALIDE

CREATION - NUMERO-VALIDE - NUMERO CONNU : NUMERO-LIGNE-COM-CLI → NUMERO-LIGNE-COM-VALIDE

de base dans NUMERO-LIGNE-COM-VALIDE

résultats externes : libelle - produit - valide, numero - produit - valide, prix - unitaire - valide, quantité - commandée - valide

internes : /

arguments externes : numero - produit - commande, libelle - produit commande, quantité-commandée - commande

tq numero - identifiable (libelle - produit-commande, numero - produit - commande, doc. produits)

internes : doc. produits.

Spécification de la relation résultats - arguments

(libelle - produit - valide) = CREATION - LIBELLE-VALIDE - NUMERO - CONNU (numero - produit - commande, doc. produits)

et

(numero - produit - valide) = CREATION - NUMERO - VALIDE - NUMERO - CONNU (numero-produit-commande)

et (prix-unitaire - produit - valide) = CREATION - PRIX - UNITAIRE - VALIDE - NUMERO - CONNU (numero - produit - commande, doc. produits)

et

(quantité commandée valide) = CREATION - QUANTITE - COMMANDEE - VALIDE - NUMERO - CONNU (numero - produit - commande - quantité - commandée - commande, doc. produits)

Spécification du problème : CREATION - LIGNE - PRODUIT - NUMERO - CONNU (suite)

DOC. PRODUITS : documentation possédée sur les produits.

prédicat.

numero - identifiable : prédicat précisant que la ligne de bon de commande est identifiable par la connaissance du numéro du produit sur lequel elle porte.

CREATION - PRIX - UNITAIRE - VALIDE - NUMERO - LIGNE - COM - CLI x DOC. PRODUITS → PRIX - UNITAIRE - LIGNE - COM - VALIDE

de base dans PRIX - UNITAIRE - LIGNE - COM - VALIDE

CREATION - QUANTITE - COMMANDEE - VALIDE - NUMERO - CONNU : NUMERO - LIGNE - COM - CLI x DOC. PRODUITS x QUANTITE - CMDE - LIGNE - COM - CLI ⟶ QUANTITE - CMDE - LIGNE - COM - VALIDE

de base dans QUANTITE - CMDE - LIGNE - COM - VALIDE

NUMERO - PRODUIT - EXISTANT - COM - CLI : NUMERO - LIGNE - COM - CLI → booléen.

de base dans NUMERO - LIGNE - COM - CLI

NUMERO - PRODUIT - EXISTANT - DOC. PROD : NUMERO - LIGNE - COM - CLI x DOC. PROD → booléen

de base dans DOC. PRODUITS

LIBELLE - PRODUIT - EXISTANT - LIGNE - COM - CLI : LIBELLE - LIGNE - COM - CLI → booléen

de base dans LIBELLE - LIGNE - COM - CLI

Structure de la garde.

numero - identifiable (libelle - produit - commande, numero - produit - commande, doc. produits)

≡

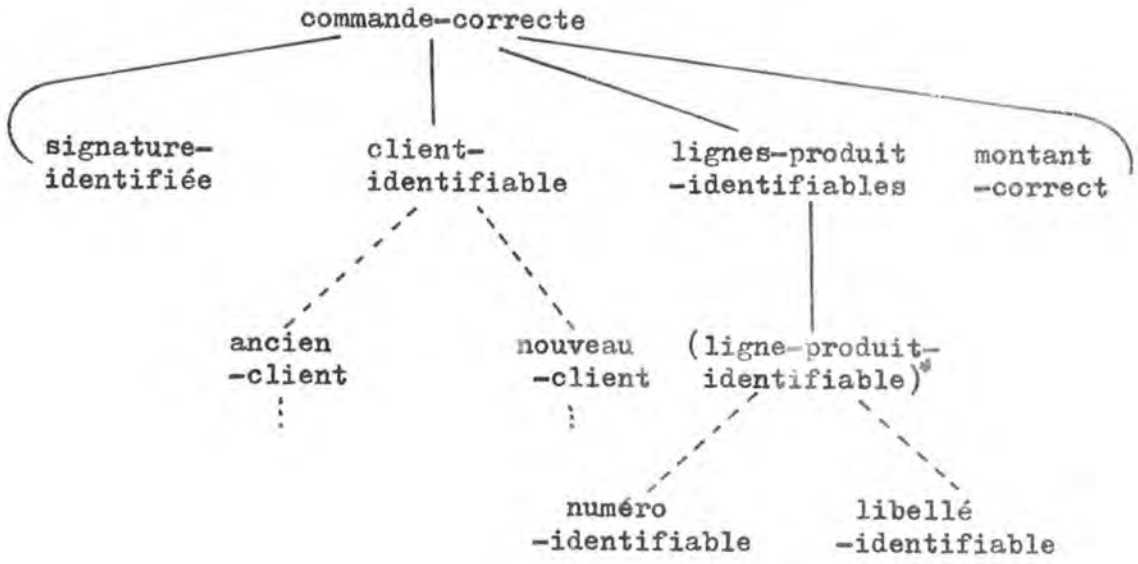
[NUMERO - PRODUIT - EXISTANT - LIGNE - COM - CLI (numero - produit - commande)

∧ non LIBELLE - PRODUIT - EXISTANT - LIGNE - COM - CLI (libelle - produit - commande)

∧ NUMERO - PRODUIT - EXISTANT - DOC. PROD (doc. produits, numero - produit - commande)

]

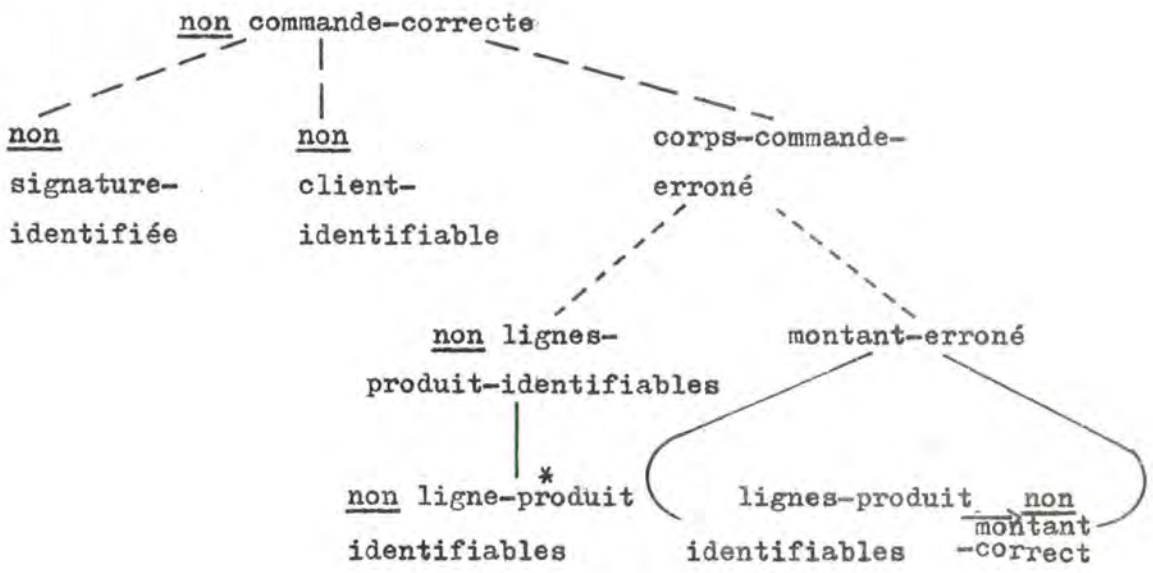
Concernant la spécification de la vérification du bon de commande, nous avons introduit pour certaines fonctions (terminales ou non) un ensemble de gardes. Rappelons la structure de cet ensemble :



A partir de cette arborescence, on peut faire un certain nombre de contrôles de cohérence et de complétude tels que :

- vérifier si toutes les feuilles de l'arborescence sont bien explicitées à l'aide de fonctions terminales ("de base") à résultats booléens.
- s'assurer que l'ensemble des cas a bien été traité en vérifiant si pour chaque garde introduite la négation de cette garde a été également traitée.

Pour une fonction de non-validation de la commande du client, la structure des gardes est la suivante :



Il faut vérifier que pour toute garde, on a traité sa négation :

+ soit directement; c'est-à-dire que pour une garde X, on trouve la garde non X.

  Exemple. commande-correcte, non commande-correcte

+ soit indirectement; c'est-à-dire que pour une garde X, on trouve :

  - une garde Y se structurant en gardes dont l'une est X pour laquelle non Y existe (car alors $\neg Y \Rightarrow \neg X$).

    Exemple. corps-commande-erroné appartient à non commande-correcte pour lequel existe commande-correcte.

  - une garde X se structurant en Y_1, Y_2, ..., Y_m et pour lesquelles non Y_1, non Y_2, ..., non Y_m existent.

    Exemple. montant-erroné se structure en lignes-produits-identifiables et en non montant-correct pour lesquelles ont été traitées non lignes-produit-identifiables et montant-correct.

- vérifier la cohérence d'une garde non-terminale en exprimant toutes les gardes terminales qui la compose sous la forme de table de décision.

Exemple. une ligne de commande est identifiable si :

libellé-produit-existant-ligne-com-cli	vrai	faux
libellé-produit-existant-doc-prod	vrai	-
numéro-produit-existant-ligne-com-cli	-	vrai
numéro-produit-existant-doc-prod	-	vrai
quantité-commandée-existant-ligne-com-cli	vrai	vrai

Après avoir donné un exemple de spécification de traitements, nous pouvons en présenter un de description de données. Il s'agit de la spécification de la donnée de départ COM-CLI (bon de commande du client). La signification des divers éléments trouvés est donnée à la page 62.

S.I. COM-CLI

Types

COM-CLI : bon de commande reçu du client.

IDENT-CLI-COM-CLI : identification du client figurant sur le bon de commande.

NOM-CLI-COM-CLI : nom sous lequel le client est connu.

PRENOM-CLI-COM-CLI : prénom permettant de distinguer le client d'un autre de même nom.

ADRESSE-CLI-COM-CLI : adresse à laquelle la firme enverra les différents documents et le colis.

NUMERO-CLI-COM-CLI : code attribué par la firme à un tiers qui a déjà passé commande.

LIGNES-CMDE-COM-CLI : ensemble des lignes de commandes passées par le client.

LIGNE-CMDE-COM-CLI : une des lignes de commande figurant sur le bon de commande du client.

LIBELLE-LIGNE-COM-CLI : nom sous lequel le produit est connu de la firme et du client.

NUMERO-LIGNE-COM-CLI : numéro sous lequel le produit est connu de la firme et du client.

PRIX-UNITAIRE-LIGNE-COM-CLI : prix d'une unité de produit.

QUANTITE-CMDE-LIGNE-COM-CLI : quantité du produit commandé.

MONTANT-LIGNE-COM-CLI : prix calculé par le client pour une ligne de sa commande.

MONTANT-COM-CLI : prix calculé par le client en échange du colis commandé.

SIGNATURE-COM-CLI : signature du client.

Types

ligne-cmde-com-cli : LIGNE-CMDE-COM-CLI

libelle-ligne-com-cli : LIBELLE-LIGNE-COM-CLI

numero-ligne-com-cli : NUMERO-LIGNE-COM-CLI

signature-commande-client : SIGNATURE-COM-CLI

montant-commande-client : MONTANT-COM-CLI

nom-client-commande : NOM-CLI-COM-CLI

prenom-client-commande : PRENOM-CLI-COM-CLI

adresse-client-commande : ADRESSE-CLI-COM-CLI

numero-client-commande : NUMERO-CLI-COM-CLI

quantité-commande-commande : QUANTITE-CMDE-LIGNE-COM-CLI

libelle-ligne-commande-client : LIBELLE-LIGNE-COM-CLI

numero-ligne-commande-client : NUMERO-LIGNE-COM-CLI

lignes-client-commande : LIGNES-CMDE-COM-CLI

libelle-client-commande : LIBELLE-LIGNE-COM-CLI

COM-CLI

=

(IDENT-CLI-COM-CLI
  (NOM-CLI-COM-CLI : chaîne alphabétique [30];
  PRENOM-CLI-COM-CLI : chaîne alphabétique [15];
  ADRESSE-CLI-COM-CLI : chaîne alphanumérique [60];
  NUMERO-CLI-COM-CLI : chaîne numérique [6])
LIGNES-CMDE-COM-CLI
  (LIGNE-CMDE-COM-CLI
  (LIBELLE-LIGNE-COM-CLI : chaîne alphabétique [50];
  NUMERO-LIGNE-COM-CLI : chaîne numérique [8];
  PRIX-UNITAIRE-LIGNE-COM-CLI : chaîne numérique [5];
  QUANTITE-CMDE-LIGNE-COM-CLI : chaîne numérique [2];
  MONTANT-LIGNE-COM-CLI : chaîne numérique [6]))*
MONTANT-COM-CLI : chaîne numérique [6]
SIGNATURE-COM-CLI : chaîne alphanumérique [30])

invariants.

- $0 \leq \#\{\text{ligne-cmde-com-cli}\} \leq 15$
- $\forall i, j : i \neq j : \text{ligne-cmde-com-cli}_i,$ ligne-cmde-com-cli$_j \in$ LIGNE-CMDE-COM-CLI :
  libelle-ligne-com-cli$_i$ ($x_i \neq \omega$) $\neq$ libelle-ligne-com-cli$_j$ ($x_j \neq \omega$)

et

numero-ligne-com-cli$_i$ ($x_i \neq \omega$) $\neq$ numero-ligne-com-cli$_j$ ($x_j \neq \omega$)

invariants.

La commande du client ne peut contenir plus de quinze lignes de commande.
Deux lignes de commande distinctes ne peuvent porter sur un même produit.

fonctions.

SIGNATURE : fonction associant à la signature un booléen indiquant si elle est présente ou pas.
MONTANT : fonction dont le résultat entier indique le montant inscrit par le client sur sa commande. S'il n'a rien indiqué, on considère qu'il vaut $\emptyset$.
NOM-CLIENT-EXISTANT-COM-CLI : fonction dont le résultat booléen indique si le nom du client est inscrit.
PRENOM-CLIENT-EXISTANT-COM-CLI : fonction dont le résultat booléen indique si le prénom du client est inscrit.
ADRESSE-CLIENT-EXISTANT-COM-CLI : fonction dont le résultat booléen indique si l'adresse du client est inscrite.
NUMERO-CLIENT-EXISTANT-COM-CLI : fonction dont le résultat booléen indique si le numéro du client est inscrit.
QUANTITE-COMMANDEE-LIGNE-COMMANDE : fonction dont le résultat booléen indique si la quantité commandée est inscrite

fonctions.

SIGNATURE : SIGNATURE-COM-CLI → booléen

MONTANT : MONTANT-COM-CLI → entier

NOM-CLIENT-EXISTANT-COM-CLI : NOM-CLI-COM-CLI → booléen

PRENOM-CLIENT-EXISTANT-COM-CLI : PRENOM-CLI-COM-CLI → booléen.

ADRESSE-CLIENT-EXISTANT-COM-CLI : ADRESSE-CLI-COM-CLI → booléen.

NUMERO-CLIENT-EXISTANT-COM-CLI : NUMERO-CLI-COM-CLI → booléen

fonctions

de consultation.

SIGNATURE (signature-commande-client) booléen
tq bool = si signature-commande-client ≠ ω
alors vrai
sinon faux.

MONTANT (montant-commande-client) entier
tq entier = si montant-commande-client ≠ $\emptyset$
alors montant-commande-client
sinon $\emptyset$

NOM-CLIENT-EXISTANT-COM-CLI (nom-client commande) bool
tq bool = si nom-client-commande ≠ ω
alors vrai
sinon faux.

PRENOM-CLIENT-EXISTANT-COM-CLI (prénom-client-commande) bool
tq bool = si prénom-client-commande ≠ ω
alors vrai
sinon faux

ADRESSE-CLIENT-EXISTANT-COM-CLI (adresse-client-commande) bool
tq bool = si adresse-client-commande ≠ ω
alors vrai
sinon faux.

NUMERO-CLIENT-EXISTANT-COM-CLI (numero-client-commande) bool
tq bool = si numero-client-commande ≠ ω
alors vrai
sinon faux.

	QUANTITE-COMMANDEE-LIGNE-COMMANDE : QUANTITE-CMDE-LIGNE-COM-CLI → booléen	QUANTITE-COMMANDEE-LIGNE-COMMANDE (quantité-commandée-commande) bool tq bool = si quantité-commandée-commande ≠ ω alors vrai sinon faux.
LIBELLE-PRODUIT-EXISTANT-LIGNE-COM-CLI : fonction dont le résultat booléen indique si le libellé du produit est inscrit.	LIBELLE-PRODUIT-EXISTANT-LIGNE-COM-CLI : LIBELLE-LIGNE-COM-CLI → booléen	LIBELLE-PRODUIT-EXISTANT-LIGNE-COM-CLI (libelle-ligne-commande-client) bool tq bool = si libelle-ligne-commande-client ≠ ω alors vrai sinon faux.
NUMERO-PRODUIT-EXISTANT-LIGNE-COM-CLI : fonction dont le résultat booléen indique si le numéro du produit est inscrit.	NUMERO-PRODUIT-EXISTANT-LIGNE-COM-CLI : NUMERO-LIGNE-COM-CLI → booléen	NUMERO-PRODUIT-EXISTANT-LIGNE-COM-CLI (numero-ligne-commande-client) bool tq bool = si numero-ligne-commande-client ≠ ω alors vrai sinon faux.
ACCES-... : fonction indiquant que la donnée à la suite d'accès est consultée en vue de son utilisation dans un autre pavé.	ACCES-NOM-CLI-COM-CLI : → NOM-CLI-COM-CLI	ACCES-NOM-CLI-COM-CLI ( ) nom-client-commande tq post : nom-client-commande
	ACCES-PRENOM-CLI-COM-CLI : → PRENOM-CLI-COM-CLI	ACCES-PRENOM-CLI-COM-CLI ( ) prénom-client-commande tq post : prénom-client-commande
	ACCES-ADRESSE-CLI-COM-CLI : → ADRESSE-CLI-COM-CLI	ACCES-ADRESSE-CLI-COM-CLI ( ) adresse-client-commande tq post : adresse-client-commande
	ACCES-NUMERO-CLI-COM-CLI : → NUMERO-CLI-COM-CLI	ACCES-NUMERO-CLI-COM-CLI ( ) numero-client-commande tq post : numero-client-commande

ACCES - LIGNES - CMDE - COM - CLI :
→ LIGNES - CMDE - COM - CLI

ACCES - LIBELLE - LIGNE - COM - CLI :
→ LIBELLE - LIGNE - COM - CLI

ACCES - NUMERO - LIGNE - COM - CLI :
→ NUMERO - LIGNE - COM - CLI

ACCES - MONTANT - COM - CLI :
→ MONTANT - COM - CLI

ACCES - QUANTITE - CMDE - LIGNE - COM - CLI :
→ QUANTITE - CMDE - LIGNE - COM - CLI

ACCES - LIGNES - CMDE - COM - CLI ( ) lignes - client - commande
tq post : lignes - client - commande

ACCES - LIBELLE - LIGNE - COM - CLI ( ) libelle - client - commande
tq post : libelle - client - commande

ACCES - NUMERO - LIGNE - COM - CLI ( ) numero - client - commande
tq post : numero - client - commande

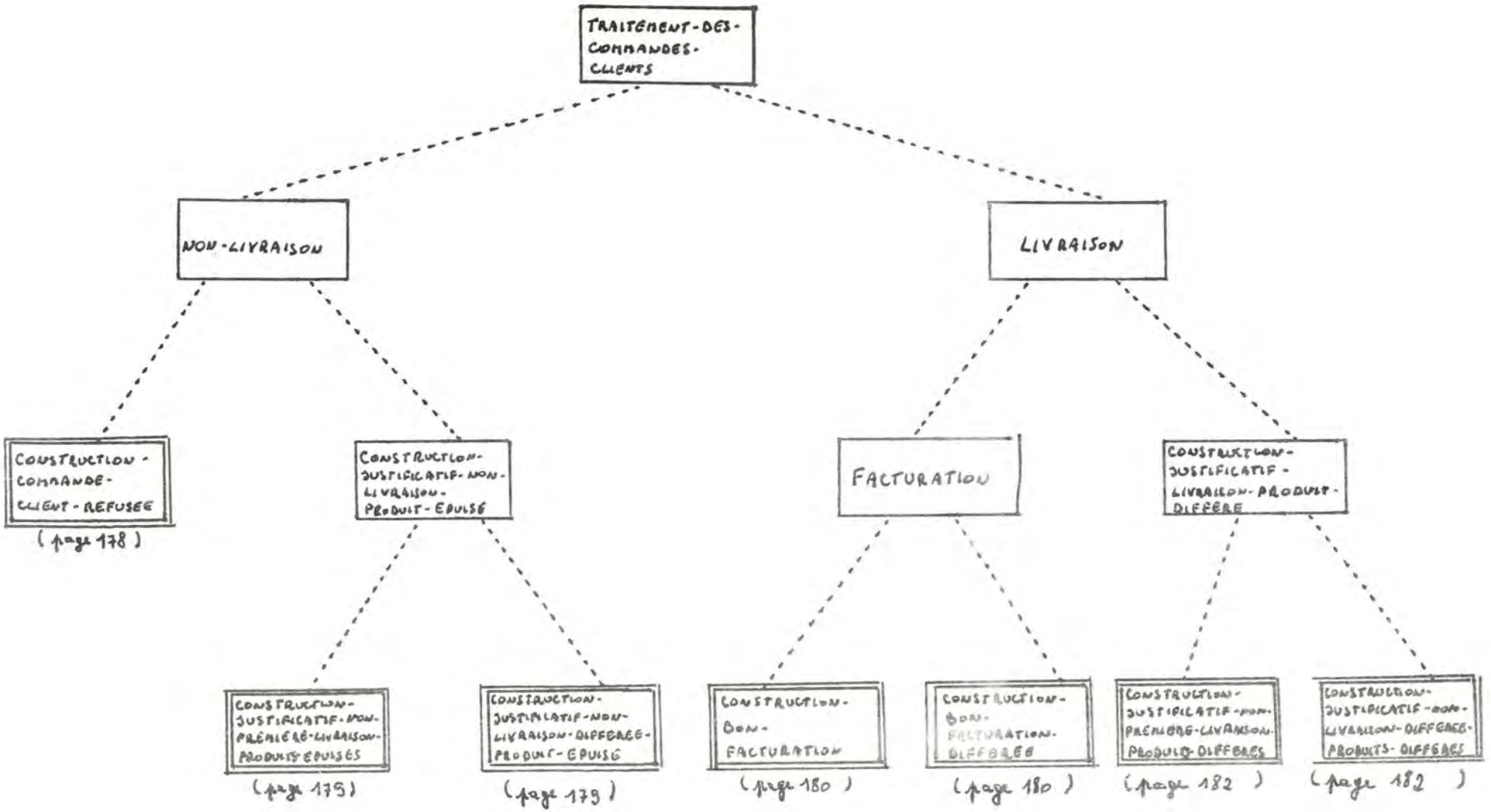
ACCES - MONTANT - COM - CLI ( ) montant - commande - client
tq post : montant - commande - client.

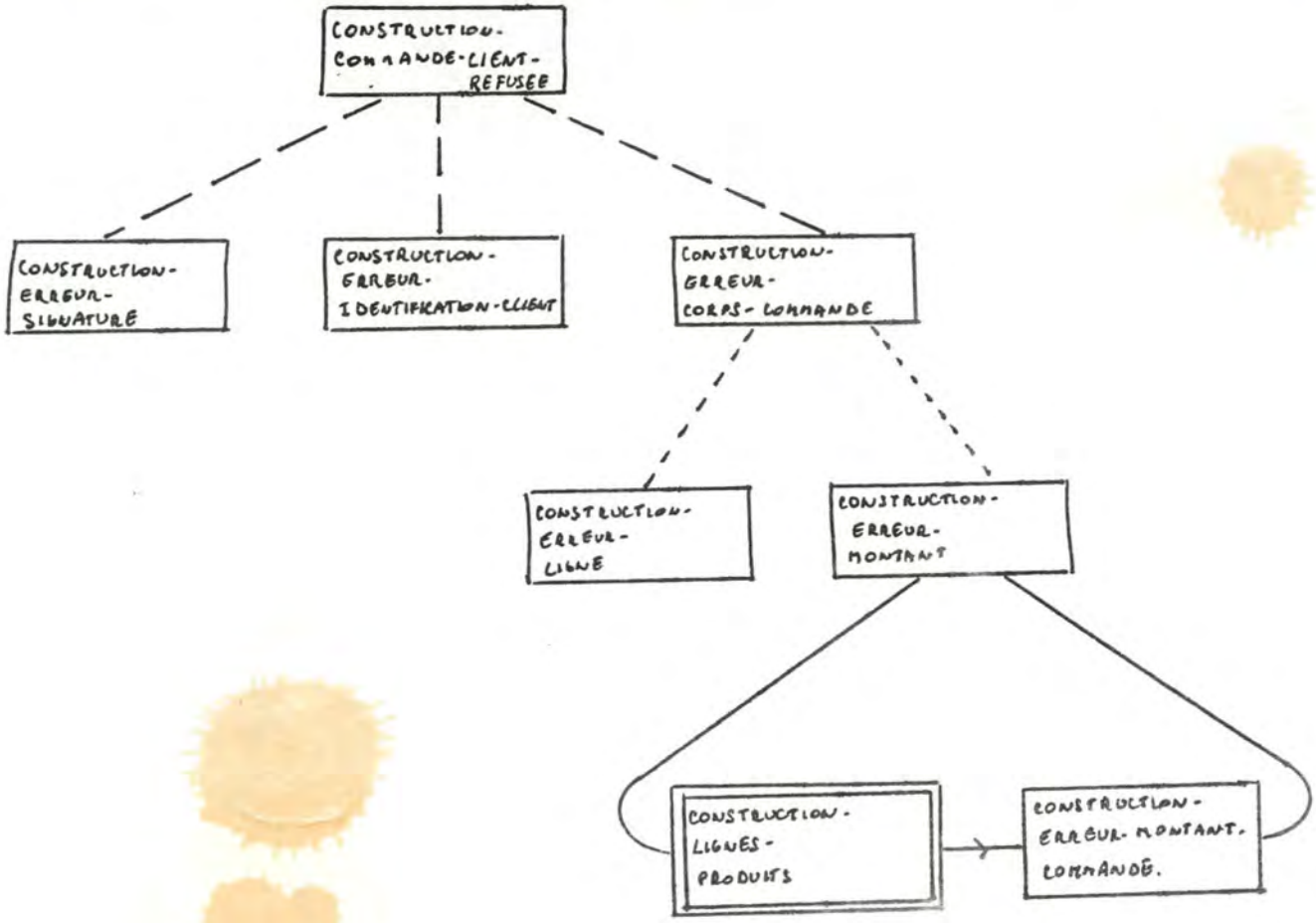
ACCES - QUANTITE - CMDE - LIGNE - COM - CLI ( ) quantité - commandée - commande
tq post : quantité - commandée - commande

Après avoir traité un sous-problème particulier, nous allons présenter l'arbre déductif relatif à l'ensemble de l'application traitée.

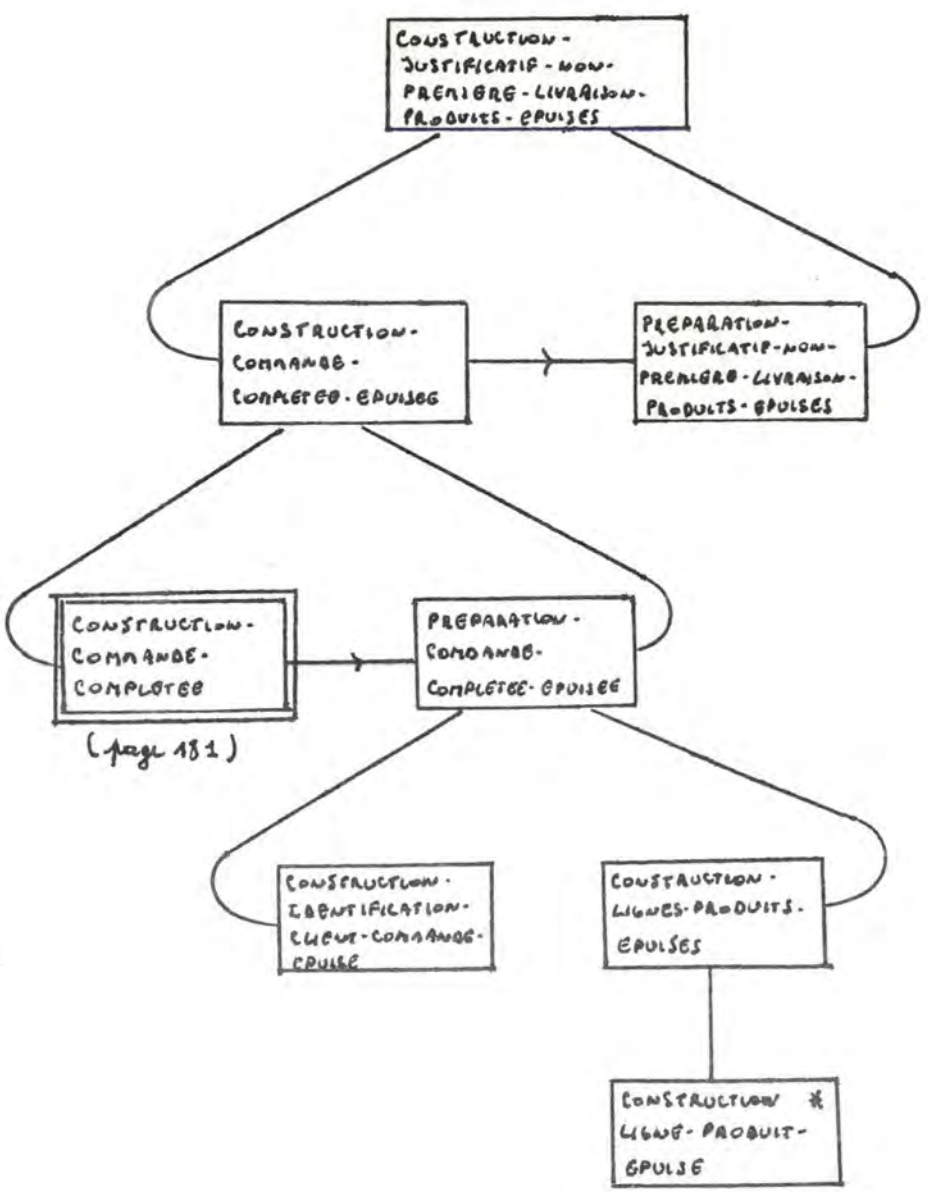
Il est à remarquer dans la description qui suit que :

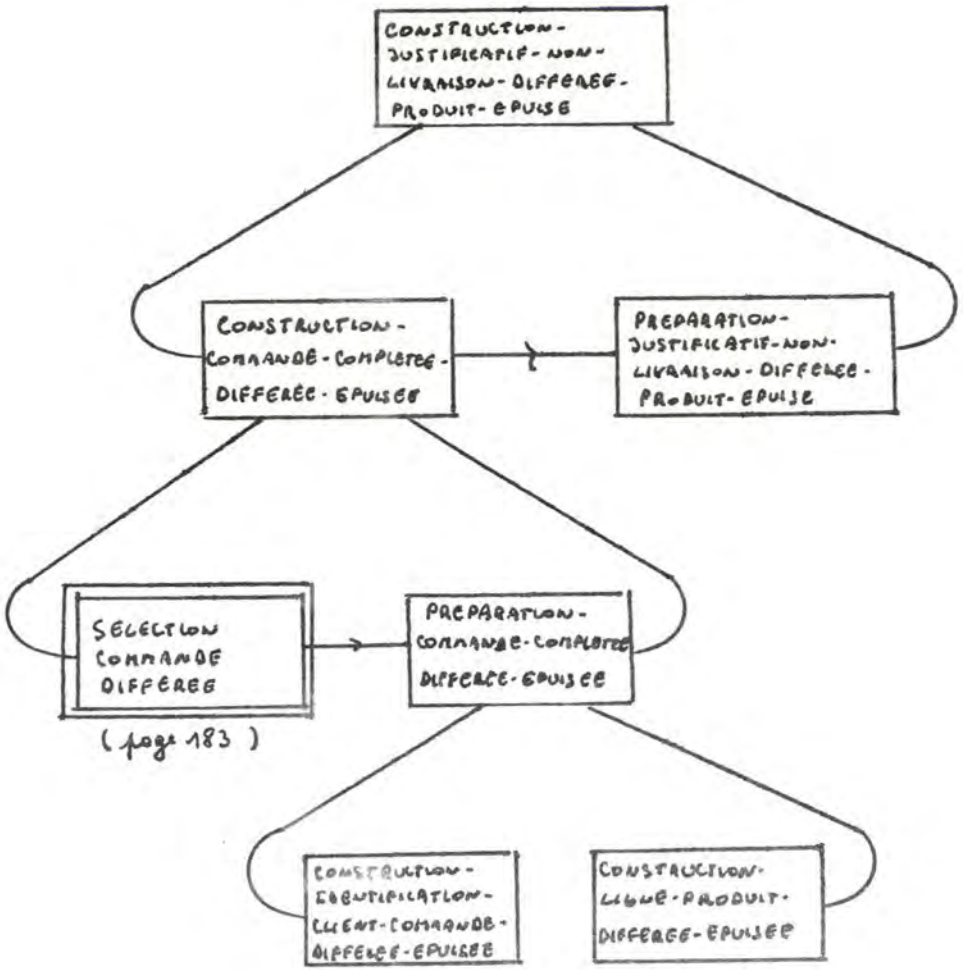
- au niveau de la distinction entre "non-livraison" et "livraison" (page 177), nous avons décidé d'étudier le cas des justificatifs pour produits différés dans le cadre d'une "livraison". Ce choix arbitraire a été motivé par le fait que, dans la plupart des cas, le produit en quantité insuffisante dans le stock sera plus tard déclaré réapprovisionné plutôt qu'épuisé.
- concernant la "sélection-commande-différée" (page 183), entre la construction d'une commande complétée différée et la sélection proprement dite d'une commande différée, nous sommes amenés à introduire une documentation sur les commandes différées. En effet, celle-ci est rendue nécessaire par le fait que la commande complétée différée doit être mémorisée entre plusieurs activations du traitement de sélection car elle ne sera traitée que lorsque l'une de ses lignes portera sur un produit déclaré réapprovisionné ou épuisé par l'application de gestion des commandes fournisseurs.

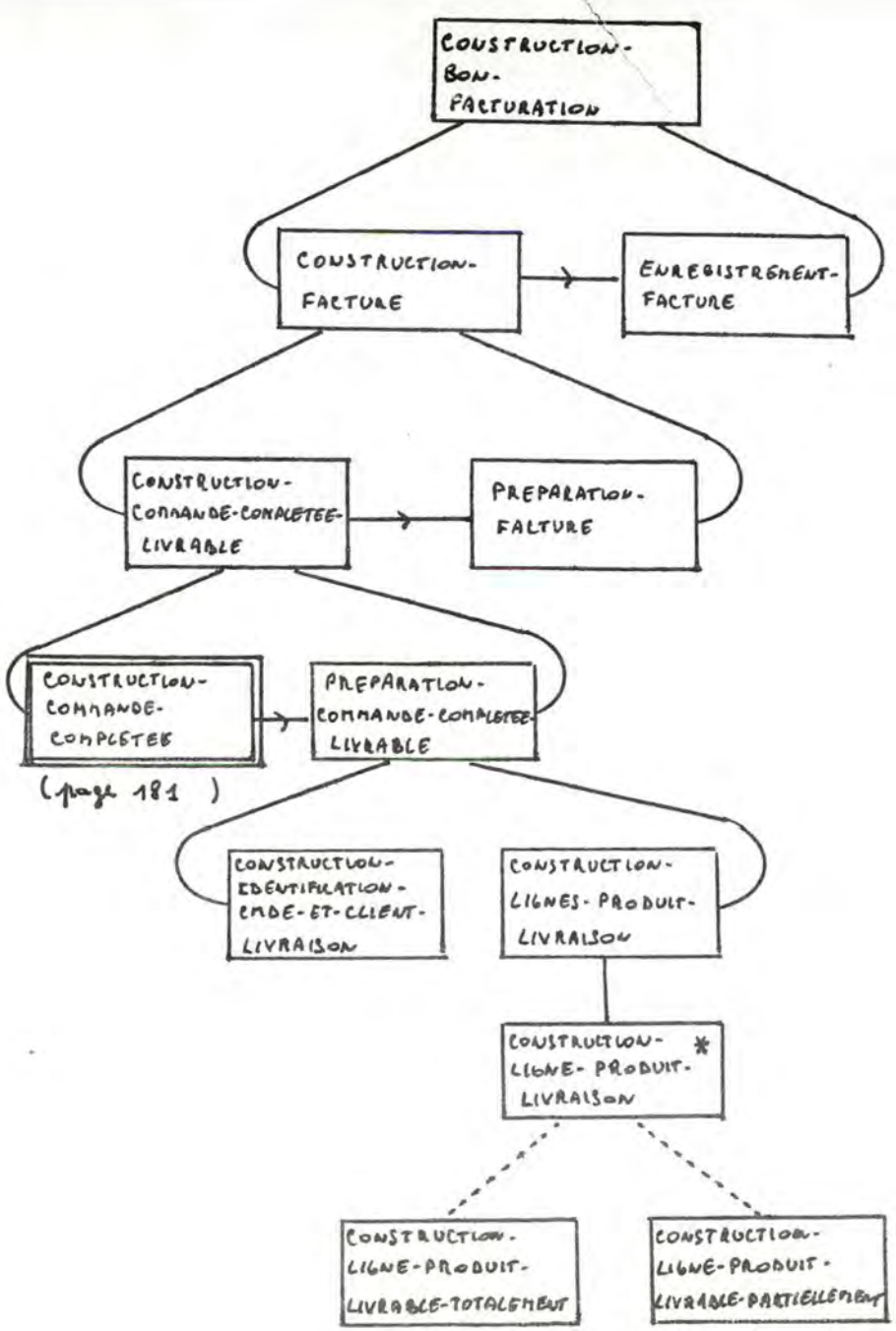

TRAITEMENT-DES-COMMANDES-CLIENTS
NON-LIVRAISON
LIVRAISON
CONSTRUCTION-COMMANDE-CLIENT-REFUSEE
(page 178)
CONSTRUCTION-JUSTIFICATIF-NON-LIVRAISON-PRODUIT-EPUISE
FACTURATION
CONSTRUCTION-JUSTIFICATIF-LIVRAISON-PRODUIT-DIFFERE
CONSTRUCTION-JUSTIFICATIF-NON-PREMIERE-LIVRAISON-PRODUITS-EPUISES
(page 179)
CONSTRUCTION-JUSTIFICATIF-NON-LIVRAISON-DIFFEREE-PRODUIT-EPUISE
(page 179)
CONSTRUCTION-BON-FACTURATION
(page 180)
CONSTRUCTION-BON-FACTURATION-DIFFEREE
(page 180)
CONSTRUCTION-JUSTIFICATIF-NON-PREMIERE-LIVRAISON-PRODUITS-DIFFERES
(page 182)
CONSTRUCTION-JUSTIFICATIF-NON-LIVRAISON-DIFFEREE-PRODUITS-DIFFERES
(page 182)

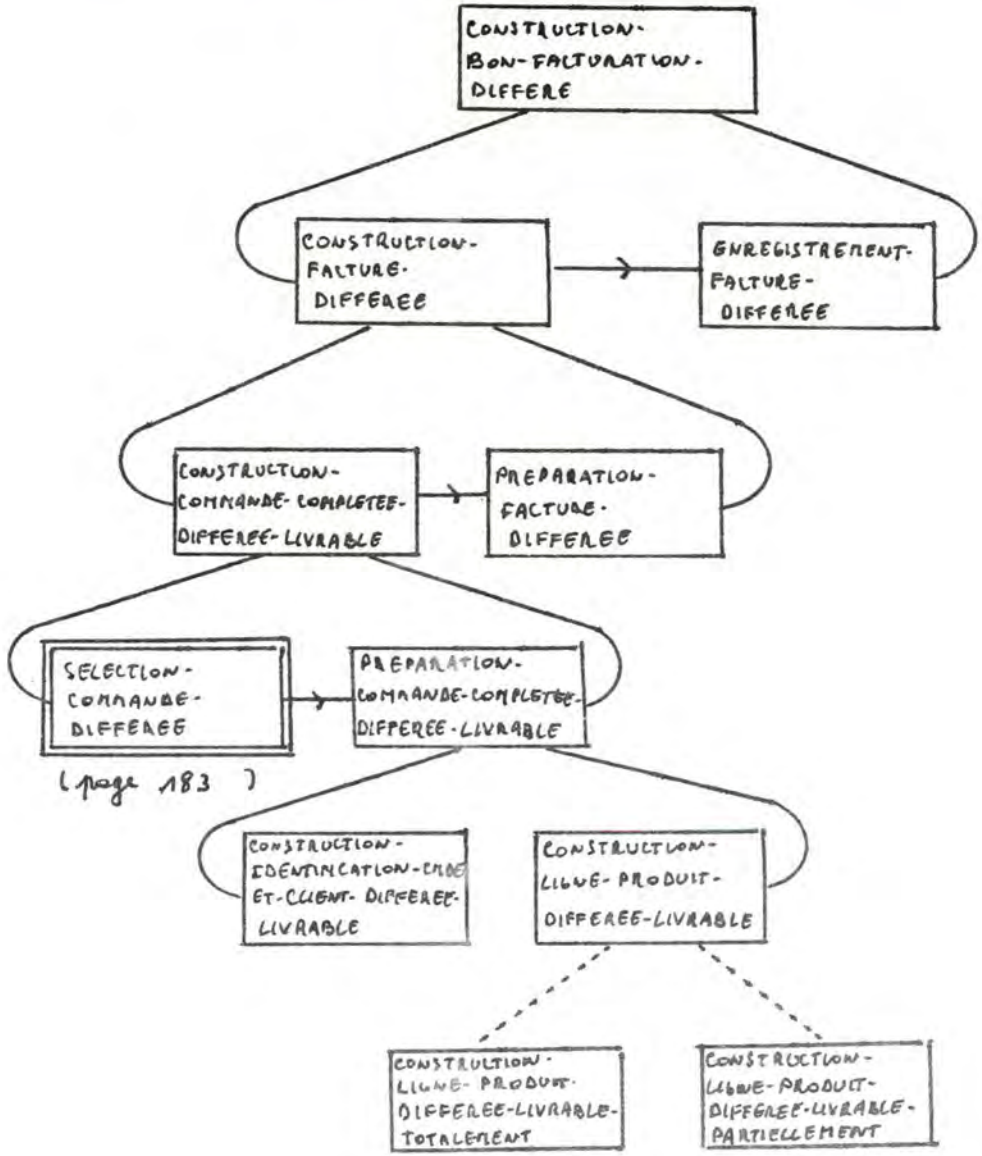

CONSTRUCTION-
COMMANDE-CLIENT-
REFUSEE
CONSTRUCTION-
ERREUR-
SIGNATURE
CONSTRUCTION-
ERREUR-
IDENTIFICATION-CLIENT
CONSTRUCTION-
ERREUR-
CORPS-COMMANDE
CONSTRUCTION-
ERREUR-
LIGNE
CONSTRUCTION-
ERREUR-
MONTANT
CONSTRUCTION-
LIGNES-
PRODUITS
CONSTRUCTION-
ERREUR-MONTANT-
COMMANDE.

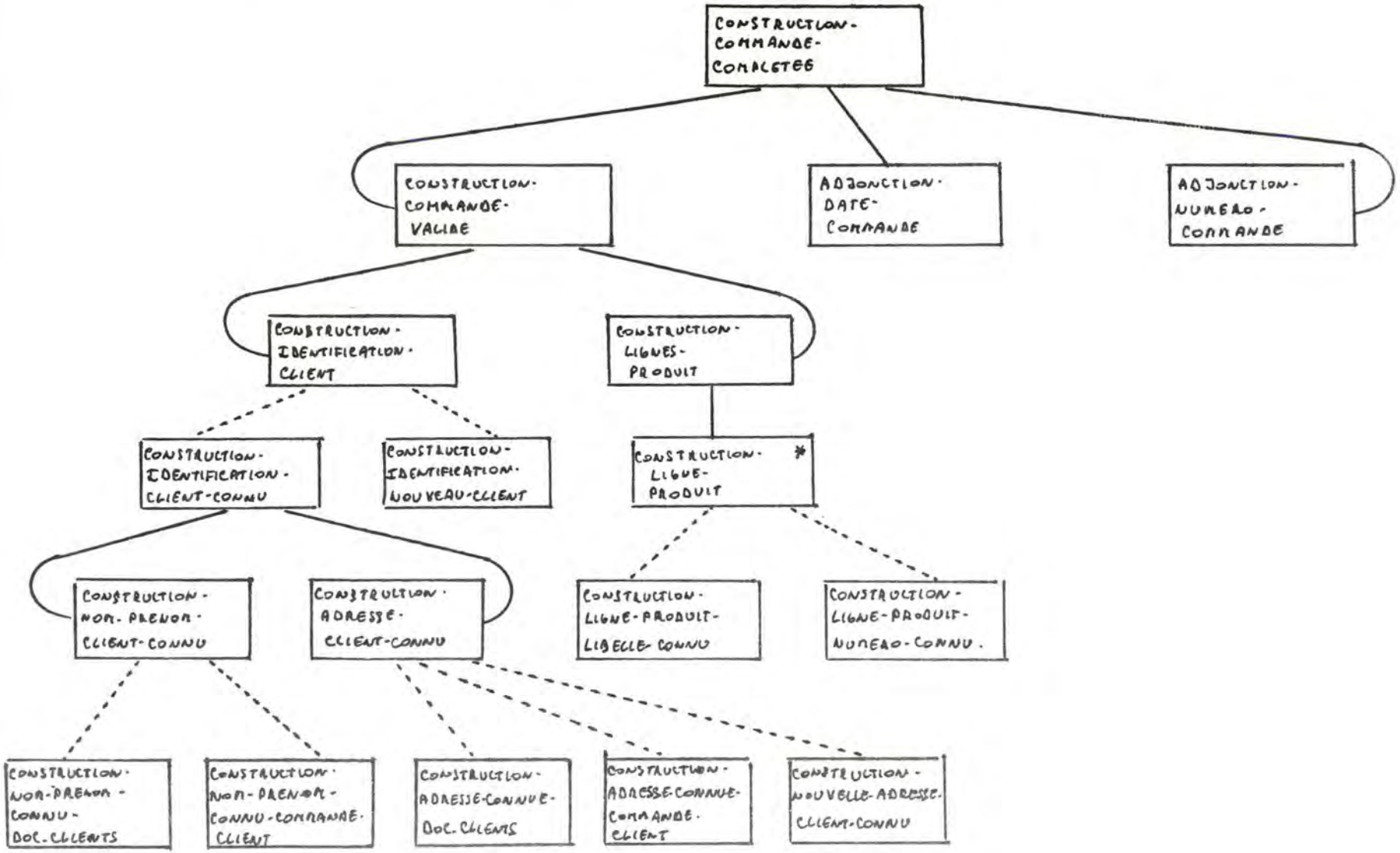
(page 181)

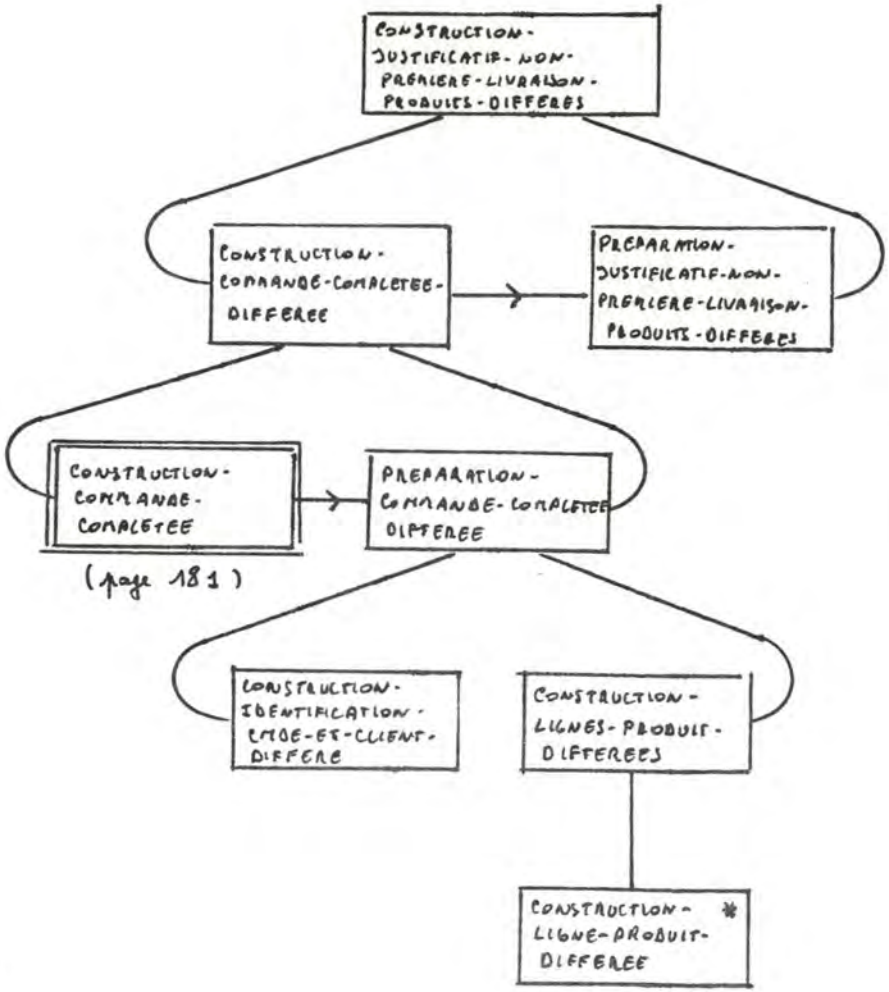

CONSTRUCTION - JUSTIFICATIF - NON - PREMIERE - LIVRAISON - PRODUITS - EPUISES
CONSTRUCTION - COMMANDE - COMPLETEE - EPUISEE
PREPARATION - JUSTIFICATIF - NON - PREMIERE - LIVRAISON - PRODUITS - EPUISES
CONSTRUCTION - COMMANDE - COMPLETEE
(page 181)
PREPARATION - COMMANDE - COMPLETEE - EPUISEE
CONSTRUCTION - IDENTIFICATION - CLIENT - COMMANDE - EPUISE
CONSTRUCTION - LIGNES - PRODUITS - EPUISES
CONSTRUCTION * LIGNE - PRODUIT - EPUISE

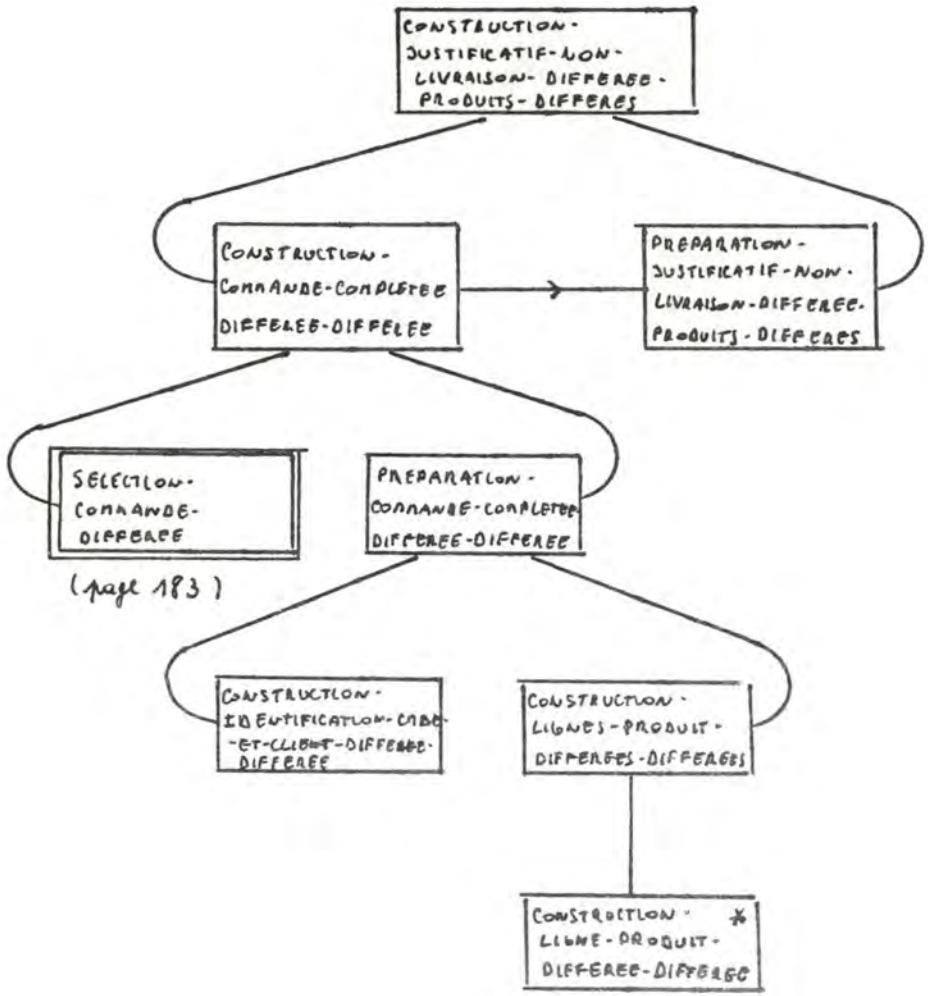

CONSTRUCTION - JUSTIFICATIF - NON - LIVRAISON - DIFFEREE - PRODUIT - EPUISE
CONSTRUCTION - COMMANDE - COMPLETEE - DIFFEREE - EPUISEE
PREPARATION - JUSTIFICATIF - NON - LIVRAISON - DIFFEREE - PRODUIT - EPUISE
SELECTION COMMANDE DIFFEREE
(page 183)
PREPARATION - COMMANDE - COMPLETEE DIFFEREE - EPUISEE
CONSTRUCTION - IDENTIFICATION - CLIENT - COMMANDE - DIFFEREE - EPUISEE
CONSTRUCTION - LIGNE - PRODUIT - DIFFEREE - EPUISEE

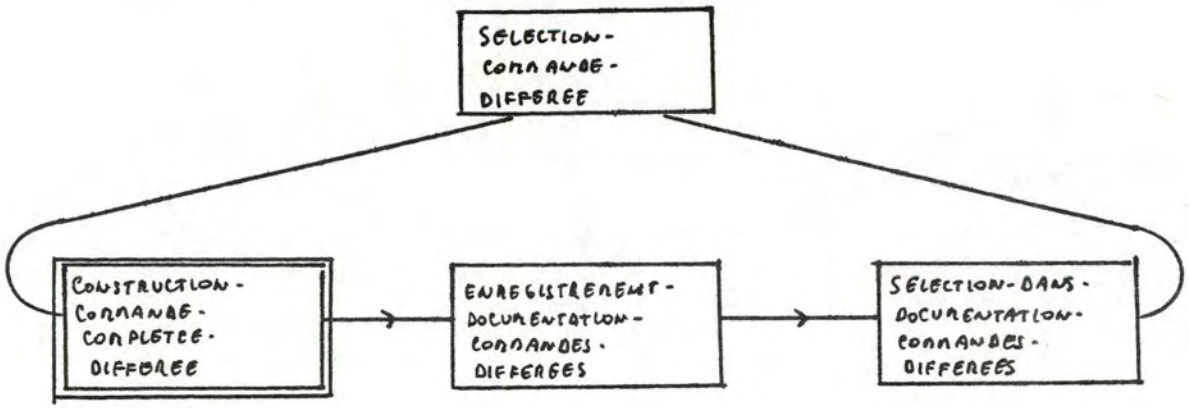

CONSTRUCTION-BON-FACTURATION
CONSTRUCTION-FACTURE
ENREGISTREMENT-FACTURE
CONSTRUCTION-COMMANDE-COMPLETEE-LIVRABLE
PREPARATION-FACTURE
CONSTRUCTION-COMMANDE-COMPLETEE
(page 181)
PREPARATION-COMMANDE-COMPLETEE-LIVRABLE
CONSTRUCTION-IDENTIFICATION-EMDE-ET-CLIENT-LIVRAISON
CONSTRUCTION-LIGNES-PRODUIT-LIVRAISON
CONSTRUCTION-LIGNE-PRODUIT-LIVRAISON *
CONSTRUCTION-LIGNE-PRODUIT-LIVRABLE-TOTALEMENT
CONSTRUCTION-LIGNE-PRODUIT-LIVRABLE-PARTIELLEMENT


CONSTRUCTION-BON-FACTURATION-DIFFERE
CONSTRUCTION-FACTURE-DIFFEREE
ENREGISTREMENT-FACTURE-DIFFEREE
CONSTRUCTION-COMMANDE-COMPLETEE-DIFFEREE-LIVRABLE
PREPARATION-FACTURE-DIFFEREE
SELECTION-COMMANDE-DIFFEREE
(page 183)
PREPARATION-COMMANDE-COMPLETEE-DIFFEREE-LIVRABLE
CONSTRUCTION-IDENTIFICATION-EMDE-ET-CLIENT-DIFFEREE-LIVRABLE
CONSTRUCTION-LIGNE-PRODUIT-DIFFEREE-LIVRABLE
CONSTRUCTION-LIGNE-PRODUIT-DIFFEREE-LIVRABLE-TOTALEMENT
CONSTRUCTION-LIGNE-PRODUIT-DIFFEREE-LIVRABLE-PARTIELLEMENT


CONSTRUCTION-
COMMANDE-
COMPLETEE
CONSTRUCTION-
COMMANDE-
VALIDE
ADJONCTION-
DATE-
COMMANDE
ADJONCTION-
NUMERO-
COMMANDE
CONSTRUCTION-
IDENTIFICATION-
CLIENT
CONSTRUCTION-
LIGNES-
PRODUIT
CONSTRUCTION-
IDENTIFICATION-
CLIENT-CONNU
CONSTRUCTION-
IDENTIFICATION-
NOUVEAU-CLIENT
CONSTRUCTION-
LIGNE-
PRODUIT
*
CONSTRUCTION-
NOM-PRENOM-
CLIENT-CONNU
CONSTRUCTION-
ADRESSE-
CLIENT-CONNU
CONSTRUCTION-
LIGNE-PRODUIT-
LIBELLE-CONNU
CONSTRUCTION-
LIGNE-PRODUIT-
NUMERO-CONNU.
CONSTRUCTION-
NOM-PRENOM-
CONNU-
DOC-CLIENTS
CONSTRUCTION-
NOM-PRENOM-
CONNU-COMMANDE-
CLIENT
CONSTRUCTION-
ADRESSE-CONNUE-
DOC-CLIENTS
CONSTRUCTION-
ADRESSE-CONNUE-
COMMANDE-
CLIENT
CONSTRUCTION-
NOUVELLE-ADRESSE-
CLIENT-CONNU


CONSTRUCTION - JUSTIFICATIF - NON - PRELIERE - LIVRAISON - PRODUITS - DIFFERES
CONSTRUCTION - COMMANDE - COMPLETEE - DIFFEREE
PREPARATION - JUSTIFICATIF - NON - PRELIERE - LIVRAISON - PRODUITS - DIFFERES
CONSTRUCTION - COMMANDE - COMPLETEE
(page 181)
PREPARATION - COMMANDE - COMPLETEE DIFFEREE
CONSTRUCTION - IDENTIFICATION - CMDE - ET - CLIENT - DIFFEREE
CONSTRUCTION - LIGNES - PRODUIT - DIFFEREES
CONSTRUCTION - LIGNE - PRODUIT - DIFFEREE *


CONSTRUCTION - JUSTIFICATIF - NON - LIVRAISON - DIFFEREE - PRODUITS - DIFFERES
CONSTRUCTION - COMMANDE - COMPLETEE DIFFEREE - DIFFEREE
PREPARATION - JUSTIFICATIF - NON - LIVRAISON - DIFFEREE - PRODUITS - DIFFERES
SELECTION - COMMANDE - DIFFEREE
(page 183)
PREPARATION - COMMANDE - COMPLETEE DIFFEREE - DIFFEREE
CONSTRUCTION - IDENTIFICATION - CMDE - ET - CLIENT - DIFFEREE - DIFFEREE
CONSTRUCTION - LIGNES - PRODUIT - DIFFEREES - DIFFEREES
CONSTRUCTION - LIGNE - PRODUIT - DIFFEREE - DIFFEREE *

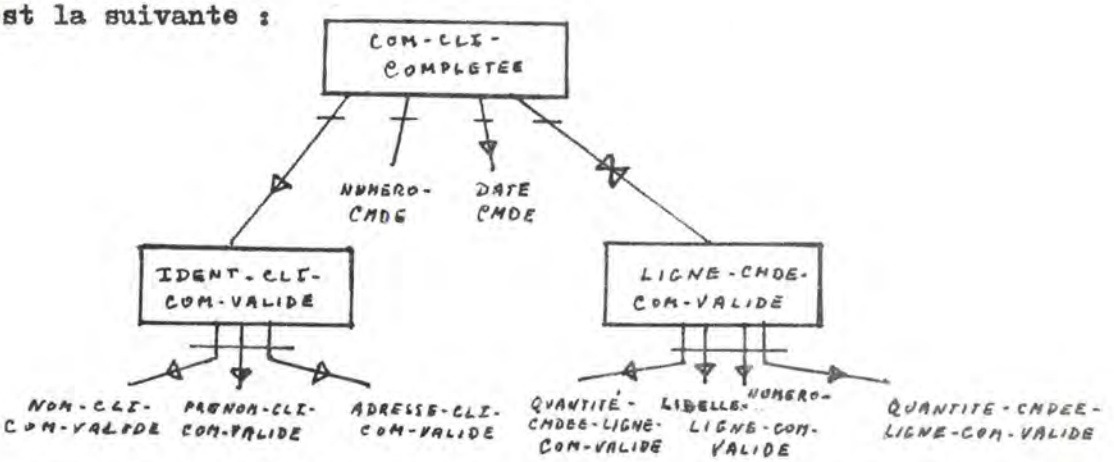

SELECTION-
COMMANDE-
DIFFEREE
CONSTRUCTION-
COMMANDE-
COMPLETEE-
DIFFEREE
ENREGISTREMENT-
DOCUMENTATION-
COMMANDES-
DIFFEREES
SELECTION-DANS-
DOCUMENTATION-
COMMANDES-
DIFFEREES

(page 182)

Annexe 3.

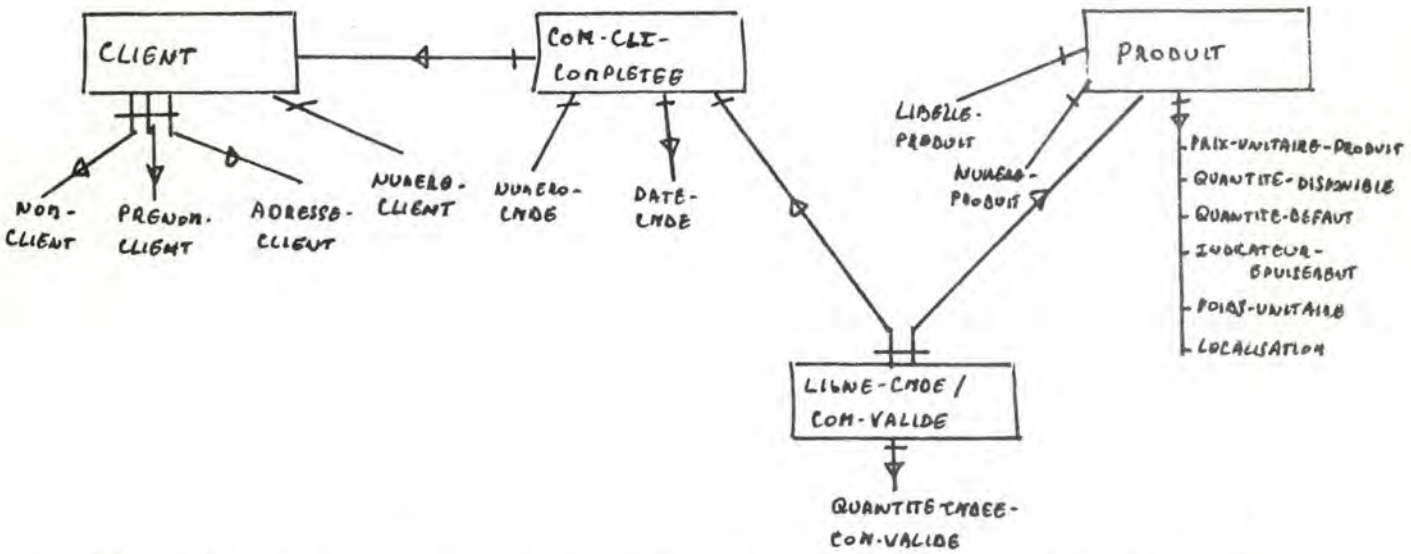
EXEMPLE D'UNIFICATION DES DONNEES.

Dans le cas du module fonctionnel "analyse d'une commande complétée" (spécification pré/post page I25), nous avons comme argument une commande-client-complétée. En termes de formalisme binaire, la structure des données est la suivante :



En fait, la COM-CLI-COMPLETEE étant identifiée par le numéro de commande (NUMERO-CMDE), on peut mémoriser toute la structure de données et passer, comme argument, uniquement le NUMERO-CMDE.

De plus, lors de la mémorisation, on peut profiter des documentations clients et produits et éviter ainsi la redondance d'une partie de l'information. La structure de données suivante peut alors être obtenue en appliquant quelques-unes des transformations décrites dans le paragraphe 2.4.2..



Il est à noter que ces transformations, en plus de permettre de rendre le modèle sans redondance, doivent rendre le modèle binaire compatible avec le système de gestion de base de données à notre disposition, en l'occurrence ici, un système en réseau (on en trouvera une définition dans

(ULL,80), p. 89).

Au niveau du module "analyse d'une commande complétée", nous aurons donc comme argument un numéro de commande (identifiant d'une commande complétée); nous voudrions construire une commande-complétée-différée, une commande--complétée-épuisée (toutes deux étant destinées à être mémorisées car appartenant à des documentations) et une commande-complétée-livrable (que nous essaierons de mémoriser afin de ne devoir passer au module "constitution-série" uniquement le numéro de commande l'identifiant).

Pour la construction de la commande-complétée-différée, on sait par l'analyse fonctionnelle que :

$\forall$ bccd $\in$ COM-CLI-COMPLETEE-DIFFEREE, $\exists!$ ccc $\in$ COM-CLI-COMPLETEE tq

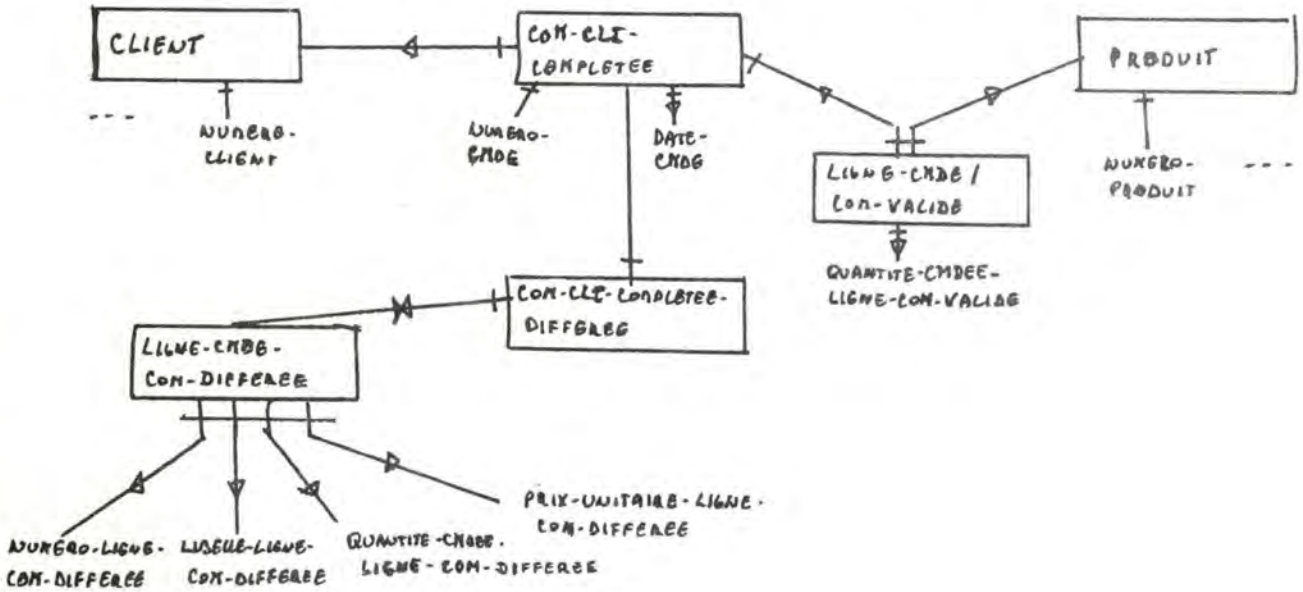
(bccd) = préparation-commande-complétée-différée (ccc)

et $\forall$ ccc $\in$ COM-CLI-COMPLETEE, $\exists$ 0,I bccd $\in$ COM-CLI-COMPLETEE-DIFFEREE tq

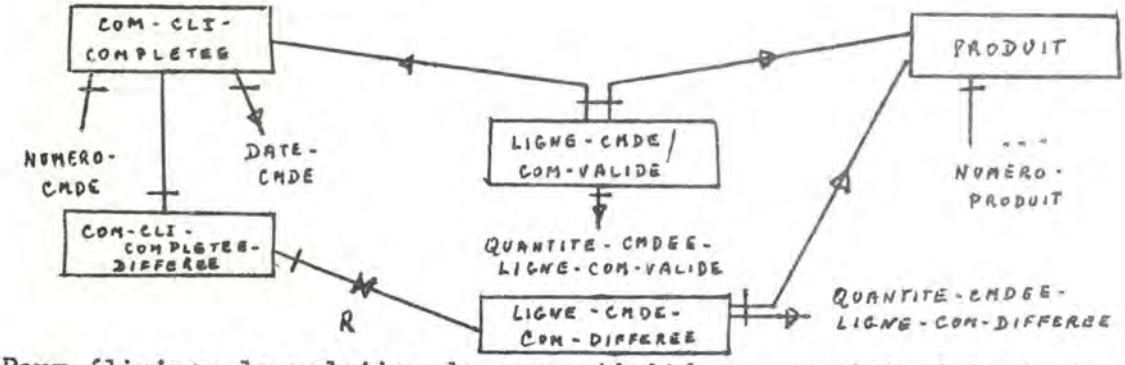
(bccd) = préparation-commande-complétée-différée (ccc)

Dans le formalisme binaire, on peut donc créer une relation entre les deux domaines suivants :

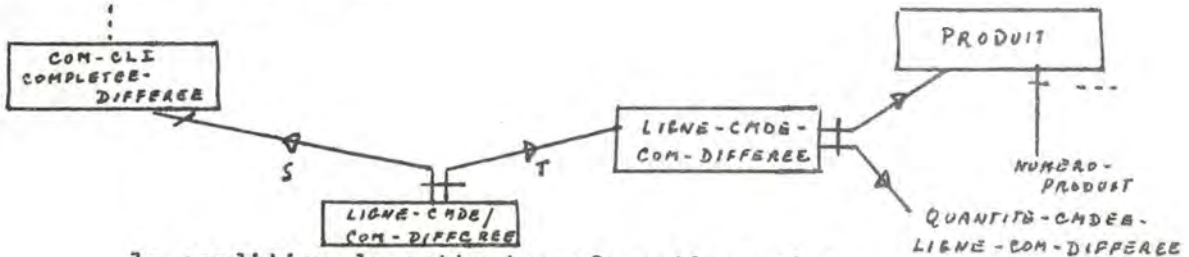
Grâce à cette relation, les renseignements (l'identification du client, la date de cmde et le numéro de cmde)qu'on avait obtenus en les recopiant sur la commande complétée, n'ont plus besoin d'être copiés si on mémorise la COM-CLI-COMPLETEE aussi longtemps que nécessaire.



En profitant du fait que les renseignements inhérents aux lignes différées sont redondants avec ceux figurant pour le produit, on peut en déduire la structure suivante :



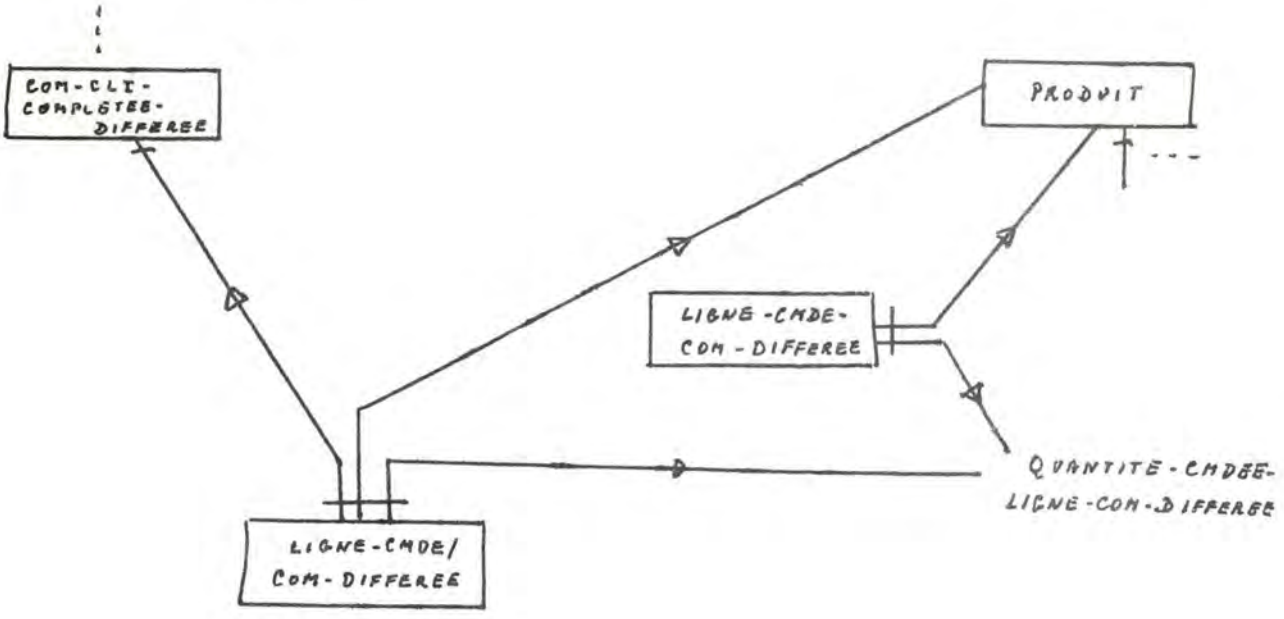
Pour éliminer la relation de connectivité n-n, on introduit le domaine LIGNE-CMDE/COM-DIFFEREE :



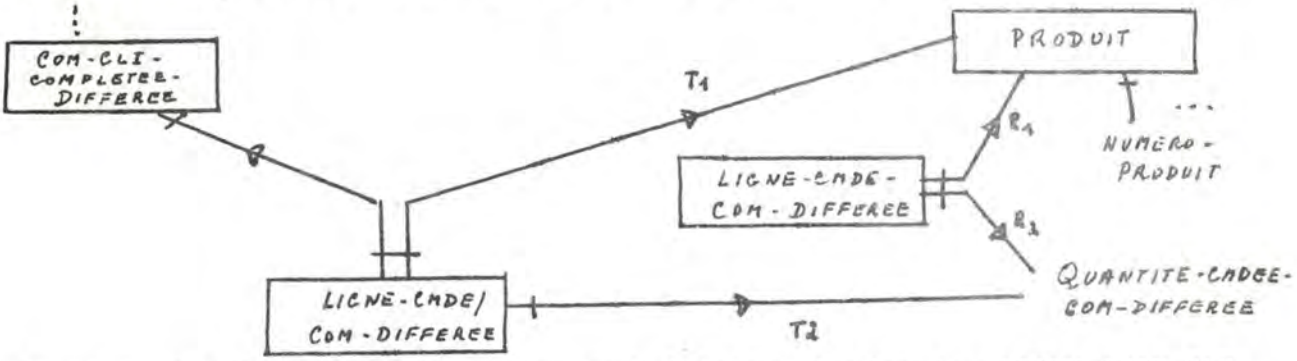
la condition de cette transformation est :

(cccd,lccd) $\in$ R, créer r dans LIGNE-CMDE/COM-DIFFEREE
créer (cccd,r) dans S
créer (lccd,r) dans T

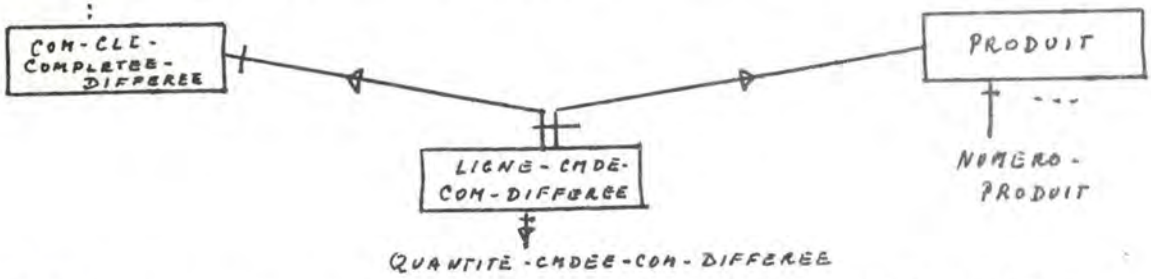
En appliquant la deuxième transformation expliquée dans le paragraphe 2.4.2., on obtient :



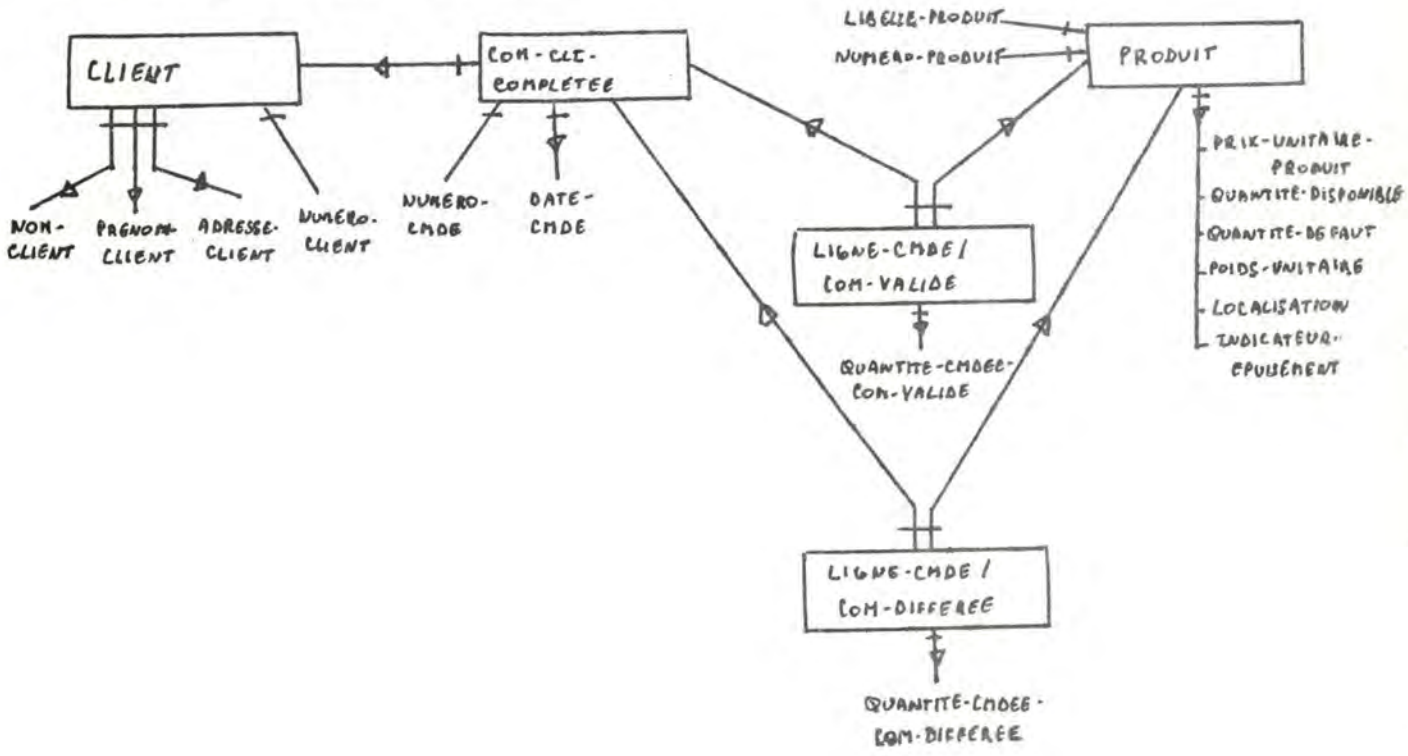
D'autre part, en profitant de la propriété disant que chaque ligne, dans une commande, est relative à un type de produit différent, on a :



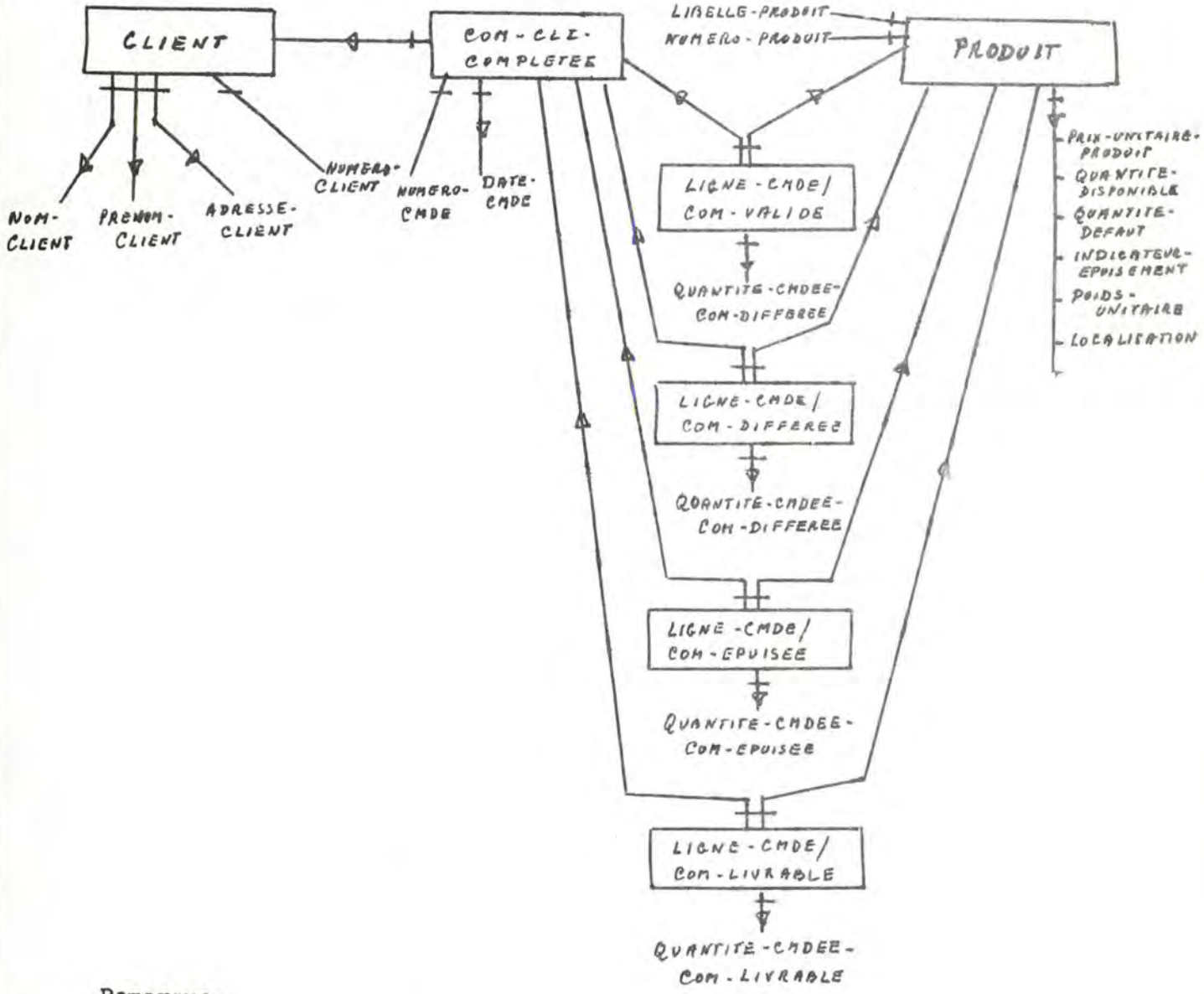
De plus, $T_1 \circ T_2 = R_1 \circ R_2$, donc, on peut éliminer le domaine LIGNE-CMDE-COM-DIFFEREE et les relations R_1 et R_2 pour obtenir :



Enfin, nous pouvons éliminer le domaine COM-CLI-COMPLETEE-DIFFEREE.



De la même manière, nous opérerons pour COM-CLI-COMPLETEE-EPUISEE et COM-CLI-COMPLETEE-LIVRABLE et nous obtiendrons la structure suivante :



Remarque.

La relation n'est plus forte pour COM-CLI-COMPLETEE par rapport à LIGNE--CMDE/COM-VALIDE parce que COM-CLI-COMPLETEE est appelée à subsister après le module "analyse-commande-complétée" pour pouvoir accéder aux lignes épuisées, différées et livrables. Seul le domaine LIGNE-CMDE/COM-VALIDE sera éliminé à la fin de ce module.

Annexe 4.

EXEMPLES DE PROGRAMMES EN PSEUDO-CODE.

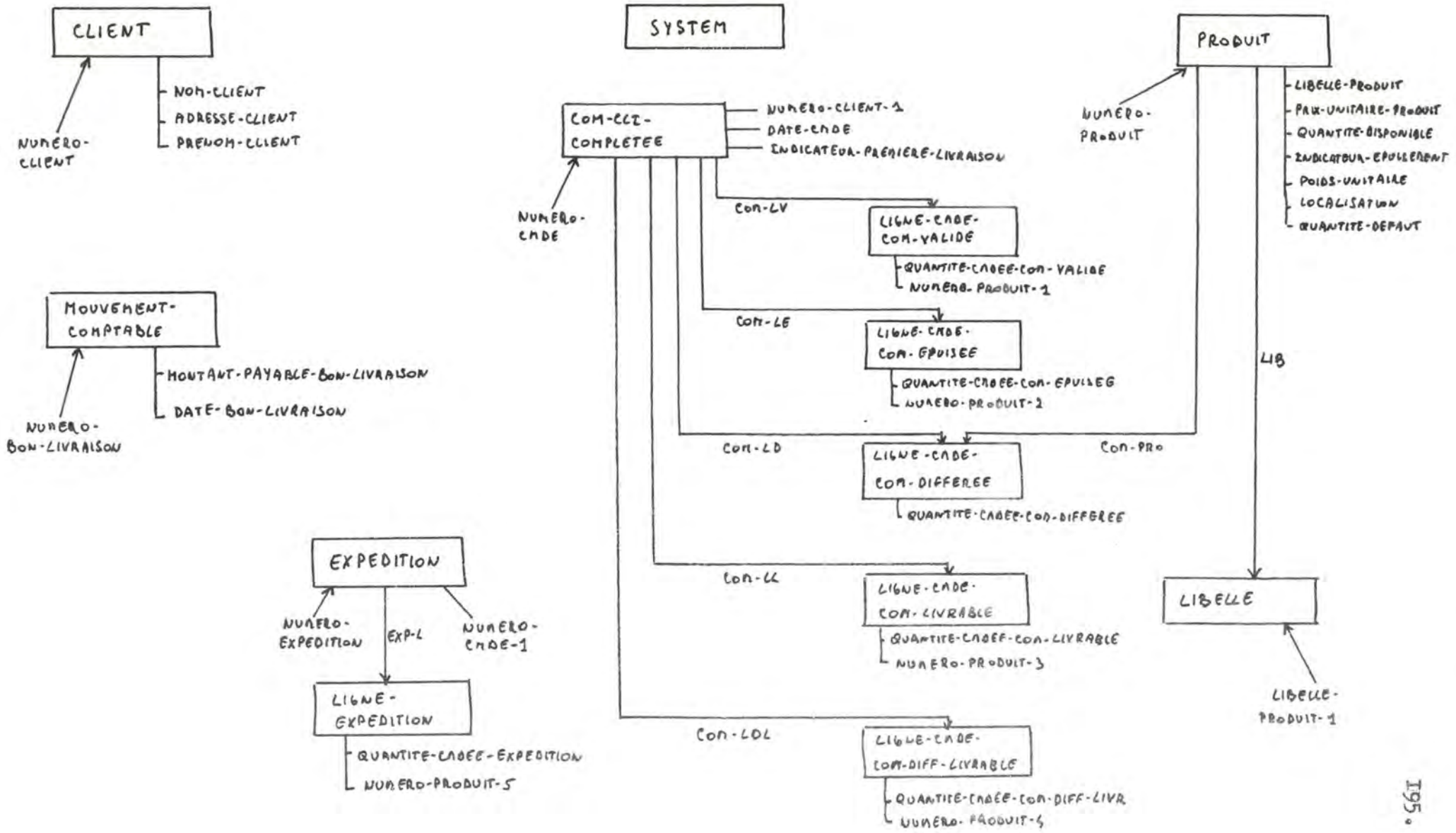
```
RECEVOIR NUMERO-COMMANDE DE < CONSTRUCTION-COMMANDE-COMPLETEE > ;
LIGNE-LIVRABLE-CREE := FALSE ;
for CC ≡ COM-CLI-COMPLETEE such that NUMERO-CMDE (:CC) = NUMERO-CMDE
   do fa each LCCV ≡ LIGNE-CMDE-COM-VALIDE from CC
      do for P ≡ PRODUIT from LCCV
         do attendre signal-moniteur ;
            if INDICATEUR-EPUISEMENT (:P) ≠ "E"
            then if QUANTITE-DISPONIBLE (:P) ≥ QUANTITE-CMDEE-COM-VALIDE (:LCCV)
                 then QCCL := QUANTITE-CMDEE-COM-VALIDE (:LCCV) ;
                      QD := QUANTITE-DISPONIBLE (:P) - QCCL ;
                      METTRE-A-JOUR P ( QUANTITE-DISPONIBLE = QD ) ;
                      CREER LIGNE-CMDE-COM-LIVRABLE ( (:CC), (:P), QUANTITE-CMDEE-COM-LIVRABLE = QCCL ) ;
                      LIGNE-LIVRABLE-CREE := TRUE
                 else if QUANTITE-DISPONIBLE (:P) > 0
                      then QCCL := QUANTITE-DISPONIBLE (:P) ;
                           QCCD := QUANTITE-CMDEE-COM-VALIDE (:LCCV) - QCCL ;
                           QD := 0 ;
                           METTRE-A-JOUR P ( QUANTITE-DISPONIBLE = QD ) ;
                           CREER LIGNE-CMDE-COM-LIVRABLE ( (:CC), (:P), QUANTITE-CMDEE-COM-LIVRABLE = QCCL ) ;
                           LIGNE-LIVRABLE-CREE := TRUE ;
                           CREER LIGNE-CMDE-COM-DIFFEREE ( (:CC), (:P), QUANTITE-CMDEE-COM-DIFFEREE = QCCD )
                      else QCCD := QUANTITE-CMDEE-COM-VALIDE (:LCCV) ;
                           CREER LIGNE-CMDE-COM-DIFFEREE ( (:CC), (:P), QUANTITE-CMDEE-COM-DIFFEREE = QCCD )
                      fi
                 fi
            else QCCE := QUANTITE-CMDEE-COM-VALIDE (:LCCV) ;
                 CREER LIGNE-CMDE-COM-EPUISEE ( (:CC), (:P), QUANTITE-CMDEE-COM-EPUISEE = QCCE )
            fi ;
            SUPPRIMER LCCV ;
            signaler moniteur
         od
      od
   od
if LIGNE-LIVRABLE-CREE = TRUE
then ENVOYER NUMERO-COMMANDE à < CONSTITUTION-SERIE >
fi
```

```
RECEVOIR <NUM-PROD> DE <DATA> ;
for P = PRODUIT such that NUMERO-PRODUIT(:P) = NUM-PROD
  do if INDICATEUR-EPUISEMENT (:P) = "E"
     then for each LCCD = LIGNE-CMDE/COM-DIFFEREE from P
          do for CC = COM-COMPLETEES from LCCD
             do QCCE := QUANTITE-CMDEE-LIGNE-COM-DIFFEREE (:LCCD);
                CREER LIGNE-CMDE/COM-EPUISEE ((:CC),(:P), QUANTITE-CMDEE-LIGNE-COM-EPUISEE = QCCE );
                SUPPRIMER LCCD
             od
          od
     else for each LCCD = LIGNE-CMDE/COM-DIFFEREE from P such that QUANTITE-DISPONIBLE (:P) > 0
          do for CC = COM-COMPLETEE from LCCD
             do if QUANTITE-DISPONIBLE(:P) >= QUANTITE-CMDEE-LIGNE-COM-DIFFEREE (:LCCD)
                then QCLCDL := QUANTITE-CMDEE-LIGNE-COM-DIFFEREE (:LCCD);
                     QD := QUANTITE-DISPONIBLE (:P) - QCLCDL;
                     CREER LIGNE-CMDE/COM-DIFF-LIVRABLE ((:CC),(:P), QUANTITE-CMDEE-LIGNE-COM-DIFF-LIVRABLE = QCLCDL );
                     SUPPRIMER LCCD;
                     METTRE-A-JOUR P( QUANTITE-DISPONIBLE = QD )
                else QCLCDL := QUANTITE-DISPONIBLE (:P);
                     QD := 0;
                     QCLCD := QUANTITE-CMDEE-LIGNE-COM-DIFFEREE (:LCCD) - QCLCDL;
                     CREER LIGNE-CMDE/COM-DIFF-LIVRABLE ((:CC),(:P), QUANTITE-CMDEE-LIGNE-COM-DIFF-LIVRABLE = QCLCDL);
                     METTRE-A-JOUR LCCD (QUANTITE-CMDEE-LIGNE-COM-DIFF (:LCCD) = QCLCD);
                     METTRE-A-JOUR P (QUANTITE-DISPONIBLE = QD)
                fi;
                NUMERO-COMMANDE := NUMERO-CMDE (:CC);
                ENVOYER <NUMERO-COMMANDE> à <CONSTITUTION-SERIE>
             od
          od
     fi
od
```

```
RECEVOIR < NUM-EXP, NUM-PROD > de < RECONSTITUTION-COMMANDE >
for P = PRODUIT such that NUMERO-PRODUIT (:P) = NUM-PROD
  do QD := ∅ ;
     METTRE-A-JOUR P (QUANTITE-DISPONIBLE = QD);
     for E = EXPEDITION such that NUMERO-EXPEDITION (:E) = NUM-EXP
       do for LE = LIGNE-EXPEDITION from P et E
            do for CC = COM-COMPLETEE from E
                 do CHERCHER (arg : cc, P résult : trouvé-ligne-diff , LCCD = LIGNE-CMDE-COM-DIFFEREE from ce et p);
                    if trouvé-ligne-diff
                    then QCLCD := QUANTITE-CMDEE-LIGNE-COM-DIFFEREE ( : LCCD) + QUANTITE-CMDEE-LIGNE-EXPEDITION (:LE);
                         METTRE-A-JOUR LCCD ( QUANTITE-CMDEE-LIGNE-COM-DIFFEREE = QCLCD )
                    else QCLCD := QUANTITE-CMDEE-LIGNE-EXPEDITION (:LE);
                         CREER LIGNE-CMDE-COM-DIFFEREE ((:CC), (:P), QUANTITE-CMDEE-LIGNE-COM-DIFFEREE = QCLCD )
                    fi
                 od
               SUPPRIMER LE
            od;
          CHERCHER (arg : E , result : trouvé-ligne-exp )
          if trouve-ligne-exp
          then NUM-EXP := NUMERO-EXPEDITION (:E) ;
               ENVOYER < NUM-EXP > à < FACTURATION >
          fi
     od
od
```

Annexe 5.

EXEMPLE D'IMPLEMENTATION DE LA STRUCTURE DE DONNEES.


CLIENT
NUMERO-CLIENT
NOM-CLIENT
ADRESSE-CLIENT
PRENOM-CLIENT
SYSTEM
COM-CLI-COMPLETEE
NUMERO-CLIENT-1
DATE-CMDE
INDICATEUR-PREMIERE-LIVRAISON
NUMERO-CMDE
CON-LV
LIGNE-CMDE-COM-VALIDE
QUANTITE-CMDEE-CON-VALIDE
NUMERO-PRODUIT-1
CON-LE
LIGNE-CMDE-COM-EPUISEE
QUANTITE-CMDEE-CON-EPUISEE
NUMERO-PRODUIT-2
CON-LD
CON-PRO
LIGNE-CMDE-COM-DIFFEREE
QUANTITE-CMDEE-CON-DIFFEREE
CON-LL
LIGNE-CMDE-COM-LIVRABLE
QUANTITE-CMDEE-CON-LIVRABLE
NUMERO-PRODUIT-3
CON-LDL
LIGNE-CMDE-COM-DIFF-LIVRABLE
QUANTITE-CMDEE-CON-DIFF-LIVR
NUMERO-PRODUIT-4
PRODUIT
NUMERO-PRODUIT
LIBELLE-PRODUIT
PRIX-UNITAIRE-PRODUIT
QUANTITE-DISPONIBLE
INDICATEUR-EPUISEMENT
POIDS-UNITAIRE
LOCALISATION
QUANTITE-DEFAUT
LIB
LIBELLE
LIBELLE-PRODUIT-1
MOUVEMENT-COMPTABLE
NUMERO-BON-LIVRAISON
MONTANT-PAYABLE-BON-LIVRAISON
DATE-BON-LIVRAISON
EXPEDITION
NUMERO-EXPEDITION
EXP-L
NUMERO-CMDE-1
LIGNE-EXPEDITION
QUANTITE-CMDEE-EXPEDITION
NUMERO-PRODUIT-5

```
ASSIGN AR-PRODUIT TO FPROD
 RPP 50
 CALC AT MOST 5 RPP
 FIRST PAGE IS 1
 LAST PAGE IS 10
 PAGE SIZE IS 512 WORDS.
ASSIGN AR-CLIENT TO FCLI
 RPP 50
 CALC AT MOST 5 RPP
 FIRST PAGE IS 11
 LAST PAGE IS 20
 PAGE SIZE IS 512 WORDS.
ASSIGN AR-COM TO FCMDE
 RPP 50
 CALC AT MOST 5 RPP
 FIRST PAGE IS 21
 LAST PAGE IS 40
 PAGE SIZE IS 512 WORDS.
ASSIGN AR-EXP TO FEXP
 RPP 50
 CALC AT MOST 5 RPP
 FIRST PAGE IS 41
 LAST PAGE IS 50
 PAGE SIZE IS 512 WORDS.
ASSIGN AR-MVT TO FVMT
 RPP 50
 CALC AT MOST 5 RPP
 FIRST PAGE IS 51
 LAST PAGE IS 60
 PAGE SIZE IS 512 WORDS.

SCHEMA NAME IS PETIBD.

AREA NAME IS AR-PRODUIT.
AREA NAME IS AR-CLIENT.
AREA NAME IS AR-COM.
AREA NAME IS AR-EXP.
AREA NAME IS AR-MVT.

RECORD NAME IS MOUVEMENT-COMPTABLE
 LOCATION MODE IS CALC USING NUMERO-BON-LIVRAISON
 DUPLICATES ARE NOT ALLOWED
 WITHIN AR-MVT.
 02 NUMERO-BON-LIVRAISON PIC 9(6).
 02 MONTANT-PAYABLE-BON-LIVRAISON PIC 9(6).
 02 DATE-BON-LIVRAISON PIC 9(6).

RECORD NAME IS CLIENT
 LOCATION MODE IS CALC USING NUMERO-CLIENT
 DUPLICATES ARE NOT ALLOWED
 WITHIN AR-CLIENT.
 02 NUMERO-CLIENT PIC 9(6).
 02 NOM-CLIENT PIC X(30).
 02 ADRESSE-CLIENT PIC X(60).
 02 PRENOM-CLIENT PIC X(13).

RECORD NAME IS COM-CLI-COMPLETEE
 LOCATION MODE IS CALC USING NUMERO-CMDE
 DUPLICATES ARE NOT ALLOWED
 WITHIN AR-COM.



 02 NUMERO-CMDE PIC 9(6).
 02 DATE-CMDE PIC 9(6).
 02 INDICATEUR-PREMIERE-LIVRAISON PIC X.
 02 NUMERO-CLIENT-1 PIC 9(6).

RECORD NAME IS LIGNE-CMDE-COM-VALIDE
 LOCATION MODE IS VIA COM-LV
 WITHIN AR-COM.
 02 QUANTITE-CMDEE-COM-VALIDE PIC 9(2).
 02 NUMERO-PRODUIT-1 PIC 9(8).

RECORD NAME IS LIGNE-CMDE-COM-EPUISEE
 LOCATION MODE IS VIA COM-LE
 WITHIN AR-COM.
 02 QUANTITE-CMDEE-COM-EPUISEE PIC 9(2).
 02 NUMERO-PRODUIT-2 PIC 9(8).

RECORD NAME IS LIGNE-CMDE-COM-DIFFEREE
 LOCATION MODE IS VIA COM-LD
 WITHIN AR-COM.
 02 QUANTITE-CMDEE-COM-DIFFEREE PIC 9(2).
```

```
RECORD NAME IS LIGNE-CMDE-COM-LIVRABLE
 LOCATION MODE IS VIA COM-LL
 WITHIN AR-COM.
 02 QUANTITE-CMDEE-COM-LIVRABLE PIC 9(2).
 02 NUMERO-PRODUIT-3 PIC 9(8).

RECORD NAME IS LIGNE-CMDE-COM-DIFF-LIVRABLE
 LOCATION MODE IS VIA COM-LDL
 WITHIN AR-COM.
 02 QUANTITE-CMDEE-COM-DIFF-LIVR PIC 9(2).
 02 NUMERO-PRODUIT-4 PIC 9(8).

RECORD NAME IS EXPEDITION
 LOCATION MODE IS CALC USING NUMERO-EXPEDITION
 DUPLICATES ARE NOT ALLOWED
 WITHIN AR-EXP.
 02 NUMERO-EXPEDITION PIC 9(6).

RECORD NAME IS LIGNE-CMDE-COM-EXPEDITION
 LOCATION MODE IS VIA EXP-L
 WITHIN AR-EXP.
 02 QUANTITE-CMDEE-EXPEDITION PIC 9(2).
 02 NUMERO-PRODUIT-5 PIC 9(8).

RECORD NAME IS PRODUIT
 LOCATION MODE IS CALC USING NUMERO-PRODUIT
 DUPLICATES ARE NOT ALLOWED
 WITHIN AR-PRODUIT.
 02 NUMERO-PRODUIT PIC 9(8).
 02 LIBELLE-PRODUIT PIC X(50).
 02 PRIX-UNITAIRE-PRODUIT PIC 9(5).
 02 QUANTITE-DISPONIBLE PIC 9(5).
 02 INDICATEUR-EPUISEMENT PIC X.
 02 POIDS-UNITAIRE PIC 99V999.
 02 LOCALISATION PIC X(30).
 02 QUANTITE-DEFAUT PIC 9(2).

RECORD NAME IS LIBELLE



 LOCATION MODE IS CALC USING LIBELLE-PRODUIT-1
 DUPLICATES ARE NOT ALLOWED
 WITHIN AR-PRODUIT.
 02 LIBELLE-PRODUIT-1 PIC X(50).


SET NAME IS COM-LV
 MODE IS CHAIN
 ORDER IS ALWAYS FIRST
 OWNER IS COM-CLI-COMPLETEE
 MEMBER IS LIGNE-CMDE-COM-VALIDE MANDATORY AUTOMATIC.

SET NAME IS COM-LE
 MODE IS CHAIN
 ORDER IS ALWAYS FIRST
 OWNER IS COM-CLI-COMPLETEE
 MEMBER IS LIGNE-CMDE-COM-EPUISEE MANDATORY AUTOMATIC.

SET NAME IS COM-LD
 MODE IS CHAIN
 ORDER IS ALWAYS FIRST
 OWNER IS COM-CLI-COMPLETEE
 MEMBER IS LIGNE-CMDE-COM-DIFFEREE MANDATORY AUTOMATIC.

SET NAME IS COM-LL
 MODE IS CHAIN
 ORDER IS ALWAYS FIRST
 OWNER IS COM-CLI-COMPLETEE
 MEMBER IS LIGNE-CMDE-COM-LIVRABLE MANDATORY AUTOMATIC.

SET NAME IS COM-LDL
 MODE IS CHAIN
 ORDER IS ALWAYS LAST
 OWNER IS COM-CLI-COMPLETEE
 MEMBER IS LIGNE-CMDE-COM-DIFF-LIVRABLE MANDATORY AUTOMATIC.

SET NAME IS COM-PRO
 MODE IS CHAIN
 ORDER IS ALWAYS LAST
 OWNER IS PRODUIT
 MEMBER IS LIGNE-CMDE-COM-DIFFEREE MANDATORY AUTOMATIC.
```

```
SET NAME IS LIB
 MODE IS CHAIN
 ORDER IS ALWAYS FIRST
 OWNER IS PRODUIT
 MEMBER IS LIBELLE MANDATORY AUTOMATIC.

SET NAME IS EXP-L
 MODE IS CHAIN
 ORDER IS ALWAYS FIRST
 OWNER IS EXPEDITION
 MEMBER IS LIGNE-CMDF-COM-EXPEDITION MANDATORY AUTOMATIC.

SUB-SCHEMA IS PETIPA-LOAD.
 AREA SECTION.
 COPY ALL AREAS.
 RECORD SECTION.
 COPY ALL RECORDS.


 SET SECTION.
 COPY ALL SETS.

END-SCHEMA.
```

Annexe 6.

EXEMPLE D'IMPLEMENTATION D'UN PROGRAMME.

Notre but, dans cette annexe 6, est de montrer comment on peut passer d'une manière systématique d'un algorithme décrit en pseudo-code à un algorithme exprimé dans le langage de programmation retenu pour l'application (en l'occurrence, ici, le COBOL).

Pour les parties du programme n'ayant pas trait aux opérations sur la base de données, il est sans difficulté d'assurer, par des règles de transcription, la traduction du pseudo-code en langage COBOL.

Pour les parties du programme ayant trait aux opérations sur la base de données, il est également possible d'assurer, grâce aux règles énoncées par (HAI,81b), une traduction en COBOL/DML utilisé par le logiciel DBMS-20.

C'est à cette traduction, plus délicate que la première, que nous allons consacrer un exemple. Celui-ci a pour base le programme "analyse d'une commande complétée" dont l'expression en pseudo-code se trouve dans l'annexe 4.

Pour la structure d'accès à la commande complétée, nous avions en pseudo-code :

```
for cc ∈ com-cli-complétée such that numero-cmde(:cc) = numero-commande
  do
                <S>
  od
```

la transcription directe est la suivante :

```
open area ar-com usage-mode is exclusive update.
move numero-commande to numero-cmde of com-cli-complétée.
find com-cli-complétée record.
nexte. if error-count > 0  go to fin-com.
move currency status for run-unit to ccc.
                <S>
find com-cli-complétée using ccc.
move numero-commande to numero-cmde of com-cli-complétee.
find next duplicate within com-cli-complétée record.
go to nexte.
fin-com.
close area ar-com.
```

Cette traduction générale peut être simplifiée grâce à la vérification des conditions suivantes :

I. Les instructions de ⟨S⟩ne modifient pas le contenu de numéro-cmde, la deuxième instruction move peut donc être omise.

2. Les instructions de ⟨S⟩ ne modifient pas le courant de la run unit, il n'est donc pas indispensable de sauvegarder et de restaurer ce courant par les instructions move currency et find com-cli-completée using ccc.

3. La clé d'accès étant identifiante, il n'y a pas d'articles suivant le premier trouvé.

La traduction peut alors se réduire à :

```
open area ar-com usage-mode is exclusive update.
move numero-commande to numero-cmde of com-cli-complétée.
find com-cli-complétée record.
if error-count > 0 go to fin-com.
    ⋮
        ⟨S⟩
    ⋮
close area ar-com.
```

Pour la structure d'accès à chaque ligne de la commande complétée :

nous avions en pseudo-code;

```
for each lccv ∈ ligne-cmde-com-valide from cc
    do
        ⟨T⟩
    od
```

la transcription directe est la suivante :

```
open area ar-com usage-mode is exclusive update.
find com-cli-complétée using numéro-cmde.
find first ligne-cmde-com-valide record of com-lv set.
trait-lig. if error-count > 0 go to fin-lig.
move currency status for run unit to lcccv.
    ⋮
        ⟨T⟩
    ⋮
find ligne-cmde-com-valide using lcccv.
find next ligne-cmde-com-valide of com-lv set.
go to trait-lig.
close area ar-com.
fin-lig.
```

Cette traduction générale peut être simplifiée grâce à la vérification des conditions suivantes :

I. Les opérations OPEN et CLOSE sont redondantes avec celles du bloc imbriquant celui-ci.

2. Le positionnement sur com-cli-complétée a déjà été assuré.

3. Les instructions de la séquence ⟨T⟩ ne modifient pas le courant du type du set com-lv; il est inutile de sauvegarder la référence lcccv et de restaurer la run-unit.

La traduction devient ainsi :

```
find first ligne-cmde-com-valide record of com-lv set.
trait-lig.
if error-count > 0 go to fin-lig.
:
 ⟨T⟩
:
find next ligne-cmde-com-valide of com-lv set.
go to trait-lig.
fin-lig.
```

Pour la structure d'accès au produit :

nous avions en pseudo-code :

```
for p = produit from lcαv
   do
   :
        ⟨V⟩
   :
   od
```

Dans l'implantation du schéma d'accès, nous avons renoncé dans un but de performance au chemin entre le produit et la ligne-cmde-com-valide en ajoutant un attribut numéro-produit-I à la ligne-cmde-com-valide.

De ce fait, pour accéder au produit en étant déjà positionner sur la ligne--cmde-com-valide, il faut faire :

```
get ligne-cmde-com-valide.
open area ar-produit  usage-mode is exclusive update.
move numéro-produit-I to numéro-produit of produit.
find produit record.
if error-count > 0 go to fin-prod.
:
```

<V>

close area ar-produit.

fin-prod.

Pour la traduction des ordres de consultation, création, mise à jour et suppression, nous profiterons généralement des positionnements obtenus par les opérations décrites ci-dessus.

Le programme Cobol obtenu, après transcription des opérations non relatives à la base de données, sera celui trouvé dans les pages qui suivent.

```
Identification Division.
  Program-id. Maj-mo.
Environment Division.
Data Division.
  Schema Section.
    Invoke Sub-schema Petipa-load of Schema Petibd.
  Working-storage Section.
    OI Ligne-livrable-crée  Pic A(5).
    OI Numero-commande  Pic 9(6).
    OI Pgm-id  Pic X(20) Value "Analyse-cmde-comp".
    OI Pgm-id-rec  Pic X(20) Value "Constitution-serie".
    OI Pgm-idx-rec Pic S9(10) computational.
    OI Fct-code  Pic S9(10) computational.
    OI Resumpt-code  Pic S9(10) computational.
    77 Error-code  Pic S9(10) computational.
Procedure Division.
*
Analyse-com-cli-complettee Section.
  Reception-information.
    Enter Macro Ipcrid using Pgm-id, Error-code.
    Enter Macro Ipcrdx using Pgm-id-rec, Pgm-idx-rec, Error-code.
    Move 0 to Fct-code.
    Move 2 to Resumpt-code.
    Enter Macro Ipwait using Fct-code, Resumpt-code, Error-code.
    Enter Macro Iprecv using Numero-commande, Pgm-idx, Error-code.
  Debut-com.
    Move "false" to ligne-livrable-crée.
    Open area ar-com usage-mode is exclusive update.
    Move numero-commande to numero-cmde of com-cli-completee.
    Find com-cli-completee record.
    If error-code > 0  go to fin-com.
  Trait-com.
    Perform trait-com-cli-completee.
    Close area ar-com.
  Fin-com.
    if ligne-livrable-cree = "true"
      Enter Macro Ipsend using numero-commande, pgm-idx-rec, error-code.
    Enter Macro Ipdldx using pgm-idx-rec, error-code.
    Enter Macro Ipdldx using pgm-idx, error-code.
    Enter Macro Ipdlid using pgm-id, error-code.
    Stop run.
```

```
*
Trait-com-cli-completee Section.
  Debut-lig.
    Find first ligne-cmde-com-valide record of com-lv set.
    If error-count > 0  go to fin-lig.
  Trait-lig.
    Perform trait-ligne-valide.
    Find next ligne-cmde-com-valide record of com-lv set.
     If error-count ≠ 0 go to trait-lig.
    Fin-lig.

*
Trait-ligne-valide Section.
  Debut-prod.
    Get ligne-cmde-com-valide.
    Open area ar-produit usage mode is exclusive update.
    Move numero-produit-I to numero-produit.
    Find produit record.
    If error-count> 0 go to fin-prod.
  Trait-prod.
    Get produit.
    If indicateur-epuisement not = "e"
      if quantite-disponible not < quantité- cmdee-com-valide
        perform produit-livrable-totalement
      else  if quantite-disponible > 0
                perform produit-livrable-partiellement
            else perform produit-differe-totalement
    else perform produit-epuise.
    Delete ligne-cmde-com-valide.
    Close area ar-produit.
  Fin-prod.
*
Produit-livrable-totalement section.
  compute quantite-disponible = quantité-disponible - quantite-cmdee-com-
                                                                  valide
  modify produit quantite-disponible.
  move numero-produit to numero-produit-3.
  move quantite-cmdee-com-valide to quantite-cmdee-com-livrable.
  store ligne-cmde-com-livrable.
  move "true" to ligne-livrable-cree.
*
```

```
Produit-livrable-partiellement Section.
  move quantite-disponible to quantite-cmdee-com-livrable.
  move numero-produit to numero-produit-3.
  compute quantite-cmdee-com-differee = quantite-cmdee-com-valide -
                                        quantite-disponible.
  compute quantite-disponible = 0.
   modify produit quantite-disponible.
  store ligne-cmde-com-livrable.
  store ligne-cmde-com-differee.
*
Produit-differe-totalement Section.
  move quantite-cmdee-com-valide to quantite-cmdee-com-differee.
  store ligne-cmde-com-differee.
*
Produit-epuise Section.
  move quantite-cmdee-com-valide to quantite-cmdee-com-epuisee.
  move numero-produit to numero-produit-2.
  store ligne-cmde-com-epuisee.
```

Références bibliographiques.

ANS, 75 ANSI/X3/SPARC, Study Group on Data Base Management Systems. Interim Report, I975.

ABR, 74 J.R. ABRIAL, Data Semantics. In : Data Base Management, North-Holland, I974.

ABR, 78 J.R. ABRIAL, Z : A Specification Language. IFIP, Tokio, I978.

BAL, 77 R. BALZER, N. GOLDMAN, D. WILE, Informality in Program Specifications. Proc. Third Int. J. Conf. Artif. Intell., Cambridge, Mass., I977, p. 389-397.

BAL, 79 R. BALZER, N. GOLDMAN, Principles of Good Software Specification and Their Implications for Specifications Language. Proceedings of Specifications of Reliable Software, I.E.E.E. Catalog 79--CH-I40I-9C, I979, p. 58-67.

BAR, 77 W. BARTUSSEK, D.L. PARNAS, Using Traces to Write Abstract Specifications for Software Modules. Tech. Report, University of North-Carolina, I977.

BAR, 80 J.G.P. BARNES, An Overview of ADA. Software Practice and Experience, vol.IO, I980.

BAS, 78 V.R. BASILI, A Panel Session- User Experience with New Software Methods. National Computer Conference, I978, p. 629-639.

BEL, 78 F. BELLEGARDE, J-P. FINANCE, B. HUC, J-M. PIERREL, A. QUERE, J-L. REMY, Initiation à une Construction Méthodique de Programmes. Rapport CRIN 78-E-8I, I978.

BOD, 79a F. BODART, Concepts, Méthodes et Outils de l'Analyse Fonctionnelle. Notes de Cours, Institut d'Informatique, Namur, I979.

BOD, 79b F. BODART, Analyse du Cas PETITPAS- Casus, Spécifications D.S.L. et Rapports de Documentation. Institut d'Informatique, Namur,

I979.

BOD, 79c F. BODART, Y. PIGNEUR, A Model and a Language for Functional Specifications and Evaluation of Information System Dynamics. IFIP Conference on Formal Models and Practical Tools for Information System Design, Oxford, I979.

BOD, 8I F. BODART, Introduction à la Conception de Systèmes d'Information, Cours de l'Ecole d'Eté de l'AFCET, Institut d'Informatique, Namur, I98I.

BOE, 76 B.W. BOEHM, Software Engineering. I.E.E.E. Transactions on Computer, vol.25, I2, I976, p.I26-I43.

BOU, 8I M. BOURGEOIS, Séminaire sur les Problèmes d'Organisation. Institut d'Informatique, Namur, I98I.

CHE, 76 P.P. CHEN, The Entity-Relationship Model : Toward a Unified View of Data. A.C.M. T.O.D.S. I, I, I976, p. 9-36.

CLA, 79 A. CLARINVAL, Langage COBOL. Institut d'Informatique, Namur, I979.

DAH, 72 O.J. DAHL, E.W. DIJKSTRA, C.A.R. HOARE, Structured Programming. Academic Press, I972.

DEM, 79 M. DEMUYNCK, B. MEYER, Les Langages de Spécification. Journée Génie Logiciel, IRIA-SESORI, Pont-à-Mousson, I979.

DEM, 80 M. DEMUYNCK, S. CHENUT, J-M. NERSON, Système Zaide d'Aide à la Spécification. Actes du Congrès AFCET : Théorie et Techniques de l'Informatique, I980, p. I79-I88.

DER, 79 J-C. DERNIAME, J-P. FINANCE, Types Abstraits de Données : Spécification, Utilisation et Réalisation. Cours de l'Ecole d'Eté de l'AFCET, Monastir, Rapport CRIN 79-E-57, I979.

DIG, 77 DIGITAL EQUIPMENT CORPORATION, Data Base Management System :

Programmer's Procedures Manual. I977.

DIG, 79 DIGITAL EQUIPMENT CORPORATION, Traffic-20 : Programmer's Manual. I979.

DIJ, 75 E.W. DIJKSTRA, Guarded Commands, Non Determinancy and Formal Derivation of Programs. Comm. A.C.M., vol.I8, 8, I975, p.453--457.

DIJ, 76 E.W. DIJKSTRA, A Discipline of Programming. Prentice Hall, I976.

DUF, 80 J-F. DUFOURD, Maquettes pour Evaluer les Systèmes d'Information des Organisations. Thèse d'Etat, Université de Nancy-I, I980.

FIN, 79 J-P. FINANCE, Etude de la Construction des Programmes : Méthodes et Langages de Spécification et de Résolution de Problèmes. Thèse d'Etat, Université de Nancy-I, I979.

FLO, 67 R.W. FLOYD, Assigning Meanings to Programs. Proceedings of the Symposium in Applied Mathematics, Mathematical Aspect of Computer Science, J.T. SCWARTZ (ed.), American Mathematical Society, I967, p. I9-32.

GER, 80 G. GERMAIN, Les Nouvelles Techniques de Spécification du Logiciel. Journée de Synthèse, Actes du Congrès AFCET : Théorie et Techniques de l'Informatique, Nancy, I980.

GOD, 76 J.B. GOODENOUGH, S. GERHART, Toward a Theory of Testing : Data Selection Criteria. In : Current Trends in Programming Methodology, vol.2, Program Validation, R.T. YEH (ed.), Prentice Hall, I976.

GOL, 75 P.C. GOLDBERG, Structured Programming for Non-Programmers. IBM Res. Rep. RC53I8, Yorktown Heights, N.Y., I975, p. 245-264.

GUT, 77 J.V. GUTTAG, Abstract Data Types and the Development of Data Structures. Comm. A.C.M., vol.20, 6, I977, p. 396-404.

GUT, 78 J.V. GUTTAG, J.J. HORNING, The Algebraic Specification of Abstract Data Types. Acta Informatica, 10, 1978, p. 27-52.

HAI, 79a J-L. HAINAUT, Un Modèle de Description de Fichiers : le Modèle d'Accès. Notes de Cours, Institut d'Informatique, Namur, 1979.

HAI, 79b J-L. HAINAUT, Analyse du Cas PETITPAS- Dossier d'Analyse Organique. Première Partie : la Base de Données. Notes de Cours, Institut d'informatique, Namur, 1979.

HAI, 80 J-L. HAINAUT, Un Modèle Relationnel Généralisé. Institut d'Informatique, Namur, 1980.

HAI, 81a J-L. HAINAUT, Theoretical and Practical Tools for Data Base Design. Institut d'Informatique, Namur, 1981.

HAI, 81b J-L. HAINAUT, Traduction de Programmes ADL en COBOL/DML Codasyl 71, Application au Système DBMS-20 (DEC). Institut d'Informatique, Namur, 1981.

HAM, 77 M. HAMMER, W.G. HOWE, V.J. KRUSKAL, I. WLADAWSKI, A Very High Level Programming Language for Data Processing Applications. Comm. A.C.M., vol.20, 11, 1977, p. 832-840.

HEN, 80a P. HENDERSON, Functional Programming, Application and Implementation. Prentice Hall International Series in Computer Science, 1980.

HEN, 80b K.L. HENINGER, Specifying Software Requirements for Complex Systems : Techniques and Their Application. I.E.E.E. Transactions on Software Engineering, vol.SE-6, 1, 1980, p. 2-13.

HOA, 69 C.A.R. HOARE, An Axiomatic Basis of Computer Programming. Comm. A.C.M., vol.12, 10, 1969, p. 576-580.

HOA, 74 C.A.R. HOARE, Monitors : An Operating System Structuring Concept. Comm. A.C.M., vol.17, 10, 1974, p. 549-557.

HOA, 78 C.A.R. HOARE, Communicating Sequential Processes. Comm. A.C.M., vol.2I, 8, I978, p. 666-677.

JAC, 75 M.A. JACKSON, Principles of Program Design. Academic Press, I975.

JAC, 78 M.A. JACKSON, Information Systems : Modelling, Sequencing and Transformations. Third International Conference on Software Engineering, I978, p. 72-8I.

LAM, 79a A. VAN LAMSWEERDE, Programming Systems for non-Programmers : Some Wishes and Some Attempts. Report R390, M.B.L.E. Research Lab., Bruxelles, I979.

LAM, 79b A. VAN LAMSWEERDE, M. SINTZOFF, Formal Derivation of Stongly Concurrent Programs. Acta Informatica, I2, I, Springer-Verlag, I979, p. I-3I.

LAM, 80 A. VAN LAMSWEERDE, Concepts, Méthodes et Outils de l'Analyse Organique. Cours Institut d'Informatique, Namur, I980.

LAM, 8I A. VAN LAMSWEERDE, Quelques Principes pour le Développement d'une Architecture Logicielle. In : F. BODART, Introduction à la Conception de Systèmes d'Information, Ecole d'Eté de l'AFCET, I98I.

LAR, 7I LAROUSSE, Dictionnaire du Français Contemporain, Librairie Larousse, I97I.

LEM, 73 J-L. LEMOIGNE, Les Systèmes d'Information dans les Organisations. P.U.F., I973.

LIS, 75a B.H. LISKOV, S.N. ZILLES, Specification Techniques for Data Abstractions. I.E.E.E. Transactions on Software Engineering, vol.SE-I, I, I975, p. 7-I9.

LIS, 75b B.H. LISKOV, An Introduction to CLU. In : New Directions in

Algorithmic Languages, S.A. SCHUMAN (ed.), IRIA, 1975.

MAL, 71 R.A. MALLET, La Méthode Informatique. Hermann, 1971.

MEY, 80 B. MEYER, Sur le Formalisme dans les Spécifications. Globule, Bulletin du Groupe de Travail "Génie Logiciel" de l'AFCET, 1980.

MIL, 74 L.A. MILLER, C.A. BECKER, Programming in Natural English. IBM Res. Rep. RC5137, Yorktown Heights, New-York, 1974.

PAI, 79 C. PAIR, La Construction des Programmes. RAIRO Informatique, vol.13, 2, 1979, p. 113-118.

PAR, 72 D.L. PARNAS, A Technique for Software Modules Specification with Examples. Comm. A.C.M., vol.15, 5, 1972, p. 330-336.

PAR, 74 D.L. PARNAS, On a Buzzword : Hierarchical Structure. Proc. IFIP, North-Holland Publ., 1974, p. 336-339.

ROS, 77 D.T. ROSS, Structured Analysis (SA) : A Language for Communicating Ideas. I.E.E.E. Transactions on Software Engineering, vol.SE-3, 1, 1977, p. 16-34.

SIM, 74 H.A. SIMON, La Science des Systèmes : Science de l'Artificiel. EPI, 1974.

SPI, 78 J.M. SPITZEN, K.N. LEVITT, L. ROBINSON, An Example of Hierarchical Design and Proof. Comm. A.C.M., vol.21, 12, 1978, p. 1064-1075.

SOF, 76 SOFTECH INC., An Introduction to SADT. Waltham, MA, Document 9022-78, 1976.

TEI, 77 D. TEICHROEW, E.A. HERSHEY, A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems. I.E.E.E. Transactions on Software Engineering, vol. SE-3, 1, 1977, p. 41-48.

ULL, 80 J.D. ULLMAN, Principles of Data Base Systems. Pitman, 1980.

WIR, 73 N. WIRTH, Systematic Programming : an Introduction. Prentice-Hall Series in Automatic Computation, 1973.

WUL, 75 W.A. WULF, R.L. LONDON, M. SHAW, Abstraction and Verification in ALPHARD. In : New Directions in Algorithmic Languages, S.A. SCHUMAN (ed.), IRIA, 1975.