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

LES TRAITEMENTS :

LEURS STRUCTURES ET LEUR GENERATION

DANS LE PROJET REMORA

THESE

pour l'obtention du

DOCTORAT DE SPÉCIALITÉ INFORMATIQUE

Soutenue le 27 Mai 1977

par

Dominique LAMY

MEMBRES DU JURY :

Président : Mme C. ROLLAND
Examinateurs : MM. G. BENCI
J.C. DERNIAME
Mme O. FOUCAUT
M. J. MAROLDT

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

LES TRAITEMENTS :

LEURS STRUCTURES ET LEUR GENERATION

DANS LE PROJET REMORA

THESE

pour l'obtention du

DOCTORAT DE SPÉCIALITÉ INFORMATIQUE

Soutenue le 27 Mai 1977

par

Dominique LAMY

MEMBRES DU JURY :

Président : Mme C. ROLLAND
Examinateurs : MM. G. BENCI
J.C. DERNIAME
Mme O. FOUCAUT
M. J. MAROLDT

Madame ROLLAND , Directrice de l'UER de Mathématiques de NANCY II , est à l'origine du projet REMORA dans lequel s'insere ce travail . Puissent ces quelques lignes exprimer toute ma reconnaissance pour ses apports tant sur le plan de ma formation scientifique que sur le plan humain . Je lui suis gré de l'honneur qu'elle me fait de présider le jury .

Je remercie vivement Monsieur J.C.DERNIAME pour la formation en Informatique que je lui dois et sa participation au jury .

Mes remerciements vont aussi à Monsieur J.MAROLD qui a accepté de participer au jury .

Que Monsieur G.BENCI trouve ici l'expression de ma gratitude pour l'intéret qu'il a témoigné à ce travail et l'honneur qu'il me fait en participant à ce jury .

J'adresse mes remerciements à Madame O.FOUCAUT pour l'aide qu'elle m'a prodiguée et pour l'honneur qu'elle me fait en participant au jury .

Je tiens à remercier spécialement Monsieur et Madame J.THIERY pour l'aide scientifique , matérielle et morale qu'ils m'ont apportée lors de la réalisation de cette thèse .

J'ai beaucoup apprécié le climat règnant au sein du groupe REMORA . Que chacun soit remercié pour l'amitié et l'entr'aide qu'on y trouve .

Enfin je remercie Madame MARCHAND , Monsieur B.JACQUET ainsi que le Secretariat de l'IUT Informatique sans qui la réalisation matérielle de cette thèse aurait été impossible .

SOMMAIRE

INTRODUCTION : Survol du projet REMORA
I Choix d'une démarche assistée dans REMORA 1
I-1 De quoi sommes-nous partis 1
I-2 Notre objectif 2
I-21 Qu'est-ce que le SI 2
I-22 Qu'est-ce que concevoir et réaliser le SI 2
I-23 Quelles sont les qualités du SI auquel on veut aboutir 3
I-24 Qu'est-ce que le processeur 3
I-25 Qu'est-ce que le processus 5
I-3 Réflections sur les processeurs 5
I-31 Les ressources à leur fournir 5
I-32 Tentative de caractérisation des automates constituant les processeurs 6
I-4 Les choix dans REMORA 9
I-41 Le processus de conception 9
I-42 Le processeur 10
I-5 Les travaux réalisés 11
II Analyse des éléments du processeur de REMORA 12
II-1 Le modèle 12
II-11 Une option 12
II-12 Un souhait 12
II-13 Un choix d'approche du monde réel 12
II-14 Un choix de modèle conceptuel 15
II-2 La méthode d'analyse 16
II-21 Un choix 16
II-22 Les étapes 17
II-3 Le langage de conception 19
II-31 Un choix 19
II-32 Les caractéristiques 20
II-4 Les outils 22
II-41 Les types d'outils 22
II-42 Les outils d'aide à la conception dans REMORA 22
III Présentation de notre étude 28

PREMIERE PARTIE : LE LANGAGE DE CONCEPTION DE REMORA

INTRODUCTION I-1

I-1 Les caractéristiques des langages de formulation des problèmes I-1

I-1-1 Les langages de programmation de haut niveau I-2

I-1-2 Les langages d'analyse I-5

I-1-3 Les langages fonctionnels I-13

I-2 Le langage de conception de REMORA I-18

I-2-1 Un langage de formulation d'énoncés I-19

I-2-2 Un langage de définitions I-19

I-2-3 Un langage de définitions explicites I-21

I-2-4 Un langage modulaire I-21

I-2-5 Un langage exploitable par l'ordinateur I-22

I-3 La syntaxe du langage de conception I-22

I-3-1 L'exemple I-23

I-3-2 Le support du langage de conception I-24

I-3-3 La déclarativité du langage I-24

I-3-4 Les trois types de définitions I-27

I-3-4-1 Les définitions consécutives I-27

I-3-4-2 Les définitions alternatives I-30

I-3-4-3 Les définitions iteratives I-32

I-3-5 La modularité du langage I-37

I-3-6 Les définitions de la nature des mots I-38

I-3-7 Définition normalisée de la syntaxe du langage I-38

I-4 L'exemple I-41

DEUXIEME PARTIE : LES STRUCTURES DE TRAITEMENTS
LA GENERATION DES TRAITEMENTS

INTRODUCTION II-1
CHAPITRE II : LES FONCTIONS D'UN GENERATEUR II-4
II-1 La fonction de transformation des structures II-5
II-11 Les structures de traitement II-6
II-111 La structure conceptuelle des traitements II-6
II-111-1 Le concept de groupe conceptuel de traitement II-7
II-111-2 La nature d'un groupe conceptuel de traitement II-10
II-111-3 La combinaison de deux groupes logiques II-13
II-111-4 Prise en compte de la modularité de la conception II-16
II-111-5 La structure conceptuelle des traitements II-17
II-111-6 Notion de module conceptuel II-18
II-112 La structure logique des traitements II-18
II-112-1 Notion de paragraphe de traitement II-18
II-112-2 Liens entre les paragraphes de traitement II-20
II-112-3 Structure logique des traitements II-20
II-112-4 Notion d'unité opérationnelle de traitement II-22
II-113 La structure physique des traitements II-23
II-114 Conclusion II-26
II-12 Caractéristiques de passage d'une structure à une autre II-27
II-121 Passage de la structuration conceptuelle à la structuration logique II-29
II-121-1 Correspondance entre groupes conceptuels et paragraphes logiques II-29
II-121-2 Passage de la structure conceptuelle à la structure logique II-32
II-121-3 Structure logique de l'Unité Opérationnelle II-38
II-122 Passage de la structure logique à la structure physique II-39
II-123 Conclusion : Justification de l'élaboration des structures logiques de l'UOT au niveau logique II-41
II-2 La fonction de transformation des textes II-43
II-21 Les transformations successives des textes II-43
II-211 L'expression conceptuelle des traitements II-43
II-212 L'expression logique des traitements II-43
II-212-1 Définition II-44
II-212-2 Langage II-44
II-213 L'expression physique des traitements II-45
II-213-1 Définition II-45

II-213-2 Langage II-45
II-22 Les passages entre formulations successives de traitement II-46
II-221 De l'expression conceptuelle à l'expression logique II-46
II-221-1 Formulation du schéma logique II-46
II-221-2 Formulation des textes des paragraphes feuilles II-48
II-221-21 de réordonancement des définitions II-48
II-221-22 Exemple II-50
II-222 De l'expression logique à l'expression physique II-51
II-222-1 Formulation du schéma physique II-51
II-222-2 Textes des paragraphes II-53
II-23 Conclusion II-53
II-3 La réalisation des programmes II-54
CHAPITRE III : LE GENERATEUR DE REMORA II-57
III-I L'organisation du générateur II-57
III-11 Le générateur primaire.....II-58
III-111 Le module GP1 du générateur primaire II-60
III-112 Le module GP2 du générateur primaire II-60
III-113 Le module GP3 du générateur primaire II-62
III-12 Le générateur secondaire II-63
III-121 Le module GS1 du générateur secondaire II-64
III-122 Le module GS2 du générateur secondaire II-65
III-123 Le module GS3 du générateur secondaire II-66
III-2 Les algorithmes des différents modules du générateur II-67
III-21 Le générateur primaire II-67
III-211 Le module GP1 .....II-67
III-2111 Format des lignes de définition de la bibliothèque des définitions conceptuelles II-67
III-2112 Format des lignes de la bibliothèque des expressions conceptuelles II-68
III-212 Le module GP2 .....II-72
III-2121 Format des lignes de la bibliothèque de définitions algorithmiques (F"1) II-72
III-2122 L'algorithme II-73
III-213 Le module GP3 .....II-75
III-2131 Format des lignes de la bibliothèque des textes II-75
III-2132 Format des lignes de la bibliothèque des structures II-76
III-22 Le générateur secondaire II-78
III-221 Le module GS1 .....II-78
III-2211 Format des paramètres logiques de la bibliothèque des paramètres F4 .....II-79

III-2212 L'algorithme II-79
III-222 Le module GS2 .....II-80
III-2221 Format de la bibliothèque des paramètres physiques F4 II-80
III-2222 Format intermédiaire de l'expression de la structure physique des traitements II-81
III-223 Le module GS3 .....II-86
III-2231 Format de la bibliothèque des paramètres F4 .....II-86
III-2232 Algorithme II-86
III-3 Exemple de fonctionnement du générateur II-87
III-31 Génération primaire des modules conceptuels II-87
III-311 Génération primaire du module "FACTURE" II-88
III-312 Génération primaire du module "CALMBHT" II-89
III-32 Génération de l'unité opérationnelle "FACTURE" II-90
III-4 CONCLUSION II-94

SURVOL DU PROJET REMORA* ET PRESENTATION DE NOTRE ETUDE

* Ce texte est, en partie, celui du rapport final du contrat ATP numéro 76.124 (1977)

I - CHOIX D'UNE DEMARCHE ASSISTEE DANS REMORA

1.1. De quoi sommes-nous partis ?

L'origine de ce projet est pratique. L'expérience vécue de l'automatisation d'applications concrètes par des méthodes traditionnelles nous a convaincus, d'une part, de leur insuffisance, d'autre part, de la médiocre qualité des systèmes d'informations (SI) construits par accumulation de chaînes de traitement indépendantes.

Fondamentalement, nous sommes partis des critiques que les utilisateurs adressent aux systèmes d'informations qu'on leur réalise et qui ne les satisfont pas et de celles des concepteurs conscients de la faiblesse et de l'insuffisance de leurs techniques.

Pour ne citer que quelques exemples :
Du point de vue utilisateur :

- La portée des applications est limitée : cantonnées à l'automatisation administrative, elles répondent mal aux besoins de la gestion.
- La qualité du service rendu est faible :
  - Les applications sont rigides : les modifications sont difficiles à faire et les délais de réalisation sont longs et trop contraignants lors de l'exploitation.
  - Elles sont longues et coûteuses à concevoir et réaliser.
  - Les données fournies par l'informatique sont peu fiables (redondantes et incohérentes) et leur accessibilité est mauvaise ; la confidentialité est mal assurée.
- La réalité de l'organisation est mal représentée.

Du point de vue du concepteur les méthodes et les outils pour concevoir et réaliser les SI sont insuffisants ou même totalement absents :

- L'indépendance des programmes et des données est nulle.
- La distinction entre structures logiques et structures physiques est insuffisante.
- Les "méthodes" d'analyse sont réduites à des règles de bon sens cantonnées au stade de l'implémentation physique.
- Les techniques de maintenance sont archaïques.

Ces difficultés nous ont convaincus de la nécessité de chercher une méthode d'approche et d'appréhension des problèmes qui redonne à la conception sa primauté et à partir de laquelle puisse être assurée la construction du système d'informations.

1.2. Notre objectif

Nous nous proposons d'inventer un *processeur* qui gère un *processus* permettant de concevoir et de réaliser un SI ayant les qualités souhaitées par les utilisateurs (fidèle cohérent - extensible - complet - non redonnant).

1.2.1. Qu'est-ce que le SI ?

On peut définir le système d'information de l'extérieur, par l'usage que l'on en fait et les services qu'il peut rendre à l'organisation.

On peut aussi le définir de l'intérieur, par le modèle qui le décrit ou expression théorique de son architecture interne.

C'est de ce point de vue que nous nous plaçons pour admettre que le SI est un objet artificiel que les concepteurs créent pour représenter la réalité de l'organisation pour qu'à tout instant les utilisateurs des SI et gestionnaires de l'organisation puissent retrouver à travers le SI les visions de l'organisation qu'ils n'ont pu observer directement.

L'état du SI est à tout instant l'image de l'état de l'organisation : les utilisateurs du SI acquièrent la connaissance instantannée de la situation de l'organisation à travers la connaissance de l'état correspondant du SI et la connaissance de son fonctionnement à travers la succession des états du SI.

Pour disposer d'une image fidèle de la réalité à tout instant, il faut que soient représentés dans le SI :

. Les éléments de base de l'organisation et leur combinaison ; son "univers" au sens de PAIR (24), sa structure.

. Les règles de transformation qui valorisent les éléments de la structure.

. Les règles d'évolution qui conditionnent le passage d'un état de l'organisation à un autre par l'exécution d'un certain nombre de règles de transformation.

La description du SI d'un point de vue statique correspond à celle des points 1 et 2 . Nous verrons dans la suite de l'exposé que nous ferons l'hypothèse que l'univer peut être décrit par une structure de données et les transformations sur cet univers par des traitements appliqués à ces structures.

La description du SI d'un point de vue dynamique correspond à celle du point 3 précisant les éléments (évènements - règles - contraintes) qui conditionnent l'exécution des règles de transformation et l'évolution du SI vers un nouvel état c'est à dire de nouvelles valeurs des données.

1.2.2. Qu'est-ce que concevoir et réaliser le SI ?

Concevoir le SI c'est définir la solution technologique et administrative qui va permettre de le réaliser : c'est en donner une description à la fois statique et dynamique.

Nous appelons *description statique* la description des structures de données et de l'ensemble des traitements que peuvent subir les éléments de ces structures complétées par l'expression de leurs conditions d'implémentation sur un matériel choisi.

Nous appelons *description dynamique* la description des règles d'évolution.

Pour cela on part de la connaissance et des besoins de l'organisation, on les traduit en fonctions que devra remplir le SI et on définit les moyens pour assurer ces fonctions.

Réaliser le SI c'est à partir de la description précédente, créer les moyens, les mettre en place et les rendre aptes à fonctionner.

C'est à partir de la description statique créer la Base de Données et la Base de Programmes.

C'est à partir de la description dynamique créer la Base de Commandes.

L'architecture du SI que nous proposons est une architecture à trois composants :

la Base de données - la Base de programmes - la Base de commandes.

Elle est associée à un modèle de représentation statique et dynamique du SI très voisin de celui que suggèrent BBC dans (6). (Données - Contraintes intégrité - Règles d'évolution).

1.2.3. Quelles sont les qualités du SI auquel on veut aboutir ?

Le SI doit être :

fidèle : représentation sans distorsion et sans déformation de l'organisation.

cohérent : sans contradiction et sans ambiguité.

extensible : pouvant évoluer sans que l'adjonction de nouveaux phénomènes n'oblige systématiquement à remettre en cause les représentations déjà implantées.

complet : complet ne veut pas dire exhaustif (on ne prend en compte que ce qui est "utile").

Traduire tous les phénomènes que l'on désire connaître et de manière à pouvoir rendre compte de leur globalité comme de leurs aspects spécifiques.

non redondant : sans répétition.

1.2.4. Qu'est-ce que le processeur ?

Le nombre et la complexité des problèmes à résoudre dans la mise en place des SI a conduit les concepteurs à se demander dans quelle mesure un automate ne pourrait pas les aider, voire les suppléer dans le contrôle de l'ordonnancement des problèmes et de la prise en compte des solutions.

Nous nous interrogerons dans le paragraphe suivant sur les différents types de processeurs possibles (nous choisissons ce terme par similitude avec la conception des

systèmes d'exploitation) mais, dans une première approche, on peut répertorier trois catégories de solutions :

La première solution, sans doute la plus séduisante, est celle d'un *automate* fonctionnant sur le principe d'une "boîte noire" dont les entrées sont les énoncés fournis par les concepteurs et les sorties, les solutions automatisées.

C'est dans ce sens qu'interviennent les propositions d'ISDOS (36) puisqu'à partir des énoncés rédigés dans le langage d'expression des problèmes (PSL) le système se charge de la construction physique des programmes et de l'élaboration des fichiers.

C'est aussi le cas de SCAPFACE (22) puisque l'utilisateur définissant ses besoins par la description des imprimés obtient, en réponse, les programmes de calcul et d'édition de ces mêmes imprimés. Il n'intervient dans le processus de passage des énoncés aux solutions que pour corriger des incohérences ou des erreurs ou pour donner des informations complémentatires, mais jamais pour faire des choix.

Dans cette approche, il est tentant de confondre concepteur et gestionnaire futur utilisateur du SI, comme le préconisent SCAPFACE (22) et ISDOS (36).

Cette position implique, de notre point de vue, le développement de langages sinon naturels au moins quasi naturels grâce auxquels le gestionnaire pourra exprimer les problèmes tels qu'il les ressent.

Elle exige, de plus, que tous les problèmes rencontrés dans la conception et la réalisation puissent être résolus par un automate qui devra être capable de définir dans tous les cas le meilleur choix.

Ces deux raisons, font que, dans l'état actuel de la connaissance, nous considérion une telle solution irréaliste.

Une deuxième solution est celle d'un processeur exclusivement *humain* ; largement expérimentée on sait ce que cette solution apporte d'erreurs et d'insatisfactions.

Une troisième solution est celle d'un *processeur* constitué par le couple (*automates, individus*), avec le pilotage du processus de conception confié soit aux automates soit au individus. Elle associe l'automatisation assurée par les outils et la possibilit de certains choix laissée à l'homme, dans le cas où la décision automatique est encore impossible ou jugée risquée.

Comme Waters (40), Peccoud (30), Couger (10), Gero (15) ce choix est aussi le nôtre dans Remora.

Dans ce cas, les interventions successives des automates et des individus doivent être définies dans le cadre d'une démarche qui précise la nature et la référence des des problèmes à résoudre.

Il y a donc dépendance de la démarche, nous disons *processus* et du processeur : le choix d'un type de processeur induit le choix d'un certain type de processus.

1.2.5. Qu'est-ce que le processus ?

La définition d'une solution obtenue comme aboutissement de la phase de conception n'est pas unique ; elle est le résultat d'un grand nombre de choix qui ne sont pas tous optimisés, mais que les concepteurs ont le souci de rendre "les meilleurs" possible

Devant cet état de fait de la connaissance, il est indispensable de définir un processus qui précise les problèmes qu'il y a à résoudre, leur nature et leur ordonnancement.

Les processus de conception des SI actuellement à l'étude sont des processus en étapes, proposant, comme nous le verrons pour REMORA, une démarche progressive pour résoudre les problèmes de manière aussi indépendante que possible les uns des autres.

1.3. Réflexion sur les processeurs

1.3.1. Les ressources à leur fournir

Quel que soit le type de processeur auquel on se réfère, il a besoin de moyens, nous disons aussi *ressources* par analogie avec la théorie des systèmes d'exploitation.

A priori les *méthodes* et les *outils* sont les deux classes de ressources dont a nécessairement besoin tout processeur.

Par méthode nous entendons tout processus opératoire qui, agissant à l'image d'un manuel d'utilisation précise, à chacune des étapes du processus de conception réalisation, les entrées utilisées, la répartition des tâches entre les hommes et les automates, les sorties qu'il faut produire et la manière d'y aboutir.

Les outils sont des processus opératoires qui fournissent automatiquement une solution aux problèmes rencontrés dans le déroulement de la méthode.

Cependant comme l'usage des méthodes d'analyse l'a mis en évidence ; une méthode ne peut être réellement satisfaisante, si elle ne s'appuye pas sur des concepts précis. Cela est encore plus vrai en matière de conception des SI : concevoir c'est avant tout représenter des phénomènes du monde réel ce qui implique que l'on dispose d'un *modèle*. L'approche Base de données a confirmé l'intérêt de ce type de ressource que nous considérons, dans l'état actuel de la connaissance, comme indispensable.

Un *modèle* est un ensemble de concepts et de règles pour les manipuler permettant de représenter les phénomènes réels.

Les *langages* sont indispensables pour donner une formulation des concepts.

Ils constituent un moyen d'échange entre les automates et les individus.

Nous considérons le *quartet* (*modèle, méthode, langage, outil*) comme l'ensemble des types de ressources dont doit disposer tout processeur d'aide à la conception et à la réalisation des SI.

1.3.2. Tentative de caractérisation des automates constituant les processeurs

Pour établir une classification des automates et à travers cela, une classification des processeurs nous définirons, d'une part, leur niveau d'intervention, d'autre part, la nature de ces interventions.

α. Les automates peuvent jouer trois *rôles* distincts qui définissent trois *niveaux* d'intervention différents.

. Un *rôle technique* d'automatisation de tâche pour laquelle on a pu définir un algorithme de résolution ne comportant pas de prise de décision multicritère.

Exemples : Traduction d'un langage en un autre ;
traduction d'un opérateur par une séquence d'instructions en langage évolué ;
.....

. Un *rôle d'assistance* grâce auquel l'outil fournit à l'homme un éventail de solutions dans lequel celui-ci a la responsabilité du choix définitif.

Exemples : détection de synonymes dans le vocabulaire et proposition de solutions possibles ;
détermination d'une classification d'un ensemble de données ;
.....

. Un *rôle décisionnel* par lequel l'outil décide à la place de l'homme à partir d'algorithmes de choix paramétrés définis à l'avance.

Exemple : choix d'une structure logique de données à partir d'une expression conceptuelle.

Un automate n'assurant qu'un rôle technique est de niveau 1, un automate d'assistance est de niveau 2 et un automate capable d'effectuer des choix optimisés est de niveau 3.

β. Les automates peuvent intervenir à différents stades de la construction du SI déterminant ainsi une certaine *nature* de leur action.

Dans le processus de conception-réalisation on peut définir les deux stades de conception ou élaboration de l'expression de la solution — et de réalisation ou de mise en oeuvre de la solution, auxquels on peut associer deux types de rôles assurés par les automates :

d'aide à la conception des SI
d'aide à la réalisation des SI

Toutefois l'entrée dans un processus de conception, réalisation de SI est conditionnée par une prise de décision : *créer* (ou *restructurer* les moyens informatique de l'organisation) le SI ou *renoncer* à sa construction (ou sa restructuration).

Cette prise de décision s'accompagne de la définition de la *finalité* du SI à travers la précision des domaines de l'organisation qui y seront représentés et de son *économie* (budget alloué - coûts - bénéfices escomptés .....).

Elle peut être effectuée sous l'action d'un processeur muni d'automates de nature différente aux deux précédents et que nous disons

d'aide à la décision de construction du SI

Il y a évidemment des relations de compatibilité ou incompatibilité entre les rôles et les natures : un processeur d'aide à la décision de construction du SI ne peut être que de niveau 2 ou 3 ; un processeur d'aide à la réalisation sera le plus souvent de niveau 1.

γ . Le tableau ci-dessous donne quelques exemples de processeurs de natures différentes et intervenant à des niveaux distincts.

NIVEAU	FONCTIONS ASSUREES SUIVANT LA NATURE DE L'INTERVENTION DES AUTOMATES			AUTOMATES	PROJETS ou PRODUITS
	Aide à la décision	Aide à la conception	Aide à la réalisation		
1			Normalisation Codification	Langage "d'analyse" et générateur	"Méthodes d'analyse" PROTEE, ARIANE .....
1		Documentation		Langage de description, Outil de gestion de dictionnaires et Editeurs	"Méthodes d'analyse" MINOS .....
2	Simulation sur prototypes ou maquettes modèles			Langage de description Simulateur	MACSI Projet "MAESTRO" NANCY
2		Contrôle Intégration Pilotage Simulation		Langage de conception - Analyseur - Pilote - Dialogeur - Simulateur - Documenteur - Editeur	REMORA
3	Optimisation			Langage de description et optimiseur	ISDOS
3		Déduction Synthèse Optimisation		Langage de conception et SODA Analyseur - Dialogeur -	ISDOS SCAPFACE

1.4. Les choix dans REMORA

1.4.1. Le processus de conception

La démarche que nous avons retenue est une démarche en trois étapes indépendantes à laquelle nous avons abouti par l'analyse diachronique des méthodes actuelles de conception des systèmes d'information.

Les méthodes d'analyse ont montré l'intérêt qu'il pouvait y avoir à dissocier l'analyse des problèmes logiques ou fonctionnels de celle des problèmes physiques ou organiques.

Les méthodes de conception des bases de données (2), (5) , (9) , (34) ont démontré l'erreur qu'avaient commise les approches précédentes en ignorant les problèmes de représentation ou conceptuels.

L'étape conceptuelle est, dans Remora, l'étape de définition des problèmes dans une forme aussi indépendante que possible des moyens qui permettront de les traiter automatiquement ou non. On y exprime les *besoins des utilisateurs par des énoncés* sous référence aux moyens de résolution du problème. Dans notre méthode d'approche un problème étant défini par ses résultats nous proposons que cet énoncé soit fait par *la définition de ces résultats*.

L'objectif de la phase de conception est alors de parvenir à une image conceptuelle complète, cohérente et non ambigüe des phénomènes du monde réel pris en compte dans leurs aspects statiques et dynamiques.

L'étape de structuration est caractérisée par la recherche de l'ordre logique des opérations concernant les traitements et la prise en compte des paramètres concernant le choix des accès qui seront utilisés dans le SI pour atteindre les résultats définis dans la phase de conception. C'est d'après ces choix que peuvent être structurés les données et les traitements.

L'aboutissement de la phase est une image structurée du SI en construction qui est logiquement compatible avec les besoins des gestionnaires et leur prise en compte dans le temps.

L'étape d'implémentation et composition correspond à la prise en compte des moyens physiques pour implémenter les structures de données, pour réaliser les structure de traitement et gérer l'automate de fonctionnement définis au niveau précédent.

C'est le niveau d'obtention de l'image interne du SI à partir d'une base de données d'une base de programmes et d'une base de commandes.

Au choix d'un certain type de processus nous avons associé un choix de type de point d'entrée dans le processus : l'ampleur des phénomènes réels à prendre en compte nous a convaincus de la nécessité de donner au concepteur la possibilité de définir ses besoins de manière *sédimentaire*, c'est à dire, par couches successives dans le temps,

sans se préoccuper des définitions antérieures qui ont déjà été fournies.

Notons que pour être cohérent avec ce choix le processeur devra disposer de ressources lui permettant d'assurer l'intégration de chaque problème dans le contexte de ceux qui ont déjà été définis et résolus en s'assurant que l'ensemble reste une image fidèle et cohérente de la réalité.

1.4.2. Le processeur

Nous intéressant au seul processus de conception-réalisation des SI (et non à celui de décision sur la construction qui nécessite des connaissances en théorie de l'organisation et gestion notamment, que nous n'avons pas), nous avons renoncé à un processeur d'aide à la conception-réalisation de niveau 3, au profit d'un processeur mixte (individus, automates) de niveau 2, assurant un rôle d'assistance au choix et d'aide technique.

Ce processeur est organisé autour d'un outil privilégié qui orchestre, le processu assure l'ordonnancement des tâches humaines ou automatiques et l'appel aux différentes ressources dont il dispose.

Cet outil appelé *pilote* :

- accepte que le concepteur considère les uns après les autres les phénomènes réels qu'il souhaite prendre en compte, et les définisse en plusieurs couches dans une démarche par étapes.
- l'aide, dans cette action, en lui proposant à mesure qu'il progresse les moyens de résolution appropriés (modèle - méthode - langage - outils).

1.5. Les travaux réalisés

Bien que notre ambition soit de résoudre, à terme, l'ensemble des problèmes nous avons pour l'instant donné la *primauté à la conception*.

Conscients qu'une large partie de l'échec des "méthodes d'analyse" doit être imputé à l'ignorance des problèmes de représentation, nous nous sommes attachés à étudier l'étape conceptuelle du processus de conception-réalisation en cherchant notamment à définir

le modèle conceptuel de représentation de la réalité,
le langage conceptuel qui lui correspond,
la méthode d'analyse et les outils d'aide à la conception

Le chapitre suivant sera consacré à l'étude de ces quatre ressources dont dispose le processeur de Remora au niveau conceptuel.

II - ANALYSE DES ELEMENTS DU PROCESSEUR DE REMORA

2.1. Le modèle

2.1.1. Une option

Les chercheurs en conception de B.D proposent de représenter la sémantique du réel de l'organisation que l'on veut prendre en compte, par des données et des combinaisons de données.

L'expression formelle obtenue, appelée schéma conceptuel, est une expression de types qui permet de rendre compte, dans une *perspective exclusivement statique* de ce qui est à un instant donné, dans le réel organisationnel.

Notre objectif a été de définir un modèle permettant une représentation formelle des phénomènes du monde réel qu'on a décidé de considérer dans le SI, *dans la totalité de leurs aspects statiques et dynamiques* c'est à dire sous forme de structures de donnée de traitements de ces structures et de paramètres d'évolution.

2.1.2. Un souhait

L'expression de cette représentation que nous appelons aussi *schéma conceptuel* (S.C est une traduction du monde réel en termes de types. Elle exclut toute considération technique correspondant à des préoccupations d'implantation de telle sorte que soit assurée l'indépendance des solutions d'implémentation et que soient possibles divers choix d'implantation pour un même schéma.

Elle est l'aboutissement de l'analyse et à ce titre, doit être une expression *fidèle* (traduction sans biais et sans déformation du réel organisationnel), *complète* (de telle sorte que chaque utilisateur retrouve dans la représentation le point de vue particulier qui l'intéresse), *cohérente* (sans contradiction ni ambiguité), *extensible* (pour que la prise en compte de nouveaux phénomènes n'oblige pas systématiquement à remettre en cause les représentations déjà implémentées) et *simple* (recourant à un minimum de concepts) des phénomènes pris en compte par l'organisation.

2.1.3. Un choix d'approche du monde réel

La définition des concepts du modèle dépend de *la vision que l'on a du monde réel*. Beaucoup d'études qui sont toutes exclusivement orientées structuration des données ont porté sur la modélisation du monde réel (1, 8 , 23 , 35 ...). Sans doute est-il vain comme le remarquent BBBC dans (6) de vouloir proposer un modèle unique du monde réel. Chaque individu de l'organisation le perçoit à sa manière, en fonction des préoccupations qu'il a.

Aussi, c'est moins en termes de modèle que nous approchons le monde réel, qu'en termes d'inventaire des phénomènes qu'il conviendra de prendre en compte dans la représentation.

Nous faisons l'hypothèse que le monde réel est constitué d'occurrences *d'objets* (la collection des employés de l'entreprise, les clients ... de l'entreprise inclus dans la liste des personnes avec lesquelles elle est en relation, les produits en stock à la date du 30 Septembre 1976 .....).

A un instant donné, le monde réel est accessible par la description de sa *situation* c'est à dire la description spatiale et temporelle des objets qui le constituent. Une situation exige que l'on décrive les objets eux-mêmes, à travers les *propriétés* qu'ils ont, mais aussi les uns par rapport aux autres à travers les associations d'objets dans lesquelles chacun d'eux joue un rôle spécifique. La représentation des objets et des associations entre objets permet de rendre compte dans une perspective *statique* de ce qui est, à un instant donné dans le réel organisationnel..

Dans une vision *dynamique* on doit aussi représenter les transformations subies par les objets ou leurs associations. Ces transformations traduisent autant la nature des objets que les propriétés explicites qu'on peut leur attribuer. Elles sont engendrées par des *évènements* déclanchés soit par des agents extérieurs à l'organisation et qui sont en relation avec elles (commande d'un client, ... demande d'une entreprise d'être dans la liste des fournisseurs pour une gamme de matières .....) soit par l'organisation elle-même. Dans ce cas, il peut s'agir d'évènements institutionnels intervenant à des fréquences établies (facturation en début de chaque semaine des envois faits aux clients de la semaine précédente, exécution de la paie tous les 20 de chaque mois .....), ou d'évènements intervenant à des moments irréguliers et provoqués par les agents de l'organisation à la suite de décisions (lancement d'un lot de fabrication, embauche du personnel .....).

Un évènement qui entraîne la transformation de l'état d'un ou plusieurs objets peut engendrer une séquence plus ou moins complexe et immédiate d'autres transformations qui lui sont logiquement rattachées. (Ex. dans le cas de la paie, le fait que l'on soit le 20 du mois va engendrer plusieurs transformations qui vont permettre de calculer le salaire mensuel de chaque membre du personnel et aussi le cumul de S.S., le cumul de base d'imposition, ...).

Il est évident que notre système de représentation devra permettre de rendre compte de l'interdépendance logique et temporelle des transformations liées à un évènement, de même qu'il devra permettre d'exprimer qu'un même objet peut subir des transformations différentes, soit en fonction de la nature des évènements qui le concernent, soit en fonction de leur séquence.

Traduire les transformations subies par les objets ou les associations d'objets c'est définir la plupart des états possibles que peut prendre le réel organisationnel en fonction d'un état initial donné et d'un ensemble d'évènements probables.

Traduire ces transformations c'est aussi décrire les caractéristiques dynamiques des objets. Leurs caractéristiques statiques et celles de leurs associations sont traduites par la description des *propriétés*.

Il n'est pas possible de s'intéresser à tous les états possibles et les transformations qui sont très nombreux. Ceux qui vont permettre de satisfaire les besoins d'information des utilisateurs du SI méritent d'être définis de façon précise et explicite. Ces besoins seront traduits sous la forme de messages indiquant la nature et le contenu des échanges entre le SI et les demandeurs d'informations. De la même manière seront décrits les échanges entre le perçu organisationnel et le SI pour que se fasse l'ajustement permanent entre le réel et sa présentation et pour que l'on détermine selon quelle procédure, avec quel rythme et avec quel décalage, les phénomènes survenus dans le réel seront repris dans le SI. L'ensemble de ces propriétés statiques et dynamiques sera exprimé par des *définitions*.

Nous retenons donc six catégories de phénomènes qui devront figurer dans le SI.

Objet : C'est un constituant de l'organisation concret ou abstrait qu'un individu ou un groupe conçoit et perçoit comme ayant une existence extrinsèque et dont la connaissance présente pour lui un intérêt qui justifie que le système d'information en donne une représentation.

Association d'objets : C'est une collection de deux ou plusieurs objets qui permet de représenter des situations complexes dans laquelle chacun d'eux joue un rôle particulier qui doit être spécifié.

propriété : c'est une qualité que l'on reconnaît aux deux classes d'éléments antérieurement définies.

définition : c'est l'expression des transformations relatives soit aux objets soit aux associations.

message : c'est la description des échanges entre le SI et l'environnement qui l'alimente ou qui l'utilise.

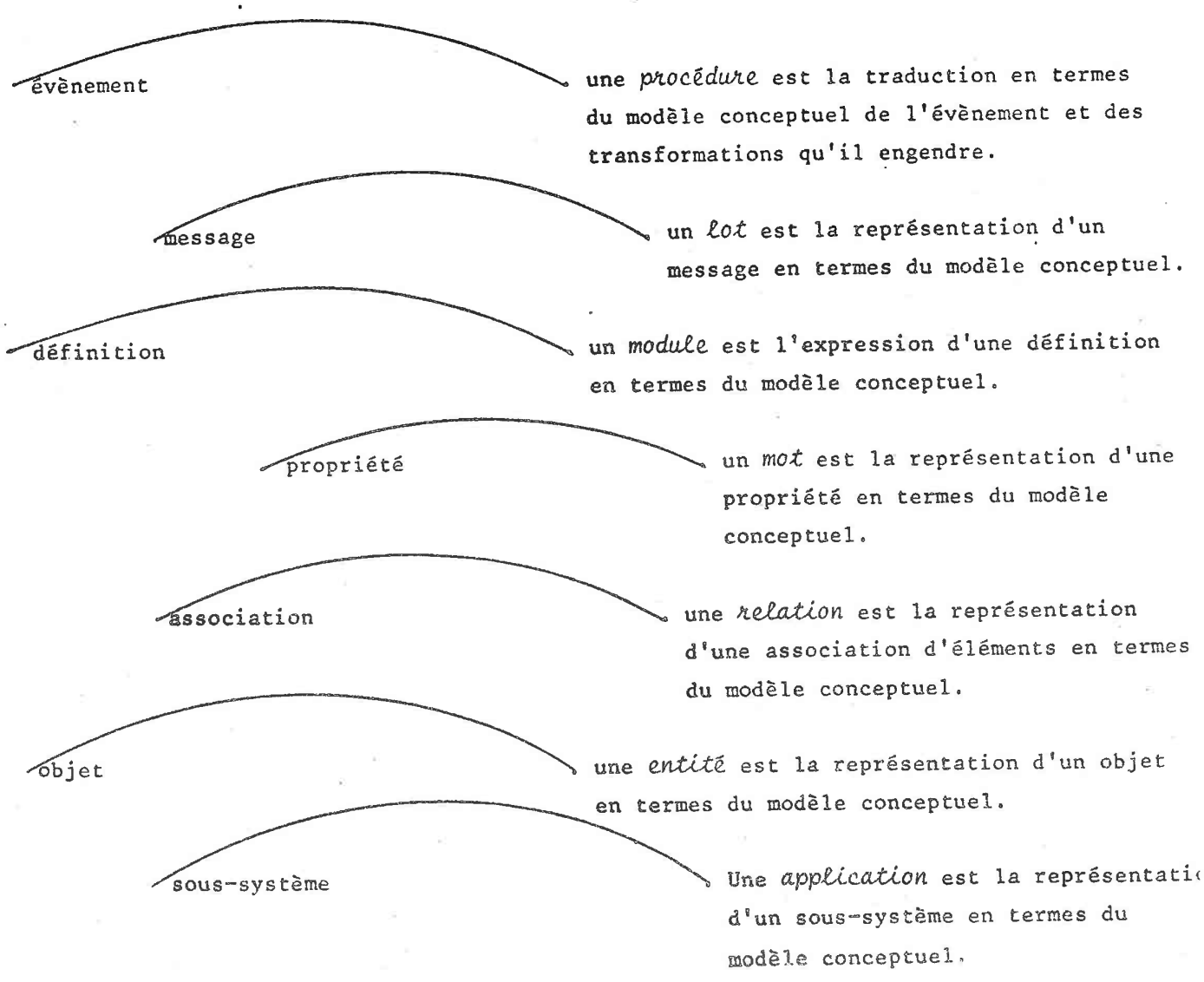
L'approche d'un problème de gestion telle que nous l'aborderons au paragraphe suivant nous a conduits à n'appliquer la méthode de conception qu'à un sous-ensemble des phénomènes de l'organisation considéré comme un tout, isolé et indépendant du reste appelé *sous-système*.

sous-système : c'est un domaine de l'organisation qui concerne une même catégorie d'objets (produits - clients - fournisseurs - employés) ou une même catégorie d'évènements relatifs à une préoccupation de l'organisation (embaucher - acheter - vendre - produire - concevoir).

On peut trouver dans PEREA (31) une méthode d'approche modulaire de l'organisation qu'induit son découpage en sous-systèmes.

2.1.4. Un choix de modèle conceptuel

A l'approche du monde réel nous faisons correspondre un modèle de représentation qui appartient à la classe des modèles conceptuels. Ce modèle comporte sept concepts que nous avons fait le choix de mettre en correspondance biunivoque avec les éléments constitutifs du monde réel.



On trouvera dans la thèse de THIERY (37) une analyse détaillée de ce modèle et en particulier des caractéristiques retenues pour la définition des types de notre modèle conceptuel.

2.2. La méthode d'analyse

2.2.1. Un choix

L'option prise par les chercheurs en conception de bases de données consiste à proposer un langage de définition qui permette la description du schéma conceptuel.

Cette expression est représentative de l'aboutissement de la réflexion conceptuelle c'est l'image du résultat de l'étape mais aucune indication n'a été fournie au concepteu sur la manière d'y parvenir.

Il est de notre point de vue, indispensable de proposer une *méthode* qui facilite au concepteur l'appréhension des phénomènes réels ; lui précise le point de départ de la réflexion, le guide vers la solution conceptuelle et l'aide à en formuler l'expression dans un langage de définition.

. Notre point de départ se situe dans le domaine de perception des problèmes le plus quotidien des gestionnaires, celui des *besoins*.

L'expérience nous a montré, en effet, que dans le domaine de la gestion administrative des organisations c'est en termes de *besoins* exprimés dans des *messages* et en réponse à un *évènement* que le gestionnaire perçoit les problèmes.

Et dans cette optique il nous semble légitime de partir de l'hypothèse que le gestionnaire est capable d'exprimer ses besoins par des *résultats*. Ce mot doit être pris au sens large : les résultats sont les éléments produits par l'exécution d'une procédure déclenchée par un évènement.

Cette hypothèse, acceptée volontiers par les théoriciens de la programmation (24) induit une approche que certains qualifient d'analytique (17) ou par les sorties (33) ou top-down (7) ou de conception descendante (11, 13).

. La méthode guide le gestionnaire dans l'expression de ses besoins en lui proposant des règles de *définition des résultats*.

. L'expression de ces définitions est facilitée par les caractéristiques du langage de conception qui lui permet *d'énoncer* les définitions dans un ordre déclaratif correspondant au cheminement de sa réflexion.

2.2.2. Les étapes

Nous proposons au concepteur de travailler en trois étapes : la première correspond à la *description des résultats* produits par une procédure à travers la description des messages qui les véhiculent ; la seconde est l'étape de *définition des résultats* ; la troisième permet de *compléter les informations fournies sur les mots introduits* dans les deux étapes précédentes.

. La première correspond à la précision du point de départ : dans le cadre d'une procédure déclenchée par un évènement il s'agit de *décrire les messages* qu'elle produit. Cette description s'effectue sur des *descripteurs de documents* qui permettent de définir leur contenu sous la forme d'une liste de mots représentant les propriétés des objets de l'organisation concernés par la procédure.
On trouvera dans la thèse de W. PEREA (31) une étude sur la structuration des états et, dans une optique plus large, sur la normalisation de l'édition, justifiant le choix de l'approche préconisée (appréhension d'un état par son dessin) et la manière de décrire les messages sur les descripteurs de documents.

. La seconde est une étape clé : celle du guidage du concepteur dans la recherche de la solution conceptuelle. Elle le conduit, à partir d'une sélection des *résultats* dans les listes précédentes, à en trouver les *définitions* puis à les *formuler par des énoncés* sur les descripteurs de traitement, supports du langage de conception.

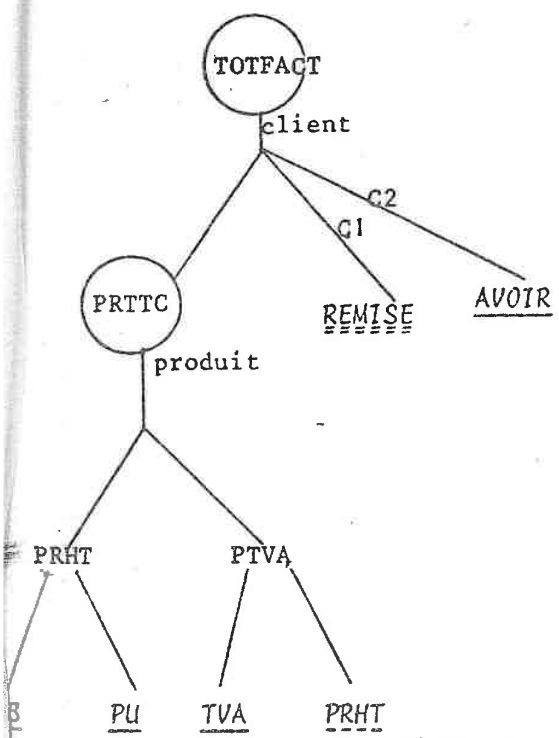
Le raisonnement que nous proposons au concepteur pour définir les résultats est un raisonnement *déductif* le conduisant à agir par raffinements successifs :
Partant du résultat qui est le point de départ de sa réflexion, le concepteur se demande comment le définir et introduit des intermédiaires qui à leur tour doivent être définis.

Il s'appuie dans sa progression étape par étape sur la notion de *nature** d'un mot (31 , 37) et définit le résultat sous la forme d'une structure arborescente de mots : le sommet de l'arborescence est le mot résultat à définir ; sur les niveaux suivants figurent les résultats intermédiaires ; les feuilles de l'arborescence sont des mots de nature "donnée".

(*) Nous distinguions trois natures : *résultat* (mot produit par une procédure) ; *donnée* (mot évalué à l'extérieur de la procédure) ; *résultat intermédiaire* (mot introduit par le processus de définition d'un résultat).

W. PEREA propose dans sa thèse de schématiser ce raisonnement par un *arbre de résultats* qui fait apparaître, en outre le *type*** des mots introduits. Nous verrons l'utilité de cette typologie, voisine de celle proposée en (24) ou (39) dans le paragraphe suivant et pour le passage à l'énoncé des définitions sur les descripteurs de traitements.

La figure 1 donne un exemple d'arbre dans le cas simple de l'analyse du résultat TOTFACT (montant total d'une facture) rencontré dans la description du message "FACTURE" d'une procédure de gestion des commandes.



- *Le total de la facture est du type liste de résultat associé à l'entité client.*
  Il est défini à partir des montants PRTTC par produ toutes taxes comprises de la remise (REMISE) dont peut bénéficier le client et de l'Avoir (AVOIR) éventuel de ce client.
- *Le prix PRTTC par produit est une liste de résulta associée à l'entité Produit.*
- *La définition conditionnelle de la remise sera explicitée plus tard.*
- *L'avoir est une donnée locale conditionnée.*
- Tout montant par produit est défini à partir d'un montant hors taxe (PRHT) et d'un montant de taxes (PTVA)
- Le prix de revient hors taxe s'évalue en fonctions des *deux données locales NB et PV.*
- La taxe s'évalue en fonction d'un taux qui est une donnée locale et du résultat intermédiaire PRHT.

- Figure 1 : Exemple d'arbre de résultats -

(**) Cette typologie permet de classer un mot en *inconditionnel* s'il apparaît toujours sur 1 message décrit ; *conditionnel* si son apparition sur message ou dans un niveau de l'arbr est conditionnée ; *liste de résultats* s'il s'agit d'un ensemble de mots indépendants les uns des autres et produits par une même séquence de définitions.

Nous verrons dans le paragraphe sur le langage les lois du passage de l'arbre à l'énoncé des définitions correspondantes sur les descripteurs de traitement.

. La troisième consiste à compléter les informations sur les mots qui n'ont pas été données sur les descripteurs de traitements. Ce sont leurs *caractéristiques sémantiques* qui les définissent comme représentant la propriété d'un objet ou d'une association d'objets du monde réel. Ce sont aussi leurs *caractéristiques dynamiques* qui les relient aux évènements ; elles sont portées sur un *descripteur de mot*.

Cette étape est encore à l'étude (travaux de M. Krieguer et M. Chesseron).

Ces trois étapes permettent d'intégrer dans une même méthode l'analyse des données et l'analyse des traitements sous leurs deux aspects statique et dynamique .

2.3. *Le langage de conception*

2.3.1. *Un choix*

Les langages de programmation et les langages d'analyse ont été induits par le désir *d'automatiser* des applications ; ils s'appuient sur un modèle d'implémentation interne, modèle composé d'objets informatiques sous forme de types de données (entiers, réels, booléens) et de types d'opérations (arithmétiques et logiques) dans les langages de programmation, sous forme plus abstraite et imagée de ces types dans les langages d'analyse (structures de files de données (18), opérateurs (27) .....).

Les langages fonctionnels ont un esprit résolument différent. Au lieu de chercher à exprimer la solution sur un modèle interne, ils tentent d'exprimer les problèmes à partir de la perception réelle que peut en avoir l'individu. Ils ont pour objectif de *représenter* des phénomènes et s'appuyent sur des modèles spécifiques qui sont totalement déconnectés de la traduction informatique de ces phénomènes par des données structurées et des opérations sur ces structures.

Les chercheurs en matière de base de données ont proposé des langages de définition des données (LDD) s'appuyant sur des modèles de représentation statique de l'organisatio à partir d'objets, de leurs propriétés et de leurs associations. Ces langages ne permettent de décrire qu'une partie des problèmes perçus mais sont de nature "fonctionnelle".

Les théoriciens de l'informatique raisonnent sur des modèles mathématiques (ensembles, opérations algébriques) et s'arrêtent à un énoncé implicite des problèmes (Trouver i tq t(i) = a) avec le désir de déduire de manière automatique de cet énoncé, l'algorithme de résolution.

D'autres s'intéressent à l'utilisation des SI et proposent des langages de très haut niveau permettant aux utilisateurs de travailler sur des concepts simples. Par exemple PAIR et GRANDMOUGIN (29) proposent un tel langage grâce auquel un utilisateur peut raisonner sur des structures quelconques et de manière non procédurale avec la seule connaissance de règles de mathématiques élémentaires, des notions d'ensemble et de fonction.

Le langage de conception de Remora appartient à cette classe.

Il a pour vocation de permettre une expression formelle des problèmes à l'image de la perception qu'en a l'individu. Il s'appuie, pour cette raison, sur le *modèle de perception du monde réel* que nous avons précédemment décrit, et, parce qu'il concerne le domaine des organisations revêt certaines caractéristiques spécifiques.

2.3.2. *Les caractéristiques*

. C'est par hypothèse un *langage d'énoncé* des problèmes de gestion et non un langage de formulation de la méthode algorithmique de leur résolution. L'énoncé est purement statique, l'ordre dans lequel il est fait ne reflète aucunement l'ordre d'exécution du calcul qui lui correspondra. C'est un énoncé en mode *déclaratif* où l'on décrit ce que l'on souhaite voir réaliser indépendamment de la façon dont ce sera effectivement réalisé. L'ordre des déclarations est à l'image de l'analyse des besoins dans la méthode d'approche que nous avons proposée au paragraphe précédent. Ainsi l'énoncé a-t-il une structure arborescente correspondant à la structure des résultats déduite de l'analyse ; soit pour l'exemple proposé :

Module : FACTURATION		
NIVEAU	DESIGNATION DES ELEMENTS	IDENT.
01	Net à payer/client	TOTFACT
02	Prix toutes taxes/produit	PRTTC
03	Prix net/produit	PRHT
04	Prix unitaire	
04	Nombre	
03	Taxe sur les produits	PTVA
04	Taux TVA	
02	Remise aux bons clients	REMISE
02	Avoir éventuel du client	AVOIR

Figure 2 : *l'arborescence de l'énoncé sur le descripteur de traitement* -

. C'est un *langage de définitions* ; puisque en accord avec la méthode que nous avons choisie, l'énoncé des besoins correspond à la définition des résultats.

Les lignes successives du descripteur définissent les noeuds des branches et la ou les données de feuilles.

A un groupe de lignes de nombres niveaux croissants sur le descripteur de traitement correspond une branche de l'arbre des résultats.

En outre, à la *typologie des mots* sur laquelle s'appuie le raisonnement déductif correspond une *typologie des définitions**, ce qui facilité la formulation de l'énoncé sur le descripteur de traitement.

Enfin les définitions sont *explicites* c'est à dire que tout mot est défini en fonction de ceux qui le définissent (salaire net = salaire de base + avoirs - retenues).

La figure 3 donne, à propos de l'exemple cité, un aperçu des formes possibles de définitions dont l'étude détaillée est proposée dans la thèse développée ici.

Module : FACTURATION

NIVEAU	DESIGNATION DES ELEMENTS	CONDITION	ENTITE		DONNEE	IDENT.	APPEL
01	Net à payer/client		CLIENT			TOTFACT	
02	Prix toutes taxes/produit		PRODUIT			+PRTTC	
03	Prix net/produit			PUxNB		+PRHT	
04	Prix unitaire				PU		
04	Nombre				NB		
03	TVA/produit			TVAxPRHT		+PTVA	
04	Taux de TVA				TVA		
02	Remise aux bons clients	C-CLIENT='BON'				-REMISE	REMIS
02	Avoir du client	C-AVOIR='OUI'				- AVOIR	

Figure 3 : *Les définitions d'un énoncé de problème*

* A tout mot de nature *inconditionnelle* est associée une définition *consécutive* (expression déclarative des calculs typiques exprimés au moyen d'expressions arithmétiques et de sommes algébriques) ; à tout mot *conditionnel* correspond une définition *alternative*, à une *liste de résultats* est associée une définition *itérative* (31).

. C'est un *langage modulaire* parce qu'il reflète la possibilité de faire d'un problème une analyse elle-même modulaire . Pendant la construction de l'arbre des résultats, le concepteur peut souhaiter reporter à plus tard la définition d'un résultat intermédiaire, soit parce qu'elle est trop complexe, soit parce qu'elle est encore trop floue (c'est le cas, sur l'exemple, de la REMISE). Pour se plier à cette exigence de l'analyse, le langage prévoit que l'on nomme (colonne APPEL du descripteur) le module définissant le résultat cité sur la ligne (REMISE).

. C'est enfin un *langage exécutable* par l'ordinateur ; le caractère explicite des définitions facilite le passage de l'énoncé à l'algorithme de résolution ; c'est le travail que réalise le *générateur de REMORA* dont nous verrons les caractéristiques au paragraphe suivant.

2.4. Les Outils

2.4.1. Les types d'outils

Notre processus méthodique d'analyse des phénomènes réels exige que le processeur d'aide à la conception et à la construction des SI dispose d'outils lui permettant d'obtenir la description conceptuelle à partir de l'analyse des énoncés fournis sur les différents descripteurs.

Cette description est implémentée dans REMORA sous forme d'une base de données appelée *Base des Objets Elémentaires*.

Elle est créée et gérée par des outils que nous appelons *outils de construction du schéma conceptuel* qui déroulent des algorithmes de dépouillement des descripteurs de reconnaissance des objets du monde réel - de transposition en termes conceptuels et de valorisation des caractéristiques. On en trouvera une analyse détaillée dans la thèse de THIERY (37).

Ce sont cependant les *outils d'aide à la conception* qui déterminent la qualité du processeur à ce niveau.

2.4.2. Les outils d'aide à la conception dans REMORA

Peu de travaux (28, 40) proposent à l'heure actuelle des outils d'aide à la conception à ce niveau et dans le cadre d'une méthode intégrant les aspects "données" et "traitements". Aussi nous sommes nous attachés, dans un premier temps, d'une part à définir les fonctions qu'un processeur d'aide à la conception doit assurer, d'autre part à proposer nos idées sur les moyens envisageables pour les assurer (THIERY (37)).

Nous avons conduit notre étude en examinant comment les caractéristiques du processus de conception (sédimentation - multiplicité des concepteurs - des points de vue - fragmentation) pouvaient entraîner une non satisfaction des qualités assignées au schéma conceptuel (cohérence - fidélité - complétude - non redondance - extensibilité) ; puis en définissant les fonctions qui permettraient d'éliminer les sources d'erreurs.

Nous sommes ainsi arrivés à la conclusion que tout processeur d'aide à la conceptio doit remplir trois fonctions de CONTROLE - d'INTEGRATION - de DOCUMENTATION.

α. *La fonction CONTROLE*

Si nous appelons *énoncé* l'image descriptive dans le schéma conceptuel d'un phénomène du monde réel, la fonction contrôle recouvre l'ensemble des actions permettant de s'assurer de la correction de cet énoncé.

Un énoncé peut être déclaré correct s'il est cohérent et s'il est une image fidèle du phénomène réel décrit ; ce qu'on peut essayer de vérifier dans un premier temps par :

α_1. *des contrôles sémantiques de l'énoncé* visant à répondre à la question : <<l'énoncé est-il cohérent, a-t-il un sens ?>>

Mais comment définir "le sens" d'un énoncé ?

Une solution partielle consiste à utiliser le fait qu'à une structure conceptuelle donnée corresponde une structuration en éléments dont les caractéristiques doivent être respectées dans tout énoncé.

On peut aussi imaginer de transposer, au niveau conceptuel, les aides à la mise au point que l'on a vu apparaître dans les méthodes d'analyse (18, 19), telles que la simulation. Cette solution est utilisée dans Remora pour produire une maquette d'état à partir d'un énoncé descriptif du document à éditer.

α_2. *des contrôles sémantiques de fidélité* grâce auxquels on doit pouvoir répondre à la question : <<l'image du phénomène réel perçu à travers l'énoncé est-elle une représentation fidèle de la réalité ?>>

Les moyens dont on dispose actuellement pour assurer de tels contrôles sont très limités : c'est souvent le concepteur qui la vérifie en interprétant lui-même le résultat de sa description.

Si l'on songe à des solutions automatiques elles consistent à mettre en parallèle une définition sémantique du phénomène réel qu'aurait fournie le concepteur et l'expression sémantique que l'on peut déduire de l'énoncé.

La sémantique pourrait être exprimée au moyen d'un langage naturel ou pseudo-naturel reconnaissable par un automate ; l'expression sémantique déduite de l'énoncé pourrait être composée par le système à partir des caractéristiques de l'image de l'objet représenté.

Cette solution s'apparente aux méthodes de preuves de programmes développées en programmation : les insertions sont une autre expression de l'algorithmique du programme qui doivent fournir des résultats identiques à ceux normalement obtenus par l'exécution du programme.

Un énoncé est un tout décrit en une seule fois et sémantiquement indissociable (exemple : le module descriptif d'une définition dans Remora) qui s'exprime à partir d'objets élémentaires de la structure conceptuelle dont il fournit en fait la définition (exemple : le module exprime les propriétés des mots).
On peut imaginer, dans un deuxième temps :

α_3. *les contrôles sémantiques des éléments*

qui tendraient à répondre à la question symétrique de α_2 au niveau de chaque élément de l'énoncé pris individuellement.
Mais "comment reconnaître l'expression sémantique d'un élément par un automate ?"
Une solution peut consister à utiliser les caractéristiques associées aux éléments et à considérer que le sens d'un élément est défini par les valeurs de l'ensemble ou d'un sous-ensemble des caractéristiques significatif. (Exemple : la définition en clair ; la nature d'un mot et son rattachement à une entité donnent un sens plus précis et plus sûr de l'élément que chacune des caractéristiques prise isolément).

On peut aussi définir des règles de compatibilité ou d'incompatibilité sur les champs des valeurs des caractéristiques et vérifier que ces règles sont ou non respectées (ex : un même mot dans Remora ne peut avoir une nature "donnée" et une nature "résultat").

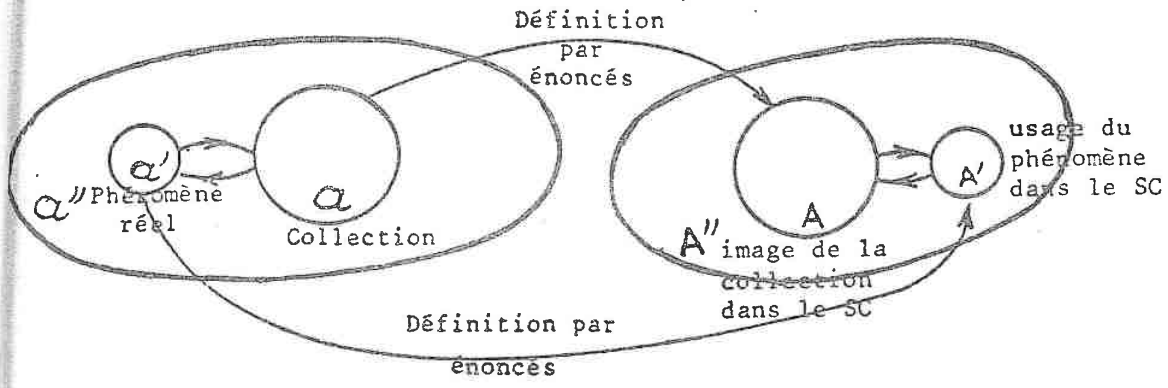
Pour être complète cette liste doit comporter des contrôles syntaxiques au sens habituellement défini en programmation.

β. *La fonction INTEGRATION*

Elle recouvre l'ensemble des actions à entreprendre pour s'assurer de la cohérence de l'insertion d'un énoncé dans l'ensemble des énoncés déjà admis dans le schéma conceptuel.

Plus précisément, partant de l'hypothèse qu'une certaine collection de phénomènes réels a déjà été décrite par une collection A d'énoncés qui a toutes les propriétés requises (la collection est cohérente - non ambigue et représente fidèlement la réalité)

si a' est un phénomène apparenté à la collection a et si A' est son image cohérente dans le schéma conceptuel (figure 4)



- Figure 4 : *La fonction Intégration* -

la fonction intégration a pour objectif de vérifier que la nouvelle collection d'images " est,d'une part,cohérente et complète et que,d'autre part, elle est une image fidèle de la collection a", en d'autres termes, que les relations de a' avec a ont été correctement traduites sous la forme des relations de A' avec A .

β_1. *les contrôles sémantiques des éléments*

Lorsqu'un élément est introduit par un énoncé il s'agit de s'assurer qu'il a un sens, non pas, par rapport à lui-même mais par rapport au contexte le plus large de l'ensemble des éléments déjà acceptés dans le schéma conceptuel.

Les contrôles de ce type s'appuyent sur la désignation des éléments et doivent donner une réponse aux deux questions :

. si l'élément est nouveau (il n'y a pas cette désignation dans le répertoire des désignations) n'y a-t-il pas un élément de même sens dont la désignation soit différente

. si l'élément existe déjà, a-t-il le même sens, c'est à dire représente-t-il le même objet réel ?

Les premiers types de contrôles correspondent à la détection des *synonymes involontaires* ; les seconds à la détection des *polysèmes*.

Les moyens que l'on peut proposer pour détecter les cas de polysémie et les cas de synonymie sont de même nature, puisqu'il s'agit de reconnaître le sens d'un élément par un autre biais que sa désignation.

On peut envisager comme en α_3 d'utiliser l'ensemble des caractéristiques de l'élément (ou un sous-ensemble).

On peut également, tout en conservant ce principe, introduire une caractéristique dont l'utilisation sera spécifique à ce type de contrôle telle que désignation par mots-clés de l'élément.

β_2 : *les contrôles globaux sur la cohérence du nouveau schéma conceptuel*

Pour répondre à la question "le nouvel état du SI est-il un état cohérent"? on peut envisager des moyens du même type que ceux proposés en α_1 ; c'est à dire par le biais de la vérification de la conformité de la structuration de l'ensemble.

Compte tenu de la structuration existante sur les éléments types du modèle conceptuel, il faut vérifier, d'une part, que les occurrences de ces éléments respectent les règles de structuration, d'autre part, que les valeurs des caractéristiqu des éléments ne sont pas incompatibles avec ces règles. En particulier certaines caractéristiques temporelles doivent permettre de vérifier la cohérence dans le temps des différents éléments du S.C. (Exemple : la structure en modules d'une procédure dans Remora doit respecter les relations temporelles entre les modules qui s'expriment à travers les lots d'informations transitant d'un module à un autre).

β_3 : *Les contrôles de fidélité*

Ils sont de même nature que ceux que nous avons proposé en α_2. Cependant, il nous semble que de tels contrôles ne sont réalistes que si l'on dispose dans le schéma d'images de classes de phénomènes de niveau supérieur à ceux que l'on fournit par un simple énoncé. (Par exemple dans Remora il faut disposer d'une procédure entièrement définie pour exécuter ce type de contrôle).

β_4 : *Les contrôles de complétude*

Ayant pour objectif de s'assurer que "l'état du SI est une représentation complète de l'ensemble des phénomènes pris en compte", ils n'ont de sens, comme précédemment que lorsqu'on dispose dans le SC de la représentation d'une classe de phénomènes réels que le concepteur a décrite intégralement.

Ces contrôles s'appuyent sur des *règles de complétude* qu'il faut préciser dans chaque cas particulier. (exemple de règles applicables dans Remora, au niveau d'une procédure : tous les lots et les modèles sont décrits - tous les résultats sont définis.....).

γ : *La fonction DOCUMENTATION*

Elle recouvre l'ensemble des actions qui permettent de renseigner le concepteur et l'automate, à tout instant, sur l'état du SI qu'ils conçoivent.

Pour mettre en oeuvre cette fonction, il faut :

1 - définir un stock documentaire pertinent

2 - disposer de moyens permettant de créer et de mettre à jour ce stock

3 - disposer de moyens d'interrogation de ce stock.

Dans notre hypothèse de travail le stock documentaire est le schéma conceptuel.

Notre choix de gestion de ce stock au moyen d'une base de données SOCRATE implique la réponse au troisième point : le langage de requête Socrate (complété par une bibliothèque de macros) est le moyen d'interrogation à des fins documentaires.

Les outils de construction du schéma conceptuel que nous avons cités précédemment sont notre solution au deuxième point.

Les outils permettant d'assurer ces fonctions d'aide à la conception sont partiellement réalisés à l'heure actuelle sur le prototype REMORA. Ce sont les *analyseurs, le gestionnaire de base, le documenteur et l'éditeur.*

III - PRESENTATION DE NOTRE ETUDE

A contrario des autres études menées parallèlement, dans le projet, notre travail correspond à une attaque verticale des problèmes liés aux *traitements* à travers le processus conception - réalisation des systèmes d'informations.

Il comporte deux parties :

La première (chapitre 1) est consacrée à la définition du *langage de conception* de REMORA et tente, à travers une typologie des langages intervenant à un niveau ou à un autre, pour la conception, la réalisation ou l'utilisation des SI, de le situer dans la classe des langages fonctionnels.

La seconde est consacrée à *la définition des structures et expressions des traitements aux trois niveaux* conceptuel - logique - physique d'un processus de conception réalisation des SI.

Le chapitre 2 s'attache à montrer ce que sont les structures conceptuelle - logique et physique des traitements et à étudier leur correspondance ; de manière symétrique à montrer ce que sont les textes de formulation des traitements aux différents niveaux et leur correspondance.

Le chapitre 3 présente comment, dans le cadre de Remora, nous avons pu définir des *algorithmes* de conversion des structures et des textes qui assurent le passage de l'énoncé d'un problème dans le langage de conception aux programmes correspondants. Ces algorithmes de génération sont exécutés par un outil, le *générateur* dont nous présentons l'organisation et le fonctionnement sur un exemple.

PREMIERE PARTIE :

LE LANGAGE DE CONCEPTION DE

REMORA

INTRODUCTION

La vie d'une organisation peut être interprétée comme une collection d'opérations de nature industrielle, commerciale, financière et administrative (acquisition, réception, contrôle, transport, stockage, transformation, emballage, expédition etc...). Ces opérations simultanées ou successives concernent des objets économiques (produits, documents, équipements, etc...) plus ou moins indépendants les uns des autres.

Quand on se propose de développer une application en informatique de gestion, on tente de décrire certaines des opérations concernant quelques uns de ces objets économiques.

Nous nous intéresserons dans ce chapitre à l'étude des langages permettant la description d'une application en informatique de gestion ou, ce qui est équivalent, permettant la formulation d'un problème de gestion en vue de son traitement automatisé. L'étude de ces langages nous permettra de définir les spécifications du langage de description conceptuelle d'un problème de gestion que nous avons retenu dans le projet REMORA.

I - 1 LES CARACTERISTIQUES DES LANGAGES DE FORMULATION DES PROBLEMES

Si on analyse l'expression définitive de la formulation d'un problème, celle qui sera exécutée par l'automate, on peut distinguer deux catégories d'éléments décrits :

- la catégorie des éléments qui se réfèrent uniquement au problème analysé et dans laquelle on va retrouver la définition des objets de gestion et la définition des opérations réalisées sur ces objets.

- la catégorie des éléments qui se réfèrent uniquement à la solution technique retenue et aux contraintes imposées par le recours à un automate particulier. Dans cette catégorie d'éléments on trouve la définition des objets informatiques qui servent à représenter les objets de gestion et la définition des opérations informatiques réalisées sur les objets informatiques pour que l'automate puisse exécuter les opérations de gestion qu'on lui commande de réaliser.

L'étude diachronique de langages de définition des problèmes fait apparaître une régulière évolution de ces langages, qui se traduit, dans un premier temps, par la séparation de plus en plus nette de la description des éléments spécifiques du problème des éléments spécifiques de la solution technique retenue et qui aboutit, dans un second temps, à la proposition de langages de définition des problèmes qui éliminent totalement la prise en compte des facteurs technologiques.

Par l'étude des trois classes de langages qui ont été successivement proposées, nous allons illustrer cette évolution. Chaque classe sera analysée de manière identique : après l'étude du modèle qui sert implicitement de fondement au langage nous présentons les caractéristiques de ce dernier.

I.1.1 Les langages de programmation de haut niveau

Les langages de programmation ont été les premiers langages d'expression formelle des problèmes ou plus exactement des algorithmes de résolution des problèmes. Nous assimilons les langages de programmation à la classe des langages d'expression des problèmes puisqu'il n'existe aucune formulation intermédiaire du problème entre son expression

par le gestionnaire dans un dossier de l'existant et son énoncé dans un langage de programmation de haut niveau tel que Cobol ou PL/1.

α. Le modèle associé :

Le point commun à tous les langages de programmation est de s'appuyer sur un modèle de la formulation du problème que l'on qualifie aujourd'hui de physique ou d'interne, qui donne une place importante a la prise en compte des contraintes technologiques de l'automate.

Les concepts de ce modèle sont les objets informatiques du niveau de l'implémentation physique d'un problème, de l'une des deux classes :

- données
- opérations sur les données.

Les domaines de variation des valeurs des données définies technologiquement déterminent leur type (entier, réel, booléen). A ces types élémentaires ont été adjoints des types composés : les structures de données sont le plus souvent limitées aux tableaux.

Les opérations sur les données sont les opérations de manipulation des données complétées par des opérations arithmétiques et des opérations logiques.

β. Les caractéristiques de classe :

Nous qualifions les langages de programmation par les trois points suivants :

β1) ces langages prévoient que l'on définisse dans le même programme les données et opérations appliquées à ces données à partir des types internes.

Cette exigence a une double conséquence :

* La dépendance des données et des traitements , les structures de données n'étant définies que pour satisfaire les besoins de calcul.

* La subordination de la formulation de l'univers d'un problème :
  - à la nature des objets internes manipulés dans le langage
  - à la représentation technologique de l'information.

De tels langages obligent le programmeur à représenter l'univers par une structure de données imposée en grande partie par la technologie. De plus l'effort doit être refait à chaque programme même si le problème lui-même n'a pas changé.

β2) Les langages de programmation exigent l'expression du problème sous forme algorithmique

Nous appelons algorithme la séquence logique et chronologique des opérations à exécuter pour résoudre un problème. Cette séquence est fixée dans tous ses détails par des opérations qui n'ont aucun sens considérées isolément et ne sont significatives que par rapport au problème lui-même et à la solution technologique qui a été retenue pour le résoudre. L'algorithme doit en particulier spécifier de manière non ambigüe l'ordre d'exécution des opérations, ou ce qui est équivalent l'ordre d'application des régles.

Les langages qui nécessitent une telle définition des problèmes sont dits "impératifs" (13). Sans faire appel à une aussi profonde connaissance de la technologie et du fonctionnement interne de l'ordinateur que les langages "machine" ou les langages d'assemblage, les langages de programmation peuvent être qualifiés d'impératifs : ils exigent qu'un problème de traitement soit formulé sous la forme d'une suite impérative d'instructions (≡ règles) respectant des contraintes syntaxiques dont l'apprentissage n'est pas toujours aisé.

β3) Les fonctions d'accès aux données physiques doivent être intégralement explicitées dans la formulation de l'algorithme.

Ces langages de programmation sont des langages de navigation (4), (11). C'est au programmeur, compte tenu de ses besoins d'accès, d'exprimer les compositions d'accès nécessaires pour naviguer dans l'espace des données. De tels langages sont dits procéduraux.

γ. Conclusion :

Les langages de programmation sont algorithmiques et procéduraux. Parce qu'ils opèrent sur des modèles internes de représentation des données et des traitements, ils permettent d'automatiser les problèmes dont ils donnent une représentation exécutable par l'ordinateur mais totalement incompréhensible pour la personne qui les a posés.

Le fossé entre le modèle de perception du problème par les gestionnaires et le modèle de l'expression algorithmique et procédurale de sa solution est si grand que toute vérification de l'adéquation de la solution au problème est impossible pendant l'élaboration de la solution, difficile et coûteuse après.

De même, toute correspondance entre une modification perçue dans le premier modèle et sa répercussion dans la solution élaborée dans le second est mal aisée à effectuer et à valider.

On peut considérer que les langages d'analyse ont tenté de combler ce fossé en proposant une expression intermédiaire entre l'énoncé et la solution technologique.

I.1.2 Les langages d'analyse

L'apparition des langages d'"analyse" se justifie par les deux objectifs qu'on peut qualifier d'invariants dans le large éventail de ces langages :

1 - Trouver un interface entre le problème réel et sa solution technologique en "remontant" à partir du niveau de la réalisation qu'est la programmation.

2 - Rechercher un passage automatique de l'interface à l'algorithme.

Interrogeons nous sur les moyens proposés pour satisfaire ces objectifs. Partant des caractéristiques des langages de programmation, précédemment énoncées, nous avons perçu trois types d'évolutions possibles portant :

1 - sur le modèle d'accès aux données dans les traitements

2 - sur le modèle d'expression des traitements

3 - sur le modèle de structuration des données.

Ces évolutions ne sont pas indépendantes mais complémentaires et peuvent avoir été intégrées simultanément dans certains langages de cette classe.

α. Les points d'évolution par rapport aux langages de programmation

α1. Atténuation de la procéduralité

Elle apparaît à travers deux aspects complémentaires qui sont

. l'existence de schéma logique de programme

. le répérage logique des fonctions technologiques.

Bien des auteurs ont constaté que les traitements de gestion, surtout lorsqu'ils s'appliquent à la gestion administrative, sont des traitements répétitifs encore appelés par lots : une même séquence d'actions est exécutée sur des lots successifs de données correspondant aux occurrences successives des propriétés d'un même objet. Cette constatation les a conduits à proposer un schéma de programme standard, sous la forme d'une imbrication hiérarchique de traitements itératifs que REIX (33) a appelé en pelure

d'oignons (figure 1), que les méthodes d'analyse ont utilisé MINOS (26), ATLAS (3), CORIG (47) ... et qu'un projet de recherche tel que CIVA (13) a suggeré par l'arborescence des "pour chaque" (figure 2).

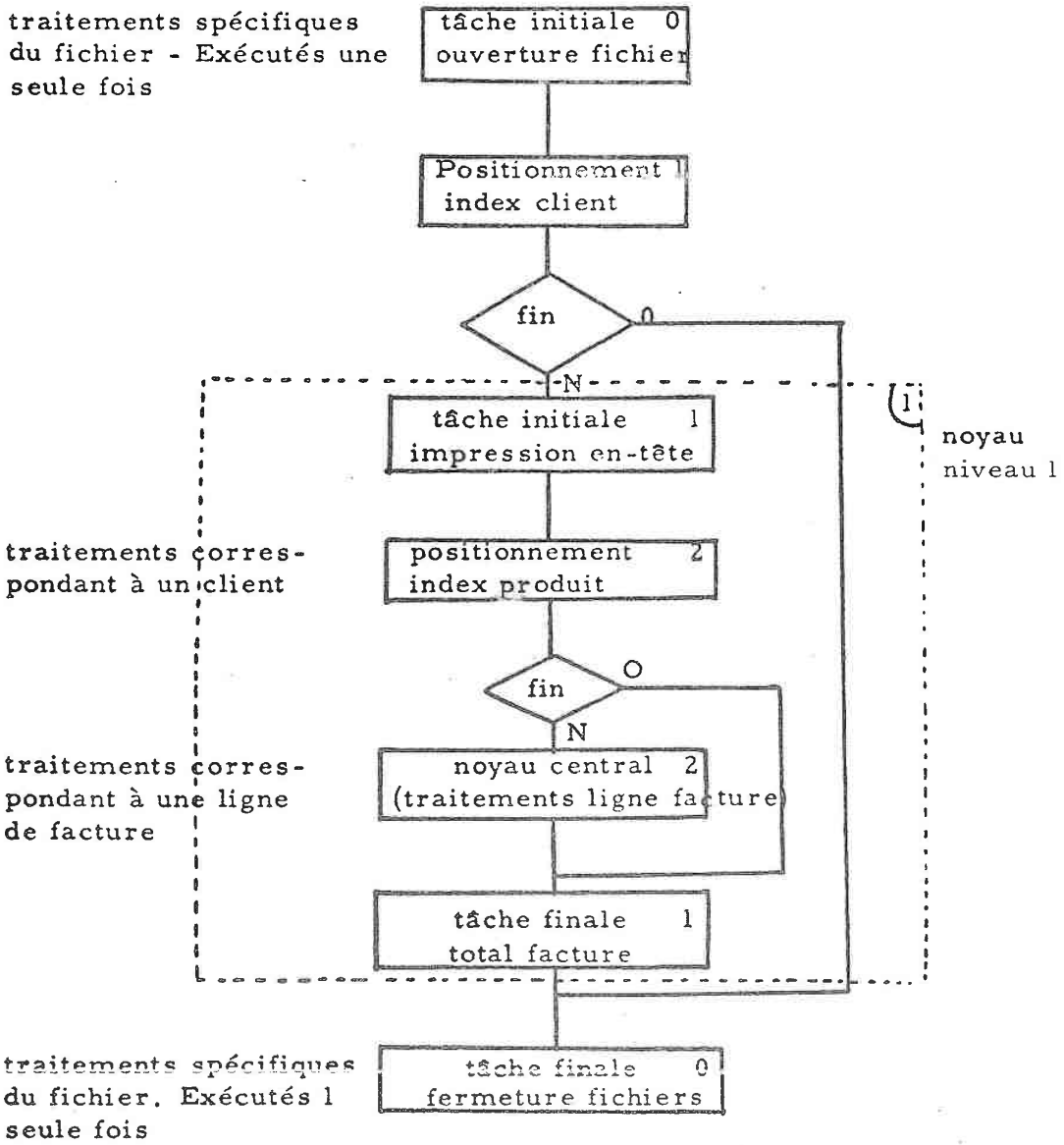


Figure 1 : Le schéma de programme en pelure d'oignons de REIX - 33 - tome 1 p. 1971

```
total général = 0 ;
POUR CHAQUE agent FAIRE
        total agent = 0 ;
        POUR CHAQUE client FAIRE
            total agent = total agent + prime fpc ;
            éditer (total agent) ;
            total général = total général + total agent fpc ;
éditer (total général) ;
```

Figure 2 : Expression de traitements itératifs dans CIVA 13 - paragraphe 8.3.4 - 1974

La programmation structurée, par l'apport de concepts supplémentaires, a permis aujourd'hui d'affiner ce schéma. Nous en verrons l'utilisation dans la deuxième partie.

Parallèlement les auteurs ont remarqué qu'il était possible de dissocier dans un programme l'expression de l'algorithme de celle des fonctions technologiques assurant la gestion des flux d'informations. Déjà en 1966, MALLET dans CORIG (8) faisait cette constatation en dissociant la recherche de la liste des taches (traitements reflétant le problème) de celle des fonctions (traitements exécutés par le programme). L'apport supplémentaire à celui de MALLET a consisté à dire qu'il était également envisageable de repérer ces fonctions technologiques au niveau logique par un nombre limité de paramètres cités dans le schéma de programme itératif.

Ils ont proposé de désigner ces fonctions par des opérateurs qui en sont les repères logiques. Dans le même sens ces auteurs ont proposé d'automatiser la traduction de l'opérateur en une séquence d'instructions d'accès aux informations compte tenu de leur mode d'implémentation.

On peut considérer que les langages d'analyse ont restreint l'aspect procédural des langages de programmation

. en autorisant le programmeur à ne citer que des opérateurs
. en proposant des générateurs qui traduisent en instructions ces opérateurs.

Les premiers générateurs de cinématique de fichiers sont apparus en complément des méthodes d'analyse comme CORIG, ARMIN etc, . . .

HERTSCHUH (18) a clairement montré quelles étaient les fonctions d'un tel générateur et nous verrons dans la deuxième partie comment nous avons utilisé son travail.

Remarquons que la voie des opérateurs ouverte sur les problèmes de cinématique a tenté de s'élargir, comme nous le suggère CATTAN dans son étude sur la percée des langages d'analyse (46), mais sans succés frappant nous semble-t-il.

$\alpha 2$ - Vers une expression moins algorithmique des traitements

Certains promoteurs de méthode d'analyse, s'appuyant sur le schéma de programme itératif, ont tenté d'intégrer ce schéma dans un langage qui en permettrait l'expression et permettrait aussi l'expression des algorithmes de traitement sous une forme syntaxiquement moins contraignante que celle des langages de programmation. C'est d'ailleurs à cette occasion qu'est né le terme de langage d'analyse dont un exemple typique (figure 3) peut être pris dans la méthode PROTEE (27) : reprenant les caractéristiques de CORIG cette méthode propose la description de l'algorithme dans un tableau donnant à gauche la liste des actions à exécuter, à droite les conditions sous lesquelles elles le sont.

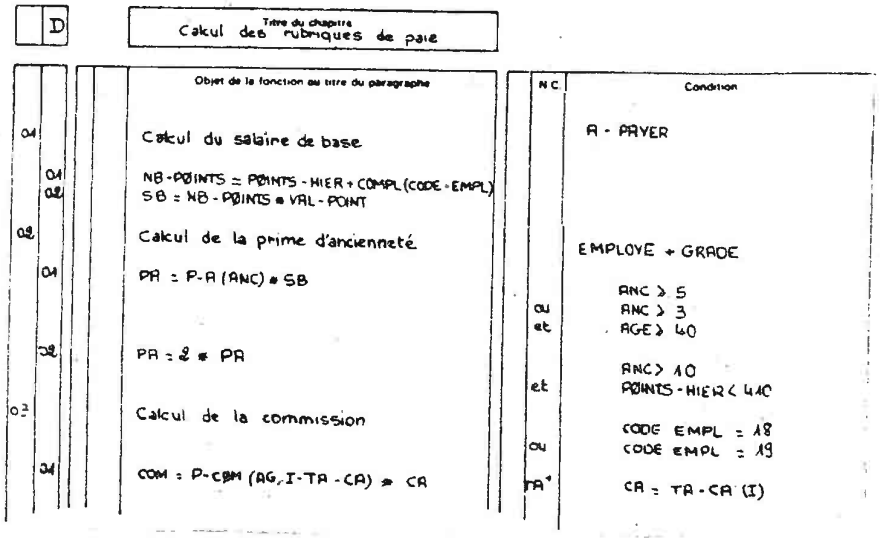


Figure 3 : Expression des traitements dans le langage PROTEE - Extrait de documents sur la méthode -

On peut se demander, malgré l'affirmation des auteurs et leur volonté de le placer à un niveau plus élevé, si le langage PSL d'ISDOS n'est pas caractéristique de cette seule évolution (figure 4). Il ajoute

α. la liberté de décrire des "blocs" du texte de manière non colonnée mais les types de bloc pour lequel on accepte ceci sont limités et repérés par des mots clés.

β. La possibilité de formuler l'algorithme en deux étapes : la première peut être assimilée à la description de l'architecture du programme ; la seconde assurant la description détaillée des blocs cités dans l'architecture.

Niveau 1 :

```
PROCESS TRAITEMENT-DU-STOCK ;

PROCEDURE :
IF BON-D-ENTREE-EN-STOCK THEN TRAITEMENT-BON-D-ENTREE-EN-STOCK
ELSE IF BON-DE-COMMANDE-CLIENT THEN TRAITEMENT-BON-DE-COMMANDE-CLIENT
ELSE IF FIN-DE-FICHIER THEN TRAITEMENT-FIN-DE-FICHIER.

RECEIVES TRANSACTION-STOCK ;

SUBPARTS ARE TRAITEMENT-BON-D-ENTREE-EN-STOCK, TRAITEMENT-
        BON-DE-COMMANDE-CLIENT, TRAITEMENT-FIN-DE-FICHIER.


INPUT TRANSACTION-STOCK ;

CONSISTS OF CODE-TRANSACTION, QUANTITE ;
RECEIVED BY TRAITEMENT-DU-STOCK ;

ELEMENT CODE-TRANSACTION ;

VALUE 0  THRU 9 .

ELEMENT QUANTITE ;

SYNONYMS ARE NOMBRE-COMMANDE, NOMBRE-RECU ;
VALUE 0  THRU 9999.
```

Niveau 2 (partiellement)

```
PROCESS TRAITEMENT-BON-DE-COMMANDE-CLIENT ;

USES NOMBRE-COMMANDE TO UPDATE CUMUL-NOMBRE-D-EXEMPLAIRES-COMMANDES ;
USES NOMBRE-A-EMBARQUER TO UPDATE NOMBRE-D-EXEMPLAIRES-DANS-LE-STOCK ;
UPDATES NOMBRE-DE-BONS-DE-COMMANDE-TRAITES ;
SUBPARTS ARE TRAITEMENT-D-INFORMATION-D-ENVOI, TRAITEMENT-DE-
REAPPROVISIONNEMENT.

PROCEDURE :

ADD NOMBRE-COMMANDE TO CUMUL-NOMBRE-D-EXEMPLAIRES-COMMANDES ;
ADD  1  TO NOMBRE-DE-BONS-DE-COMMANDE-TRAITES ;
IF NOMBRE-D-EXEMPLAIRES-DANS-LE-STOCK ≥ NOMBRE-COMMANDE
   THEN NOMBRE-A-EMBARQUER = NOMBRE-COMMANDE
   ELSE BLOCK

   NOMBRE-A-EMBARQUER = NOMBRE-D-EXEMPLAIRES-DANS-LE-STOCK ;
   NOMBRE-D-EXEMPLAIRES-EN-REAPPROVISIONNEMENT = NOMBRE-D-EXEMPLAIRES-
   EN-REAPPROVISIONNEMENT + (NOMBRE-COMMANDE — NOMBRE-D-EXEMPLAIRES-
   DANS-LE-STOCK) ;
   TRAITEMENT-DE-REAPPROVISIONNEMENT
   END BLOCK ;
```

Figure 4 : Description des traitements dans le PSL d'ISDOS

extrait de 21.

De tels langages sont évidemment compilables ou plus exactement générables afin de faciliter l'insertion des textes traduits par le générateur dans une chaîne de programmes écrite par ailleurs manuellement, dans un langage de programmation évolué tel que COBOL ou PL/1.

Les langages d'analyse sont de "super" langages de programmation autorisant une formulation plus aisée de l'algorithme. Cette formulation est de plus simplifiée si ces langages incluent des opérateurs tels que ceux que nous avons développés dans le point 1.

α3 - Vers des structures logiques de données

La recherche des paramètres concernant les données qu'il faut citer, pour décrire la logique des traitements, a poussé les auteurs à proposer à l'analyste des visions simplifiées des structures de données physiques qu'il doit normalement prendre en compte dans son raisonnement.

Ainsi a été introduite la manipulation de files d'éléments dans CIVA (13), des SET et ENTITY (avec une signification voisine de celle qui a été fournie dans le rapport CODASYL (9)) dans ISDOS (figure 5).

```
 .....
SET INFORMATIONS-RELATIVES-AU-STOCK ;
  CONSISTS OF DESCRIPTION-DU-STOCK, ETAT-DU-STOCK...

ENTITY ETAT-DU-JOUR
   CONSIST OF NOMBRE-D-EXEMPLAIRES-DANS-LE-STOCK, NOMBRE-D-EXEMPLAIRES-
   EN-REAPPROVISIONNEMENT
 ....
PROCESS TRAITEMENT-DE-REAPPROVISIONNEMENT
   GENERATES ORDRE-DE-REAPPROVISIONNEMENT ;
```

Figure 5 : Description de SET et ENTITY dans le PSL d'ISDOS - Extrait de 21.

En proposant de telles structures on tente de se dégager des contraintes imposées par la technologie. Mais pour autant, il ne s'agit pas de ce qu'aujourd'hui, avec l'apport des Bases de données, on a pu définir comme structure logique de données (44).

β. Conclusion

Les langages d'analyse correspondent à une tentative de définition du problème sous la forme d'une expression logique des traitements comme des données.

Cette tentative est dans tous les cas incompléte ou insuffisante parce que les problèmes n'ont pas été posés dans ces termes. Les deux caractéristiques que nous en retenons sont :

- la possibilité qu'ils offrent de formuler l'algorithme des traitements de manière plus aisée que dans les langages de programmation et quelquefois simplifiée par l'usage d'opérateurs.
- la suppression d'une partie du travail de programmation par l'automatisation de fonctions technologiques repérées par les opérateurs.

I.1.3 Les langages fonctionnels

Les langages fonctionnels aussi appelés langages de haut ou très haut niveau ont une optique résolument différente. Leur objectif est de se limiter à l'expression logique des problèmes ; ils visent une représentation formelle des phénomènes du monde réel en ignorant les aspects techniques qui permettront de transformer cette représentation en expression physique de la solution. Ils répondent au désir de modéliser la réalité, en se limitant à l'expression de la sémantique du réel perçu indépendamment de toute réflexion liée à l'implémentation du problème.

α. Les modèles

Tous les langages de cette nature s'appuyent sur des modèles abstraits dont un des exemples les plus connus et les plus exploités à l'heure actuelle est celui des bases de données.

Les modèles du type relationnel tel que celui de CODD (45) permettent de représenter les objets de la réalité au moyen du concept de relation (au sens mathématique du terme) et conduisent à décrire les phénomènes réels par des tableaux de valeurs (6_b). Ils correspondent à un modèle de perception de la réalité qu'on peut assimiler, malgré la diversité des approches, à celui que B.B.B.C (6) ont présenté à la conférence de l'IFIP TC2 à Freudenstadt : les objets - leurs propriétés - leurs associations.

Les langages de définition des données relationnels proposent une description conceptuelle des problèmes par une formulation mathématique à syntaxe simple. Ils permettent la description des relations et de leurs propriétés au moyen de prédicats (45). On peut seulement regretter que de tels modèles soient restrictifs puisqu'ils se limitent à la représentation des objets et ignorent celle des opérations.

Les langages de définition des données assurent une description du seul aspect statique des problèmes de gestion.

Les auteurs qui s'intéressent à l'utilisation (18 b) des systèmes d'information se sont posés, dans des termes voisins, le problème de la description logique d'un acte de gestion qui a pour but de produire des informations à partir d'un ensemble de transformations opérées sur un ensemble de données existant.

Le principe est double :

. tout problème peut être défini sur un "univers" au sens de PAIR (29) dont on connait la structure et les caractéristiques des éléments qui la composent.
Pour spécifier un nouveau problème il suffit donc d'en donner un énoncé en fonction des éléments connus de la structure d'univers c'est-à-dire, de la structure des données de l'organisation.

. pour décrire les transformations il faut disposer de concepts spécifiques qui doivent être connus des utilisateurs de ces langages et pour cette raison doivent être élémentaires. PAIR et GRANDMOUJIN (16) basent par exemple cette approche sur les notions de fonctions d'ensembles et sur les règles de mathématiques élémentaires.

β. Les caractéristiques

β1. Les langages fonctionnels s'appuyent sur des modèles de représentation formelle de la réalité.

Ils correspondent à la description des concepts retenus pour représenter une partie plus ou moins complète de la réalité. Nous dirons que cette description pour un problème donné est un énoncé : l'énoncé du problème, l'expression de sa sémantique ou encore sa définition.

β2. Ils permettent d'établir un consensus entre utilisateurs (futurs) et concepteurs du SI

Les langages fonctionnels ne sont pas obligatoirement des langages mis à la disposition des gestionnaires. Leur forme est, dans bien des cas encore, trop hermétique (figure 6) et la manipulation des concepts sur lesquels ils reposent trop délicate pour que des non spécialistes puissent les utiliser sans ambiguité.

Exemple de description de la structure conceptuelle

STRUCTURE CONCEPTUELLE ENTREPRISE

DONNEES :

ND "Numéro de département"
FORMAT FORM-1 DECIMAL 1
CI.1 VALUE 0,9

DB "Budget du département"
FORMAT FORM-2 DECIMAL 6
CI.2 VALUE 0, 999999

MATD "Matricule du Manager du département"
FORMAT FORM-3 DECIMAL 3

NE "Numéro d'employé"
FORMAT FORM-3

NP "Numéro de projet"
FORMAT FORM-4 DECIMAL 2
CI.3 CARDINAL 50

NT "Numéro de Téléphone"
FORMAT FORM-4
CI.4 = CI.3

DATE "Date de début de Jobs"
FORMAT FORM-5 CARACTERES 6

NJ "Numéro de Job"
FORMAT FORM-1

SALAIRE "Salaire mensuel de l'employé"
FORMAT FORM-6 DECIMAL 5

PB "Budget du projet"
FORMAT FORM-2

SURFACE "Superficie du bureau en m^2"

FORMAT FORM-4

CI.5 : VALUE 9,35

RELATIONS

D (ND, DB, MATD) "Département"

IC.6 : CLE (ND)

IC.12 : REL. FONCT (ND) (DB)

E (NE, NP, NT) "Employé"

IC.7 : CLE (NE)

IC.13 : REL. FONCT (NE) (NP)

IC.14 : REL. FONCT (NE) (NT)

H (DE, DATE, NJ, SALAIRE) "Historique des jobs de l'employé"

IC. 8 : CLE (NE, DATE)

P (NP, ND, PB) "Projet"

IC.9 : CLE (NP)

B (NB, ND, SURFACE) "Bureau"

IC.10 : CLE (NB)

T (NT, NB) "Téléphone"

IC.11 : CLE (NT)

Figure 6 : Définition conceptuelle des données

Extrait de 6b.

Les langages de conception ont eux vocation à être utilisés directement par des utilisateurs non spécialistes et doivent nécessairement tenir compte de leur profil.

Cependant, tous les langages de ce niveau doivent avoir pour objectif d'être compréhensibles par les non spécialistes. Même si les gestionnaires n'ont pas eux-mêmes élaboré les expressions conceptuelles des problèmes, ils doivent pouvoir les comprendre. Aussi, un consensus sur la solution peut s'établir dès ce stade entre les concepteurs et les futurs utilisateurs du système informatique en construction.

Il est ainsi possible de vérifier plus tôt et plus facilement l'adéquation de la solution au problème posé.

β3. La forme du passage à la solution technologique n'est pas unique

Il est évident que l'expression conceptuelle ainsi obtenue doit faciliter le passage à la solution technologique qui sera effectivement implémentée. Une solution séduisante consiste à imaginer un automate qui, moyennant l'introduction de paramètres logiques et technologiques, assure ce passage automatiquement. Une telle solution est envisageable en gestion parce qu'on souhaite, le plus souvent, définir explicitement les problèmes : nous en verrons une application dans Remora au paragraphe suivant.

Ce n'est cependant pas toujours le cas et l'acceptation de définitions implicites telles que les théoriciens de la programmation les prennent en compte (par exemple :"trouver i tel que t (i) = a") conduit si l'on veut aboutir automatiquement à un algorithme à faire la "synthèse des programmes" (14), ce qui revient à résoudre des problèmes du domaine de l'intelligence artificielle.

I.2 LE LANGAGE DE CONCEPTION DE REMORA

Le langage de conception du projet REMORA appartient à la classe des langages fonctionnels. Il s'appuie sur une modélisation du monde réel qui a été définie dans la thèse de THIERY (37) et s'inscrit dans une approche méthodique des problèmes de gestion présentée dans la thèse de PEREA-VILLACORTA (31).

I.2.1 Un langage de formulation de l'énoncé d'un problème

Le langage de conception est pas hypothèse un langage d' énoncé. Mais que doit être un énoncé en gestion ?

Nous pensons que les problèmes de gestion sont naturellement perçus par les gestionnaires externes de besoins et que, d'autre part, les besoins sont exprimés par des résultats ; le terme résultat doit être ici pris au sens large : les résultat sont les informations (les mots dans la terminologie de REMORA) produites par l'exécution d'actions déclenchées par un événement.

Enoncer un problème de gestion c'est exprimer les besoins ressentis par les gestionnaires à travers le modèle de perception du monde réel que nous avons choisi et présenté dans (37) ; c'est définir les résultats qui matérialisent ces besoins.

Le langage d'énoncé est donc un langage de définitions.

I.2.2 Un langage de definitions

L'étude que nous avons faite dans (31) nous a permis de définir un résultat (un mot) par son type - sa nature - sa décomposition.

α. Le type des mots

Un mot est inconditionnel : s'il apparaît toujours dans un message ou la décomposition d'un autre résultat.

Un mot est conditionnel : s'il apparaît sur un message ou la décomposition d'un résultat seulement lorsqu'une condition est vérifiée.

Un mot est listes de résultats : s'il s'agit d'un ensemble de résultats de même structure dont chaque terme est défini indépendamment des précédents et obtenu par une même séquence de définitions.

β. La nature d'un mot

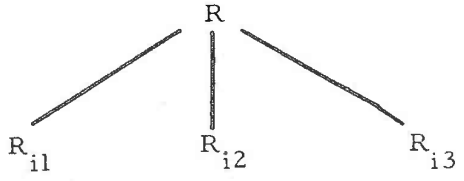
Elle peut être :

- donnée : si le contenu est utilisé dans la procédure sans y être ni modifié, ni évalué
- résultat : si le contenu est évalué dans la procédure et appartient à au moins un lot
- résultat intermédiaire : si le contenu est évalué dans la procédure mais n'appartient à aucun lot.

γ. La décomposition d'un résultat

Lorsqu'on cherche à définir un résultat on est conduit à introduire des intermédiaires qui à leur tour pourront être définis.

Nous partons de l'hypothèse que la décomposition d'un résultat en résultats intermédiaires peut se représenter par une arborescence du type :



chacun des résultats intermédiaires pouvant à son tour être décomposés suivant un schéma identique.

W. PEREA VILLACORTA (31) a montré comment on est alors amené à définir un résultat par un graphe de résultats, dont les feuilles sont éventuellement des mots de nature donnée.

Le langage doit permettre la définition des mots de nature résultat : il doit donc prévoir la possibilité de définir leur décomposition en mots et de préciser le type et la nature des mots ainsi introduits.

I.2.3 Un langage de définitions explicites

En gestion les traitements qui traduisent la décomposition d'un résultat R en résultats intermédiaires R_1, R_2, ... R_n sont des formules de calcul du type fonction arithmétique $f(R_1, R_2, \ldots, R_n)$.

Ces formules font apparaître explicitement les résultats intermédiaires qui entrent dans la décomposition d'un résultat.

Nous dirons pour cette raison que les définitions sont explicites puisqu'elles définissent un élément en fonction des éléments qui le définissent.

I.2.4 Un langage modulaire

L'idée de décomposer un problème en modules puisqu'elle remonte au moins à Descartes n'est pas nouvelle, tant au niveau de l'analyse du problème, qu'au niveau de sa description (13), (27), (12).

La méthode d'analyse que nous avons proposée dans REMORA est modulaire (31) puisqu'elle autorise qu'on reporte à une étape ultérieure la définition d'un mot qu'il est difficile ou impossible de définir au moment où on le rencontre.

Le langage de conception est aussi modulaire : le module est l'expression de ce langage de la définition d'un résultat.

I.2.5 Un langage exploitable par l'ordinateur

Le caractère explicite des définitions rend plus aisé le passage de l'énoncé à l'algorithme.

Nous montrerons dans la deuxième partie comment, à partir de l'expression conceptuelle des problèmes, un outil (le générateur) peut assurer le passage aux programmes correspondants à condition de prendre en compte, au niveau logique et au niveau physique, les choix associés à ces niveaux : choix d'exécution unitaire des traitements, choix des structures physiques de données, choix du langage de programmation...

I.3 LA SYNTAXE DU LANGAGE DE CONCEPTION

Nous nous proposons dans ce paragraphe de reprendre les caractéristiques du langage de conception pour en examiner la syntaxe que nous présentons dans une forme normalisée, à l'aide des conventions de Backus, dans le paragraphe suivant.

Pour éclairer notre exposé, nous nous appuierons sur un exemple d'école volontairement simplifié.

I.3.1 L'exemple

Supposons que dans le cadre d'une procédure FACTURATION, on veuille définir le résultat TOTFAC soit le total à payer de la facture.

L'analyse de ce résultat suivant la méthode que nous avons développée dans la thèse de PEREA-VILLACORTA (31) conduit à construire le graphe des résultats suivants :

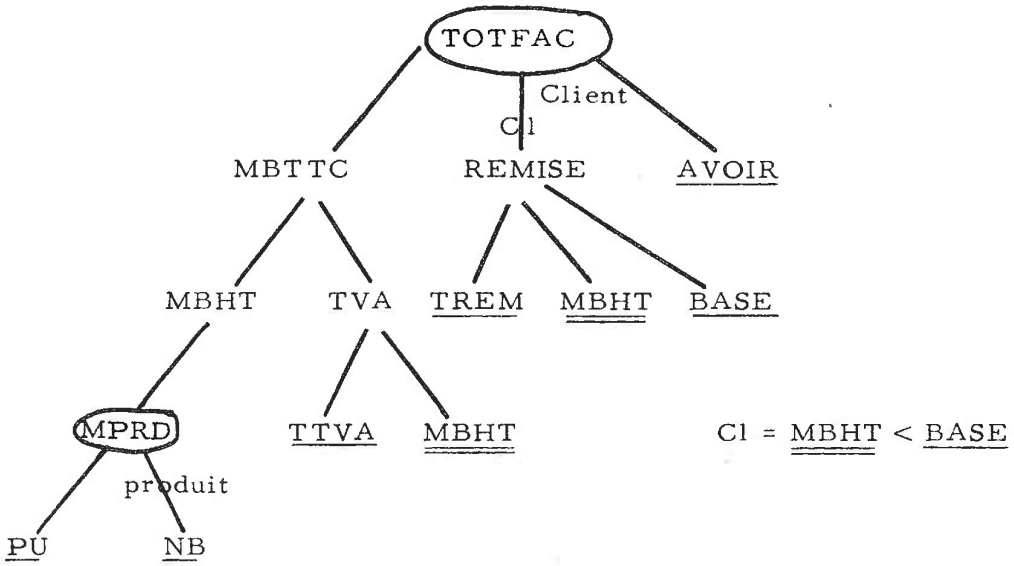


Figure 7

où TOTFAC et MPRD sont des listes de résultats associées aux entités CLIENT et PRODUIT

REMISE est un résultat intermédiaire conditionnel

MBTTC, TVA, MBHT, sont des résultats intermédiaires inconditionnels

PU, NB, TTVA, TREM, BASE et AVOIR sont des données (pour le module FACTURATION).

I.3.2 Le support du langage de conception

Le support du langage de conception est un support papier, appelé descripteur de traitement. : c'est un document structuré en forme de tableau à deux entrées (figure 8), dont le format est figé. Les mots clés du langage y sont exprimés par le biais de leur position typographique sur une ligne du descripteur.

La définition d'un résultat comporte plusieurs composantes visant à préciser sa désignation - son type - sa nature et sa décomposition en résultats.

Chacune des composantes est précisée sur une ligne de définition du descripteur dans une zone réservée (respectivement IDENTIFICATEUR et DESIGNATION DES ELEMENTS ; ENTITE, CONDITION ou CALCUL ; DONNEE ou APPEL ; CALCUL).

Nous en ferons l'étude à partir de l'analyse des caractéristiques du langage.

I.3.3 La déclarativité du langage

La déclarativité du langage REMORA est liée à l'autorisation de définir les résultats dans l'ordre où ils ont été analysés. L'ordre des définitions sur le descripteur est donc l'ordre de l'analyse des résultats des graphes, tel qu'il a été défini dans (31).

L'arborescence des définitions est l'image de l'arborescence des résultats : elle permet d'exprimer qu'un résultat R est défini à partir de n résultats intermédiaires R_1, R_2, ..., R_n qui peuvent à leur tour être définis par un résultat intermédiaire....

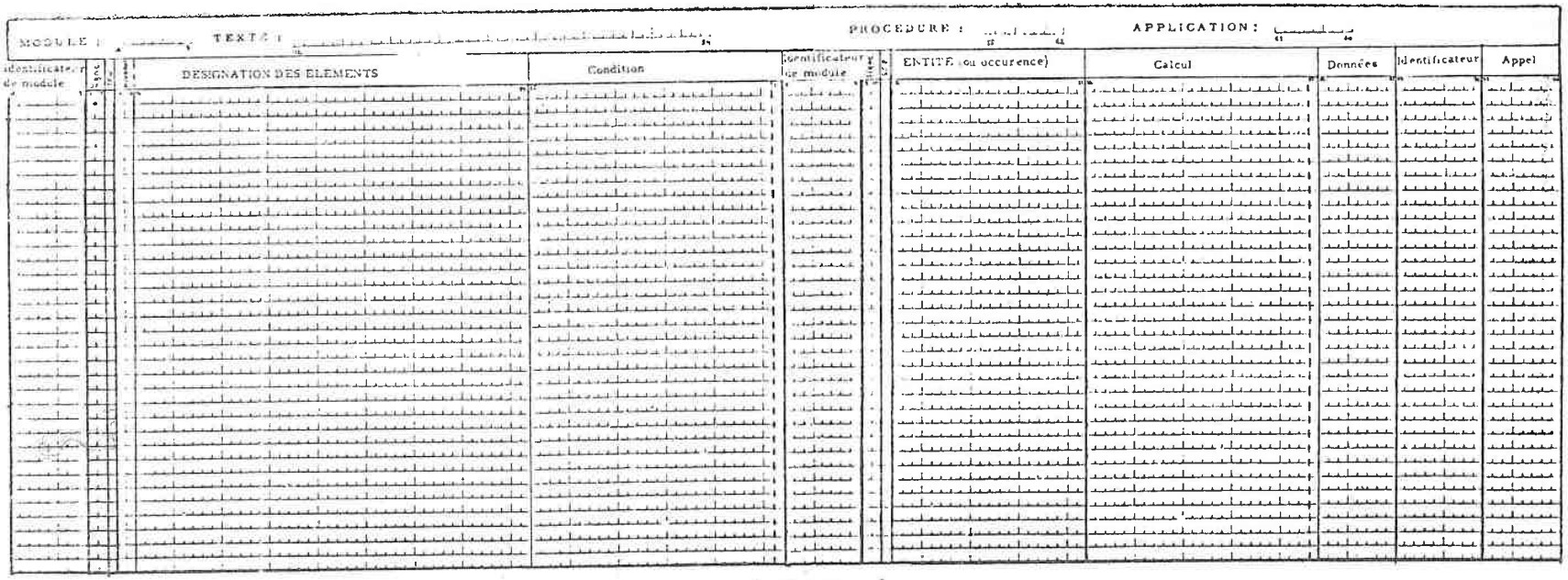
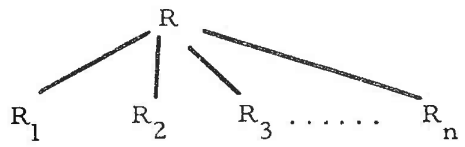

MODULE :
TEXTE :
PROCEDURE :
APPLICATION :
Identificateur de module
DESIGNATION DES ELEMENTS
Condition
Identificateur de module
ENTITE (ou occurence)
Calcul
Données
Identificateur
Appel

Figure 8

Pour une sous-structure telle que



nous disons que R est <u>père</u>, que tout R_i $\forall i \in (1, n)$ est <u>fils</u> ; et que les différents R_i sont <u>frères</u>.

Cette sous-structure est traduite sur le descripteur par le jeu des nombres - niveaux dans l'ordre croissant traduit la relation "fils" de l'arborescence des résultats.

Si nous admettons que pour définir un résultat sur une ligne des descripteurs, il est nécessaire de lui associer :

* <u>Une définition en clair</u> portée dans la colonne DESIGNATION DES ELEMENTS. (Ce texte est archivé comme caractéristique du mot dans la base des objets élémentaires et sont utilisés à des fins de documentation et de contrôle par le système (37)).
* <u>Une désignation</u>, placée dans la colonne IDENTIFICATEUR, qui sert de référence pendant le développement de l'analyse et l'expression des calculs proprement dits.

A l'arbre des résultats de la figure 7 correspond le groupe de lignes des définitions suivantes :

niveau	Désignation des éléments		Identificateurs	
01	Total facture		TOTFAC	
02	Montant brut TTC		MBTTC	
03	Montant brut HT		MBHT	
04	Montant produit HT		MPRD	
05	Prix unitaire		PU	
05	Nombre de produit		NB	
02	Montant Taxes : TVA		TVA	
03	Taux de TVA		TTVA	
02	Remise		REMISE	
03	Taux de remise		TREM	
03	Base minimum d'achat		BASE	
02	Avoir		AVOIR	

Figure 9

A chaque branche de l'arbre des résultats correspond une suite de lignes sur le descripteur ; l'ordre de représentation des branches est leur ordre d'apparition sur l'arbre des résultats quand on l'explore du haut vers le bas est de la gauche vers la droite. Les nombres niveaux croissent au fur et à mesure que l'on s'approche d'une feuille et restent constants quand on se déplace horizontalement sur un même niveau de l'arbre des résultats.

Remarquons qu'à certaines feuilles de l'arbre des résultats ne correspondent pas de lignes de définition sur le descripteur. En effet, implicitement tout mot situé à droite sur l'arbre n'a plus besoin d'être défini s'il a été à gauche dans ce même arbre ou ailleurs dans un autre arbre : il n'est donc pas nécessaire de le citer sur le descripteur. C'est le cas, en particulier, de MBHT, au niveau de la définition de TVA et de REMISE.

I.3.4 Les trois types de définitions

Pour définir plus complétement un résultat sur le descripteur, il est nécessaire de rajouter aux trois éléments précédemment introduits (nombre - niveau - identificateur - désignation en clair) ceux qui vont permettre :

. de traduire en termes de traitements la décomposition d'un résultat père en résultats fils.

. d'exprimer son type.

Les trois types de définition du langage permettent de satisfaire ce besoin.

I.3.4.1 Les définitions consécutives

Elles assurent la formulation de l'évaluation d'un résultat "père" en fonction des résultats "fils" qui le définissent.

Compte tenu de la fréquence d'apparition des sommes algébriques dans les calculs de gestion nous avons prévu deux formes de définitions consécutives : les sommes algèbriques - les expressions arithmétiques.

α) Les sommes algèbriques

Nous avons souhaité en donner une expression simple dans le langage.

Ainsi pour exprimer qu'un "résultat intermédiaire" doit être additionné (ou soustrait) à des frères, il suffit de faire précéder du signe + (ou -) sa désignation dans la colonne IDENTIFICATEUR.

Exemple : La définition de TOTFAC se fait à partir de la somme algèbrique : MBTTC - REM - AVOIR soit sur le descripteur :

nb niveau	Désignation des éléments	Identificateur
01	Total facture	TOTFAC
02	Montant brut TTC	+ MBTTC
02	Remise	- REMISE
02	Avoir	- AVOIR

Figure 10

Remarque 1 : le jeu des nombres-niveaux permet d'exprimer que MBTTC, REMISE et AVOIR sont des résultats intermédiaires frères qui sont les fils de TOTFACT.

Remarque 2 : Le jeu des nombres niveaux permet par ailleurs de définir aisément des cumuls : l'évaluation de TOTFAC s'obtient en soustrayant REMISE et AVOIR de MBTTC.

Remarque 3 : L'un quelconque des résultats intermédiaires fils de TOTFAC peut à son tour être défini par une somme algèbrique. C'est le cas en particulier de MBTTC défini à partir de MBHT et de TVA (figure 11).

nb niveau	Désignation des éléments	Identificateur
01	Total facture	TOTFAC
02	Montant brut TTC	+ MBTTC
03	Montant brut HT	+ MBHT
03	Montant des Taxes	+ TVA
02	Remise	- REMISE
02	Avoir	- AVOIR

Figure 11

β) Les expressions arithmétiques

Pour exprimer qu'un résultat est évalué par une expression arithmétique, il suffit de mentionner cette expression dans la zone CALCUL de la ligne où figure la désignation du résultat à définir en respectant les règles de priorité habituelles sur les opérateurs.

Ainsi pour déclarer que TVA est définie par le produit de TTVA et de MBHT on écrit :

nb niveau	Désignation des éléments	Calcul	Identificateur
03	Montant des taxes	MBHT * TTVA	(+) TVA
04	Taux TVA		TTVA

Figure 12

Remarque : Le fait d'avoir inscrit MBHT et TTVA dans la colonne CALCUL sur la même ligne que TVA, exprime déjà le fait que MBHT et TTVA sont les fils de TVA. Cependant pour définir TTVA à son tour, il est nécessaire de spécifier à nouveau cette relation de filiation à l'aide du jeu des nombres niveaux.

γ) Conclusion

Ces deux types de définitions consécutives suffisent à définir les résultats de type inconditionnel.

Pour définir un résultat d'un autre type, il est nécessaire de compléter sa description à l'aide d'une des deux définitions (alternative ou itérative) que nous allons introduire.

I.3.4.2 Les définitions alternatives

Au cours de l'analyse d'un problème le concepteur peut rencontrer des résultats qui sont conditionnels, c'est-à-dire qu'ils ne sont définis que si certaines conditions sont vérifiées.

Par ailleurs l'analyse à laquelle procéde habituellement le gestionnaire, celle qui lui est familière est une analyse par cas.

C'est la raison pour laquelle l'expression que nous avons choisie pour les définitions alternatives est du type :

SI condition ALORS définition explicite que l'on rencontre dans la plupart des méthodes d'analyse : CORIG, PROTEE, ARIANE
de préférence à la forme :
SI condition ALORS action 1 SINON action 2
préconisée dans CIVA (13) et la majorité des langages de programmation.

L'expression conditionnelle associée à la définition d'un résultat est portée dans la colonne CONDITION en respectant les règles de priorité habituelles sur les opérateurs logiques.

Par exemple, pour exprimer que la définition du résultat REMISE n'est valide que si la condition C1 = MBHT > BASE est vérifiée, on écrit la ligne suivante :

Niveau	Désignation des éléments	Condition	Calcul	Identificateur
02	Remise sur gros achats	MBHT > BASE	(MBHT-BASE) * TREM	- REMISE

Figure 13

Remarque 1 : La portée d'une condition exprimée sur une ligne de niveau i est limitée par l'apparition sur le descripteur d'une ligne de niveau inférieur ou égal à i.

Ainsi dans l'exemple de la figure (14) :

nb Niveau	Désignation des éléments	Condition	Calcul	Identificateur
02	Remise	MBHT > BASE	(MBHT-BASE) * TREM	- REMISE
03	Taux de réduction			TREM
02	Avoir			- AVOIR

Figure 14

la condition C1 = MBHT > BASE porte sur les définitions de REMISE et de TREM, mais ne porte pas sur la définition d'AVOIR.

Remarque 2 : Il n'est pas possible de faire intervenir, dans l'expression d'une condition C1, un résultat intermédiaire dont la définition est assujétie à la condition C1.

I.3.4.3 Les définitions itératives

α) Les définitions itératives associées aux occurrences d'un objet

Les travaux de gestion sont par nature des traitements par lots, qui produisent des résultats dont la structure est celle d'une liste.

Cette idée a imprégné les refléxions des chercheurs sur la méthodologie de programmation en gestion, PAIR-MAROLDT (24), WARNIER (39), BERTINI-TALLINEAU (7) ; elle s'est traduite par la mise en évidence d'un schéma de programme standard des problèmes de gestion qu'on retrouve sous des formes voisines dans REIX (33), CIVA (13), ATLAS (3), faisant apparaître des séquences de traitements répetées sur des lots de données qui sont les occurrences successives des caractéristiques d'un même objet.

De même, lorsque le concepteur rencontre au cours de l'analyse un résultat qui a une structure de liste(31), le gestionnaire sait que les transformations qui produisent chacun des termes de la liste, ne diffèrent que par le lot de données sur lesquelles elles portent. Pour définir une structure de liste, il suffit de donner la définition commune à tous les termes de la liste et de désigner l'objet du monde réel intéressé.

Sur le descripteur, la marque d'une définition itérative se fait en inscrivant la désignation de l'objet sur lequel porte la définition dans la colonne ENTITE.

Exemple : pour exprimer que TOTFAC est défini pour chaque occurrence de l'objet CLIENT on écrit :

Niveau	Désignation des éléments	Condition	Entité	Calcul	Identificateur
01	Total facture		CLIENT		TOTFAC
02	Montant brut TTC				+MBTTC
02	Remise	MBHT > BASE		(MBHT - BASE) * TREM	-REMISE
02	Avoir				-AVOIR

Figure 15

Remarque 1 : La portée d'une définition itérative exprimée sur une ligne de niveau i est limitée par l'apparition d'une ligne de niveau j inférieur ou égal à i.

Ainsi dans le cas de la figure (16) :

Niveau	Désignation des éléments	Condition	Entité	Calcul	Identificateur
02	Montant brut TTC				+ MBTTC
03	Montant brut HT				+ MBHT
04	Montant produit		PRODUIT	PU * NB	+ MPRD
02	Montant des taxes			MBHT * TTVA	+ TVA

Figure 16

la définition de TVA, étant portée sur une ligne de niveau j = 2, est indépendante de la définition itérative exprimée sur la ligne de niveau 4 définissant MPRD (montant produit).

Remarque : La définition d'une liste de résultats peut être conditionnée, pour exprimer que les termes de la liste ne sont définis que pour un sous-ensemble des occurrences de l'objet. Il est alors nécessaire d'associer à la définition de la liste de résultats deux lignes de définitions sur le descripteur :

. une première ligne où l'on porte la désignation de l'objet dans la colonne ENTITE.

. une deuxième ligne où l'on exprime les critères de sélection des occurrences de l'objet dans la colonne CONDITION.

Ainsi si l'on veut exprimer que MPRD est une liste de résultats définie seulement pour le sous-ensemble des produits "disponibles" :

Niveau	Désignation des éléments	Condition	Entité	Calcul	Identificateur
03	Montant brut HT				+ MBHT
04	Pour chaque produit		PRODUIT		
05	Montant ligne produit disponible	QUAL = 'DISPO'		PU * NB	+ MPRD

Figure 17

β. Les définitions itératives associées à la manipulation des variables indicées

Le processus de conception choisi dans Remora définit l'étape conceptuelle comme une étape de représentation indépendante des choix technologiques ; en particulier les structures de données de ce niveau sont conceptuelles et ne prennent en considération que les classes d'objets représentatives des phénomènes en compte indépendamment de la manière dont on

souhaitera accéder à ces données et à fortiori de la façon dont on pourra exploiter les données à partir de leurs structures d'implémentation physique.

Dans le langage de conception, le concepteur manipule les objets par leur désignation et construit peu à peu la structure conceptuelle des données en associant les désignations des propriétés à celles des objets auxquelles elles se rapportent (19).

Nous avons fait une exception à cette règle pour les structures de table.

Remarquons tout de suite qu'une structure logique de données pourra être implémentée sous forme de tableau en mémoire centrale à partir d'une définition conceptuelle habituelle.

Cependant, considérant que la manipulation des variables indicées est posée dans la pratique quotidienne du gestionnaire, nous admettons les notions de tableau et de variable indicée dans le langage.

Pour représenter l'ensemble des valeurs que peut prendre un indice (repère de la variable indicée prenant des valeurs entières) on porte dans la colonne ENTITE :

* une désignation de l'INDICE
* une désignation de la valeur initiale
* une désignation de la valeur finale
* une désignation du pas.

Eventuellement une condition explicite d'arrêt de la progression de l'indice peut être exprimée dans la colonne CONDITION sur la même ligne.

Par exemple, si dans le calcul de REMISE de la figure (7) le taux de remise est donné par le tableau T figure (18) :

tranche d'achat		taux
5 000	10 000	0.05
10 000	30 000	0.08
30 000	100 000	0.10

Figure 18

la définition REMISE peut être :

Nb niveau	Désignation des éléments	Condition	Entité	Calcul	Résultat
02	Remise	MBHT> 5 000		TREM* (MBHT-5000)	+ REMISE
03	Recherche de la tranche d'achat	T(I,1) < MBHT≤ T (I, 2)	I:1, 3, 1	T (I, J)	TREM

Figure 19

où TREM est défini comme l'élément du tableau T correspondant à la tranche d'achat à laquelle appartient MBHT.

Remarque : De telles définitions itératives facilitent l'expression de certains calculs de type franchement numériques, comme la manipulation de deux vecteurs A (x_1, ..., x_n) et B (y_1, ..., y_n) figure (20),

Nb niveau	Désignation des éléments	Condition	Entité	Calcul	Identificateur
01	Produit vectoriel				PROVEC
02	Produit XiYi		I = 1, N, 1	A(I)*B(I)	+ XIYI

Figure 20

qui peuvent être utiles dans certains problèmes décisionnels de gestion.

1.3.5 La modularité du langage

Si au cours de l'élaboration d'un module, le concepteur désire reporter à un instant ultérieur la description d'un résultat, il lui suffit de porter dans la colonne APPEL la désignation du module fils qui en donne la définition (31).

Exemple :

niveau	Désignation des éléments	Condition	Entité	Calcul	Identificateur	Appel
01	Montant total				TOTFAC	
02	Montant brut TTC				+ MBTTC	
03	Montant brut HT				+ MBHT	CALMBHT

Figure 21

La ligne de niveau 3 exprime que la définition de MBHT est (ou sera) donnée par le module CALMBHT.

Remarque 1 : A tout ensemble de définitions portées sur un descripteur est associé un module conceptuel, repéré par sa désignation portée sur la ligne introductrice du descripteur. Outre sa désignation, la définition d'un module comporte sa définition en clair et les désignations de la procédure et de l'application pour lesquelles il est décrit.

Remarque 2 : Le choix de la désignation d'un module est indépendant du choix de la désignation du résultat dont il donne la définition. Elles peuvent être différentes ou identiques.

I.3.6 Les définitions de la nature des mots

Comme on le fait apparaître sur l'arbre des résultats, les éléments de l'arbre ont une nature (donnée - résultat - résultat intermédiaire) (THIERY), (PEREA VILLACORTA). Les feuilles sont généralement des données. Nous avons souhaité conserver cette information dans le langage. Pour exprimer qu'un mot est en fait une donnée, il suffit d'inscrire sa désignation dans la colonne DONNEE.

Exemple :

Nb niveau	Désignation des éléments	Condition	Entité	Calcul	Donnée	Identificateur	Appel
02	Avoir				AVOIR	+AVOIR	

Figure 22

Cette ligne de définition spécifie que AVOIR désigné dans la colonne IDENT est en fait une donnée.

Remarque 1 : La spécification de la nature donnée d'un mot du langage est utilisée par les outils d'aide à la conception développés par THIERY (37).

Remarque 2 : Cette définition est, d'autre part, utilisée et complétée dans la phase d'élaboration de la structure conceptuelle qu'étudie actuellement M. KRIEGUER (19).

I.3.7 Définition normalisée de la syntaxe du langage

Nous présentons la syntaxe du langage avec les conventions de Backus dont nous rappelons les principales :

Conventions de Backus :

- la partie gauche et la partie droite d'une production sont séparées par le symbole ::=
- les alternatives sont données de part et d'autre du signe /
- les mots entre < > sont les métatermes du langage
- la présence du symbole { } indique qu'une et une seule des alternatives supposées est obligatoire.
- les parties de production entre [ ] sont facultatives.
- les parties de production entre [ ]* peuvent se répéter de o à n fois.
- les mots **employés** sont des symboles du langage traduit de manière typographique sur le descripteur suivant le tableau de correspondance :

Métaterme	Zone
* *nombre-niveau*	*NIVEAU*
* *définition en clair*	*DESIGNATION DES ELEMENTS*
* *désignation R*	*IDENTIFICATEUR*
* *désignation D*	*DONNEE*
* *appel*	*APPEL*
* *expression arithmétique*	*CALCUL*
* *SI condition ALORS*	*CONDITION*
* *JUSQUA condition*	
* *POUR CHAQUE désignation*	*ENTITE*

*<module>::=<entête> <définition> [<définition>]**

<entête>::=<désignation module> <texte>

*<définition>::=<définition consécutive> [<définition>]**

*::=<définition alternative> [<définition>]**

*::=<définition itérative> [<définition>]**

<Définition consécutive> ::= <spécification> <expression arithmétique> <désignation R>

/ <spécification> <désignation R>

/ <spécification> <désignation R> <appel>

/ <spécification> <désignation D> <désignation R>

<spécification> ::= <nombre niveau> <définition en clair>

<désignation R> ::= |<signe>| <désignation>

<désignation D> ::= <désignation>

<désignation> ::= 1 à 8 chiffres ou lettres commençant par une lettre

<signe> ::= <+> | <->

<expression arithmétique> ::= <expression arithmétique au sens de PL1>

<définition alternative> ::= <spécification> <SI condition ALORS>

/ <SI condition ALORS> <définition consécutive>

<condition> ::= expression conditionnelle munie des opérateurs :

ET
OU
NON
<
+
< =
>
> =
≠

<définition itérative> ::= <spécification> <critère d'itération>

/ <critère d'itération> <définition consécutive>

<critère d'itération> ::= <POUR CHAQUE Entité>

/ <POUR CHAQUE Indice> <JUSQUA condition>

<entité> ::= <désignation>

<POUR CHAQUE Indice> ::= <désignation indice> <valeur initiale> <valeur finale> <pas>

/ <désignation indice> <valeur initiale> <valeur finale>

/ <désignation indice> <valeur initiale> <JUSQUA condition>

/ <désignation indice> <JUSQUA condition>

<indice> ::= <désignation>

<valeur initiale> ::= <désignation 1>

<valeur initiale> ::= <désignation 1> | <nombre>

<valeur finale> ::= <désignation 1> | <nombre>

<désignation 1> ::= 1 à 4 chiffres ou lettres commençant par une lettre.

I - 4 - L'EXEMPLE

Le descripteur de traitement associé à l'exemple du paragraphe I.22.1 est finalement :

n° ligne	MODULE : FACTURE TEXTE : Calcul total facture PROCEDURE : FACTURATION ..							
	niveau	Désignation des éléments	Condition	Entité	Calcul	Donnée	Identificateur	Appel
1	01	Total facture		CLIENT			TOTFAC	
2	02	Mont. brut TTC					+ MBTTC	
3	03	Mont. brut HT					+ MBHT	CALMBHT
4	03	Taxes : TVA			TVA*MBHT		+ TVA	
5	04	Taux de TVA				TTVA	TTVA	
6	02	Remise éventuelle	MBHT > 5000		(MBHT-5000)*TREM		- REMISE	
7	03	Taux Remise				TREM	TREM	
8	02	Avoir Client				AVOIR	- AVOIR	

Figure 23

Remarquons que compte tenu des correspondances entre les caractéristiques de la méthode et celles du langage, l'algorithme de passage de l'arbre des résultats à l'expression arborescente des définitions se ramène à deux actions :

1 - exploration de l'arbre des résultats branche par branche, de haut en bas et de gauche à droite

2 - interprétation des nœuds basée sur leur nature, leur type et le choix qui a pu être fait d'un appel à une définition ultérieure.

L'exploration de l'arbre se traduit par l'écriture de lignes successives respectant le jeu des niveaux.

Ligne 7 : Définition de TREM : fils de REMISE ⟹ nombre niveau = 3

* c'est une donnée, sa désignation est reportée dans la colonne "Donnée".

Ligne 8 : Définition de AVOIR : fils de TOTFAC ⟹ nombre niveau = 2

* il entre comme élément soustractif dans la somme algébrique définissant TOTFAC. On lui associe le signe '-'.
* c'est une donnée : sa désignation est reportée dans la colonne "Donnée".

Ecriture du module CALMBHT

A l'arbre des résultats issu de l'analyse de MBHT :

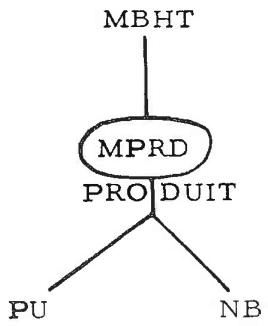


Figure 24

correspond le module CALMBHT :

Module : CALMBHT TEXTE : calcul montant brut hors taxe PROCEDURE : FACTURATION ..							
Niveau	Désignation des éléments	Condition	Entité	Calcul	Donnée	Identificateur	Appel
01	Mont. brut HT						
02	Mont. ligne produit		PRODUIT	PU*NB		+MPRD	
03	Prix unitaire				PU		
03	Nombres commandés				NB		

Figure 25

DEUXIEME PARTIE :

LES STRUCTURES DE TRAITEMENTS

LA GENERATION DES TRAITEMENTS

INTRODUCTION

Dans le chapitre I, nous avons proposé un langage qui permet au concepteur de donner une définition conceptuelle d'un problème de gestion. Notre objectif, maintenant, est de montrer que ce langage est directement exploitable par l'ordinateur, c'est-à-dire, qu'il est possible d'obtenir un programme exécutable à partir d'énoncés écrits dans le langage de conception.

En fait, quand nous parlons de programme, nous ne considérons que la formulation algorithmique des traitements en fonction d'un certain nombre d'objets informatiques (réels, entiers, booléens, caractères ...). Nous supposerons que les descriptions de ces objets sont générées par des outils spécifiques.

L'étude des langages de programmation nous a montré qu'un programme peut être considéré comme :

- un ensemble d'instructions où sont décrites d'une part les opérations d'accès aux informations, d'autre part les opérations arithmétiques dont l'exécution en séquence permet d'obtenir les résultats.

- une structure qui sous-tend cet ensemble d'énoncés, que nous appellerons structure d'implémentation physique des traitements, dans laquelle nous distinguons d'une part la logique des traitements, d'autre part la logique de mise en œuvre technologique utilisée pour les automatiser.

De même le langage de conception que nous proposons peut être défini comme :

- un ensemble d'enoncés où sont formulées les définitions des résultats à obtenir.

- une structure, qui sous-tend cet ensemble de textes, que nous appellerons structure conceptuelle des traitements, caractéristique de la

sémantique des transformations subies par les objets du monde réel représentés dans le si.

Réaliser la génération d'un langage de conception des problèmes, c'est-à-dire assurer le passage de la définition conceptuelle de traitements à leur définition programmée, revient :

1. à définir deux types de fonctions :

   11. une fonction de transformation de structures
   12. une fonction de transformation des énoncés.

2. à construire un outil, que nous appellerons générateur de programmes, automatisant ces fonctions.

Par ailleurs, il est intuitif que ces transformations sont dépendantes des choix que le concepteur fait au cours du processus de conception du SI, comme par exemple la détermination du langage de programmation dans lequel seront exprimés les programmes, ou la structure d'implémentation physique des données.

Schématiquement, un générateur de programmes associé à un langage de conception des traitements, tels que celui de REMORA est un transformateur :

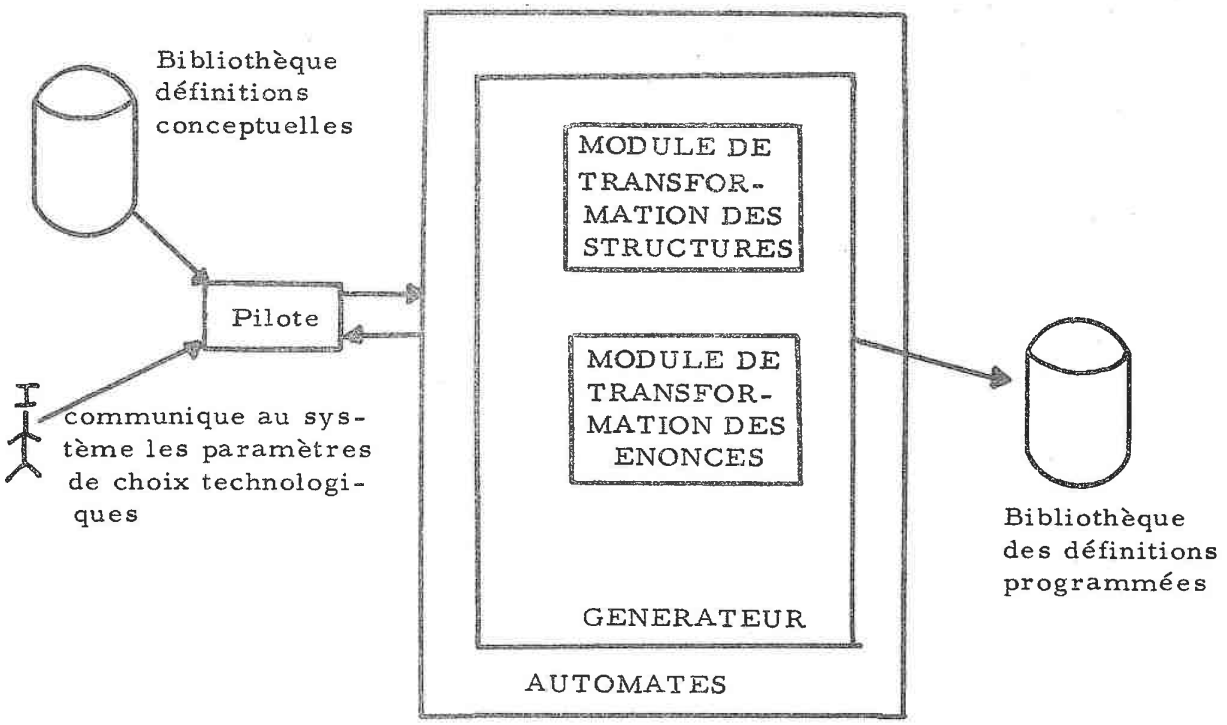


Figure 1 - Schéma standard d'un générateur de programme.

Nous étudierons au chapitre II les deux fonctions d'un tel générateur ; nous présenterons au chapitre III les algorithmes définis et implémentés dans les outils de REMORA pour assurer ces fonctions du générateur. Enfin, nous validerons ces algorithmes sur l'exemple exposé au premier chapitre.

CHAPITRE II

LES FONCTIONS D'UN GENERATEUR

Nous pensons qu'il n'est pas logique de produire les programmes directement à partir de la définition conceptuelle, mais qu'il est préférable, comme nous en avons fait l'hypothèse pour tout le processus de conception réalisation des SI, de procéder par étapes et de dissocier :

. la structuration des traitements et l'étude des formes du langage correspondant à la conception des traitements,

. la génération des programmes correspondant à leur réalisation.

Nous aborderons dans ce chapitre :

1. Le problème de la conception c'est-à-dire, l'étude du passage de la définition conceptuelle des traitements à leur définition programmable ; nous procéderons en deux étapes :

   1.1 Analyse de la fonction de transformation des structures !

   * Nous montrerons les structures de traitements successives ;
   * nous étudierons les règles de passage de l'une à l'autre.

   1.2 Analyse de la fonction transformation des textes.

   * Nous présenterons les formes du langage dans lesquelles sont exprimées les textes successifs ;
   * et nous étudierons le passage d'une forme à l'autre.

2. Le problème de la génération, c'est-à-dire l'étude du passage de l'expression physique des traitements au programme.

II.1 LA FONCTION DE TRANSFORMATION DES STRUCTURES

L'objectif de cette fonction est d'assurer le passage entre la structure conceptuelle des traitements et une structure d'implémentation physique de ces traitements.

Ce passage n'est pas unique, puisqu'à une structure conceptuelle on peut faire correspondre plusieurs structures physiques. Il dépend, en particulier, des choix effectués au niveau logique sur les fonctions d'accès aux données et au niveau physique sur l'implémentation de ces fonctions ; c'est la raison pour laquelle nous proposons un passage en deux étapes s'appuyant sur une structure intermédiaire de traitements que, par analogie avec les étapes du processus de conception du SI, nous appellerons structure logique.

Schématiquement :

- la conception des traitements passe par la définition de trois structures :
  - la structure conceptuelle est celle des énoncés fournis par le concepteur en s'appuyant sur le modèle de perception du réel proposé dans les modules conceptuels formulés sur les descripteurs de traitement ;
  - la structure logique est celle des algorithmes déduits d'une certaine composition d'énoncés dans des unités opérationnelles. Elle reflète la logique d'exécution des traitements en fonction des contraintes de disponibilité des résultats et indépendamment des contraintes liées à l'accès physique aux données ;
  - la structure physique est celle du programme déduit de l'étape précédente et compte tenu des choix d'implémentation physique des structures de données ;

. la réalisation des traitements correspond à la génération des programmes dans le langage de programmation choisi.

II.11 Les structures de traitement

Nous nous proposons dans ce paragraphe de présenter la structure de traitement que nous avons choisie pour chacun des niveaux conceptuel, logique et physique.

II.111 La structure conceptuelle des traitements

A partir des résultats, on se propose d'exprimer complètement un problème en inventoriant toutes les définitions qui sont nécessaires sans se préoccuper de la chronologie de l'utilisation de chacune de ses définitions.

La structure conceptuelle des traitements représente, pour un problème donné, la combinaison des trois types de définitions antérieurement examinées,

- définitions itératives
- définitions consécutives
- définitions alternatives

associées aux trois types de résultats :

- liste de résultats
- résultats inconditionnels
- résultats conditionnels.

Nous définirons les concepts nécessaires à la construction de la structure conceptuelle des traitements avant d'en donner une définition complète.

II.III.1 Le concept de groupe conceptuel de traitement

Définition : Nous appelons groupe conceptuel de traitement de niveau i et nous noterons G_i l'ensemble des définitions non décomposées nécessaires à l'obtention d'un résultat.

Si l'on choisit de représenter la combinaison des définitions sous forme de tableau comme nous l'avons fait dans Remora, alors le groupe conceptuel de traitement de niveau i peut se définir comme :

l'ensemble des lignes comprises entre une ligne de niveau i et la première ligne de niveau j inférieur ou égal à i (cette ligne ne fait pas partie du groupe conceptuel) ; soit :

$$G_i = [\text{ ligne i, ligne j }[\quad \text{avec} \quad j \leq i.$$

Ainsi sur le module conceptuel FACTURE suivant :

Niveau	Désignation élément	Condition	Entité	Calcul	Donnée	Identificateur	Appel
0 1	Total facture d'l client		CLIENT			TOTFACT	
0 2	Montan: Brut TTC					+MTBTTC	
0 3	Montan: Brut HT					+MTBHT	
0 4	Mont. ligne commande		PRODUIT	PV * QTC		+MTLIHT	
0 3	Montan: TVA			MTBHT * TTVA		+TVA	
0 2	Remise	MTBHT > 5000		MTBHT*TREM		-REMISE	
0 2	Avoir client				AVOIR	-AVOIR	

on définit les groupes logiques suivants :

* 1 groupe logique de niveau 1

0 1	Total facture d'un client	
0 2	Montant brut TTC	
0 3	Montant brut HT	
0 4	Montant ligne commande	
0 3	Montant TVA	
0 2	Remise	
0 2	Avoir client	

* 3 groupes logiques de niveau 2

0 2	Montant brut TTC	
0 3	Montant brut HT	
0 4	Montant ligne commande	
0 3	Montant TVA	

0 2	Remise	

0 2	Avoir client	

* 2 groupes logiques de niveau 3

0 3 0 4	Montant brut HT Montant ligne commande	

0 3	Montant TVA	

* 1 groupe logique de niveau 4

0 4	Montant ligne commande	

On remarque que le nombre de groupes logiques d'un module est égal au nombre de lignes de définitions du descripteur associé.

A chaque groupe logique on peut alors associer un numéro d'ordre égal au rang, sur le descripteur, de la ligne de définition qui introduit ce groupe logique.

On appelle groupe logique n° k de niveau i noté G_i^k le groupe de niveau i introduit par la kème ligne de définition.

Les groupes logiques du module conceptuel FACTURE sont alors :

n° ligne	niveau	Désignation des éléments		
1	0 1	Total facture d'l client		G_1^1
2	0 2	Montant brut TTC		G_2^2
3	0 3	Montant brut HT	G_3^3	
4	0 4	Montant ligne commande	G_4^4	
5	0 3	Montant TVA	G_3^5	
6	0 2	Remise		G_2^6
7	0 2	Avoir client		G_2^7

II.III.2 La nature d'un groupe conceptuel de traitement

Par correspondance avec la typologie des définitions, nous définissons :

- des groupes conceptuels itératifs
- des groupes conceptuels conditionnels
- des groupes conceptuels inconditionnels.

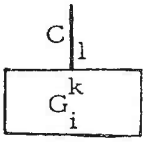
Définition : La nature d'un groupe conceptuel de traitement découle du caractère itératif ou alternatif ou consécutif du résultat qu'il définit.

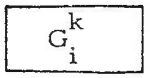
Ainsi, les natures des groupes conceptuels de traitement définis sur le module conceptuel FACTURE sont les suivants :

Groupe logique	N° LIGNE	DESCRIPTEUR				nature définition	nature groupe logique
		niveau	Désignation des éléments	Condition	Entité		
G_1^1	1	0 1	Total facture d'l client		CLIENT	Itérative	Itératif
G_2^2	2	0 2	Montant brut TTC			Consécutive	Inconditionnel
G_3^3	3	0 3	Montant brut HT			Consécutive	Inconditionnel
G_4^4	4	0 4	Montant ligne commande		PRODUIT	Itérative	Itératif
G_3^5	5	0 3	Montant TVA			Consécutive	Inconditionnel
G_2^6	6	0 2	Remise	MTBHT > 5000		Alternative	Conditionnel
G_2^7	7	0 2	Avoir client			Consécutive	Inconditionnel

Pour représenter graphiquement un groupe logique muni de sa nature, nous utiliserons le symbolisme suivant :

groupe logique inconditionnel 

groupe logique conditionnel 

groupe logique itératif 

ENTITE

II. III. 3 La combinaison de deux groupes logiques

Soient deux groupes conceptuels distincts G_i et G_j. Examinons les positions de l'un par rapport à l'autre.

Deux groupes peuvent être indépendants, associés ou consécutifs ; dépendants ou imbriqués.

* Groupes conceptuels dépendants ou imbriqués dans le groupe G_i^k $(i<j)$

On dit que G_j^k est enchassé dans G_i^k si G_j^k participe à la définition des résultats qui constituent G_i^k. En d'autres termes, on dit que deux groupes conceptuels G_i^k et $G_j^{k'}$ de niveau i et j $(i < j)$ sont imbriqués si toutes les lignes appartenant au groupe conceptuel $G_j^{k'}$ appartiennent au groupe conceptuel G_i^k.

On note : $G_j^{k'} \subset G_i^k$.

La réunion de G_i^k et de $G_j^{k'}$ est égale à G_i^k et l'intersection de G_i^k et de $G_j^{k'}$ à $G_j^{k'}$.

Définition :

$$\forall i, \forall j \quad i < j \quad (G_j^{k'} \subset G_i^k \iff G_i^k \cup G_j^{k'} = G_i^k)$$

ou

$$\forall i, \forall j \quad i < j \quad (G_j^{k'} \subset G_i^k \iff G_i^k \cap G_j^{k'} = G_j^{k'})$$

Sur l'exemple de la facture précédent G_4^4 imbriqué dans le groupe G_1^1

niveau	désignation élément	condition	Entité	
0 1	Total facture d'l client		CLIENT	
0 2	Montant brut TTC			G_1^1
0 3	Montant brut HT			
0 4	Mont. ligne commande		PRODUIT	G_4^4
0 3	Montant TVA	MTBHT > 5000		
0 2	Remise			
0 2	Avoir client			

En fait, pour qu'un groupe $G_j^{k'}$ soit imbriqué dans un groupe conceptuel G_i^k, il faut et il suffit que la ligne introductrice de $G_j^{k'}$ appartienne au groupe conceptuel G_i^k.

* groupes conceptuels consécutifs ou associés si leurs définitions sont indépendantes et si elles participent à la définition d'un même G^k

Deux groupes G_i^k et $G_j^{k'}$ de rang k et k' ($k < k'$) sont consécutifs si aucune ligne du groupe $G_j^{k'}$ n'appartient au groupe G_i^k. On dit alors que $G_j^{k'}$ succède à G_i^k et on note : $G_i^k \longrightarrow G_j^{k'}$.

En conséquence, l'intersection de G_i^k et $G_i^{k'}$ est vide.

Définition :

$$\forall k,\ \forall k' \quad k' < k \quad (G_i^k \longrightarrow G_j^{k'} \iff G_i^k \cap G_j^{k'} = \emptyset)$$

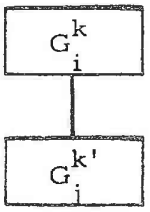
Sur l'exemple de la facturation G_3^4 succède au groupe conceptuel G_3^3

niveau	Désignation élément	Condition	Entité	
0 1 0 2	Total facture d'l client Montant brut TTC		CLIENT	
0 3 0 4	Montant brut HT Mont. ligne commande		PRODUIT	G_3^3
0 3	Montant TVA			G_3^4
0 2 0 2	Remise Avoir client	MTBHT > 5000		

En fait, pour qu'un groupe $G_j^{k'}$ succède à un groupe G_i^k, il faut et il suffit dans le langage REMORA, que la ligne introductrice de $G_j^{k'}$

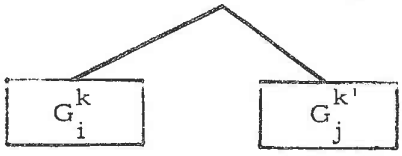
- succède à la ligne introductrice de G_i^k sur le descripteur de traitement
- n'appartienne pas au groupe G_i^k .

Nous représentons graphiquement les deux types de structures en groupes possibles par :



$\sim \quad G_j^{k'} \subset G_i^k,$

structure élémentaire de groupes imbriqués



$G_i^k \longrightarrow G_j^{k'}$

structure élémentaire des groupes consécutifs.

II.111.4 Prise en compte de la modularité de la conception

Au cours de la détermination des définitions d'un résultat peuvent apparaître des groupes conceptuels qui ont déjà été décrits par ailleurs, ou qu'on souhaite décrire ultérieurement. Dans la structure conceptuelle ces groupes sont désignés.

La ligne comportant un appel introduit un tel groupe et définit un groupe que nous qualifions de groupe d'appel.

Le module conceptuel FACTURE introduit précédemment pourrait prendre la forme suivante :

DESIGNATION DU MODULE : FACTURE							
	Niveau	DESIGNATION	Condition	ENTITE		IDENT.	APPEL
	0 1	Total facture d'l client		CLIENT		TOTFACT	
	0 2	Montant brut TTC				+MTBTTC	
G_3^3	0 3	Montant brut HT				+MTBHT	B
	0 3	Montant TVA	MTBHT > 5000		MTBHT*TTVA	+TVA	
	0 2	Remise			MTBHT*TREM	-REMISE	
	0 2	Avoir du client			AVOIR	-AVOIR	

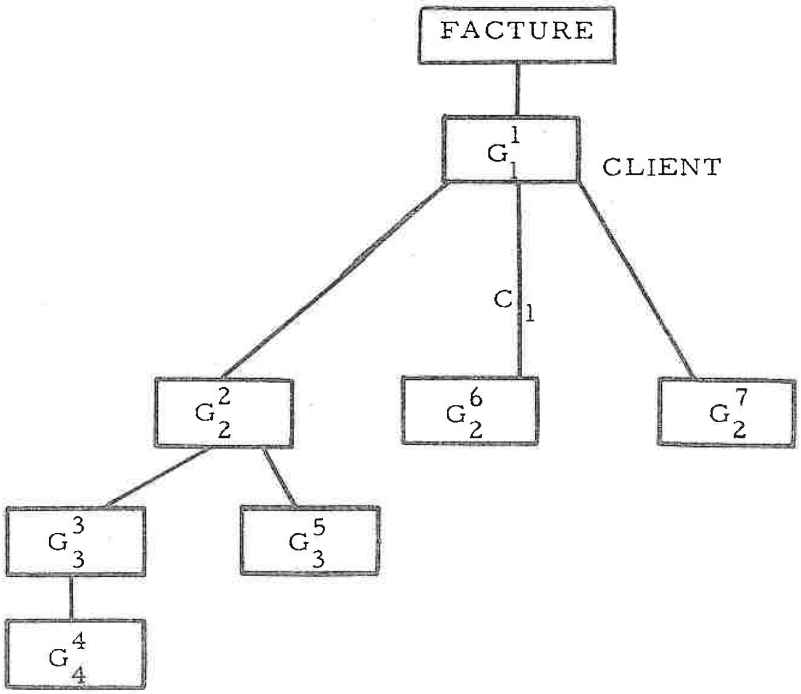
Ce module comporte un groupe conceptuel d'appel : G_3^3 .

II.111.5 La structure conceptuelle des traitements

A partir des concepts de groupe conceptuel et de leurs liens, on peut construire la structure conceptuelle d'un traitement qui est la représentation, à l'aide des deux structures de base (imbriquée, consécutive), de la combinaison des groupes conceptuels dans le module conceptuel qui définit ce traitement.

La structure conceptuelle est une hiérarchie arborescente de groupes conceptuels. On peut, dans une structure conceptuelle imbriquer ou associer les groupes conceptuels de toute nature (conditionnels, inconditionnels, itératifs, d'appel).

Ainsi, au module conceptuel FACTURE correspond la structure conceptuelle suivante :



On peut remarquer que la structure conceptuelle des traitements définie reflète la méthode d'analyse des problèmes développée dans la thèse de W. PEREA [31].

La "conception descendante" se retrouve dans la hiérarchie arborescente des groupes ; à la typologie ternaire des propriétés des résultats du monde réel correspond la typologie ternaire des groupes de définitions conceptuelles.

II.111.6 Notion de module conceptuel

Remarquons qu'un module conceptuel est l'ensemble structuré des définitions d'un résultat de niveau 0.

II.112 La structure logique des traitements

Au niveau logique on poursuit l'analyse des traitements et on complète leur description du niveau conceptuel en effectuant soit des regroupements, soit des éclatements de modules conceptuels pour tenir compte des contraintes liées,

ou à l'usage des résultats (fréquence, régularité, ...)

ou à leur chronologie de production.

Comme l'avons fait pour le niveau conceptuel, nous présenterons les concepts du niveau logique et définirons la structure logique des traitements.

II.112.1 Notion de paragraphe de traitement

Définition : Un paragraphe de traitement est une suite de fonctions élémentaires de traitement de même niveau, exécutables dans cet ordre comme un tout homogène dans les mêmes conditions logiques.

Chaque fonction élémentaire, peut être, dans une décomposition plus fine, décrite comme un paragraphe de traitement de niveau inférieur au paragraphe de traitement qu'elle décrit.

Nature d'un paragraphe de traitement :

Il peut être inconditionnel

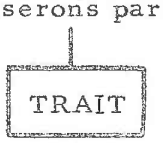
conditionnel

itératif

- les paragraphes de traitements conditionnels sont a exécuter 0 ou 1 fois suivant qu'une condition est réalisée ou non ; nous les représentons par :

TRAIT : désignation du paragraphe (et ultérieurement étiquette de la suite d'instructions dans le programme) ;

- les paragraphes de traitements inconditionnels sont à exécuter 1 fois ; nous les schématiserons par :



- les paragraphes de traitements itératifs sont à exécuter n fois ($n \geq 0$) ; n est un critère d'itération qui peut être
  - une constante précisant le nombre fixe d'itérations
  - une condition d'arrêt
  - ou un indicatif, repère d'un lot de données dont chaque occurrence doit subir le traitement spécifié.

Nous les représenterons graphiquement par :

Paragraphe d'appel

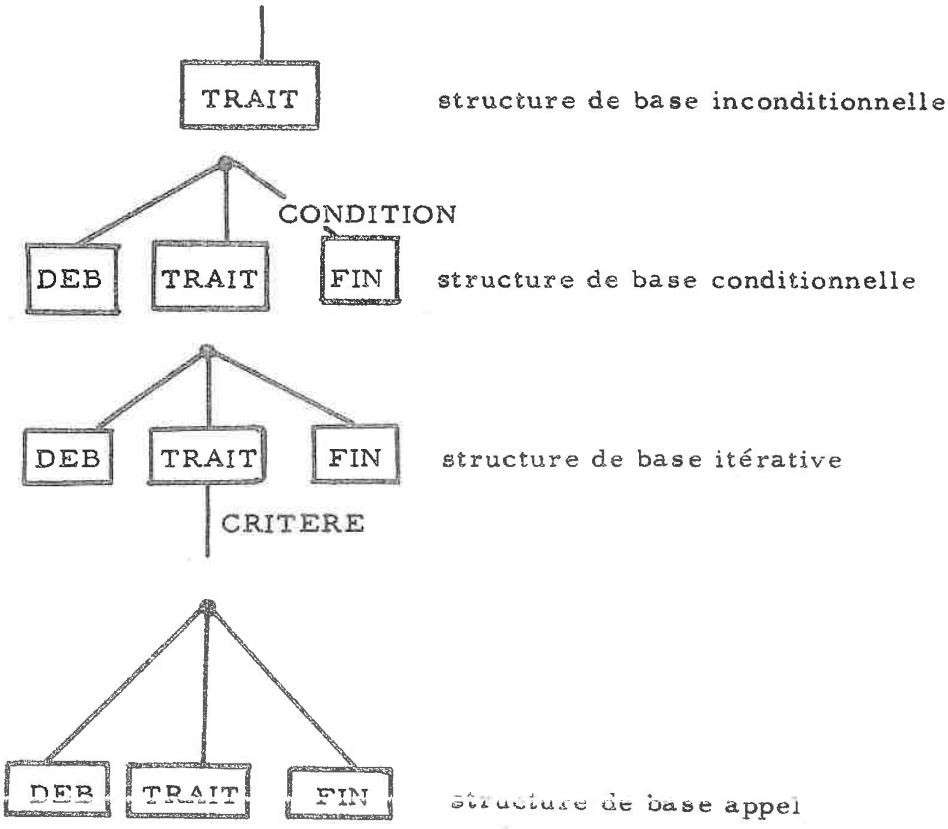
Comme au niveau conceptuel et en appliquant les mêmes règles, on fait apparaître, dans la structure logique des traitements, des paragraphes d'appel qui renvoient à des paragraphes décrits dans d'autres structures logiques de traitement.

Remarque : Compléments relatifs à la description des paragraphes de type conditionnel itératif ou d'appel

L'introduction dans la structure logique de tout paragraphe conditionnel, d'appel ou itératif, entraîne pour chacun d'eux l'introduction de paragraphes d'initialisation et de terminaison.

Ces paragraphes d'initialisation ou de terminaison sont par nature des paragraphes inconditionnels.

Ainsi, les trois structures élémentaires de la structure logique deviennent



II.112.2 Liens entre les paragraphes de traitement

Si nous reprenons la terminologie et les définitions utilisées au niveau conceptuel, les paragraphes de traitement, comme les groupes conceptuels sont consécutifs voir imbriqués.

II.112.3 Structure logique des traitements

La structure logique des traitements est obtenue soit par imbrication à plusieurs niveaux des trois structures de base précédemment citées, soit par concaténation de ces structures au même niveau.

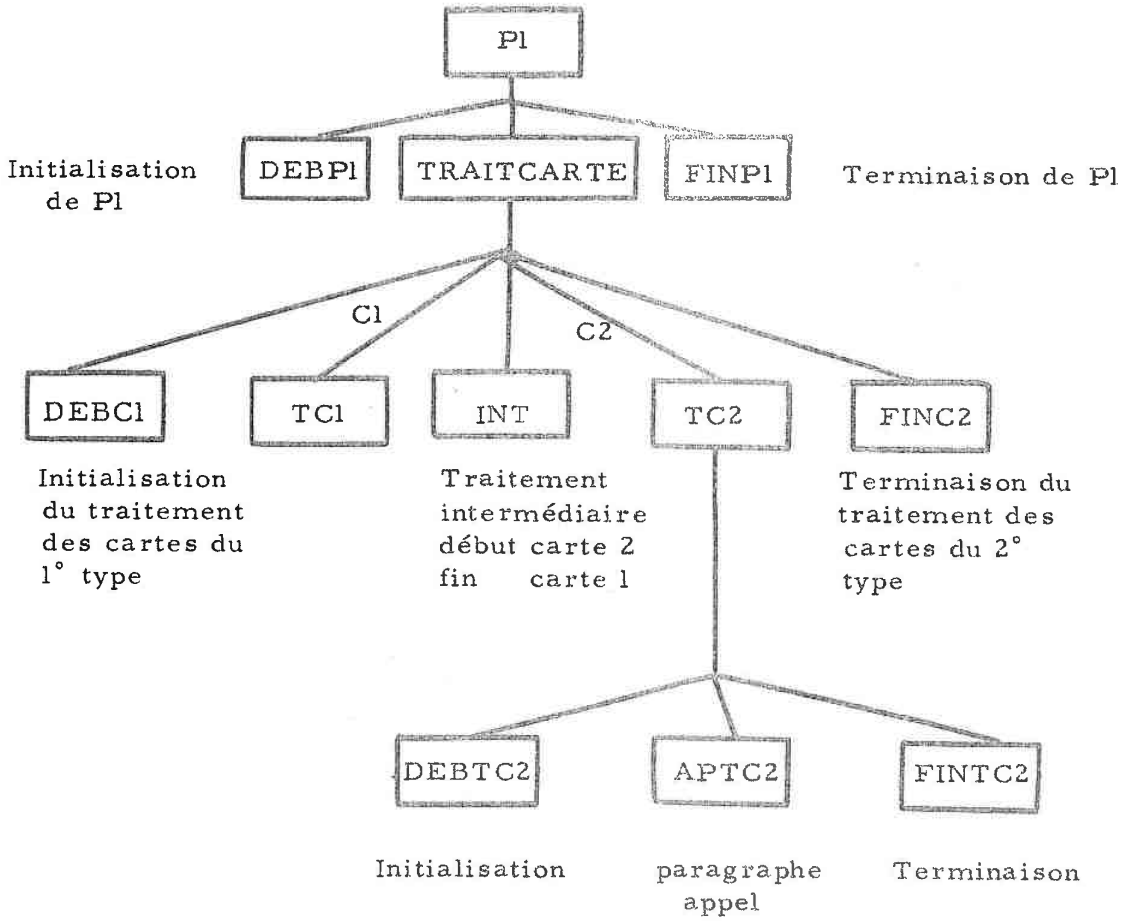
Toute expression algorithmique d'un problème peut être représentée par une hiérarchie arborescente de paragraphes de traitement à partir des trois structures de base.

Dans cette hiérarchie la racine est constituée par un paragraphe de traitement de niveau 0 qui est défini par n paragraphes de traitement de niveau 1 qui se définissent à leur tour par des paragraphes de traitement de niveau n pour aboutir à des paragraphes de traitement qui ne sont plus décomposables qu'en fonctions élémentaires de traitement indécomposables et que nous appelons paragraphes élémentaires.

Dans une structure de traitement, on ne peut pas imbriquer des paragraphes inconditionnels à des paragraphes inconditionnels. Par nature, des paragraphes inconditionnels sont des paragraphes élémentaires.

En effet, si pour répondre aux besoins de l'analyse en étapes il était naturel d'autoriser le concepteur à définir un groupe conceptuel inconditionnel par une hiérarchie de groupes, il est normal, au niveau logique, pour des raisons d'efficacité prévisibles, de supprimer les niveaux.

Exemple :



II.112.4 Notion d'unité opérationnelle de traitement

Nous avons vu qu'au niveau conceptuel, le concepteur était amené à structurer son problème en modules. Similairement au niveau logique il est amené à le structurer en unités opérationnelles de traitement. Chaque unité opérationnelle de traitement correspond à un regroupement de paragraphes logiques de traitement de niveau 0 combinés dans une seule structure logique de traitement. Ce regroupement est commandé par des exigences d'exécution unitaire des différents paragraphes de traitement concernés.

II.113 La structure physique des traitements

Au niveau physique, la description des traitements est complétée par la prise en compte des paramètres relatifs à l'organisation sur les supports de mémorisation des données à traiter et des paramètres relatifs aux moyens de traitement qui pourraient rendre nécessaire ou utile le regroupement ou l'éclatement des unités opérationnelles de traitement telles qu'elles ont été définies lors de l'étape logique.

Dans nos développements, nous ne prenons en compte que les paramètres relatifs à l'organisation des données. Nous faisons l'hypothèse que les moyens de traitement sont suffisants et n'introduisent pas de contraintes spécifiques.

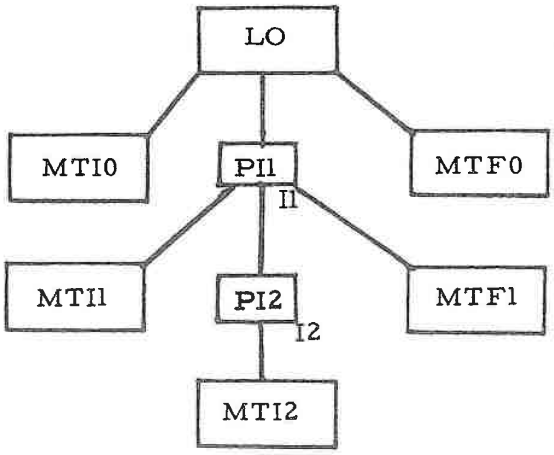
Au niveau physique, nous n'avons pas jugé nécessaire d'introduire de nouveaux concepts. Nous avons conservé le concept du paragraphe de traitement en le complétant par des renseignements qui concernent les organisations de données et le concept de structure logique des traitements qui présente au niveau physique des caractéristiques identiques à celles du niveau logique.

Ainsi, la structure physique de l'unité opérationnelle qui sera la structure du programme associé, repose sur les mêmes concepts que la structure logique. Ce sont ceux de la programmation structurée au sens de DIJKSTRA [12], repris en France par BERTINI et TALLINEAU [7], WARNIER [39], PAIR [29].

La structure physique de l'unité opérationnelle comportera tous les éléments de la structure logique de l'unité opérationnelle. En outre, elle contiendra les paragraphes de traitement exprimant la prise en compte des fonctions d'accés aux structures de données implémentées.

Dans l'état actuel de nos travaux, nous nous sommes limités à étudier et traiter les structures physiques des traitements dans le cadre d'unités opérationnelles travaillant sur des données implémentées dans des fichiers à organisation séquentielle et directe.

Notre solution est inspirée des travaux d'ATLAS [3] et de HERTSCHUH [18] qui ont clairement montré les problèmes associés à la cinématique des fichiers séquentiels et ont proposé des outils pour les traiter ; outils que nous avons repris dans le générateur.
A une structure logique ainsi schématisée :



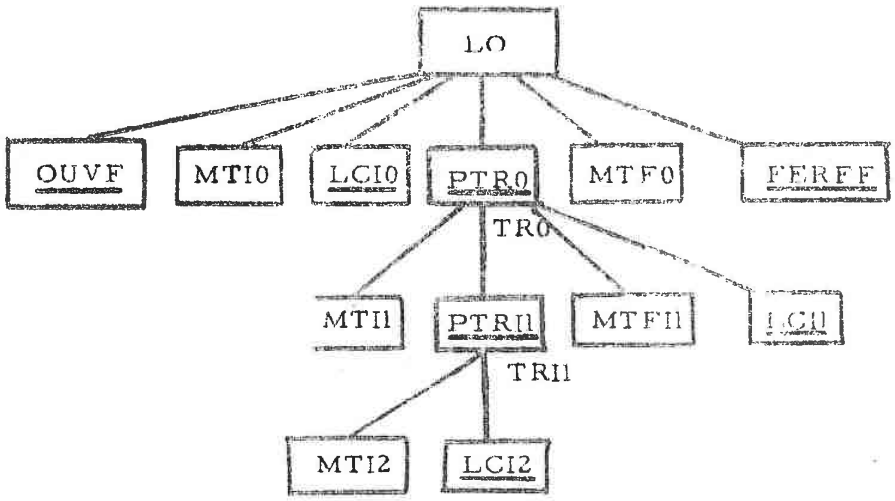
que N. HERTSCHUH décrit par la méta introduction

```
        MTI0
pour chaque I1  faire
        MTI1  ;
pour chaque I2  faire
        MTI2
                fin
        MTFI1
                fin
        MTF0 ;
```

correspond, dans le cadre de données organisées en fichiers séquentiels, la structure physique décrite par la méta instruction :

	Commentaires
OUVF	Ouverture des fichiers
MTI0	
LCI0	Lecture initiale de tous les fichiers séquentiels
jusqu'à TR0=VRAI faire	Test de fin de tous les fichiers séquentiels
MTI1	
jusqu'à TR1= VRAI faire	Test de rupture de l'indicatif I1
MTI2	
LCI2	Lecture des fichiers d'indicatif niveau I2 et choix de la valeur de l'indicatif I2 à traiter
fin	
MTFI1	
LCI1	Lecture des fichiers et indicatif niveau I1 et choix de la valeur de l'indicatif I1 à traiter
fin	
MTF0	
FERF	Fermetures des fichiers

que nous symbolisons par le schéma :



II.114 Conclusion

Les trois structures :

conceptuelle

logique

physique

correspondent à trois expressions de la solution en termes de traitement de plus en plus dependantes des choix et contraintes techniques.

La structure conceptuelle est le reflet des transformations subies par les objets représentatifs de l'organisation (ses clients, ses produits ...) pour l'application de règles de gestion

La structure logique est l'image de fusions ou éclatements de structures conceptuelles à travers des unités opérationnelles compte tenu de choix sur les modes de traitements (différé, temps réel), sur les priorités accordées à tel traitement vis à vis de tel autre, sur leurs périodicités.

La structure physique est l'image du programme tel qu'il sera exécuté pour des fichiers ou une base de données choisis. Chaque structure

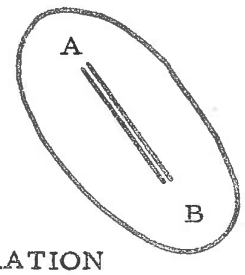
est indépendante de la structure qui la suit dans le processus de conception. A un problème schématisé par la structure conceptuelle peuvent correspondre plusieurs structures logiques équivalentes sémantiquement mais traitant différemment les contraintes introduites au niveau logique ; à une même structure logique, peuvent correspondre, si l'on modifie les techniques d'implémentation des données, plusieurs structures physiques.

II.12 Caractéristiques de passage d'une structure à une autre

Nous nous attacherons dans ce paragraphe à présenter les principes de passage d'une structure à l'autre en nous appuyant :

- sur les caractéristiques des structures de départ et d'arrivée
- sur les paramètres de choix introduits aux différents niveaux.

Pour éclairer notre exposé, nous présenterons ces transformations sur un exemple de FACTURATION défini par la composition



FACTURATION

à partir des modules conceptuels A et B suivants :

DESIGNATION DU MODULE : A							
Niveau	Désignation Elément	Condition	Entité	Calcul	données	Identificateur	Appel
01	Total fact. d'un client		CLIENT			TOTFACT	
02	Montant brut TTC					+ MTBTTC	
03	Montant brut HT					+ MTBHT	B
03	Montant TVA			MTBHT * TTVA		+ TVA	
02	Remise	MTBHT > 5000		MTBHT * TREM		- REMISE	
02	Avoir Client				AVOIR	- AVOIR	

DESIGNATION DU MODULE : B							
Niveau	Désignation élément	Condition	Entité	Calcul	données	Identificateur	Appel
01	Montant brut HT					MTBHT	
02	Montant ligne commande		PRODUIT	PU * QTC		+ MTLIHT	

II.121 Passage de la structuration conceptuelle à la structuration logique

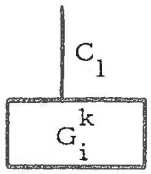
Au niveau conceptuel l'unité avec laquelle on raisonne est le module auquel on associe une structure conceptuelle de traitements qui est une hiérarchie arborescente de groupes conceptuels (inconditionnel, conditionnel, itératif, d'appel).

Au niveau logique l'unité sur laquelle on raisonne est l'unité opérationnelle de traitement à laquelle on associe une strucure logique de traitements qui est une hiérarchie de paragraphes de traitement (inconditionnel, conditionnel, itératif, d'appel).

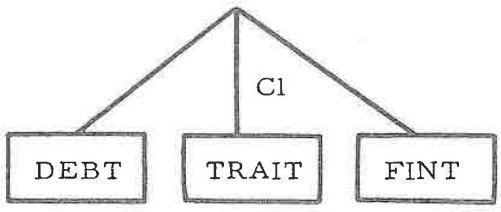
II.121.1 Correspondance entre groupes conceptuels et paragraphes logiques.

La correspondance s'établit entre les groupes de la structure conceptuelle et les paragraphes de traitement de la structure logique type par type :

. au groupe conceptuel conditionnel



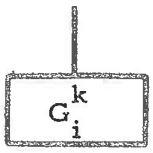
est associé



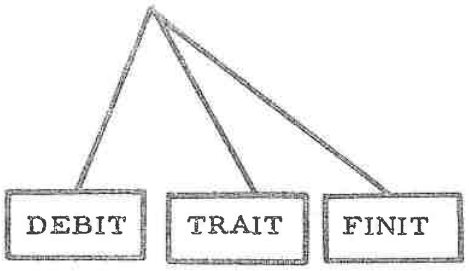
une structure conditionnelle de base

qui comporte un paragraphe conditionnel précédé d'un paragraphe inconditionnel d'initialisation, et aussi d'un paragraphe inconditionnel de terminaison. La description nécessite, en outre, que l'on explicite la condition du paragraphe conditionnel.

. au groupe conceptuel itératif



est associé



ENTITE — CRITERE-ITER

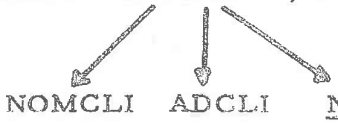
une structure itérative de base qui comporte un paragraphe itératif précédé d'un paragraphe inconditionnel d'initialisation et suivi d'un paragraphe inconditionnel de terminaison. La description nécessite qu'en outre on explicite l'indicatif conceptuel sur lequel se fait l'itération.

Remarquons qu'au cours de cette correspondance nous supposons résolu le passage de l'ENTITE du niveau conceptuel à l'INDICATIF du niveau logique. Ce problème relève, dans le processus de conception proposé dans REMORA, de l'aspect gestion des données (définition des objets, de leurs propriétés, de leur regroupement, . . .) traité par KRIEGUER (19) : en résumé, au niveau conceptuel,chaque entité (objet ou association d'objets) est décrite par l'ensemble de ses propriétés et repérée par un identifiant (au sens de CODD (45)) ; il lui est associé, au niveau logique, un type article (au sens de CABANES (44)), repéré par un indicatif. L'indicatif peut être l'identifiant :

exemple : l'objet CLIENT, au niveau conceptuel, a pour identifiant un numéro individuel attribué par l'entreprise : NCLI

CLIENT (NOMCLI, ADCCLI, <u>NCLI</u>)

le type article CLIENT , au niveau logique,



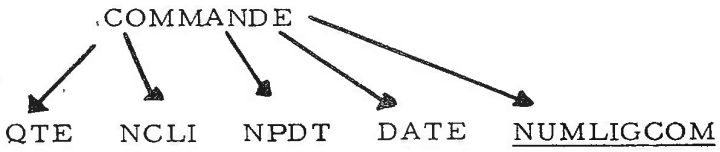
conserve NCLI comme indicatif (repère physique de l' ensemble des valeurs d'une occurrence du type article CLIENT).

- L'indicatif peut être distinct de l'identifiant :

exemple : l'association COMMANDE entre les objets CLIENT et PRODUIT du niveau conceptuel, a pour identifiant NCLI, NPDT, DATE

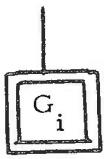
COMMANDE (QTE, NCLI, NPDT, DATE).

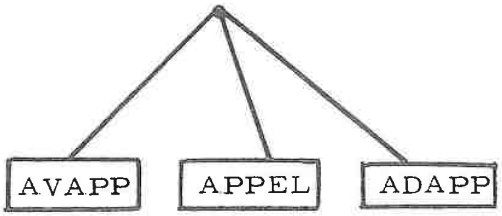
Le type article COMMANDE qui lui est associé au niveau logique,



pour des raisons d'économie prévisibles au niveau de l'implémentation, sera muni d'un indicatif spécifique NUMLIGCOM (numéro de ligne de commande).

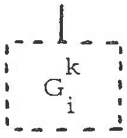
. Au groupe conceptuel appel





est associée une structure d'appel de base qui comporte un paragraphe d'appel encadré par deux paragrahes inconditionnels, l'un l'initialisation, l'autre de terminaison.

. Au groupe conceptuel inconditionnel



est associé

un paragraphe inconditionnel.

II.121.2 Passage de la structure conceptuelle à la structure logique

Le passage se fait en plusieurs étapes que nous allons différencier.

a) Expression de chaque module conceptuel en termes logiques

Par application des règles de correspondance que nous venons de présenter, chaque module conceptuel est exprimé en termes logiques. Deux causes appliquent les différences entre les deux expressions.

. Modification de la hiérarchie

On retrouve dans cette expression logique une hiérarchie de traitement proche de la hiérarchie du niveau conceptuel. En effet, dans la hiérarchie conceptuelle, les niveaux introduits par les groupes de nature inconditionnelle, itérative ou appel, se retrouvent intégralement dans la hiérarchie logique et à chacun de ces groupes correspond une structure de base équivalente (cf. supra).

En revanche, la prise en compte des groupes inconditionnels va introduire des différences entre la hiérarchie conceptuelle et la hiérarchie logique. Ainsi, disparaissent dans la hiérarchie logique les niveaux qui, dans la hiérarchie conceptuelle, sont introduits par des groupes inconditionnels. En effet, s'il est admissible que le concepteur, au niveau conceptuel imbrique des groupes dans des groupes inconditionnels, créant ainsi, des niveaux supplémentaires dans la hiérarchie, cela n'est plus permis dans la hiérarchie logique, car son paragraphe inconditionnel est par définition élémentaire.

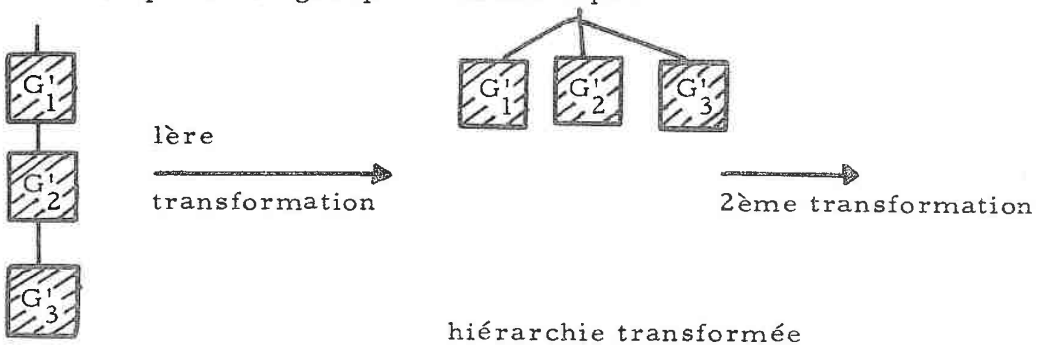
. Transformation des groupes en paragraphes

On a vu au II.121.1 les correspondances entre groupes et paragraphes. D'autres transformations interviennent aussi conduisant à confondre en un

seul paragraphe inconditionnel, des paragraphes inconditionnels consécutifs à un même niveau.

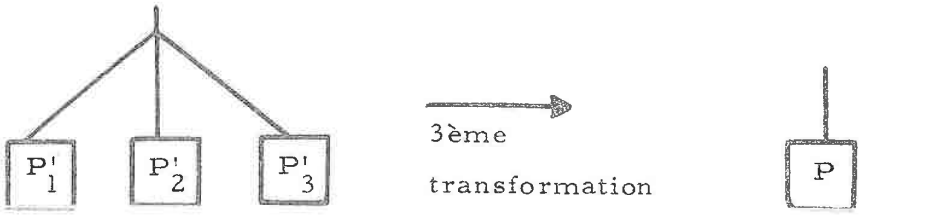
C'est cette règle qui conduit par exemple à définir des paragraphes INTER par confusion du paragraphe TERMINAISON et du paragraphe INITIALISATION de deux structures de base inconditionnelles. C'est cette même règle qui aboutit, une fois réalisées toutes les transformations, à faire correspondre un seul paragraphe inconditionnel à deux ou plusieurs groupes inconditionnels qui étaient soit imbriqués, soit juxtaposés à un même niveau :

cas ou deux ou plusieurs groupes sont imbriqués



hiérarchie de départ

hiérarchie transformée après élimination des niveaux introduits par des groupes inconditionnels



remplacement des groupes par des paragraphes

confusion en un seul paragraphe des trois paragraphes inconditionnels juxtaposés

Ce cas contient aussi le cas où la hiérarchie conceptuelle comporte deux ou plusieurs groupes inconditionnels juxtaposés de même niveau (dans ce cas la lère transformation est inutile).

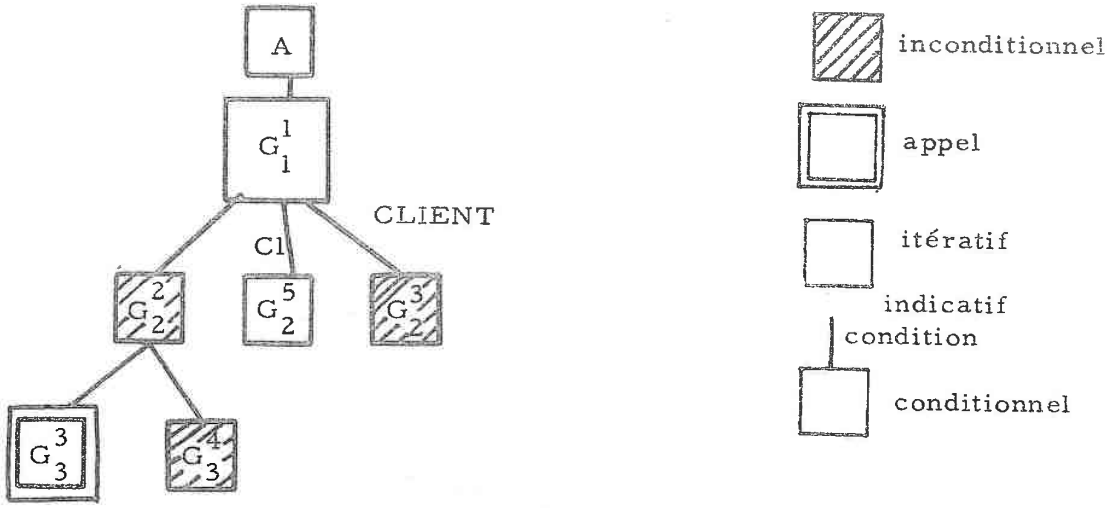
Le processus de transformation de l'expression de chaque module conceptuel est présenté dans l'exemple qui suit.

Exemple : Considérons la structure conceptuelle déduite du module A

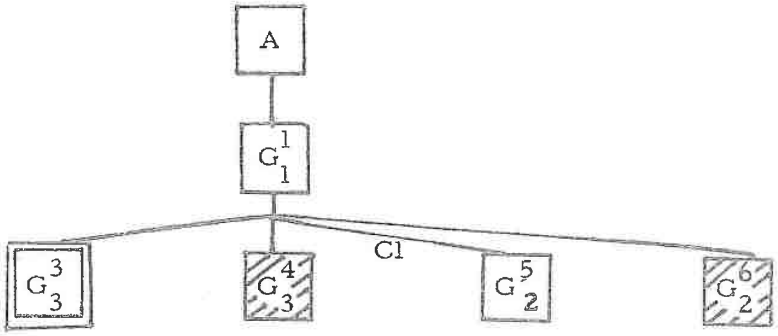
DESIGNATION DU MODULE : A

	Niveau	Désignation éléments	Condition	Entité		Appel	
G_1^1	0 1	Total facture d'un client		CLIENT			Itératif
G_2^2	0 2	Montant Brut TTC					Inconditionnel
G_3^3	0 3	Montant Brut HT				B	Appel
G_3^4	0 3	Montant TVA					Inconditionnel
G_2^5	0 2	Remise	MTBHT > 5000				Conditionnel
G_2^6	0 2	Avoir client					Inconditionnel

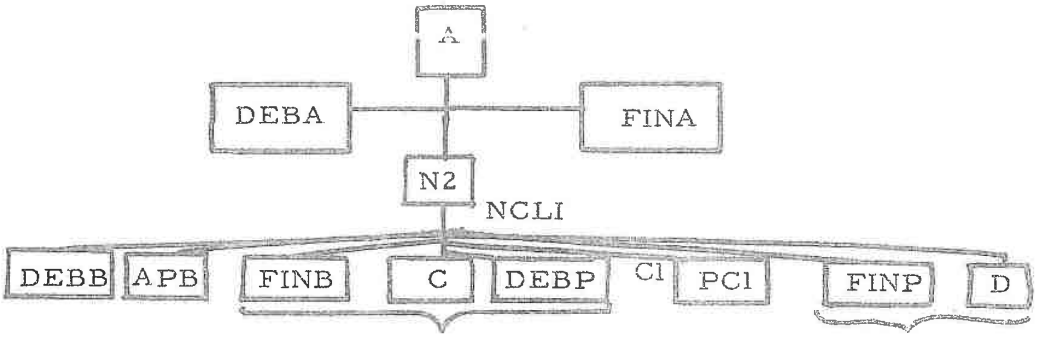
que nous représenterons par la hiérarchie suivante, en utilisant les symboles de représentation



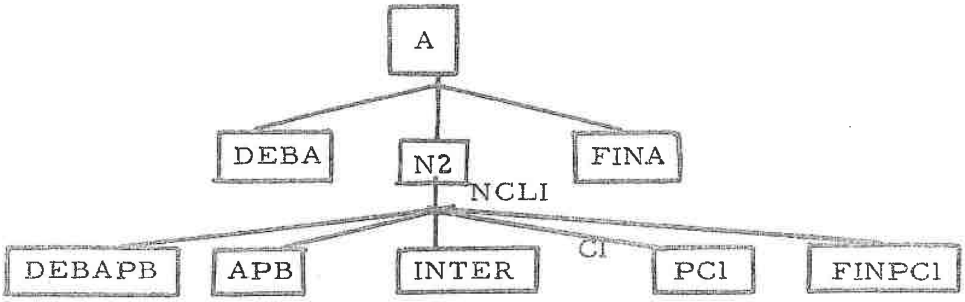
Un première transformation aboutit à supprimer dans cette hiérarchie les niveaux qui correspondent aux groupes inconditionnels



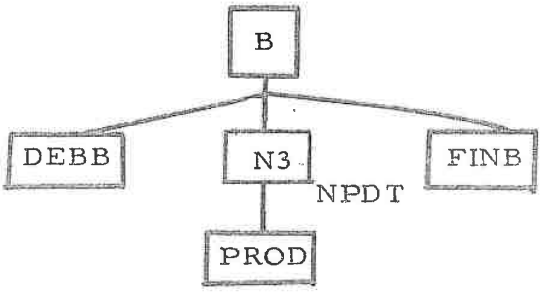
Une seconde transformation consiste à remplacer les groupes par les structures de base correspondantes



Une troisième transformation consiste à confondre les paragraphes inconditionnels juxtaposés à un même niveau (ici en regroupant les paragraphes de l'accolade)



De la même manière, on obtient pour le module conceptuel B de l'exemple la structure logique équivalente



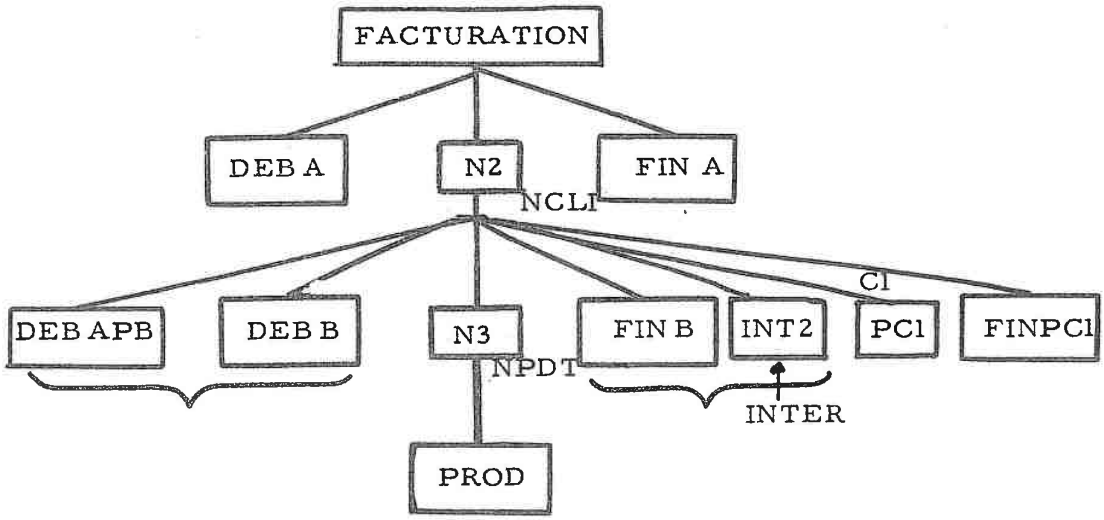
b) Construction des unités opérationnelles de traitement

On suppose résolu le passage de l'expression en modules à l'expression en unités opérationnelles (pour l'instant, c'est une opération que nous laissons totalement à la charge de concepteur. Dans un travail ultérieur seront étudiées les conditions d'éclatement et/ou de regroupement des modules en unités opérationnelles de traitement).

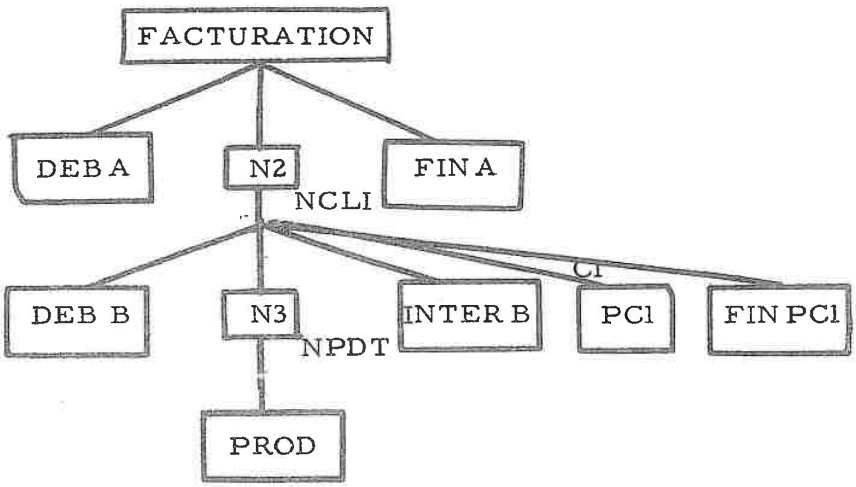
La structuration de l'UOT se fait par composition des modules traduits en termes logiques, en exploitant les liens entre modules qui sont représentés par les paragraphes d'appel. Dans les modules "appelant" les paragraphes d'appel sont remplacés par leur description logique développée.

II.121.3 Structure logique de l'Unité Opérationnelle

Exemple : Le remplacement dans la structure A du paragraphe d'appel APB de la figure par son développement, conduit à la structure logique de l'unité opérationnelle FACTURATION suivante :



dont l'expression logique définitive est :



II.122 Passage de la structure logique à la structure physique

Ce passage suppose que soit effectué le choix des organisations de données et s'appuie, dans le cas de notre travail, sur l'hypothèse d'une organisation des informations en fichiers séquentiels.

Rappelons que les structures logique et physique en correspondance reposent sur les mêmes concepts ; le passage de l'une à l'autre résulte de l'application des règles que d'aucuns ont appelé de cinématique [13], [27], [43],... et que nous résumons :

1 - Aux correspondances entre éléments des structures de départ et d'arrivée suivantes :

Structure logique	devient	Structure physique
Le paragraphe de traitement TO		Le paragraphe de traitement TO . Une condition d'arrêt du traitement itératif associé
L'usage de l'indicatif Ii		. Un paragraphe de lecture LCIi des fichiers séquentiels d'indicatif mineur Ii et de choix de la valeur de Ii à traiter . Un paragraphe de lecture LDi des fichiers à accés direct d'indicatif Ii . Un paragraphe d'initialisation (à blanc) des buffers associés aux fichiers résultats d'indicatif niveau Ii : MABIi . Un paragraphe d'écriture ECRI des fichiers résultats d'indicatif mineur Ii

L'ouverture de l'unité opérationnelle	. Un paragraphe d'ouverture OUVF de tous les fichiers . Un paragraphe de lecture initiale LCI0 de tous les fichiers séquentiels en entrée et de choix des valeurs initiales des indicatifs de l'UO . Un paragraphe de fermeture FERMF de tous les fichiers.

2 - Aux règles de placement des nouveaux paragraphes de traitement dans la structure logique de l'unité opérationnelle suivante :

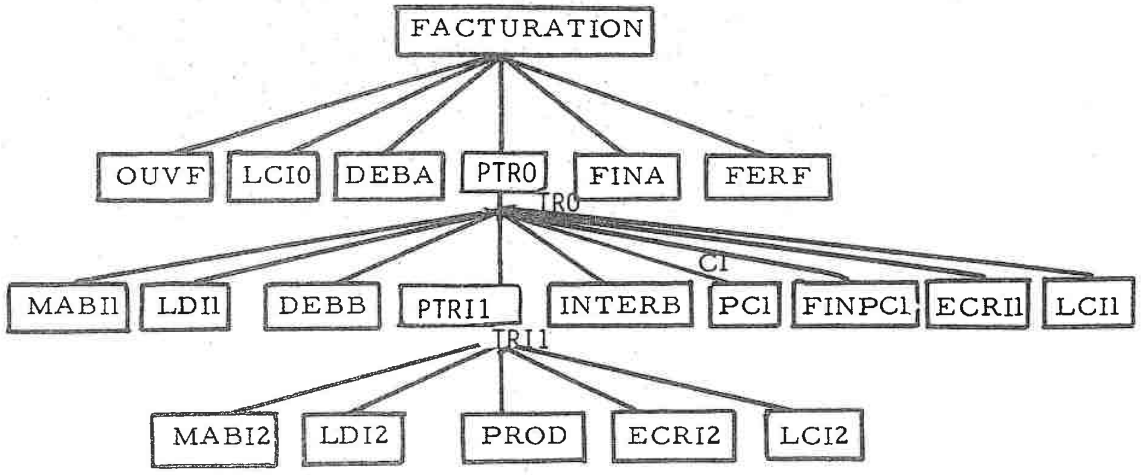
règle 1 : Les paragraphes OUVF et LCI0 se placent avant (dans notre choix de représentation à gauche) des paragraphes de traitement initial de l'UO et le paragraphe FERMF après (à droite) des paragr. de traitement final de l'UO.

règle 2 : Les paragraphes MABIi et LDIi se placent avant (à gauche) des paragraphes de traitements "initiaux" associés à l'indicatif Ii.

règle 3 : Les paragraphes ECRIi et LCIi se placent après (à droite) des paragraphes de traitements "finaux" associés à l'indicatif Ii.

règle 4 : La condition d'arrêt d'un traitement itératif associé à un indicatif Ii est : pour les indicatifs Ii de plus haut niveau, un test d'épuisement de tous les fichiers d'indicatif majeur Ii,
pour les indicatifs des niveaux inférieurs un test de rupture sur les indicatifs de niveaux supérieurs.
La condition d'arrêt vient alors prendre la place de l'indicatif correspondant.

Ces règles conduisent à la structure physique de l'unité opérationnelle FACTURATION suivante :



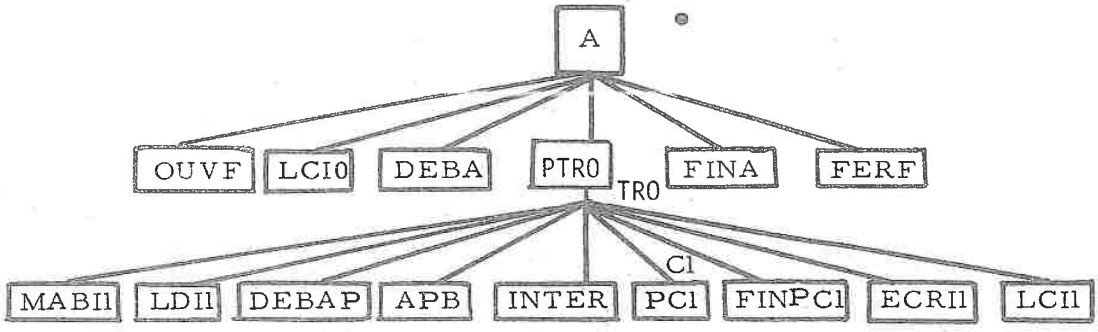
II.123 Conclusion : Justification de l'élaboration des structures logiques d'UOT au niveau logique.

Nous avons fait le choix de structurer l'UOT au niveau logique, par fusion des structures logiques associées aux modules conceptuels entrant dans la composition de l'UOT.

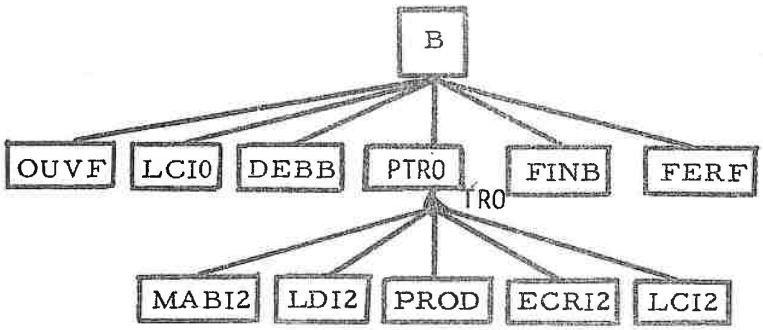
On peut se demander si cette structuration ne pourrait pas être reportée au niveau physique, après que les choix d'implémentation aient déjà été faits.

L'exemple de l'UOT de facturation va nous montrer que cette manière de faire est erronnée.

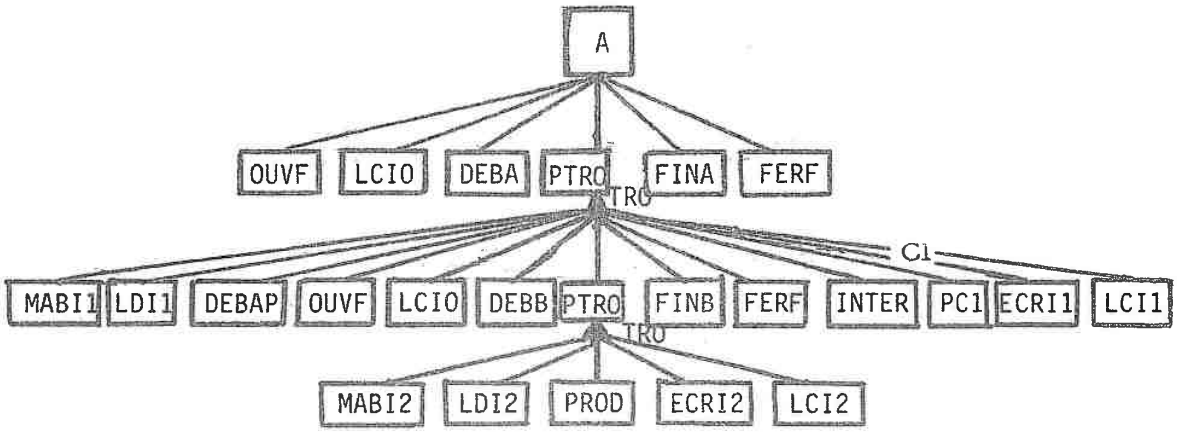
La structure physique de A est :



Celle de B est :



La règle de structuration proposée au paragraphe II.121.2 pour le niveau logique, appliquée au niveau physique conduit à la structure physique de l'UOT suivante :



Cette structure est manifestement fausse puisqu'elle comporte :

. deux paragraphes d'ouverture et de fermeture des fichiers OUVF et FERF
. deux paragraphes de lecture initiale des fichiers LCI0
. deux tests de fin de fichiers (TRO) mais aucun test de rupture de l'indicatif I1 (TRI1).

On constate que toutes les erreurs concernent les opérations liées à la cinématique des fichiers.

La structuration de l'UOT doit se faire au niveau logique.

II. 2 LA FONCTION DE TRANSFORMATION DES TEXTES

De manière symétrique à l'étude de la fonction de transformation des structures, nous dégageons trois niveaux de formulation des textes

le niveau conceptuel
le niveau logique
le niveau physique

et deux étapes de transformations du niveau conceptuel au niveau logique, du niveau logique au niveau physique.

II. 21 Les formulations successives des textes

II. 211 L'expression conceptuelle des traitements

Les textes conceptuels sont ceux des groupes conceptuels. Ils correspondent aux énoncés de problèmes fournis par le concepteur, dans le langage de conception, sur les descripteurs de traitements. Ce sont des définitions fournies de manière désordonnée dans un ordre déclaratif (13), (11). Nous en avons fait l'étude au chapitre I.

II. 212 L'expression logique des traitements

II. 212. 1 Définition

L'expression logique des traitements reflète la logique d'exécution du problème. C'est l'expression de l'algorithme déduit de l'énoncé conceptuel sous la forme d'une séquence d'ordres impérative (13), (11) c'est-à-dire reflétant l'ordre d'exécution du programme associé.

L'expression logique des traitements revêt un double aspect :

1 - Celui de la formulation du schéma logique, image de la structure logique qui décrit la hiérarchie des paragraphes.

2 - Celui de la formulation des paragraphes qui sont les feuilles de la structure qui définit les séquences d'ordres impératives.

II. 212. 2 Langage

L'un et l'autre de ces aspects sont formulés dans un langage standard comportant quatre types d'ordres :

α . des ordres inconditionnels sous la forme :

FAIRE eti

Cet ordre correspond à l'exécution de la séquence d'ordres associée au nom de paragraphe ou de module eti

β. des ordres conditionnels sous la forme :

SI condition FAIRE eti

Cet ordre correspond à l'exécution de la séquence d'ordres associée au nom de paragraphe ou de module eti si la condition est vérifiée

γ. des ordres itératifs sous la forme :

FAIRE eti JUSQU'A condition

Cet ordre correspond à l'exécution répetée de la séquence d'ordres associée au nom de paragraphe ou de module eti jusqu'à ce que la condition soit vérifiée.

δ. des ordres de calcul sous la forme

CALCULER des = expression arithmétique

Cet ordre définit la fonction de calcul de la désignation des d'une propriété d'objet du monde réel. L'expression arithmétique est formulée suivant les règles habituelles de priorité sur les opérateurs.

II.213 L'expression physique des traitements

II.213.1 Définition

L'expression physique des traitements reflète la logique d'exécution du problème dans des conditions technologiques de fonctionnement données. Elle correspond à l'expression d'une part de la logique inhérente au problème découverte et formulée à l'étape précédente, d'autre part de la logique d'accès aux données exprimée, compte tenu des choix d'implémentation des structures d'informations.

Elle revêt, comme la structure logique, un double aspect :

1 - Celui de la formulation du schéma physique image de la structure physique.

2 - Celui de la formulation des paragraphes qui sont les feuilles de la structure et qui sontsoit des paragraphes définis au niveau logique, soit des paragraphes spécifiques du traitement des fonctions technologiques introduits au niveau physique.

II.213.2 Langage

L'étude de la fonction de transformation des structures a fait apparaître que la structure logique s'obtient par

adjonction, à la structure logique, des paragraphes relatifs à la cinématique des fichiers.

Il suffit donc de complèter le langage précédemment défini pour qu'il permette la formulation des ordres d'accès aux informations, dans le cadre des fichiers séquentiels où nous nous sommes placés.

α. ordre d'ouverture des fichiers

OUVRIR
liste est une suite de désignations de fichiers ;

β . ordres d'accès aux articles des fichiers ;

LIRE des
ECRIRE des
SUPPRIMER des

des est la désignation de l'enregistrement du fichier concerné.

γ. ordre de fermeture des fichiers

FERMER liste

II.22 Les passages entre formulations successives de traitement

II.221 De l'expression conceptuelle à l'expression logique

Ce passage s'effectue en deux étapes :

1. Formuler le schéma logique
2. Formuler les textes des paragraphes feuilles.

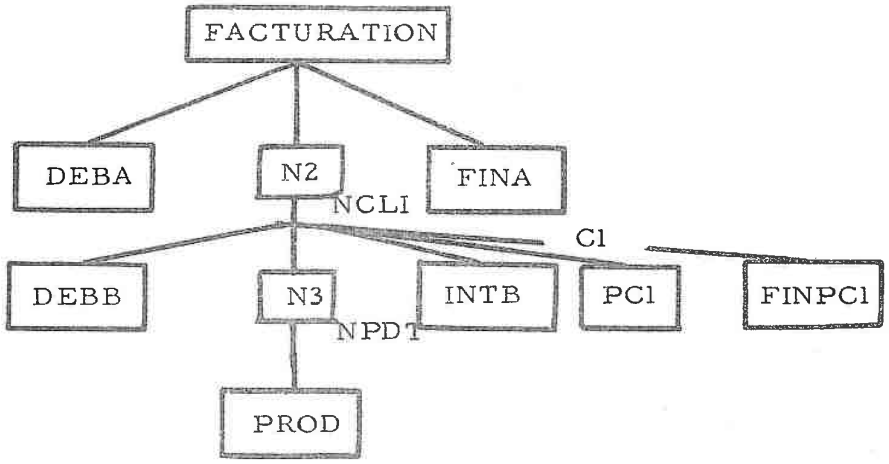
II.221.1 Formulation du schéma logique

Le passage de la structure logique au schéma logique s'effectue :

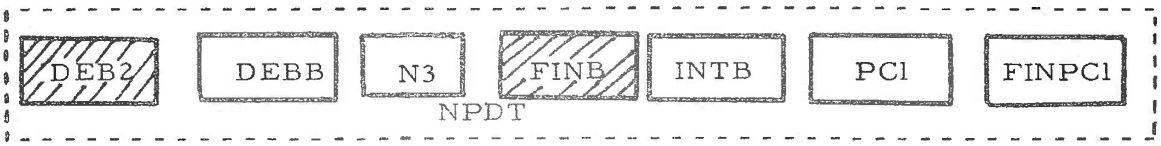
- par l'exploration de la structure, niveau par niveau, de gauche à droite
- par la désignation de chaque niveau
- par la reconnaissance du type de l'élément
- par la traduction au moyen de l'ordre correspondant du langage.

Exemple :

à la structure logique de l'unité FACTURATION élaborée au paragraphe précédent :



dans laquelle on désigne par N2 l'ensemble des paragraphes du niveau 2 :



N2

correspond le schéma logique suivant :

```
FACTURATION .    FAIRE    DEBA
                 FAIRE    N2      JUSQUA NCLI = 999
                 FAIRE    FINA
```

```
N2.     FAIRE    DEBB
        FAIRE    PROD    JUSQUA NPDT = 9999
                 INTB
        Si C1    FAIRE   PC1
        FAIRE    FINPC1.
```

II.221.2 Formulation des textes des paragraphes feuilles

La construction de la structure logique, associée à un module conceptuel, est telle que les paragraphes feuilles sont de nature inconditionnelle. D'autre part, l'ensemble des textes de ces paragraphes feuilles est en correspondance avec l'ensemble des définitions consécutives du module conceptuel.

Un module conceptuel est composé d'un ensemble de lignes de définitions de résultats, fournies de manière désordonnée, déclaratives dans l'ordre où l'analyse de ces résultats a été faite.

Chaque paragraphe logique inconditionnel est une séquence impérative d'ordres de calcul correspondant à l'ordre d'exécution du calcul.

Exprimer cet ensemble de séquences d'ordres c'est passer du mode déclaratif utilisé dans les modules conceptuels, au mode impératif du niveau logique, par un réordonnancement des définitions d'un module conceptuel, puis une affectation de ces définitions dans les paragraphes feuilles de la structure logique associée.

II.221.21 Le réordonnancement des définitions

Il s'effectue au moyen de deux piles ; celle des lignes accumulant les lignes de définitions d'un module conceptuel dans

l'ordre où elles sont décrites sur le descripteur de traitement ; celle des cumulateurs accumulant les désignations des résultats intermédiaires calculés au moyen de sommes algébriques.

Il s'appuie sur une exploration de l'arborescence des définitions, branche par branche, dans l'ordre où elles sont décrites sur les descripteur de traitement.

Le réordonnancement des définitions d'un module conceptuel s'opère alors en deux phases alternées :

1 Une phase d'empilement et de production d'ordres d'initialisation

L'exploration d'une branche de l'arborescence de haut en bas , de la racine jusqu'à la rencontre d'une feuille, provoque :

1.1 l'empilement des lignes successives

1.2 la production d'ordres d'initialisation des éléments calculés par une somme algébrique

1.3 l'empilement des désignations de ces éléments sur la pile des cumulateur.

2 Une phase de désempilement et de production d'ordres de calcul

L'exploration d'une branche de l'arborescence de bas en haut, de la feuille jusqu'à la rencontre d'une branche frère provoque :

2.1 le désempilement ligne à ligne

2.2 la production de l'ordre de calcul associé à la définition de l'élément résultat de la ligne

2.3 le désempilement de l'élément calculé sur la ligne dans la pile des cumulateurs s'il en est le sommet

2.4 la production d'un ordre de cumul avec le sommet de la pile des cumulateurs si l'élément résultat de la ligne est signé.

Connaissant d'une part la position relative des définitions réordonnées par rapport aux groupes logiques de la structure conceptuelle et

d'autre part la structure logique correspondante, les ordres de calcul associés aux définitions sont ventilés dans les paragraphes feuilles.

II.221.22 Exemple

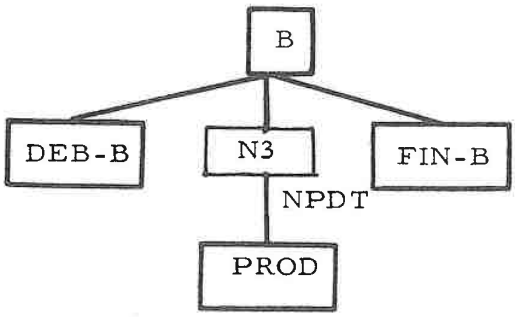
Le réordonnancement des définitions du module B

Niv.	Désignation	Condition	Entité	Calcul	Donnée	Identificateur	Appel
01	Montant brut HT					MBHT	
02	Montant ligne produit		PRODUIT	NB*PU		+ MPRD	

se déroule alors comme suit

Pile ligne	Pile "cumulateur"	Ordres générés
Empilement		
1. Montant brut HT	MBHT	INITIALISER MBHT A 0
.		
2. Montant ligne produit	MBHT	INITIALISER MBHT A 0
1 Montant brut HT		
Désempilement		
		AJOUTER MPRD A MBHT
		CALCULER MPRD = NB * QU
1 Montant brut HT	MBHT	INITIALISER MBHT A 0
		[fin]
		AJOUTER MPRD A MBHT
		CALCULER MPRD = NB * QU
		INITIALISER MBHT A 0.

Les textes logiques des paragraphes feuilles de la structure logique associée au module B



sont alors

DEB-B
INITIALISER MBHT A 0
[fin]

PROD
CALCULER MPRD = NB * PU
AJOUTER MPRD A MBHT
[fin]

FIN B
[fin]

II.222 De l'expression logique à l'expression physique

Le passage de l'expression logique des traitements à l'expression physique s'effectue, de manière analogue au passage étudié précédemment, entre le niveau conceptuel et le niveau logique, en deux étapes :

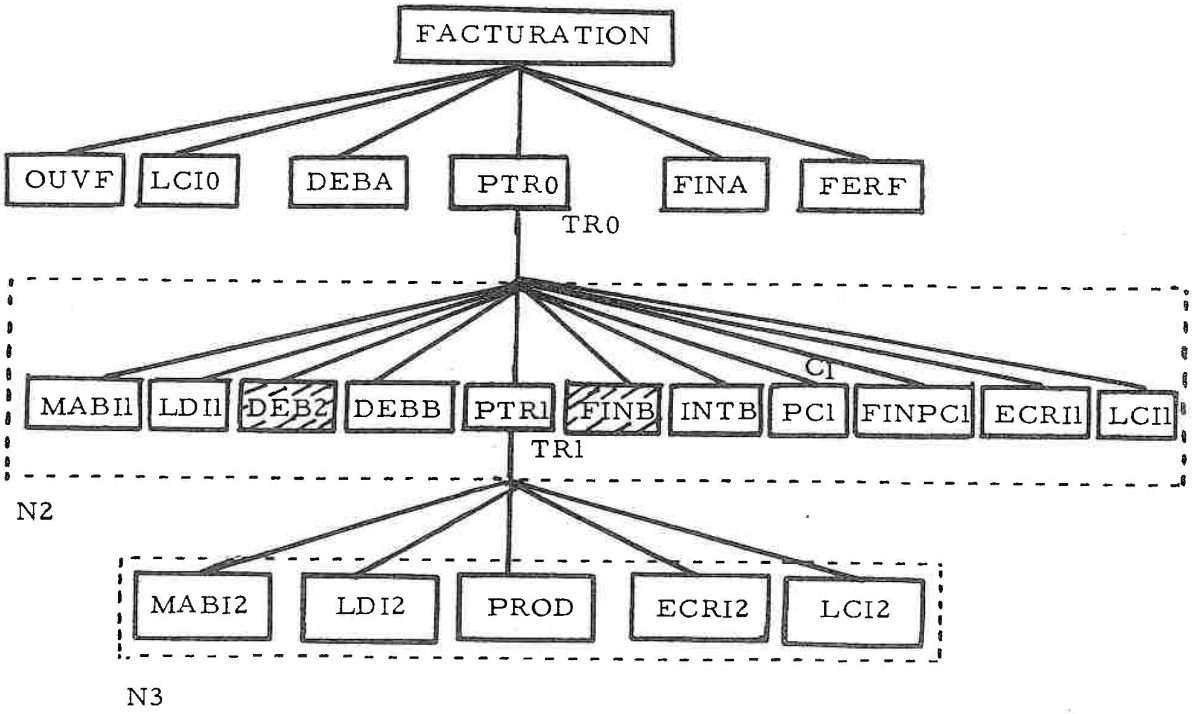
1. Formulation du schéma physique
2. Formulation des textes des paragraphes technologiques.

II.222.1 Formulation du schéma physique

Elle s'effectue suivant les mêmes règles qu'au niveau logique, par une exploration de la structure physique, niveau par niveau, de gauche à droite, par la reconnaissance du type de l'élément et sa traduction au moyen de l'ordre correspondant.

Exemple :

A la structure physique de l'unité FACTURATION



correspond le schéma physique.

FACTURATION	FAIRE OUVF
	FAIRE LCI0
	FAIRE DEBA
	FAIRE N2 JUSQUA TR0
	FAIRE FINA
	FAIRE FERF
N2	FAIRE MABI1
	FAIRE LDI1
	FAIRE DEBB
	FAIRE N3 JUSQUA TR1
	FAIRE INTB
	Si C1 FAIRE PC1

FAIRE FINPC1
FAIRE ECRI1
FAIRE LCI1

N3 FAIRE MABI2
FAIRE LDI2
FAIRE PROD
FAIRE ECRI2
FAIRE LCI2.

II.222.2 Textes des paragraphes

Le passage de la structure logique à la structure physique s'opère par adjonction de nouveaux pargraphes assurant les fonctions technologiques d'accès aux informations. Il s'agit de construire les séquences d'ordres de ces paragraphes en prenant en compte les paramètres relatifs au choix d'implémentation des structures de données.

Nous nous sommes inspirés à ce stade du travail de HERTSCHUH dont on trouvera en (18) une étude détaillée.

Nous l'avons étendu, d'une part en traitant la cinématique des fichiers en sortie de manière symétrique à celle des fichiers en entrée, d'autre part en acceptant les organisations de fichiers à accès direct.

II.23 Conclusion

L'étude de la fonction de transformations des textes nous a conduit à définir l'expression physique des traitements à partir de trois composantes :

. le schéma physique
. les procédures technologiques
. les algorithmes de calcul

que l'on peut formuler dans un langage standard tel que celui que nous avons proposé comportant trois types d'ordres :

. les ordres structurels permettant l'expression du schéma
. les ordres technologiques assurant l'expression des fonctions technologiques
. les ordres de calcul.

Le programme est l'image de l'expression physique des traitements dans le langage choisi. Il est indispensable que ce langage inclue le langage standard, si l'on veut aboutir à une programmation structurée.

II. 3 LA REALISATION DES PROGRAMMES

L'étape de conception a conduit à une expression physique des traitements telle que celle que nous venons de présenter.

L'étape de réalisation conduit à la production du programme (Rappelons que par programme nous entendons "ensemble d'instructions exécutables").

Dans notre hypothèse de travail, le passage de l'expression physique des traitements au programme est une opération de traduction d'un langage (le langage standard) dans un autre (le langage de programmation choisi).

Elle est immédiate lorsque le langage de programmation comporte les ordres équivalents aux ordres du langage standard.

Ainsi, dans le cas de COBOL, où les correspondances s'établissent suivant le tableau ci-après

Langage standard	COBOL
FAIRE eti	PERFORM eti
FAIRE eti JUSQUA condition	PERFORM eti UNTIL condition
si condition FAIRE eti	IF condition PERFORM eti
CALCULER des = expression arith.	COMPUTE des = expression arith.
INITIALISER des A val	MOVE val TO des
AJOUTER desl A des2	ADD desl TO des2
RETRANCHER desl De des2	SUBTRACT des2 FROM desl
OUVRIR liste	OPEN liste
FERMER liste	CLOSE liste
LIRE des	READ des AT END PERFORM eti
ECRIRE des }	WRITE des
SUPPRIMER des }	

L'arbre programmatique, selon l'expression de BERTINI et TALLINEAU dans "COBOL STRUCTURE", associée au schéma physique de l'exemple précédent est :

```
FACTURATION  PERFORM OUVF
             PERFORM LCI0
             PERFORM DEBA
             PERFORM N2 UNTIL   NCLI = 999
             PERFORM FINA
             PERFORM FERF

N2           PERFORM MABI1
             PERFORM LDI1
             PERFORM DEBB
             PERFORM N3  UNTIL  NPDT = 9999
             PERFORM INTB
```

```
                    IF MTBHT > 5000  PERFORM  PC1
                    PERFORM  FINPC1
                    PERFORM  LCI1

N3                  PERFORM  MABI2
                    PERFORM  LDI2
                    PERFORM  PROD
                    PERFORM  ECRI2
                    PERFORM  LCI2.
```

De la même manière tous les paragraphes cités sont traduits par l'intermédiaire du tableau de correspondance.

Ainsi, les paragraphes DEB-B et PROD générés au niveau logique au paragraphe II. 221. 22 deviennent :

```
DEB-B                       PROD
MOVE 0 TO MBHT.             COMPUTE MPRD = NB * PU
                            ADD MPRD TO MBHT.
```

CHAPITRE III

LE GENERATEUR DE REMORA

Le chapitre précédent a été consacré à l'étude fonctionnelle d'un générateur de traitements ; nous nous interesserons dans ce chapitre à l'étude organique du générateur de REMORA.

Cet outil assure les deux fonctions que nous avons précédemment décrites : de transformation des structures et de transformation des textes qui permettent de passer d'un ensemble de modules conceptuels à un ensemble de programmes.

Nous étudierons dans ce chapitre, l'organisation du générateur et décrirons les algorithmes qu'il exécute. Nous concluerons par un exemple de démonstration.

III.1 L'ORGANISATION DU GENERATEUR

Le générateur assure simultanément les deux fonctions de transformation des structures et des textes grace à deux modules

. le générateur primaire

. le générateur secondaire

qui interviennent en séquence dans un découpage horizontal du procédé de génération : le générateur primaire assure les transformations relatives aux modules conceptuels, le générateur secondaire celles qui concernent les unités opérationnelles suivant le schéma général :

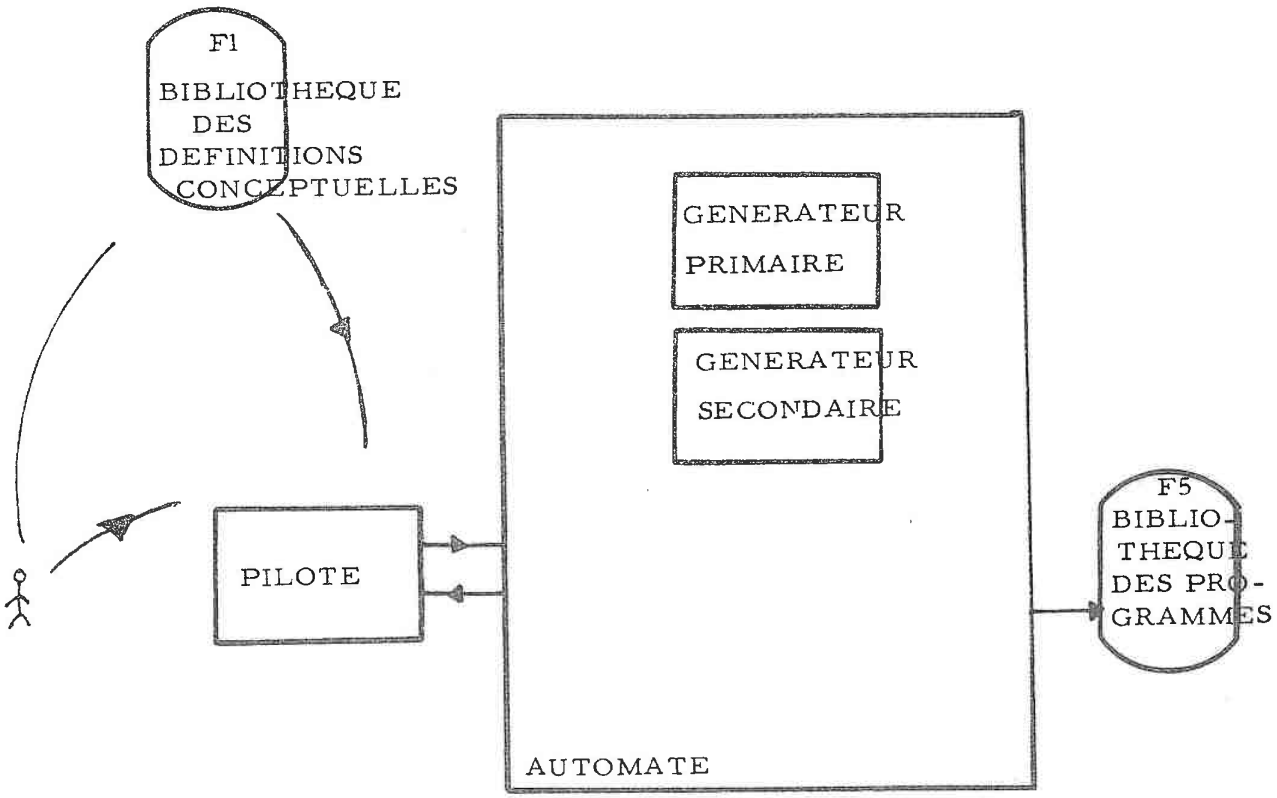


Figure 1 : Fonctionnement général du générateur

III.11 Le générateur primaire

Il travaille sur les modules conceptuels stockés dans la bibliothèque des définitions conceptuelles après qu'ils aient subi d'une part les contrôles syntaxiques, de cohérence et d'intégration prévus dans le processus de conception de REMORA et qui sont étudiés dans la thèse de THIERY (37), d'autre part certaines transformations liées à la définition des unités opérationnelles actuellement à l'étude (D. BANON).

Le générateur primaire assure le passage de l'expression conceptuelle des traitements d'un module conceptuel à l'expression logique équivalente.

Les entrées sont donc les modules conceptuels de la bibliothèque des définitions conceptuelles. Les sorties sont les expressions logiques sous la forme, d'une part des descriptions des structures logiques dans une forme interne du langage standard stockées dans la bibliothèque des structures, d'autre part des textes des paragraphes dans la forme interne du langage standard, également stockés dans une bibliothèque (figure 2) :

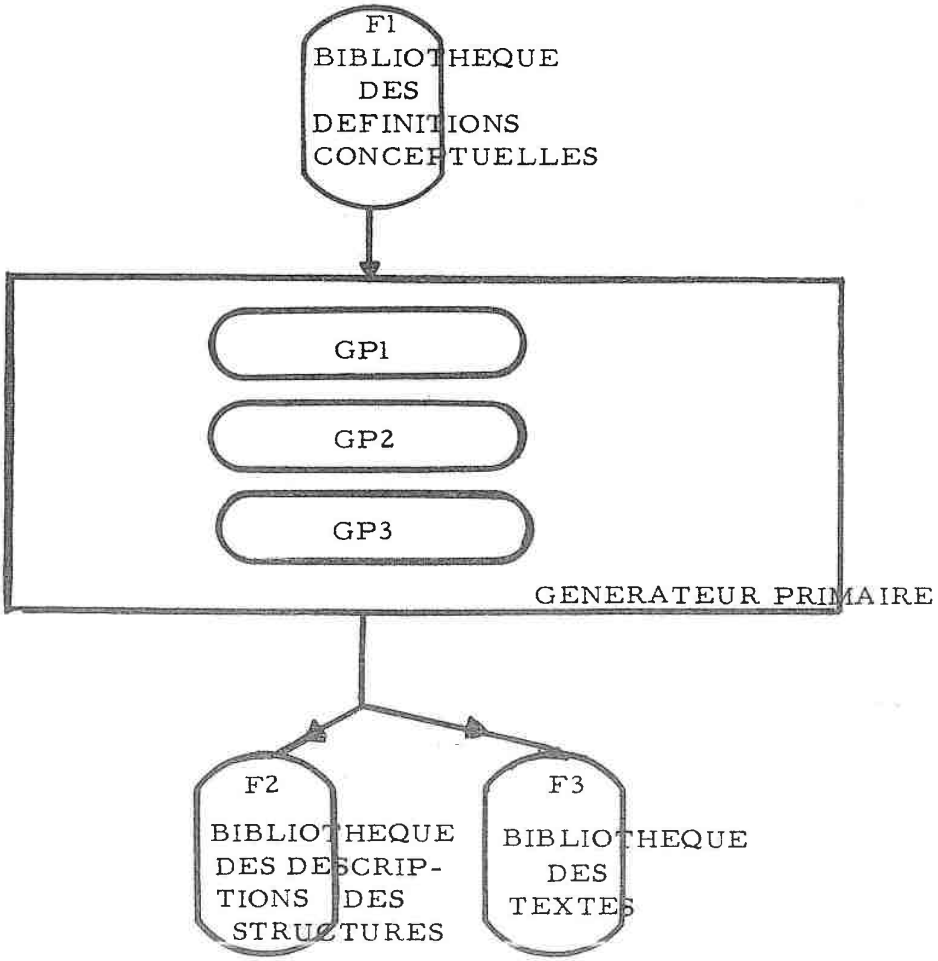


Figure 2 : Fonctionnement du générateur primaire.

Le générateur primaire est composé de trois modules GP1, GP2, GP3 (figure 2).

III.111 Le module GP1 du générateur primaire

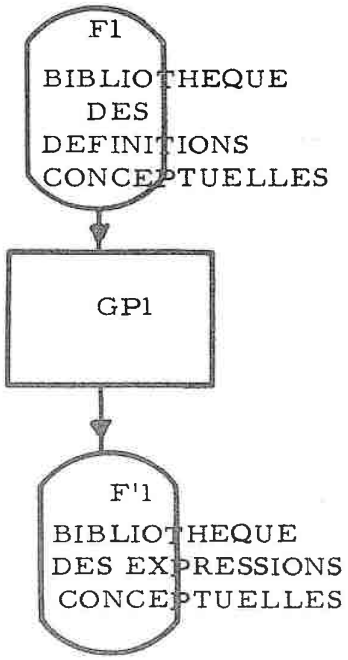
Le module GP1 permet le passage à une expression conceptuelle structurée. Pour un module conceptuel donné :

1 . il effectue l'analyse

2 . il assure la reconnaissance des groupes conceptuels munis de leur type (conditionnel - inconditionnel - itératif - appel)

3 . il identifie les opérations de calcul des lignes de définition (pour préparer le travail du module GP2)

4 . il construit l'expression conceptuelle des traitements structurée en groupes et la stocke dans une bibliothèque suivant le schéma :



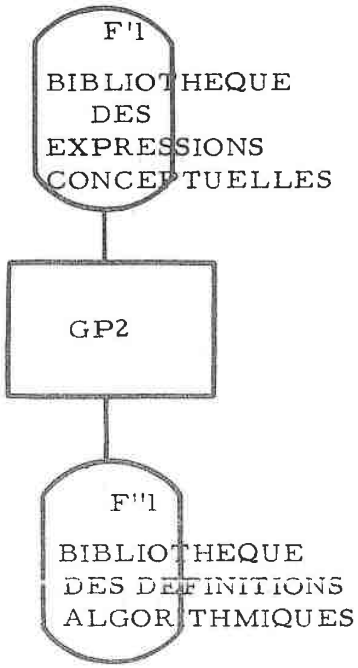
III.112 Le module GP2 du générateur primaire

Le module GP2 assure le passage de l'expression conceptuelle de tout module entrant dans la composition de l'unité opérationnelle à l'expression logique équivalente.

Pour y parvenir :

1. il analyse l'expression conceptuelle concernée stockée par le module GP1 dans la bibliothèque des expressions conceptuelles
2. il établit la correspondance ente les groupes conceptuels et les paragraphes logiques
3. il définit la structure logique
4. il réordonne les lignes de définition pour assurer le passage du mode déclaratif au mode impératif
5. il exprime le résultat de l'analyse dans une forme interne stockée dans la bibliothèque des définitions algorithmiques,

soit schématiquement :



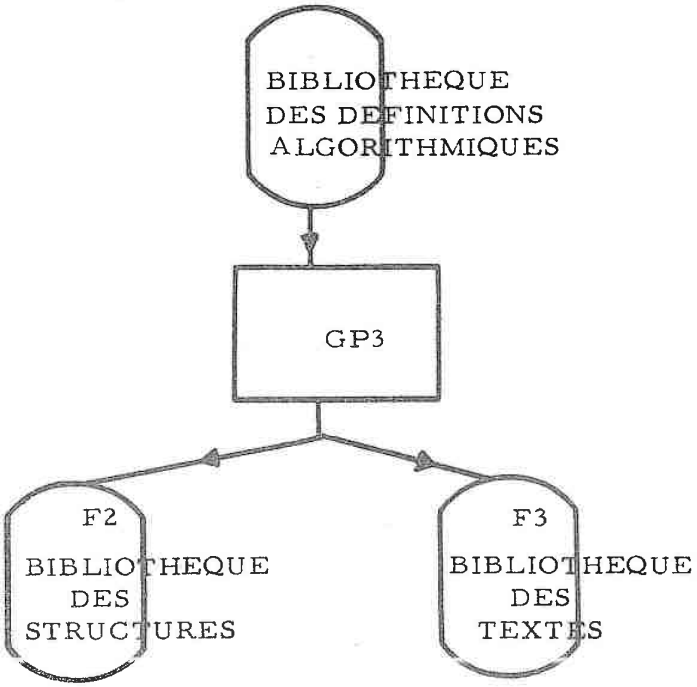
III.113 Le module GP3 du générateur primaire

Le module GP3 traduit le résultat du module GP2, dans la forme standard proposée au chapitre précédent pour l'expression logique des traitements.

Il produit en fait deux résultats :

. les textes logiques des paragraphes feuilles dans la forme interne du langage standard, ce qui facilitera son traitement par l'interface du langage de programmation choisi. (Bibliothèque des textes).

· une expression de la structure logique dans une forme interne du langage standard proposé au chapitre précédent. (Bibliothèque des structures).



L'éditeur peut à la demande, à partir des expressions combinées de la structure et des textes de paragraphes produire une expression logique des traitements identique à celle que nous avons écrite au chapitre II.

III.12 Le générateur secondaire

Le générateur secondaire travaille au niveau des unités opérationnelles pour lesquelles il va produire les programmes associés.

Le générateur secondaire se charge de prendre en compte les paramètres relatifs aux choix du niveau logique (composition de l'UO, correspondance identifiants, indicatifs) pour définir l'expression logique des traitements associée, à partir des expressions logiques de chacune des composantes produites par le générateur primaire, puis de prendre en compte les paramètres des choix d'implémentation physique pour passer de l'expression logique de l'UO à son expression physique et enfin le choix du langage de programmation pour passer au programme.

Les entrées sont donc les deux bibliothèques élaborées par le module 3 du générateur primaire : la bibliothèque des structures logiques, la bilbliothèque des expressions logiques et celle des paramètres des niveaux logique et physique. Les sorties sont les expressions physiques des traitements des UO (figure 3).

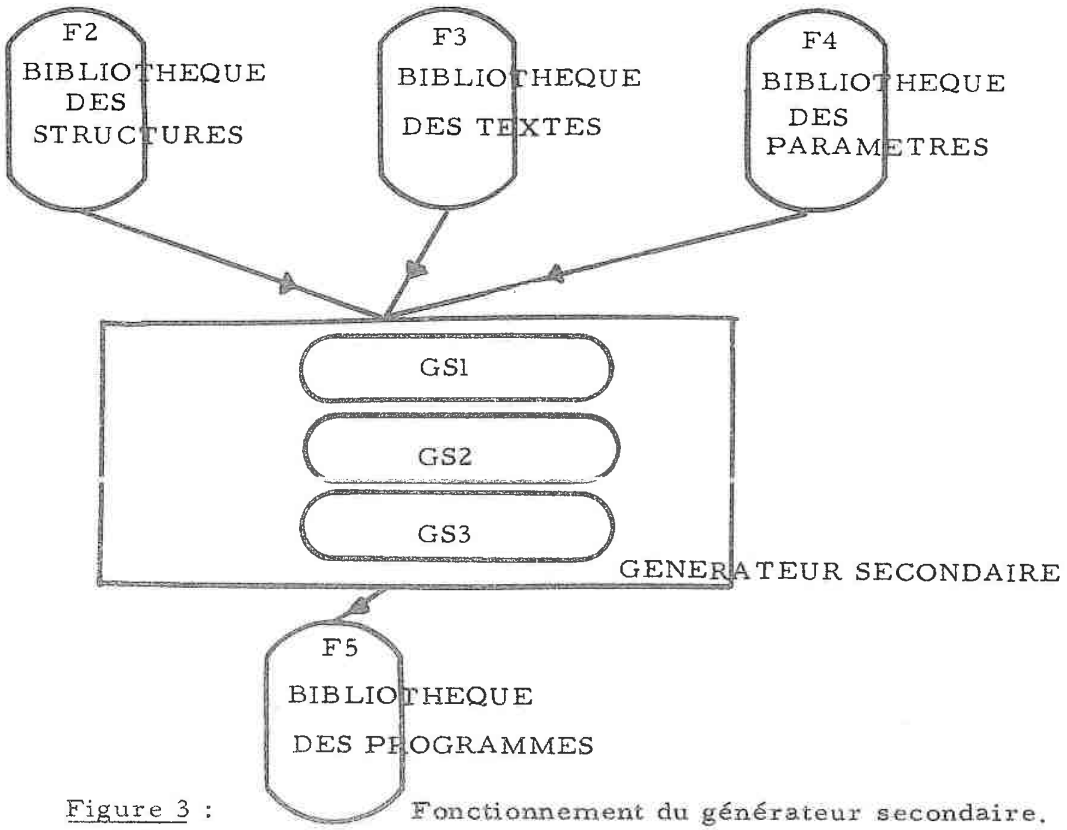


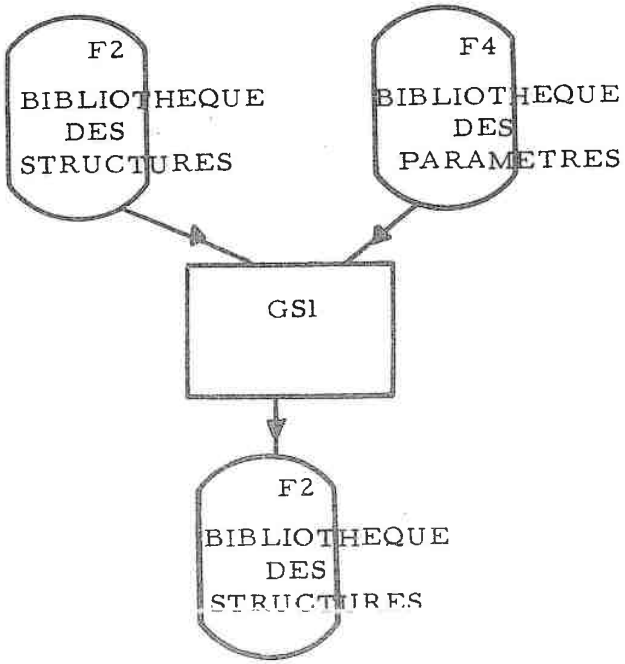
Figure 3 : Fonctionnement du générateur secondaire.

III.121 Le module GS1 du générateur secondaire

Le module GS1 produit l'expression logique des traitements de l'unité opérationnelle.

Pour cela

1. il utilise la définition de l'unité opérationnelle fournie par la bibliothèque des paramètres (composition de l'UO à partir d'un module conceptuel principal et de modules fils conceptuels).
2. il opère la fusion des structures logiques intermédiaires qu'il trouve dans la bibliothèque correspondante élaborée par GP3 et construit la structure logique de l'UO qu'il stocke dans la même bibliothèque.



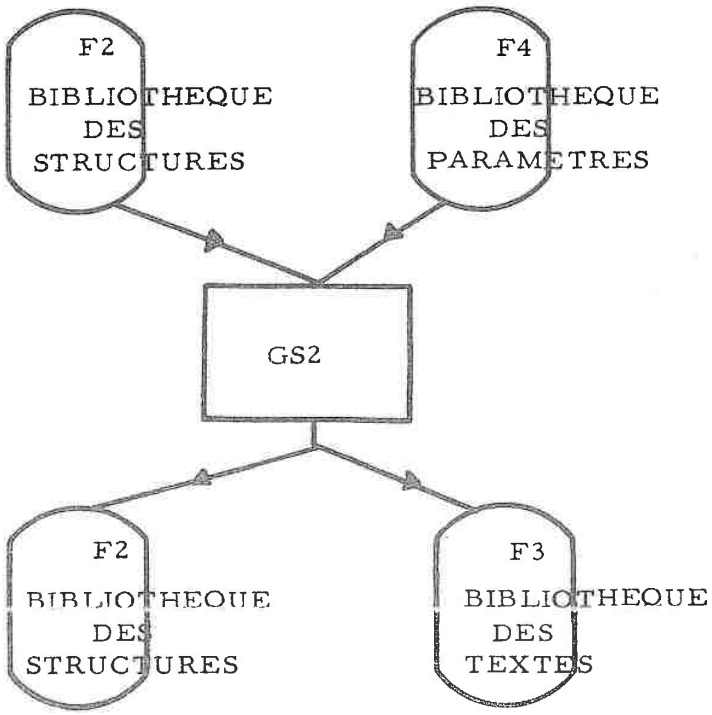
Comme précédemment l'EDITEUR peut, à la demande, à partir de la structure logique de l'UO et des textes des paragraphes correspondants produire l'expression logique des traitements de l'UO dans le langage standard.

III.122 Le module GS2 du générateur secondaire

Le module GS2 produit l'expression physique des traitements de l'UO.

A partir des paramètres de choix d'implémentation des structures de données stockées dans la bibliothèque des paramètres et de la structure logique de l'UO (bibliothèque des structures), le module GS2 :

1. construit la structure physique de l'UO décrite dans la forme interne du langage standard et la stocke dans la bibliothèque des structures
2. génère les textes des paragraphes de cinématique et les stocke dans la bibliothèque des textes .



L'EDITEUR utilisant ces deux bibliothèques, peut produire l'expression physique des traitements de l'UO dans le langage standard.

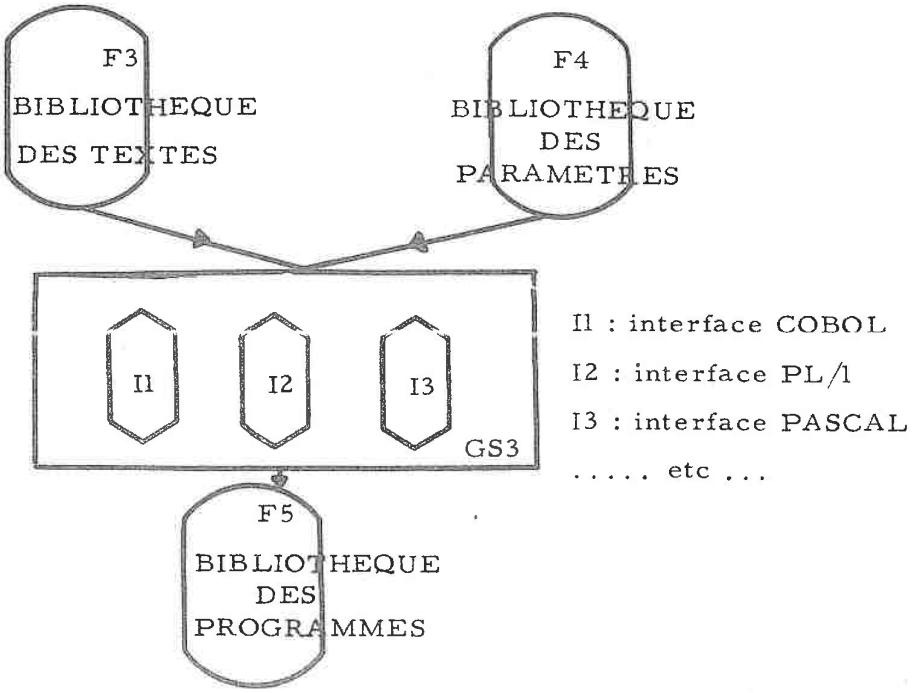
III.123 Le module GS3 du générateur secondaire

Le module GS3 produit le programme associé à l'unité opérationnelle.

Ce module est un interface qui assure le passage des textes établis dans le langage standard aux textes formulés dans le langage de programmation choisi.

A partir de l'expression physique des traitements de l'UO (la structure dans la bibliothèque des structures, les textes des paragraphes dans la bibliothèque des textes), du choix du langage de programmation (bibliothèque des paramètres) , le module GS3 :

1. sélectionne l'interface du langage choisi
2. opère la traduction des textes en langage standard
3. stocke le résultat dans la bibliothèque des programmes (programmes et pris ici dans le sens de partie exécutive d'un programme)



III. 2 LES ALGORITHMES DES DIFFERENTS MODULES DU GENERATEUR

Pour présenter les algorithmes des différents modules du générateur, nous avons choisi d'une part une description modulaire et d'autre part, lorsque différents cas se présentent, une description prédicative de la forme :

$$\begin{aligned} &. C_1 \Longrightarrow I_1 \\ &. C_2 \Longrightarrow I_2 \\ &\vdots \\ &. C_n \Longrightarrow I_n \end{aligned}$$

où C_1, C_2, ..., C_n, conditions s'excluant par définition, sont évaluées collatérallement.

Le I_j correspondant au C_j vrai est exécuté.

III. 21 Le générateur primaire

III. 211 Le module GP1

III. 2111 Format des lignes de définition de la bibliothèque des définitions conceptuelles

Le format de chaque ligne de la bibliothèque F1 où sont stockées les définitions, est le suivant :

L(1)	L(2)	L(3)	L(4)	L(5)	L(6)	L(7)	L(8)	L(9)	L(10)
niveau	Condition	nature occurrence	occurrence	nature calcul	Calcul	Donnée	signe	Identificateur	Appel

avec comme conventions :

. Un module est introduit par une ligne de définition de niveau 0 et clos par une ligne de définition de niveau 99.

. Nature occurrence :

0 aucune occurrence n'est définie

1 si (4) contient une entité

2 si (4) contient un indicatif de boucle avec indice, valeur initiale, valeur finale, pas

3 si (4) contient un indicatif de boucle avec indice, valeur intiale, valeur finale

4 si (4) contient un indicatif de boucle avec indice, valeur initiale

5 si (4) contient un indicatif de boucle avec indice.

. Nature calcul : spécifie le mode de calcul du résultat dont l'identificateur est inscrit dans (0).

0 si (9) n'est pas obtenu par calcul dans le module

1 si (9) est le résultat d'une expression arithmétique contenue dans (6)

2 si (9) est obtenu par transfert d'une valeur contenue dans (6)

3 si (9) est le résultat d'une somme algébrique

III. 2112 Format des lignes de la bibliothèque des expressions conceptuelles

Le début et la fin d'un groupe conceptuel sont repérés par une marque. Chaque ligne d'un groupe conceptuel est transformée en une expression à trois composantes :

vecteur structure - vecteur action - paramètres

de format suivant :

							(1)	(2)	(3)	(4)	(5)	(6)
VL1	VL2	VL3	VL4	VAI	VAD	VRE	Condition	Occurrence	Calcul	Donnée	Identificateur	Appel
vecteur structure					vecteur action		paramètres : PA					

et avec les conventions suivantes :

+ VL1 spécifie le rôle de la ligne résultat :

1 marque de fin d'un module conceptuel

0 marque de fin d'un groupe conceptuel

1 marque de début d'un groupe conceptuel

2 marque de début d'un module conceptuel (racine de l'arborescence)

Dans le cas où la ligne résultat introduit un groupe conceptuel (VL1 = 1) :

+ VL2 et VAI donnent les éléments caractéristiques du type du groupe conceptuel introduit

VL2 = 0 aucune condition et aucune occurrence ne sont associées au groupe conceptuel

VL2 = 1 une condition est associée au groupe conceptuel

VL2 = 2 un indicatif de boucle* est associé au groupe conceptuel

VL2 = 3 une entité est associée au groupe conceptuel

VAI = 1 un appel de descripteur fils est associé au groupe conceptuel

* VL3 et VL4 spécifie alors le critère d'itération.

+ VAD et VRE caractérisent les actions définies sur la ligne.

VAD = 1 le résultat s'obtient par calcul d'une expression arithmétique

VAD = 2 le résultat s'obtient par transfert de valeur

VAD = 3 le résultat s'obtient par calcul d'une somme algébrique

VRE = 1 l'identificateur est un élément soustractif d'une somme algébrique

VRE = 2 l'identificateur est un élément additif d'une somme algébrique.

III. 2113 L'algorithme

Niveau 1

Pour chaque ligne de définition d'un module conceptuel faire

- L(1) ligne étudiée = 0 ⟹ Initialisation du processus
  - . VL1 = 2.
  - . PA (7) = nom du module.
- L(1) ligne étudiée > 0 et < 99 ⟹ Analyse de la ligne de définition
  - . Analyse des éléments typologiques du G.C. introduit.
  - . Analyse des actions définies sur la ligne.
  - . Détermination des G.C. clos par ligne.
- L(1) ligne étudiée = 99 ⟹ Cloture du processus
  - . VL1 = -1.

Niveau 2

2.1 Analyse des éléments typologiques du groupe conceptuel introduit

Co chaque ligne de définition introduit un groupe conceptuel dont le type est fonction du contenu des zones "condition", "occurrence" et "appel" de cette ligne OC

2.1.1 VL1 = 1

2.1.2 Détermination de VL2, VL3 et VL4

. L(2) = $\mathcal{A}$ et L(3) = 0 ⟹ VL2 = VL3 = VL4 = 0

. L(2) ¬ = $\mathcal{A}$ et L(3) = 0 ⟹ { VL2 = 1 ; VL3 = VL4 = 0.

. L(2) = $\mathcal{A}$ et L(3) > 1 ⟹ { VL2 = 2 ; VL3 = 0 ; VL4 = L(3).

. L(2) $\neg$ = Λ et L(3) > 1 $\Longrightarrow$ VL2 = 2 ;
VL3 = 1 ;
VL4 = L(3).

. L(2) = Λ et L(3) = 1 $\Longrightarrow$ VL2 = 3 ;
VL3 = VL4 = 0.

2.1.3 Détermination de VAI

. L(10) = Λ $\Longrightarrow$ VAI = 0.

. L(10) $\neg$ = Λ $\Longrightarrow$ VAI = 1.

Niveau 2

2.2 Analyse des actions définies sur la ligne.

Co cette analyse est menée à partir de l'étude des zones "nature calcul" - L (5) - et signe - L (8) - de la ligne de définition oc

2.2.1 VAD = L (5)

2.2.2 Détermination de VRE

. L (8) = Λ $\Longrightarrow$ VRE = 0.

. L (8) = '-' $\Longrightarrow$ VRE = 1.

. L (8) = '+' $\Longrightarrow$ VRE = 2.

Niveau 2

2.3 Détermination des groupes conceptuels clos par la ligne de définition

Co Pour chaque conceptuel dont la ligne de définition constitue la ligne de plus haut niveau, on construit une ligne "fin de groupe conceptuel" OC

2.3.1 Détermination du nombre de groupes conceptuels clos

. L (1) ligne suivante = 99 $\Longrightarrow$ NBCLOS = L(1) ligne étudiée.

. L (1) ligne suivante > L (1) ligne étudiée $\Longrightarrow$ NBCLOS = 0.

. L((1) ligne suivante $\leq$ L (1) ligne étudiée

= $\Longrightarrow$ NBCLOS = L (1) ligne étudiée
- L (1) ligne suivante
+ 1.

2.3.2 Construction des lignes "fin de groupe conceptuel"

Pour I allant de 1 à NBCLOS pas 1 faire VL1 = 0.

III. 212 Le module GP2

III. 2121 Format des lignes de la bibliothèque de définitions algorithmiques (F''_1)

Chaque expression logique en correspondance avec un module conceptuel est une suite d'actions de la forme :

A (1)	A (2)	
Action	Spécification	paramètres

que l'on peut classer en deux catégories :

Les actions à entreprendre au niveau des structures :

ACTION = 1	SPECIFICATION = 1	Début d'un module conceptuel
	SPECIFICATION = 2	Fin d'un module conceptuel.
ACTION = 2	SPECIFICATION = 1	ouverture d'un traitement conditionnel
	SPECIFICATION = 2	ouverture d'un traitement itératif associé à un indicatif de boucle
	SPECIFICATION = 3	ouverture d'un traitement itératif associé à une entité
ACTION = 3	SPECIFICATION = 1	fermeture d'un traitement conditionnel
	SPECIFICATION = 2	fermeture d'un traitement itératif associé à un indicatif de boucle
	SPECIFICATION = 3	fermeture d'un traitement itératif associé à une entité
ACTION = 4	SPECIFICATION = 1	Appel de "module fils"

Les actions de type arithmétiques :

ACTION = 5	SPECIFICATION = 1	mise à zéro d'un "cumulateur"
	SPECIFICATION = 2	transfert de valeur
	SPECIFICATION = 3	cumul additif
	SPECIFICATION = 4	cumul soustractif
	SPECIFICATION = 5	calcul d'expression arithmétique.

III. 2122 L'algorithme

Niveau 1

Pour chaque ligne du texte analysé d'un module conceptuel faire

. VL1 = 2 (début d'un module conceptuel) ⟹ Initialisation du processus

. VL1 = 1 (début d'un groupe conceptuel) ⟹ Exploration descendante du texte

- Examen du groupe conceptuel introduit.
- Ordonnancement "descendant" des actions
- Empiler la ligne étudiée cas dans pile LIGNE.

. VL1 = 0 (fin d'un groupe conceptuel) ⟹ Exploration "ascendante" du texte

- Désempiler la ligne étudiée de la pile LIGNE
- Ordonnancement descendant des "actions"
- Examen du groupe conceptuel qui vient d'être exploré

. VL1 = -1 (fin d'un module conceptuel) ⟹ Clôture du Processus
A(1) = 1
A(2) = 2

Niveau 2

2.1 Initialisation processus

2.1.1 Générer une action début d'un module conceptuel :
- A (1) = 1 ;
- A (2) = 1 ;
- Paramètre = désignation du module

2.1.2 Initialiser les piles LIGNE et CUM

Niveau 2

2.2 Examen du groupe conceptuel introduit

2.2.1 Recherche du type du groupe conceptuel

. VL2 = 1 ⟹ { A (1) = 2
A (2) = 1
Paramètre = condition .

. VL2 = 2 ⟹ { A (1) = 2
A (2) = 2
Paramètre = critères d'itérations .

. VL2 = 3 ⟹ { A (1) = 2
A (2) = 3
Paramètre = Entité.

2.2.2 Recherche d'un appel de module

. VAI = 1 ⟹ { A (1) = 4
A (2) = 1
Paramètre : désignation du module "fils" .

Niveau 2

2.3 Ordonnancement "descendant" des actions

. VAD = 3 ⟹ Empiler Identificateur dans pile CUM
{ A (1) = 5
A (2) = 1
Paramètre : Identificateur .

Niveau 2

2.4 Ordonnancement ascendant des actions

2.4.1 Recherche du type de "calcul"

. VAD = 1 ⟹ { A (1) = 5
A (2) = 5
Paramètre : Zone "Identidifcateur"
Zone "Calcul".

. VAD = 2 ⟹ { A (1) = 5
A (2) = 2
Paramètre : Zone "Identificateur"
Zone "Calcul" .

. VAD = 3 ⟹ Desempiler Identifícateur de la pile CUM .

2.4.2 Génération partielle de sommes algébriques

. VRE = 1 ⟹ { A (1) = 5
A (2) = 4
Paramètre : Zone "Identificateur" dernier élément de la pile CUM .

. VRE = 2 ⟹ { A (1) = 5
A (2) = 3
Paramètre : Zone "Identificateur" dernier élément de la pile.

Niveau 2

2.5 Examens du groupe conceptuel qui vient d'être exploré.

. VL2 = 1 :⟹ { A (1) = 3
A (2) = 1 .

. VL2 = 2 ⟹ { A (1) = 3
A (2) = 2 .

. VL3 = 3 ⟹ { A (1) = 3
A (2) = 3.

III. 213 Le module GP3

III. 2131 Format des lignes de la bibliothèque des textes

Chaque ligne de la bibliothèque des textes F3 représente une instruction du langage standard sous la forme suivante :

I (1)

code	paramètres

La liste des instructions nécessaires à l'expression du texte des paragraphes feuilles complète est la suivante :

I (1) = 1	Marque de début de paragraphe	Paramètre : désignation du paragraphe
I (1) = 4	Marque de fin de paragraphe	
I (1) = 20	Transfert de valeur	Paramètres { Désignation de l'Emetteur Désignation du Récepteur
I (1) = 22	Cumul Additif	Paramètres { Désignation du cumulateur Désignation de l'élément à ajouter
I (1) = 23	Cumul soustractif	Paramètres { Désignation du cumulateur Désignation de l'élément à retrancher
I (1) = 24	Expression Arithmétique	Paramètres { Désignation du résultat Expression arithmétique.

III. 2132 <u>Format des lignes de la bibliothèque des structures</u>

La structure des traitements d'un module conceptuel est représentée par une suite de lignes dans le format est le suivant :

ST(1)	ST(2)	ST(3)	ST(4)	ST(5)
Nature	Type	Itératif	Condition	Désignation

avec les conventions suivantes :

Nature de la ligne

ST (1) = 0	faire paragraphe feuille
ST (1) = 1	marque de début de traitement
ST (1) = 2	marque de fin d'un traitement
ST (1) = 3	appel de module "fils".

Type du traitement :

ST (2) = 1	Traitement conditionnel
ST (2) = 2	Traitement itératif associé à un indicatif de boucle
ST (2) = 3	Traitement itératif associé à un indicatif d'entité.

Niveau 1

Pour chaque ligne action de la représentation logique d'un module conceptuel faire

. A(1) = 1 ∧ A(2) = 1 | A (2) = 2 | A (1) = 3 | A (1) = 4 ⟹ génération d'éléments structurels

. A (1) = 5 ⟹ génération d'ordre de calcul dans le texte

. A (1) = 1 ∧ A (2) = 2 ⟹ { Insertion d'une marque de fin de paragraphe dans le texte
I (1) = 4.

Niveau 2

2.1 Génération d'éléments structurels

2.1.1 Génération de ligne (s) dans la table structure

2.1.2 Création d'une désignation de paragraphe feuille

2.1.3 Insertion d'une marque de fin de paragraphe dans le texte

. A (1) ¬ = 1 ⟹ I (1) = 4.

2.1.4 Insertion d'une désignation de paragraphe feuille dans le texte

{ I (1) = 1
Paramètre = désignation du paragraphe

2.1.5 Génération d'une ligne "faire paragraphe feuille" dans la table structure

{ ST (1) = 0
ST (4) = désignation du paragraphe.

Niveau 3

3.1 Génération dans la table structure

. A (1) = 2 ∧ A (2) = 1 ⟹ { ST (1) = 1 ; ST (2) = 1 ; ST (4) = condition .

. A (1) = 2 ∧ A (2) = 2 ⟹ { ST (1) = 1 ; ST (2) = 2 ; ST (3) = Indicatif de boucle ; ST (4) = Condition d'arrêt .

. A (1) = 2 ∧ A (2) = 3 ⟹ { ST (1) = 1 ; ST (2) = 3 ; ST (3) = Indicatif d'entité.

. A (1) = 3 ∧ A (2) = 1 ⟹ { ST (1) = 2 ; ST (2) = 1 .

. A (1) = 3 ∧ A (2) = 2 ⟹ { ST (1) = 2 ; ST (2) = 2 .

. A (1) = 3 ∧ A (2) = 3 ⟹ { ST (1) = 2 ; ST (2) = 3 .

. A (1) = 4 ⟹ { ST (1) = 3 ; ST (6) = désignation du module appelé.

III. 22 Le générateur secondaire

III. 221 Le module GS1

III. 2211 Format des paramètres logiques de la bibliothèque des paramètres F4

Pour une unité opérationnelle, la codification des paramètres logiques se présente sous la forme :

1 - d'une table "MODULES" dont chaque ligne contient la désignation d'un module conceptuel entrant dans la composition de l'UO. La première ligne stocke la désignation du module principal.

2 - d'une table "INDICATIFS" dont chaque ligne à la forme suivante :

Entité	Indicatif

III. 2212 L'algorithme

Niveau 1

co les tables structures des modules composant l'UO sont transférées ligne à ligne dans la table structure UO de même format que les précédentes, les lignes appels (ST (1) = 3) étant éliminées oc

Pour chaque ligne de la table structure du module principal faire

. ST (1) ¬ = 3 ⟹ Transfert de la ligne

. ST (1) = 3 ⟹ Insertion du module appelé

Niveau 2

2.1 Insertion du module appelé

2.1.1 Pour chaque ligne de la table ouverture du module appelé faire

. ST (1) ¬ = 3 ⟹ transfert de la ligne

. ST (1) = 3 ⟹ Insertion du module appelé

2.1.2 Retour à la table structure du module appelant

Niveau 3

3.1 Transfert de la ligne

3.1.1 Transfert de la ligne de la table structure du module dans la table structure de l'UO.

3.1.2 Substitution des indicatifs aux entités

ST (1) = 1 ∧ ST (2) = 3 ⟹ Recherche de l'indicatif associé à l'entité

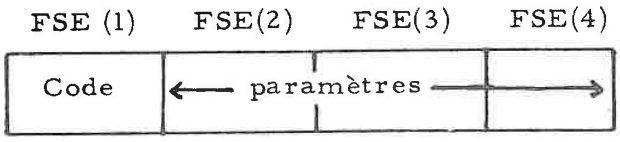
ST (3) = indicatif

III. 222 Le module GS2

III. 2221 Format de la bibliothèque des paramètres physiques F4

Les fichiers utilisés dans l'unité opérationnelle sont décrits à l'aide de trois tables :

1) la table des fichiers séquentiels en entrée FICHSQE a pour format de ligne :



avec les conventions suivantes :

FSE (1) = 1 description du fichier

FSE (2) = désignation du fichier

FSE(3) = étiquette logique du fichier

FSE (4) = désignation du buffer associé

FSE(1) = 2 description d'un indicatif

FSE(2) = désignation de l'indicatif

FSE(3) = ordre des valeurs de l'indicatif

FSE(4) = dimension des valeurs de l'indicatif.

2) La table des fichiers à accès a décrit, FICHDIR, a pour format de ligne

FD (1)	FD (2)	FD (3)	FD (4)
désignation du fichier	étiquette logique	désignation du buffer	désignation indicatif clé

3) La table des fichiers séquentiels en sortie, FICHSQO, a pour format de ligne :

désignation du fichier	étiquette logique	désignation du buffer	désignation indicatif mineur

III. 2222 Format intermédiaire de l'expression de la structure physique des traitements

Le passage de la table structure logique de l'unité opérationnelle au texte dans le langage standard de la structure physique, utilise une table intermédiaire STRPHY dont chaque ligne à le format suivant :

PY(1)	PY(2)	
niveau	code	paramètre

PY (1) donne le niveau de l'instruction dans la structure physique des traitements

PY (2) correspond au code de l'instruction du langage standard :

PY (2) = 1 marque de début de paragraphe
PY (2) = 3 marque de fin de programme
PY (2) = 4 marque de fin de paragraphe
PY (2) = 10 FAIRE eti

PY (2) = 11	SI condition FAIRE eti
PY (2) = 12	POUR I ALLANT DE I1 AVEC PAS I2 JUSQUA Condition d'arrêt FAIRE eti
PY (2) = 13	JUSQUA condition d'arrêt FAIRE eti.

III. 2223 L'algorithme

Nous ne donnerons ici que les algorithmes permettant de passer de la structure logique à la structure physique On trouvera dans [HERTSCHUH] la description détaillée de la génération des paragraphes de cinématique.

Niveau 1

1.1 Initialisation du Processus

1.2 Traitement des lignes du tableau représentatif de la structure logique

1.3 Clôture du Processus.

Niveau 2

2.1 Initialisation du processus

2.1.1 { NIVREF = 1 ; N° IND = 0 ; NIVMAX = 0

2.1.2 Créer une désignation de paragraphe de traitement

2.1.3 { PY (1) = NIVREF ; PY (2) = 10 ; Paramètre : désignation paragraphe de traitement

2.1.4 { NIVREF = NVIREF + 1 ; NIVMAX = SUP (NIVMAX, NIVREF)

2.1.5 { PY (1) = NIVREF
PY (2) = 1
Paramètre : désignation paragraphe de traitement

2.1.6 { PY (1) = NIVREF
PY (2) = 10
paramètre = OUVFICH

2.1.7 { PY (1) = NIVREF
PY (2) = 10
paramètre = LCI0

2.1.8 Générer les paragraphes de cinématique OUVF et LCI0

Niveau 2

2.2 Traitement des lignes du tableau structure

Pour chaque ligne du tableau structure faire

. ST (1) = 0 ⟹ { PY (1) = NIVREF
PY (2) = 10
Paramètre = ST(5)

. ST (1) = 1 Début d'un traitement

. ST (1) = 2 Fin d'un traitement.

Niveau 2

2.3 Clôture du Processus

2.3.1 { PY (1) = NIVREF
PY (2) = 10
Paramètre = FERMFICH

2.3.2 { PY (1) = NIVREF
PY (2) = 3

2.3.3 Passage au langage standard.

Niveau 3

3.1 Début d'un traitement

3.1.1 Créer une désignation de paragraphe de traitement

3.2.2 Analyse du type de traitement

. ST (2) = 1 ⟹ PY (1) = NIVREF
PY (2) = 11
Paramètres = Condition
Désignation du paragraphe de traitement

. ST (2) = 2 ⟹ PY (1) = NIVREF
PY (2) = 12
Paramètres = Indicatif de boucle
Condition d'arrêt
Désignation du paragraphe de traitement

. ST (2) = 3 ⟹ Générer condition d'arrêt TR [N°IND]
PY (1) = NIVREF
PY (2) = 13
Paramètres = Condition d'arrêt TR [N°IND]
Désignation du paragraphe de traitement
N° IND = N° IND + 1

3.2.3 NIVREF = NIVREF + 1

3.2.4 PY (1) = NIVREF
PY (2) = 1
Paramètre = désignation du paragraphe de traitement

3.2.5 Génération des opérations de cinématique

. ST (2) = 3 ⟹ Génération des paragraphes de cinématique initiaux.

Niveau 3

3.2 Fin d'un traitement

3.2.1 Génération des opérations de cinématique

ST (2) = 3 ⟹ Génération des paragraphes de cinématique finaux

3.2.2 PY (1) = NIVREF
PY (2) = 4

3.2.3 NIVREF = NIVREF - 1

Niveau 3

3.3 Passage au langage standard

Pour I allant de 1 à NIVMAX avec pas de 1 faire

Pour chaque ligne de STRPHY jusqua ST (1) = 0 faire

{ I (1) = PY (2)
{ Transfert des paramètres

Niveau 4

4.1 Génération des paragraphes de cinématique initiaux

4.1.1 Générer un paragraphe du type MAB [N° IND]

4.1.2
{ PY (1) = NIVREF
{ PY (2) = 10
{ Paramètre = MAB [N° IND]

4.1.3 Générer un paragraphe du type LD [N°IND]

4.1.4
{ PY (1) = NIVREF
{ PY (2) = 10
{ Paramètre = LD [N°IND]

4.1.5
{ PY (1) = NIVREF
{ PY (2) = 4

Niveau 4

4.2 Génération des paragraphes de cinématique finaux

4.2.1 Générer un paragraphe du type ECRI [N° IND]

4.2.2
{ PY (1) = NIVREF
{ PY (2) = 10
{ Paramètre = ECRI [N° IND]

4.2.3 Générer un paragraphe du type LCI [N° IND]

4.2.2 { PY (1) = NIVREF
PY (2) = 10
Paramètre = LCI [N° IND]

III.223 Le module GS3

III.2231 Format de la bibliothèque des paramètres F4

Le choix du langage de programmation est transmis par sa désignation stockée dans le mot LANGAGE.

III.2232 Algorithme

Niveau 1

1.1 Sélection de l'interface

1.2 Transformation des textes.

Niveau 2

2.1 Transformation des textes

Pour chaque ligne du texte du langage standard faire

Associer l'instruction correspondante au code I (1).

III - 3 EXEMPLE DE FONCTIONNEMENT DU GENERATEUR

Nous présentons dans ce paragraphe quelques uns des résultats obtenus aux différents stades de la génération traitement de la procédure "Facturation" présentée dans le chapitre I.

On peut se reporter, pour plus de précision, au dossier technique sur le fonctionnement du générateur,

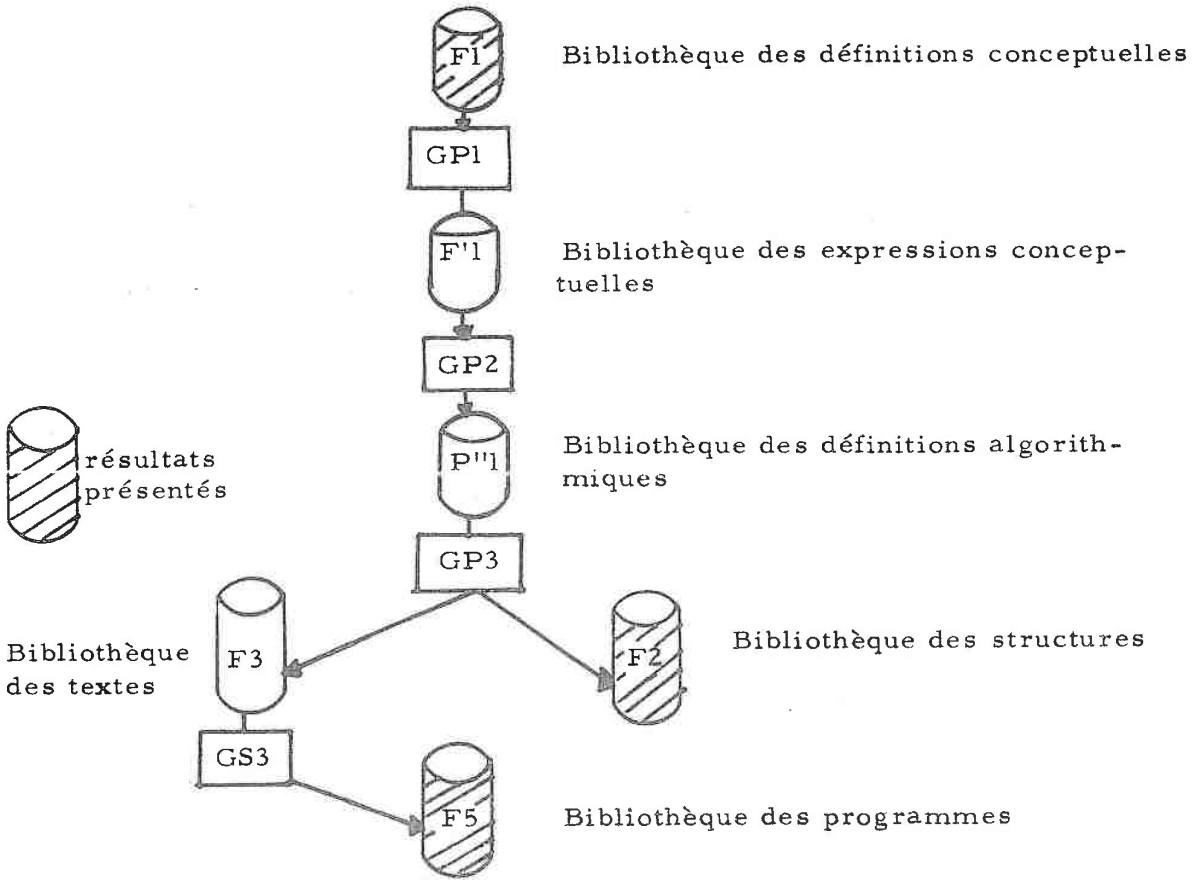
III. 31 Génération primaire des modules conceptuels

Pour chaque module nous donnons :

1 le texte du descripteur (bibliothèque F1)

2 la table structure logique du module (bibliothèque F2)

3 le texte des paragraphes feuilles, exprimé en COBOL pour en faciliter la lecture (bibliothèque F5).



III. 311 Génération primaire du module FACTURE

```
00                                                    FACTURE
01              1 CLIENT  3                           TOTFAC
02                        3                         + MBTTC
03                        0                         + MBHT        CALMBHT
03                        1 TTVA * MBHT             + TVA
02 MBHT > 5000            1 (MBHT - 5000) * TREM    - REM
02                                              AVOIR - AVOIR
99
```

Le texte du module FACTURE

```
                                          FAIRE   P01FACTURE
     JUSQUA        CLIENT   EPUISE
                                          FAIRE   P02FACTURE
                                          APPELEZ CALMBHT
                                          FAIRE   P03FACTURE
     SI CONDITION 01
                                          FAIRE   P04FACTURE
     FINSI
                                          FAIRE   P05FACTURE
     FINJUSQUA
-------------------------------------------------------------
                                          FAIRE   P06FACTURE

     01 MBHT > 5000
```

La table représentative de la structure logique du module FACTURE

```
P01FACTURE.
     EXIT.
P02FACTURE.
     MOVE ZERO                    TO TOTFAC .
     MOVE ZERO                    TO MBTTC  .
P03FACTURE.
     ADD MBHT       TO MBTTC  .
     COMPUTE
     TVA          = TTVA * MBHT               .
     ADD TVA         TO MBTTC     .
     ADD MBTTC       TO TOTFAC    .
P04FACTURE.
     COMPUTE
     REM          = (MBHT - 5000) * TREM      .
     SUBTRACT REM          FROM TOTFAC    .
P05FACTURE.
     SUBTRACT AVOIR        FROM TOTFAC    .
P06FACTURE.
     EXIT.
```

Le texte COBOL des paragraphes feuilles de la structure logique du module FACTURE

III. 312 Génération primaire du module CALMBHT

```
00                                                          CALMBHT
01                          3                               MBHT
02          1 PRODUIT 1 NB * PU                         * MPRD
99
```

Le texte du module CALMBHT

```
                                          FAIRE   P01CALMBHT
JUSQUA        PRODUIT  EPUISE
                                          FAIRE   P02CALMBHT
FINJUSQUA
                                          FAIRE   P03CALMBHT
```

La table représentative de la structure logique du module CALMBHT

```
P01CALMBHT.
     MOVE ZERO                    TO MBHT                .
P02CALMBHT.
     COMPUTE
     MPRD        = NB * PU                               .
     ADD MPRD      TO MBHT      .
P03CALMBHT.
     EXIT.
```

Le texte COBOL des paragraphes feuilles de la structure logique du module CALMBHT

III.32 Génération de l'unité opérationnelle "FACTURE"

Soit l'unité opérationnelle FACTURE définie à partir de la hiérarchie des modules

FACTURE
|
CALHBHT.

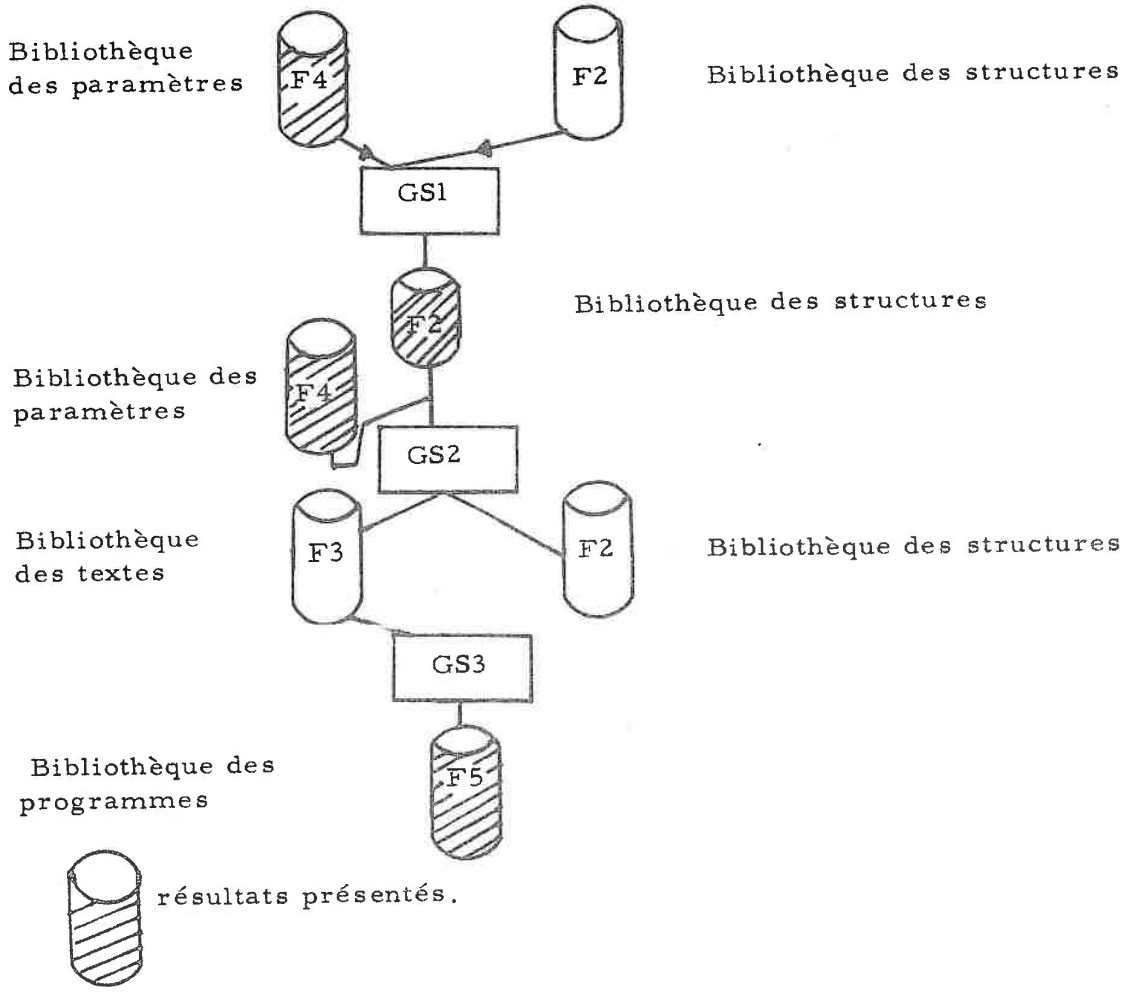
Nous donnons :

. le contenu de la bibliothèque des paramètres (F4)

. la table représentative de la structure logique de l'unité opérationnelle FACTURE

. les textes de la structure physique et des paragraphes de cinématique exprimés en COBOL (Bibliothèque F5).

Ces résultats correspondent à la séquence d'actions suivantes :



```
* TABLE 'MODULES'
  FACTURE   CALMBHT

* TABLE 'INDICATIFS'

  CLIENT    NCLI
  PRODUIT   NPROD
```

Contenu partiel de la bibliothèque F4

```
                                    FAIRE    P01FACTURE

JUSQUA          NCLI      EPUISE

                                    FAIRE    P02FACTURE

                                    FAIRE    P01CALMBHT

JUSQUA          NPROD     EPUISE

                                    FAIRE    P02CALMBHT

FINJUSQUA

                                    FAIRE    P03CALMBHT

                                    FAIRE    P03FACTURE

SI CONDITION 01

                                    FAIRE    P04FACTURE

FINSI

                                    FAIRE    P05FACTURE

FINJUSQUA

                                    FAIRE    P06FACTURE

01 MBHT > 5000
```

Table représentative de la structure logique de l'UO FACTURE

```
* TABLE 'FICHSQE'

  1 FCLIENT     SE01      FICHE
  2 NCLI        CR01      X(5)
  1 FCOMD       SE02      COMMANDE
  2 NCLI        CR01      X(5)
  2 NPROD       CR01      X(4)
```

co FCLIENT a pour indicatif NCLI et donne les caractéristiques des clients.

FCOND a pour indicatif NCLI/NPROD et décrit les commandes de chaque client.

Table des fichiers séquentiels en entrée

```
* TABLE 'FICHDIR'
FPROD       FD01       FPRODUIT      NPROD
```

co FPROD ayant pour indicatif NPROD est un catalogue de prix des produits.

Table des fichiers à accès direct en entrée

```
* TABLE 'FICHS00'
RCLIENT     S001       VALFACT       NCLI
RPROD       S002       VALIGNE       NPROD
```

co RCLIEN archive tous les résultats attachés à l'indicatif NCLI

RPROD archive tous les résultats attachés à l'indicatif NPROD

Table des fichiers séquentiels en sortie

Contenu partiel de la bibliothèque F4

```
      PERFORM M01FACTURE .
M01FACTURE.
      PERFORM OUVFICH     .
      PERFORM LCI0        .
      PERFORM P01FACTURE  .
      PERFORM M02FACTURE UNTIL V01 AND V02                    .
      PERFORM P06FACTURE  .
      PERFORM FERMFICH    .
      STOP RUN .
M02FACTURE.
      PERFORM MAB01       .
      PERFORM LD01        .
      PERFORM P02FACTURE  .
      PERFORM P01CALMBHT  .
      PERFORM M03FACTURE UNTIL V02
          OR   ZR01 NOT = ZT02I01                             .
      PERFORM P03CALMBHT  .
      PERFORM P03FACTURE  .
      IF        MBHT > 5000
      PERFORM M04FACTURE  .
      PERFORM P05FACTURE  .
      PERFORM ECRI01      .
      PERFORM LCI01       .
M03FACTURE.
      PERFORM MAB02       .
      PERFORM LD02        .
      PERFORM P02CALMBHT  .
      PERFORM ECRI02      .
      PERFORM LCI02       .
M04FACTURE.
      PERFORM P04FACTURE  .
```

Texte COBOL : image de la structure physique de l'unité opérationnelle FACTURE

```
OUVFICH .
        OPEN INPUT      FCLIENT
                        FCOMD
                        FPROD .
        OPEN OUTPUT     RCLIENT
                        RPROD .
LCI0    .
        MOVE 0                          TO FF01
        MOVE 0                          TO FF02
        PERFORM LF01 .
        PERFORM LF02 .
        PERFORM CI01 .
        PERFORM CI02 .
MAB01   .
        MOVE SPACES                     TO VALFACT .
LD01    .
        EXIT.
MAB02   .
        MOVE SPACES                     TO VALIGNE .
LD02    .
        MOVE ZR01                       TO NPROD    OF PRODUIT
        READ FPROD  INVALID KEY         PERFORM ERRFPROD
ECRI02  .
        MOVE ZR01                       TO NCLI     OF VALIGNE
        MOVE ZR02                       TO NPROD    OF VALIGNE
        WRITE VALIGNE .
LCI02   .
        PERFORM LI02 .
        PERFORM CI02 .
LI02    .
        IF      F02
          AND NCLI      OF COMMANDE   = ZR01
          AND NPROD     OF COMMANDE   = ZR02
        PERFORM LF02 .
        IF      V02
        MOVE HIGH-VALUES                TO NCLI     OF COMMANDE .
LF02    .
        READ FCOMD      AT END
        MOVE 1                          TO FF02
CI02    .
        MOVE NCLI       OF COMMANDE     TO ZT02I01
        MOVE NPROD      OF COMMANDE     TO ZR02
ECRI01  .
        MOVE ZR01                       TO NCLI     OF VALFACT .
        WRITE VALFACT .
LCI01   .
        PERFORM LI01 .
        PERFORM CI01 .
LI01    .
        IF      F01
          AND NCLI      OF FICHE      = ZR01
        PERFORM LF01 .
        IF      V01
        MOVE HIGH-VALUES                TO NCLI     OF FICHE .
LF01    .
        READ FCLIENT    AT END
        MOVE 1                          TO FF01
CI01    .
        MOVE HIGH-VALUES                TO ZR01

        IF      NCLI    OF FICHE < ZR01
        MOVE NCLI       OF FICHE        TO ZR01
        IF      NCLI    OF COMMANDE < ZR01
        MOVE NCLI       OF COMMANDE     TO ZR01
FERMFICH .
        CLOSE           FCLIENT
                        FCOMD
                        FPROD
                        RCLIENT
                        RPROD .
```

Textes des paragraphes de cinématique générés par le générateur d'

HERTSCHUH

III.4 CONCLUSION

Les outils que nous venons de présenter sont écrits en COBOL et sont actuellement opérationnels sur l'IRIS 80 dans le cadre du prototype REMORA.

Ils fonctionnent sous deux hypothèses restrictives :

- un interface COBOL
- une organisation des données en fichiers séquentiels ou directs.

Nous pensons les étendre incessamment en prenant en compte les structures de données plus complexes (hiérarchiques, réseaux) associées à la gestion des informations dans des bases de données et l'expérimenter en SOCRATE.

Nous compléterons également la partie documentation liée à l'impression des différents niveaux de structure ou de texte dans le langage standard.

CONCLUSION

CONCLUSION

Le travail que nous avons présenté avait un double objectif théorique :

1. Mieux comprendre à travers l'évolution des langages de programmation, d'analyse, des langages fonctionnels, ce que doit être un langage conceptuel d'énoncé des problèmes de gestion et définir le langage de conception que nous avons choisi dans le projet REMORA pour donner l'expression des problèmes au niveau conceptuel du processus de conception - réalisation du SI.

2. Définir parallélement à ce qui a été fait sur les données, les structures de traitements des niveaux conceptuel -logique - physique intervenant dans un processus de conception - réalisation du SI en 3 étapes, maintenant admis par la communauté scientifique, ainsi que les expressions des traitements à ces trois mêmes niveaux.

Il avait également un objectif pratique que nous avons satisfait : celui de réaliser un automate assurant, sous certaines conditions de fonctionnement du SI, le passage de l'expression conceptuelle des problèmes aux programmes Cobol correspondants.

Nous pensons qu'il serait intéressant de faire une analyse plus approfondie de tous les langages permettant la formulation des traitements de gestion pour en définir une classification permettant de saisir leurs caractéristiques : les possibilités qu'ils offrent, leurs limitations, leurs conditions d'utilisation, et peut être leurs performances...
Nous entreprenons cette étude en collaboration avec O. THIERY.

Parallèlement nous exploitons simultanément les résultats de ce travail et les expériences pédagogiques menées par C. ROLLAND et O. FOUCAUT en matière d'enseignement de la programmation de gestion pour approfondir notre méthode de programmation.

Nous souhaitons étendre les conditions de fonctionnement de l'outil générateur en particulier en acceptant la gestion des informations dans une base de données (expérimentalement SOCRATE).

Enfin, ce travail comme ceux qui ont déjà été proposés dans le projet, est intégré à l'ensemble des solutions au problème de la conception - réalisation du SI - que nous proposons et dont O. FOUCAUT s'attache à préparer une présentation globale dans son travail de thèse d'état.

---

* Un document sera disponible à l'UER de Mathématiques et Informatique de NANCY II en Janvier 1978.

BIBLIOGRAPHIE

IV - BIBLIOGRAPHIE

1 - J.R. ABRIAL "Data Semantics"
IFIP Working Conference Cargese - 1974 -

2 - ACTES du séminaire international : Data structure models for information systems -
Presses Universitaires - Namur - 1975 -

3 - ATLAS - SEMA

3b- ATOS - SLIGOS

4 - BACKUS The syntax and semantics of the proposed international algebrave language of the Zurich ACM-GAMM conference (ICIP) Paris juin 1959

5 - G. BENCI - "Les données : une ressource économique à mieux gérer".
INFORMATIQUE ET GESTION - 1976 - n° 78

6 - G. BENCI, F. BODART, H. BOGAERT, A. CABANES "Concept for the design of a Conceptuel schema".
IFIP Working Conference - Freudenstadt - (ALL) - 1976 -

6b- G. BENCI, C. ROLLAND : "La conception des bases de données et les modèles de données".
Cours MIAG, IRIA - 1976 -

7 - M.T. BERTINI, Y. TALLINEAU "Cobol structuré : un modèle de programmation".
Ed. d'organisation - 1974 -

8 - F. BODART "Structuration fonctionnelle du système informatique d'une organisation : propositions méthodologiques".
Institut d'Informatique de Namur - Séminaire INFORSID - AIX en Provence - 1974 -

8b- CGI - Méthode d'analyse CORIG

9 - Club Banque de Données "Modèles de structures de données dans les systèmes d'information
IRIA - 1973 -

9b- CODASYL Codasyl data base task group report - April 71 - IFIP Data Processing Group;

10 - D. COUGER "Evolution of Business System Analysis Techniques". Computing surveys
vol. 5 n° 3 - 1973 -

11 - CROCUS "Principes de conception des systèmes d'exploitation". Dunod - 1974 -

11b- DELOBEL "Les systèmes de gestion des bases de données". Cours Ecole d'Eté d'Informatique
Rabat - 1976 -

12 - DIJKSTRA "Notes on structured programming in structured programming".
ADIS studies in Data Processing - Academic Press - 1972 -

13 - DERNIAME "CIVA : un système de programmation modulaire". Thèse d'Etat - Nancy - 1974 -

14 - M. GRANDBASTIEN, P. LESCANNE, C. PAIR, A. QUERE, D. REMY "Blanche Neige".
Cours à l'école d'été d'informatique - Tarbes - 1975 -

15 - JS. GERO "Ethics in Computer Aided Design : a polemic" ACM SIGMA Newsletter
vol. 4 n° 4 - 1975 -

16 - GRANDMOUJIN "Un langage de très haut niveau" Journées logique et programmation -
IRIA - St Pierre de Chartreuse - 1975 -

17 - H. HABRIAS "Méthode analytique - Méthode synthétique en information d'organisation" INFORMATIQUE ET GESTION - 1975 -

18 - HERTSCHUH "L'analyse dans le projet CIVA" - Thèse de 3° cycle - Nancy - 1974 -

18b- INFORSID II : "Modèles - Méthodes - Outils pour la Conception et la Réalisation des SI". Congrès de CAEN - 1976 -

19 - M. KRIEGUER "Les données et leurs structures dans le projet REMORA (thèse de 3° cycle à paraître).

21 - LE SUISSE "Exposé introductif au projet ISDOS" Club Banque de données n° 16 - 1976 -

22 - LUGUET "SCAPEACE" Thèse d'état - Toulouse - 1975 -

23 - MAC GEE "File structures for generalized data management". Proccedings IFIP Conference - 1968 -

24 - J. MAROLD, C. PAIR "Introduction à une méthode de programmation déductive". Institut National Polytechnique - Nancy - 1975 -

25 - METACOBOL - CAP

26 - Méthode MINOS - SLIGOS - 1970 -

27 - Méthode PROTEE - SIS - 1970 -

28 - J.C. NUNAMEKER "A methodology for the design and optimisation of information processing systems" STCC - 1971 -

29 - C. PAIR "Propositions pour un langage de très haut niveau" IFIP TC2 - 1975 -

30 - F. PECCOUD "MACSI - Méthode d'aide à la conception des systèmes d'informations" Thèse d'état - Grenoble - 1973 -

31 - W. PEREA VILLACORTA "Méthode d'analyse dans le projet Remora" Thèse de 3° cycle - Nancy - 1976 -

32 - PIROTTE "Comparaison de langages d'interrogation de bases de données relationnelles" Club Banque de données n° 16

33 - REIX "L'analyse en informatique de gestion" DUNOD - 1971 -

34 - M.E. SENKO "Data description language in the context of a multilevel structured description" - DIAM II - IFIP TC-2 - Special Working Conference.

35 - TAYLOR "Generalized data base management system data structures and their mapping to physical storage". ISDOS Working paper n° 49 - 1971 -

36 - D. TEICHROEW, W.J. RATAJ, E.A. HERSCHEY "An introduction to computer aided documentation of user requirements for computer based information processing systems". ISDOS Working Paper n° 72 - 1973 -

37 - O. THIERY - "L'aide à la Conception dans le projet REMORA". Thèse de 3° cycle - Nancy - 1976 -

38 - ULYSSE - GACHOT S.A.

39 - WARNIER "La transformation des programmes" Ed. d'organisation - 1973 -

40 - WATERS "Méthodologie assistée par ordinateur dans la conception des systèmes informatiques" L'informatique - 1976 -

41 - WIRTH "Algorithms + data structures = Programs" Prentice-Hall - 1976 -

42 - WORKTEN - NATIONAL COMPUTER INDUSTRIE

43 - GAMMA CONSEIL "ARIANE" - 1970 -

44 - CABANES "Un modèle d'accès standard dans les systèmes de gestion de Bases de Données"
Club Banque de données n° 16

45 - CODD "Normalized Data Base structure : a brief tutorial"
Sigfider workshop 11-12 Novembre 1971 -

46 - CATTAN "La percée des langages d'analyse". Informatique et gestion n° 51

NOM DE L'ETUDIANT : *LAMY Dominique*

NATURE DE LA THESE : *DOCTORAT DE SPECIALITE en INFORMATIQUE*

VU, APPROUVE

& PERMIS D'IMPRIMER

NANCY, le *13 mai 1977*

LE PRESIDENT DE L'UNIVERSITE DE NANCY I