INSTITUT NATIONAL POLYTECHNIQUE
DE LORRAINE

CENTRE DE RECHERCHE
EN INFORMATIQUE DE NANCY

Cadre et méthode de spécification de systèmes d'information fondés sur les types de données

THESE

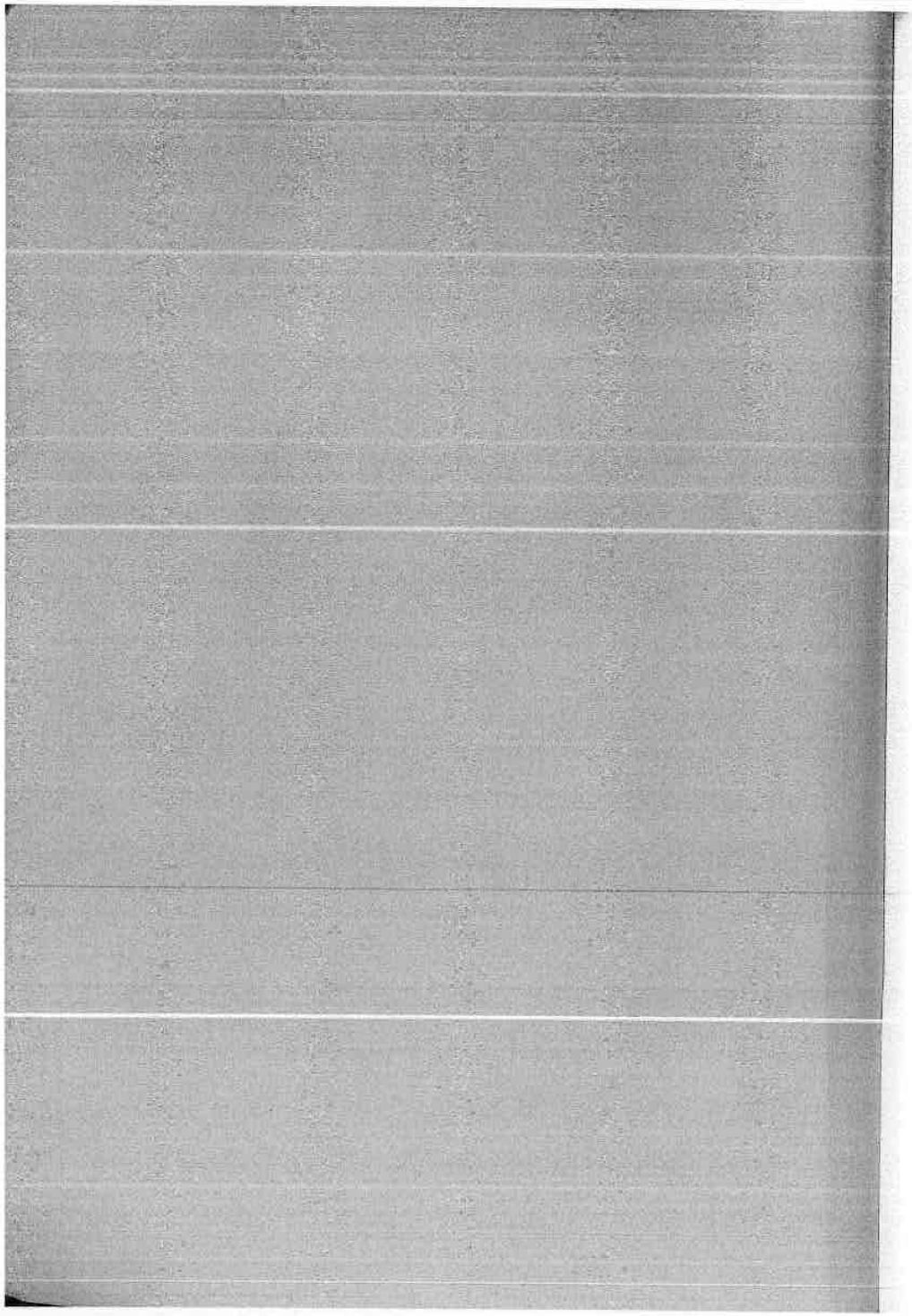
présentée pour l'obtention du diplôme de
DOCTEUR INGENIEUR EN INFORMATIQUE

par

Eric DUBOIS

SOUTENUE LE 22 MARS 1984
DEVANT LA COMMISSION D'EXAMEN

M. J.-P. FINANCE	Président
M. D. COULON	Examinateurs
M. A. VAN LAMSWEERDE	
M. B. MEYER	
M. A. VAISSIERE	



INSTITUT NATIONAL POLYTECHNIQUE
DE LORRAINE

CENTRE DE RECHERCHE
EN INFORMATIQUE DE NANCY

Cadre et méthode de spécification de systèmes d'information fondés sur les types de données

THESE

présentée pour l'obtention du diplôme de
DOCTEUR INGENIEUR EN INFORMATIQUE

par

Eric DUBOIS

SOUTENUE LE 22 MARS 1984
DEVANT LA COMMISSION D'EXAMEN

M. J.-P. FINANCE	Président
M. D. COULON	Examinateurs
M. A. VAN LAMSWEERDE	
M. B. MEYER	
M. A. VAISSIERE	

C'est un grand plaisir pour moi de remercier tous ceux qui, à titres divers, ont contribué à la réalisation de ce travail:

- J.-P. FINANCE, Professeur à l'Université de Nancy-1, pour m'avoir donné l'opportunité de faire ce travail. C'est lui qui m'a initié, pendant ces trois années, à la recherche d'une démarche rigoureuse en matière de spécification. Je le remercie vivement pour avoir été le guide et le conseiller de mon travail.

- A. VAN LAMSWEERDE, Professeur à l'Institut d'Informatique de Namur, qui m'a orienté il y a près de quatre ans vers ce sujet de recherche. Je suis très reconnaissant de l'intérêt qu'il a toujours manifesté pour mon travail au travers des discussions que nous avons eues et des conseils qu'il m'a donnés.

- B. MEYER, Ingénieur-chercheur chez EDF, actuellement en séjour à l' "University of California". Spécialiste des problèmes de spécification et de programmation, je le remercie beaucoup d'avoir bien voulu sacrifier quelques heures de son séjour en France pour faire partie de cette commission d'examen.

- D. COULON, Professeur à l'Institut National Polytechnique de Nancy, spécialiste des problèmes d'intelligence artificielle. Je lui suis très reconnaissant d'avoir accepté de siéger à ce jury et d'y apporter son point de vue sur un sujet dont les interrelations avec son domaine de recherche sont certains.

- A. VAISSIERE, Ingénieur à la Société Nationale Elf Aquitaine, initiateur de cette possibilité qui m'a été donnée de confronter un certain nombre de mes idées dans un environnement industriel. Je le remercie vivement pour l'intérêt qu'il a toujours porté à ce travail ainsi que pour avoir accepté un long déplacement pour faire partie de ce jury.

Ces remerciements s'adressent également à tous mes amis et collègues du groupe "Spécification et Programmation". Je tiens à remercier particulièrement Nicole Lévy pour son amitié ainsi que pour les discussions passionnantes et passionnées que nous avons eues sur des sujets communs de recherche.

Ces remerciements ne seraient pas complets si je n'évoquais toutes les personnes avec lesquelles j'ai pu collaborer et avoir des échanges de vue au sein de la Société Nationale Elf Aquitaine (Production). Je remercie plus spécialement J.-R. Laplace pour les conseils et le suivi de mon travail qu'il a assuré pendant près de deux ans au sein de cette société.

Je ne voudrais pas terminer sans remercier P. Cuvelier qui s'est assuré avec compétence et dans des délais records la dactylographie de cette thèse.

Je sais gré à la Société Nationale Elf Aquitaine (Production) de leur contribution déterminante à ce projet par le contrat n° 4203.

SOMMAIRE

INTRODUCTION

INTRODUCTION

1. La spécification des systèmes d'information

Depuis plusieurs années, les systèmes d'information, et plus particulièrement ceux qui sont automatisés, sont considérés comme des outils indispensables dans la gestion d'une organisation, du fait qu'ils assurent la communication entre le système opératoire, le système de gestion (associé aux comportements décisionnels) et l'environnement.

Nous retiendrons comme définition d'un système d'information celle de N.C. CHURCHILL, C.H. KRIEBEL et A.C. STREDY citée dans [LEM,73] :

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

Du fait de la complexité de tels systèmes, il est devenu traditionnel de décomposer leur conception et leur exploitation en un certain nombre d'étapes constituant ce que l'on appelle le cycle de vie d'un système.

On peut distinguer les étapes suivantes :

1° la définition préliminaire des besoins;

2° la spécification détaillée du système retenu pour satisfaire les besoins;

3° l'analyse de conception (ou de programmation) pour la partie informatique du système décrit;

4° le codage;

5° la validation;

6° l'installation et la maintenance.

En dépit de nombreux systèmes d'information à caractère informatique déjà installés dans les organisations, le coût de développement et de maintenance de tels systèmes restent très importants au point que l'on parle depuis un certain nombre d'années de "crise du logiciel".

Ces dernières années, deux types de solution ont été proposés pour faire face à ce problème :

- d'une part, la mise au point d'outils (la plupart automatisés) susceptibles d'assister le travail nécessaire dans une ou plusieurs étapes du cycle de vie;

- d'autre part, le développement de méthodes permettant d'appréhender à la fois l'aspect technique (que faut-il faire à chaque étape, comment le faire ?) et l'aspect gestion du projet (établissement des budgets, délais, audit,...).

Des études, telle que celle réalisée par [BOE,76] , ont montré que le coût des erreurs de spécification représente en moyenne 2/3 du coût total de maintenance du logiciel, soit le tiers de son coût total.

L'étape de spécification s'avère, en effet, généralement difficile et cruciale dans le cycle de vie d'un système d'information, du fait du quadruple rôle joué par la description d'un problème :

- La spécification reflète les premiers choix du demandeur quant au système attendu; l'étude de ces choix est d'autant plus importante qu'ils vont conditionner tous les choix ultérieurs de réalisation.

- Les spécifications sont à la base du contrat liant le demandeur qui pose le problème et l'informaticien qui le résout. En ce sens, elle doit être à la fois précise et compréhensible par les deux parties.

- De bonnes spécifications peuvent grandement faciliter la mise en oeuvre ultérieure du système et permettre d'établir la validité du produit réalisé eu égard aux besoins spécifiés.

- Les spécifications constituent un élément de base de la documentation destinée à la maintenance du produit réalisé.

2. L'environnement nancéen : le projet SPES

L'objectif du projet SPES est le développement d'un système interactif de spécification et de résolution de problèmes [FIN,83] .

Dans ce système, l'activité de programmation se décompose en deux étapes :

1° La spécification du problème;

2° La construction d'un algorithme, puis d'un programme pour résoudre ce problème.

Dans la première étape, on se contente simplement d'exprimer une propriété caractérisant le résultat à partir de données sans se préoccuper ni d'algorithme de résolution, ni de représentation des données manipulées.

Pour aider le demandeur à franchir cette étape de formalisation, il faut disposer d'un langage de spécification, d'une méthode de spécification et, si possible, d'aides logicielles.

L'expression d'un algorithme, puis d'un programme est alors abordée comme une deuxième étape à partir de la spécification. La construction de l'algorithme et du programme peut alors être vue comme des étapes de transformation "formelles".

Supportant ces deux étapes, le système offre un certain nombre d'outils se répartissant en différentes couches :

- la première couche contient les fonctions de manipulation de la spécification (création, modification, contrôles syntaxiques,...);

- la deuxième couche intègre des outils de transformation de la spécification et de visualisation graphique de parties de spécification;

- la troisième couche correspond au pilote de la spécification, son rôle est de guider l'utilisateur afin qu'il puisse compléter de façon systématique sa spécification du problème.

Revenons maintenant plus en détail sur l'étape de spécification.

Au niveau de l'expression de la spécification, le langage de spécification utilisé est un langage formel fondé sur le calcul des prédicats du premier ordre fortement typé.

Cependant, on peut compléter ces définitions formelles par des définitions informelles considérées comme de simples commentaires et constituant un lexique.

Exemple :

La spécification du problème suivant : calculer le nombre de maxima relatifs dans une suite donnée sur un ensemble totalement ordonné (en cas de "palier", le maximum est le dernier élément du palier).

nbre-max-rel : nombre de maxima relatifs d'une suite donnée d : suite donnée sur un ensemble totalement ordonné par $\leqslant$ card : fonction de calcul du cardinal d'un ensemble ens-max-rel : ensemble des indices des éléments constituant des maxima relatifs de la suite de données $\in$: prédicat d'appartenance d'un élément à un ensemble max-rel : indice d'un élément constituant un maximum relatif de la suite donnée bs : fonction donnant la borne supérieure d'une suite j : indice d'un élément de la suite donnée	résultat : nbre-max-rel donnée : α nbre-max-rel = card(ens-max-rel) $\forall$ max-rel $\in$ ens-max-rel (max-rel < bs(d)) $\Rightarrow$ (α[max-rel + 1] < α[max-rel]) et (1 < max-rel $\Rightarrow$ $\exists$ j tq j < max-rel et α[j] < α[max-rel] et α[j+1,...,max-rel-1] < α[max-rel]) d : SUITE [ENTIER] ens-max-rel : ENSEMBLE [ENTIER] nbre-max-rel, max-rel, j : ENTIER

Les constituants de base d'une spécification sont des relations. Une relation porte sur des objets parmi lesquels on distingue un résultat et des données (données de départ du problème ou résultats intermédiaires); le corps d'une relation exprime une propriété reliant résultat et donnée.

Les différentes relations (prédicats ou fonctions) ainsi décrites constituent l'énoncé du problème.

L'ensemble des définitions de types constituent l'univers du problème.

Pour construire l'énoncé et l'univers du problème, il convient de suivre une méthode de spécification favorisant une approche progressive et structurée.

La méthode proposée, issue de la méthode de programmation déductive [PAI,79], invite à spécifier d'abord le résultat puis les différents intermédiaires introduits [FIN,79].

Le point de départ est la relation associant données et résultats dans le problème considéré. La caractérisation des résultats conduit à introduire progressivement des objets, des relations intermédiaires et des types; les relations intermédiaires devant être définies à leur tour. Les différentes relations sont, par conséquent, structurées en un arbre associé au graphe de dépendance des objets. La construction de l'énoncé est achevée lorsque tous les objets, types et relations introduits ont été définis formellement.

En parallèle, on est amené à préciser progressivement la structure des différents types introduits jusqu'à l'utilisation de types élémentaires (entier, booléen,...); par conséquent, l'univers se structure en une famille de types hiérarchiques.

Illustrons le déroulement de la méthode sur la construction de la spécification du problème associé au calcul du nombre de maxima relatifs dans une suite donnée.

Initialement

a) donner un nom au résultat, soit : nbre-max-rel

b) définition informelle du résultat : nbre-max-rel est le nombre de maxima relatifs d'une suite donnée.

Etape 1 : détermination de nbre-max-rel

a) définition formelle du résultat : nbre-max-rel = card(ens-max-rel)
nbre-max-rel : ENTIER

b) définition informelle de l'intermédiaire : ens-max-rel est l'ensemble des indices des éléments constituant des maxima relatifs de la suite donnée.

(Remarque : la caractérisation de la fonction "card" n'est pas nécessaire, c'est une opération standard de calcul du cardinal d'un ensemble).

Etape 2 : détermination de ens-max-rel

a) définition formelle de l'intermédiaire

$\forall$ max-rel $\in$ ens-max-rel

(max-rel $<$ bs[$\propto$] $\Rightarrow$ $\propto$[max-rel+1] $<$ $\propto$ [max-rel] et

(1 $<$ max-rel $\Rightarrow$ $\exists$ j tq j $<$ max-rel et

$\propto$ [j] $<$ $\propto$ [max-rel] et

$\propto$ [j+1,...,max-rel-1] $\leq$ $\propto$[max-rel]

ens-max-rel : ENSEMBLE [ENTIER]

$\propto$: SUITE [ENTIER]

max-rel,j : ENTIER

b) définitions informelles des données intermédiaires

max-rel : indice d'un élément constituant un maximum relatif de la suite donnée

j : indice d'un élément de la suite donnée

$\propto$: suite donnée sur un ensemble totalement ordonné par $\leq$

(Remarque : la caractérisation des fonctions "$\in$" et "bs" n'est pas nécessaire, il s'agit d'opérateurs standards indiquant respectivement l'appartenance d'un élément à un ensemble (fonction booléenne - prédicat) et donnant la borne supérieure d'une suite).

3. Objectifs du travail

Le but de notre travail est de préciser les idées développées dans SPES pour les rendre davantage praticables à la spécification de systèmes d'information au sein d'une organisation.

Plus particulièrement, nos objectifs sont :

- de particulariser et préciser la méthode et le langage dans le cas de systèmes d'information;

- de valider ces idées et propositions sur des exemples en vraie grandeur.

Dans la spécification d'un système d'information, nous distinguons :

a) la spécification d'un noyau du système : c'est-à-dire la description des traitements et des données caractéristiques du type d'application considérée; cette description est la plus abstraite en ce sens qu'elle évite au maximum d'intégrer des contraintes dictées par l'organisation (en particulier, les contraintes de réalisation);

b) la spécification de caractéristiques apparaissant comme dépendantes de l'environnement organisationnel spéficique (contraintes de réalisation, de performances attendues,...).

Le travail se concentre davantage sur la première étape de description des traitements et données.

La spécification du noyau se décompose d'une part, en la spécification des fonctions devant être réalisées par le système (l'énoncé du problème) et d'autre part, en la spécification des différents types de données manipulées (l'univers du problème).

Pour faciliter la tâche du spécifieur de gestion, nous avons défini une fois pour toutes un ensemble de relations et de types de données dont l'utilisation s'est avérée fréquente dans le domaine d'application considéré; à chaque type est associée une famille d'opérations caractéristiques.

En particulier, un certain nombre de ces opérations portent sur des suites; nous avons, en effet, constaté que le concept de suite occupe un rôle privilégié dans la description des systèmes d'information.

L'utilisation de ces mécanismes de base par le spécifieur nécessite l'existence d'un guide méthodologique favorisant leur mise en oeuvre. Force, cependant, est de constater que dans le contexte de la spécification des systèmes d'information, il existe peu de méthodes suffisamment rigoureuses et structurées pour favoriser la construction correcte de la spécification. Dès lors, nous proposons, dans ce travail, une méthode permettant de construire, de façon systématique, la spécification du problème; la démarche suggérée est caractérisée par un processus descendant et déductif dans lequel on spécifie en parallèle l'énoncé et l'univers.

Le langage de spécification utilisé constitue un sous-ensemble du langage de spécification formel utilisé dans SPES.

En effet, nous avons constaté que le langage basé sur le calcul des prédicats du premier ordre était :

- d'une part, beaucoup trop riche, par rapport à la plupart des formes d'énoncé nécessaires pour décrire les applications de gestion;

- d'autre part, difficilement praticable par un spécifieur de gestion sans une formation spécifique.

L'occasion nous a été donnée, pendant deux ans, de pouvoir expérimenter un certain nombre de nos idées en matière de spécification dans un environnement industriel, à savoir la Société Nationale Elf Aquitaine Production (SNEA.P).

Nous avons été ainsi amenés successivement à spécifier :

- un système comptable de suivi des dépenses informatiques

- une application de suivi du planning des forages.

La présentation de ce travail est la suivante :

- le premier chapitre est consacré à un panorama des approches à la spécification des systèmes d'information (modèles, langages, méthodes et outils de spécification);

- dans le second chapitre, nous présentons les caractéristiques générales de notre approche à la spécification de l'énoncé du problème et de son univers;

- le troisième chapitre est consacré à la présentation et à la définition formelle des différents outils de construction de la spécification;

- dans le chapitre quatre, les étapes successives de la méthode de spécification sont indiquées; le déroulement du "méta-algorithme" proposé est illustré au travers de son application à un exemple;

- dans le chapitre cinq, nous rendons compte d'expériences que nous avons eues pendant deux ans dans l'environnement industriel d'une société française;

- dans le sixième chapitre, nous évoquons quelques prolongements possibles à ce travail;

Ce travail est un approfondissement des chapitres 1.2. et 1.4. de la thèse de Jean-Pierre FINANCE [FIN,79] ; il constitue également un prolongement d'un travail de fin d'études précédemment réalisé [DUB,81] .

o o o

Chapitre 1

ETAT DE L'ART EN MATIERE DE SPECIFICATION DE SYSTEME D'INFORMATION

1.1. LA PROBLEMATIQUE DE LA SPECIFICATION

L'étape d'analyse fonctionnelle

Depuis un certain nombre d'années, le coût de développement et de maintenance des systèmes informatiques n'a cessé de croître, au point que l'on parle actuellement de "crise du logiciel". Un des facteurs inhérent à cette crise est dû au fait que les systèmes développés ne respectent un certain nombre de qualités, telles que :

- l'adéquation aux besoins des demandeurs;
- la robustesse face à certains incidents;
- les performances offertes;
- la correction et la fiabilité;
- l'évolutivité lors de l'expression de nouveaux besoins.

Pour permettre aux informaticiens de pouvoir davantage se concentrer sur les aspects difficiles et critiques de leur travail, il est devenu traditionnel de subdiviser leurs activités en un certain nombre d'étapes; celles-ci constituent le cycle de vie du projet informatique.

Une étape importante dans ce cycle de vie est constituée par l'analyse fonctionnelle (qualifiée également par certains d'analyse conceptuelle); celle-ci établit la charnière entre l'étude d'opportunité (étape pendant laquelle les utilisateurs décrivent le cadre générale du système d'information à implanter) et l'analyse de programmation (étape concernant la réalisation de la partie informatique du système d'information).

Généralement, l'étape d'analyse fonctionnelle consiste en la rédaction d'un document appelé "Cahier des Charges"; dans la plupart des cas, celui-ci se compose de pages et de pages de descriptions informelles dans lesquelles se trouvent mélangés l'énoncé d'objectifs, les stratégies pour y parvenir, des bruits et la description prématurée de choix de réalisation.

Un mauvais cahier des charges se répercute sur chacune des étapes ultérieures :

- il est fréquent que des utilisateurs soient mécontents du fonctionnement du système, ce dernier ne répondant pas à leur attente;

- lors de l'analyse de programmation et lors de l'implémentation, des difficultés surgissent et des choix sont à faire, lesquels auraient dû être réglés lors de l'analyse fonctionnelle;

- il devient illusoire de concevoir des plans de tests significatifs en vue de valider le système;

- les coûts et délais de maintenance du système deviennent prohibitifs.

La nécessité est venue de disposer d'une véritable technique permettant de poser un problème de manière systématique tout en aidant à le maîtriser. Cette technique a pris le nom en génie logiciel de "spécification". A chaque étape du cycle de vie, on peut associer une spécification; celle-ci décrivant les propriétés attendues du système en terme d'un certain nombre d'informations jugées pertinentes à ce niveau d'analyse tout en ignorant d'autres considérées comme des détails de moindre importance.

Nous considérons, dans le cadre de ce travail, la spécification au niveau de l'analyse fonctionnelle.

L'étape de spécification

Une spécification a pour rôle de fournir une définition précise du problème tout en excluant tout élément relatif à sa résolution.

L'étape de spécification est essentielle pour plusieurs raisons :

- les spécifications sont à la base du contrat liant le demandeur et les réalisateurs du système d'information;

- de bonnes spécifications peuvent grandement faciliter la mise en oeuvre ultérieure du système et permettre sa validation par rapport à celles-ci; elles réduisent le coût important des erreurs détectées en cours de réalisation et d'exploitation, et relatives à une mauvaise compréhension du problème initial [BOE,76]

- les spécifications constituent un élément de base de la documentation destinée à la maintenance ultérieure du produit réalisé.

Il faut évidemment préciser quelque peu ce que l'on entend par "bonne spécification".

On peut, à cet effet, énoncer un certain nombre de qualités; certaines d'entre elles sont étudiées plus en détail dans [MEY,80] : fidélité, cohérence, complétude, minimalité, non-ambiguïté, absence de bruit, compréhensibilité, structuration adéquate, constructibilité, modifiabilité, ...

Parmi celles-ci, quatre nous semblent fondamentales et pourtant rarement remplies par une spécification :

- la fidélité : l'énoncé du problème posé traduit celui-ci de manière adéquate;

- la compréhensibilité : cette qualité est souvent difficile à obtenir du fait de la multiplication des gens impliqués (demandeur, chef de projet, équipe de réalisation, opérateur, ...); ceci étant particulièrement vrai dans le cas où l'énoncé utilise un autre formalisme que la langue naturelle;

- la cohérence : il n'y a pas d'éléments contradictoires dans la description du problème;

- la constructibilité : il existe une démarche sous-jacente qui permet de guider l'élaboration de la spécification.

Par conséquent, il apparaît qu'écrire une bonne spécification est au moins aussi difficile qu'écrire un bon programme.

Il n'est donc pas surprenant qu'un certain nombre de méthodes, de langages et d'outils d'aide à la spécification aient récemment vu le jour.

Citons, ainsi, SADT (Structured Analysis and Design Technique [ROS,77]), SA (Structured Analysis [MAR,79]), REMORA [FOU,82] PSL/PSA [TEI,77] , Z [ABR,78] .

Ces différentes démarches peuvent être comparées selon les critères suivants :

- le degré plus ou moins formel du langage d'expression retenu;

- l'existence d'une méthode guidant le spécifieur dans la construction et la structuration de la description du problème;

- le type de modélisation retenu pour le domaine d'application considéré;

- la qualité des environnements offerts.

1.2. LES LANGAGES DE SPECIFICATION

Le but d'un langage de spécification est de fournir un cadre permettant d'exprimer d'une manière rigoureuse et précise la définition du système d'information.

Plusieurs cadres d'expression sont possibles, nous en dressons ci-dessous un inventaire :

1° Spécification en langue naturelle

Actuellement, la plupart des cahiers des charges sont rédigés en langue naturelle. Il en résulte, dans beaucoup de cas, un certain nombre d'écueils tels que bruit (éléments n'apportant d'information sur aucune caractéristiques du problème), ambiguïté (éléments permettant de comprendre une caractéristique du problème de deux façons ou plus),... On trouverea une classification de ces écueils dans [MEY,80] .

2° Spécification en langage graphique

Beaucoup d'approches (réseaux de Petri, modèle entité-association, R-nets graphiques de SREM, actigrammes et datagrammes de SADT, BDL,...) expriment la spécification d'une manière graphique. Il est, en effet, indéniable que le dessin constitue souvent un meilleur moyen de communication que le texte.

Le danger réside cependant dans le fait que, dans certain cas, le dessin suscite de la part de l'utilisateur moins de réflexion; celui-ci étant davantage "charmé" par la syntaxe que par la sémantique exprimée (celle-ci étant, de plus, souvent inexistante).

3° Spécification formattée

Dans des approches telles que Structured English (utilisée dans S.A. [MAR,79] et basée sur un pseudo-code de type PDL [CAI,75]) ou dans l'expression des spécifications externes du A7 [HEN,80], les spécifications doivent être exprimées dans un certain format bien précis; ce qui en facilite certaines vérifications.

4° Spécification semi-formelle

Des langages tels que PSL/PSA [TEI,77] , INCOD/DTE [ATZ,82] ont une syntaxe définie formellement mais pas de sémantique. A nouveau, ceci permet des contrôles encore plus poussés : simulateurs, prototypes,...

5° Spécification formelle

Un langage de spécification formel est caractérisé par une syntaxe et une sémantique définies formellement. Ces bases théoriques bien identifiées et rigoureuses peuvent être :

- mathématiques (la langage Z [ABR,78] repose sur la théorie des ensembles)

- logique mathématique (le langage développé dans [FIN,79] est basé sur le calcul des prédicats du premier ordre)

- liés à la sémantique des langages de programmation (BDL [HAM,77] , SSL [RUT,81]).

L'existence de ce cadre permet des contrôles de validation plus poussés et la possibilité de syntèse automatique (ou assitée).

*
* *

Langage naturel - langage formel

Une importante controverse s'est développée depuis quelques années quant à savoir s'il faut utiliser langage formel ou un langage naturel. Notre propos n'est pas ici de trancher en faveur de l'un ou de l'autre, mais plutôt de rappeler un certain nombre d'avantages ou d'inconvénients à l'utilisation de chacun.

1- Langue naturelle

- elle est plus facilement communicable et compréhensible
- son pouvoir d'expression est pratiquement infini

- sans une grande rigueur de la part de celui qui l'emploie, elle pose souvent beaucoup de problèmes d'ambiguïté
- les outils automatiques utilisables ne peuvent pas avoir de fonctions beaucoup plus sophistiquées que celles de traitement de textes
- il n'y a pas moyen d'appliquer de règles de vérification entre le programme et la spécification.

2- Langage formel

- certains formalismes peuvent poser des problèmes de communicabilité à des personnes non habituées à celui-ci [MIT,82] ; de plus, ils offrent parfois un pouvoir d'expression limité, ce qui peut amener à des formulations peu naturelles
- le problème de la concision [BAL,77] ; dans l'expression de certains problèmes, l'équivalent d'une ligne de langue naturelle nécessite parfois plusieurs lignes de langage formel; pour pallier à cet inconvénient, il est important que le langage formel dispose de concepts bien adaptés au domaine d'application traité (par exemple, pour la gestion, les concepts introduits dans BDL ou SSL) ainsi que des mécanismes d'extension puissants
- la définition du langage formel nécessite de recourir tôt ou tard à la langue naturelle
- l'utilisation d'un cadre formel permet l'application de traitements automatiques très sophistiqués (cohérence, complétude, maquettage,...)
- il y a moyen d'appliquer des règles de vérification entre le programme et la spécification.

1.3. LES METHODES DE SPECIFICATION

Depuis quelques années, beaucoup d'approches ont vue le le jour en matière de spécification.

Un certain nombre d'entre elles peuvent être qualifiées de"passives".

En effet, elles reposent sur l'utilisation de langages permettant d'exprimer sous une forme précise un (ou plusieurs) modèle(s) du système d'information mais elles n'assistent pas vraiment l'analyste dans la construction d'un tel modèle.

A l'opposé, nous qualifierons d'"actives" des approches présentant, en plus d'un langage et d'un modèle, une démarche méthodologique favorisant la construction systématique du modèle. C'est à ces dernières que nous allons nous intéresser dans cette partie.

Notre propos n'est pas ici de dresser un panorama exhaustif de toutes les méthodes proposées mais plutôt passer en revue quelques points les différentiant dans leur manière d'aborder l'analyse d'un problème.

C'est ainsi que nous aborderons successivement les points suivants :

- à quel moment est prise en compte la spécification de la dynamique ?
- quelles sont les interactions possibles entre la spécification des traitements et celles des données ?

- faut-il adopter une approche ascendante ("bottom-up") ou descendante ("top-down") dans la conception des modèles de traitements et de données ?
- faut-il être guidé, dans la spécification de traitements par l'analyse des données (inductif) ou par l'analyse des résultats (déductif) ?

1.3.1 Prise en compte de la dynamique

Assurément, lors de la présentation finale à l'utilisateur de un ou plusieurs scénarios [MIT,82] alternatifs de fonctionnement du système, des éléments de description de la dynamique doivent figurer. La question est de savoir si dans l'analyse, on va s'intéresser d'abord à la description de séquences d'enchainements de traitements ou bien d'abord, à la description logique des traitements en ignorant, dans un premier temps, toutes ces contraintes liées au flux de contrôle.

Un exemple illustrant la première approche est proposé dans [MAS,82] où la construction de la spécification se fait de manière itérative; à chaque étape, le spécifieur propose aux utilisateurs un scénario global de fonctionnement du système et à partir de celui-ci, un dialogue s'engage, lequel aboutit sur la proposition d'un nouveau scénario. Lorsque le scénario définitif est trouvé, chacune des fonctions est alors décrite.

Un des avantages de cette approche est qu'elle associe à chaque étape l'utilisateur dans la conception du système du fait de la bonne communicabilité des scénarios.

Par contre, la découpe dictée par les contraintes organisationnelles peut gêner par la suite la description de la sémantique de chaque traitement; celle-ci devenant parfois artificiellement compliquée.

Cette démarche nous paraît intéressante dans la spécification de problèmes bien précis, tels que :

- la description de systèmes dont les fonctions sont simples mais où les contraintes temps réel sont nombreuses (SREM [ALF,77] concerne également ces systèmes);
- la description de petits systèmes de gestion où la sémantique des traitements est assez élémentaire.

1.3.2 Spécification des traitements - spécification de données

Le fait de privilégier la description du modèle des données par rapport à celui des traitements constitue un des points importants différentiant les démarches proposées.

A. La description des traitements avant celle des données

Dans des approches reposant sur l'utilisation des graphes de circulation, on s'intéresse uniquement à la structuration et à la description des traitements sans prendre en compte véritablement la description des données, il en résultat certains inconvénients, tels que saisie multiple de mêmes données, redondance dans le stockage des données, risques d'incohérence du fait de données se trouvant réparties dans plusieurs fichiers...

Le manque de structuration et d'intégration des données est également un des problèmes posés par certaines approches fonctionnelles (c'est-à-dire, dont la description est guidée par les fonctions à assurer) dans lesquelles l'aspect données n'est pas pris en compte ou très tard (par exemple : BDL [HAM,77] , SA [MAR,79]).

B. La description des données avant celle des traitements

Ce type d'approche (qualifiée également d'approche "base de données") vise à éliminer les inconvénients de l'approche présentée ci-dessus en conduisant à décrire une représentation unique et non redondante des données manipulées dans le système d'information.

Dans une approche telle que Merise [TAR,78] (où, par ailleurs, il n'y a pas de modèle de traitements), la description des données se fait d'une manière tout-à-fait indépendante des traitements.

Un modèle ainsi obtenu est parfois la source de difficultés telles que :

- dans certains cas, la non-identification de certaines informations (en l'occurrence, pour Merise, d'"individus" (entité au sens de [CHE,76]) peut conduire à une complexification artificielle de la description des traitements (nous reviendrons sur ce problème dans le chapitre 5);

- sans une analyse poussée des traitements, il est parfois difficile d'identifier toutes les relations existant entre les différents "individus";

- enfin, certains "individus" ne sont identifiables que par la mise en évidence d'opérations les caractérisant.

Ce dernier inconvénient disparaît dans les approches ou l'on utilise des types abstraits; en effet, les différents concepts sont alors mis en évidence avec leurs opérations caractéristiques (qui correspondent en fait aux constructeurs et aux simplifieurs); la structuration ultérieure des traitements consistera alors à appliquer les différentes opérations sur les différents concepts et à répondre aux requêtes par la construction de fonctions de consultation (d'accès) plus ou moins sophistiquées. Par expérience, nous avons cependant constaté que dans la description de systèmes d'informations dont les concepts fondamentaux manipulés sont nombreux, il est difficile, même par un dialogue intensif avec l'utilisateur de mettre en évidence l'ensemble de ces concepts (ceux-ci n'étant pas toujours évidents pour eux-mêmes). De telles approches sont cependant assez bien adaptées dans le cas :

- de petits problèmes de gestion où les quelques concepts existants sont assez facilement identifiables par le spécifieur (cfr. les problèmes traités par Jackson dans [JAC,83]);

- de problèmes où les concepts manipulés sont familiés du spécifieur (s'il est informaticien de formation) (cfr. les problèmes d'éditeur de texte et de gestion de fichiers traités par Sufrin [SUF,81], [SUF,82] en utilisant le langage Z).

B. La description des traitements en parallèle avec celle des données

Décrire les traitements avant les données conduit souvent à une expression non redondante des données; par contre, le processus inverse risque de provoquer une complication de la description des traitements due à une mauvaise structuration des données.

Une troisième consiste à décrire conjointement les traitements et les données de manière à ce que :

- pour chaque sous-problème, on dispose d'une structure de données bien adaptée au traitement à décrire (SADT [ROS,77] fournit un cadre d'expression de cette dualité via l'élaboration conjointe de l'actigramme et du datagramme);

- pour chaque nouvelle structure de donnée, on dispose de moyens de l'intégrer avec les autres structures préalablement identifées.

Une telle approche (suivie notamment dans Ida [BOD,83] - nécessite l'existence d'outils permettant l'intégration des différentes "vues" (structures de données) identifiées [BAT,83] .

En particulier, l'utilisation d'un cadre formel puissant (tel que celui fournit par les types abstraits) devrait permettre d'identifier pour chaque nouvelle structure dé données introduite si elle ne peut pas l'exprimer en fonction d'autres structures préalablement identifiées (problèmes des librairies et de la réutilisation de connaissances [FOI,82] [BID,82]).

1.3.3 Approche ascendante - approche descendante

La plupart des approches proposées sont descendantes; l'idée est de travailler par raffinements successifs en partant d'une description générale du problème, puis à le détailler et le décomposer de plus en plus finement.

Parallèlement à ce raffinement successif, est associé également un principe d'abstraction.

Ce dernier est une facilité permettant de distinguer dans une description plusieurs niveaux de détail (niveaux d'abstraction); arrivé à un niveau, un certain nombre de propriétés sont exprimées tandis que d'autres jugées plus concrètes, sont rejetées dans des niveaux inférieurs.

Dans des approches telles que SA, SADT, PSL ou IDA, le principe retenu est un principe de décomposition permettant de considérer un composant d'un certain niveau d'être vu comme une collection de sous-composants (relation "est partie de"); ce mécanisme, appelé dans le domaine des bases de données, mécanisme d'aggrégation, permet de considérer un composant tout en ignorant les détails caractéristiques des sous-composants. L'aspect méthodique lié à ce principe doit pouvoir guider le spécifieur quant à savoir à quel niveau d'aggrégation doit être exprimée une propriété.

- Ainsi, dans IDA [BOD,83], une nomenclature hiérarchique des traitements est proposée dans laquelle les repères guidant la décomposition sont liés aux principes d'analyse d'un système organisationnel (principe de "quasi-décomposabilité" [SIM,74]), le concept d'"unité spatio-temporelle de traitement",...).

On peut reprocher au modèle du système obtenu d'être trop calqué sur une organisation particulière et par conséquent, de surimposer des contraintes de nature organisationnelle aux caractéristiques logiques intrinsègues du système.

- Dans SA [MAR,79] , seul l'aspect logique du système est dans un premier temps décrit, celui-ci est décrit à l'aide d'un diagramme de flux, les critères de raffinement suivis sont les suivants :

  . structure d'un niveau devenant trop complexe (pas plus de 7 fonctions [MIL,56])

  . suivre un partitionnement naturel

  . structure d'interface d'une fonction de transformation devenant trop compliquée.

- Dans une approche telle que High Order Software (HOS [HAM,76] , qui s'adresse davantage à la description de problèmes scientifiques) les critères de décomposition se veulent plus précis; ceux-ci reposent en effet sur l'utilisation des structures primitives de contrôles qui sont la composition, la sélection, l'itération et la récursion.

Notons que toutes ces approches utilisent le même principe de décomposition (aggrégation), il en existe cependant un autre : celui de "généralisation". (Nous reviendrons plus en détail dans le chapitre 4 sur les différents mécanismes d'abstraction).

Ce mécanisme de généralisation permet à un composant de contenir les propriétés qui ont en commun une collection de sous-composants tout en ignorant leurs individualités propres. L'approche préconisée par RMF [GRE,82] repose sur l'utilisation conjointe des deux mécanismes d'abstraction.

L'alternative à une approche descendante est l'approche ascendante : celle-ci consiste à composer des morceaux de description détaillée pour petit à petit en arriver à une description globale du système d'information débarassée des aspects concrets.

L'approche préconisée par Jackson [JAC,83] ou Dijkstra [DIJ,68] est un exemple de cette démarche ascendante; son argument est le suivant : dans une approche descendante le spécifieur, dans les premiers niveaux de sa décomposition, doit décider de la structuration du système.

Ces décisions sont capitales puisqu'elles vont avoir des conséquences importantes, or, elles sont sujettes à d'importantes erreurs puisque le spécifieur à ce moment ne connaît pas encore le système. La conséquence en est que beaucoup plus tard, ces premières décisions peuvent être invalidées et donc entraîner des retours en arrière importants.

D'autre part, une approche descendante, si elle favorise l'obtention d'une spécification complète, peut cependant, entraîner des problèmes quant à la façon de faire évoluer la spécification.

L'approche de Jackson peut paraître tout-à-fait judicieuse cependant, dans une démarche ascendante, il est important d'avoir au départ les "bons" composants élémentaires qui permettent une construction facile; or, dans un système d'information important, il n'est pas toujours facile de parvenir à dégager ces composants adéquats.

A notre avis, une approche descendante est praticable s'il est possible d'y associer à tout moment une démarche ascendante, il faut pour cela :

- pouvoir décrire de manière incomplète certains niveaux tout en assurant la possiblité de les compléter à posteriori;

- pouvoir factoriser a posteriori des caractéristiques communes à différents niveaux.

Les conditions de telles transformations nécessitent l'expression rigoureuse des décisions qui ont permis de passer d'un niveau à un autre [WIL,82] .

1.3.4 Démarche inductive - démarche déductive

A chaque niveau de l'analyse, deux stratégies sont possibles :

- soit, partir d'une analyse des données du problème considéré (approche inductive)

- soit, partir d'une analyse guidée par les résultats à produire (approche déductive).

La plupart des approches adoptent un point de vue inductif, celui-ci paraît assez naturel et facile à appliquer; cependant, il peut engendrer des problèmes de complétude de la spécification.

En effet, partir des données et introduire des données intermédiaires jusqu'à atteindre les résultats ne garantit pas que tous les résultats que l'on veut atteindre seront nécessairement atteints et décrits.

Par contre, une approche déductive comme celle préconisée dans BDL [HAM,77] ou [PAI,79] semble davantage convenir, en particulier dans l'analyse des premiers niveaux de décomposition où, à ce stade, l'utilisateur connaît en général mieux les résultats que l'on cherche à obtenir que les données de départ.

1.4. LES MODELES DE DESCRIPTION D'UN SYSTEME D'INFORMATION

1.4.1. Le concept de modèle

Le but d'une spécification est de donner une description aussi complète et fidèle que possible de tous les aspects constitutifs d'un système d'information (traitements, données, dynamique,...). Il va de soi que la représentation obtenue n'est pas neutre; elle reflète la vision subjective qu'a le spécifieur du système d'information (ce que les concepteurs de base de données appelent le "réel perçu").

De la partie du monde réel dont on désire faire la représentation, deux catégories de connaissances peuvent être identifiées[LUN,83] : la connaissance "concrète" et la connaissance "abstraite".
La connaissance concrète est relative à la description d'occurrences de faits survenant dans le monde réel, par exemple :
- Jean gagne 1000 Frs,
- sur un salaire de 1000 Frs, on retient, à titre d'impôts, 110 Frs.

La connaissance abstraite est relative à la description de types de faits (cette notion permet de considérer les caractéristiques partagées par les occurrences d'un ensemble de faits tout en ignorant leurs différences individuelles), par exemple :
- tous les employés ont un salaire
- l'imposition d'un salaire correspond au prélèvement de 11 % de ce salaire.

La représentation des faits que nous cherchons à obtenir est la description de la connaissance abstraite. Dans le cas d'une base de données, cette expression abstraite est appelée "schéma conceptuel" [BEN,76] ou "information model" [ISO,81] ; dans le cas de programmes, elle est appelée "spécification" [ABR,78] ,[PAI,79],[FIN,79] .

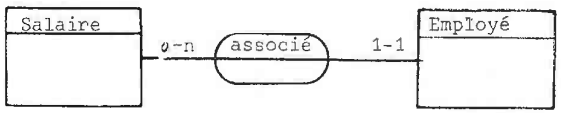
Pour construire cette représentation, il est nécessaire de disposer d'un ensemble de concepts bien adaptés au domaine auquel appartient le problème étudié. En logique mathématique, cet ensemble de concepts constitue le "système formel" (un exemple en est le calcul des prédicats du premier ordre); dans le domaine des bases de données, il constitue le "modèle conceptuel" (un exemple en est le modèle entité-association).

Construire la représentation abstraite dans un système formel choisi, revient à énoncer un certain nombre de formules valides dans ce système; par exemple :

- dans le calcul des prédicats du premier ordre :

$\forall$ x (l'employé (x) $\Rightarrow \exists$ y (salaire (y) et est-salaire(y,x))

- dans le modèle entité-association :



La représentation abstraite obtenue ainsi que le système formel choisi doivent avoir un certain nombre de qualités, telles que :

- La minimalité : Il s'agit de construire des spécifications qui présentent les propriétés intéressantes des concepts rencontrés à l'exclusion de tout élément de réalisation. La non-expression de telles sur-spécifications est étroitement liée aux concepts contenus dans le système formel. (Ainsi, le choix du modèle physique des accès comme modèle conceptuel d'expression des données ne convient évidemment pas).

- La facilité d'utilisation : Il est fondamental que les concepts manipulés soient adaptés au domaine d'application considéré. La bonne adéquation de ces concepts permet d'assurer plus facilement le contrôle de fidélité de la spécification.

- La cohérence : La représentation abstraite est dite cohérente (ou encore consistante) si elle ne comprend pas de contradictions au niveau des définitions des objets manipulés. La non-consistance d'une spécification implique la non-existence de modèle (au sens donné par la logique mathématique) de cette spécification; cela signifie qu'il n'existe pas, dans la réalité, de faits vérifiant cette spécification; la représentation que l'on a construite est, par conséquent, mauvaise, par exemple :

$\forall$ x (secrétaire(x) $\Rightarrow$ âge(x) $\leq$ 35)

et $\forall$ x (secrétaire(x) $\Rightarrow$ âge(x) $\leq$ 60)

- La complétude : Une spécification décrit complètement un problème si elle définit précisément toutes les propriétés caractéristiques des objets manipulés. Pour toute question qui se pose à un utilisateur à propos d'un objet manipulé dans le monde réel, on doit pouvoir lui répondre si cet objet possède la propriété ou non.

Un des premiers modèles conceptuels utilisé pour spécifier un système d'information est le graphe de circulation des informations (à titre d'exemple, citons [ORIG [MAL,71]]). Dans une telle approche, la circulation de l'information est décrite à l'aide de "procédures", chacune définie comme une réponse à la survenance dans l'organisation à des "événements". Chaque procédure met en évidence des "interventions" (actions) accomplies par différents "agents d'activité", elles sont liées par un cheminement d'information ou par l'intermédiaire de "documentations manuelles" ou automatiques.

Un tel schéma a le mérite de donner une vue globale du système d'information (en particulier, il constitue un excellent outil de communication avec les utilisateurs). Cependant, considéré comme outils de conception, il présente des inconvénients tels que :

- la sémantique des "interventions" est généralement peu, voire pas décrite

- aucune structuration des données n'est présentée

- rien n'est dit sur les conditions de déroulement en parallèle

- l'expression de la dynamique est insuffisante.

.....

Le fait de vouloir intégrer la description de toutes les composantes du système d'information au sein d'une même description a abouti à en donner pour chacune une description incomplète et ambigüe.

Depuis, bon nombre de modèles conceptuels, couvrant, tantôt l'un, tantôt l'autre aspect de la description d'un système d'information sont apparues.

Dans les pages qui suivent, nous essayons de dresser un panorama des différents cadres formels utilisés :

- pour la description abstraite des traitements :

  . d'une part l'aspect statique,

  . d'autre part, l'aspect dynamique (cette dernière partie étant moins développée, l'expression de la dynamique n'étant pas prise dans notre travail);

- l'expression abstraite des données.

Enfin, nous évoquerons quelques approches visant à intégrer davantage la description des traitements et celle des données.

1.4.2. Panorama des modèles de description des traitements

A. Description statique des traitements

1) Le diagramme de flux (Data Flow Diagrams: D.F.D.)

Le diagramme de flux vise à une description logique des flux de données à travers un système.

Il présente un graphe dont :

- les sommets sont : . des fonctions, chacune identifiant une transformation sur les données;
  . des supports de mémorisation (pas nécessairement matériels) chacun de ceux-ci fournissant un moyen de stockage pour les données.
- les arcs constituent des pipelines; chacun définissant un flux de données.

Remarque : Lorsque plusieurs flux se dirigent vers une même fonction, on ajoute parfois des annotations exprimant certaines interdépendances entre ces flux.

Dans le cas de système d'information de taille importante, tout ne peut être exprimé à l'aide d'un seul graphe. Dès lors, le système est modélisé sous la forme d'une hiérarchie de sous-systèmes; chaque sous-système correspondant à un diagramme de flux.

Dans les premiers niveaux, les transformations sont assez grossières et seuls quelques flux significatifs sont identifiés; ensuite, peu à peu, les transformations se précisent tandis que tous les flux existants sont identifiés.

L'analyse du modèle obtenu permet la détection de différentes erreurs.

D'une part des omissions de définitions (incomplétudes), par exemple, un flux d'entrée, présent à un niveau, est absent au niveau inférieur.

D'autre part, des erreurs de cohérence (consistance) telle que la règle du "balancement" (tous les flux en entrée et en sortie relatifs à un niveau doivent être compatibles avec le flux d'entrée et de sortie de la fonction de transformation du niveau supérieur qui a fait l'objet du raffinement).

Par expérience, nous avons constaté que l'utilisation du diagramme de flux comme outil de communication avec le demandeur est très utile (en particulier, les problèmes d'adéquation de la spécification se trouvent facilités par une présentation proche d'un scénario de fonctionnement).

Par contre, lors de la construction du modèle, les problèmes suivants se posent :

- souvent, des séquencements liés à l'exécution (flux de contrôle) viennent se superposer aux séquencements liés aux données (flux de données);
- il est parfois difficile pour le demandeur de ne pas associer aux supports de mémorisation des supports physiques qu'il pense avoir identifiés;
- lors de la description des fonctions de transformations terminales (c-à-d non raffinées par un nouveau graphe), la confusion peut régner quant à savoir si la donnée d'entrée est un objet ou une suite d'objets.

Un exposé plus détaillé sur le diagramme de flux est présenté dans [WEI,78] ; il est utilisé, entre autres, dans les approches suivantes : BDL ("Business Definition Language" [HOW,75]), SADT [ROS,77] , SA [MAR,79] .

2). La hiérarchie fonctionnelle

Le premier type d'approche suggérée repose essentiellement sur l'identification et l'analyse des différents flux de données; c'est en les décrivant que, progressivement, sont identifiées les fonctions de transformation qui, à leur tour, viennent se greffer sur le modèle initial.

Dans ce second modèle présenté, la démarche suivie est sensiblement différente, en ce sens qu'elle privilégie davantage l'analyse de la fonction et de ses données d'entrée et de sortie.

Le but de ce modèle est d'organiser la description des diverses fonctions en une série de niveaux distincts et ordonnés; l'utilisation d'une structure hiérarchique permet une restriction des interrelations possibles entre fonctions et, par conséquent, est davantage maîtrisable.

Plus formellement, une structure est qualifiée de hiérarchique si pour une relation R(A,B) (A et B étant des fonctions appartenant à des niveaux différents) , on a :

- si A appartient au niveau 0 de la hiérarchie (sommet), alors il n'existe pas de B tel que R(A,B);
- si A appartient au niveau i de la hiérarchie, alors il existe un B appartenant au niveau i^{-1} tel que R(A,B); et pour tout autre C tel que R(A,C) cela signifie que C appartient à un niveau inférieur ou égal à i^{-1}.

Notons l'utilisation fréquente d'une hiérarchie particulière : la structure arborescente.

La structuration du système se fait, en général, par niveaux hiérarchiques d'abstraction (cf. [DIJ,68]). Le mécanisme de l'abstraction est une facilité permettant d'inclure à un niveau de description certaines informations tout en excluant d'autres détails jugés "trop bas niveaux" ou "moins importants".

Ce mécanisme permet ainsi à chaque niveau de raisonner tout en ignorant des propriétés et des objets de niveaux inférieurs (nous reviendrons davantage sur ce principe d'abstraction dans le chapitre 4).

Il va de soi qu'un certain nombre de contrôles élémentaires de cohérence et de complétude sont possibles, tels que : détection de données non utilisées, de non-concordance entre sorties d'une fonction et entrées dans une autre, de redondance d'une même fonction à différents niveaux de la hiérarchie... Ces contrôles peuvent être plus sophistiqués lorsque la sémantique associée au lieu entre une fonction et des fonctions plus élémentaires, est davantage explicitée (par exemple, la règle des six axiomes dans High Order Software [HAM,76]).

Ce type d'approche est bien adapté à l'analyse de système où :

- les flux de données ne sont que quelques-uns, tandis que les fonctions sont très nombreuses;
- la structuration du S.I. se surimpose complètement à une structure d'organisation existante (c-à-d à une découpe particulière des traitements) qui guide sa structuration (cf. e.g. [BOD,83]).

Un des inconvénients de cette approche réside dans le fait que, de manière générale, la description de chaque fonction n'explicite la relation qu'entre une donnée et un résultat, la perte de la notion de flux explique dans certains cas l'introduction "artificielle" de fonctions telles que : point d'accumulation, stockage temporaire d'informations...

Le concept de hiérarchie a été tout particulièrement bien étudié par Parnas [PAR,74] ; des exemples de son utilisation se trouvent dans HIPO ("Hierarchy plus imput output" [KAT,76]), "Structured Chart" [STE,74] , ISDOS [TEI,77] , RMF ("A Framework for Requirements Models" [GRE,82]), IDA [BOD,83] .

3) Les systèmes de production

Ce modèle, utilisé par ailleurs dans d'autres domaines tels que les systèmes experts (cf. par exemple [GRE,76]) se prête bien à la description de problèmes de gestion dans lesquels sont utilisées des règles dont la structure est de la forme : Condition ⟶ Action.

La spécification du système repose sur une description par "états" et "transactions", dans laquelle à chaque étape, un traitement est décrit par les actions à effectuer en fonction de la situation présente et d'une certaine combinaison de conditions externes.

Dans le cas de l'utilisation d'une table de décision, un certain nombre d'opérations sont possibles afin de "normaliser" la description obtenue (condensation des conditions, analyse et élimination de la redondance, simplification, éclatement et enchaînement de description de traitement); de plus, un certain nombre de contrôles concernant l'exhaustivité (complétude) et la cohérence sont possibles.

L'utilisation de ce modèle présente les avantages suivants :

- approche adaptée à une classe de problèmes à laquelle appartient la majorité des problèmes de gestion;
- approche simple et facile à enseigner;
- la spécification obtenue se prête à un certain nombre de vérifications.

Ses inconvénients sont les suivants :

- l'approche est difficilement utilisable dans le cas de certains problèmes de gestion

- l'enchaînement des traitements au niveau d'une table est trop procédural et algorithmique (surspécifications)

- pour la description de problèmes un peu compliqués, on manque d'outils de structuration et d'enchaînements entre les diverses tables de traitements.

L'utilisation de ce modèle se fait dans beaucoup d'approches par la structuration de la description sous forme de "règles" exprimées en langue naturelle [BOD,83] ; une expression plus standardisée est obtenue par l'emploi des "tables de décision" [POL,71] , [GAL,73] .

4) Le modèle pré/post

Un des inconvénients majeurs que l'on peut reprocher à beaucoup d'approches proposées réside dans le fait qu'elles ne se contentent pas d'exprimer quelles propriétés sont attendues du résultat mais de plus, décrivent la manière de calculer le résultat. Le modèle pré/post répond uniquement à cette première préoccupation.

Dans le modèle pré/post, la spécification d'un traitement s'exprime à l'aide d'une paire de précondition et de postconditions [FLO,67] , [HOA,69] . Une précondition décrit la condition sur les données d'entrée (messages, données locales et/ou globales) pour que le traitement s'exécute correctement, une postcondition détaille l'effet attendu d'une exécution du traitement (propriétés du résultats, éventuellement en fonction des données d'entrée).

De plus, cette paire peut être complétée de conditions devant être maintenues invariantes (propriétés devant être vérifiées avant et après tout traitement).

Pour des problèmes de taille importante, on peut en structurer la description en introduisant dans l'expression des conditions des opérations de haut niveau qui, à leur tour, seront décrites de la même manière.

A nouveau, un certain nombre de contrôles de cohérence et de complétude du type de ceux déjà décrits précédemment sont possibles; en particulier, certains des contrôles propres au modèle conditions-actions et concernant l'exhaustivité des conditions sont possibles.

Cette approche, bien que encore relativement peu utilisée dans le cadre de la description de problèmes de taille importante ([HEN,80] ,[GRE,82]) est intéressante en ce sens qu'elle fournit une spécification très abstraite (c'est-à-dire débarassée d'un maximum de détails de réalisation) qui permet de faciliter le contrôle d'adéquation.

B. Description dynamique des traitements

Les modèles de description de traitements présentés jusqu'ici introduisaient le minimum de caractéristiques dynamiques en particulier, leurs qualités essentielles sont d'être :

- statiques; c'est-à-dire, que l'on ne décrit pas l'évolution des traitements dans le temps;
- non-procéduraux; c'est-à-dire, que les seuls liens existant entre les traitements sont des liens dictés par les données.

A côté de telles considérations, il en existe d'autres qui peuvent influer sur le fonctionnement du système sans pour autant faire référence aux faits réels décrits dans le modèle statique de l'application. Ces secondes considérations sont davantage liées à des facteurs extérieurs, tels que : critères économiques ou techniques, conditions de sécurité.

Par exemple :

- la description d'un cycle particulier de mises à jour d'un fichier (par exemple, un fichier est mis à jour journellement en soirée);

- la périodicité d'un traitement en fonction de la charge particulière (par exemple, un traitement de facturation est lancé lorsque le traitement de toutes les commandes est terminé)

- la possibilité de différents traitements se déroulant en parallèle;

...

L'intégration dans le modèle initial de telles considérations conduit à la description d'un nouveau modèle dont les concepts fondamentaux sont le processus et l'événement (cf. [BOD,79] [DUF,80] [FOU,82]).

Un processus est une activité correspondant à l'exécution d'une occurrence d'un traitement (jusqu'ici, nous avons parlé en termes de "type" de traitement). Les états intéressants d'un processus sont l'état actif et l'état terminé.

Les changements d'état d'un processus sont :

- le déclenchement, correspondant à la création d'un processus

- la terminaison, survenant lorsque le processus a atteint son état final.

Ces changements d'états constituent autant d'événements.

Un événement correspond à un changement d'état dans le système.

On distingue l'événement simple et l'événement composé.

- Un événement simple peut être externe ou interne. Un événement externe correspond à un stimuli généré par l'environnement du système (par exemple, l'arrivée d'un bon de commande, le déclenchement d'une horloge interne,...)

- Un événement composé correspond à une combinaison d'événements simples éventuellement qualifiés. Un élément composé est appelé généralement "point de synchronisation".

Parmi de nombreuses approches visant à la description du modèle de la dynamique; citons :

- Les réseaux de Petri [PET,76], [PET,77] . Basée sur les concepts de "place" et de"transition" entre places, cette approche est très largement répandue bien que n'intégrant pas la description des données;

- Le R-NET (Requirements Nets). Développés par TRW [ALF,77] ils reprennent la même philosophie que les réseaux de Petri, mais en se limitant uniquement à l'utilisation des constructions de la programmation structurée;

- L'approche DSL [BOD,79]. Il s'agit d'un approfondissement des travaux mentionnés ci-dessus dans lequel une meilleure intégration est réalisée avec le modèle de description statique des traitements et le modèle des données;

- L'approche REMORA [ROL,82] [FOU,82]. Dans ces travaux, l'aspect intégration avec les traitements et les données est davantage poussé parmi l'utilisation d'un modèle dont les concepts sous-jacents sont identiques (le modèle relationnel); le plus, les problèmes de prise en compte du temps sont abordés.

1.4.3. Panorama des modèles conceptuels de description des données

A. Les dictionnaires de données

Historiquement, la première tentative consacrée à la présentation et à la définition des données a été le dictionnaire de données.

Cet effort s'est avéré tout-à-fait nécessaire dans le cas où il y a beaucoup d'informations à manipuler; en effet, dans de tels cas, les problèmes suivants peuvent surgir :

- un même nom de donnée peut recouvrir deux réalités (informations) tout-à-fait différentes (incohérence);
- deux noms de données distinctes peuvent recouvrir la même réalité (problème de redondance).

Le but d'un dictionnaire de données est de présenter un ensemble ordonné de définitions.

Une définition ne doit introduire (en principe) aucune caractéristique d'implémentation (le code ASCII est, bien entendu, à proscrire !).

La définition de chacune des données doit présenter un double aspect :

- la syntaxe : description de la structure de la donnée;
- la sémantique : signification de la donnée au sein de l'organisation.

En plus, on trouve souvent d'autres rubriques, telles que :

- La nature de donnée :

  les qualifications suivantes peuvent être données :

  . aspect temporel (durée déterminée dans le temps)
  . éléments de situation (en cours de traitement, en attente, ...)
  . éléments de service (éléments propres au fonctionnement du système comme date de création, date de de mise à jour, ...)

- L'origine de la donnée :

  c'est-à-dire le lieu de son apparition

- Le responsable de son encodage

- Ses règles de validation :

  contrôles intrinsèques subis par la donnée lors de sa création ou de sa mise à jour.

La qualité essentielle d'un tel modèle est de fournir un référentiel commun aux différents utilisateurs du système d'information.

Un certain nombre de traitements élémentaires sont possibles.

Citons en exemple :

- la détermination d'éléments s'auto-définissant;
- la détermination de noms de données désignant une même information;
- l'élimination des redondances;
- l'élimination des polysémes (deux informations ayant le même nom et des sens différents).

L'inconvénient majeur réside dans le manque de structuration systématique.

Il paraît évident que cette approche ne peut se concevoir qu'en complément d'un autre modèle dans lequel une structuration plus poussée existe ainsi que la présence de liens sémantiques entre différentes catégories d'informations.

B. La structure logique des données

Dans le dictionnaire des données, une donnée est en général définie par son lien (produit cartésien, itération) avec d'autres données plus élémentaires.

Ce type de structuration unidimensionnelle ne suffit cependant plus dans le cas où les données sont nombreuses; une structuration multi-dimensionnelle devient alors nécessaire.

Le modèle logique propose un moyen de décrire de telles structures, sans avoir à introduire de redondance (c'est-à-dire une même information se trouvant dupliquée à plusieurs endroits); ainsi l'utilisation de la notion de clé d'accès et de chemin d'accès répond à ces soucis; sans pour autant à avoir à s'occuper de problèmes plus techniques liés à l'implémentation physique de ces techniques d'accès.

Les composants de ce modèle (par ailleurs beaucoup utilisé dans des approches américaines sous le nom de "Data Structure Diagram", e.g. [WEI,78] , [MAR,79]) sont généralement les suivants :

- Le type d'article correspond à la classe générale d'éléments pour lesquels de l'information est enregistrée (on peut aussi distinguer des "sous-articles", c'est-à-dire des ensembles particuliers d'éléments à l'intérieur d'un article);

- L'attribut; chaque article se compose d'informations élémentaires appelées attributs; un attribut correspond à une propriété possédée par l'article;

- L'attribut-clé ou clé d'accès correspond à un attribut dont la valeur prise permet d'identifier une occurence d'un article;

- L'attribut pointeur est un attribut utilisé comme pointeur vers un autre ensemble particulier d'informations (article).

Bien souvent, dans le cas de description de S.I., ce modèle ne sera pas unique et obtenu directement, mais sera la résultante de l'union de plusieurs modèles partiels. Un certain nombre de contrôles de cohérence et de complétude peuvent être faits à l'occasion de cette intégration.

Le gros défaut que l'on peut reprocher à ce modèle logique est que, non seulement, il permet de décrire le réel perçu (le "quoi") mais, de plus, il intègre un certain nombre de paramètres de réalisation de la structure de données (par exemple, la notion d'attribut pointeur).

Ces considérations informatiques (le "comment") ne peuvent que compliquer la perception de la description des données par les demandeurs; c'est ainsi que le rapport ANSI/SPARC [ANS,75] propose, pour faire la différence entre l'utilisation des données (c'est-à-dire les fonction de l'organisation) et le fonctionnement du système informatique (c'est-à-dire le système de gestion de la base de données) de découpler le processus de construction de la description des structures d'informations en un modèle conceptuel des données et un modèle logique.

Par la suite, nous allons nous attacher à la présentation de deux exemples de modèles conceptuels.

C. Le modèle relationnel

Le modèle de données proposé dans le modèle relationnel repose sur une présentation des données sous la forme de différentes tables. Chaque table correspond à l'expression d'une relation [COD,70] .

La définition mathématique de cette relation est la suivante :

Soient $D_1, \ldots D_n$: n ensembles, une relation R sur ces n ensembles est un sous-ensemble du produit cartésien $D_1 \times D_2 \times \ldots D_n$

Une occurence de la relation correpond par conséquent à un n-uplet $\langle d_1, \ldots, d_n \rangle$ tel que $d_1 \in D_1, \ldots d_n \in D_n$.

Pour désigner de manière non ambigüe les ensembles D_i, on utilise n noms distincts $A_1, A_2, \ldots A_n$ appelés attributs de la relation (sélecteurs de champs dans le produit cartésien).
D_i est appelé le domaine de l'attribut A_i.

Un schéma de relation définit une classe de relations de même forme; pour appartenir à un schéma, une relation doit en générale respecter un certain nombre de règles appelées contraintes d'intégrité.

Les différentes règles (dont on trouvera un exposé plus précis dans [DAT,77] ou [DEL,82]) peuvent être exprimées sous la forme :

- de dépendance entre attributs :
  - . notion de clé : deux n-uplets quelconques ont des valeurs de clés distinctes
  - . notion de dépendance fonctionnelle entre attributs
  - . notion de dépendance multi-valuée.

- de prédicats :

ceux-ci font intervenir les opérateurs de l'algèbre relationnelle (projection sur un attribut de la relation, intersection, union, différence, complémentation, jointure (c'est-à-dire construction d'une relation à partir de relations ayant des caractéristiques communes)).

Des améliorations ont été apportées à ce modèle.

Citons en particulier, celles proposées dans le projet REMORA [FOU,82] : l'intégration du temps par la définition d'un nouveau type de dépendance fonctionnelle; l'introduction de types de relations permettant de structurer davantage le modèle.

Signalons enfin l'existence de modèles particuliers où les seules relations acceptées sont des relations binaires (Data Semantics de Abrial [ABR,74] , Foral de Senko [SEN,75]).

Un avantage important du modèle relationnel est qu'il bénéficie d'un cadre algébrique formel bien étudié [COD,71] ; celui-ci favorisant des contrôles de cohérence et de complétude plus poussés.

En particulier, des propriétés intéressantes peuvent être étudiées à partir de relations mises sous une forme normalisée.

Ce modèle a été utilisé pour décrire une grande variété d'applications très différentes. En particulier, des approches telles que SBA ([DEJ,75], [ZLO,77]) indiquent la facilité d'utilisation du concept de table dans la description de petites applications de gestion.

Il semble néanmoins que dans le cadre de la description de larges systèmes, il serait assez utile de disposer de mécanismes de structuration encore plus puissants.

D. Le modèle entité-association

Dérivé du modèle relationnel, le modèle entité-association ([BEN,76] , [CHE,76]) apporte un niveau de structuration supplémentaire en distinguant les relations représentant des objets de celles représentant des associations.

Les données sont structurées en entités, associations et propriétés dont les définitions sont les suivantes ([TAR,79] , [BOD,83]) :

- Entité :

  C'est une famille d'objets concrets ou abstraits du monde réel :

  . caractérisée par un même ensemble de propriétés quantitatives et qualitatives
  . ayant une existence propre, c'est-à-dire pouvant être définie sans faire référence à un autre objet
  . considérée comme un tout par un individu (ou un groupe d'individus.

- Association :

  C'est une relation qu'un individu (ou un groupe d'individus) perçoit entre deux ou plusieurs entités. A partir de l'association, le concepteur peut définir des propriétés qui n'ont un sens que par rapport à l'association de ces entités.

- Propriétés :

  C'est une qualité que confère un individu (ou un groupe d'individus) à une entité ou à une association.

Ces différents composants peuvent faire l'objet de restrictions exprimées à l'aide de contraintes d'intégrité :

- Sur les propriétés :

  Elles peuvent être d'ordre syntaxique (formats d'écriture) ou sémantiques (restriction sur les valeurs prises);

- Sur les entités :

  Elles peuvent être relatives au nombre d'occurences minimum et maximum de chaque entité;

- Sur les associations entre entités :

  Les connectivités expriment le nombre de fois maximum et le nombre de fois minimum qu'une même occurence d'une entité peut apparaître dans les occurences de l'association; on peut, en outre, mentionner des contraintes d'intégrité fonctionnelle indiquant l'existence d'une dépendance fonctionnelle.

A partir d'un modèle entité-association, il est possible de faire un certain nombre de contrôles tels que :

- Vérification :

  Il s'agit de s'assurer que dans chaque occurence d'entité ou d'association, on ne trouve qu'une seule valeur de chaque propriété.

- Normalisation :

  Elle permet de s'assurer que chacune des propriétés ne peut être vérifiée sur un sous-ensemble de la collection de l'association.

- Décomposition :

  Elle permet d'éclater une association de dimension n en plusieurs associations de dimensions moindre, sous perte de sémantique, en utilisant les dépendances fonctionnelles existant sur l'association.

Signalons quelques enrichissements apportés au modèle entité-association tels que :

- Une possibilité supplémentaire de hiérarchisation des données par l'utilisation des mécanismes suivants [ATZ,82] :
  . la définition d'une entité comme étant un sous-ensemble d'une autre entité ("Subset Relationship");
  . le principe de "généralisation" permettant de factoriser les propriétés communes de plusieurs entités.

- L'expression des contraintes d'intégrité davantage sophistiquées exprimées à partir d'une structure des occurences dérivée du modèle entité-association.

Sans nul doute, le modèle entité-association apparaît actuellement comme étant le plus intéressant quant à son utilisation au stade de la spécification des structures de données; en particulier, une grande partie de son intérêt réside dans le fait qu'il apparaît comme un outil privilégié de dialogue avec les utilisateurs (présentation d'une vue synthétique, simplicité des concepts manipulés, ...).

Certaines difficultés subsistent cependant dans la construction d'un tel modèle; citons par exemple :

- la difficulté, dans certains cas, de faire la distinction entre entité et association, ou entre propriété et entité;
- le fait de devoir exprimer une propriété comme étant caractéristique d'une entité alors qu'elle n'est caractéristique que de certaines occurences de l'entité;
- l'expression du "temps"; il est peu aisé d'exprimer la conservation d'un historique des occurences d'un individu.

E. Le modèle des réseaux sémantiques

Une réseau sémantique permet de représenter, au moyen d'une même structure de données, différents types d'informations constituant une banque de connaissances. Un réseau sémantique se présente comme un graphe exprimant les différentes relations définies entre entités pertinentes au domaine d'application considéré.

RMF (A Framework for Requirements Models [GRE,82]) constitue un exemple d'utilisation d'un réseau sémantique dans le cadre de la spécification d'un système d'information. On distingue trois types de noeuds :

- les objets : ils sont associés aux éléments du monde réel (client, bon de commande,...);

- les activités : il s'agit des différentes fonctionnalités assurées par le système (facturation, mise à jour stock,...);

- les assertions : il s'agit de prédicats booléens devant être vérifiés pour déclencher certaines activités.

Un certain nombre de relations sont possibles entre les différents noeuds; des exemples de tels liens sont :

- la relation entre une activité et ses données et résultats (objets);

- la relation entre une activité et l'assertion associée à son déclenchement;

- les relations exprimant les hiérarchies existantes entre les concepts manipulés ("part of", "sub-class of",...).

1.4.4. Vers une intégration des modèles de description des traitements et des données

Jusqu'ici, nous avons présenté de manière séparée le modèle des traitements et celui des données.

Il va de soi qu'une telle séparation est tout-à-fait abusive; en effet, la définition d'une information n'est pas seulement exprimée au travers de la définition de sa structure de données et de ses relations avec d'autres informations, mais également, au travers de la définition des fonctions qui créent, modifient et manipulent cette information.

Quelques approches tentent d'apporter une réponse à ce problème en proposant au spécifieur des modèles permettant de décrire conjointement les traitements et les données (citons, par exemple, SADT [ROS,77] , SA [MAR,79] ou IDA [BOD,83]).

En plus, des contrôles de cohérence et de complétude propres à chacun des modèles, un certain nombre de contrôles "croisés" supplémentaires peuvent avoir lieu (toute donnée utilisée dans une fonction doit apparaître dans le modèle des donnnées; dualité actigramme / datagramme dans SADT,...)).

Il est à noter cependant que les contrôles évoqués ci-dessus sont assez élémentaires. Cela semble dû au fait que les fonctions terminales définies dans le modèle de traitement ne spécifient pas assez précisément et suffisamment les transitions d'un état à un autre subies par les données concernées.

A. La modélisation par transactions

Dans une approche telle que ACM/PCM (Active and Passive Modelling [BRO,81] , [BRO,82]), l'accent est davantage mis sur les interactions possibles entre données et traitements.

Au niveau des traitements, ceux-ci s'expriment sous forme d'un certain nombre de transactions. A chaque transaction est associé un événement déclencheur (cfr. modèle de la dynamique); une transaction renferme un certain nombre d'"actions" ; une action (définie de manière pré/post) étant relative à une donnée particulière et modifiant l'état de cette donnée.

Le modèle des données se présente comme un modèle hiérarchique où les liens suivants sont susceptibles d'unir des données appartenant à deux niveaux distincts· le produit cartésien, l'union et la suite.

Lors de l'élaboration de ces modèles, plusieurs contrôles sont possibles, tels que :

- la composition des actions doit être compatible avec les relations unissant les données manipulées;
  (par exemple, un traitement basé sur une itération doit porter un ensemble répétitif de données);

- le déroulement des actions sur une donnée est soumis à certaines vérifications (par exemple, la suppression d'une donnée ne peut se faire que si les données associées sont également supprimées).

Notons cependant que la description "pré/post" de chaque action figure dans la description de la transaction et non dans la description de la donnée; la conséquence en est la dispersion "géographique" des propriétés de la donnée.

Le caractère local des définitions est une caratéristique intéressante que l'on retrouve dans JSD (Jackson System Development [JAC,83]).

En effet, le modèle de donnée y est constitué de différentes entités où chaque entité décrit une réalité appartenant au monde extérieur; une entité exerce ou subit un certain nombre d'actions.

La description de chaque entité comprend cette liste d'actions et la description de chacune d'entre elles.

Un tel modèle, dans lequel chaque objet est caractérisé par ses opérations, s'apparente à un modèle de type abstrait. Comme nous allons le voir, le modèle type abstrait se distingue du modèle Entité/Actions par des possibilités plus riches de structuration et par l'existence d'un cadre formel davantage étudié.

B. Le modèle type abstrait

De façon intuitive, nous pouvons dire qu'un type abstrait permet la définition d'un ensemble d'objets par un ensemble d'opérations (définies par des spécifications syntaxiques et sémantiques) servant à manipuler ces objets [LIS,75] .

Nous allons présenter ici la notion de type abstrait algébrique [GUT,77] , [ADJ,78] ; nous indiquerons ensuite des avantages associés à l'utilisation de ce cadre.

La spécification d'un type abstrait peut se présenter sous la forme d'un triplet (S,Σ,R) où :

- S est une liste de symboles de domaines et codomaines appelés sortes et contenant une sorte d'intérêt (type décrit);

- Σ, appelé signature est une liste de noms d'opérations munies de leur profils (domaines et codomaines de l'opération); la sorte d'intérêt doit apparaître dans chaque profil;

- R est une liste d'équations entre des termes, construits en utilisant les symboles de Σ et des symboles de variables quantifiées universellement. Les équations permettent la caractérisation des opérations à partir d'autres, appelées constructeurs primitifs.

A titre d'exemple, voici la spécification du type FILE

Le type FILE paramétré par le type ELEM est représenté par

$< \text{FILE}, \Sigma_{\text{FILE}} , R_{\text{FILE}} >$

où ELEM est une sorte formelle

Σ_{FILE} = Fvide, Ajout, Sup , Tête, Vide?

Profils

Fvide : → FILE	% File vide %
Ajout : FILE X ELEM → FILE	% Adjonction en queue de file %
Sup : FILE → FILE	% Suppression en tête de file %
Tête : FILE → ELEM	% Accès en tête de file %
Vide? : FILE → BOOL	% Prédicat indiquant si une file est vide %

R_{FILE} contient les équation suivantes

où f est une variable de type FILE, e de type ELEM

1. Sup(Fvide) = Fvide
2. Tête(Fvide) = $\perp$
3. Vide?(Fvide) = Vrai
4. Sup(Ajout(f,e)) = si Vide?(f) alors Fvide
   sinon Ajout(Sup(f),e)
5. Tête(Ajout(f,e)) = si vide?(f) alors e
   sinon Tête(f)
6. Vide?(Ajout(f,e)) = Faux

Notons les caractéristiques suivantes :

- la spécification du type FILE nécessite en fait l'introduction d'un environnement incluant la spécification

$$SPEC_0 = \langle S_{BOOL}, \Sigma_{BOOL}, R_{BOOL} \rangle \text{ avec } \Sigma_{BOOL} :$$

$$\{non, et, ou, ou_{ex}, \omega_{Bool}, vrai, faux\}$$

- les opérations peuvent se classifier en constructeurs (Fvide,Ajout), simplificateurs (Sup) et fonctions d'accès (Vide?,Tête).

Un certain nombre de mécanismes permettent la construction incrémentale de la spécification :

- le mécanisme d'extension permet la construction d'une spécification à partir d'une spécification plus simple par adjonction de nouvelles opérations;

- le mécanisme de restriction, par contre, conduit à la suppression d'opérations;

- les opérateurs de construction paramétrée sont des opérations d'un usage si général qu'il en est donné une définition standard (le spécifieur n'a pas chaque fois à les redéfinir).

  Associés au mécanisme de paramétrisation [FIN,79] [ADJ,82] , ils constituent de puissants outils de construction.

Le cadre formel associé au type abstrait algébrique permet la vérification de nombreuses propriétés dont :

- la cohérence : appelée plus souvent pour les types abstraits "consistance". Les différentes équations sont prouvées non contradictoires si on ne peut prouver à partir d'elles, la formule Vrai = Faux.

- la complétude : tout terme généré par application des opérations peut être réduit à la valeur Vrai ou à la valeur Faux si le résultat du terme est un booléen ou, si ce n'est pas le cas, à un terme appelé"Forme Normale" du type (c'est-à-dire, l'expression de tous les termes du type à partir des constructeurs).

Signalons que certaines de ces propriétés sont démontrables par utilisation de systèmes de réécriture.

Dans [LEV,84] , d'autres propriétés du cadre algébrique des types abstraits sont indiquées.

Il n'y a encore jusqu'ici que peu d'applications des types abstraits à la spécification de gros problèmes (des exemples sont : [HEN,80] ou [CHA,82]); ce cadre devrait cependant être davantage utilisé par la suite du fait qu'il permet d'assurer :

- la minimalité (pas de surspécifications) en distinguant dans la description d'un objet l'expression des propriétés intrinsèques de l'expression des détails de représentation de cet objet;

- un regroupement "géographique" de toutes les opérations caractéristiques au type; en conséquence, la maintenance de la spécification en sera facilitée puisque les conséquences d'un changement seront facilement localisables;

- un cadre formel dans lequel peut être validé le passage de la spécification à la résolution du problème par l'introduction de différents niveaux de conception [DER,79] .

Concernant leur utilisation, nous avons constaté que pour un spécifieur peu habitué, il est assez difficile de décrire la sémantique des opérations en utilisant des équations récursives.

En particulier, pour certains problèmes, la liste des équations peut être assez longue et fastidieuse à élaborer et à valider alors que seules, quelques équations sont pertinentes par rapport à la caractérisation désirée.

Un autre problème est lié à la forme récursive des équations; en effet, une telle forme pose parfois des problèmes de communicabilité de la spécification avec l'utilisateur.

Une solution pour surmonter ce problème est proposée dans [FIN,79] où la spécification de chaque opération se fait à l'aide de pré/post conditions.

1.5. LES SYSTEMES D'AIDE A LA SPECIFICATION

Dans le cas de description d'un système d'information, les spécifications sont si importantes qu'il est nécessaire de disposer d'outils permettant de les gérer et de les exploiter.

En particulier, nous pensons que la meilleure méthode et le meilleur langage (s'ils existaient !) ne pourraient être vraiment praticables dans un milieu industriel que s'il sont accompagnés d'outils pertinents.

Notre but n'est pas ici de faire une étude exhaustive de toutes les aides existantes mais plutôt de mentionner quelques outils typiques (le lecteur intéressé pourra se référer à [LAM,82] ou [WAS,82] pour une description précise)

La première catégorie d'aides est relative à la gestion et au contrôle des spécifications; outre la création et la modification de texte, ces outils assurent un certain nombre de contrôles de complétude et de cohérence plus ou moins poussés.

Citons parmi les logiciels proposés :

- Isdos pour le langage PSL [TEI,77]
- Mentor pour tout langage BNF [DON,80] [MED,81]
- des outiles graphiques développés pour supporter SA [DEL,82]
- SAS pour supporter SADT [LIS,83] .

Remarquons que ces outils pourraient être encore enrichis en guidant davantage le spécifieur selon une méthode retenue (notion de pilote) ou encore en gérant la réutilisation de spécifications préexistantes (notion de base de connaissances [FOI,82]).

La deuxième catégorie d'outils est relative à l'évaluation des spécifications obtenues; cela peut concerner :

- l'adéquation des spécifications aux besoins de l'utilisation par un système de maquettage ([MAS,82] , [MIT,82] [WAS,82]) ou d'évaluation symbolique

- les mesures de faisabilité des spécifications par rapport aux ressources disponibles et aux performances désirées par une simulation (par exemple, [BOD,78] [DUF,80]).

Pour des spécifications rédigées en langue naturelle, un traducteur en langue naturelle permet de s'assurer plus facilement de l'adéquation des spécifications aux besoins du spécifieur [BAL,77].

Chapitre 2
IDEES GENERALES CARACTERISANT L'APPROCHE PROPOSEE

2.1. LA SPECIFICATION D'UN SYSTEME D'INFORMATION

Nous pensons qu'il est possible de décomposer le travail de spécification en deux tâches :

- D'une part, la spécification structurée des besoins énoncés par le demandeur (habituellement, un certain nombre de pages de descriptions informelles dans lesquelles se trouvent entremêlées des considérations quant aux objectifs poursuivis, aux stratégies pour y parvenir mais également des bruits et l'expression de choix prématurés).

- D'autre part, la formulation de un ou plusieurs scénarios de fonctionnement [MIT,82] de systèmes opératoires pouvant être proposés aux demandeurs.

Concernant la première tâche, nous proposons l'approche suivante [DUB,82] :

1° Définir un noyau du système de manière déductive; le noyau doit inclure toutes les caractéristiques apparaissant comme inhérentes à l'application considérée;

2° Greffer sur ce noyau un certain nombre de caractéristiques apparaissant comme dépendantes de l'environnement organisationnel spécifique à considérer; ces caractéristiques reflètent des règles et habitudes particulières qui sont souvent sujettes à des changements;

3° Surimposer enfin certaines contraintes additionnelles relatives aux performances attendues, à la protection de certaines informations, à la sécurité du système face à certaines pannes,...

Dans le cadre du travail proposé, nous nous sommes concentrés davantage sur le premier point, à savoir la formulation et la structuration de la spécification du noyau.

Concernant les autres points, nous ne nous sommes pas consacrés à leur étude systématique; nous donnerons seulement quelques idées relatives à ceux-ci ultérieurement dans le sixième chapitre.

Dans ce chapitre, nous commencerons par présenter les modèles de spécification résultant de l'analyse du noyau; ensuite, nous indiquerons quelques caractéristiques de notre approche quant à la construction de cette spécification.

2.2. LES MODELES SOUS-JACENTS A LA SPECIFICATION ET LEUR REPRESENTATION

Nous présentons dans cette partie, les différents modèles obtenus à la suite de la spécification du problème; ce sont ceux-ci qui seront à la base par la suite de tout dialogue entre spécifieurs, demandeurs et utilisateurs.

La spécification du noyau du système d'information est partitionnée en une spécification des fonctions devant être réalisées par le système (l'énoncé du problème) et une spécification des différents types de données manipulés (l'univers du problème).

A. L'énoncé du problème

L'énoncé du problème se compose d'un certain nombre de "pavés de traitement" structurés sous la forme d'un arbre.

Le pavé au sommet de cet arbre décrit le problème initial au moyen de relations entre, d'une part, les résultats attendus et, d'autre part, les données de départ et d'éventuelles données intermédiaires introduites; chaque relation définit un sous-problème qui, à son tour, peut être spécifié de la même manière par un pavé de niveau inférieur.

De manière itérative, ce sous-problème est décrit de la même manière jusqu'à ce que des relations"terminales" soient introduites. Les relations "terminales" sont des relations identifiées comme élémentaires dans la description préliminaire du client; elles sont décrites au niveau de l'univers du problème.

Les liens entre les relations introduites au niveau d'un pavé peuvent être spécifiés au moyen des connecteurs OU, ET, POUR TOUT, AVEC qui connectent leurs définitions.

Une présentation graphique de la structure a été introduite, elle permet de présenter une vue globale de la spécification permettant de localiser plus facilement une modification à apporter; elle constitue également une aide pour le spécifieur pendant le processus de spécification en indiquant à tout moment le contexte courant et les sous-problèmes déjà décrits.

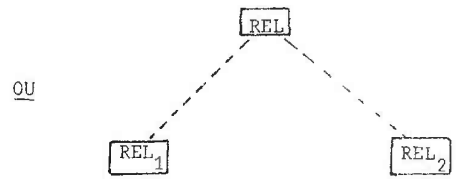
1° le connecteur OU

Il permet d'exprimer que la spécification d'une relation entre le résultat et l'argument peut être différente d'après les propriétés de l'argument.

D'éventuelles préconditions (au sens de [DIJ,76]) peuvent alors être introduites; elles expriment des conditions devant être remplies par l'argument.

Les définitions de relations connectées par OU impliquent une sélection non déterministe de leurs réalisations; ce type de connection reflète une analyse par cas.

La représentation graphique est la suivante :



Sa sémantique est la suivante :

r = Rel(d)

resultat : r

argument : d

tq (garde$_1$(d) et r = Rel$_1$(d)) OU
(garde$_2$(d) et r = Rel$_2$(d))

Remarquons que le fait de ne pas mentionner de garde rend la réalisation des relations complètement indéterministe.

On peut citer, comme exemple, le choix d'une formule de politesse parmi plusieurs formules possibles; la sélection se fait dans ce cas de manière aléatoire.

Exemple d'application :

La relation

ristourne = Calcul-ristourne(montant-total,$seuil_1$, $\%ristourne_1$,$\%ristourne_2$)

se raffine en

$seuil_1$-atteint(montant-total,$seuil_1$) et
ristourne = Calcul-ristourne-maximum(montant-total, $\%ristourne_2$)

ou

non $seuil_1$-atteint(montant-total,$seuil_1$) et
ristourne = Calcul-ristourne-minimum(montant-total, $\%ristourne_1$)

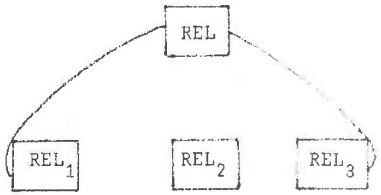
2° Le connecteur ET

Il permet d'exprimer que la spécification d'une relation entre résultat et argument peut être raffinée moyennant la décomposition du résultat et/ou de l'argument sous forme de n-uple.

Les définitions de relations connectées par ET peuvent être réalisées dans un ordre quelconque (même en parallèle).

La représentation graphique est la suivante :

ET



Sa sémantique est la suivante :

r = Rel(d)

résultat : r

argument : d

tq $\exists$ x_1,x_2,x_3,y_1,y_2 : $r = (x_1,x_2,x_3), d = (y_1,y_2)$

$x_1 = Rel(y_1)$ et $x_2 = Rel_2(y_2)$ et $x_3 = Rel_3(y_1,y_2)$

Exemple d'application :

La relation

facture = Préparation-facture(livraison, table-prix)

se raffine en

facture = (id-fact,corps-fact)
livraison = (id-livr,corps-livr)

id-fact = Def-identification-client(id-livr)
et
corps-fact = Def-corps-facture(corps-livr,table-prix)

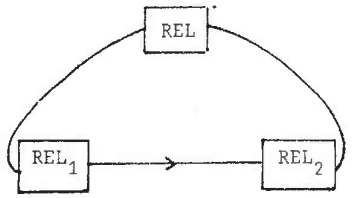
3° Le connecteur AVEC

Il permet de spécifier que l'expression de la relation entre le résultat et l'argument peut être raffinée moyennant l'introduction d'une donnée intermédiaire.

Les définitions des relations connectées par AVEC doivent être réalisées dans un ordre dicté par la disponibilité des arguments.

La représentation graphique est la suivante :

AVEC



Sa sémantique est la suivante :

$$r = Rel(d)$$

résultat : r

argument : d

tq $\exists$ x, $r = Rel_2(x)$ et $x = Rel_1(d)$

Exemple d'application :

La relation

fact = Gestion-commande(commande)

se raffine en

facture = Préparation-facture(livraison)

avec

livraison = Préparation-livraison(commande)

4° Le connecteur POUR TOUT

Il permet d'exprimer que l'expression de la relation entre le résultat et l'argument peut être raffinée moyennant la réalisation de relations entre composantes, car le résultat et/ou l'argument sont des suites ou ensembles.

La représentation graphique est la suivante :

POUR TOUT

Sa sémantique est la suivante :

$$r = Rel(d)$$

résultat : r

argument : d

tq $r_i = Rel_i(d_i)$

Exemple d'application

la relation

$facture^* = Facturation\ (livraison^*)$

se raffine en

facture = Def-facture(livraison)

La figure 1 illustre la représentation par arbre en présentant une partie de la spécification d'un système de gestion des commandes des clients à une société.

Remarque:
il est intéressant de rapprocher l'arbre présenté des arbres "ET/OU" utilisés en intelligence artificielle [NIL,80]

Nous allons maintenant, sur base de la figure 2, introduire la syntaxe et la sémantique associée à un pavé de traitement.

Remarquons, au préalable, que tous les mots soulignés font partie du langage de base que nous nous donnons pour la description des spécifications et ne nécessitent pas d'explications.

Outre le titre, la spécification de chacun des pavés de l'arbre déductif est constitué de trois parties comme dans [FIN,79] : la description formelle, le profil et le lexique.

a) Le titre

Nous indiquerons le nom de la relation décrite par le pavé à la suite de l'intitulé : Spécification du problème.

b) La description formelle

Dans la description formelle, on définit successivement :

- les résultats et les arguments :

  Il s'agit de données en provenance ou à destination d'un pavé de niveau égal ou supérieur (sous-problème racine du sous-arbre) dans la hiérarchie arborescente.

- les relations :

  Dans le cadre de la spécification du lien entre les résultats et les arguments, on peut être amené à introduire de nouvelles relations définissant des sous-problèmes plus fins. L'expression de ceux-ci peuvent conduire à l'introduction de nouvelles données intermédiaires.

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

. dans le domaine, on trouve des arguments et des données intermédiaires (ces dernières se trouvent également dans les codomaines d'autres relations);

. dans le codomaine, on trouve des résultats et des intermédiaires.

Associés aux arguments d'une relation, on peut également définir un certain nombre de préconditions.

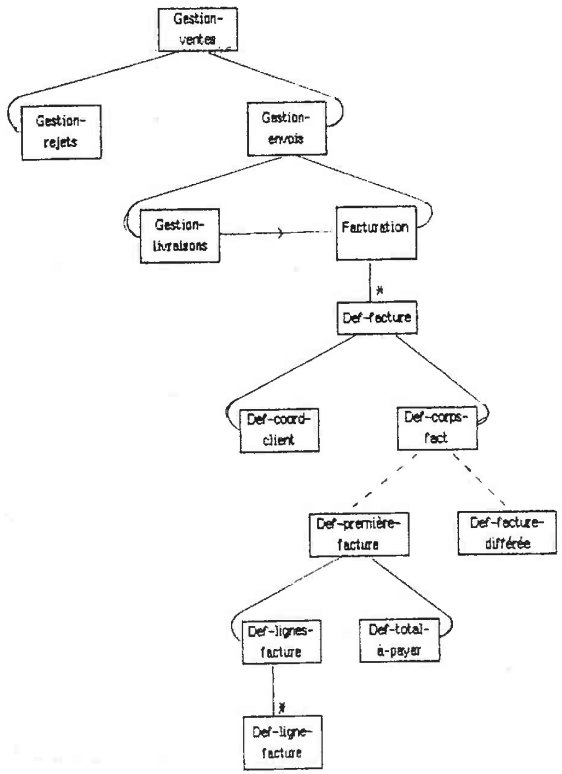


Figure 1 : Représentation graphique de l'arbre

Spécification du problème : Définition-maj-stock

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Il s'agit de réaliser un mouvement de produit sous la forme d'une mise à jour des stocks à destination du service de gestion de l'état des stocks Données maj-stock: mise à jour du stock de produit maj-stock-: mise à jour négative du stock de produit maj-stock+: mise à jour positive du stock de produit mvt: mouvement de produit constaté Relations Def-maj-: définition d'une mise à jour négative Def-maj+: définition d'une mise à jour positive Mvt-livraison: prédicat indiquant que le mouvement de de produit constaté est un mouvement de livraison à un client Mvt-approvisionnement: prédicat indiquant que le mouvement de produit constaté est un mouvement d'approvisionnement du produit auprès d'un fournisseur	Types maj-stock: MAJ-STOCK maj-stock-: MAJ-MOINS-STOCK maj-stock+: MAJ-PLUS-STOCK mvt: MVT Relations Définition-maj-stock: MAJ-STOCK <-- MVT Def-maj+: MAJ-MOINS-STOCK <-- MVT de base Def-maj+: MAJ-PLUS-STOCK <-- MVT de base Mvt-livraison: BOOLEEN <-- MVT de base Mvt-approvisionnement: BOOLEEN <-- MVT	résultats: maj-stock arguments: mvt relation résultats-arguments maj-stock = [maj-stock-, maj-stock+] Mvt-livraison (mvt) ssi maj-stock- = Def-maj- (mvt) ou Mvt-approvisionnement (mvt) ssi maj-stock+ = Def-maj+ (mvt)

Figure 2 : Spécification d'un pavé de traitement

c) Le profil

D'une part, on précise le type de chaque donnée manipulée dans la description formelle (résultat, argument ou donnée intermédiaire); c'est-à-dire, le nom attribué à l'ensemble des données caractérisées par les mêmes propriétés.

D'autre part, on décrit la signature des relations et des préconditions, c'est-à-dire, les types de données apparaissant respectivement dans leurs domaines et codomaines (en particulier, le codomaine d'une précondition est toujours booléen).

De plus, pour les relations terminales dont la description figure dans la description de l'univers, nous indiquerons la mention "de base".

d) Le lexique

Dans le lexique, sont décrits en langue naturelle les différents éléments suivants :

- la description de l'objectif de la relation étudiée;
- la description des différentes données que sont les résultats, les arguments et données intermédiaires;
- la description des nouvelles relations et préconditions introduites.

La rédaction de cette partie en langue naturelle permet d'améliorer la compréhension de la spécification pour le client; elle permet également de faire un certain nombre de vérifications.

B. L'univers du problème

Comme nous l'avons indiqué, les "pavés de traitement" font apparaître progressivement un certain nombre de types de données et de relations terminales portant sur ceux-ci.

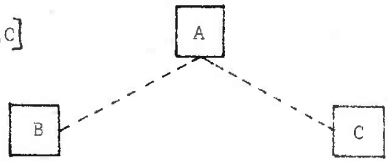
Leurs définitions précises figurent dans des "pavés de données"; ceux-ci s'apparentent aux types abstraits [LIS,75], [GUT,77], [ADJ,78].

La spécification des pavés de données est également structurée d'une manière hiérarchique; en effet, chaque type rencontré dans la spécification ainsi que ses opérations associées (les relations terminales) sont définis en terme d'un autre type plus élémentaire avec ses opérations associées.

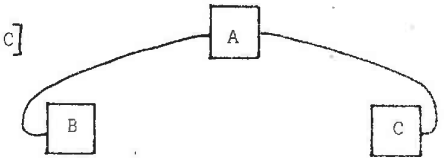
Ce processus est répété jusqu'à l'introduction dans la définition d'un type "de base" (booléen, entier, chaîne de caractères,...). Les liens entre types et types plus élémentaires s'expriment à l'aide des constructeurs suivants : produits cartésien (PC), union (UNION), suite (SUITE), ensemble (ENSEMBLE) et table (TABLE).

Nous avons associé une visualisation graphique à chacun de ces constructeurs :

A = UNION [B,C]



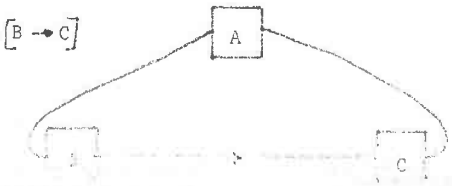
A = PC [B,C]



A = SUITE [B]

A = ENS [B]

A = TABLE [B → C]



La figure 3 illustre la représentation de l'arbre associé au type Facture; une facture étant un bordereau émis par la société à destination d'un client lors de l'envoi de produits.

La figure 4 présente un exemple de spécification de "pavé de données", la description est assez semblable à celle d'un "pavé de traitement".

a) Le titre

TYPE indique qu'il s'agit de la description d'un "pavé de donnée"

b) La description formelle

Dans la description formelle, sont successivement décrits :

- Le type de donnée décrite

La définition de ce type se fait sous la forme d'une expression faisant intervenir un ou plusieurs constructeurs sur des types plus élémentaires. Les constructeurs sont des types abstraits paramétrés (produit cartésien, suite, union,...) considérés comme prédéfinis du fait qu'ils sont associés à des structures de données caractéristiques des problèmes traités. Ils constituent des outils de construction de la spécification de l'univers [DER,79] ,[FIN,79] .

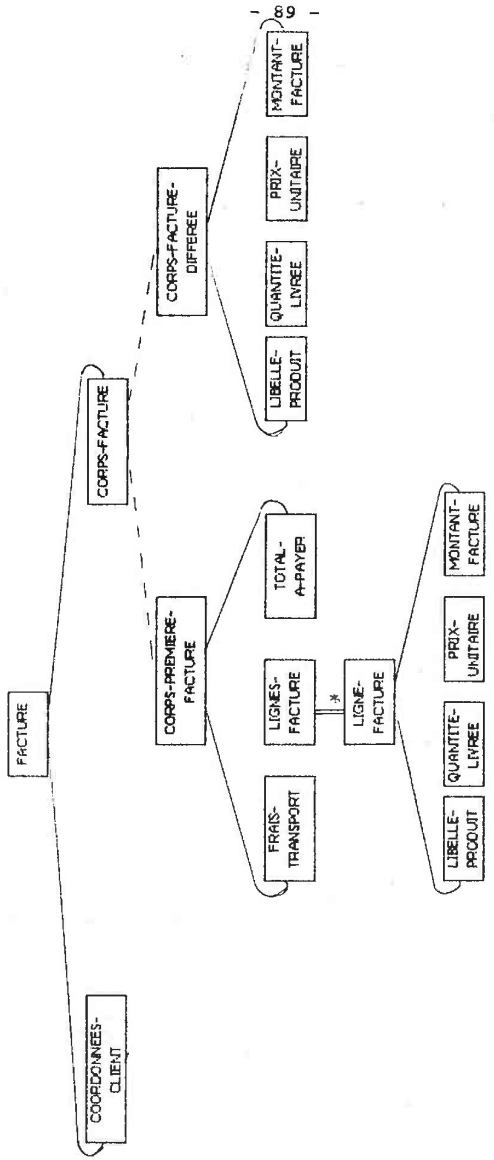


Figure 3 : Représentation graphe de l'arbre associé à un type

TYPE : LIGNE-PRODUIT

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Types: LIGNE-PRODUIT: il s'agit d'une ligne composée d'un numéro de produit et d'une quantité de ce produit ETAT-STOCK: table faisant correspondre à un numéro de produit la quantité de ce produit possédée en stock QTE-PROD: quantité de produit NUM-PROD: numéro de produit Invariant La quantité indiquée sur une ligne de produit doit toujours être strictement positive Opérations Ligne-livrable-partiellement: prédicat indiquant que le produit commandé ne se trouve que partiellement en stock Ligne-livrable-totalement: prédicat indiquant que le produit commandé se trouve entièrement en stock Def-ligne-livr-part: définition d'une ligne de livraison à partir d'une ligne de commande pour laquelle seulement une partie du produit commandé peut être livré Def-ligne-livr-tot: définition d'une ligne de livraison à partir d'une ligne de commande pour laquelle tout le produit commandé peut être livré	Types ligne-cmde, ligne-livr : LIGNE-PRODUIT état-stock: ETAT-STOCK bool: BOOLEEN Opérations Ligne-livrable-totalement: BOOLEEN <-- LIGNE-PRODUIT, ETAT-STOCK Ligne-livrable-partiellement: BOOLEEN <-- LIGNE-PRODUIT, ETAT-STOCK Def-ligne-livr-part: LIGNE-PRODUIT <-- LIGNE-PRODUIT, ETAT-STOCK Def-ligne-livr-tot: LIGNE-PRODUIT <-- LIGNE-PRODUIT	LIGNE-PRODUIT = PC [n-pi: NUM-PROD; qté: QTE-PROD] Invariant qté (: ligne-cmde) > 0 Opérations Consulter-stock Ligne-livrable-totalement (ligne-cmde,état-stock) bool tq bool = si Accès (état-stock, n-pi(ligne-cmde)) ≥ qté(ligne-cmde) alors Vrai sinon Faux Ligne-livrable-partiellement(ligne-cmde,état-stock) bool tq bool = si 0 < Accès (état-stock, n-pi(ligne-cmde) < qté(ligne-cmde) alors Vrai sinon Faux Création et mise à jour Def-ligne-livr-part (ligne-cmde,état-stock) ligne-livr tq ligne-livr = < n-pi(ligne-cmde), Accès (état-stock,n-pi(ligne-cmde) > Def-ligne-livr-tot (ligne-cmde,état-stock) ligne-livr tq ligne-livr = ligne-cmde

Figure 4 : Spécification d'un pavé de données

- Les invariants :

Il s'agit de prédicats permettant de restreindre les objets, occurrences du type défini, à ceux satisfaisant les propriétés énoncées. Celles-ci doivent être vérifiées avant et après toute opération définie sur toute donnée du type concerné.

- Les relations :

Celles-ci portent sur les données du type concerné; elles peuvent être de consultation, de création ou de mise à jour. Chaque relation est décrite par une équation conditionnelle, la partie droite de cette équation fait intervenir des relations associées, soit aux types élémentaires intervenant dans la définition, soit aux constructeurs utilisés.
Cette approche permet une définition progressive de l'univers puisque, à chaque étape, on construit un type à l'aide de types qui seront décrits ultérieurement.

c) Le profil

Dans le profil, on spécifie le type de chacune des données apparues dans l'expression des invariants ou des opérations. On donne également la signature de chacune des opérations introduites.

d) Le lexique

Le lexique présente à nouveau une description en langue naturelle des différents types, invariants et opérations apparues dans ce pavé.

2.3. LA CONSTRUCTION DE LA SPECIFICATION

Comme l'indique J.P. FINANCE [FIN,79] le problème de la spécification peut s'énoncer de la manière suivante :

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

La recherche d'une spécification précise a pour complément indispensable la recherche d'une approche aidant à sa construction. Etant donné le grand nombre de détails qu'il est nécessaire de prendre en considération dans la spécification de S.I., il est indispensable que l'approche suivie favorise une structuration modulaire de la spécification en isolant des sous-problèmes indépendants.

Les idées sous-jacentes à la construction structurée d'une spécification ne sont guère éloignées de celles développées depuis la fin des années 60 pour la construction d'un algorithme. En effet, jusqu'à cette époque, l'empirisme qui était de règle dans la résolution des problèmes de programmation avait eu comme résultat des programmes non conformes aux spécifications, contenant de nombreuses erreurs, mal documentés et illisibles. Cette anarchie intolérable a conduit, petit à petit, à l'avènement d'un certain nombre d'approches regroupées sous le vocable de programmation structurée [DAH,72], [WIR,73], [JAC,75], [DIJ,76] .
L'idée générale consiste à travailler par raffinements successifs, en partant d'une idée générale du traitement et à la détailler et décomposer de plus en plus finement.

La programmation déductive proposée par C.PAIR [PAI,77] , fait partie de ces méthodes : l'idée est de procéder à une analyse "déductive" en définissant, d'abord, les données "résultats" puis en caractérisant celles-ci en fonction de données "intermédiaires" considérées comme résultats de sous-problèmes et en appliquant itérativement cette démarche jusqu'à ce qu'on n'ait plus comme données "intermédiaires" que des "données de départ" du problème.

Les idées essentielles sous-jacentes, adaptées par [FIN,79] à la structuration d'une spécification sont les suivantes :

A. On part de la définition des données "résultats". Dans toute description du problème, elles sont, en effet, généralement décrites assez précisément; on peut dont prendre cette base comme spécification initiale;

B. Il est rarement possible d'exprimer directement les résultats à partir des données "de départ"; on est donc conduit progressivement à introduire des objets, des relations intermédiaires et des types (ceux-ci palliant le problème de la redondance). Cette approche permet de se poser les sous-problèmes (liés à la définition des intermédiaires) aux bons moments.
D'autre part, elle permet de dresser petit à petit un inventaire de tous les types de données qui sont amenés à jouer un rôle dans la spécification du problème.

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

Le développement de cette approche est caractérisé par les idées suivantes :

1° Dans le chapitre consacré à la présentation de l'état de l'art (1.3.2), nous avons indiqué l'existence de deux types d'approche à la spécification.

D'une part, les approches qualifiées d'approche "base de données"; celles-ci sont davantage orientées vers la description a priori d'un modèle des données unique et non-redondant.

D'autre part, les méthodes, dites fonctionnelles, qui se concentrent davantage à la description des traitements au détriment de celle des données (multiplication, redondance).

Un autre inconvénient réside en la faible évolutivité des systèmes obtenus; ceci étant en partie dû à la faiblesse des mécanismes de structuration des données.

La démarche que nous proposons constitue un compromis entre ces deux points de vue.

En effet, nous construisons en parallèle la spécification des relations (l'énoncé du problème) et la spécification des types de données (l'univers du problème).

Les types se structurent en une famille de types abstraits hiérarchiques; mais la hiérarchie ainsi obtenue peut être arbitraire et contraignante si l'on veut faire évoluer le sytème en lui ajoutant de nouvelles fonctionnalités.

Pour résoudre ce problème, le système SPES, sur lequel nous nous appuyons, met à la disposition du spécifieur une famille d'opérations permettant de réorganiser une structure hiérarchique de types en vue d'une modification (transformations au niveau des structures de types [FIN,79]). Ce problème d'équivalence de structures de types est évoqué dans le Chapitre 6.

2° La plupart des approches reposent sur une analyse "inductive"; c'est-à-dire, allant des données vers les résultats. Si une telle approche paraît assez naturelle et facile à appliquer, nous avons expérimenté qu'elle pouvait être à la base de problèmes au niveau de la complétude de la spécification; en effet, partir des données ne garantit pas nécessairement que tous les résultats attendus sont effectivement atteints et décrits. D'autre part, l'application de la démarche inductive nécessite de définir certains éléments avant d'en connaître leur utilisation; ce qui, à priori, est parfois difficile.

Par contre, il semble qu'un processus déductif dans lequel on spécifie les résultats attendus d'abord, puis les données intermédiaires nécessaires pour définir les résultats, jusqu'à atteindre les données de départ, paraît davantage adapté, en particulier :

- Il permet d'écrire la spécification dans un ordre totalement indépendant de l'ordre dans lequel les programmes correspondants seront exécutés;
- Il permet de formuler les différents problèmes et sous-problèmes de façon modulaire.

3° Enfin, une des qualité essentielles d'une spécification est sa facilité de modification.
Une première facilité réside dans le caractère structuré et modulaire de la spécification obtenue; cependant, comme c'est le cas pour la modification d'un programme, le spécifieur a besoin, en plus des textes d'énoncés, des décisions successives qui ont conduit à l'élaboration de ces énoncés [DAR,79] [SCH,83] .
A cet effet, nous avons développé un méta-algorithme dont l'application devrait permettre, non seulement d'obtenir la spécification du problème, mais, de plus, de garder une trace de ces choix faits; la spécification modifiée peut alors souvent s'obtenir par application du même méta-algorithme.

Associé à ce méta-algorithme, un certain nombre de stratégies sont proposées, chacune d'entre elles propose un schéma de raffinement particulier pour la relation étudiée. Bon nombre de ces stratégies sont basées sur l'étude du type du résultat; elles consistent à déduire le schéma de raffinement de la relation considérée à partir de la description de la structure du résultat (cette dernière s'exprimant par application d'un constructeur de type sur des types plus élémentaires).

2.4. CONCLUSION

Nous avons, dans ce chapitre, introduit les idées caractérisant la démarche que nous préconisons; à savoir :

- la construction en parallèle de l'énoncé (spécification des différentes fonctions devant être remplies par le système) et de l'univers du problème (spécification des différents types de données manipulés);

- une démarche induite par l'objectif à réaliser, c'est-à-dire, allant des résultats vers les arguments;

- l'existence d'un catalogue de stratégies de spécification; le suivi de l'application de celles-ci permettant de garder une trace des décisions successives prises par le spécifieur.

La construction incrémentale de la spécification de l'énoncé et de l'univers nécessite de disposer d'un certain nombre d'outils de construction prédéfinis; à savoir un certain nombre de types et leurs opérations caractéristiques.

La description formelle de ces outils est faite dans le chapitre suivant.

° ° °

Chapitre 3
LES OUTILS DE CONSTRUCTION DE LA SPECIFICATION

Comme nous l'avons indiqué, pour construire la spécification de l'énoncé et de l'univers du problème, nous disposons d'un certain nombre d'outils prédéfinis.

Cet ensemble se compose :

- de types constituant les bases canoniques de toute spécification de système d'information;

- d'outils de construction de types à partir de types plus élémentaires.

Chacun des types ou outils de construction utilisés est caractérisé par un certain nombre d'opérations propres au domaine d'application considéré; leur présentation est faite sous forme de type abstrait [LIS,75] , [GUT,77] , [ADJ,78] .

Avant d'aller plus en avant dans la description de cet ensemble d'outils prédéfinis, nous rappelons le cadre formel associé à un type abstrait.

3.1. SPECIFICATION D'UN TYPE ABSTRAIT

La spécification d'un type abstrait se compose généralement des éléments suivants :

1° la partie syntaxique comprenant :

- le nom du type décrit;
- la description de l'environnement; c'est-à-dire les noms des autres types intervenant dans la définition des opérations du type décrit;

- la description des opérations, c'est-à-dire leur nom et leur profil; parmi ces opérations, nous distinguons les constructeurs d'objets du type, les opérations de modification et les opérations externes (fonctions d'accès);

2° la partie sémantique du type est défini par un ensemble d'équations conditionnelles caractérisant l'effet des combinaisons autorisées des opérations associées au type;

3° on peut, en outre, restreindre les objets, occurrences d'un type défini, à ceux satisfaisant une propriété appelée invariant.

La description du type est dite "stricte" si elle ne fait pas apparaître d'autres noms de type dans sa description.

A titre d'exemple, voici la description stricte du type abstrait associé aux entiers naturels inférieurs à 10 (le type est représenté dans un langage apparenté au langage SPES [FIN,83] .

```
TYPE NATINF10 = =
     ENVIRONNEMENT
          BOOL
     OPERATIONS
          CONSTRUCT    Zero :                  →NATINF10
                       Succ : NATINF10 →NATINF10      % plus 1%
          MODIF        Pred : NATINF10 →NATINF10      % moins 1%
          EXTERN       Eq   : NATINF10,NATINF10→BOOL  % égalité%
                       Inf10 : NATINF10 →BOOL         %≤ 10? %
```



```
DECLARATIONS
     b : BOOL
     e,e' : NATINF10
INVARIANTS
     Inf10(e) = = Vrai
EQUATIONS
     DEF-MODIF    Pred(Zero) = = Zero
                  Pred(suc(e)) = = e
     DEF-EXTERN   Eq(Zero,Suc(e)) = = Faux
                  Eq(Zero,Zero) = = Vrai
                  Eq(Suc(e),Suc(e')) = = Eq(e,e')
                  Inf10(Zero) = = Vrai
                  Inf10(Suc(e)) = = (Inf10(e)∧
                                     Non(Eq(e,Succ^9(Zero))))
```

Remarques :

- Succ ^{9}(Zero) est équivalent à :

  Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Zero)))))))))

- Les opérations "∧", "Eg" et Non" sont des opérations caractéristiques des booléens.

La description du type peut se faire sous la forme d'une expression faisant intervenir d'autres types.

Dans l'exemple ci-dessous, la description d'une écriture comptable entre deux comptes se fait par l'intermédiaire d'une expression (avec le constructeur "produit cartésien") faisant intervenir des types plus élémentaires.

```
TYPE TRANSACTION = = PC [Date : DATE,cpte-débit : COMPTE,
                         cpte-crédit : COMPTE, Montant : NAT]
•
    ENVIRONNEMENT
          DATE,COMPTE,NAT
    OPERATIONS
          MODIF Cré-trans : DATE,COMPTE,NAT → TRANSACTION
                            %création transaction%
                Modif-montant : TRANSACTION,NAT →TRANSACTION
                            %modif-montant-transaction%
          EXTERN Accès-montant : TRANSACTION→NAT
                            %accès-montant-transaction%
    DECLARATIONS
          date : DATE
          tran : TRANSACTION
          cpte,cpte' : COMPTE
          val,val' : NAT
    EQUATIONS
          DEF-MODIF
                Cré-trans(date,cpte,cpte',val) = = < date,
                          cpte,cpte',val>
                Modif-montant(tran,val') = = <date(:tran),
                          cpte-débit(:tran),cpte-crédit
                          (:tran),val' >
          DEF-EXTERN
                Accès-montant(tran) = = Montant(:tran)
```

Remarque :

Les opérations "<.>" et "(:)" sont respectivement les opérations de construction d'un n-uplet et d'accès à un des champs.

Dans la description de type, on peut également exprimer la possibilité de ré-utiliser des types :

- par renommage, lorsque le type décrit a les mêmes caractéristiques qu'un autre type (il faut éventuellement renommer des opérations identiques dont les noms devraient être différents).

  Exemple :

```
TYPE   MOUVEMENT = = TRANSACTION
       RENOMME EN REMPLACANT  Cré-mvt  PAR  Cré-trans
```

- par enrichissement, lorsque le type est décrit par l'ajout d'opérations à un type précédemment défini.

  Exemple :

```
TYPE   TRANSAC = =  ENRICHISSEMENT DE TRANSACTION PAR
     OPERATION
       EXTERN  Valeur-débitée : TRANS→NAT  %Montant débité
                                            d'un compte%
     DECLARATIONS
       tran : TRANS
     EQUATIONS
       Valeur-débitée(tran) = =  - Montant(:tran)
```

Outre les avantages offerts par le cadre des types abstraits qui ont déjà été évoqués dans le premier chapitre (cf. le modèle des types abstraits), nous pouvons encore citer les points suivants :

- la minimalité : on est encouragé à n'énoncer qu'un minimum de propriétés utiles sans introduire de surspécification;
- le mécanisme de construction incrémentale permet de définir un type en termes de types plus élémentaires;
- l'histoire d'un objet peut être conservée au travers de la définition de sa forme normale qui est l'expression de tous les éléments du type ne faisant intervenir que les constructeurs de type; cette propriété est particulièrement intéressante pour des objets comme des suites et des tables auxquels nous associons parfois un rôle historique.

3.2. SPECIFICATION DES OUTILS DE CONSTRUCTION

Successivement, nous présentons :

3.2.1. Les types de base (bases canoniques des systèmes d'information);

3.2.2. Les constructeurs de types à partir de types plus élémentaires.

3.2.1. Les types de base

La construction incrémentale du système d'information permet une définition progressive du S.I. : la mise en oeuvre d'opérations de construction nécessite de disposer de types primitifs et prédéfinis.

Dans la suite, nous supposons disposer des types de base suivants :

- les entiers naturels (NAT)
- les entiers (INT)
- les booléens (BOOL)
- les caractères (CHAR)
- les chaînes de caractères (CHARST) avec les opérations habituelles de concaténation, extraction d'une sous-chaîne,...
- les dates (DATE), ce type est appelé à jouer un rôle privilégié en informatique de gestion, ses opérations caractéristiques sont : comparaison entre deux dates, durée entre deux dates, contrôle de validité d'une date,...

Chacun de ces types est décrit de manière algébrique à titre d'exemple, est présentée, dans le langage de description retenu, la spécification du type BOOL.

TYPE BOOL

OPERATIONS

CONSTRUCT : Vrai : → BOOL
Faux : → BOOL

MODIF $\vee$: BOOL,BOOL → BOOL %ou%
$\wedge$: BOOL,BOOL → BOOL %et%
Non : BOOL → BOOL %Non%
$\vee_{ex}$: BOOL,BOOL → BOOL %ou$_{ex}$%

EXTERN Eg : BOOL,BOOL → BOOL %égalité%

EQUATIONS

DEF-MODIF $\vee$(Vrai,Vrai) = = Vrai
$\vee$(Vrai,Faux) = = Vrai
$\vee$(Faux,Vrai) = = Vrai
$\vee$(Faux,Faux) = = Faux
$\wedge$(Vrai,Vrai) = = Vrai
$\wedge$(Vrai,Faux) = = Faux
$\wedge$(Faux,Vrai) = = Faux
$\wedge$(Faux,Faux) = = Faux
Non(Vrai) = = Faux
Non(Faux) = = Vrai
$\vee_{ex}$(Vrai,Vrai) = = Faux
$\vee_{ex}$(Vrai,Faux) = = Vrai
$\vee_{ex}$(Faux,Vrai) = = Vrai
$\vee_{ex}$(Faux,Faux) = = Faux

DEF-EXTERN Eg(Vrai,Vrai) = = Vrai
Eg(Faux,Faux) = = Vrai
Eg(Faux,Vrai) = = Faux
Eg(Vrai,Faux) = = Faux

3.2.2. Les constructeurs de type

Lors de la spécification de systèmes d'information, nous avons remarqué qu'un certain nombre de structures de données caractéristiques reviennent fréquemment, et qu'elles peuvent être combinées entre elles.

Les opérations attachées à ces diverses structures sont d'un usage si général que nous avons jugé pertinent de les définir une fois pour toutes et ainsi de permettre au spécifieur de les utiliser comme outils de construction de la spécification du système d'information [FIN,79], [ADJ,82].

Les constructeurs retenus sont les suivants :

A. Le produit cartésien

B. L'union

C. Le produit cartésien étendu

D. La suite

E. L'ensemble

F. La table

A. Le produit cartésien

Un objet de type produit cartésien est un n-uple; chaque composant du n-uple correspond à un champ, chaque champ étant nommé :

Exemple :

TYPE IDENTIFICATION-CLIENT = = PC[Nom-cli : CHARST, Prénom-cli : CHARST, Ville : CHARST]

Les opérations de base sont :

- la construction d'un n-uplet, notée : $\langle .,..,. \rangle$

  Exemple :

  ident-cli = $\langle$Dupond, André, Nancy$\rangle$

  est un objet du type IDENTIFICATION-CLIENT

- l'accès à un des composants par le nom du champ correspondant (sélecteur), notée : (:...)

  Exemple :

  Prénon-cli(:ident-cli) = André

La spécification formelle du constructeur produit cartésien paramètré par les types S_1 et S_2 est la suivante :

TYPE PC[$Sel_1 : S_1$, $Sel_2 : S_2$]

ENVIRONNEMENT
BOOL
OPERATIONS
CONSTRUCT $\langle\ \rangle : S_1, S_2 \rightarrow PC[S_1, S_2]$
MODIF $Sel_1 : PC[S_1, S_2] \rightarrow S_1$ %sélecteur 1:champ%
$Sel_2 : PC[S_1, S_2] \rightarrow S_2$ %sélecteur 2:champ%
DECLARATIONS
$s_1 : S_1$ $s_2 : S_2$
EQUATIONS
DEF-MODIF
$Sel_1 (: \langle s_1, s_2 \rangle) = = s_1$
$Sel_2 (: \langle s_1, s_2 \rangle) = = s_2$

B. L'union disjointe

Un objet de type union disjointe de plusieurs types est d'un et un seul type figurant dans l'union.

Exemple :

TYPE FACTURE = = UNION [FACTURE-RAPIDE, FACTURE-DIFFEREE]

Un objet de type FACTURE est, soit du type FACTURE-RAPIDE, soit du type FACTURE-DIFFEREE.

Grâce à l'utilisation de ce constructeur, les opérations communes aux types figurant dans l'union peuvent être également associées au type construit.

Les opérations de base sont :

- la construction d'un objet du type union de types plus élémentaires notée : [ ]

  Exemple :

  Si fact-diff est un objet du type FACTURE-DIFFEREE, [fact-diff] est un objet du type FACTURE.

- un prédicat pour chacun des types constitutifs de l'union, indiquant si un objet est de ce type ou non :

  Exemple :

  Facture-différée-type ([fact-diff]) = Vrai
  Facture-rapide-type ([fact-diff]) = Faux

  Remarque : le nom du prédicat sera constitué, pour sa première partie, du nom du type constitutif de l'union.

La spécification formelle du constructeur Union paramétré par les types S_1 et S_2 se définit comme un enrichissement du type Produit Cartésien :

TYPE UNION $[S_1,S_2]$ = = ENRICHISSEMENT DE
PC$[Sel_1 : S_1{}^{\omega}, Sel_2 : S_2{}^{\omega}]$ PAR

OPERATIONS

MODIF $[\ ] : S_1{}^{\omega}, S_2{}^{\omega} \rightarrow$ UNION$[S_1,S_2]$

EXTERN S_1-type : UNION $[S_1,S_2] \rightarrow$ BOOL

S_2-type : UNION $[S_1,S_2] \rightarrow$ BOOL

DECLARATIONS

$s_1 : S_1{}^{\omega}$

$s_2 : S_2{}^{\omega}$

INVARIANT

$Eg(Sel_1(:[s_1,s_2]),\omega) \ V_{ex} \ Eg(Sel_2(:[s_1,s_2]),\omega)$

EQUATIONS

DEF-MODIF $[s_1,s_2] = = \langle s_1,s_2 \rangle$

DEF-EXTERN S_1-type $([s_1,s_2]) = =$ Faux si $Eg(Sel_1(:[s_1,s_2]),\omega)$

S_1-type $([s_1,s_2]) = =$ Vrai si $Non(Eg(Sel_1(:[s_1,s_2]),\omega))$

S_2-type $([s_1,s_2]) = =$ Vrai si $Non(Eg(Sel_2(:[s_1,s_2]),\omega))$

S_2-type $([s_1,s_2]) = =$ Faux si $Eg(Sel_2(:[s_1,s_2]),\omega)$

C. Le produit cartésien étendu

Il s'agit d'un produit cartésien dans lequel les composants sont facultatifs.

Exemple :

TYPE IDENTITE-PRODUIT = = PC°[libellé-prod : CHARST, numéro-prod : NAT]

L'identification d'un produit se fait sur base, soit de la connaissance de son libellé, soit de la connaissance de son numéro, soit de la connaissance des deux.

Les opérations de base sont :

- la construction, notée : $\langle\ \rangle^\circ$

  Exemple :

  id-prod = $\langle \omega, 1213 \rangle^\circ$; id-prod' : $\langle$sucre,$\omega\rangle^\circ$;
  id-prod" : $\langle$vin,1121$\rangle^\circ$

  sont des objets du type IDENTITE-PRODUIT

- un prédicat associé à chacun des types constitutifs du produit cartésien étendu indique si un objet du type constitutif est présent ou non

  Exemple :

  Libellé-prod-présent(id-prod) = faux
  Libellé-prod-présent(id-prod') = vrai

Remarque : le nom du prédicat est constitué, pour sa première partie, soit du nom du type constitutif (s'il est discriminant), soit du nom du champ.

- les fonctions d'accès aux composants sont identiques à celles introduites pour le produit cartésien à l'exception du fait qu'elles peuvent avoir comme résultat un élément vide

  Exemple :

  Libellé-prod(id-prod) = ω

La spécification formelle du constructeur Produit Cartésien Etendu paramètré par les types S_1 et S_2 se définit comme un enrichissement du type Produit Cartésien.

TYPE $PC^\circ[Sel_1' : S_1 \ , \ Sel_2' : S_2]$ = = ENRICHISSEMENT DE

UNION $[PC\ [Sel_1 : S_1 , Sel_2 : S_2]$, UNION $[S_1, S_2]]$ PAR

OPERATIONS

MODIF $PC^\circ : S_1 \cup \{\omega\}, S_2 \cup \{\omega\} \rightarrow PC^\circ[S_1, S_2]$

EXTERN $Sel_1' : PC^\circ [S_1, S_2] \rightarrow S_1$

$Sel_2' : PC^\circ [S_1, S_2] \rightarrow S_2$

S_1-présent : $PC^\circ [S_1, S_2] \rightarrow$ BOOL

S_2-présent : $PC^\circ [S_1, S_2] \rightarrow$ BOOL

DECLARATIONS

$s_1 : S_1 \cup \{\omega\}$

$s_2 : S_2 \cup \{\omega\}$

INVARIANT

$Eg(Sel_1'(:\langle s_1,s_2\rangle^\circ),\omega) \vee Eg(Sel_2'(:\langle s_1,s_2\rangle^\circ),\omega)$

EQUATIONS

DEF-MODIF

$\langle s_1,s_2\rangle^\circ == [\langle s_1,s_2\rangle] == \langle s_1,s_2\rangle$ si (Non(Eg(s_1,ω)) Non(Eg(s_2,ω)))

$\langle s_1,s_2\rangle^\circ == [[s_1]] = \langle s_1\rangle$ si Eg(s_2,ω)

$\langle s_1,s_2\rangle^\circ == [[s_2]] = \langle s_2\rangle$ si Eg(s_1,ω)

DEF-EXTERN

S_1-présent($\langle s_1,s_2\rangle^\circ$) == Vrai si Non(Eg($s_1,\omega$))

S_1-présent($\langle s_1,s_2\rangle^\circ$) == Faux si Eg ($s_1,\omega$)

S_2-présent($\langle s_1,s_2\rangle^\circ$) == Vrai si Non(Eg($s_2,\omega$))

S_2-présent($\langle s_1,s_2\rangle^\circ$) == Faux si Eg($s_2,\omega$)

$Sel_1'(:\langle s_1,s_2\rangle^\circ) == s_1$ si Non(Eg(s_1,ω))

$Sel_1'(:\langle s_1,s_2\rangle^\circ) == \omega$ si Eg(s_1,ω)

$Sel_2'(:\langle s_1,s_2\rangle^\circ) == s_2$ si Non(Eg(s_2,ω))

$Sel_2'(:\langle s_1,s_2\rangle^\circ) == \omega$ si Eg(s_2,ω)

Remarque : Dans beaucoup de cas, nous aurons des produits cartésiens où seul un champ est facultatif

Exemple :

$A = PC[PC^\circ[B],C,D]$

Dans ce cas, nous simplifierons la notation en écrivant : $A = PC[B^\circ,C,D]$

De même, pour la construction d'un objet, à la place de noter "$a = \langle\langle b,\rangle^\circ,c,d\rangle$", nous écrivons : "$a = \langle b^\circ,c,d\rangle$" (avec en particulier $\langle^\circ,c,d\rangle$).
Le prédicat testant la présence de b s'écrira quant à lui : B-présent(a).

D. Les suites et ensembles

Dans la spécification de systèmes d'information, les concepts de suite et d'ensemble jouent un rôle privilégié.

En effet, la modélisation de beaucoup de problèmes posés nécessite une structuration des données sous la forme, soit d'un ensemble ordonné (par exemple, un ordonnancement temporel dans la production des factures), soit d'un ensemble non ordonné. **Les concepts de suite et d'ensemble répondent respectivement à chacun de ces besoins; notons que l'ordre désiré est fréquemment un ordre temporel reposant sur le fait qu'à chacun des objets est associé une date.**
Plusieurs approches parmi celles présentées dans le premier chapitre n'introduisent pas le concept de lot de données, ni d'outils permettant de le manipuler; en particulier, nous avons rencontré plusieurs spécifications dans lesquelles il ne ressort pas clairement si l'argument d'une fonction est un objet d'un certain type ou une suite/ensemble d'objets de ce type.

Le fait de ne pas disposer de moyens permettant de manipuler ces ensembles/suites en tant que tels peut conduire le spécifieur, soit à confondre suite et élément d'une suite, ce qui introduit des incohérences, soit à utiliser des paliatifs qui souvent entraînent des surspécifications : ainsi, l'introduction de point d'accumulation ou l'utilisation du "fichier" ou de la "documentation" comme support de mémorisation de données devant persister entre différentes activations d'une même fonction; dans ce cas, ce support constitue en fait une solution au problème de l'accès à un historique de ces données. (Notons que ce problème des références historiques est également mis en évidence par Feather [FEA,80] dans le cadre de la gestion d'une librairie dans laquelle certains fichiers doivent pouvoir être restaurés dans un état antérieur).

Associées aux concepts de suite et d'ensemble sont définies un certain nombre d'opérations caractérisant ces structures (par exemple : premier élément, dernier élément d'une suite; appartenance, adjonction d'un élément à un ensemble); ces opérations sont cependant assez élémentaires et il nous paraît utile de disposer d'opérations plus riches qui apparaissent comme typiques du domaine d'application considéré.

De telles"primitives" peuvent être obtenues par composition des opérations élémentaires; les avantages sont les suivants :

- Elles permettent de construire une spécification en s'arrêtant à un certain niveau au-delà duquel les détails introduits ne seront généralement plus pertinents du point de vue du demandeur et, par conséquent, risquent de compliquer sa compréhension de la spécification. Un exemple de l'intérêt d'une telle primitive est donné par BAUER [BAU,82] dans le cas d'un problème de tri. En effet, si du point de vue de la réalisation d'un tri, il importe de connaître la technique retenue, du point de vue de la spécification, connaître la technique a non seulement peu d'importance mais de plus peu nuire à la clareté de la spécification (ainsi, BAUER évoque deux manières de décrire le tri qui sont formellement équivalentes mais dont l'une est moins facilement compréhensible pour le non-informaticien).
  L'utilisation de primitives implique bien-sûr qu'il y ait sur leur signification précise un consensus entre les différentes personnes appelées à les utiliser; celui-ci peut être atteint par la connaissance de la sémantique associée à chaque primitive;

- La présence de primitives au sein d'une spécification rend la spécification plus compacte. De telles primitives pourraient ainsi permettre de se rapprocher d'une des qualités primordiales d'une spécification informelle qu'est la concision [BAL,77] ; en effet, dans cette dernière, seule une partie du problème est rendue explicite;

- L'utilisation de primitives simplifie la tâche du spécifieur tout en lui assurant d'avoir une définition précise et correcte une fois pour toutes; elle favorise également la réutilisation de définitions de base.

Par leur aspect spécifique à la modélisation de comportements propres aux application de gestion, les primitives proposées s'apparentent à certaines opérations introduites dans les langages de haut niveau tels que BDL ou SSL (une description plus détaillée se trouve dans [LAM,83]) :

- Dans le langage BDL (Business Data Language [HAM,76]), la base du raisonnement est le concept de flux de documents; ces documents subissent un certain nombre de transformations (appelées "Step"). Au niveau du flux de documents (notés "...'s"), il existe un certain nombre de transformations prédéfinies permettant, par exemple, l'accumulation de documents suivant un certain nombre ou la survenance d'un signal, le filtrage de certains documents caractérisés par une propriété commune ("with common"), le groupage de documents en différents groupes, chacun étant caractérisé par une propriété commune ("one per"), etc.

- Dans le langage SSL (System Specification Language [RUT,81]), un programme est structuré sous la forme d'un ensemble de "calculs" parmi lesquels certains permettent une manipulation globale d'agrégats de données. Ainsi, le "calcul qui assortit" effectue un même traitement sur tous les objets assortis par une même valeur de clé et sélectionnés à partir de différents ensembles de données. Un tel calcul permet de recouvrir toutes les opérations de gestion telles que itérations, sélection d'articles assortis, test de fin de fichier, ... Un autre type de calcul est le "calcul qui groupe"; celui-ci permet d'associer à un ensemble de données en entrée différents groupes d'articles de même valeur de clé; un résultat étant alors construit pour chaque groupe ainsi formé par applications d'opérateurs tels que SUM, MAX, MIN, ...

De la même manière, associées aux structures de suite et d'ensemble, nous allons définir un certain nombre de primitives.

Celles-ci se présente sous la forme d'une hiérarchie de type "utilise" où :

- le niveau 0 contient le noyau des opérations élémentaires sur les suites/ensembles;
- le niveau 1 se compose d'opérations habituellement associées aux suites/ensembles;
- le niveau 2 contient des opérations caractéristiques des transformations spécifiques rencontrées dans les applications de gestion;
- le niveau 3 contient d'autres opérations, nécessaires à la gestion, obtenues par enrichissement d'opérations des niveaux 1 et 2.

Du point de vue de leur définition, les opérations de niveau 0 seront définies de manière algébrique; certaines opérations de niveau 1 seront également définies de manière algébrique; les autres, et celles des niveaux 2 et 3 seront définies par extension de celles des niveaux inférieurs.

Du point de vue de leur utilisation, seules les opérations des niveaux 1, 2 et 3 seront accessibles au spécifieur; celui-ci peut également enrichir le niveau 2 par de nouvelles opérations s'il existe entre lui, les autres spécifieurs et les demandeurs un consensus quant à leur sémantique.

1° La suite

Comme nous l'avons déjà indiqué, à la notion de suite est toujours associée une notion de structuration d'une collection suivant un ordre; en particulier, nous sommes souvent conduits à considérer un ordonnancement des éléments d'une suite suivant leur apparition dans le temps, dans ce cas, une date est alors implicitement associée à chacun des éléments. On supposera alors l'existence d'une fonction Date associant une date à tout élément; une suite ordonnancée de manière temporelle est alors définie par :

$$s = \text{Ajout}(\text{ajout}(s', e_1), e_2)) \Rightarrow \text{Date}(e_2) > \text{Date}(e_1)$$

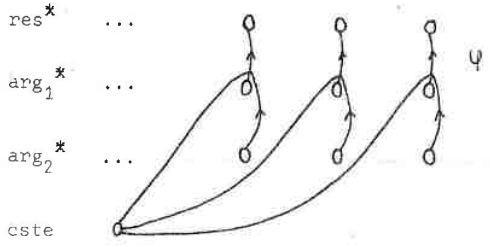
ou encore

$$\forall s : \forall i : 1 \leq i < \text{Taille}(s) \; : \; \text{Date}(\text{Accès}(s,i)) < \text{Date}(\text{Accès}(s, i+1))$$

Niveau 0

Les opérations appartenant au noyau de base sont les suivantes :

- La suite vide : Svide
- L'adjonction d'un élément en queue : Ajout
- La définition d'une suite résultat (res^*) par applications successives d'une même transformation φ sur chaque n-uple formé en prenant, dans chacune des deux suites arguments (arg_1^* et arg_2^*), l'élément suivant celui qui a été considéré pour former le n-uple précédent; la fonction peut de plus faire intervenir un paramètre constant (cste) ainsi que la suite des résultats précédemment définis.
  Graphiquement, la transformation se présente de la manière suivante :



- Le prédicat de test d'une suite vide : Vide

La spécification formelle du constructeur Suite paramètré par le type RES est la suivante :

TYPE SUITE [RES]

ENVIRONNEMENT

BOOL, ARG_1, ARG_2, PAR

OPERATIONS

CONSTRUCT	Svide :	→ SUITE [RES]
	Ajout :	SUITE [RES], RES → SUITE [RES]
MODIF	Iterφ :	SUITE [ARG_1], SUITE [ARG_2] → SUITE [RES]
EXTERN	Vide :	SUITE [RES] → BOOL

DECLARATIONS

b : BOOL
r^* : SUITE [RES]
r : RES
a_1^* : SUITE [ARG_1]
a_2^* : SUITE [ARG_2]
a_2 : ARG_2
cst : PAR
φ : ARG_1, ARG_2, PAR, SUITE [RES] → SUITE [RES]

EQUATIONS

DEF-MODIF

$Iter_\varphi(Ajout(a_1^*, a_1), Ajout(a_2^*, a_2)) =$
$Ajout(Iter_\varphi(a_1^*, a_2^*), \varphi(a_1, a_2, cst, Iter_\varphi(a_1^*, a_2^*)))$

$Iter_{\varphi}$ (Ajout(a_1^*,a),svide) = =

Ajout($Iter_{\varphi}$(a_1^*,svide), φ(a,w,cst,$Iter_{\varphi}$ (a^*,svide)))

$Iter_{\varphi}$ (svide, Ajout(a_2^*,a_2)) = = svide

DEF-EXTERN

Vide(Svide) = = Vrai
Vide(Ajout(r^*,r)) = = Faux

Notations

Nous noterons $\bar{x}^*$ toute suite x^* récurente intervenant comme argument de la fonction .

Lorsque certains arguments sont absents dans la fonction φ, nous indquerons Λ dans le profil de la fonction.

Niveau 1

Les opérations habituellement associées aux suites sont, tantôt définies de manière algébrique, tantôt définies sous forme de théorèmes.

1. Test d'appartenance d'un élément à une suite : Appartenant

Appartenant : SUITE[ARG],ARG → BOOL

Appartenant(Svide,a) = = Faux
Appartenant(Ajout(a^*,a'),a) = = Vrai si Eg(a',a)
Appartenant(Ajout(a^*,a'),a) = = Appartenant(a^*,a')
si Non(Eg(a',a))

2. Accès au premier élément de la suite : Premier

Premier : SUITE[ARG] → ARG

Premier(Svide) = = ⊥
Premier(Ajout(a^*,a)) = = Premier(a^*) si Non(Vide(a^*))
Premier(Ajout(a^*,a)) = = a si Vide(a^*)

3. Accès au dernier élément de la suite : Dernier

Dernier : SUITE[ARG] → ARG

Dernier(Svide) = = ⊥
Dernier(Ajout(a^*,a)) = = a

4. Longueur de la suite : Taille

Taille : SUITE[ARG] → ENT

Taille(Svide) = = 0
Taille(Ajout(a^*,a)) = = Taille(a^*) + 1

5. Suite privée de son dernier élément : Début

Début : SUITE[ARG] → SUITE[ARG]

Début(Svide) = = Svide
Début(Ajout(a^*,a)) = = a^*

6. Suite privée de son premier élément : Queue

Queue : SUITE[ARG] → SUITE[ARG]

Queue(Svide) = = Svide
Queue(Ajout(a^*,a)) = = Svide si Vide(a^*)
Queue(Ajout(a^*,a)) = = Ajout(Queue(a^*),a) si
Non(Vide(a^*))

7. Concaténation de deux suites : Concat

$$\text{Concat} : \text{SUITE}[\text{ARG}_1], \text{SUITE}[\text{ARG}_2] \rightarrow \text{SUITE}[\text{UNION}[\text{ARG}_1, \text{ARG}_2]]$$

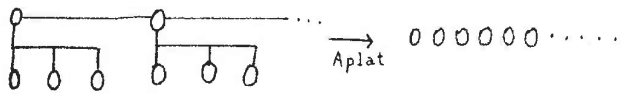
$$\text{Concat}(\text{arg}_1^*, \text{arg}_2^*) = \text{Dernier}(\text{Iter}_\varphi(\text{arg}_2^*, \text{svide}))$$

avec

$$\varphi : \text{ARG}_2, \Omega, \text{SUITE}[\text{ARG}_1], \text{SUITE}[\text{UNION}[\text{ARG}_1, \text{ARG}_2]] \rightarrow \text{SUITE}[\text{UNION}[\text{ARG}_1, \text{ARG}_2]]$$

$$\varphi(\text{arg}_2, \omega, \text{arg}_1^*, \bar{r}^*) = \underline{\text{si}}\ \text{Vide}(\bar{r}^*) \ \underline{\text{alors}}\ \text{Ajout}(\text{arg}_1^*, \text{arg}_2) \ \underline{\text{sinon}}\ \text{Ajout}(\text{Dernier}(\bar{r}^*), \text{arg}_2)$$

8. Mise à plat d'une suite de suites : Aplat



$$\text{Aplat} : \text{SUITE}[\text{SUITE}[\text{ARG}]] \rightarrow \text{SUITE}[\text{ARG}]$$

L'idée de la construction est de concaténer entre elles les différentes sous-suites de la suite de départ.

$$\text{Aplat}((a^*)^*) = \text{Dernier}(\text{Iter}_\varphi((a^*)^*, \text{svide}))$$

avec

$$\varphi : \text{SUITE}[\text{ARG}], \Omega, \Omega, \text{SUITE}[\text{SUITE}[\text{ARG}]] \rightarrow \text{SUITE}[\text{ARG}]$$

$$\varphi(a^*, \omega, \omega, \bar{r}^*) = \underline{\text{si}}\ \text{Vide}(\bar{r}^*)\ \underline{\text{alors}}\ a^*\ \underline{\text{sinon}}\ \text{Concat}(\text{Dernier}(\bar{r}^*), a^*)$$

9. Couplage deux à deux des éléments d'une suite : Couples

$$\text{Couples} : \text{SUITE}[\text{ARG}_1], \text{SUITE}[\text{ARG}_2] \rightarrow \text{SUITE}[\text{PC}[\text{ARG}_1, \text{ARG}_2]]$$

Invariant : $\text{Taille}(a_1^*) = \text{Taille}(a_2^*)$

$$\text{Couples}(a_1^*, a_2^*) = \text{Iter}_\varphi(a_1^*, a_2^*)$$

avec

$$\varphi : \text{ARG}_1, \text{ARG}_2, \Omega, \Omega \rightarrow \text{PC}[\text{ARG}_1, \text{ARG}_2]$$

$$\varphi(a_1, a_2, \omega, \omega) = \langle a_1, a_2 \rangle$$

10. Projections d'une suite de couples sur les suites constituantes

a) $\underline{\text{Découplage}_1}$: Projection sur la première suite constituante

$$\text{Découplage}_1 : \text{SUITE}[\text{PC}[\text{SEL}_1{:}\text{ARG}_1, \text{Sel}_2{:}\text{ARG}_2]] \rightarrow \text{SUITE}[\text{ARG}_1]$$

$$\text{Découplage}_1(a^*) = \text{Iter}_\varphi(a^*, \text{svide})$$

avec

$$\varphi : \text{PC}[\text{Sel}_1{:}\text{ARG}_1, \text{Sel}_2{:}\text{ARG}_2], \Omega, \Omega, \Omega \rightarrow \text{ARG}_1$$

$$\varphi(a, \omega, \omega, \omega) = \text{Sel}_1(:a)$$

b) $\underline{\text{Découplage}_2}$: Projection sur la seconde suite constituante

$$\text{Découplage}_2 : \text{SUITE}[\text{PC}[\text{Sel}_1{:}\text{ARG}_1, \text{Sel}_2{:}\text{ARG}_2]] \rightarrow \text{SUITE}[\text{ARG}_2]$$

$$\text{Découplage}_2(a^*) = \text{Iter}_\varphi(a^*, \text{svide})$$

avec

$$\varphi : \text{PC}[\text{Sel}_1{:}\text{ARG}_1, \text{Sel}_2{:}\text{ARG}_2], \Omega, \Omega, \Omega \rightarrow \text{ARG}_2$$

$$\varphi(a, \omega, \omega, \omega) = \text{Sel}_2(:a)$$

11. Accès au i^{er} élément d'une suite : Accès

Accès : SUITE[ARG],NAT → ARG

Accès(Svide,i) = ⊥
Accès(Ajout(a^*,a),i) = a si Eg(Taille(Ajout(a^*,a)),i)
Accès(Ajout(a^*,a),i) = Accès(a^*,a)
si Non(Eg(Taille(Ajout(a^*,a)),i))

12. Test pour savoir si une suite est une sous-suite : Est-sous-suite

Est-sous-suite : SUITE[ARG],SUITE[ARG] → BOOL

Par convention, la deuxième suite est la sous-suite :

Est-sous-suite(Svide,a_2^*) = Faux
Est-sous-suite(a_1^*,Svide) = Vrai
Est-sous-suite(Ajout(a_1^*,a_1),a_2^*) =
Est-sous-suite(Ajout(Queue(a_1^*),a_1),Queue(a_2^*))
si Eg(Prem(a_1^*),Prem(a_2^*))
Est-sous-suite(Ajout(a_1^*,a_1),a_2^*) =
Est-sous-suite(Ajout(Queue(a_1^*)a_1),a_2^*)
si Non(Eg(Prem(a_1^*),Prem(a_2^*)))

13. Complémentaire d'une suite par rapport à une sous-suite : Extr-sous-suite

Extr-sous-suite : SUITE[ARG],SUITE[ARG] → SUITE[ARG]

Par convention, la deuxième suite est la sous-suite :

Invariant : Est-sous-suite(a_1^*,a_2^*)

Extr-sous-suite(Svide,a_2^*) = ⊥
Extr-sous-suite(Ajout(a_1^*,a_1),Svide) = Ajout(a_1^*,a_1)
Extr-sous-suite(Ajout(a_1^*,a_1),a_2^*) = Concat(Prem(a_1^*),
Extr-sous-suite(Ajout(Queue(a_1^*),a_2^*))
si Non(Eg(Prem(a_1^*),Prem(a_2^*)))
Extr-sous-suite(Ajout(a_1^*,a_1),a_2^*) =
Extr-sous-suite(Ajout(Queue(a_1^*),a_1),Queue(a_2^*))
si Eg(Prem(a_1^*),Prem(a_2^*))

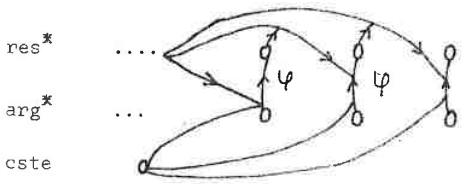
Niveau 2

Les opérations propres à la gestion se définissent par extension des opérations précédemment décrites.

Nous avons retenu cinq opérations qui, par expérience, nous paraissent être caractéristiques de transformations recontrées dans les applications de gestion.

1. La transformation simple

La suite résultat (res^*) est définie par applications successives d'une même transformation φ sur chaque argument choisi dans la suite argument(arg^*) en prenant l'élément suivant l'argument pris pour former le résultat précédent; la fonction φ peut de plus faire intervenir un paramètre constant (cste) ainsi que la suite des résultats précédemment définis.



Nous noterons la transformation de la manière suivante :

$$res^* = Simple(arg^*, \varphi)$$

Exemple d'application :

Il s'agit de facturer des commandes reçues; pour chaque facture, il faut lui affecter un numéro (son calcul dépend des factures antérieurement envoyées) et il est nécessaire de consulter le catalogue des produits :

$$fact^* = Simple(cmde^*, Facturation\ (cmde, \overline{fact}^*, catal\text{-}prod))$$

Sa définition est :

$$Simple_{\varphi} : SUITE[ARG] \rightarrow SUITE[RES]$$

$$\varphi : ARG, PAR, SUITE[RES] \rightarrow RES$$

$$Simple(arg^*, \varphi) = Iter_{\varphi'}(arg^*, svide)$$

avec

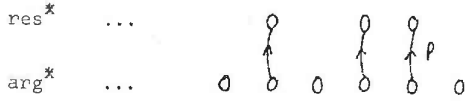
$$\varphi'(arg^*, \omega, cste, \overline{res}^*) = \varphi(arg^*, cste, \overline{res}^*)$$

Des variantes de la transformation sont les cas où :

- la récurence n'intervient pas : $\varphi(arg, cste, \omega)$
- il n'y a pas de paramètre : $\varphi(arg, \omega, \overline{res}^*)$

2. La transformation de filtrage

La suite résultat (res^*) se définit comme une sous-suite de la suite donnée (arg^*) dont chacun des éléments vérifie une propriété donnée (p); cette propriété peut faire intervenir un paramètre donné (cste) et la sous-suite déjà construite ($\overline{res}^*$) à un instant donné :



Nous noterons la transformation suivante :

$$res^* = Filtrage(arg^*, p)$$

Exemple d'application :

Il s'agit de sélectionner toutes les factures dont le montant dépasse un certain seuil :

$$\text{fact-sel}^* = \text{Filtrage}(\text{fact}^*, \text{montant-important}(\text{fact-seuil}))$$

$$\text{montant-important}(\text{fact-seuil}) = \underline{\text{si}}\ \text{montant (facture)} \gg \text{seuil}$$
$$\underline{\text{alors}}\ \text{vrai}$$
$$\underline{\text{sinon}}\ \text{faux}$$

La définition est la suivante :

$$\text{Filtrage}_p : \text{SUITE}[\text{ARG}] \rightarrow \text{SUITE}[\text{ARG}]$$
$$p: \text{ARG}, \text{PAR}, \text{SUITE}[\text{ARG}] \rightarrow \text{BOOL}$$
$$\text{Filtrage}(\text{arg}^*, p) = \text{Iter}_{\varphi}(\text{arg}^*, \text{svide})$$

avec

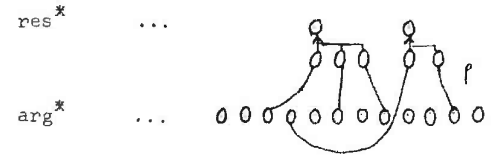
$$\varphi(\text{arg}, \omega, \text{cste}, \overline{\text{res}^*}) = \underline{\text{si}}\ p(\text{arg}, \text{cste}, \overline{\text{res}^*})$$
$$\underline{\text{alors}}\ \text{arg}$$
$$\underline{\text{sinon}}\ \omega$$

Les variantes de cette transformation sont les cas où :

- la récurrence n'intervient pas : $p(\text{arg}, \text{cste}, \omega)$
- il n'y a pas de paramètre : $p(\text{arg}, \omega, \overline{\text{res}^*})$

3. La transformation de groupage

Chacun des éléments de la suite résultat (res^*) correspond à une sous-suite de la suite donnée (arg^*) vérifiant une certaine propriété; toute propriété peut faire intervenir un paramètre (cste) et la suite des résultats déjà construite ($\overline{\text{res}^*}$). Un élément de la suite donnée peut se trouver au plus dans un seul groupe.



Nous noterons la transformation de la manière suivante :

$$\text{res}^* = \text{Groupage}(\text{arg}^*, p)$$

Exemple d'application :

a) Il s'agit de regrouper des factures en lots d'un certain nombre :

$$\text{lot-fact}^* = \text{Groupage}(\text{fact}^*, \text{lot-atteint})$$
$$\text{lot-atteint}(\text{fact}, \text{taille-lot}, \overline{\text{lot-fact}}) =$$
$$(\text{Taille}(\overline{\text{lot-fact}}) + 1 < \text{taille-lot})$$

b) Il s'agit de regrouper des factures par pays où elles sont adressées :

$lot\text{-}fact^{*}$ = Groupage($fact^{*}$,lot-pays)
lot-pays(fact,$\overline{lot\text{-}fact}$) = (Pays(:fact) =
(Pays(:Dernier($\overline{lot\text{-}fact}$)))

Notations :

- $\overline{res}$ indique le groupe en cours de construction
- $\overline{res}^{*}$ indique les groupes déjà construits

La définition est la suivante :

$Groupage_{p}$: SUITE[ARG] → SUITE[SUITE[ARG]]

p : ARG,PAR,SUITE[ARG],SUITE[SUITE[ARG]] → BOOL

Groupage(arg^{*},p) = $Iter_{\varphi}$(arg^{*},svide)
(avec p(arg,cste,$\overline{res}$,$\overline{res}^{*}$))

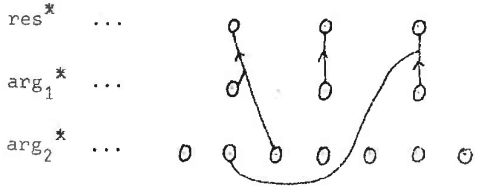
Le principe de construction consiste à itérer une opération de filtrage sur la suite de départ en évitant de sélectionner deux fois un même élément (extraction des éléments déjà sélectionnés).

avec φ (arg,ω,($\overline{res}^{*}$,cste,arg^{*}),$\overline{res}$) =
si Vide(arg'^{*})
alors ω
sinon Filtrage(arg'^{*},p')

où arg'^{*} = Extr-sous-suite(arg^{*},Concat(Aplat($\overline{res}^{*}$,$\overline{res}$)))
p'(arg',(cste,$\overline{res}$),$\overline{arg'}^{*}$) = si vide($\overline{arg'}^{*}$)
alors vrai
sinon p(arg',cste,res,$\overline{arg'}^{*}$)

4. La transformation de couplage :

Chacun des éléments de la suite résultat (res^{*}) est un couple dont le premier composant est, soit un élément de la seconde suite donnée ($arg_2{}^{*}$,) s'il vérifie une propriété p et s'il n'a pas déjà été choisi pour former un autre couple, soit l'élément vide.



Nous noterons la transformation de la manière suivante :

res^{*} = Couplage($arg_1{}^{*}$,$arg_2{}^{*}$,p)

Exemple d'application :

On associe à chacune des commandes sa facture si du moins elle existe.

$cmde\text{-}fact^{*}$ = Couplage($cmde^{*}$,$fact^{*}$,fact-associée(cmde,fact))
fact-associée(cmde,fact) = Eg(Num-cmde(:cmde),Num-cmde
(:fact))

La définition est la suivante :

$$\text{Couplage}_p : \text{SUITE}[\text{ARG}_1], \text{SUITE}[\text{ARG}_2] \rightarrow \text{SUITE}[\text{PC}[\text{Selarg}_1:\text{ARG}_1, \text{Selarg}_2:\text{ARG}_2]]$$

$$p : \text{ARG}_1, \text{ARG}_2 \rightarrow \text{BOOL}$$

avec :

$$\text{Couplage}_p(\text{arg}_1^*, \text{arg}_2^*, p) = \text{Iter}_\varphi(\text{arg}_1^*, \text{svide})$$

Le principe de construction consiste à itérer, sur chaque élément de la suite arg_1^*, une fonction de recherche d'un élément de la suite arg_2^* vérifiant la propriété et n'ayant pas déjà été sélectionné.

$$(\text{arg}_1^*, w, \text{arg}_2^*, \overline{\text{res}}^*) =$$

si Vide(Filtrage(Extr-sous-suite(arg_2^*,int),p(arg_1,arg_2)))
alors ⟨arg_1, w⟩
sinon ⟨arg_1,Premier(Filtrage(Extr-sous-suite(arg_2^*,int),p(arg_1,arg_2)))⟩

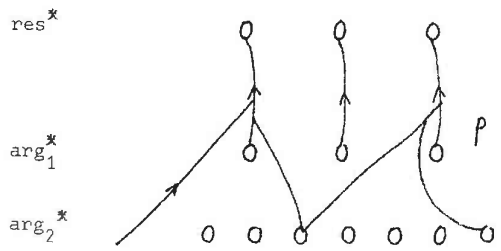
où int est la sous-suite des éléments de arg_2^* déjà sélectionnés

$$\text{int} = \text{Iter}_\Psi(\overline{\text{res}}^*, \text{svide})$$

$$\Psi(\overline{\text{res}}, w, w, w) = \underline{\text{si}}\ \text{Arg}_2\text{-present}(\overline{\text{res}})\ \underline{\text{alors}}\ \text{Selarg}_2(\overline{\text{res}})\ \underline{\text{sinon}}\ w$$

5. La transformation de Sélection

Chacun des éléments de la suite résultat (res^*) est une suite dont le premier élément appartient à la première suite donnée (arg_1^*) et dont la queue correspond à une sous-suite d'éléments appartenant à la seconde suite donnée (arg_2^*), éléments dont chacun vérifie une propriété (p) par rapport à l'élément de la première suite. Un élément de la seconde suite donnée peut intervenir dans la construction de plusieurs éléments de la suite résultat.



Nous noterons la transformation de la manière suivante :

$$\text{res}^* = \text{Sel}(\text{arg}_1^*, \text{arg}_2^*, p)$$

Exemple d'application :

Il s'agit de regrouper les différentes livraisons relatives à une même commande :

$$\text{gr-livr-cmde}^{*} = \text{Sel}(\text{cmde}^{*},\text{livr}^{*},\text{livraison-cmde}(\text{cmde-livr}))$$

$$\text{livraison-cmde}(\text{cmde-livr}) = \text{Eg}(\text{Num-cmde}(:\text{cmde}),\ \text{Num-cmde}(:\text{livr}))$$

Sa définition est la suivante :

$$\text{Sel}_{p} : \text{SUITE}[\text{ARG}_1],\ \text{SUITE}[\text{ARG}_2] \rightarrow \text{SUITE}[\text{PC}[\text{ARG}_1],\text{SUITE}[\text{ARG}_2]]$$

$$p : \text{ARG}_1,\text{ARG}_2,\text{PAR} \rightarrow \text{BOOL}$$

$$\text{Sel}(\text{arg}_1^{*},\text{arg}_2^{*},p) = \text{Iter}_{\varphi}(\text{arg}_1^{*},\text{svide})$$

avec : $\varphi(\text{arg}_1.\omega,(\text{arg}_2^{*},\text{cste}),\omega) = \text{Concat}(\text{arg}_1,\text{Filtrage}(\text{arg}',p'))$

$$p'(\text{arg}',(\text{arg},\text{cste})) = p(\text{arg},\text{arg}',\text{cste})$$

Niveau 3

Nous donnons ci-dessous quelques exemples d'opérations rendues nécessaires du fait de leur emploi fréquent et définies par extension des autres opérations.

1. Recherche dans une suite (arg^{*}) d'un élément (arg) dont la valeur d'un champ (Sel(:arg)) est supérieure à la valeur du même champ des autres éléments (s'il y en a plusieurs, on conserve le premier).

   Cette opération nécessite un paramètre qui est la fonction de sélection d'un champ dans un n-uplet.

   Nous notons l'opération :

   $$\text{arg} = \text{Max}(\text{arg}^{*},\text{Sel})$$

   Sa définition est :

   $$\text{Max} : \text{SUITE}[\text{ARG}],\text{Sel}:[\text{ARG}\rightarrow\text{VAL}] \rightarrow \text{ARG}$$

   $$\text{Max}(\text{arg}^{*},\text{Sel}) = \text{Dernier}(\text{Simple}_{\varphi}(\text{arg}^{*}))$$

   avec

   $$\varphi(\text{arg},\text{Sel},\overline{\text{arg}^{*}}) = \underline{\text{si}}\ \text{Vide}(\overline{\text{arg}^{*}})\ \underline{\text{alors}}\ \text{arg}\ \underline{\text{sinon si}}\ \text{Sel}(\text{arg}) > \text{Sel}(\text{Dernier}(\overline{\text{arg}^{*}}))\ \underline{\text{alors}}\ \text{arg}\ \underline{\text{sinon}}\ \text{Dernier}(\overline{\text{arg}^{*}})$$

   Exemple d'application :

   Il s'agit de trouver, parmi les factures, celles dont le montant facturé est le plus important :

   $$\text{fact} = \text{Max}(\text{fact}^{*},\text{montant-facturé})$$

2. Le résultat est obetnu par application d'une loi de composition (+,*,-,..) sur tous les éléments d'une suite sur base d'un champ commun à tous.

Cette opération nécessite deux paramètres :

- la fonction de sélection du champ concerné (Sel)
- la loi de composition (op)

Nous notons l'opération :

$$res = Total(arg^*, Sel, op)$$

Invariant : $Non(Vide(arg^*))$

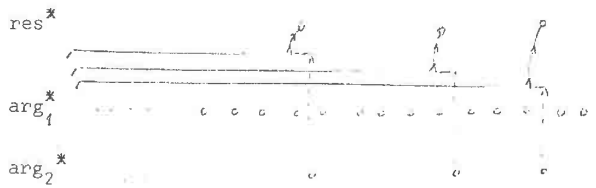
Sa définition est :

$$Total(arg^*, Sel, op) = Dernier(Simple(arg^*, \Psi))$$

avec

$$\varphi(arg,(Sel,op),\overline{res}^*) = \underline{si}\ Vide(\overline{arg}^*)$$
$$\underline{alors}\ Sel(:arg)$$
$$\underline{sinon}\ op(Sel(:arg),$$
$$Sel(:Dernier(\overline{res}^*)))$$

3. Le $n^{\underline{ième}}$ élément de la suite résultat(res^*) correspond à une sous-suite de la première suite d'arguments ($arg_1{}^*$), sous-suite caractérisée par le fait que chaque élément de cette sous-suite a une date de survenance inférieure à celle du $n^{\underline{ième}}$ élément d'une seconde suite d'arguments ($arg_2{}^*$)



Nous notons l'opération :

$$res^* = Historiques(arg_1{}^*, arg_2{}^*)$$

Sa définition est :

$$Historiques : SUITE[ARG_1], SUITE[ARG_2] \rightarrow SUITE[SUITE[ARG_1]]$$

Invariant : $Date(premier(arg^*)) < Date(Premier(arg_1{}^*))$

$$Historiques(arg_1{}^*, arg_2{}^*) = Simple(arg_2{}^*, \Psi)$$

avec $\Psi(arg_2, arg_1{}^*, \omega) = Queue(Sel(arg_2, arg_1{}^*, p))$

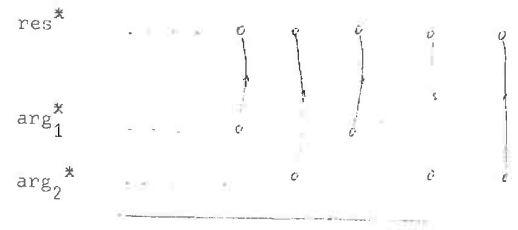
$$p = (Date(arg_1) < Date(arg_2))$$

Exemple d'application :

Il s'agit lors de survenances de requêtes, de pouvoir accéder, pour chacune des requêtes, à toutes les écritures comptables passées avant la requête :

$$lot\text{-}écri\text{-}ante^* = Historiques(ecri^*, req^*)$$

4. La suite résultat (res^*) est le résultat d'un interclassement de deux suites arguments ($arg_1{}^*$,$arg_2{}^*$), l'interclassement se faisant suivant les dates associées aux éléments des deux suites.



Nous notons l'opération :

$res^* = Fusion(arg_1^*, arg_2^*)$

Sa définition est :

$Fusion : SUITE\,[ARG_1], SUITE\,[ARG_2] \rightarrow SUITE\,[UNION\,[ARG_1, ARG_2]]$

$Fusion(arg_1^*, arg_2^*) = Aplat(res^*)$

avec : $res^* = Simple(int^*, \Psi(int, \overline{res}^*))$

$int^* = Sel(arg_1^*, arg_2^*, p)$

$p = (Date(arg_2) < Date(arg_1))$

avec : $\Psi(int, \overline{res}^*)$ = si Ou (Vide($\overline{res}^*$), Taille(int) = 1)

alors int

sinon Concat(Extr-sous-suite

(Queue(int), Déjà-choisis($\overline{res}^*$)),

Premier(int))

avec : Déjà-choisis ($\overline{res}^*$) = Aplat(Simple($\overline{res}^*$, $\Psi(\overline{res}^*)$))

$\Psi(\overline{res}^*)$ = si Taille($\overline{res}^*$) = 1

alors w

sinon Debut($\overline{res}^*$)

Exemple d'application :

Chacun des bordereaux produits est soit une facture, soit un bon de livraison

$bord^* = Fusion(fact^*, livr^*)$

Ci-dessous est présenté un panorama des différentes opérations introduites sur les suites et utilisables par le spécifieur lors de la description de son système d'information:

Niveau 1 : Opérations caractéristiques sur les suites

Appartenant : test d'appartenance d'un élément à une suite

Premier : Accès au premier élément de la suite

Dernier : Accès au dernier élément de la suite

Taille : Longueur de la suite

Début : Suite privée de son dernier élément

Queue : Suite privée de son premier élément

Concat : Concaténation de deux suites

Aplat : Mise à plat d'une suite de suites

Couples : Couplage deux à deux des éléments de deux suites

Découplage : Projection d'une suite de couples sur les suites constituantes

Accès : Accès au $i^{ième}$ élément d'une suite

Est-sous-suite : Test pour savoir si une suite est une sous suite d'une autre suite

Extr-sous-suite : Complémentaire d'une suite par rapport à une sous-suite

Niveau 2 : Opérations de base sur les suites propres à la gestion

Simple : Itération d'une opération sur chacun des éléments d'une suite

Filtrage : Extraction d'une sous-suite d'éléments de la suite, chaque élément extrait étant caractérisé par une condition

Groupage : Association à une suite de différentes sous-suites, chacune étant caractérisée par une certaine condition

Couplage : Construction, à partir de deux suites, d'une suite résultat dont chaque élément est constitué d'un élément de la première suite et d'un élément de la seconde vérifiant une certaine condition

Sel : Construction à partir de deux suites, d'une suite résultat dont chaque élément est constitué d'un élément de la première suite et de tous les éléments de la seconde vérifiant une certaine condition.

Niveau 3 : Opérations d'enrichissement spécifique à la gestion

Max : Recherche de l'élément maximum d'une suite

Total : Application d'une loi de composition (+,-,..) sur tous les éléments d'une suite

Historiques : Extraction des sous-suites d'une suite, chaque sous-suite étant caractérisée par le fait que chaque élément de la sous-suite a une date de survenance inférieure à celle d'un élément d'une autre suite

Fusion : Interclassement de deux suites suivant la date associée à chacun des éléments.

⋮

Le spécifieur peut, s'il le désire, ajouter d'autres opérations à ce niveau, la définition de chacune de celles-ci étant obtenue par composition d'opérations des niveaux 1 et 2.

2° L'ensemble

A la différence du concept de suite, le concept d'ensemble ne repose pas sur le caractère ordonné de ses éléments; une des conséquences en est que certaines opérations caractéristiques définies sur les suites n'ont pas leur équivalent ici.

Niveau 0

Les opérations appartenant au noyau théorique de base sont les suivantes :

- L'ensemble vide : ∅
- L'adjonction d'un élément : Ajout
- L'itération d'une fonction φ sur chacun des éléments d'un ensemble, outre l'élément de l'ensemble, la fonction peut avoir également comme argument une paramètre (cste) : Iter
- Le prédicat de test d'un ensemble vide : Vide

La spécification formelle du constructeur Ensemble paramétré par le type RES est la suivante :

TYPE ENS[RES]

ENVIRONNEMENT
b : BOOL
OPERATIONS
CONSTRUCT ∅ : → ENS[RES]
Ajout : ENS[RES], RES → ENS[RES]
MODIF Iter : ENS[ARG] → ENS[RES]
EXTERN Vide : ENS[RES] → BOOL

DECLARATIONS
φ ARG, PAR → RES
ensa : ENS[ARG]
ensr : ENS[RES]
a : ARG
r : RES
cst : PAR
EQUATIONS
DEF-MODIF $Iter_{\varphi}(\emptyset) == \emptyset$
$Iter_{\varphi}(Ajout(ensa,a)) == Ajout(Iter_{\varphi}(ens), \varphi(a,cst))$

DEF-EXTERN $Vide(\emptyset) == Vrai$
$Vide(Ajout(ensa,a)) == Faux$

Remarque :

1) la définition algébrique de l'ensemble fait apparaître une des limites inhérentes à ce type de formalisme; en effet, dans la définition de $Iter_{\varphi}$, nous sommes amenés à spécifier un ordre de construction (celui-ci étant induit par la forme normale) alors qu'en fait, il n'y a pas d'ordre à respecter; (en effet, la construction des différents éléments peut se faire en parallèle); une définition davantage correcte serait :

$Iter_{\varphi}(ensa) == ensr$
tq $\exists a \in ensa : ensr = Ajout(ens_1, \varphi)$
et $ens_1 = Iter_{\varphi}(ensa / \{x\})$
$ens_1 : ENS[ARG]$

2) Certaines opérations sur les ensembles ont le même nom que des opérations sur les suites; le type d'opération à appliquer dans un contexte dépendra du type de l'élément (notion d'opération "surchargée").

Niveau 1

Pour les opérations caractéristiques des ensembles, nous donnons une définition algébrique (plutôt que sous forme de théorème).

1. Test d'appartenance d'un élément à un ensemble : Appartenant

Appartenant : ENS[ARG],ARG → BOOL

Appartenant(∅,a) = = Faux
Appartenant(Ajout(ensa,a'),a) = = Vrai si Eg(a,a')
Appartenant(Ajout(ensa,a'),a) = = Appartenant(ensa,a) si Non(Eg(a,a'))

2. Intersection de deux ensembles : Intersect

Intersect : ENS[ARG],ENS[ARG] → ENS[ARG]

Intersect(∅,ensa) = = ensa
Intersect(Ajout(ensa',a'),ensa) = = Ajout(Intersect(ensa',ensa) a') si Appartenant(ensa,a')
Intersect(Ajout(ensa',a'),ensa) = = Intersect(ensa',ensa) si Non(Appartenant(ensa,a'))

3. Union de deux ensembles : Uni

Uni : ENS[ARG],ENS[ARG] → ENS[ARG]

Uni(∅,ensa') = = ensa'
Uni(Ajout(ensa,a),ensa') = = Ajout(Uni(ensa,ensa'),a)

4. Différence de deux ensembles : Diff

Diff : ENS[ARG],ENS[ARG] → ENS[ARG]

Diff(ensa,∅) = = ensa
Diff(∅,ensa') = = ∅
Diff(Ajout(ensa,a),ensa') = = Uni(Diff(ensa,ensa'),a) si Non(Appartenant(ensa',a))
Diff(Ajout(ensa,a),ensa') = = Diff(ensa,ensa') si Appartenant(ensa',a)

5. Cardinalité d'un ensemble

Card : ENS[ARG] → ENTIER

Card(∅) = = 0
Card(Ajout(ensa,a)) = = Card(ensa) + 1

Niveau 2

Les opérations sur les ensembles propres à la gestion.

Nous avons retenu trois opérations dont l'objectif est le même que celui de certaines sur les suites.

Remarquons que les opérations de Groupage et de Couplage définies sur les suites ne trouvent pas d'opérations équivalentes sur les ensembles du fait que leur définition implique que la structure soit ordonnée.

1. La transformation simple

L'ensemble des résultats (ensres) est défini par application d'une fonction donnée (Ψ) sur chacun des éléments de l'ensemble des données (ensarg); la fonction Ψ peut avoir comme argument supplémentaire un paramètre donné (cste).

Nous noterons la transformation de la manière suivante :

ensres = Simple(ensarg, Ψ)

Sa définition est :

$Simple_{\Psi}$: ENS[ARG] → ENS[RES]
$Simple_{\Psi}(ensarg) = Iter_{\Psi}(ensarg)$

Exemple d'application :

Il s'agit d'augmenter le prix de tous les produits figurant dans le catalogue d'une certaine somme

catal-prod' = Simple(catal-prod, augmentation(prod-hausse))

2. La transformation de filtrage

L'ensemble des résultats (ensres) se définit comme un sous-ensemble de l'ensemble des données (ensarg); sous-ensemble dont chacun des éléments vérifie une propriété (p), celle-ci pouvant faire intervenir un paramètre (cste)

Nous noterons la transformation de la manière suivante :

ensres = Filtrage(ensarg,p)

Sa définition est :

$Filtrage_p$: ENS[ARG] → ENS[ARG]
$Filtrage_p(ensarg) = Iter_{\Psi}(ensarg)$
avec
Ψ(arg,cste) = si p(arg,cste) alors arg
sinon ω

Exemple d'application :

On extrait du catalogue des produits, les coordonnées des produits dont le poids dépasse un certain seuil :

lot-prod-sel = Filtrage(catal-prod,poids-important(prod,seuil))
poids-important = si poids(prod) ⩾ seuil alors vrai
sinon faux

3. La transformation de Sélection

Chacun des éléments de l'ensemble des résultats (ensres) est un ensemble dont un élément appartient au premier ensemble de données (ensarg) et dont les autres constituent un sous-ensemble d'un second ensemble de données (ensarg'); sous-ensemble dont chacun des éléments vérifie une propriété (p) par rapport à l'élément du premier ensemble.

Un élément du second ensemble peut être sélectionné plusieurs fois.

Nous noterons la transformation de la manière suivante :

ensres = Sel(ensarg, ensarg',p)

Sa définition est :

Sel_p ENS[ARG],ENS[ARG] → ENS[ENS[UNION[ARG,ARG']]]

Sel(ensarg,ensarg',p) = Iter$_{\varphi}$(ensarg) (p(arg,arg',cste))

φ (arg,ensarg',cste) = Filtrage(ensarg',p')

p'(arg',arg,cste) = p(arg,arg',cste)

Exemple d'application :

On affecte, à chacun des produits d'un catalogue, différentes fiches portant des caractéristiques de ce produit

Catal-enrichi = Sel(catal,fiches,rens-prod(prod,fiche))

rens-prod(prod,fiche) = si num-prod(prod) = num-prod(fiche)
alors vrai
sinon faux

Niveau 3

Nous donnons ci-dessous un exemple d'opération enrichissant l'ensemble des opérations déjà définies.

Il s'agit de calculer le pourcentage d'éléments différents d'un premier ensemble (ensarg) par rapport à un second (ensarg') :

Nous notons l'opération :

e = Pourcentage(ensarg,ens rg')

Sa définition est :

Pourcentage : ENS[ARG],ENS[ARG] → NAT

$$\text{Pourcentage(ensarg,ensarg')} = \frac{\text{Card(Différence(ensarg,ensarg'))}}{\text{Card(ensarg)}}$$

Exemple d'application :

On désire connaître le pourcentage de nouveaux produits dans le catalogue de l'année (catal) par rapport au catalogue de l'année dernière (catal')

Pourcentage(catal,catal')

E. La table (ou fonction)

Ce dernier constructeur constitue également un des composants essentiels facilitant la construction de la spécification d'un S.I.; en effet, dans beaucoup de problèmes de gestion, il y a une notion d'accès à un élément précis suivant une propriété identifiante.

Remarquons, en outre, l'utilisation de cette structure dans les travaux de Codd [COD,70] sur le modèle relationnel ou ceux de De Jong et Zloof dans le cas de description de petites applications de gestion [DEJ,75].

Les caractéristiques essentielles d'une table sont :

- L'accès direct à un élément par la connaissance de la valeur de son identificateur associé

- L'oubli d'une partie de l'histoire associée aux éléments présents dans la table (oubli de l'histoire des suppressions et modifications); l'ordre d'apparition des éléments présents à un instant dans la table reflète uniquement l'ordre des création.

Un exemple de table est :

TYPE DOC-CLIENT == TABLE [NUMERO-CLIENT → NOM-CLIENT]

Les opérations de base sont :

- L'insertion d'un nouvel élément dans la table : Créer

  Exemple :

  Créer(doc-cli,1022,Dupond)

  Cette opération étant valide si le numéro 1022 ne figure pas déjà dans la table.

- La modification d'un élément de la table : Mod

  Exemple :

  Mod(doc-cli,1022,Dupond)

  Cette opération étant valide si le numéro 1022 appartient à la table

- La suppression d'un élément de la table : Sup

  Exemple :

  Sup(doc-cli,1022)

  Cette opération étant valide si le numéro 1022 appartient à la table

- Fournir la liste des identificateurs de tous les éléments appartenant à la table : Domaine

- L'existence d'un élément dans la table sur base de la connaissance de la valeur de son identificateur associé : Exist

- L'accès à l'attribut d'un élément dans la table via la connaissance de la valeur de son identificateur associé : Accès.

La spécification formelle du constructeur table paramétré par les types s_1 et s_2 est la suivante :

TYPE TABLE $[S_1 \rightarrow S_2]$

ENVIRONNEMENT
BOOL
OPERATIONS
CONSTRUCT Tabvide : $\rightarrow$ Table $[S_1 \rightarrow S_2]$
Créer : Table$[S_1 \rightarrow S_2]$, S_1, $S_2 \rightarrow$ Table$[S_1 \rightarrow S_2]$
MODIF Sup : Table$[S_1 \rightarrow S_2]$, $S_1 \rightarrow$ Table$[S_1 \rightarrow S_2]$
Mod : Table$[S_1 \rightarrow S_2]$, S_1, $S_2 \rightarrow$ Table$[S_1 \rightarrow S_2]$
EXTERN Accès : Table$[S_1 \rightarrow S_2]$, $S_1 \rightarrow S_2$
Exist : Table$[S_1 \rightarrow S_2]$, $S_1 \rightarrow$ BOOL
Domaine : Table$[S_1 \rightarrow S_2] \rightarrow$ ENS$[S_1]$
DECLARATIONS
b : BOOL
t : TABLE
s_1, s_1' : S_1
s_2, s_2' : S_2

EQUATIONS

DEF-MODIF
Sup(Tabvide,s_1) = = Tabvide
Sup(Créer(t,s_1,s_2),s_1') = = t si Eg(s_1,s_1')
Sup(Créer(t,s_1,s_2),s_1') = = Créer(Sup(t,s_1'), s_1,s_2) si Non(Eg(s_1,s_1'))
Mod(Tabvide,s_1,s_2) = = Tabvide
Mod(Créer(t,s_1,s_2),s_1',s_2') = = Créer(t,s_1,s_2') si Eg(s_1,s_1')
Mod(Créer(t,s_1,s_2),s_1',s_2') = = Créer(Mod(t,s_1',s_2'),s_1,s_2) si Non(Eg(s_1,s_1'))

DEF-EXTERN
Accès(Tabvide,s_1) = = ω_{s_2}
Accès(Créer(t,s_1,s_2),s_1') = = s_2 si Eg(s_1,s_1')
Accès(Créer(t,s_1,s_2),s_1') = = Accès(t,s_1') si Non(Eg(s_1,s_1'))
Exist(Tabvide,s_1) = = Faux
Exist(Créer(t,s_1,s_2),s_1') = = Vrai si Eg(s_1,s_1')
Exist(Créer(t,s_1,s_2),s_1') = = Exist(t,s_1') si Non(Eg(s_1,s_1'))
Domaine(Tabvide) = = Svide
Domaine(Créer(t,s_1,s_2)) = = Ajout(Domaine(t), s_1)

Du point de vue du spécifieur, seules lui seront accessibles les opérations externes; en effet, nous introduisons une primitive de plus haut niveau permettant de construire une table, c'est-à-dire, de répercuter les créations, modifications et suppressions, à partir d'une suite de mises-à-jour.

Son format est le suivant :

table = Tabulation(maj*)
maj* = Simple(arg*,Def-maj(arg,$\overline{table}$)

avec

Def-maj(arg,$\overline{table}$) = Def-maj$_1$(arg,$\overline{table}$) si Pré$_1$(arg,$\overline{table}$)
Def-maj$_2$(arg,$\overline{table}$) si Pré$_2$(arg,$\overline{table}$)
⋮

où : - maj* est la suite des mises-à-jour à répercuter pour construire la table; chacun de ces éléments se définit en bijection avec une sous-suite des données (arg*), sous-suite dont chacune des données doit entraîner une modification par rapport à la table construite à ce moment ($\overline{table}$)

- Pré$_1$,Pré$_2$ sont des prédicats indiquant lorsqu'il y a une modification à apporter; Def-maj$_1$, Def-maj$_2$ décrivent la forme de la mise-à-jour.

La définition formelle de la primitive est la suivante :

1) Nous définissons d'abord la table à partir de la suite de mises-à-jour :

Tabulation : SUITE[MAJ = PC[a:A,b:B°]] → TABLE[A → B]

Tabulation(maj*) = Dernier(Simple(maj*, φ(maj,$\overline{table}$*))

avec

φ(maj,$\overline{table}$*) = si B-present(maj)
alors
si Exist(Dernier($\overline{table}$*),a¢maj))
alors Mod(Dernier($\overline{table}$*),a¢maj),
b¢maj))
sinon Créer(Dernier($\overline{table}$*),a¢maj),
b¢maj))
sinon Sup(Dernier($\overline{table}$*),a(maj))

2) Définition de la suite des mises-à-jour (maj*) : Def-majs

Def-majs$_{(Def\text{-}maj_1,Def\text{-}maj_2,Pré_1,Pré_2)}$ SUITE[ARG]
→ SUITE [MAJ]

avec
Def-maj$_1$: ARG,TABLE → MAJ
Def-maj$_2$: ARG,TABLE → MAJ
Pré$_1$: ARG,TABLE → BOOL
Pré$_2$: ARG,TABLE → BOOL

$$\text{Def-majs}(\text{arg}^*) = \text{Simple}(\text{arg}^*, \Psi(\text{arg}, \overline{\text{maj}}^*))$$

avec

$$\Psi(\text{arg}, \overline{\text{maj}}^*) = \underline{\text{si}}\ \text{Pre}_1(\text{arg}, \text{Tabulation}(\overline{\text{maj}}^*))$$
$$\underline{\text{alors}}\ \text{Def-maj}_1(\text{arg}, \text{Tabulation}(\overline{\text{maj}}^*))$$
$$\underline{\text{sinon}}$$
$$\underline{\text{si}}\ \text{Pre}_2(\text{arg}, \text{Tabulation}(\overline{\text{maj}}^*))$$
$$\underline{\text{alors}}\ \text{Def-maj}_2(\text{arg}, \text{Tabulation}(\overline{\text{maj}}^*))$$
$$\underline{\text{sinon}}\ \omega$$

Nous pouvons également définir d'autres opérations "de haut niveau"; elles correspondent aux opérations "de gestion" définies sur les suites; elles s'apparentent aux opérateurs du calcul relationnel introduits par Codd [COD,71] tels que Projection, Produit, Jointure...

Ces opérations - auxquelles nous donnons le même nom que celles sur les suites (opérations surchargées) - se définissent par rapport aux opérations sur les suites .

Ainsi, l'opération d'itération simple sur une table notée : Simple(table, φ(a,b)), se définit formellement :

$$\text{Simple}_\varphi : \text{TABLE}[A \rightarrow B] \rightarrow \text{SUITE}[\text{RES}]$$
$$\varphi : A, B \rightarrow \text{RES}$$
$$\text{Simple}(\text{table}, \varphi(a,b)) = \text{Simple}(a^* = \text{Domaine}(\text{table}), \varphi(a, \text{Accès}(\text{table}, a)))$$

↑ opération sur un ensemble

Il en est de même pour les autres opérations : Filtrage, Sélection.

3.3. CONCLUSION

Nous avons introduit dans ce chapitre, un certain nombre d'outils susceptibles d'aider le spécifieur dans sa tâche :

- D'une part, la possibilité d'utiliser des opérations sans avoir à les définir; celles-ci se rapportent à des types de base tels que Booléen, Entier, Caractère,...

- D'autre part, la possibilité d'utiliser des outils de construction de types à partir de types plus élémentaires.

  Ces outils de construction correspondent à un certain nombre de structures de données, telles que produit cartésien, union,...; chaque structure est caractérisée par ces opérations habituelles.

Parmi les constructeurs introduits, mentionnons le type "suite"; celui-ci a fait l'objet d'une attention particulière.

En effet, nous avons expérimenté que le concept de "suite" occupe un rôle privilégié dans la description des systèmes d'information.

Pour cette raison, en plus des opérations habituelles de manipulation de ces suites, nous en avons défini d'autres plus spécifiques au domaine d'application considéré.

En particulier, un certain nombre de ces opérations portent sur des suites à caractère "temporel", c'est-à-dire des suites pour lesquelles une date est associée à chacun des éléments.

Nous avons constaté que l'utilisation de telles opérations facilitent la description de problèmes à caractère temps-réel; d'autre part, elles permettent d'éviter un certain nombre de surspécifications (par exemple, l'introduction d'un fichier temporaire, d'un accumulateur,...).

Dans le chapitre suivant, nous proposons une approche permettant de construire, à partir du catalogue des types prédéfinis et des constructeurs, la spécification de l'énoncé et de l'univers d'un problème.

o o o

Chapitre 4
UN META-ALGORITHME DE SPECIFICATION

4.1. INTRODUCTION

Il est certain qu'écrire une bonne spécification est au moins aussi difficile que d'écrire un bon programme.

Il n'est, par conséquent, pas surprenant qu'un certain nombre d'outils et de démarches aient été proposés afin de faciliter la tâche du spécifieur (cfr. Le panorama présenté dans le chapitre 1).

Cependant, dans le contexte de système d'information à spéficier, nous pensons qu'il existe actuellement peu de méthodes suffisamment rigoureuses et structurées pour favoriser la construction correcte de la spécification.

Ce chapitre est, à ce titre, capital puisqu'il tend à proposer quelques aides en ce sens.

De manière informelle, le méta-problème de la spécification peut s'énoncer comme suit : assurer le passage d'une description imprécise des besoins à une spécification précise et rigoureuse.

Fournir une spécification formelle et prouver son adéquation à l'énoncé initial constitue une réponse à ce problème; cependant, un autre aspect, qui nous semble important, est la possibilité d'expliciter et structurer le processus de construction de la spécification [DAR,79] [SCH,83] .

Plus spécifiquement, à partir du catalogue des types prédéfinis et constructeurs introduits dans le chapitre 3, il s'agit de décrire le processus intellectuel mis en oeuvre quant à leur choix et à leur application pour obtenir, d'une part, la spécification de l'énoncé du problème (c'est-à-dire la spécification des différentes fonctions devant être remplies par le système) et, d'autre part, la spécification de l'univers du problème (c'est-à-dire la spécification des différents types de données manipulés). L'expression de ces différents choix se fera au travers d'une méthode (ou méta-algorithme de la spécification [FIN,79])que nous proposons.

L'application du méta-algorithme favorise une construction de la spécification respectant deux qualités qui nous paraissent fondamentales; à savoir : l'abstraction et la structuration :

- Le mécanisme d'abstraction permet, à un certain niveau d'analyse, d'exprimer un certain nombre d'informations jugées pertinentes à ce niveau tout en en ignorant d'autres considérées comme des détails de moindre importance. En particulier, l'application du mécanisme d'abstraction correspond à notre souci de définir le problème sans introduire de caractéristiques propres à une de ses solutions;

- Arrivé à un certain niveau d'abstraction, l'obtention d'une spécification structurée est assurée du fait de l'utilisation d'un certain nombre de mécanismes de structuration :

  . au niveau de l'énoncé : chaque relation introduite est spécifiée en termes de relations plus élémentaires; les liens entre celles-ci pouvant être spécifiés au moyen de connecteurs OU, ET, POUR TOUT, AVEC;

  . au niveau de l'univers : les types de données sont structurés sous la forme d'une hiérarchie où les liens entre un type et des types plus élémentaires s'expriment à l'aide des constructeurs (produit cartésien, union, suite, ensemble et table).

Pour résoudre le méta-problème de la spécification, on cherchera à le décomposer en un ensemble de méta-sous-problèmes à résoudre; associés à ces différents méta-sous-problèmes, nous proposerons un certain nombre de stratégies (appelées stratégies de spécification).

Chacune des stratégies est caractérisée par la mise en oeuvre d'un certain nombre de règles dont l'application permet l'expression d'une solution ou méta-sous-problème.

Le reste du chapitre se présente comme suit : dans la deuxième partie, les différentes stratégies de spécification retenues sont décrites; un méta-algorithme assurant le choix l'application et l'enchainement des stratégies est proposé dans la troisième partie; enfin, un exemple d'application du méta-algorithme est présenté dans la quatrième partie.

4.2. LES STRATEGIES DE SPECIFICATION

Comme indiqué, le processus de construction de la spécification se subdivise en deux tâches :

a) expliciter, de manière récursive, chacune des relations introduites entre résultats et arguments (données du problème ou données intermédiaires) depuis la relation initiale définissant le problème traité jusqu'à l'introduction de relations "prédéfinies" (c'est-à-dire primitives de construction appartenant à l'univers prédéfini) ou la "réutilisation" de relations précédemment explicitées;

b) introduire et structurer de manière progressive les différents types de données et de résultats introduits.

Ces deux tâches se déroulent de manière conjointe et descendante; c'est-à-dire qu'à un quelconque niveau de raffinement, nous avons à spécifier une relation introduite à un niveau supérieur définie par :

r = Rel(d)

où le type de r est RES, le type de d est DATA; son profil est par conséquent :

Rel : RES←DATA

Le processus d'analyse nous amène à approfondir l'analyse de :

- l'arbre des énoncés dont Rel est un noeud à la base d'un sous-arbre qu'il s'agira d'expliciter

- les arbres des types RES et DATA étant des noeuds figurant dans des arbres distincts et à la base, chacun, d'un nouveau sous-arbre (structure du type) qu'il s'agira d'expliciter.

Nous proposons un certain nombre de stratégies dont l'application permet d'obtenir la spécification de ces différents arbres. Le choix, à un certain moment, d'une stratégie plutôt que d'une autre dépend d'un certain nombre d'éléments tels que nature du résultat; nature de l'argument,...; en particulier, ce choix peut dépendre du niveau de raffinement où l'on se trouve.
A cet effet, nous proposons une démarche globale qui nous paraît convenir à l'analyse de la plupart des problèmes relatifs à des systèmes d'information.

4.2.1. Une démarche globale d'analyse

Sous-jacent au problème global du système d'information à spécifier, nous avons identifié un sous-problème rencontré fréquemment et qui présente les caractéristiques suivantes :

- le type du résultat est une suite pour laquelle est associée une date à chacun de ces éléments (suite temporelle);
- le type d'un des arguments est également une suite temporelle; il s'agit de la suite "guide"; celle-ci constitue un guide méthodologique important puisqu'elle simplifie la définition du résultat en permettant une définition itérative;

- les autres arguments, de types quelconques (suite, ensemble, table, ...) interviennent dans la construction d'un élément de la suite résultat.

Exemples :

1° Il s'agit de produire des factures (suites de résultats) à partir de livraisons (suite guide des arguments) et d'une table des prix des produits et d'une documentation sur les adresses des clients (ces deux derniers contituant les autres arguments).

2° Il s'agit de produire des livraisons (suite des résultats) à partir des messages de réapprovisionnement d'un produit en stock (suite guide d'argument) et d'une table des commandes dont les lignes portent sur les produits manquant en stock (autre argument intervenant dans la confection d'un résultat).

On notera dans les deux exemples, la différence entre les suites guides :

- dans le premier exemple : l'information concernant chaque livraison se retrouve sur la facture;

- dans le second exemple : l'information concernant le message de réapprovisionnement (en l'occurence, un identifiant de produit) est à la base de la recherche de l'information nécessaire à la confection d'une livraison (en l'occurence, la commande en attente du produit réapprovisionné).

Cependant, dans les deux cas, les suites guides remplissent bien leur rôle de suites "déclencheuses".

A partir de là, la démarche globale d'analyse repose sur l'utilisation de stratégies permettant :

- d'une part d'assurer la décomposition du problème initial en un certain nombre de ces sous-problèmes privilégiés; certaines propriétés globales pouvant être exprimées au niveau des résultats de ces sous-problèmes (par exemple, un interclassement entre deux suites);

- d'autre part, assurer la construction d'un des résultats de la suite associée au sous-problème privilégié à partir d'un des arguments de la suite guide.
  A cet effet, nous distinguons la partie du résultat toujours calculée de la même manière (factorisation de certaines propriétés) de la partie du résultat pour laquelle ce n'est pas le cas (propriétés différentes d'après certaines conditions).

Remarquons qu'arrivé au niveau de la construction de chacun des composants du résultat, il n'est plus possible de formuler de stratégie générale.

Il va de soi que cette démarche ne s'applique pas à l'analyse de tous les problèmes.

Cependant, dans bon nombre de cas, nous avons pu juger de sa pertinence; c'est pourquoi, nous en tiendrons compte par la suite dans la formulation des conditions d'utilisation associées à chacune des stratégies.

4.2.2. Présentation des stratégies

Nous distinguons trois classes de stratégies :

- les stratégies de réutilisation des connaissances
- les stratégies déductives
- les stratégies inductives.

Pour chacune de ces stratégies, après une présentation informelle, nous en donnerons :

1° L'énoncé d'un certain nombre de règles dont l'application permet d'exprimer la relation initiale en termes de relations plus élémentaires ainsi que les types des résultats et arguments en termes de types plus élémentaires;

2° Les préconditions éventuelles liées à la mise en oeuvre de la stratégie; il s'agit d'une liste de propriétés devant être vérifiées par le problème à spécifier (par exemple, le choix d'une stratégie d'itération sur la définition d'un élément d'une suite implique que le résultat soit de type "suite");

3° Un certain nombre d'heuristiques de choix de la stratégie; en effet, suivant des propriétés telles que la nature du résultat, le niveau de raffinement dans lequel on se trouve (en particulier, par rapport à la démarche globale d'analyse proposée)..., le spécifieur est amené à choisir une stratégie de spécification plutôt qu'une autre;

4° Enfin, une présentation de la spécification obtenue après application de la stratégie.

A. Stratégie de réutilisation des connaissances

Plutôt que de construire la spécification courante, cette stratégie consiste à chercher à rapprocher ce problème d'un problème déjà spécifié, c'est-à-dire de l'identifier :

- soit à une "primitive" : c'est-à-dire une relation prédéfinie;
- soit à un autre problème préalablement étudié dans l'analyse de l'application

Présentation :

au départ :

r = Rel (d)

Rel : RES←DATA
r : RES
d : DATA

Règles à appliquer :

1) Chercher dans l'univers prédéfini ou dans la partie déjà spécifiée de l'application, une relation dont la sémantique est équivalente à celle que l'on veut exprimer; soit :

r' = Rel' (d')

2) S'assurer pour la relation trouvée de la compatibilité des profils.
S'il s'agit d'une primitive, il faut s'assurer que les constructeurs des types génériques utilisés soient compatibles avec les constructeurs de types utilisés pour RES et DATA; ainsi dans l'exemple :

r' : X = $CONS_i$ [XI] RES = $CONS_i$ [RESI]
d' : Y = $CONS_j$ [YI] DATA = $CONS_j$ [DATAI]

S'il s'agit d'une relation déjà spécifiée, s'assurer de la concordance des types; ainsi dans l'exemple :

r' : RES
d' : DATA

3) Remplacer la relation à définir par la relation retenue à l'arrivée :

r = Rel' (d)

Rel' : RES←DATA
r : RES
d : DATA

Précondition :

Il faut que les domaines et codomaines de la relation soient complètement définis.

Heuristique :

Réutiliser une partie de la spécification déjà définie, relève de la capacité du spécifieur à percevoir, de façon globale, son problème; s'il s'agit d'utiliser une primitive, le constructeur utilisé dans la définition du type du résultat et/ou de l'argument constitue un guide (cfr. CH.3, Présentation de catalogues de primitives associés aux constructeurs de types).

Exemple d'application :

Il s'agit de spécifier la relation suivante :

adr = Recherche-adr-cli (no,doc-cli)

adr : ADRESSE-CLIENT
no : NUMERO-CLIENT
doc-cli : TABLE [NUMERO-CLIENT→ADRESSE-CLIENT]
Recherche-adr-cli : ADRESSE-CLIENT←NUMERO-CLIENT, TABLE [NUMERO-CLIENT→ADRESSE-CLIENT]

La recherche de l'adresse courante d'un client dans la documentation appropriée peut s'exprimer à partir d'une primitive associée au constructeur TABLE de la manière suivante :

adr = Accès (doc-cli,no)

Remarque :

Nous n'avons pas approfondi davantage, dans le cadre de ce travail, cette stratégie.
En particulier, nous ne proposons pas de véritables heuristiques permettant à partir d'un dialogue avec le demandeur d'identifier les parties de spécification pouvant être réutilisées.
Ces idées relatives à la réutilisation de connaissances se retrouvent :

- dans les systèmes à banque de connaissance ayant pour but la génération automatique de programmes à partir d'un dialogue avec les demandeurs (voir, par exemple, SAFE [BAL,77] ou PSI [GRE,76]);

- dans le système SPRAC [FOI,81][FOI,82] dont le but est la conception assistée de logiciels de base par réutilisation, à l'aide d'un système expert, de connaissances propres au domaine traité stockées dans une base.

B. Stratégies déductives

La démarche que nous suivons est descendante; à chaque niveau de raffinement, deux approches sont possibles :

- une approche inductive : c'est-à-dire guidée par l'analyse des données;
- une approche déductive : c'est-à-dire guidée par l'analyse des résultats.

Dans le cadre de ce travail, nous privilégeons davantage l'approche déductive.
En effet, nous avons expérimenté qu'une démarche induite par l'objectif à atteindre semble généralement mieux convenir; le demandeur ayant souvent plus de facilités à caractériser les résultats qu'il cherche à obtenir plutôt que les données dont il dispose.

Une telle démarche favorise un ordre d'étude totalement indépendant de l'ordre d'exécution; en outre, elle est également préconisée par certains langages de très haut niveau (cfr. BDL [HAM,77] , [FEA,80]).

Dans une approche déductive, la forme de la relation "Rel : RES←DATA" est un peu différente.

En effet, DATA est rarement complètement défini, du fait qu'il est assez difficile d'identifier a priori tous les arguments nécessaires sans une analyse davantage approfondie du résultat et de sa relation associée; par conséquent, nous écrirons plutôt :

Rel : RES←DATA ...

A chaque niveau d'analyse, nous appliquerons une des stratégies décrites ci-dessous; l'application de celle-ci fera apparaître, outre de nouvelles relations devant à leur tour être spécifiées, un certain nombre de données intermédiaire (c'est-à-dire des données distinctes des arguments de la relation initiale, intervenant dans le codomaine de chacune des nouvelles relations introduites et ne correspondant pas à des constantes d'un des types de base prédéfinis tel Boaleen, Entier,...).

A leur tour, des relations définissant ces données intermédiaires devront être introduites et étudiées.

Le schéma global de raffinement sera de la forme suivante au niveau de l'énoncé :

au départ :

$r = Rel\ (d,...)$

à l'arrivée :

$r_1 = Rel_1\ (d_1, di_1, ...di_n)$
$\vdots$
$r_m = Rel_m\ (d_m, di_1, ...di_n)$

avec

$di_1 = Reli_1\ (d,...)$
$di_n = Reli_n\ (d,...)$

Notons :

- L'introduction du "avec" correspond à un raffinement basé sur le schéma du "et séquentiel"
- Les résultats $r_1,...r_m$ sont des composants du résultat r
- Les données $d_1...d_m$ sont des composantes de la donnée d
- Les données $di_1...di_n$ sont les données intermédiaires communes aux relations "$Rel_1...Rel_m$"
- L'analyse se poursuivra en étudiant d'abord "$Rel_1,...Rel_m$" avant "$Reli_1,...Reli_n$".

Nous distinguons, par la suite, deux classes de stratégies déductives :

I. Les stratégies basées sur l'étude du type des résultats; celles-ci se décomposent en :

   1° stratégie d'éclatement d'un ensemble ou d'une suite

   2° stratégie de décomposition d'un ensemble ou d'une suite

   3° stratégie de construction d'une table

   4° stratégie de construction d'un produit cartésien

   5° stratégie de construction d'une union

II. La stratégie d'analyse par cas.

I. Stratégies basées sur l'étude du type des résultats

Cette stratégie consiste à déduire le schéma de raffinement de la relation considérée à partir de la description de la structure du type du résultat par application d'un constructeur de type sur des types plus élémentaires.

Notons qu'une telle approche basée sur la structuration des objets manipulés est à la base de beaucoup de méthodes d'aide à la programmation (par exemple, en gestion [JAC,75] ou [WAR,75]).

1° Stratégie d'éclatement d'un ensemble ou d'une suite

Cette stratégie s'applique au cas où le type du résultat se structure par application du constructeur SUITE ou ENSEMBLE. Son application permet de passer du problème " définir un ensemble/suite" au problème " définir deux ou plusieurs sous-ensembles/suites constitutifs"; des interactions possibles entre sous-ensembles/suites peuvent être, en outre, spécifiées.

Présentation :

au départ :

r = Rel (d,...) — Rel : RES ← DATA ; r : RES ; d : DATA

Règles à appliquer :

1) Vérifier que le type du résultat fait intervenir le constructeur suite ou ensemble :

   par exemple : RES = SUITE [ ]

2) Mettre en évidence les différentes sous-suites (sous-ensembles) constitutives de la suite (ensemble) résultat :

   par exemple :

   r_1, r_2 avec : r_1 : RES_1 = SUITE [ ]
   r_2 : RES_2 = SUITE [ ]

3) Spécifier à l'aide d'une des primitives définies sur les suites/ensembles, les éventuelles interférences entre sous-suites/ensembles mises en évidence (par exemple, sur une suite, les primitives de Couplage, Interclassement, Fusion...; sur un ensemble, les primitives de Différence, Intersection, Interclassement,...);

   Par exemple :

   r = Primitive (r_1,r_2) — Primitive : RES ← RES_1, RES_2

4) Définir chacune des sous-suites/ensembles mises en évidence; essayer de faire apparaître de nouvelles données intermédiaires communes;

   Par exemple :

   r_1 = Rel_1 (d,int,...) — Rel_1 : RES_1 ← DATA,INT,...
   r_2 = Rel_2 (d,int,...) — Rel_2 : RES_2 ← DATA,INT,...

5) Exprimer une relation définissant les données intermédiaires introduites;

   Par exemple :

   int = Reli (d,...) — Reli : INT ← DATA,...

A l'arrivée :

   r = Primitive (r_1,r_2) — r : RES, int : INT, d : DATA
   et
   r_1 = Rel_1 (d,int,...) — r_1 : RES_1 = SUITE [ ]
   r_2 = Rel_2 (d,int,...) — r_2 : RES_2 = SUITE [ ]

avec

int = Reli (d,...) Primitive : RES ← RES_1, RES_2

Rel_1 : RES_1 ← DATA, INT, ...

Rel_2 : RES_2 ← DATA, INT, ...

Reli : INT ← DATA, ...

Remarques :

1) Notons l'application des schémas suivants de raffinement : le schéma du "et séquentiel" et celui du "et non séquentiel";

2) La définition formelle de la relation Rel est :

Rel(d,...) = Primitive(Rel_1(d,Reli(d,...),...), Rel_2(d,Reli(d,...),...))

Précondition :

Il faut que le type de RES se structure de l'une des deux manières suivantes :

RES = SUITE [ ]

RES : ENS [ ]

Heuristique :

- D'une manière générale, cette stratégie sera choisie lorsque le résultat est perçu comme un ensemble/suite auquel il n'est pas possible d'associer un mécanisme de construction unique; celui-ci étant davantage propre à un sous-ensemble/suite du résultat.

- Plus particulièrement, cette stratégie est utilisée pour passer du problème global à spécifier à la mise en évidence des sous-problèmes privilégiés dont nous avons parlé dans la démarche globale d'analyse. Dans ce cas, le résultat est une suite temporelle se décomposant en deux ou plusieurs sous-suites auxquelles sont associés les sous-problèmes; d'éventuelles interactions d'ordre "temporel" pourront s'exprimer entre les sous-suites. Un exemple d'application de cette heuristique est donné ci-dessous.

Exemple d'application :

Soit un système de guichet automatique produisant, à partir de requêtes formulées par des utilisateurs, un certain nombre de messages.

La description initiale est la suivante :

tra^* = Guichet-auto (req^*,...)

Guichet-auto : SUITE [TRANSACTION] ← SUITE [REQUETE] ,...

tra^* : SUITE [TRANSACTION]

req^* : SUITE [REQUETE]

Notons l'utilisation du symbole * pour indiquer une suite d'éléments.

A un niveau plus fin d'analyse, on se rend compte qu'il existe deux sortes de messages :

d'une part, des transactions indiquant l'état du compte; d'autre part, des transactions indiquant l'historique des opérations relatives à un compte
Les deux catégories de transactions s'interclassent au fur et à mesure de la production de chaque transaction.
L'application de la stratégie aboutit à la spécification suivante :

tra^* = Fusion($etat^*$, $histo^*$)
$etat^*$ = Gestion-etats(req^*,ope^*,...)

et

$histo^*$ = Gestion-historiques(req^*,ope^*,...)

avec

ope^* = Def-operations (...)

tra^* : SUITE [TRANSACTION]
$etat^*$: SUITE [ETAT-COMPTE]
$histo^*$: SUITE [HISTORIQUE-COMPTE]
req^* : SUITE [REQUETE]
ope^* : SUITE [OPERATION]
fusion* : SUITE [TRANSACTION] ← SUITE [ETAT-COMPTE], SUITE [HISTORIQUE-COMPTE]
Gestion-etats : SUITE [ETAT-COMPTE] ← SUITE [REQUETE], SUITE [OPERATION] ...
Gestion-historiques : SUITE [HISTORIQUE-COMPTE] ← SUITE [REQUETE] , SUITE [OPERATION] ...
Def-operations : SUITE [OPERATION] ...

Notons la mise en évidence de données intermédiaires que sont les opérations passées sur un compte.

2° Stratégie de décomposition d'un ensemble ou de suite

Cette stratégie s'applique dans le cas où le type du résultat s'exprime en utilisant le constructeur SUITE ou ENSEMBLE. Elle a pour but de passer de la définition d'une suite/ensemble à la définition de chacun de ses éléments; on utilisera pour cela la primitive "Simple" dont la définition dépend du fait que l'on travaille sur un ensemble ou sur une suite (opérateur surchargé).

Présentation :

Au départ :

r = Rel(d,...)

Rel : RES ← DATA,...
r : RES
d : DATA

Règles à appliquer :

1) Vérifier que le type du résultat fait intervenir le constructeur suite ou ensemble; par exemple : RES = SUITE [RESI]

2) Mettre en évidence la donnée (ou la partie de la donnée) constituant la suite guide; par exemple : DATA = SUITE [DATAI]

3) Définir la relation exprimant la construction de chaque résultat; mettre en évidence d'éventuelles données intermédiaires communes; par exemple :

$r = r_i^*$ $\quad d = d_i^*$

$r_i = Rel_i\ (d_i, int, ...)$

4) Définir la relation de construction des données intermédiaires; par exemple :

int = Relint (d,...)

A l'arrivée :

$r = r_i^* \quad d = d_i^*$

$r_i^* = Simple(d_i^*\ Reli\ (d_i, int, ...))$

avec

int = Relint(d,...)

r : RES = SUITE [RESI]
d : DATA = SUITE [DATAI]
r_i : RESI int : INT
d_i : DATAI
Rel_i : RESI ← DATAI,INT...
Relint : INT ← DATA,...

Remarques :

1) Notons . l'application des schémas suivants de raffinement : le schéma du "pour tout" et celui du "et séquentiel";

2) La définition formelle de la relation Rel est :

Rel (d,...) = Simple(d,Reli(d_i,Relint(d,...),...))

Précondition :

Il faut que les types RES et DATA se structurent d'une des deux manières suivantes :

RES = SUITE[] DATA = SUITE[]

ou

RES = ENS[] DATA = ENS[]

Heuristiques :

- La mise en évidence d'une suite ou d'un ensemble d'éléments comme arguments du problème liée à la définition d'un résultat de type suite/ensemble constitue un guide méthodologique important.

  Dans la mesure où le lien entre résultat et argument ne s'exprime pas à l'aide d'une des primitives prédéfinies, l'application de la stratégie proposée est possible.

- Dans certains problèmes où l'on identifie plusieurs suites/ensembles guides mais pour lesquels la construction du résultat se fait de la même manière indépendamment du fait qu'un élément provient d'une suite/ensemble plutôt que d'une autre (par exemple : numéroter et dater des messages), il est nécessaire d'introduire une suite/ensemble intermédiaire constituée de l'interclassement des différentes suites/ensembles guides.

- En particulier, dans le cadre de la démarche globale d'analyse proposée, l'application de cette stratégie est possible au moment où on a identifié le sous-problème privilégié.

Exemple d'application :

Continuant à décrire le guichet automatique, nous allons spécifier le problème de la production des états de compte.

La relation initiale est la suivante :

$etat^*$ = Gestion-etats(req^*,ope^*,...)

$etat^*$: SUITE [ETAT-COMPTE]
req^* : SUITE [REQUETE]
ope^* : SUITE [OPERATION]
Gestion-etat : SUITE [ETAT-COMPTE]←SUITE [REQUETE], SUITE [OPERATION],...

L'application de la stratégie débouche sur la spécification suivante :

$etat^*$ = Simple($req\text{-}etat^*$,Def-etat(req-etat,ope^*,...))

avec

$req\text{-}etat^*$ = Filtrage (req^*,Demande-etat(req))

$req\text{-}etat^*$: SUITE [REQUETE]
req,$req\text{-}etat^*$: REQUETE = UNION [REQUETE-ETAT,..]
Def-etat : ETAT-COMPTE←REQUETE, SUITE [OPERATION],...
Demande-etat : Booleen← REQUETE

Notons l'utilisation de la primitive de "Filtrage" pour sélectionner, parmi les requêtes, celles relatives à la production d'un état de compte; de plus, on introduit le type UNION pour indiquer qu'il existe plusieurs catégories de requêtes.

3° Stratégie de construction d'une table

Cette stratégie s'applique dans le cas où le type du résultat s'exprime en utilisant le constructeur TABLE. Elle permet de passer de la définition globale de la table à la définition de chacune des mises à jour influant le contenu de la table.

Présentation :

Au départ :

r = Rel(d,...)

Rel : RES←DATA
r : RES
d : DATA

Règles à appliquer :

1) Vérifier que le type du résultat fait intervenir le constructeur TABLE;
par exemple : RES = TABLE [ ]

2) Introduire les types des objets composant la table; d'une part, le composant identifiant; d'autre part, le composant identifié (attribut).
par exemple :
RES = TABLE [RESID→RESAT]

3) Pour appliquer la primitive prédéfinie de construction d'une table, il s'agit d'identifier successivement :

- le type de la suite des mises à jour;
- la relation de définition d'une mise à jour, les arguments de celles-ci (avec l'introduction d'éventuelles données intermédiaires); parmi les arguments, il sera fréquent d'introduire l'état de la table tel qu'il est avant la répercussion de la mise à jour (notion de récurrence);
- ensuite, on raffine la relation de définition d'une mise à jour en identifiant :
  . les différentes formes de mises à jour (différentes relations)
  . les différentes conditions liées au choix d'une forme de mise à jour plutôt qu'à une autre (différents prédicats).

A l'arrivée :

La présentation est directement basée sur la primitive de construction de la table :

- au niveau i :

$$r = \text{Tabulation}\ (maj^*)$$
$$maj^* = \text{Simple}(d_i^*, \text{Def-maj}(d_i, \bar{r}, int, \ldots))$$

avec

$$int = \text{Relint}\ (d, \ldots)$$

$r, \bar{r}$: RES = TABLE [RESID → RESAT]
d : DATA = SUITE [DATAI]
d_i : DATAI maj : MISE-A-JOUR
int : INT
Def-maj : MISE-A-JOUR ← DATAI, RES, INT, ...
Relint : INT ← DATA, ...

- au niveau i + 1 : spécification de Def-maj

$$maj = \text{Def-maj}_1(d_i, \bar{r}, int, \ldots) \ \underline{si}\ \text{Prédicat}_1(d, \bar{r}, int, \ldots)$$
$$= \text{Def-maj}_2(d_i, \bar{r}, int, \ldots) \ \underline{si}\ \text{Prédicat}_2(d, \bar{r}, int, \ldots)$$
$$\vdots$$

maj : MISE-A-JOUR = PC [RESID, RESAT°]
Def-maj_1 : MISE-A-JOUR ← DATAI, RES, INT, ...
Def-maj_2 : MISE-A-JOUR ← DATAI, RES, INT, ...
Prédicat_1 : Booleen ← DATAI, RES, INT, ...
Prédicat_2 : Booleen ← DATAI, RES, INT, ...

Remarques :

1) Notons . l'application des schémas suivants de raffinement :

   - au niveau i : le schéma du "pour tout" et deux fois celui du "et séquentiel"
   - au niveau i + 1 : le schéma du "ou"

2) La définition formelle sous-jacente a été présentée lors de la description de l'opération de construction d'une table (cfr. CH.3).

Précondition :

Il faut que le type du résultat se structure en utilisant le constructeur TABLE.

Heuristique :

Outre le fait d'identifier que le résultat est une table, il convient de pouvoir mettre en évidence les différentes catégories de mises à jour ainsi que la suite à partir de laquelle on peut les construire.

Exemple d'application :

On désire construire une table associant à un client, identifié par son numéro, les différents comptes possédés auprès d'une banque.

La description initiale est la suivante :

doc-cptes-cli = Gestion-etat-cptes (ouv*,ferm*)

doc-cptes-cli : DOC-COMPTES-CLIENT
ouv* : SUITE [OUVERTURE-COMPTE]
ferm* : SUITE [FERMETURE-COMPTE]

Après application de la stratégie :

- au niveau i :

doc-cptes-cli = Tabulation (maj-cptes*)
maj-cptes* = Simple(mvt-cpte*,Def-maj-cpte (mvt-cpte,doc-cptes-cli))

avec

mvt-cpte* = Fusion (ouv*,ferm*)

doc-cptes-cli : DOC-COMPTES-CLIENT = TABLE[NUM-CLIENT → SUITE [COMPTE]]
maj-cptes* : SUITE [MISE-A-JOUR-COMPTES-CLIENT]
mvt-cpte : MOUVEMENT-COMPTE = UNION [OUVERTURE-COMPTE , FERMETURE COMPTE]
Def-maj-cpte : MISE-A-JOUR-COMPTES-CLIENT ← MOUVEMENT-COMPTE, DOC-COMPTES-CLIENT

- au niveau i + 1 : pour définir "Def-maj-cpte

maj-cpte = Ajout-Compte(mvt-compte, doc-cptes-cli)
si Ouverture-Compte(mvt-compte, doc-cptes-cli)
= Suppression-Compte(mvt-compte, doc-cptes-cli)
si Fermeture-Compte(mvt-Compte, doc-cptes-cli)

maj-cpte = MISE-A-JOUR-COMPTES-CLIENT
= UNION [NUM-CLIENT, SUITE [COMPTE]°]
Ajout-Compte : MISE-A-JOUR-COMPTES ← MOUVEMENT-COMPTE, DOC-COMPTES-CLIENT
Suppression-Compte : MISE-A-JOUR-COMPTES ← MOUVEMENT-COMPTE, DOC-COMPTES-CLIENT
Ouverture-compte : Booleen ← MOUVEMENT-COMPTE, DOC-COMPTES-CLIENT
Fermeture-Compte : Booleen ← MOUVEMENT-COMPTE, DOC-COMPTES-CLIENT

4° Stratégie de construction d'un produit cartésien

Cette stratégie s'applique dans le cas où le type du résultat s'exprime en utilisant le constructeur PRODUIT CARTESIEN.

L'application de cette stratégie permet de passer de la définition d'un objet à la définition de ses constituants.

Présentation :

Au départ :

r = Rel(d,...) Rel : RES ← DATA,...
r : RES
d : DATA

Règles à appliquer :

1) Vérifier que le type du résultat fait intervenir le constructeur Produit Cartésien; par exemple : RES = P.C. [ ]

2) Mettre en évidence les types constituant le produit cartésien ainsi que des objets de ces types; par exemple :

RES = P.C. $[RES_1, RES_2]$

$r_1 : RES_1$, $r_2 : RES_2$

3) Etablir les relations de définition de ces nouveaux objets; essayer de faire apparaître de nouvelles données intermédiaires communes: par exemple :

$r_1 = Rel_1(d,int,...)$ $\quad Rel_1 : RES_1 \leftarrow DATA,INT,...$

$r_2 = Rel_2(d,int,...)$ $\quad Rel_2 : RES_2 \leftarrow DATA,INT,...$

4) Exprimer une relation définissant les données intermédiaires introduites; par exemple :

$int = Reli(d,...)$ $\quad Reli : INT \leftarrow DATA,...$

A l'arrivée :

$r = (r_1,r_2)$ $\quad r : RES = P.C.[RES_1,RES_2]$

$r_1 = Rel_1(d,int,...)$ $\quad r_1 : RES_1,\ r_2 : RES_2,$

et $\quad d : DATA,\ int : INT$

$r_2 = Rel_2(d,int,...)$ $\quad Rel_1 : RES_1 \leftarrow DATA,INT,...$

avec $\quad Rel_2 : RES_2 \leftarrow DATA,INT,...$

$int = Reli(d,...)$ $\quad Reli : INT \leftarrow DATA,...$

Remarques :

1) Notons :

l'application des schémas suivants de raffinement : le schéma du "et séquentiel" et celui du "et non séquentiel".

2) La définition formelle de la relation Rel est :

$$Rel(d,...) = \langle Rel_1(d,Reli(d,...),...), Rel_2(d,Reli(d,...),...)\rangle$$

Précondition :

Il faut que le type du résultat soit du type produit cartésien : RES = P.C.[ ]

Heuristique :

- De manière générale, cette stratégie sera choisie lorsque le résultat se présente comme un agrégat dont les composants peuvent être étudiés séparément. Associée à ce mécanisme d'agrégation, est alors utilisée la technique de décomposition.

- Un cas plus particulier est celui présenté dans la stratégie au niveau du sous-problème privilégié; dans ce cas, le résultat se présente comme un agrégat dont une partie est toujours construite de la même manière tandis que l'autre ne l'est pas. Un des composants exprime alors les propriétés communes de l'objet tandis que l'autre présente ses caractéristiques individuelles.

Exemple d'application :

Soit le problème suivant : il s'agit de construire un bon de commande correctement libellé à partir d'une commande validable.

La relation initiale est la suivante :

cmde-val = Construction-cmde-valide(cmde,...)

cmde, cmde-val : COMMANDE

Construction-cmde-valide : COMMANDE←COMMANDE,...

La construction de la commande correcte repose sur deux tâches : d'une part, la construction de l'identité du client, d'autre part, la construction du corps de la commande.

La spécification est la suivante :

Cmde-val = (id-cmde,corps-cmde)
id-cmde = Def-id-cli(cmde,...)

et

corps-cmde = Def-corps(cmde,...)

cmde-val,cmde : COMMANDE = PC [IDENT-CLIENT,CORPS-COMMANDE]
id-cmde : IDENT-CLIENT
corps-cmde : CORPS-COMMANDE
Def-id-cli : IDENT-CLIENT←COMMANDE,...
Def-corps : CORPS-COMMANDE←COMMANDE,...

5° Stratégie de construction d'une union

Cette stratégie s'applique dans le cas où le type du résultat s'exprime en utilisant le constructeur UNION.

L'application de cette stratégie permet de passer de la définition d'un objet à une définition conditionnelle plus précise.

Présentation :

Au départ :

r = Rel(d,...)

Rel : RES←DATA
r : RES
d : DATA

Règles à appliquer :

1) Vérifier que le type du résultat fait intervenir le constructeur UNION; par exemple :

RES = UNION[ ]

2) Mettre en évidence les types intervenant dans l'union ainsi que les objets de ces types : par exemple :

RES = UNION [RES_1,RES_2]

r_1 : RES_1 , r_2 : RES_2

3) Mettre en évidence les différents prédicats précisant le domaine considéré des données; faire apparaître d'éventuelles données intermédiaires communes; par exemple :

$Pre_1(d,int)$	$Pré_1$: Booleen←DATA,INT
$Pre_2(d,int)$	$Pré_2$: Booleen←DATA,INT

4) Mettre en évidence les relations définissant les différents résultats; faire apparaître d'éventuelles nouvelles données intermédiaires communes; par exemple :

$r_1 = Rel_1(d,int',...)$	Rel_1 : RES_1←DATA,INT,...
$r_2 = Rel_2(d,int',...)$	Rel_2 : RES_2←DATA,INT,...

5) Mettre en évidence les prédicats conditionnant la production des différents résultats; faire apparaître d'éventuelles nouvelles données intermédiaires communes; par exemple :

$Pred_1(d,int,...)$	$Pred_1$: Booleen←DATA,INT,...
$Pred_2(d,int,...)$	$Pred_2$: Booleen←DATA,INT,...

6) Exprimer les relations définissant les données intermédiaires introduites; par exemple :

$INT = Reli(d,...)$	Reli : INT←DATA,...
$INT' = Reli'(d,...)$	Reli' : INT'←DATA,...

A l'arrivée :

$r = r_1$ si $Pre_1(d,int)$
$\quad r_2$ si $Pre_2(d,int)$

$Pred_1(d,int,...)$ et
$r_1 = Rel_1(d,int',...)$

ou

$Pred_2(d,int,...)$ et
$r_2 = Rel_2(d,int',...)$

avec
$int = Reli(d,...)$
$int' = Reli'(d,...)$

r : RES = UNION [RES_1,RES_2]
r_1 : RES_1 d : DATA int' : INT'
r_2 : RES_2 int : INT
Pre_1 : Booleen←DATA,INT
Pre_2 : Booleen←DATA,INT
$Pred_1$: Booleen←DATA,INT,...
$Pred_2$: Booleen←DATA,INT,...
Rel_1 : RES_1←DATA,INT',...
Rel_2 : RES_2←DATA,INT',...
Reli : INT←DATA,...
Reli' : INT'←DATA,...

Remarques :

1) Au niveau des contraintes, la contrainte suivante doit être respectée :

$Pred_1 \Rightarrow Pre_1$ et $Pred_2 \Rightarrow Pre_2$

En général, les prédicats Pre_1,Pre_2 seront respectivement identiques à $Pred_1$ et $Pred_2$; dans ce cas, le schéma de construction sera le suivant :

$r = [r_1,r_2]$
$Pred_1(d,int,...)$ et
$r_1 = Rel_1(d,int',...)$
ou
⋮

2) Notons :

l'application des schémas suivants de raffinement : le schéma du "ou", celui du "et séquentiel" et celui du "et non séquentiel"; (ce dernier pour définir Reli et Reli')

3) La définition formelle de la relation Rel est :

Rel(d,...) =

si $Pred_1$(d,Reli(di,...)) alors [Rel_1(d,Reli'(d,...),...)]
si $Pred_2$(d,Reli(di,...)) alors [Rel_2(d,Reli'(d,...),...)]

4) Nous n'avons pas développé de stratégie particulière pour le cas où le type du résultat est du type Produit cartésien étendu (PC°). Le schéma de raffinement reprend en fait le schéma basé sur le produit cartésien et celui basé sur l'union.

Ainsi, par exemple :

r = (a,b,c°)	RES : PC [A,B,C°]
a = Rel_1(...)	
et b = Rel_2(...)	a : A
et Pré(...) et	b : B
c = Rel_2(...)	c : C
avec ⋮	

Précondition :

Il faut que le type du résultat soit du type UNION :

RES = UNION [ ]

Heuristiques :

Seul le besoin de raffiner une relation entraîne l'application de cette stratégie (cfr. présentation ci-dessus des heuristiques concernant le produit cartésien).

Exemple d'application :

Soit le problème suivant de facturation : suivant le fait qu'on effectue une première livraison à un client suite à sa commande ou une livraison différée, le corps de la facture est établi différemment.

La relation initiale suivante :

corps-fact = Def-corps-fact(livr)

corps-fact : CORPS-FACTURE
livr : LIVRAISON
Def-corps-fact : CORPS FACTURE ← LIVRAISON,...

se raffine de la manière suivante :

corps-fact = [corps-fact-dir, corps-fact-diff]
Première livraison (livr) et
corps-fact-dir = Def-corps-fact-dir (livr)

ou

Livraison-différée(livr) et
corps-fact-diff = Def-corps-fact-diff(livr)

corps-fact : CORPS-FACTURE = UNION [CORPS-FACTURE-DIRECTE, CORPS-FACTURE DIFFEREE]

corps-fact-dir : CORPS-FACTURE-DIRECTE

corps-fact-diff : CORPS-FACTURE-DIFFEREE

Première-livraison : Booleen ← LIVRAISON

Livraison-différée : Booleen ← LIVRAISON

Def-corps-fact-dir : CORPS-FACTURE-DIRECTE ← LIVRAISON

Def-corps-fact-diff : CORPS-FACTURE-DIFFEREE ← LIVRAISON

II. Stratégie d'analyse par cas

A côté de stratégies basées sur l'étude des types de résultats, il en existe une autre, souvent utilisée à un niveau plus fin d'analyse : l'analyse par cas.

Celle-ci consiste, pour expliciter une relation Rel, à faire apparaître deux relations (Rel_1,Rel_2) de manière à ce que la relation initiale s'écrive

$$Rel = Rel_1 \vee Rel_2$$

Présentation :

Au départ :

r = Rel(d,...) Rel : RES ← DATA,...
r : RES
d : DATA

Règles à appliquer :

1) Faire apparaître la condition (prédicat) conditionnant la réalisation d'une relation plutôt que l'autre, par exemple :

Pré(d,...) Pré : Booleen ← DATA,...

2) Mettre en évidence les relations définissant les valeurs du résultat, faire apparaître d'éventuelles nouvelles données intermédiaires communes, par exemple :

$r = Rel_1(d,int,...)$ $\quad Rel_1$: RES←DATA,INT,...

$r = Rel_2(d,int,...)$ $\quad Rel_2$: RES←DATA,INT,...

3) Exprimer une relation définissant les données intermédiaires introduites, par exemple :

int = Reli(d,...) $\quad$ Reli : INT←DATA

A l'arrivée :

r = si Pré(d,...) alors Rel_1(d,int,...)
sinon Rel_2(d,int,...)

r : RES $\quad$ d : DATA $\quad$ int : INT
Pré : Booleen←DATA,...
Rel_1 : RES← DATA,INT,...
Rel_2 : RES←DATA,INT,...
Reli : INT←DATA

avec

int = Reli(d,...)

Remarque :

Notons l'application des schémas de raffinement suivants : le schéma du "ou" et le schéma du "et séquentiel".

Heuristique :

Dans le cas où le type du résultat ne s'exprime pas à l'aide d'un constructeur (par exemple, un type de base) mais que, néanmoins, un besoin est ressenti pour raffiner la définition du résultat suivant l'évaluation d'une condition, cette stratégie peut être retenue.

Exemple d'application :

Les frais de transport sur une facture sont calculés de manière différente selon qu'il s'agit d'une première livraison ou d'une livraison différée.

La relation initiale est la suivante :

frais = calcul-frais-transp(livr)

frais : FRAIS-TRANSPORT
livr : LIVRAISON
Gestion-frais-transp : FRAIS-TRANSPORT ← LIVRAISON

Se raffine de la manière suivante :

frais = si Première-livraison(livr)
alors Calcul-premier-frais(livr)
sinon Calcul-frais-différé(livr)

frais : FRAIS-TRANSPORT ; ENTIER
Calcul-premier-frais : FRAIS-TRANSPORT← LIVRAISON
Calcul-frais-différé : FRAIS-TRANSPORT ← LIVRAISON

C. Stratégies inductives

Dans la paragraphe qui précède, nous avons présenté en détail la démarche déductive. Comme nous l'avons indiqué, nous pensons que cette approche guidée par la nature des résultats constitue un excellent guide; en particulier dans la construction des premiers niveaux de la spécification.

Il va de soi, cependant, qu'une telle démarche n'est pas systématiquement applicable et qu'il est parfois nécessaire de recourir à une stratégie inductive dans laquelle on est guidé, cette fois, par la nature des données.

Dans une approche inductive, le schéma global de raffinement de l'énoncé est de la forme suivante :

Au départ :

(r,...) = Rel(d)

A l'arrivée :

(r,int,...) = Rel_1(d')

(r,int',...) = Rel_2(d")

avec

(r,...) = Reli(int,int',...)

Notons : .l'approche inductive repose généralement sur un partitionnement des données; l'analyse de chaque partition permet la construction de certains résultats mais aussi d'intermédiaires qui sont eux-mêmes à la base de nouveaux problèmes à étudier;

.l'ordre d'étude est le suivant : Rel_1,Rel_2 avant Reli

Exemple d'application :

Soit un problème de gestion des stocks dans lequel on s'occupe des commandes de clients ainsi que des livraisons des fournisseurs.

La relation initiale est la suivante :

(...) = Gestion-stocks(livr*,cmde*)

livr* : SUITE LIVRAISON-FOURNISSEUR
cmde* : SUITE COMMANDE-CLIENT

Se raffine, suite à une analyse inductive, de la manière suivante :

(mess-reap*) = Réception-livraison-fournisseur(livr*)

et

(cmde-valide*,cmde-invalide*) = Réception-cmdes-clients(cmde*)

avec

(facture*,...) = Gestion-livraisons(cmde-valide*,mess-reap*)

cmde-valide*,cmde-invalide* : SUITE[COMMANDE-CLIENT]
mess-reap* : SUITE[MESSAGE-REAPPROVISIONNEMENT]

où "cmde-invalide" est identifié comme un résultat du problème tandis que "mess-reapp" et "cmde-valide" sont identifiés comme des intermédiaires.

Nous n'avons pas développé, dans le cadre de ce travail, de stratégies générales concernant l'application de l'approche inductive.

Cependant, nous n'excluons pas l'utilisation d'une stratégie inductive dans le cadre de problèmes assez élémentaires, tels que :

- l'analyse de prédicats à résultat booléen :
- le cas où le résultat correspond à un composant de l'argument (par exemple, le résultat est l'adresse d'un client; cette adresse étant prise dans une table où l'on accède via la connaissance du nom du client).

Ces problèmes assez élémentaires sont caractérisés par le fait que les résultats et arguments de la relation sont complètement identifiés.

Deux types de stratégies sont possibles :

- une stratégie d'analyse par cas;
- une stratégie basée sur l'étude des types des arguments.

3. Le méta-algorithme

Dans le paragraphe qui précède, nous avons présenté différentes stratégies de spécification; en particulier, nous avons insisté sur l'intérêt présenté par les stratégies déductives.

Nous allons maintenant décrire un méta-algorithme assurant le choix et l'enchaînement des différentes stratégies; le processus de spécification adopté est descendant.

A un niveau i de raffinement, la spécification obtenue jusque là se structure sous la forme d'une double arborescence :

- d'une part, un arbre décrivant la partie "énoncé" de la spécification; c'est-à-dire les différentes relations introduites ainsi que leur structuration;
- d'autre part, un arbre décrivant la partie "univers" de la spécification; c'est-à-dire les différents types manipulés ainsi que leur structuration.

Le processus de spécification se poursuit alors, en choisissant une relation non encore définie; soit :

```
Rel : RES←DATA,...          r = Rel(d,...)
  r : RES
  d : DATA
```

où DATA se compose :

- d'une part, des données de départ du problème qui ont pu être déjà identifiées;

- d'autre part, de données intermédiaires (c'est-à-dire qui à leur tour devront être spécifiées) qui ont pu être déjà identifiées comme étant communes à cette relation et à une autre relation définie au même niveau.

Remarquons que lors de l'application de stratégies déductives, il est en général impossible d'identifier immédiatement tous les arguments nécessaires; ceux-ci ne pourront l'être sous une analyse davantage approfondie des différents résultats apparus et de leurs relations associées.

Par conséquent, l'approche descendante que nous préconisons doit nous permettre :

- de structurer davantage le type du résultat;

- de compléter et de structurer davantage le type des arguments;

- de raffiner la définition de la relation Rel par le choix d'une stratégie particulière de spécification; l'application de celle-ci fera apparaître de nouvelles relations et données intermédiaires (ainsi que leurs types associés).

Ce processus sera appliqué récursivement jusqu'à l'utilisation d'éléments appartenant à la "base de connaissance"; c'est-à-dire :

- des types de "base" ainsi que des types déjà construits à partir de ceux-ci;

- les relations prédéfinies ainsi que des relations déjà construites à partir de celles-ci.

La figure ci-dessous présente le méta-algorithme; l'application de celui-ci à la spécification de la relation Rel se fait de la manière suivante :

Specif-rel (Rel)

Specif-rel (Rel)

. Choix d'une stratégie : déductive, réutilisation de connaissances, inductive (cette dernière étant davantage applicable à un niveau assez fin de sous-problème où l'analyse guidée par le résultat n'est plus pertinente)

. Cas de

- Choix d'une stratégie de réutilisation de connaissances

  - s'asurer que l'on a défini complètement RES et DATA
  - s'assurer de la conformité des types résultats et arguments

- Choix d'une stratégie déductive

  a) Définir RES (c'est-à-dire exprimer RES soit à partir d'autres types plus élémentaires mais non encore définis, soit à partir de types appartenant à la base de connaissances);

  b) Faire apparaître les relations plus fines :

  $$Rel(d) = (Rel_1(...),...Rel_n(...));$$

  c) Pour chaque Rel_i : identifier dans la mesure du possible, ses arguments (d_i)

Si certains des arguments identifiés (d_i) appartiennent aux arguments de Rel(d)

Alors préciser éventuellement la structuration des types d'arguments (DATA) et des objets (d) (par exemple, à l'aide du sélecteur de champ dans le cadre d'un produit cartésien)

Si certains des arguments identifiés (d_i) sont des objets d'un nouveau type

Alors . typer ces nouveaux objets
. s'il y a moyen, définir ces nouveaux types par rapport à des types de base ou à d'autres types déjà introduits dans la base de connaissances
. exprimer d'éventuels invariants
. ajouter les nouveaux objets (et leur type) aux arguments de la relation Rel (c-à-d. d) s'il s'agit de données de départ du problème.

d) S'assurer que chacune des données préalablement identifiées pour la relation Rel (c-à-d. "d") intervient dans au moins une Rel_i;

e) S'assurer que chaque donnée identifiée pour chaque relation Rel_i intermédiaire introduite est:
   - soit une donnée de départ du problème;
   - soit une donnée intermédiaire; alors elle doit se retrouver également dans le domaine d'une autre relation intermédiaire.

Choix d'une stratégie inductive

. S'assurer que l'on a identifié complètement DATA

. Faire apparaître les relations plus fines :

$(Rel_1(...),...Rel_n(...))$

. Pour chaque Rel_i

Mettre en correspondance les arguments et résultats de Rel_i avec ceux de Rel(RES et DATA), préciser éventuellement les résultats.

. Pour chaque Rel_i introduite qui n'est pas une primitive ou une relation déjà décrite antérieurement :

. Appliquer Spécif-rel(Rel_i)

. Compléter les arguments de la relation Rel avec les arguments de la relation Rel_i identifiés comme des données de départ (ces arguments sont ceux découverts après avoir appliqué Specif-rel)

. Pour chaque donnée intermédiaire (qui n'est pas une donnée de départ) communes aux relations Rel_i identifiées :

. Appliquer la règle du AVEC et faire apparaître une relation ($Relint_i$) la définissant

. Identifier dans la mesure du possible les arguments de $relint_i$ ($dint_i$)

Si certains arguments identifiés ($dint_i$) appartiennent aux arguments de Rel(d)

Alors préciser éventuellement la structuration des types des arguments (DATA) et des objets (d)

Si certains arguments identifiés ($dint_i$) sont des objets d'un nouveau type

Alors . S'assurer qu'il s'agit bien de données de départ du problème

. Typer ces nouveaux objets

. S'il y a moyen, définir ces nouveaux types par rapport à des types de base ou à d'autres types déjà introduits dans la base de connaissance

. Exprimer d'éventuels arguments

. Ajouter les nouveaux objets (et leur type) aux arguments de la relation Rel

. Appliquer Specif-rel($Relint_i$)

. Compléter les arguments de la relation Rel avec les arguments de la relation $Relint_i$ idengifier après avoir appliqué Specif-rel (ces arguments sont des données de départ du problème)

Remarque :

Dans le processus de construction de la spécification, nous faisons appel à un certain nombre de mécanismes d'abstraction; ceux-ci s'apparentent à un certain nombre de mécanismes connus dans la littérature sous les noms de : classification, agrégation, généralisation et association.

Chacun peut concerner soit un traitement, soit une donnée; le premier mécanisme est relatif au traitement (ou à la donnée) en lui-même; les autres concernent les relations entre traitement (ou données).

Le mécanisme de classification :

Ce mécanisme permet de considérer une collection de traitements (ou de données) de plus haut niveau. La classe définie précise les propriétés partagées par chacun des éléments tout en ignorant leurs différences. Un élément est considéré comme une occurence d'une classe s'il possède toutes les propriétés définies dans la classe. Ainsi, Entier est une classe dont une occurence est 7; Client (dont les propriétés sont un numéro et un nom) est une classe dont une occurence est <2425,Dupond>

Au niveau des traitements, le calcul d'une ristourne (dont les propriétés sont différents niveaux de ristournes selon le montant d'une facture) est une classe dont une occurence est le calcul d'une ristourne comme étant 10 % du montant de la facture n° 254.

Le mécanisme d'agrégation :

Ce mécanisme permet de considérer un type de donnée ou de traitement comme un tout, alors qu'il s'agit en fait d'une collection de composants ou de parties; il existe, dès lors, une relation "est partie de" (au sens de [PAR,74]) entre l'agrégat et un des composants.

Des exemples d'utilisation de ce mécanisme sont le concept de relation dans le modèle de Codd ou la clause "Part of" dans le langage PSL [TEI,77].

Le mécanisme d'agrégation est utilisé dans le cadre de notre démarche :

- au niveau des types de données, lors de l'utilisation du constructeur "produit cartésien";
- au niveau des traitements, lors de l'utilisation du schéma de raffinement basé sur le connecteur ET.

Le mécanisme de généralisation :

Ce mécanisme permet à une classe de données ou de traitements (qualifiée de classe "générique") d'être caractérisée par les propriétés communes à plusieurs classes de données ou de traitements (ces différentes classes étant appelées "catégories") tout en ne considérant pas leurs caractéristiques individuelles.

Ainsi, la classe des personnes peut être considérée comme une généralisation des classes d'employés et de clients dans la mesure où elle partage des propriétés communes telles que, nom, adresse,... De même, la classe de traitement "Facturation" peut être considérée comme une généralisation des classes "Facturation-immédiate" et "Facturation-différée" dans la mesure où certaines opérations sont communes, telles que le calcul de la TVA, le calcul de la ristourne,...

Le mécanisme de généralisation a d'abord été largement utilisé dans le domaine de l'intelligence artificielle dans l'expression des réseaux sémantiques. Il a été ensuite exploité au niveau de la conception de bases de données en complément du mécanisme d'agrégation pour exprimer des structures de types de données basées sur le constructeur "union" [SMI,77] . Il a enfin été introduit récemment comme mécanisme d'abstraction au niveau de la spécification des types de données ([BRO,81] , [BRO,82]) et au niveau de la spécification des types de traitements [GRE,82] . Il est particulièrement bien adapté à la description de problèmes où la difficulté réside essentiellement dans le grand nombre de détails à exprimer.

Le mécanisme de généralisation est utilisé dans le cadre de notre démarche :

- au niveau des types de données, lors de l'utilisation du constructeur "union" ainsi que lors de l'utilisation du mécanisme d'enrichissement;
- au niveau des types de traitements, l'application de ce mécanisme est favorisée dans la démarche globale d'analyse lorsque pour la confection d'un résultat, on dissocie la partie du résultat toujours calculée de la même manière de la partie dont ce n'est pas le cas.

Le mécanisme d'association :

Ce mécanisme permet de considérer un ensemble répétitif de types de données ou de traitements comme un tout. Au niveau des types de données, le mécanisme d'association peut s'appliquer à des structures reposant sur la notion d'ensemble ou de suite; ainsi, le type "Catalogue-produits" est une association de "caractéristiques d'un produit". Ce mécanisme est également très important; en effet, les ensembles répétitifs sont très nombreux en informatique de gestion et il est, dès lors, important de pouvoir définir des propriétés globales sur ceux-ci. Le mécanisme d'association est utilisé dans l'approche ACM/PCM ([BRO,81] , [BRO,82]).

Le mécanisme d'association est utilisé dans le cadre de notre démarche :

- au niveau des types de données, lors de l'utilisation des constructeurs "suite" ou "ensemble";
- au niveau des types de traitements, lors de l'utilisation des primitives sur les suites ou ensembles.

A ces mécanismes, d'abstraction que sont l'agrégation, la généralisation et l'association, sont associés des schémas de raffinement. Ainsi, les agrégats et les association sont décrits par "composition de " (approche ascendante) ou "décomposition de " (approche descendante) de types plus élémentaires sans tenir compte de leurs propriétés. Par contre, les catégories sont définies comme des"spécialisations" (approche ascendante) dans lesquelles seules les propriétés les distinguant de la classe générique sont définies. Le processus inverse (approche descendante) s'appelle "généralisation".

Les deux techniques de raffinement bénéficient de l'héritage des propriétés. Ainsi, les agrégats et ensembles décrits bénéficient des propriétés de leurs composants (par exemple, les propriétés del'agrégat Client, défini ci-dessus, sont les propriétés de ses composants numériques et alphanumériques).

D'autre part, la technique de généralisation fait hériter aux catégories les propriétés caractérisant la classe générique (ainsi, dans l'exemple qui précède, les employés et les clients héritent chacun des propriétés de personne).

5. Illustration du déroulement du méta-algorithme

Dans ce paragraphe, nous illustrons le déroulement du méta-algorithme en l'appliquant au traitement d'un exemple.

Nous avons choisi le cas d'un système de gestion des commandes clients, celui-ci est caractérisé par des entrées et des sorties bien définies :

- en entrée :
  - des bons de commande en provenance de clients
  - des renseignements concernant les caractéristiques des produits vendus (libellé, prix,...) ainsi que les avoirs en stock de ces produits (stocks initiaux, rentrée de produits,...)
- en sortie :
  - des bons de commande jugés erronés (client inconnu, produit commandé inexistant,...)
  - des factures concernant les produits livrés
  - des commandes à destination des fournisseurs concernant d'éventuels produits épuisés.

Dans le cadre de ce travail, nous ne présentons qu'une partie de la spécification obtenue pour ce problème; celle-ci concerne le sous-problème de la production de factures à partir de livraisons préparées de produits (Remarquons, à ce sujet, que suite à une commande précise, plusieurs livraisons sont possibles suivant une disponibilité immédiate ou différée d'une partie ou de la totalité des produits commandés).

En annexe, pour ce sous-problème, le résultat de l'application du méta-algorithme et des stratégies invoquées est présenté.

La présentation de chacune des étapes est la suivante :

- Choix de la stratégie appliquée
- Définition du résultat (la plupart des cas, nous adoptons une stratégie déductive)
- Expression des relations définissant la construction de chaque résultat;
- Spécification de ces différentes relations (application itérative du méta-algorithme sur chacune de ces relations);
- Définition des données apparues dans ces relations (données de départ du problème et/ou données intermédiaires);
- Expression des relations définissant la construction de chaque donnée intermédiaire (application itérative du méta-algorithme sur chacune de ces relations).

A chaque moment de la construction de la spécification d'une relation, nous indiquons les différents états de la spécification obtenue au travers de la description progressive des différents pavés de traitement (formalisme des trois colonnes) associés aux relations.

Chapitre 5
EXPERIENCES DANS UN ENVIRONNEMENT INDUSTRIEL

Depuis deux ans, l'occasion nous a été donnée de pouvoir expérimenter un certain nombre de nos idées en matière de spécification dans un environnement industriel, à savoir, la Société Nationale Elf-Aquitaine Production (SNEA-P).

Cette expérience a été profitable d'un double point de vue :

1° D'une part, nous avons été amenés à appliquer notre approche à deux systèmes d'information. L'étude de ces applications nous a conduit progressivement à préciser et à approfondir certains aspects de notre recherche (citons, par exemple, l'aspect outils de construction de systèmes d'information avec en particulier les primitives de manipulation des suites).

2° D'autre part, de nombreux contacts avec des personnes ayant une certaine expérience en matière d'analyse fonctionnelle nous ont fait ressentir la nécessité de disposer d'un véritable guide méthodique dans la construction de la spécification. Cet aspect nous a conduit à développer davantage les aspects du méta-algorithme et par conséquent, le type du dialogue devant être tenu par le spécifieur auprès des utilisateurs.

Dans ce chapitre, après avoir évoqué brièvement l'environnement informatique de la société, nous rendons compte des deux applications que nous avons eues à traiter.

Dans cette présentation, nous ne discuterons pas de tous les avantages que nous semble procurer notre approche (spécification formelle, contrôles de cohérence et de complétude pouvant être effectués sur la spécification...) mais plutôt de quelques aspects caractéristiques de l'application de notre démarche à chacune des applications.

5.1. L'ENVIRONNEMENT INFORMATIQUE DE LA SNEA(P)

Le centre de traitement de l'information de la SNEA(P) a été créé à Pau en 1959; depuis cette année, ses activités n'ont cessé de s'accroître tant dans le domaine des applications de gestion que dans le domaine des applications scientifiques.

Quelques exemples des moyens dont le centre dispose suffisent à situer son importance : utilisation d'ordinateurs de haut de gamme de chez CII et IBM; existence d'un réseau interconnectant les différents matériels; traitement sur machine vectorielle (CRAY.1);...

Dans le domaine de développement de logiciels, le besoin d'améliorer la productivité a conduit à l'introduction de divers outils et techniques. Ainsi, dans le domaine de l'analyse de programmation et de la programmation, citons : l'utilisation du Fortran Structuré, l'utilisation de Pseudo-Code (P.D.L. [CAI,75]) lors de la rédaction de programme; enfin, plus récemment, la mise en oeuvre d'un projet de recherche relatif à l'emploi d'une méthode d'analyse structurée [CHA,77] .

Dans le domaine de la spécification, un projet de recherche a été initialisé il y a quelques années; celui-ci reposait sur l'utilisation de l'approche Merise [LER,82] .

Ce projet s'est avéré très positif :

- d'une part, parce que l'utilisation de l'approche a conduit à de meilleures structuration et description des données;

- d'autre part, parce que la facilité de compréhension du modèle entité-association (individu-relation) a permis de susciter des discussions constructives entre les divers utilisateurs.

L'intérêt porté par la société à notre projet provient du fait que, d'une part, nous utilisons un langage formel de spécification et d'autre part, que nous proposons quelques aides quant à la structuration et à la description des traitements.

5.2. L'APPLICATION GESTI

5.2.1. Présentation générale et historique du projet

Le but de l'application GESTI (**Gestion du centre de** Traitement de l'information) est d'assurer un suivi des frais engagés pour les applications informatiques (frais de personnel, heures machines,...) ainsi que de pouvoir, à tout moment, récolter des renseignements sur l'état d'avancement des différents travaux.

Initialement, (au début des années 70), le suivi était assuré manuellement; la décision d'automatisation du traitement sous le nom de GESTI-1 a été prise en 1971.

Ce premier système réalisé ne prenait pas en compte l'aspect comptable, il assurait uniquement un suivi sur base des temps.

A cette époque, la réalisation du système depuis l'étape de rédaction du cahier des charges jusqu'à l'étape d'implémentation a été confiée à une seule et même personne.

Ce premier système ayant suscité de l'intérêt de la part de la direction, il est décidé en 1973, de l'étoffer en développant et en complexifiant certaines de ces fonctions.

Pour développer GESTI-2, le responsable initial s'est vu adjoindre un technicien (davantage spécialisé en informatique); celui-ci ayant à charge de résoudre certains problèmes particuliers ainsi que d'implémenter l'application d'une manière plus performante.

Le nouveau système ainsi conçu permettait à la fois :

- un suivi des différents projets informatiques; celui-ci aidant au fonctionnement du service informatique;

- la génération d'écritures comptables à destination de la comptabilité analytique.

Au fur et à mesure des années, des modifications ont été apportées au système; citons, par exemple :

- de nouvelles ventilations dans la présentation des résultats;

- l'introduction d'un système de facturation commerciale;

- de nouvelles contraintes d'édition;

- ...

Un certain nombre de ces modifications ont pu être facilement apportées; celles-ci entraîneront surtout des modifications en aval et en amont de l'application proprement dite.

Par contre, d'autres modifications ont eu des conséquences plus directes sur l'application; au fur et à mesure de leur répercussion, le coût d'une modification s'est avéré progressivement de plus en plus important.

Le schéma ci-dessus indique clairement que le seuil de saturation au niveau du coût de la maintenance a été atteint courant 1978. (Remarquons que ce schéma est également applicable à la plupart des applications informatiques).

Un autre événement important à cette époque, a été l'absence du responsable principal pendant une certaine période; il en est résulté une impossibilité d'assurer l'exploitation convenable de l'application, principalement faute d'une documentation suffisante.

Toutes ces considérations, associées au désir d'utiliser de nouveaux moyens techniques (un système conversationnel et un système de base de données) ont conduit la direction informatique à lancer en 1978 le développement d'une troisième version du système (GESTI-3).

Un point important a été cette fois la décision de l'établissement d'un cahier des charges destiné à établir avec exactitude les besoins du demandeur, de manière à fournir un document précis de travail aux informaticiens.

Comme nous le verrons dans le paragraphe suivant, ce cahier des charges, même s'il constitue un progrès, ne peut être considéré comme une véritable spécification remplissant toutes les qualités nécessaires.

Bon nombre des problèmes apparus dans les stades ultérieurs de développement du système en sont la conséquence; citons ainsi :

- les informaticiens ont été amenés à tout moment à rediscuter avec les utilisateurs;

- des changements dans le personnel informatique entraînent le nouveau personnel à recomprendre tout ce qui a été développé par leurs collègues, et cela, en raison du manque de fiabilité du cahier des charges ainsi que de l'analyse de documentation des solutions déjà retenues.

La conséquence de ces différents retards a été le non respect de la date d'échéance initialement fixée à Janvier 81.

A cette époque, les responsables de la direction informatique n'ont pas accordé de délais supplémentaires et ont décidé de rétrocéder le travail d'achèvement de l'application à une personne extérieure de la société. Celle-ci a travaillé par la suite avec un seul interlocuteur chargé de centraliser les requêtes en provenance des divers demandeurs.

A partir de Mai 81, GESTI-3 est devenu peu à peu opérationnel; en Décembre 81, seules quelques erreurs au niveau comptable subsistent.

Notons qu'actuellement, la seule documentation concernant l'application est constituée du code macnine (rédigé en Fortran) accompagné de quelques phrases de commentaires.

5.2.2. Le cahier des charges de GESTI-3

Le cahier des charges se présente comme un document d'une quarantaine de pages rédigées en langue naturelle.

Dans cette description, on trouve des exemples des principaux écueils caractérisant habituellement une spécification [MEY,80] .

Donnons-en quelques exemples :

- Des silences (éléments non définis) :

> "Une écriture est valide lorsqu'elle répond aux conditions générales de traitement (ressource connue, affaire ouverte, tarif en vigueur, compte-client ouvert,...)".

Les conditions de validation ne sont pas complètement décrites.

- Des références en avant (éléments utilisés avant d'être définis :

> "A chaque standart correspond une unité d'oeuvre et les tarifs permettant de facturer cette unité d'oeuvre".

La notion d'"unité d'oeuvre" n'est définie que deux pages plus loin.

- Des contradictions :

> "Une écriture est prise en compte, lorsqu'elle est valide, lors d'un traitement mensuel; elle est alors représentée par une valeur dans les résultats de gestion".

Un schéma indique, à la page suivante, que l'écriture, même si elle fait l'objet d'un traitement mensuel, n'apparaît pas pour autant nécessairement dans les résultats de gestion (elle peut, en effet, sous certaines conditions, rester dans une attente différée illimitée).

Outre ces éléments, on trouve également des bruits (éléments n'apportant rien à la spécification), des ambiguïtés et surtout des surspécifications (éléments ne décrivant pas le "Quoi" mais le "Comment").

Ces dernières sont, en effet, très nombreuses; en voici deux caractéristiques :

- "Chaque vacation ne comporte qu'un job de 2 ou 3 étapes. Aux deux fichiers précédents s'ajoutent un fichier d'archives et 3 fichiers de références seulement dont la base de données de GESTI".

- Ces surspécifications se retrouvent également dans un certain nombre de schémas présentés. Outre un modèle de données sous la forme d'un schéma binaire IMS, on trouve également une description des traitements sous la forme d'une chaîne de programmes incluant des contraintes inhérentes à l'utilisation du logiciel conversationnel TSO (cfr. figure 5 ci-dessous).

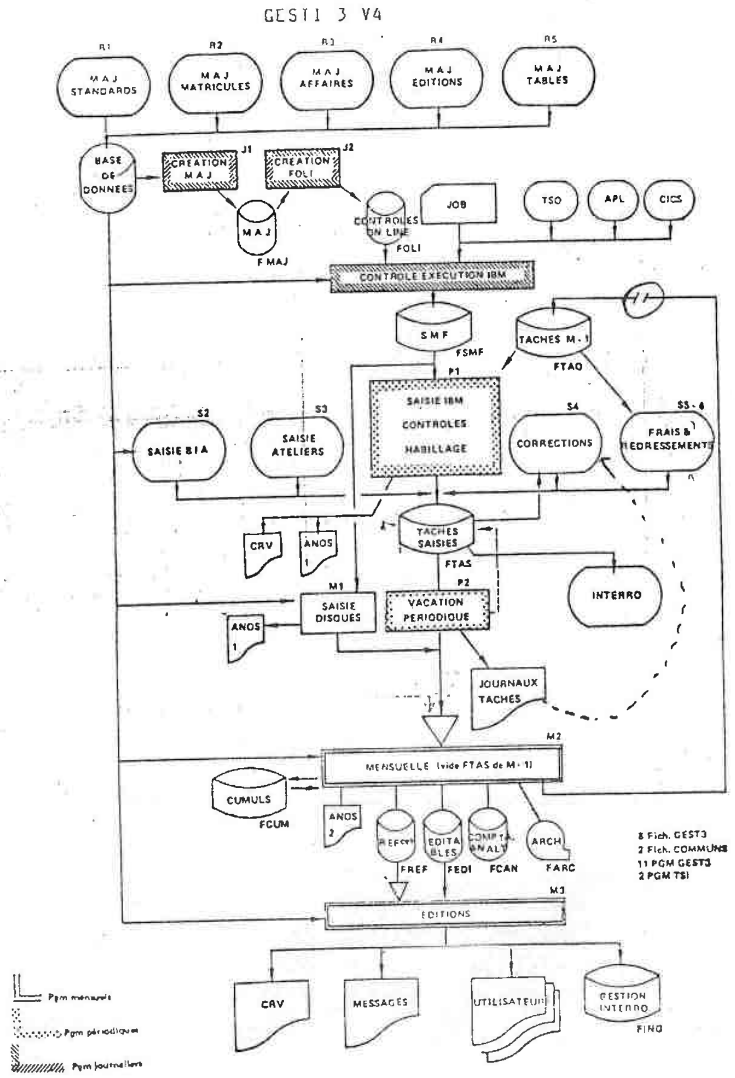


Figure 5

5.2.3. Contributions particulières inhérentes à l'approche suivie

Outre le fait d'avoir évité bon nombre des écueils cités ci-dessus, l'application de notre démarche à la spécification de GESTI a permis de mettre en évidence un certain nombre de points caractéristiques.

Le premier point à souligner est l'intérêt suscité par le caractère déductif de l'approche.

La spécification initiale rédigée par le chef de projet (informaticien) a abouti à une présentation inductive de la description (de la saisie des données à la restitution des résultats). Or, ce type de présentation n'a pas semblé rencontrer le point de vue du demandeur qui, lui, voyait davantage son problème sous la forme suivante : partir de la définition du résultat et remonter progressivement vers les arguments de départ.

Il va de soi que notre approche basée sur l'analyse des résultats a constitué dans ces conditions un excellent guide méthodologique favorisant une mise en évidence plus facile et plus claire de bon nombre d'éléments.

Le deuxième point à mentionner est lié au fait qu'en suivant notre démarche, nous avons été amenés à éliminer bon nombre de surspécifications introduites. De ce fait, la spécification obtenue fait apparaître beaucoup plus clairement la logique sous-jacente à l'application que dans le cahier des charges initial.

D'autre part, outre cette meilleure structuration des traitements, nous sommes également parvenus à une description et une structuration des données; celle-ci était, en effet, absente de la spécification initiale obtenue suivant une démarche fonctionnelle (redondance d'une donnée du fait de son apparition sur différents fichiers ou bordereaux).

Enfin, un troisième point, qui nous paraît le plus important, est relatif à l'expression de certains éléments présentant un caractère historique. En effet, le caractère comptable de l'application rend nécessaire le suivi d'une même dépense sur différentes périodes de temps :

- au niveau mensuel;
- au niveau annuel;
- au niveau pluri-annuel.

Les traitements étant différents à chaque niveau; la spécification initiale débouchait sur la description de trois applications quasi-indépendantes communiquant entre elles via différents fichiers (état mensuel, état annuel,...).

Une des difficultés réside dans les redressements (modifications ultérieures) pouvant être apportées aux différentes écritures; en effet, certaines modifications doivent avoir une répercussion sur des écritures passées au niveau mensuel, mais pas sur ces mêmes écritures passées au niveau annuel; d'autres amènent le contraire;...

Pour résoudre ce problème, il convient d'associer un historique de ces modifications à chacune des écritures; cet aspect historique s'est trouvé difficilement exprimé dans le système décrit, ce qui a été la source d'erreurs au niveau des résultats comptables produits par l'application une fois réalisée.

Pour faire face à ce problème, nous avons expérimenté que l'utilisation de la notion de "suite temporelle" et des primitives associées semble bien convenir en offrant des outils permettant d'exprimer plus simplement ce type de problèmes.

5.3. L'APPLICATION DU PLANNING DES FORAGES

5.3.1. Présentation du projet

Le but du système développé est la gestion de l'occupation des appareils de forage utilisés par le groupe. Cette gestion recouvre un double aspect :

- d'une part, elle doit permettre d'établir un tableau récapitulatif de l'occupation par pays et par appareil (à partir de là, divers rapport statistiques peuvent être établis);

- d'autre part, la gestion est également financière dans la mesure où, à tout moment, on doit pouvoir rendre compte des écarts monétaires, sur l'activité d'exploitation, correspondant à la part financée par le groupe.

Le système a été développé en deux phases successives : d'abord, le suivi de l'occupation des appareils, ensuite, l'aspect budgétaire associé à ce suivi. L'application a été programmée en APL, elle repose de plus sur une large utilisation d'outils graphiques de visualisation de résultats (consoles Tektronix) et utilise un système de base de données simplifié interrogeable à partir des programmes APL.

Il est à noter que l'utilisation du langage APL a permis de présenter, à différents stades de conception, des maquettes de fonctionnement du système; celles-ci ont permis d'apporter assez rapidement les corrections nécessaires au fur et à mesure des remarques des utilisateurs (surtout au niveau de la présentation des résultats).

5.3.2. Le cahier des charges de l'application du planning des forages

Dans ce système, l'aspect "données" étant assez important, il a été décidé d'adapter une démarche analogue à celle préconisée par Merise [TAR,83] conduisant à une structuration des données sous la forme d'un modèle entité-association (ou "individu-relation").

L'application de cette démarche a été jugée globalement satisfaisante dans la mesure où l'utilisaton de ce formalisme a permis d'améliorer le dialogue et la communication avec les utilisateurs et les demandeurs : il est à remarquer cependant, dans certains cas, la difficulté de choisir si une information est une entité, une association ou une propriété.

Au niveau des traitements, pour la phase 1, ceux-ci ayant été jugés assez simples, un cahier des charges n'a pas été élaboré et seul un manuel d'utilisation d'une quinzaine de pages a été rédigé.

Pour la phase 2, par contre, un cahier des charges a été rédigé de manière informelle; malgré un effort de complétude au niveau de sa rédaction, il est apparu au stade de la réalisation qu'il subsistait encore bon nombre d'éléments non spécifiés, voire même d'éléments décrits de manière contradictoire.

Globalement, nous pensons, que dans l'état actuel du système, le manque de spécifications peut constituer un inconvénient dans la mesure où l'application aurait à être prise en charge par un autre responsable (ce qui n'a pas été le cas jusqu'à présent).

En effet, le code APL, même s'il offre un caractère structuré et fonctionnel ainsi qu'un certain nombre de primitives de haut niveau, ne peut pas être considéré comme un véritable langage de spécification dans la mesure où, d'une part, il n'est facilement communicable, ni à des non-informaticiens, ni à des informaticiens; et, d'autre part, il renferme un certain nombre de caractéristiques techniques de réalisation (primitives graphiques et gestion de la base de données).

5.3.3. Contributions particulières inhérentes à l'approche suivie

Au niveau de la description des traitements, la sémantique de ceux-ci étant assez simple, il a été facile de les spécifier, en particulier, par l'utilisation de primitives telles que le tri, la somme, la recherche d'un maximum,...

Au niveau de la description des données, il s'est révélé être très instructif de confronter le résultat de notre spécification des données avec celui obtenu par application de l'approche Merise. Celle-ci est caractérisée par une description à priori du système d'information et est fondée sur l'utilisation du modèle entité-association (cf. présentation de ce modèle dans le chapitre 1).

Les enseignements suivants sont à noter :

- Les mécanismes de structuration de données utilisés dans le modèle entité-association ne sont pas suffisants. En particulier il apparaît que l'impossibilité d'exprimer un type comme étant une union de deux autres constitue une source d'ambiguïté dans le schéma conceptuel.

Exemple :

Dans les appareils de forage, on est amené à distinguer les véritables appareils de forage et d'autres qualifiés de "pseudo-appareils". Certaines propriétés sont communes aux deux types d'appareils; d'autres sont caractéristiques de chacun d'entre eux.

Dans le modèle décrit, les deux réalités se trouvent cachées derrière la même entité : seule d'adjonction d'un indicateur (surspécification) permet de marquer la distinction.

Un autre de nos constructeurs de type largement utilisé est le constructeur "Table"; celui-ci s'avère en effet facilement manipulable du point de vue des demandeurs pour exprimer certaines des structures.

La figure 6 indique la structuration des données de départ du problème :

- Le fait d'avoir établi le modèle à priori sans prendre en considération la description des traitements a conduit à la non-expression de certaines associations reflétant cependant certaines réalités exprimées par le demandeur dans ses besoins. Il est intéressant de comparer à ce sujet les figures 7 et 8 :

  . la figure 7 correspond au schéma conceptuel de la phase 1 établi en suivant l'approche Merise;

  . la figure 8 correspond au modèle entité-association obtenu en reprenant les mêmes entités et en prenant comme association les différentes relations apparues dans la spécification de l'énoncé (cf. figure 9) ainsi que les structures associées aux données nécessaires (cf. figure 6; par exemple, la structure de table).

- Un autre problème à mentionner est qu'il est difficile d'exprimer dans le formalisme entité-association des contraintes particulières au niveau d'un objet d'un type. En particulier, ce problème se pose lorsque les occurences ont des propriétés variables selon le"temps" (par exemple, l'historique des occurences d'un individu).

  Exemple :

  Il s'est avéré difficile d'exprimer dans le modèle entité-association qu'un forage particulier à un instant t soit un forage "réel", soit une "rétrocession", soit une "activité particulière".

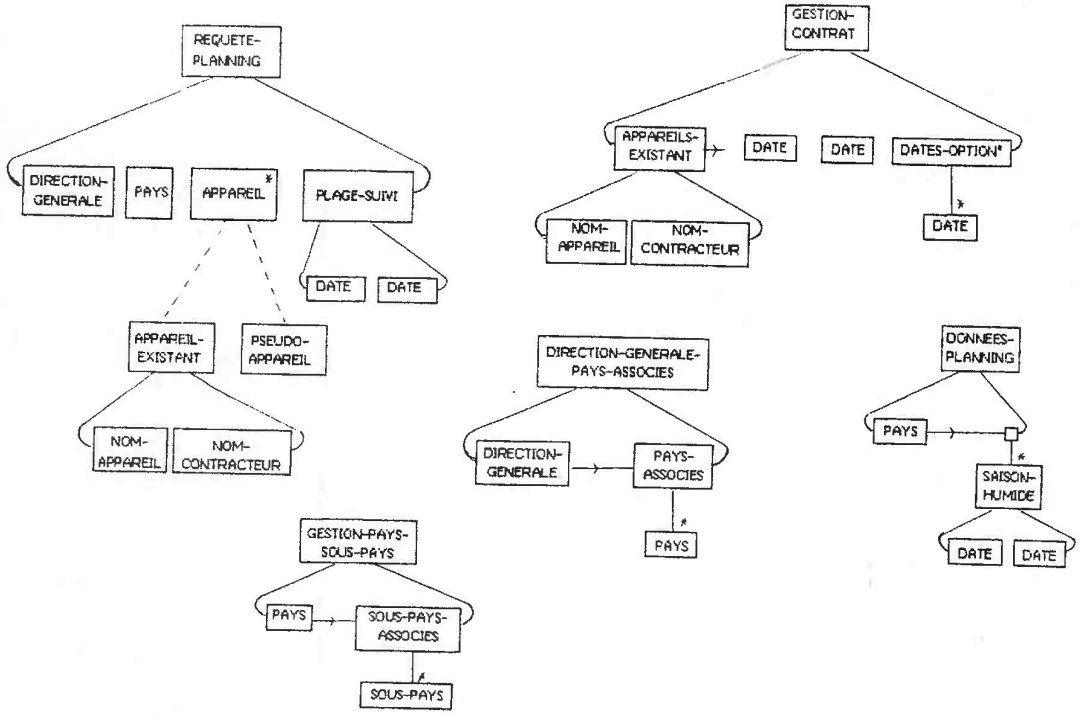


Figure 6: Structuration des données

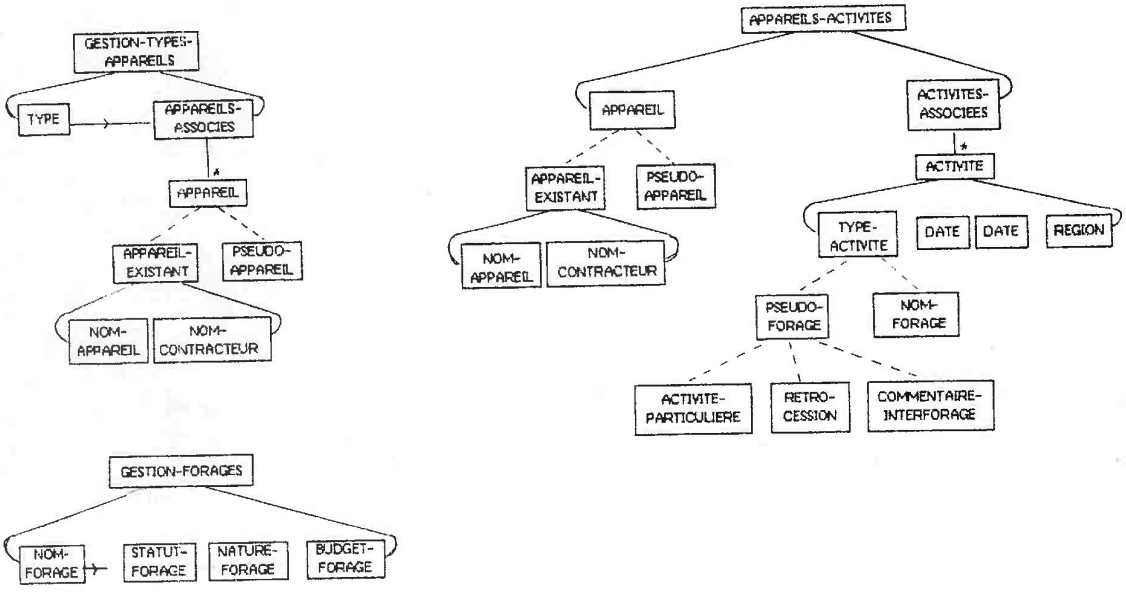


Figure 6 (suite)

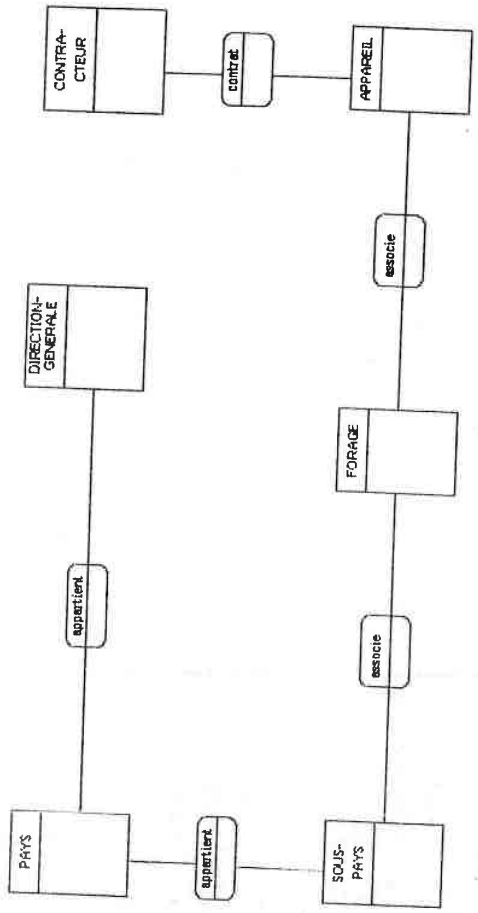


Figure 7: Schéma conceptuel obtenu suivant l'approche MERISE

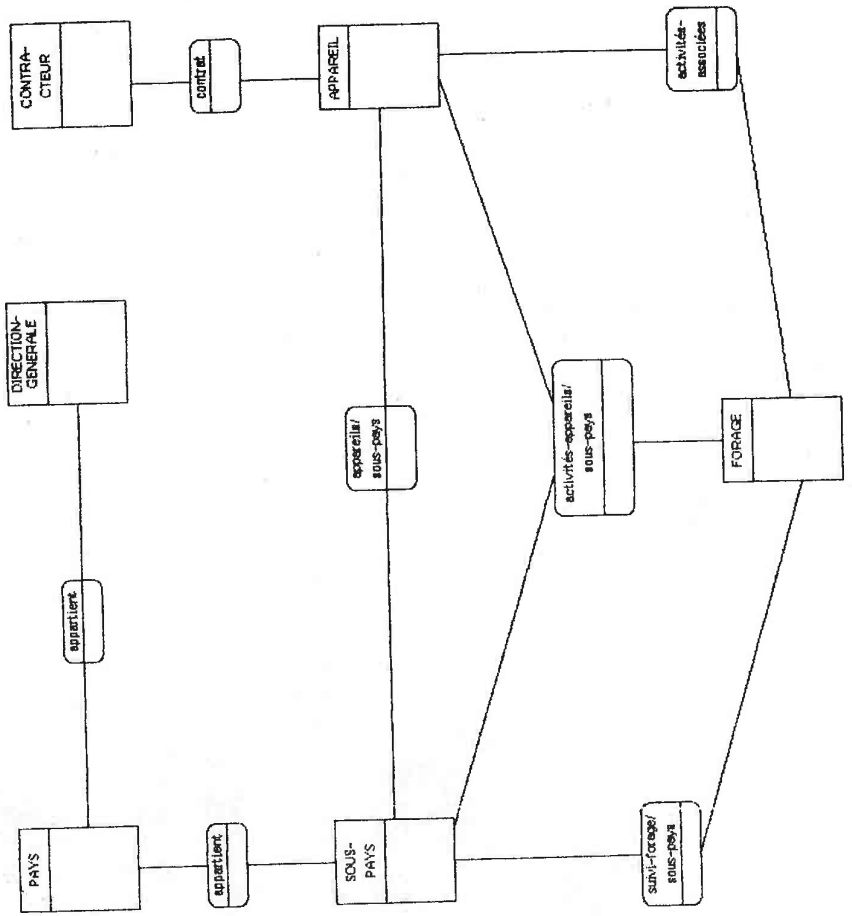


Figure 8: Modèle Entité-Association obtenu après analyse du

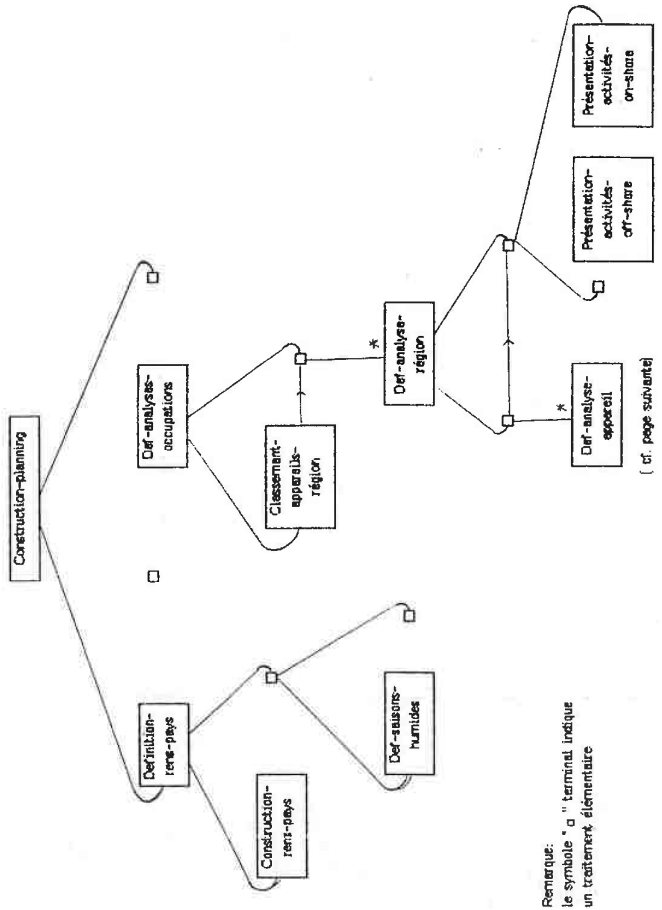


Figure 9: Arbre de traitements

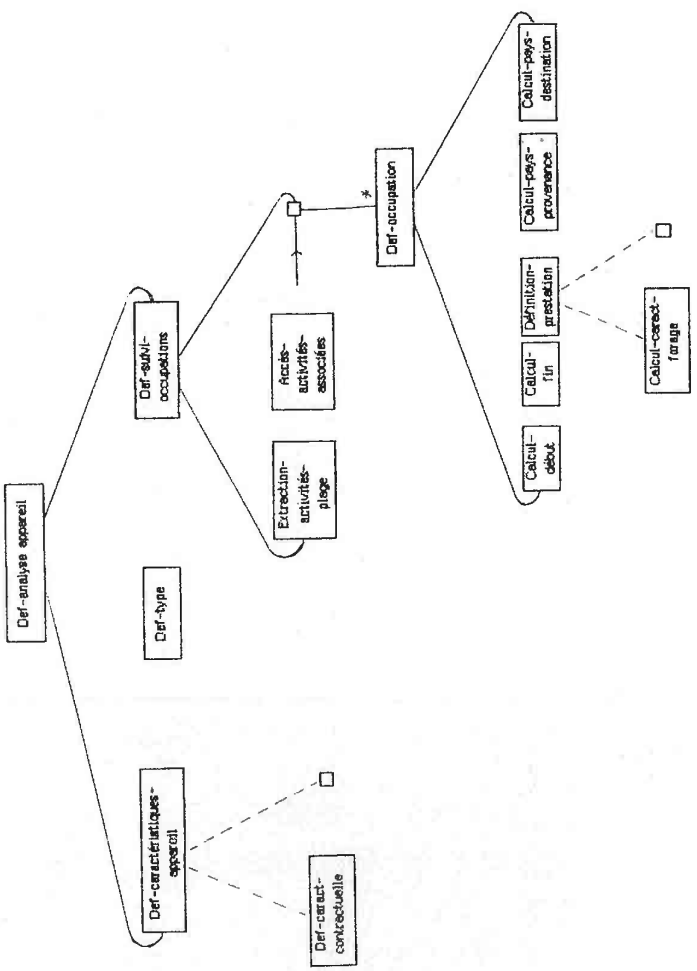


Figure 9: (suite)

Chapitre 6
PROLONGEMENTS POSSIBLES DU TRAVAIL

Il va de soi que le présent travail compte un certain nombre de limites; parmi les aspects qui nous semblent requérir une étude plus approfondie, nous avons retenu les points suivants :

- La mise en évidence et l'intégration des contraintes "organisationelles" dans la spécification initiale;

- Le contrôle et l'élimination de la redondance dans la spécification initiale;

- Le passage quasi systématique de la spécification à la résolution du problème;

- Le besoin d'un système d'outils automatiques d'aide à la spécification.

Il n'a pas été possible dans le cadre de ce travail d'étudier de manière approfondie chacun de ces points.

Cependant, il nous paraît important d'énoncer un certain nombre d'idées que nous avons concernant chacun des sujets et que nous souhaiterions développer ultérieurement.

6.1. L'ADAPTATION DE LA SPECIFICATION A L'ENVIRONNEMENT ORGANISATIONNEL

Comme nous l'avons indiqué dans le chapitre 2, nous décomposons le travail de spécification du système d'information en deux tâches :

- d'une part, une spécification structurée des besoins;
- d'autre part, la formulation d'un ou plusieurs scénarios possibles de fonctionnement du système devant être soumis aux utilisateurs et demandeurs.

Concernant la réalisation de la première tâche, notre approche est la suivante :

- définir un noyau du système; celui-ci incluant toutes les caractéristiques inhérentes à l'application traitée;
- greffer sur ce noyau un certain nombre de caractéristiques apparaissant comme dépendantes de l'environnement organisationnel spécifique à considérer;
- surimposer enfin certaines contraintes additionnelles relatives aux performances attendues, à la protection de certaines informations...

Notre travail s'est concentré sur la première étape; il nous est possible, cependant, de formuler quelques idées quant aux points devant être pris en compte dans les deuxième et troisième étapes.

La spécification initiale présente le noyau du système reprenant la logique de l'application à traiter; à côté de cela, il y a un certain nombre de contraintes propres à un environnement particulier qu'il faut en quelque sorte "greffer" sur le noyau logique.
Les points (2) et (3) de l'approche proposée concernent des contraintes classées en deux catérogries :

1° Les contraintes entraînant une modification du problème initial (les contraintes organisationnelles)

2° Les contraintes orientant la manière de calculer la solution du problème (les contraintes de réalisation).

6.1.1. Spécification des contraintes organisationnelles

La spécification de ces contraintes peut se subdiviser en trois catégories :

A. La spécification des traitements organisationnels manquants

B. La spécification des traitements d'interface

C. La spécification des interrelations organisationnelles entre traitements.

A. La spécification des traitements organisationnels manquants

La spécification initiale décrit le comportement du système attendu par le demandeur; ce comportement est décrit indépendamment d'une structure particulière d'organisation ou de règles et coutumes que l'on ne peut remettre en cause.

La prise en compte de tels éléments conduit la plupart du temps à enrichir la spécification initiale de nouvelles "relations" exprimant ces éléments particuliers.

Ainsi, par exemple, une procédure de redressement des stocks est spécifique à un mode d'organisation où une divergence peut exister entre le stock calculé automatiquement et le stock physiquement présent en magasin (vol, casse,...).

B. La spécification des traitements d'interface

Nous avons vu que chaque pavé de traitement décrit est caractérisé par un ensemble d'arguments et un ensemble de résultats.

Dans une organisation particulière, il est probable que les résultats et les arguments soient appréhendés sous une autre forme; dès lors, il s'agit de spécifier des pavés de traitement supplémentaires permettant de transformer les objets logiques définis en objets tels qu'il sont attendus par le demandeur.

De tels pavés d'"interface" doivent contenir, outre les règles d'acquisition et d'édition de données, des définitions de format des documents servant de support à ces données, des définitions de critères de validité pour ces données - permettant de spécifier les contrôles à effectuer sur celles à l'entrée -, ainsi que des définitions d'éventuels traitements additionnels spécifiques à leur appliquer.

C. Les contraintes d'interrelations organisationnelles entre traitements

Jusqu'ici, les interrelations spécifiées entre les différentes relations sont uniquement dictées par les dépendances entre données (cfr. langages data flow).

Il se peut cependant, que des relations supplémentaires d'enchaînement doivent être prises en compte à cause de certaines contraintes organisationnelles.

Ainsi, par exemple, on pourrait être amené à surimposer un ordre de séquencement entre Def-coord-client et Def-corps-fact (cfr. l'exemple du chapitre 4) du fait qu'une seule personne est chargée de ces deux tâches et doit nécessairement accomplir l'une avant l'autre.

La description formelle de cette contrainte revient dès lors à appliquer la règle du "et séquentiel" à la place de la règle du "et indéterministe".

6.1.2. Spécification des contraintes de réalisation

Nous distinguons deux types de contraintes :

A. Les contraintes liées aux processeurs utilisés

B. Les contraintes de performance

A. Les contraintes liées aux processeurs utilisés

En vue de délimiter la tâche de l'informaticien vis-à-vis du système à développer, il faut encore définir les différents processeurs associés aux traitements identifiés.

Pour chaque relation figurant dans la spécification du noyau, pour chaque greffe organisationnelle et pour chaque pavé d'interface, il s'agit de mentionner le ou les processeurs nécessaires à la réalisation des relations de base : ordinateur, terminal, une ou plusieurs personnes d'un service déterminé.

D'autre part, tout processeur doit être décrit par un ensemble de caractéristiques telles que : coût, capacité de traitement, calendrier d'utilisation, l'unité de mesure de son activié,...

B. Les contraintes de performances

Outre la définition du noyau du système d'information et des particularités organisationnelles venant s'y greffer, il faut encore inclure dans la spécification un certain nombre de contraintes que le système devra satisfaire et qui sont relatives aux performances attendues : temps de réponse en circonstances normales, sécurité face à certains incidents, confidentialité de certaines informations...

Satisfaire ces nouvelles contraintes peut entraîner l'introduction de nouveaux pavés de traitement destinés à améliorer les performances du système décrit.

Ainsi, par exemple, une procédure d'ordonnancement de commandes livrables est spécifique d'une certaine performance attendue par le système.

D'autre part, afin de pouvoir préjuger des performances du système, il est intéressant de connaître, en plus des caractéristiques des processeurs utilisés, des informations qualitatives quant aux données traitées : taux d'arrivée des transactions, volumes des structures de données (volumes minimum, maximum et moyen), taux d'occupation d'une ressource,...

Enfin, il existe une autre catégorie de contraintes : celles relatives aux possiblités de parallélisme. Ainsi, dans le cas d'un système multiutilisateurs en temps réel, on peut être amené par exemple à spécifier des clauses d'interférence entre traitements, puis, à partir de là, identifier le parallélisme potentiel entre traitements n'interférant pas; de plus, on pourrait être amené à préciser des règles d'ordonnancement en cas de conflit entre traitements interférents. Nous pensons que, dans bon nombre de cas, l'expression de ces contraintes pourra se faire en exprimant de nouvelles relations (à l'aide des primitives associées) sur les différentes suites "temporelles" introduites dans la spécification initiale.

6.2. LE CONTROLE DE LA REDONDANCE

La spécification obtenue se présente sous la forme d'un certain nombre d'arbres :

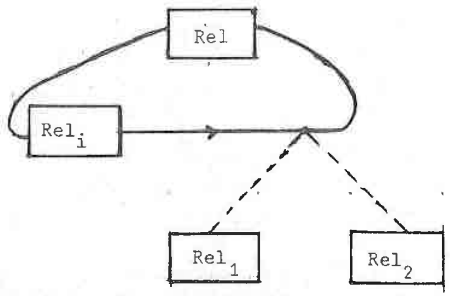
- un arbre des fonctions : un noeud de cet arbre correspondant à une relations définissant un sous-problème qui, à son tour, peut être spécifié par un noeud de niveau inférieur, chaque relation manipule un certain nombre de données (résultats ou arguments);

- plusieurs arbres de types : un arbre décrivant la structure du type du résultat initial, les autres décrivant les structures des types des arguments et éventuelles données intermédiaires introduites.

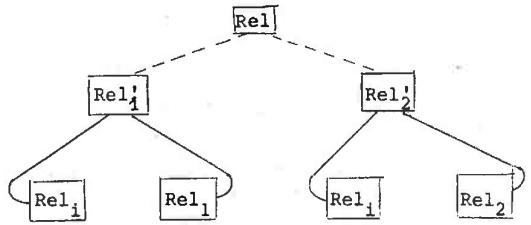
Au niveau de l'arbre des fonctions, la redondance des relations peut se manifester sous deux formes :

A. L'application du méta-algorithme conduit lors du raffinement d'une relation Rel en deux nouvelles relations Rel_1,Rel_2 à identifier et à mettre en évidence des relations intermédiaires (Rel_i) communes à Rel_1,Rel_2 (avec application de la règle du "et séquentiel").

Graphiquement, dans le cas où la règle de raffinement appliquée est celle du "ou", l'arbre se présente comme suit :



Il va de soi que, a priori, le demandeur n'est pas toujours capable d'exprimer la mise en évidence de Rel_i; graphiquement l'arbre ci-dessus se présenterait comme suit :



Les causes de cette redondance peuvent être multiples; citons, par exemple :

- soit le demandeur ne possède pas une vision suffisamment globale pour identifier à priori la relation;

- soit, le demandeur est amené à répéter volontairement cette relation pour des raisons organisationnelles; ainsi, il n'est pas rare de voir dans certaines organisations des traitements "dédoublés" du fait de rivalités entre différents services.

Pour éliminer cette redondance, il convient de modifier la structure de l'arbre des relations afin de faire remonter les relations dupliquées.

B. Dans d'autres cas, l'expression d'une même relation peut prendre des formes distinctes à différents endroits de la spécification.

Ainsi, dans certains cas de traitements de suite, l'expression d'une même relation peut se faire par différentes compositions de primitives associées aux suites.

Dans ce cas, l'idée de "rapprocher" deux relations sera guidée en général par la constatation qu'elles font intervenir les mêmes types de données, que leurs descriptions informelles semblent identiques, il restera bien-sûr à prouver l'équivalence des deux spécifications formelles.

Au niveau des types de données, les inconvénients majeurs d'une approche fonctionnelle (c'est-à-dire guidée par la mise en évidence des fonctionnalités du système) par rapport à une approche "base de donnée" (dans laquelle on fige a priori une description des types de données) sont traditionnellement les suivants :

- d'une part, une non-structuration des types

- d'autre part, une redondance possible.

Au niveau de la non-structuration des types, notre approche pallie cet inconvénient en utilisant de véritables outils de structuration (produit cartésien, suite, table, ...); par contre la redondance entre types est possible et par conséquent, doit être contrôlée.

Nous pensons que l'utilisation du cadre offert par les types abstraits (le type et ses opérations associées) est utile pour résoudre ce problème.

Nous distinguons deux types de redondances (les notations employées ont été dans le chapitre lors de la présentation du concept de type abstrait) :

a) les types équivalents : deux types définis par $\langle S,\Sigma,E\rangle$ et $\langle S,\Sigma,E'\rangle$ sont dits équivalents :

si et seulement si

1. $S = S'$
2. $\Sigma = \Sigma'$
3. Les axiomes de E sont des formules valides dans l'algèbre définie par les axiomes de E' et réciproquement

Cette définition de la redondance est très forte puisqu'elle fait apparaître des types apparemment distincts qui, en fait, sont manipulés de la même manière.

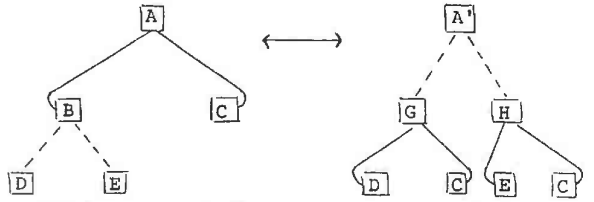
b) l'égalité entre types :

Dans d'autres cas, la redondance ne viendra pas du fait que des types apparemment distincts sont manipulés de la même manière, mais plutôt par le fait que leurs stuctures - c'est-à-dire, une composition de constructeurs - sont identiques.

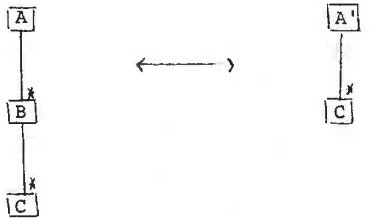
Eliminer ces redondances revient alors à appliquer un certain nombre de transformations sur la structure d'un type de manière à la rapprocher de la structure d'un autre type.

Mentionnons quelques exemples de transformations que nous avons pu mettre en évidence :

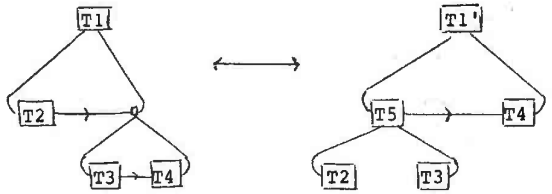
1° A = [B = UNION [D,E] ,C] A' = UNION [G = PC[D,C], H = PC[E,C]]



2° A = SUITE [B = SUITE[C]] A' = SUITE [C]



3° T1 = TABLE [T2 → TABLE [T3 → T4]] T1' = TABLE [T5 = PC [T2,T3] → T4]



Mentionnons finalement que les différents types de redondances répertoriés ci-dessus interagissent; ainsi, la fusion de deux types équivalents conduisent vraisemblablement à devoir éliminer des redondances au niveau de l'arbre des relations.

Le lecteur intéressé pourra trouver dans [LEY,84] un approfondissement de ces idées concernant les transformations de types.

6.3. DE LA SPECIFICATION A LA RESOLUTION DU PROBLEME

Comme [BAU,82] nous pensons que le processus de construction d'un programme se compose de trois étapes :

- la spécification du problème;
- la construction d'un programme rédigé en code machine à partir de la spécification et en suivant un certain nombre d'étapes (analyse de programmation, programmation et codage);
- la certification du produit réalisé; c'est-à-dire, son test et sa maintenance.

Au niveau de la construction du programme, nous adaptons le point de vue de [FIN,79] :

1) Transformation de la spécification initiale du problème en une seconde spécification plus explicite dans le sens où elle permet de calculer des solutions.

2) Dérivation, à partir de cette seconde spécification encore relativement inefficace, du programme final.

Nous allons présenter un exemple de transformation de la spécification initiale vers une solution implémentable. Cette transformation concerne le passage d'un type décrit par ses opérations dans la spécification initiale en un type plus "dynamique" se rapprochant davantage de la représentation finale; par la même occasion, nous montrerons en quoi l'utilisation du cadre formel constitué par la types abstraits permet la justification du passage systématique.

L'exemple concerne le type DOC-STOCK relatif à la description de la gestion des stocks de produit; sa spécification initiale est la suivante :

TYPES DOC-STOCK : Documentation sur les stocks de produits NO-PROD : Numéro du produit QTE-PROD : Quantité du produit OPERATIONS Creat-stock-prod : Création d'un stock initial de produit Modif-stock-prod : Modification d'un stock de produit suivant une quantité entrée ou sortie du produit Sup-stock-prod : Suppression du stock d'un produit Qte-stock-prod : Accès à la quantité en stock d'un produit Exist-stock-prod : Prédicat d'existence d'un stock du produit	TYPES stock,stock' : DOC-STOCK qté : QTE-PROD no : NO-PROD b : BOOLEEN OPERATIONS Créat-stock-prod : DOC-STOCK ← DOC-STOCK,NO-PROD,QTE-PROD Modif-stock-prod : DOC-STOCK ← DOC-STOCK,NO-PROD,QTE-PROD Sup-stock-prod : DOC-STOCK ← DOC-STOCK,NO-PROD Qte-stock-prod : QTE-PROD ← DOC-STOCK,NO-PROD Sup-stock-prod : DOC-STOCK ← DOC-STOCK,NO-PROD	DOC-STOCK = TABLE [NO-PROD → QTE-PROD] OPERATIONS Création et mise-à-jour Creat-stock-prod(stock, no,qte)stock' tq Créer(stock,no,qte) Modif-stock-prod(stock no,qte)stock' tq Mod(stock,no,Accès (stock,no)+qte) Sup-stock-prod(stock,no) stock' tq Sup(stock,no) CONSULTATION Qte-stock-prod(stock,no) qte tq Accès(stock,no) Exist-stock-prod(stock, no)b tq Exist(stock,no)

Connaissant les axiomes caractérisant la table, nous pouvons donner une spécification algébrique du type DOC-STOCK (Remarque : nous ajoutons l'opération d'initialisation "Doc-stock-vide") :

TYPE DOC-STOCK

OPERATIONS

CONSTRUCT	Doc-stock-vide	→ DOC-STOCK
	Creat-stock-prod : DOC-STOCK,NO-PROD,QTE-PROD	→ DOC-STOCK
MODIF	Modif-stock-prod : DOC-STOCK,NO-PROD,QTE-PROD	→ DOC-STOCK
	Sup-stock-prod : DOC-STOCK,NO-PROD	→ DOC-STOCK
EXTERN	Qte-stock-prod : DOC-STOCK,NO-PROD	→ QTE-PROD
	Exist-stock-prod : DOC-STOCK,NO-PROD	→ BOOL

DECLARATIONS

b : BOOL
stock : DOC-STOCK
no,no' : NO-PROD
qte,qté' : QTE-PROD

EQUATIONS

DEF-MODIF

Modif-stock-prod(Doc-stock-vide,no',qté') = = Doc-stock-vide
Modif-stock-prod(Creat-stock-prod(stock,no,qte)no',qte') = = Creat-stock-prod(stock,no,qte') si no = no'
Modif-stock-prod(Creat-stock-prod(stock,no,qte),no'qte') = = Creat-stock-prod(Modif-stock-prod(stock,no',qte')no,qte) si no ≠ no'
Sup-stock-prod(Doc-stock-vide,no') = = Doc-stock-vide
Sup-stock-prod(Creat-stock-prod(stock,no,qte),no') = = stock si no = no'
Sup-stock-prod(Creat-stock-prod(stock,no,qte),no') = = Creat-stock-prod (Sup-stock-prod(stock,no'),no,qte)

Qte-stock-prod(Doc-stock-vide,no') == ω
Qte-stock-prod(Creat-stock-prod(stock,no,qte),no') == qte si no = no'
Qte-stock-prod(Creat-stock-prod(stock,no,qte),no') == Qte-stock-prod (stock,no') si no ≠ no'
Exist-stock-prod(Doc-stock-vide,no') == Faux
Exist-stock-prod(Creat-stock-prod(stock,no,qte),no') == Vrai si no = no'
Exist-stock-prod(Creat-stock-prod(stock,no,qte),no') == Exist(stock-no') si no ≠ no'

Remarque : Pour l'opération Modif-stock-prod, l'argument qte est en fait le résultat de

(Accès(stock,no') + qte')

L'objet de type algébrique DOC-STOCK étant assez éloigné des objets habituellement manipulés dans une étape de résolution du problème, il est donc nécessaire d'en définir une représentation en termes d'objets plus proches de la représentation finale.

En l'occurrence, nous allons utiliser les types PILE et TABLEAU que nous définissons de la manière suivante : (voir page 6/17).

TABLEAU [VAL = PC [NO-PROD, QTE-PROD]]

OPERATIONS
CONSTRUCT
Tabvide: TABLEAU %création tableau vide%
Assigne: TABLEAU,ENTIER,VAL TABLEAU
%assignation dans le tableau de la valeur VAL à l'indice ENTIER%
EXTERN
Accès: TABLEAU,ENTIER VAL %accès à la valeur VAL située à l'indice ENTIER dans le tableau%
Position:TABLEAU,NO-PROD ENTIER %indice de la position de la valeur NO-PROD dans le tableau%
Exist: TABLEAU,NO-PROD BOOL %prédicat d'existence de la valeur NO-PROD dans le tableau%

DECLARATIONS
b: BOOL i,j: ENTIER no,no':NO-PROD
t: TABLEAU qté: QTE-PROD

EQUATIONS
DEF-EXTERN
Accès(Tabvide,i) == ω
Accès (Assigne(t,j,(no,qte)),i) == (no,qte) si i = j
Accès (Assigne(t,j,(no,qte)),i) == Accès(t,i) si i ≠ j
Position (Tabvide,no) == ω
Position(Assigne(t,j,(no',qte)),no) == j si no = no'
Position(Assigne(t,j,(no',qte)),no) == Position(t,no) si no ≠ no'
Exist (Tabvide,no) == Faux
Exist (Assigne(t,j,(no',qte)),no) == Vrai si no = no'
Exist (Assigne(t,j,(no',qte)),no) == Exist (t,no) si no ≠ no'

```
PILE [ENTIER]

   OPERATIONS
      CONSTRUCT
           Pvide:      PILE   %pile vide%
           Empiler: PILE,ENTIER    PILE   %adjonction d'un entier
                                  au sommet de la pile%
      MODIF
           Depiler: PILE    PILE %retrait de l'entier au
                                  sommet de la pile%
      EXTERN
           Sommet: PILE   ENTIER  %accès à l'entier au sommet
                                   de la pile%
           Vide: PILE   BOOL  %prédicat indiquant si la pile
                               est vide%
   DECLARATIONS
        p: PILE
        i: ENTIER
   EQUATIONS
      DEF-MODIF
           Depiler (Pvide) == Pvide
           Depiler (Empiler(p,i)) == p
      DEF-EXTERN
           Sommet (Pvide) == ω
           Sommet (Empiler(p,i)) == i
           Vide (Pvide)  == Vrai
           Vide (Empiler (p,i)) == Faux
```


On choisit alors comme type représentant TABLEAU X PILE X ENTIER, le tableau contiendra les quantités de chaque produit en stock, ENTIER représentant la longueur du tableau, PILE contient les indices de "trous" dans le tableau, c'est-à-dire des produits pour lesquels le stock est supprimé.

Nous allons maintenant définir la fonction p de représentation du type DOC-STOCK.

DOC-STOCK[NO-PROD,QTE-PROD] → TABLEAU[VAL = PC[NO-PROD, QTE-PROD] X PILE[ENTIER] X ENTIER]

Représentation des constructeurs primaires :

p(Doc-stock)vide) = «Tabvide,Pvide,0»
p(Creat-stock-prod(stock,no,qte)) = si Vide(p)
alors «Assigne(t,ent + 1,(no,qte)),p,ent + 1»
sinon «Assigne(t,Depiler(p),(no,qte)),Depiler(p),ent»

Représentation des autres opérations :

p(Modif-stock-prod(stock,no,qte)) =
«Assigne(t,Position(t,no),qte),p,ent»
p(Sup-stock-prod(stock,no)) = «t,Empiler(p,Position(t,no)),ent»
p(Qte-stock-prod(stock,no)) = Qte-prod(Accès(t,Position (t,no)))
p(Exist-stock-prod(stock,no)) = Exist(t,no)

Il reste à démontrer que la représentation obtenue est valide; pour cela, il importe de montrer que les types abstraits associés à la solution présentent bien le même "comportement" vis-à-vis de types externes que le type abstrait de départ (notion de représentation faible [FIN,79]).

Pour cela, il s'agit de démontrer que :

Si $t_1 = t_2$ est une instance d'un axiome de DOC-STOCK et concerne les fonctions d'observation (opérations externes)

alors $t_1' = t_2'$ (tout terme t de Doc-Stock se traduisant par un terme t' de TABLEAU X PILE X ENTIER) est un théorème de TABLEAU X PILE X ENTIER

1. ρ(Qte-stock-prod(Doc-stock-vide,no')) = = ω
   ρ(Qte-stock-prod(Doc-stock-vide,no'))
   = Qte-prod(Accès(Tabvide,Position(Tabvide,no')))
   = Qte-prod(Accès(Tabvide,ω))
   = Qte-prod(ω)
   = ω

2. ρ(Qte-stock-prod(Creat-stock-prod(stock,no,qte),no')= qte
   si no = no'
   ρ(Qte-stock-prod(Creat-stock-prod(stock,no,qte),no')
   = Qte-prod(Accès(ρ(Creat-stock-prod(stock,no,qte)),
   Position(ρ(Creat-stock-prod(stock,no,qte)),no'))

   a) si Vide(p)
      = Qte-prod(Accès(Assigne(t,ent+1,(no,qte)),Position
      (Assigne(t,ent+1,(no,qte)),no'))' [[no = no']]
      = Qte-prod(Accès(Assigne(t,ent+1,(no,qte)),ent+1)
      = Qte-prod((no,qte))
      = qte

   b) si non Vide(p)
      = Qte-prod(Accès(Assigne(t,Depiler(p),(no,qte)),
      Position(Assigne(t,Depiler(p),(no,qte)),no'))[no = no']]
      = Qte-prod(Accès(Assigne(t,Depiler(p),(no,qte)),
      Depiler(p))
      =Qte-prod((no,qte))
      = qte

3. ρ(Qte-stock-prod(Creat-stock-prod(stock,no,qte),no)) = =
   ρ(Qte-stock-prod(stock,no')) si no' ≠ no
   ρ(Qte-stock-prod(Creat-stock-prod(stock,no,qte))
   = Qte-prod(Accès(ρ(Creat-stock-prod(stock,no,qte),
   Position(ρ(Creat-stock-prod(stock,no,qte)),no'))

   a) si Vide(p)
      = Qte-prod(Accès(Assigne(t,ent+1,(no,qte)),Position
      (Assigne(t,ent+1,(no,qte)),no')[no ≠ no']]
      = Qte-prod(Accès(Assigne(t,ent+1,(no,qte)),Position
      (t,no'))
      = Qte-prod(Accès(t,Position(t,no')))
      = ρ(Qte-stock-prod(stock,no'))

   b) si non Vide(p)
      = Qte-prod(Accès(Assigne(t,Depiler(p),(no,qte)),
      Position(Assigne(t,Depiler(p),(no,qte)),no)[no ≠ no']]
      = Qte-prod(Acces(Assigne(t,Depiler(p),(no,qte)),Position
      (t,no'))
      = Qte-prod(Acces(t,Position(t,no')))
      = ρ(Qte-stock-prod(stock,no'))

4. p(Exist-stock-prod(Doc-stock-vide,no')) = = Faux
   p(Exist-stock-prod(Doc-stock-vide,no')
   = Exist(Tabvide,no')
   = Faux

5. p(Exist-stock-prod(Creat-stock-prod(stock,no,qte)) = =
   Vrai si no = no'
   p(Exist-stock-prod(Creat-stock-prod(stock,no,qte),no'))
   = Exist(p(Creat-stock-prod(stock,no,qte),no'))

   a) si Vide(p)

      = Exist(Assigne(t,ent+1,(no,qte)),no')[no = no']
      = Vrai

   b) si non Vide(p)

      = Exist(Assigne(t,Depiler(p),(no,qte)),no')[no = no']
      = Vrai

6. p(Exist-stock-prod(Creat-stock-prod(stock,no,qte),no')) = =
   p(Exist-stock-prod(stock,no')) si no ≠ no'
   p(Exist-stock-prod(Creat-stock-prod(stock,no,qte),no")))
   = Exist(p(Creat-stock-prod(stock,no,qte),no')

   a) si Vide(p)

      = Exist(Assigne(t,ent+1,(no,qte))no')[no ≠ no']
      = Exist(t,no')
      = p(Exist-stock-prod(stock,no'))

   b) si non Vide(p)

      = Exist(Assigne(t,Depiler(p),(no,qte)),no')[no ≠ no']
      = Exist(t,no')
      = p(Exist-stock-prod(stock,no'))

Le processus de dérivation vers une solution finale consiste alors à représenter ces nouveaux types en termes d'autres types plus proches de ceux utilisables dans un langage de programmation.

Ce passage a été étudié, dans le cadre des types abstraits, par [WUL,75] [LIS,75] [FIN,78] .

La construction de la solution à partir du problème se présente comme le résultat de la composition d'un certain nombre de transformations appliquées à la spécification initiale.

$$\text{Solution} = \text{Trans}_n(\text{Trans}_{n-1}(\ldots\text{Trans}_1(\text{spécification initiale})\ldots))$$

Notons cependant que dans le cadre de systèmes d'information, ce passage n'est pas toujours systématique.

Ainsi, des retours en arrière sont parfois nécessaires lorsque, à un niveau i de transformation, apparaissent des contraintes spécifiques ne permettant pas de valider le passage de la description de niveau i - 1 à celle de niveau i. [SWA,82] .

Ainsi, dans un problème de tenue de stock en temps réel, arrivé à un niveau i de transformation dans lequel on introduit le concept de fichier "physique", on constate que la répercussion d'une mise à jour de produit prend un certain temps et qu'une file d'attente de mises à jour peut se former.

La conséquence en est que lors de l'interrogation du fichier, on est plus certain d'accèder au véritable état des stocks.

Deux solutions sont alors envisageables :

1° soit, on ne peut tolérer ce risque d'incertitude; la description d'une solution doit alors tenir compte d'un accès possible à la file d'attente en plus d'un accès au fichier;

2° soit, on tolère cette incertitude; cela implique un retour en arrière avec une modification de la spécification initiale (simplification de la notion d'accès à l'état des stocks en "temps réel").

6.4. CAHIER DES CHARGES D'UN SYSTEME D'AIDE A LA SPECIFICATION

Dans ce travail, nous nous sommes consacrés uniquement aux aspects méthode et langage. Cependant, nous sommes convaincus que la meilleure démarche de spécification ne peut être utilisée de manière vraiment effective que si elle est supportée par un ensemble d'outils automatiques d'aide. C'est pourquoi, nous décrivons succinctement ci-dessous quelques fonctionnalités essentielles d'un système d'aide à la spécification.

Nous pensons que l'environnement intégré d'outils d'aide à la spécification ne doit pas se présenter comme un ensemble complexe d'outils fortement interreliés mais plutôt comme un ensemble d'outils autonomes exploitant une base de données commune de spécifications. La base des spécifications se présente comme une base de données devant être créée, maintenue, analysée et interrogée. La structure de donnée sous-jacente à cette base est l'arbre. Cet arbre représente la structure syntaxique de la spécification : il s'agit de sa syntaxe abstraite.

L'arbre abstrait comprend :

- d'une part, un sous-arbre décrivant la partie "énoncé" de la spécification, c'est-à-dire, les différentes relations introduites ainsi que leur structuration;

- d'autre part, un sous-arbre décrivant la partie "univers" de la spécification, c'est-à-dire, les différents types manipulés ainsi que leur structuration.

Associées à la manipulation de cet arbre abstrait, un certain nombre de fonctions sont nécessaires :

- créer l'arbre

- se déplacer dans l'arbre

- modifier l'arbre

Les outils venant se greffer sur cette base de spécifications peuvent se classer en deux catégories :

- les outils d'introduction de la spécification

- les outils d'exploitation de la spécification

Un premier outil nécessaire à l'introduction de la spécification est un éditeur syntaxique [DON,80] [MED,81] .
Cet éditeur conversationnel permet de rentrer progressivement les spécifications tout en faisant un certain nombre de contrôles de cohérence et de complétude au niveau des types et des énoncés au fur et à mesure de leur entrée.

Couplé à cet éditeur de spécification, il est important d'associer un outil guidant le spécifieur dans la construction systématique de l'arbre des relations et des types. Nous appelons cet outil le pilote de la spécification; son fonctionnement est étroitement lié au méta-algorithme que nous proposons en matière de systèmes d'information.

Concernant les outils d'exploitation de la spécification, il est important de définir des outils de visualisation associés aux modèles de spécification proposés (cf. les différentes figures dans le chapitre 2.2.), c'est-à-dire :

- la représentation graphique de l'arbre des énoncés et des types

- la visualisation des pavés "trois colonnes" associés à la description de chaque relation et type introduit.

Remarquons que ces outils de visualisation sont appelés à être largement utilisés par le spécifieur lors de l'introduction de la spécification avec le pilote.

D'autres outils plus complexes peuvent également venir se greffer sur le noyau de ceux déjà définis; citons :

- des outils de production de documents particuliers répondant à des requêtes exprimées dans un langage d'interrogation spécifique;

- des outils de maquettage reposant sur une évaluation symbolique de la spécification formelle;

- des outils de traduction en langue naturelle; l'utilisation de ceux-ci permettant de générer un cahier des charge rédigé en langue naturelle à partir de la spécification formelle et des commentaires introduits;

- des outils de transformations; ceux-ci permettant la construction de programmes à partir de la spécificaiton formelle;

- des outils liés à la réutilisation de connaissances propres au domaine traité (cet outil étant utilisé conjointement avec le pilote).

o o o

CONCLUSION

C O N C L U S I O N

L'évolution de la gestion des organisations vers une complexité croissante a rendu nécessaire la conception et la réalisation d'outils permettant d'aider les gestionnaires dans leur tâche d'analyse et de décision.

Parmi ceux-ci, les systèmes d'information, et plus particulièrement, ceux qui sont automatisés, sont devenus indispensables à la gestion d'une organisation.

Malheureusement, force est de constater que bon nombre d'échecs dans la mise en oeuvre de tels systèmes sont dus au fait qu'il existe peu de démarches permettant de décrire et de développer ces systèmes d'information de manière effective.

La problématique de la spécification des systèmes d'information

Une étape cruciale concerne la description des besoins exprimés par les demandeurs : il s'agit de l'étape de spécification.

Il est indispensable de disposer d'une "bonne" spécification, car celle-ci :

- constitue un document qui est à la base du contrat liant le demandeur aux réalisateurs;
- réduit le coût total de développement du logiciel; en effet, le coût des erreurs de spécification représente en moyenne un tiers du coût total du logiciel (270 milliards de FF en 1975) [BOE,76] .

- permet par la suite d'établir la correction du produit réalisé par rapport aux besoins exprimés;

- constitue la documentation destinée à la maintenance ultérieure du produit réalisé.

Depuis une dizaine d'années, un certain nombre de démarches d'aide à la spécification ont vu le jour; parmi les plus représentatives actuellement, citons :

- en ce qui concerne la modélisation des données, l'approche MERISE [TAR,83] fondée sur l'utilisation d'un modèle analogue au modèle entité-association [BEN,76] [CHE,76];

- en ce qui concerne la modélisation de la statique des traitements; les approches SADT [ROS,76] et SA [MAR,79] basées toutes deux sur l'utilisation des diagrammes de flux ("Data Flow Diagrams" [WEI,79]).

De telles démarches peuvent se comparer d'après :

- le type de modélisation du domaine d'application considéré;

- la compétence requise de la part des gens qui spécifient;

- les environnements offerts : supports graphiques, contrôles de certaines cohérences et complétudes, production de dictionnaires de données, de rapports récapitulatifs.

.....

Cependant, nous pensons que ce type de démarche d'aide à la spécification des systèmes d'information souffre généralement de deux inconvénients :

- d'une part, elles ne guident **pas véritablement le spécifieur** dans la construction et la structuration des spécifications dans le cadre de la description de systèmes d'information complexes;

- d'autre part, elles manquent généralement de bases théoriques solides.

A l'opposé, des approches plus générales de la spécification telles que celles proposées dans Zaide [ABR,78] [MEY,78] [DEM,80] Cip [BAU,78] [BAU,79] ou Spes [FIN,79] [FIN,83] ne souffrent pas des inconvénients mentionnés ci-dessus; cependant, elles n'ont été, le plus généralement, testées que sur de petits problèmes ou sur des problèmes à caractère informatique (par exemple, la gestion d'un système de fichier [FEA,80] ou la description d'un éditeur de texte [SUF,82] . Nous pensons que leur application telle quelle dans le domaine des systèmes d'information est susceptible d'engendrer les problèmes suivants :

- la nécessité de former de véritables "spécifieurs" capables de maîtriser la technique proposée;

- une spécification trop formelle est généralement difficilement communicable aux utilisateurs (remarquons, cependant, que l'utilisation de la technique de maquettage permet de pallier ce problème [MIT,82]);

- une spécification formelle est beaucoup moins concise que la description informelle équivalente [BAL,77] .

Le but de notre travail est de développer davantage les idées présentées dans le système SPES, compte tenu du domaine d'application considéré; plus précisément :

- de proposer une méthode guidant le spécifieur dans la construction des besoins exprimés par le demandeur;

- de développer un langage de spécification formel davantage adapté à la description des applications.

Notons, en outre, que l'approche proposée a été expérimentée dans un environnement industriel (Société Nationale Elf Aquitaine) où nous avons été amené successivement à spécifier un système comptable du suivi des dépenses informatiques et une application relative à la gestion d'un planning des forages.

Caractéristiques de l'approche proposée

Dans notre travail, nous nous sommes davantage consacrés à proposer une démarche quant à la spécification d'un noyau du système d'information, c'est-à-dire, à une description statique des traitements et des données caractéristiques du type d'application à l'exclusion de toute contrainte, telles que :

- des contraintes de réalisation liées à l'affectation des processeurs;

- l'expression de la dynamique d'enchainement des traitements;

- des contraintes quant aux performances attendues;

....

La spécification du noyau se décompose :

- d'une part, en la spécification des fonctions devant être réalisées par le système (l'énoncé du problème);

- d'autre part, en la spécification des différents types de données manipulés (l'univers du problème).

Tant l'énoncé que l'univers du problème se présentent de manière structurée (arborescente); les types de données se présentent sous la forme de types abstraits [LIS,75] [GUT,77] .

Les descriptions de l'énoncé et de l'univers sont données à la fois sous forme de texte en langue naturelle (en vue de faciliter, de la part du demandeur, le contrôle de l'adéquation de la spécification à ses besoins) et sous forme de propriétés formelles (exprimées en utilisant un sous-ensemble du calcul des prédicats du premier ordre typé).

De plus, pour faciliter la tâche du spécifieur, nous avons développé une bibliothèque de primitives spécifiques au domaine d'application considéré.

La démarche suggérée de construction de l'énoncé et de l'univers est caractérisée par :

- une analyse non-procédurale, tout séquencemenet étant dicté uniquement par la dépendance des données;

- un processus d'analyse dans lequel on spécifie en parallèle l'énoncé et l'univers;

- une démarche descendante et déductive permettant de structurer conjointement l'énoncé et l'univers;

- l'existence d'un catalogue de stratégies de spécification possibles susceptibles de guider, à chaque étape, le spécifieur dans la construction de la spécification; l'application successive des différentes stratégies indique la suite des décisions qui ont conduit à l'élaboration de la spécification.

Prolongements et approfondissements

Nous avons déjà signalé, dans le chpitre 6, un certain nombre de points méritant un approfondissement dans un travail ultérieur; ainsi, nous nous contenterons ici d'indiquer deux autres points constituant des axes de recherche privilégiés dans le cadre de ce travail.

Un premier sujet est davantage pratique; il est relatif à la présentation des spécifications aux demandeurs et utilisateurs.

Les modèles de spécification basés sur le formalisme des trois colonnes (cfr. chapitre 2), même s'ils sont très complets et relativement compréhensibles par un informaticien, ne paraissent cependant pas constituer un excellent moyen de communication avec les utilisateurs.

Notre expérience nous a, en effet, indiqué que les utilisateurs :

- préfèrent des spécifications présentées graphiquement plutôt que sous la forme de textes;

- appréhendent plus facilement des modèles où le nombre de concepts utilisés n'est pas très grand (même si le trop petit nombre de concepts utilisés ne permet pas d'exprimer totalement la richesse d'une spécification).

Compte tenu de ces deux remarques, nous pensons qu'il est nécessaire de pouvoir transformer et présenter nos spécifications dans d'autres formalismes tels que :

- celui du modèle entité-association [CHE,76] dans l'expression de l'univers;

- celui du modèle "Diagramme de flux de données" [WEI,78] dans l'expression de l'énoncé.

Un second sujet, davantage théorique, est lié au couplage possible entre le domaine de la spécification de systèmes d'information et celui de l'Intelligence Artificielle.

Deux voies de recherche nous paraissent intéressantes à explorer :

1. L'aspect lié au dialogue en langue naturelle

   Il est certain qu'il serait beaucoup plus agréable à l'utilisateur de pouvoir spécifier son problème dans un sous-ensemble de la langue naturelle. Dans ce cas, le système devrait pouvoir convertir, à l'aide de règles de dérivation la description de l'utilisateur en une spécification formelle respectant la syntaxe édictée.

Un exemple d'un tel système est le système PSI [GRE,76] dont le domaine d'application concerne des problèmes de calcul symbolique tels que traitement d'ensembles, de listes, recherches, tri, etc. Nous pensons qu'il est également envisageable dans le domaine des applications de gestion; d'autant plus, qu'une expérience telle que celle décrite dans [MAL,75] en démontre la faisabilité (78 % des phrases utilisées par 23 gestionnaires tombent dans 10 formes syntaxiques de base et couvrent un vocabulaire d'un millier de mots).

2. L'aspect lié au problème de la réutilisation des connaissances

Dans le chapitre 6, nous avons déjà abordé la possibilité d'unification a posteriori des parties de spécification; cependant, il serait également agréable de pouvoir éliminer cette redondance a priori par un dialogue avec l'utilisateur.

Pour cela, deux étapes sont envisageables :

a) Identifier des parties de spécification pouvant être réutilisées suite à un dialogue avec l'utilisateur quant à la nature du problème qu'il se pose;

b) Fournir des outils permettant à l'utilisateur de modifier la partie de la spécification réutilisée pour que celle-ci corresponde exactement à ses besoins.

Un exemple d'un tel système est BDS [HOW,75b] , son objectif est de permettre à tout utilisateur capable d'écrire et d'éditer des programmes en BDL [HAM,77] de construire et de modifier son propre logiciel individualisable (une description plus détaillé de ce système favorisant une programmation évolutive se trouve dans [LAM,83]).

*

* *

Annexe

EXEMPLE DE DEROULEMENT DU META-ALGORITHME

ANNEXE: exemple de déroulement du méta-algorithme

Dans cette annexe, nous illustrons le déroulement du méta-algorithme sur l'exemple proposé dans le chapitre 4.4..

Nous présentons, par la suite, la spécification relation par relation; cependant, la lecture n'est pas séquentielle car elle reflète le caractère descendant et itératif de la méthode suggérée.

Pour chacune des relations, sont décrits successivement les éléments suivants:

1) les arguments et réultats préalablement identifiés avant l'analyse de la relation;
2) la stratégie de spécification choisie par le spécifieur (cf. "étape 1")
3) les différentes questions auxquelles le spécifieur a à répondre suite au choix d'une certaine stratégie (cf. "étape 2", ..., "étapeN");
4) les differents choix exprimés par le spécifieur suite aux questions posées (ceux-ci sont indiqués en caractères gras);
5) l'appel au méta-algorithme lorqu'on veut analyser une nouvelle relation est indiqué par "Specif-rel"

Remarque: les commentaires sont indiqués entre []

A différents moments de la construction d'une relation, nous exprimons les spécifications déjà obtenues en présentant une visualisation sous la forme d'un pavé de traitement. La présence de "..." indique que la relation n'est pas encore complètement spécifiée.

Aux pages A66-A68, sont successivement présentés:

- l'arbre des traitements associé au sous-problème traité;
- l'arbre des types associé au résultat étudié;
- un exemple de pavé de donnée "DOC-COOR-CLIENT"

INITIALEMENT

(res) = Gestion-ventes (. . .) res: RESULTAT

Gestion-ventes: RESULTAT <-- . . .

Gestion-ventes: Application de gestion des ventes

RESULTAT: Différents types de bordereau émis par la société à destination du client

res: Différents documents émis

Specif-rel (Gestion-ventes) [cf. page A.3]

GESTION-VENTES

(res) = Gestion-ventes (. . .) res: RESULTAT

étape 1: Choix de la stratégie

Choix de la stratégie déductive d'éclatement d'une suite car le résultat est perçu comme la composition de plusieurs suites distinctes (cf. démarche globale d'analyse)

étape 2: Choix du constructeur associé au type du résultat (SUITE ou ENS)

RESULTAT = SUITE [ ]

étape 3: Mise en évidence des différentes sous-suites constitutives de la suite résultat

définition formelle: **fact*, rejet***

fact*: **FACTURE'S**

rejet*: **REJET'S**

définition informelle:

fact*: **suite des factures**

rejet*: **suite des documents de rejet**

FACTURE'S: **Documents de facturation émis par la société à destination des clients**

REJET'S: **Documents de rejet émis par la société à destination de ses clients**

étape 4: Spécification à l'aide d'une des primitives définies sur les suites des éventuelles interférences entre sous-suites mises en évidence

définition formelle

res = Fusion (fact*, rejet*)

étape 5: Définition de chacune des sous-suites intermédiaires mises en évidence

définition formelle

fact* = Gestion-envois (. . .)

rejet* = Gestion-rejets (. . .)

définition informelle

Gestion-envois: Sous-partie de l'application géreant les envois de produits aux clients

Gestion-rejets: Sous-partie de l'application géreant le non-envoi de produits aux clients

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.5]

étape 6:

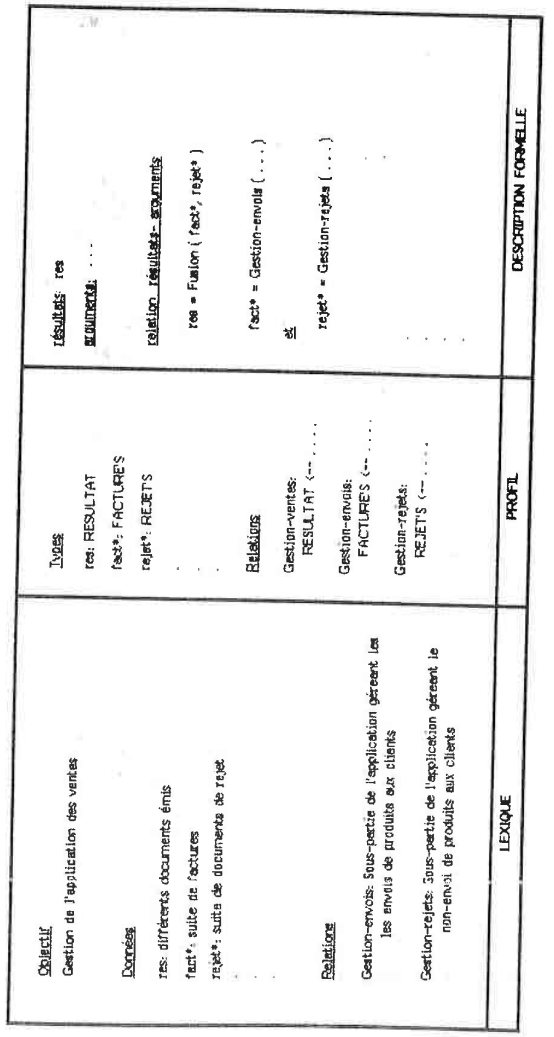
Specif-rel (Gestion-envois) [cf. page A.7]

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.6]

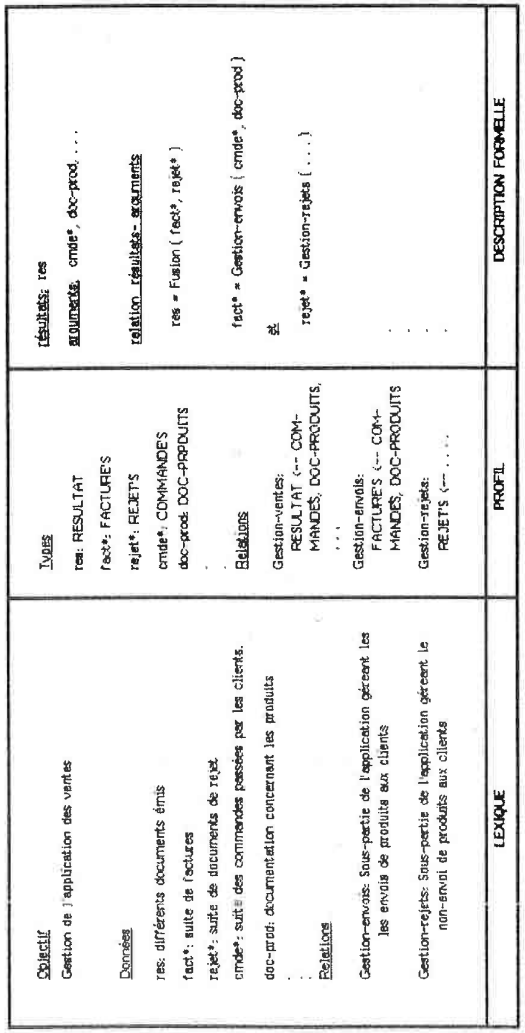
étape 7:

Specif-rel (Gestion-rejet)

[Cette relation n'est pas explicitée dans les pages qui suivent]



Spécification du problème: Gestion-ventes



GESTION-ENVOIS

fact* = Gestion-envois (. . .) fact*: FACTURE'S

étape 1: Choix de la stratégie

Choix de la stratégie déductive de décomposition d'une suite; une suite "guide" pouvant être mis en évidence (cf. démarche globale d'analyse)

étape 2: définition du type du résultat

définition formelle: **FACTURE'S = SUITE [FACTURE]**

définition informelle:

FACTURE: Facture émise par la société à destination d'un client lors de l'envoi de produits

étape 3: Définition de la suite guide

définition formelle: **livr*: SUITE [LIVRAISON]**

définition informelle:

LIVRAISON: Document établissant la livraison de produits

livr*: suite des livraisons établies

étape 4: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

fact = Def-facture (livr, . . .)

définition informelle:

Def-facture: établissement d'une facture

fact: facture produite

livr: livraison établie

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.9]

étape 5:

Specif-rel (Def-facture) [cf. page A.11]

étape 6: Définition de la relation de construction des données intermédiaires

définition formelle

livr* = Gestion-livraisons (. . .)

définition informelle

Gestion-livraisons: Gestion des livraisons répondant à des commandes

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A10]

étape 7:

Specif-rel (Gestion-livraisons)

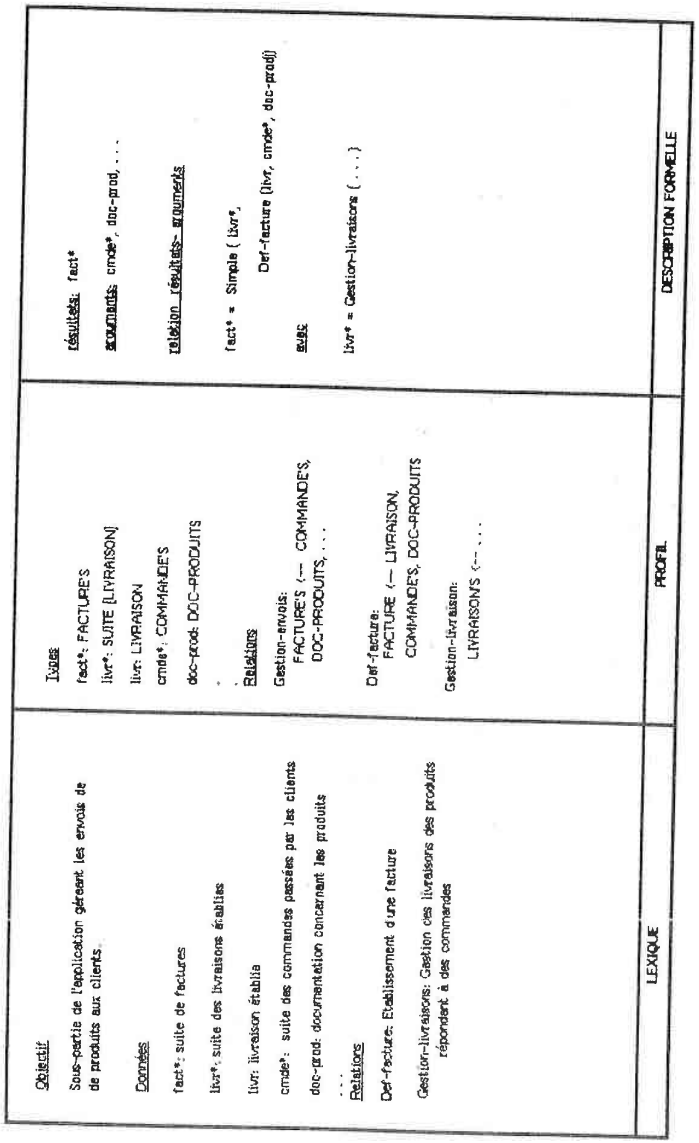
[Cette relation n'est pas explicitée dans les pages qui suivent]

[Retour à la page A-4]

Spécification du problème: Gestion-envois

Objectif	Types	résultats: fact*
Sous-partie de l'application gérant les envois de de produits aux clients	fact*: FACTURES livr*: SUITE [LIVRAISON] livr: LIVRAISON	arguments: . . . relation résultats- arguments fact* = Simple (livr*, Def-facture (livr, . . .))
Données fact*: suite de factures livr*: suite des livraisons établies livr: livraison établie . . .	Relations Gestion-envois: FACTURES <-- . . . . . . Def-facture: FACTURE <-- LIVRAISON, . . .	
Relations Def-facture: Etablissement d'une facture		
LEXIQUE	PROFIL	DESCRIPTION FORMELLE

Spécification du problème: Gestion-envois



DEF-FACTURE

fact= Def-facture (livr) fact: FACTURE
livr: LIVRAISON

étape 1: Choix de la stratégie

Choix de la stratégie déductive de construction d'un produit cartésien (application de la démarche globale d'analyse)

étape 2: Définition du type du résultat

définition formelle: **FACTURE = PC [COOR-CLIENT, CORPS-FACTURE]**

définition informelle:

COOR-CLIENT: Coordonnées d'identification d'un client

CORPS-FACTURE: Renseignements sur la facture relatifs aux produits vendus

étape 3: Mise en évidence des nouveaux objets à définir

définition formelle: **id-cli-fact: COOR-CLIENT**
corps-fact: CORPS-FACTURE

définition informelle:

id-cli-fact: identification du client figurant sur la facture

corps-fact: corps de la facture

étape 4: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

(id-cli-fact) = Def-coord-cli (Num-cli(:livr), ...)

(corps-fact) = Def-corps-fact (Corps-livr(:livr), ...)

définition informelle

Def-coord-cli: Définition des coordonnées du client devant figurer sur la facture

Def-corps-fact: Définition du corps de la facture

étape 5: Définition des intermédiaires apparus

définition formelle:

LIVRAISON = PC [Num-client: NUM-CLIENT, Corps-livr: CORPS--LIVRAISON]

NUM-CLIENT: ENTIER

(0 < num-cli ≤ 9999, num-cli: NUM-CLIENT)

définition informelle:

NUM-CLIENT: numéro identifiant le client

CORPS-LIVRAISON: livraison établie de produits suite à une commande

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.13]

étape 6

Specif-rel (Def-coord-cli) [cf. page A.16]

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.14]

étape 7:

Specif-rel (Def-corps-fact) [cf. page A.36]

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page15]

[Remarque: il n'y a pas eu de données intermédiaires communes à spécifier]

[Retour à la page A-8]

Spécification du problème: Def-facture

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Etablissement d'une facture Données fact: facture établie livr: livraison établie id-cli-fact: identification du client figurant sur la facture corps-fact: renseignements sur la facture relatifs aux produits vendus . . . Relations Def-coord-cli: Définition des coordonnées du client devant figurer sur la facture Def-corps-fact: Définition du corps de la facture . . .	Types fact: FACTURE livr: LIVRAISON id-cli-fact: COOR-CLIENT corps-fact: CORPS-FACTURE . . . Relations Def-facture: FACTURE <-- LIVRAISON, . . . Def-coord-cli: COOR-CLIENT <-- NUM-CLIENT, . . . Def-corps-fact: CORPS-FACTURE <-- CORPS-LIVRAISON . . .	résultats: fact arguments: livr relation résultats- arguments fact = (id-cli-fact, corps-fact) id-cli-fact = Def-coord-cli (Num-client(livr), . . .) et corps-fact = Def-corps-fact (Corps-livr(livr), . . .)

Spécification du problème: Def-facture

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Etablissement d'une facture Données fact: facture établie livr: livraison établie id-cli-fact: identification du client figurant sur la facture corps-fact: renseignements sur la facture relatifs aux produits vendus cmde*: suite des commandes passées par les clients . . Relations Def-coord-cli: Définition des coordonnées du client devant figurer sur la facture Def-corps-fact: Définition du corps de la facture . . .	Types fact: FACTURE livr: LIVRAISON id-cli-fact: COOR-CLIENT corps-fact: CORPS-FACTURE cmde*: COMMANDE'S . Relations Def-facture: FACTURE <-- LIVRAISON, . . . Def-coord-cli: COOR-CLIENT <-- NUM-CLIENT, COMMANDE'S Def-corps-fact: CORPS-FACTURE <-- CORPS-LIVRAISON, . . .	résultats: fact arguments: livr relation résultats- arguments fact = (id-cli-fact, corps-fact) id-cli-fact = Def-coord-cli (Num-client(: livr), cmde*) et corps-fact = Def-corps-fact (Corps-livr(: livr), . . .)

Spécification du problème: Def-facture

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Etablissement d'une facture Données fact: facture établie livr: livraison établie id-cli-fact: identification du client figurant sur la facture corps-fact: renseignements sur la facture relatifs aux produits vendus cmde*: suite des commandes passées par les clients doc-prod: documentation concernant les produits Relations Def-coord-cli: Définition des coordonnées du client devant figurer sur la facture Def-corps-fact: Définition du corps de la facture	Types fact: FACTURE livr: LIVRAISON id-cli-fact: COOR-CLIENT corps-fact: CORPS-FACTURE doc-prod: DOC-PRODUITS cmde*: COMMANDE'S Relations Def-facture: FACTURE <-- LIVRAISON, COMMANDE'S, DOC-PRODUITS Def-coord-cli: COOR-CLIENT <-- NUM-CLIENT, COMMANDE'S Def-corps-fact: CORPS-FACTURE <-- CORPS-LIVRAISON, DOC-PRODUITS	résultats: fact arguments: livr, cmde*, doc-prod relation résultats- arguments fact = (id-cli-fact, corps-fact) id-cli-fact = Def-coord-cli (Num-client(: livr), cmde*) et corps-fact = Def-corps-fact (Corps-livr(: livr), doc-prod)

DEF-COORD-CLI

id-cli-fact = Def-coord-cli (num-cli-livr, . . .)
id-cli-fact: COOR-CLIENT
num-cli-livr: NUM-CLIENT

étape 1: Choix de la stratégie

Choix de la stratégie déductive de construction d'un produit cartésien

étape 2: Définition du type du résultat

définition formelle:

COOR-CLIENT = PC [NUM-CLIENT, NOM-CLIENT, ADRESSE-CLIENT]
NOM-CLIENT: CHARST
ADRESSE-CLIENT: CHARST

définition informelle:

NOM-CLIENT: nom du client
ADRESSE-CLIENT: adresse du client

[Remarque: NUM-CLIENT a déjà été défini]

étape 3: Mise en évidence des nouveaux objets à définir

définition formelle:

num-cli-fact: NUM-CLIENT
nom-cli-fact: NOM-CLIENT
adr-cli-fact: ADRESSE-CLIENT

définition informelle:

num-cli-fact: numéro du client figurant sur la facture
nom-cli-fact: nom du client figurant sur la facture
adr-cli-fact: adresse du client figurant sur la facture

étape 4: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

num-cli-fact = num-cli-livr
nom-cli-fact = Recherche-nom-cli (doc-cli, num-cli-livr)
adr-cli-fact = Recherche-adr-cli (doc-cli, num-cli-livr)

définition informelle:

Recherche-nom-cli: recherche du nom du client dans la documentation "clients"
Recherche-adr-cli: recherche de l'adresse du client dans la documentation "clients"

étape 5: Définition des intermédiaires apparus

définition formelle:

num-cli-livr: NUM-CLIENT
doc-cli: DOC-COOR-CLIENT

définition informelle:

DOC-COOR-CLIENT: il s'agit d'un répertoire contenant les caractéristiques d'identification des clients
doc-cli: documentation sur les clients de la société
num-cli-livr: numéro du client figurant sur la livraison

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.19]

étape 6:

Specif-rel (Recherche-nom-cli) [cf. page A.21]

étape 7:

Spécif-rel (Recherche-adr-cli) [cf. page A.23]

étape 8: Expression de la relation définissant la construction de l'intermédiaire

définition formelle:

doc-cli = Def-doc-cli (...)

définition informelle:

Def-doc-cli: définition de la documentation concernant les identifications du client

étape 9:

Specif-rel (Def-doc-cli) [cf. page A.25]

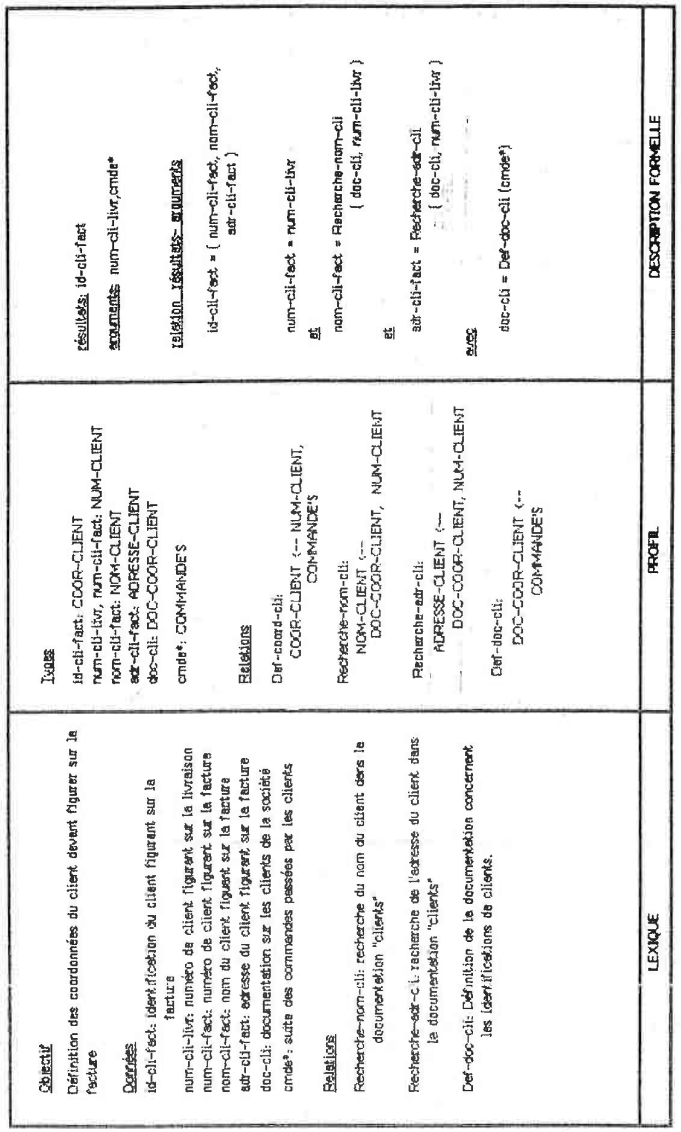
[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.20]

[Retour à la page A-12]

Spécification du problème: Def-coord-cli

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Définition des coordonnées du client devant figurer sur la facture Données id-cli-fact: identification du client figurant sur la facture num-cli-livr: numéro de client figurant sur la livraison num-cli-fact: numéro de client figurant sur la facture nom-cli-fact: nom du client figurant sur la facture adr-cli-fact: adresse du client figurant sur la facture doc-cli: documentation sur les clients de la société . . . Relations Recherche-nom-cli: recherche du nom du client dans la documentation "clients" Recherche-adr-cli: recherche de l'adresse du client dans la documentation "clients" . . .	Types id-cli-fact: COOR-CLIENT num-cli-livr, num-cli-fact: NUM-CLIENT nom-cli-fact: NOM-CLIENT adr-cli-fact: ADRESSE-CLIENT doc-cli: DOC-COOR-CLIENT . . . Relations Def-coord-cli: COOR-CLIENT <-- NUM-CLIENT, . . . Recherche-nom-cli: NOM-CLIENT <-- DOC-COOR-CLIENT, NUM-CLIENT Recherche-adr-cli: ADRESSE-CLIENT <-- DOC-COOR-CLIENT, NUM-CLIENT . . .	résultats: id-cli-fact arguments: num-cli-livr, . . . relation résultats-arguments id-cli-fact = (num-cli-fact, nom-cli-fact, adr-cli-fact) num-cli-fact = num-cli-livr et nom-cli-fact = Recherche-nom-cli (doc-cli, num-cli-livr) et adr-cli-fact = Recherche-adr-cli (doc-cli, num-cli-livr) . . .

Spécification du problème: Def-coord-cli



RECHERCHE-NOM-CLI

nom-cli-fact = Recherche-nom-cli (doc-cli, num-cli-livr)

nom-cli-fact: NOM-CLIENT

num-cli-livr: NUM-CLIENT

doc-cli: DOC-COOR-CLIENT

étape 1: Choix de la stratégie

Choix de la stratégie de réutilisation des connaissances

étape 2: Définition du type de la donnée.

définition formelle:

DOC-COOR-CLIENT = TABLE [NUM-CLIENT —> RENS-CLIENT]

RENS-CLIENT = PC [Nom-client: NOM-CLIENT, Adresse-client: ADRESSE-CLIENT]

définition informelle:

RENS-CLIENT: renseignements caractérisant un client à l'exception de son numéro

étape 3: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

nom-cli-fact = Nom-client (: Accès(doc-cli, num-cli-livr))

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.22]

[Remarque: Lors de l'utilisation de primitives (tel Accès), on n'indique pas de profil, ni d'explication dans le lexique]

[Retour à la page A-17]

Spécification du problème: Recherche-nom-cli

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Recherche du nom du client dans la documentation "clients" Données nom-cli-fact: nom du client figurant sur la facture doc-cli: documentation sur les clients de la société num-cli-livr: numéro du client figurant sur la livraison Relations	Types nom-cli-fact: NOM-CLIENT num-cli-livr: NUM-CLIENT doc-cli: DOC-COOR-CLIENT Relations Recherche-nom-cli: NOM-CLIENT <-- DOC-COOR-CLIENT, NUM-CLIENT	résultats: nom-cli-fact arguments: doc-cli, num-cli-livr relation résultats-arguments nom-cli-fact = Nom-client (: Accès (doc-cli, num-cli-livr))

RECHERCHE-ADR-CLI

adr-cli-fact = Recherche-adr-cli (doc-cli, num-cli-livr)
adr-cli-fact: ADRESSE-CLIENT
doc-cli: DOC-COOR-CLIENT
num-cli-livr: NUM-CLIENT

étape 1: Choix de la stratégie

Choix de la stratégie de réutilisation des connaissances

étape 2: Expression de la relation définissant la construction de chaque résultat.

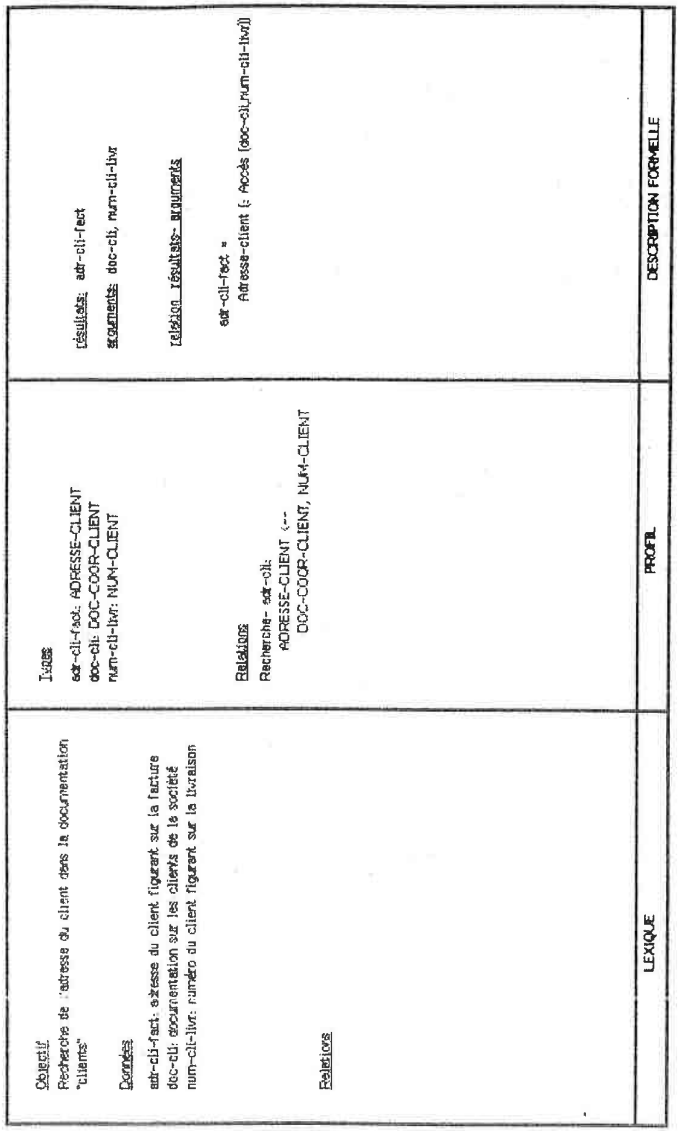
définition formelle:

adr-cli-fact = Adresse-client(: Accès(doc-cli, num-cli-livr))

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.24]

[Retour à la page A-17]

Spécification du problème: Recherche-adr-cli



DEF-DOC-CLI

doc-cli = Def-doc-cli (...)
doc-cli: DOC-COOR-CLIENT

étape 1: Choix de la stratégie

Choix de la stratégie déductive de construction d'une table (on sait, en effet, que doc-cli est une table.)

étape 2: Définition de la suite des mises-à-jour

définition formelle:

maj-coord-cli*: SUITE [MAJ-COORD-CLI]

définition informelle:

MAJ-COORD-CLI: mise-à-jour des coordonnées d'un client à repercuter dans la documentation

maj-coord-cli: mise-à-jour de l'identification d'un client

étape 3: Définition de la suite guidant la construction de la suite des mises-à-jour.

définition formelle:

cmde*: COMMANDE'S

définition informelle:

cmde*: suite des commandes passées par des clients

COMMANDE'S: commandes émanant des clients de la société

étape 4. Expression de la relation définissant la construction de chaque résultat.

définition formelle:

maj-coord-cli = Def-maj-coord-cli (cmde, doc-cli)

cmde: COMMANDE

doc-cli: DOC-COOR-CLIENT

définition informelle:

Def-maj-coord-cli: Définition d'une modification des coordonnées du client à répercuter dans la documentation

cmde: commande d'un client

COMMANDE: commande émanant d'un client de la société

doc-cli: état de la documentation avant de répercuter la mise-à-jour

étape 5:

Spécif-rel (Def-maj-coord-cli) [cf. page A.28]

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.27]

[Remarque: cmde* n'est pas défini à l'aide d'une relation intermédiaire (règle du avec) car il s'agit d'un argument du problème]

[Retour à la page A-17]

Spécification du problème: Def-doc-cli

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Définition de la documentation concernant les identifications de clients Données doc-cli: documentation sur les clients de la société cmde*: suite des commandes passées par les clients cmde: commande d'un client doc-cli: documentation intérieure sur les clients de la société maj-coord-cli: mise-à-jour de l'identification d'un client Relations Def-maj-coord-cli: Définition d'une modification des coordonnées du client à répercuter dans la documentation	Types doc-cli, doc-cli: DOC-COOR-CLIENT cmde*: COMMANDES maj-coord-cli: MAJ-COORD-CLI cmde: COMMANDE Relations Def-doc-cli: DOC-COOR-CLIENT <-- COMMANDES Def-maj-coord-cli: MAJ-COORD-CLI <-- COMMANDE, DOC-COOR-CLIENT	résultats: doc-cli arguments: cmde* relation résultats-arguments doc-cli = Tabulation (maj-coord-cli*) maj-coord-cli* = Simple (cmde*, Def-maj-coord-cli (cmde, doc-cli))

DEF-MAJ-COORD-CLI

maj-coord-cli = Def-maj-coord-cli (cmde, $\overline{\text{doc-cli}}$)

maj-coord-cli: MAJ-COORD-CLI

cmde: COMMANDE

$\overline{\text{doc-cli}}$: DOC-COOR-CLIENT

étape 1: Choix de la stratégie

Choix de la stratégie déductive de construction d'une table (suite de l'application de la stratégie commencée au pavé précédent)

étape 2: Définition du type du résultat

définition formelle:

MAJ-COORD-CLI = PC [NUM-CLIENT, RENS-CLIENT°]

étape 3: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

maj-coord-cli = Def-nouveau-client (cmde, $\overline{\text{doc-cli}}$)
si Nouveau-client (cmde)

maj-coord-cli = Def-modif-nom-client (cmde, $\overline{\text{doc-cli}}$)
si Nom-client-changé (cmde, $\overline{\text{doc-cli}}$)

maj-coord-cli = Def-modif-adr-client (cmde, $\overline{\text{doc-cli}}$)
si Adr-client-changée(cmde, $\overline{\text{doc-cli}}$)

maj-coord-cli = Def-modif-ident-client (cmde, $\overline{\text{doc-cli}}$)
si Identité-client-changée (cmde, $\overline{\text{doc-cli}}$)

définition informelle:

Def-nouveau-client: Définition d'une mise-à-jour correspondant à l'adjonction d'un nouveau client

Nouveau-client: Prédicat indiquant que l'identité du client sur

la commande correspond à celle d'un nouveau client

Def-modif-nom-client: Définition d'une mise-à-jour entraînant la modification du nom d'un client connu

Nom-client-changé: Prédicat indiquant qu'un client connu a changé de nom

Def-modif-adr-client: Définition d'une mise-à-jour entraînant la modification de l'adresse d'un client connu

Adr-client-changée: Prédicat indiquant qu'un client connu a changé d'adresse

Def-modif-ident-client: Définition d'une mise-à-jour entraînant la modification de l'identité d'un client connu (c-à-d. nom et adresse)

Identité-client-changée: Prédicat indiquant qu'un client connu a changé d'identité

étape 4.

Specif-rel (Nouveau-client) [cf. page A.34]

Specif-rel (Def-nouveau-client) [cf. page A.35]

étape 5.

Specif-rel (Nom-client-changé)

Specif-rel (Def-modif-nom-client)

[Remarque: ces relations n'ont pas été explicitées dans les pages qui suivent]

étape 6.

Specif-rel (Adr-client-changée)

Specif-rel (Def-modif-adr-changée)

[Remarque: ces relations n'ont pas été explicitées dans les pages qui suivent]

étape 7.

Specif-rel (Identité-client-changée)

Specif-rel (Def-modif-ident-client)

[Remarque: ces relations n'ont pas été explicitées dans les pages qui suivent]

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A-31]

[Retour à la page A-26]

Spécification du problème: Def-maj-coord-cli

LEXIQUE

Objectif
Définition d'une modification des coordonnées du client à répercuter dans la documentation

Données
maj-coord-cli: mise-à-jour de l'identification d'un client
doc-cli: documentation antérieure sur les clients de la société
cmde: commande d'un client

Relations
Def-nouveau-client: Définition d'une mise-à-jour correspondant à l'adjonction d'un nouveau client
Nouveau-client: Prédicat indiquant que l'identité du client sur la commande correspond à celle d'un nouveau client
Def-modif-nom-client: Définition d'une mise-à-jour entraînant la modification du nom d'un client connu
Nom-client-changé: Prédicat indiquant qu'un client a changé de nom
Def-modif-adr-client: Définition d'une mise-à-jour entraînant la modification de l'adresse d'un client connu
Adr-client-changée: Prédicat indiquant qu'un client connu a changé d'adresse
Def-modif-ident-client: Définition d'une mise-à-jour entraînant la modification de l'identité d'un client connu (c'est-à-dire nom et adresse)
Identité-client-changée: Prédicat indiquant qu'un client connu a changé d'identité

PROFIL

Types
maj-coord-cli: MAJ-COORD-CLI
doc-cli: DOC-COOR-CLIENT
cmde: COMMANDE

Relations
Def-nouveau-client: MAJ-COORD-CLI <--
COMMANDE
Nouveau-client: BOOLEEN <--
COMMANDE, DOC-COOR-CLIENT
Def-modif-nom-client: MAJ-COORD-CLI <--
COMMANDE, DOC-COOR-CLIENT
Nom-client-changé: BOOLEEN <--
COMMANDE, DOC-COOR-CLIENT
Def-modif-adr-client: MAJ-COORD-CLI <--
COMMANDE, DOC-COOR-CLIENT
Adr-client-changée: BOOLEEN <--
COMMANDE, DOC-COOR-CLIENT
Def-modif-ident-client: MAJ-COORD-CLI <--
COMMANDE, DOC-COOR-CLIENT
Identité-client-changée: BOOLEEN <--
COMMANDE, DOC-COOR-CLIENT

DESCRIPTION FORMELLE

résultats: maj-coord-cli

arguments: cmde, doc-cli

relation liant les arguments

maj-coord-cli =

Def-nouveau-client (cmde, doc-cli)
si Nouveau-client (cmde)

Def-modif-nom-client (cmde, doc-cli)
si Nom-client-changé (cmde,doc-cli)

Def-modif-adr-client (cmde, doc-cli)
si Adr-client-changée (cmde, doc-cli)

Def-modif-ident-client (cmde, doc-cli)
si Identité-client-changée (cmde,doc-cli)

NOUVEAU-CLIENT

bool = Nouveau-client (cmde)

bool: BOOLEEN

cmde: COMMANDE

étape 1: Choix de la stratégie

Choix d'une stratégie inductive basée sur la structure de l'argument

étape 2: Définition de la donnée

définition formelle:

COMMANDE = PC [Nom-client: NOM-CLIENT°,
Adresse-client: ADRESSE-CLIENT°,
Numéro-client: NUMERO-CLIENT°, . . .]

étape 3: Expression de la relation définissant le prédicat

définition formelle:

Nom-client-présent(cmde) ∧ Adresse-client-présent(cmde) ∧
Non(Numéro-client-présent(cmde))

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.31]

[Retour à la page A-29]

Spécification du problème: Nouveau-client

Objectif Prédicat indiquant que l'identité du client sur la commande correspond à celle d'un nouveau client Données cmde: commande d'un client Relations	Types Bool: BOOLEEN cmde: COMMANDE Relations	Résultats: Bool Arguments: cmde Relation résultats-arguments Nom-client-présent (cmde) ∧ Adresse-client-présent (cmde) ∧ non Numéro-client-présent (cmde)
LEXIQUE	PROFIL	DESCRIPTION FORMELLE

DEF-NOUVEAU-CLIENT

maj-coord-cli = Def-nouveau-client (cmde, $\overline{\text{doc-cli}}$)

maj-coord-cli: MAJ-COORD-CLI

cmde: COMMANDE

doc-cli: DOC-COOR-CLIENT

étape 1: Choix de la stratégie

Choix de la stratégie déductive de construction d'un produit cartésien

étape 2: Définition du résultat

définition formelle:

num-cli: NUM-CLIENT

nom-cli: NOM-CLIENT

adr-cli: ADRESSE-CLIENT

définition informelle:

num-cli: nouveau numéro de client attribué

nom-cli: nom du nouveau client

adr-cli: adresse du client

étape 3: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

num-cli = Taille ($\overline{\text{doc-cli}}$) + 1

nom-cli = Nom-client(:cmde)

adr-cli = Adresse-client (:cmde)

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.35]

[Retour à la page A-29]

Spécification du problème: Def-nouveau-client

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Définition d'une mise-à-jour correspondant à l'adjonction d'un nouveau client Données maj-coord-cli: mise-à-jour de l'identification d'un client cmde: commande d'un client doc-cli: documentation antérieure sur les clients de la société num-cli: nouveau numéro de client attribué nom-cli: nom du nouveau client adr-cli: adresse du nouveau client Relations	Types cmde: COMMANDE doc-cli: DOC-COOR-CLIENT num-cli: NUM-CLIENT nom-cli: NOM-CLIENT adr-cli: ADRESSE-CLIENT maj-coord-cli : MAJ-coord-CLI Relations	résultats: maj-coord-cli arguments: cmde, doc-cli relation résultats= arguments maj-coord-cli = (num-cli,(nom-cli, adr-cli)) num-cli = Taille ($\overline{\text{doc-cli}}$) + 1 et nom-cli = Nom-client (: cmde) et adr-cli = Adresse-client (: cmde)

DEF-CORPS-FACT

corps-fact = Def-corps-fact (corps-livr, . . .)
corps-fact: CORPS-FACTURE
corps-livr: CORPS-LIVRAISON

étape 1: Choix de la stratégie

Choix de la stratégie déductive de construction d'une union (application du principe de généralisation; cf. démarche globale d'analyse)

étape 2: Définition du type du résultat

définition formelle:

CORPS-FACTURE = UNION [CORPS-PREMIERE-FACTURE, CORPS-FACTURE-DIFFEREE]

définition informelle:

CORPS-PREMIERE-FACTURE: corps de la facture correspondant à un premier envoi au client suite à sa commande

CORPS-FACTURE-DIFFEREE: corps de la facture correspondant à un envoi différé au client suite à sa commande

étape 3: Mise en évidence des nouveaux objets à définir

définition formelle:

corps-prem-fact: CORPS-PREMIERE-FACTURE
corps-fact-diff: CORPS-FACTURE-DIFFEREE

définition informelle:

corps-prem-fact: corps d'une première facture
corps-fact-diff: corps d'une facture différée

étape 4: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

Première-livraion (corps-livr) et

corps-prem-fact = Def-première-fact (corps-prem-livr, ...)
Livraison-différée (corps-livr) et
corps-fact-diff = Def-fact-différée (corps-livr-diff, ...)

définition informelle:

Première-livraison: prédicat indiquant qu'il s'agit d'une première livraison

Def-première-fact: définition du corps de la facture lorsqu'on envoie pour la première fois, suite à la commande, les produits commandés

Livraison-différée: prédicat indiquant qu'il s'agit d'une livraison différée

Def-fact-diff: définition du corps de la facture lorsqu'on envoie en différé les produits suite à la commande

étape 5: Définition des intermédiaires apparus

définition formelle:

corps-livr: CORPS-LIVRAISON
corps-prem-livr: CORPS-PREMIERE-LIVRAISON
corps-livr-diff: CORPS-LIVRAISON-DIFFEREE

définition informelle:

corps-livr: corps d'une livraison
corps-prem-livr: corps d'une première livraison
corps-livr-diff: corps d'une livraison différée
CORPS-PREMIERE-LIVRAISON: corps de la livraison correspondant à la préparation d'un premier envoi au client suite à sa commande
CORPS-LIVRAISON-DIFFEREE: corps de la livraison correspondant à la préparation d'un envoi différé au client suite à sa commande

étape 6:

Specif-rel (Première-livraison) [cf. page A.41]
Spécif-rel (Def-première-fact) [cf. page A.43]

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.39]

étape 7:

Specif-rel (Livraison-différée) [cf. page A.62]

Specif-rel (Def-fact-diff) [cf. page A.64]

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.40]

[Remarque: il n'y pas eu de données intermédiaires communes à spécifier]

[Retour à la page A-12]

Spécification du problème: Def-corps-fact

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Définition du corps de la facture Données corps-fact: renseignements sur la facture relatifs aux produits vendus corps-livr: corps d'une livraison corps-prem-fact: corps d'une première facture corps-fact-diff: corps d'une facture différée corps-prem-livr: corps d'une première livraison corps-livr-diff: corps d'une livraison différée . . . Relations Première-livraison: Prédicat indiquant qu'il s'agit d'une première livraison Def-première-facture: Définition du corps de la facture lorsqu'on envoie pour la première fois, suite à la commande, les produits commandés Livraison-différée: Prédicat indiquant qu'il s'agit d'une livraison différée Def-fact-diff: Définition du corps de la facture lorsqu'on envoie en différé les produits suite à la commande . . .	Types corps-fact: CORPS-FACTURE corps-livr: CORPS-LIVRAISON corps-prem-livr: CORPS-PREMIERE-LIVRAISON corps-livr-diff: CORPS-LIVRAISON-DIFFEREE corps-prem-fact: CORPS-PREMIERE-FACTURE corps-fact-diff: CORPS-FACTURE-DIFFEREE Relations Def-corps-fact: CORPS-FACTURE <--> CORPS-LIVRAISON, . . . Def-première-fact: CORPS-PREMIERE-FACTURE <--> CORPS-PREMIERE-LIVRAISON, . . . Première-livraison: BOOLEEN <--> CORPS-LIVRAISON Def-fact-différée: CORPS-FACTURE-DIFFEREE <--> CORPS-LIVRAISON-DIFFEREE, . . . Livraison-différée: BOOLEEN <--> CORPS-LIVRAISON . . .	résultats: corps-fact arguments: corps-livr, . . . relation résultats-arguments corps-fact = [corps-prem-fact, corps-fact-diff] Première-livraison (corps-livr) ssi corps-prem-fact = Def-première-fact (corps-prem-livr, . . .) ou Livraison-différée (corps-livr) ssi corps-fact-diff = Def-fact-diff (corps-livr-diff, . . .)

Spécification du problème: Def-corps-fact

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Définition du corps de la facture Données corps-fact: renseignements sur la facture relatifs aux produits vendus corps-livr: corps d'une livraison corps-prem-fact: corps d'une première facture corps-fact-diff: corps d'une facture différée corps-prem-livr: corps d'une première livraison corps-livr-diff: corps d'une livraison différée doc-prod: documentation concernant les produits Relations Première-livraison: Prédicat indiquant qu'il s'agit d'une première livraison Def-première-facture: Définition du corps de la facture lorsqu'on envoie pour la première fois , suite à la commande, les produits commandés Livraison-différée: Prédicat indiquant qu'il s'agit d'une livraison différée Def-fact-diff: Définition du corps de la facture lorsqu'on envoie en différé les produits suite à la commande	Types corps-fact: CORPS-FACTURE corps-livr: CORPS-LIVRAISON corps-prem-livr: CORPS-PREMIERE-LIVRAISON corps-livr-diff: CORPS-LIVRAISON-DIFFEREE corps-prem-fact: CORPS-PREMIERE-FACTURE corps-fact-diff: CORPS-FACTURE-DIFFEREE doc-prod: DOC-PRODUITS Relations Def-corps-fact: CORPS-FACTURE <-- CORPS-LIVRAISON, DOC-PRODUITS Def-première-fact: CORPS-PREMIERE-FACTURE <-- CORPS-PREMIERE-LIVRAISON, DOC-PRODUITS Première-livraison: BOOLEEN <-- CORPS-LIVRAISON Def-fact-différée: CORPS-FACTURE-DIFFEREE <-- CORPS-LIVRAISON-DIFFEREE, DOC-PRODUITS Livraison-différée: BOOLEEN <-- CORPS-LIVRAISON	résultats: corps-fact arguments: corps-livr, doc-prod relation résultats-arguments corps-fact = [corps-prem-fact, corps-fact-diff] Première-livraison (corps-livr) ssi corps-prem-fact = Def-première-fact (corps-prem-livr, doc-prod) ou Livraison-différée (corps-livr) ssi corps-fact-diff = Def-fact-diff (corps-livr-diff, doc-prod)

PREMIERE-LIVRAISON

bool = Première-livraison (corps-livr)

bool: BOOLEEN

corps-livr: CORPS-LIVRAISON

étape 1: Choix de la stratégie

Choix de la stratégie inductive basée sur la structure de l'argument

étape 2: Définition du type de la donnée.

définition formelle:

CORPS-LIVRAISON = UNION [CORPS-PREMIERE-LIVRAISON, CORPS-LIVRAISON-DIFFEREE]

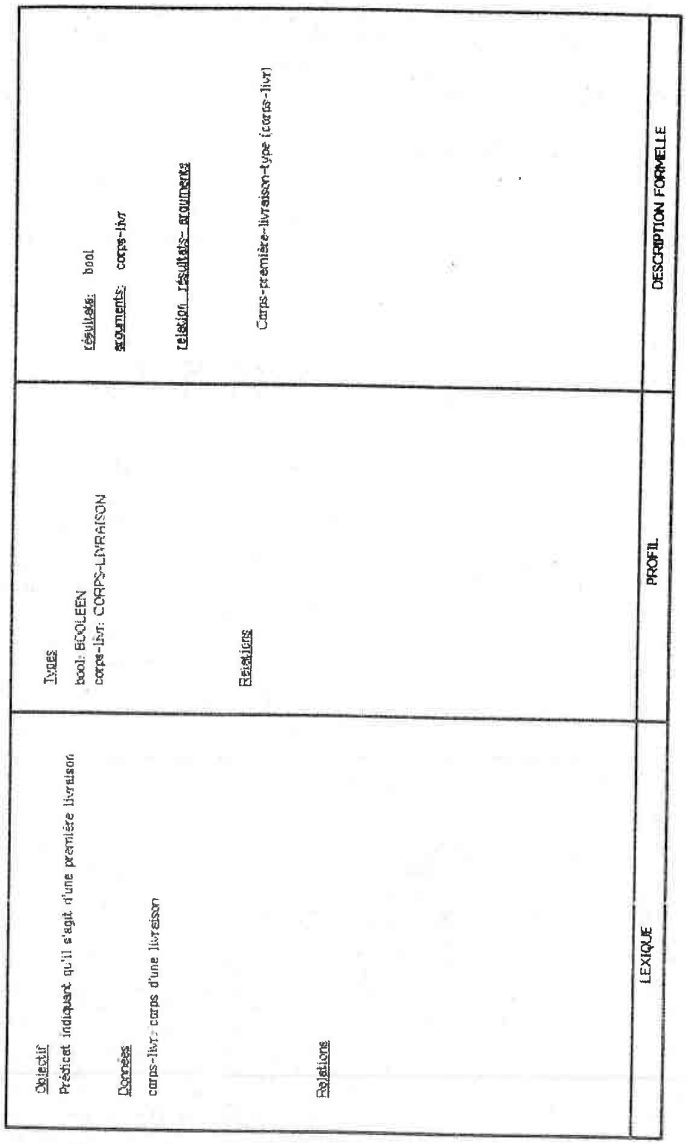
étape 3: Expression de la relation définissant le prédicat

définition formelle:

Corps-première-livraison-type (corps-livr)

[La spécification du pavé de traitement à ce stade se présente comme indiqué à la page A.42]

[Retour à la page A-37]



DEF-PREMIERE-FACT

corps-prem-fact = Def-première-fact (corps-prem-livr, ...)

corps-prem-fact: CORPS-PREMIERE-FACTURE

corps-prem-livr: CORPS-PREMIERE-LIVRAISON

étape 1: Choix de la stratégie

Choix de la stratégie déductive de construction d'un produit cartésien

étape 2: Définition du type du résultat

définition formelle:

CORPS-PREMIERE-FACTURE = PC [LIGNES-FACTURE, FRAIS-TRANSPORT, TOTAL-A-PAYER]

définition informelle:

LIGNES-FACTURE: lignes concernant la facturation des produits livrés

FRAIS-TRANSPORT: frais de transport facturé au client

TOTAL-A-PAYER: montant à payer par le client

étape 3: Mise en évidence des nouveaux objets à définir

définition formelle:

lignes-fact: LIGNES-FACTURE

fr-trans: FRAIS-TRANSPORT = ENTIER

tot: TOTAL-A-PAYER = ENTIER

définition informelle:

lignes-fact: lignes de facturation

fr-trans: frais de transport à payer

tot: total à facturer

étape 4: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

lignes-fact = Def-lignes-facture (corps-prem-livr, ...)

fr-trans = '40'

tot = Def-total-à-payer (corps-prem-livr, ...)

définition informelle:

Def-lignes-facture: définition des lignes de produits facturés

Def-total-à-payer: calcul du montant à payer

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.45]

étape 5:

Specif-rel (Def-lignes-facture) [cf. page A.47]

étape 6:

Specif-rel (Def-total-à-payer) [cf. page A.60]

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.46]

[Retour à la page A-37]

Spécification du problème: Def-première-fact

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Définition du corps de la facture lorsqu'on envoie pour la première fois, suite à la commande, les produits commandés Données corps-prem-fact: corps d'une première facture corps-prem-livr: corps d'une première livraison lignes-fact: lignes de facturation fr-trans: frais de transport à payer tot: total facturé Relations Def-lignes-facture: Définition des lignes de produits facturés Def-total-à-payer: Calcul du montant total à payer	Inass corps-prem-fact: CORPS-PREMIERE-FACTURE corps-prem-livr: CORPS-PREMIERE-LIVRAISON lignes-fact: LIGNES-FACTURE fr-trans: FRAIS-TRANSPORT tot: TOTAL-A-PAYER . Relations Def-première-facture: CORPS-PREMIERE-FACTURE (— CORPS-PREMIERE-LIVRAISON, . . . Def-lignes-facture: LIGNES-FACTURE (— CORPS-PREMIERE-LIVRAISON, . . . Def-total-à-payer: TOTAL-A-PAYER (— CORPS-PREMIERE-LIVRAISON, . . .	résultats: corps-prem-fact arguments: corps-prem-livr, . . . relation résultats- arguments corps-prem-fact = (lignes-fact, fr-trans, tot) lignes-fact = Def-lignes-facture (corps-prem-livr, . . .) et fr-trans = '40' et tot = Def-total-à-payer (corps-prem-livr, . . .)

Spécification du problème : Def-première-fact

Objectif	Types	résultats: corps-prem-fact
Définition du corps de la facture lorsqu'on envoie pour la première fois suite à la commande, les produits commandés	corps-prem-fact: CORPS-PREMIERE-FACTURE	arguments: corps-prem-livr, doc-prod
Données	corps-prem-livr: CORPS-PREMIERE-LIVRAISON	relation résultats-arguments
corps-prem-fact: corps d'une première facture	lignes-fact: LIGNES-FACTURE	corps-prem-fact =
corps-prem-livr: corps d'une première livraison	fr-trans: FRAIS-TRANSPORT	(lignes-fact, fr-trans, tot)
lignes-fact: lignes de facturation	tot: TOTAL-A-PAYER	
fr-trans: frais de transport à payer	doc-prod: DOC-PRODUITS	lignes-fact = Def-lignes-facture (corps-prem-livr, doc-prod)
tot: total facturé	Relations	et
doc-prod: documentation concernant les produits	Def-première-facture: CORPS-PREMIERE-FACTURE (-- CORPS-PREMIERE-LIVRAISON, DOC-PRODUITS	fr-trans = "40"
Relations	Def-lignes-facture: LIGNES-FACTURE (-- CORPS-PREMIERE-LIVRAISON, DOC-PRODUITS	et
Def-lignes-facture: Définition des lignes de produits facturés	Def-total-à-payer: TOTAL-A-PAYER (-- CORPS-PREMIERE-LIVRAISON, DOC-PRODUITS	tot = Def-total-à-payer (corps-prem-livr, doc-prod)
Def-total-à-payer: Calcul du montant total à payer		
LEXIQUE	PROFIL	DESCRIPTION FORMELLE

DEF-LIGNES-FACTURE

lignes-fact = Def-lignes-facture (corps-prem-livr, ...)
lignes-fact: LIGNES-FACTURE
corps-prem-livr: CORPS-PREMIERE-LIVRAISON

étape 1: Choix de la stratégie

Choix de la stratégie déductive de décomposition d'une suite; une suite "guide" pouvant être mis en évidence (cf. démarche globale d'analyse)

étape 2: Définition du type du résultat

définition formelle:

LIGNES-FACTURE = SUITE [LIGNE-FACTURE]

définition informelle:

LIGNE-FACTURE: ligne de facturation d'un produit livré

étape 3: Définition de la suite guide

définition formelle:

CORPS-PREMIERE-LIVRAISON = SUITE [LIGNE-LIVRAISON]

définition informelle:

LIGNE-LIVRAISON: ligne caractérisant la quantité d'un produit livré

étape 4: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

ligne-fact = Def-ligne-fact (ligne-livr, ...)
ligne-fact: LIGNE-FACTURE
ligne-livr: LIGNE-LIVRAISON

définition informelle:

Def-ligne-fact: définition d'une ligne de facture
ligne-fact: ligne de facturation
ligne-livr: ligne de livraison

étape 5:

Specif-rel (Def-ligne-fact) [cf. page A.50]

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.49]

[Remarque: doc-prod n'est pas défini par une relation intermédiaire (règle du avec) car il s'agit d'un argument du problème]

[Retour à la page A-44]

Spécification du problème : Def-lignes-facture

Objectif Définition des lignes de produits facturés Données lignes-facture: lignes de facturation corps-prem-livr: corps d'une première livraison doc-prod: documentation concernant les produits ligne-fact: ligne de facturation ligne-livr: ligne de livraison Relations Def-ligne-fact: Définition d'une ligne de facture	Lyas lignes-fact: LIGNES-FACTURE corps-prem-livr: CORPS-PREMIERE-LIVRAISON doc-prod: DOC-PRODUITS ligne-fact: LIGNE-FACTURE ligne-livr: LIGNE-LIVRAISON Relations Def-lignes-facture: LIGNES-FACTURE <-- CORPS-PREMIERE-LIVRAISON, DOC-PRODUITS Def-ligne-facture: LIGNE-FACTURE <--- LIGNE-LIVRAISON, DOC-PRODUITS	résultats: ligne-fact arguments: corps-prem-livr, doc-prod relation résultats-arguments lignes-fact = ligne-fact* ligne-fact* = Simple (ligne-livr*, Def-ligne-fact (ligne-livr, doc-prod))
LEXIQUE	PROFIL	DESCRIPTION FORMELLE

DEF-LIGNE-FACT

ligne-fact = Def-ligne-fact (ligne-livr, ...)
ligne-fact: LIGNE-FACTURE
ligne-livr: LIGNE-LIVRAISON

étape 1: Choix de la stratégie

Choix de la stratégie déductive de construction d'un produit cartésien (application de la démarche globale d'analyse)

étape 2: Définition du type du résultat

définition formelle:

LIGNE-FACTURE = PC [LIBELLE-PRODUIT, QUANTITE-LIVREE, PRIX-UNITAIRE, MONTANT-LIGNE]
LIBELLE-PRODUIT: CHARST
PRIX-UNITAIRE: ENTIER
MONTANT-LIGNE: ENTIER
QUANTITE-LIVREE: ENTIER

définition informelle:

LIBELLE-PRODUIT: libellé du produit vendu
QUANTITE-LIVREE: quantité livrée d'un produit commandé
PRIX-UNITAIRE: prix unitaire du produit vendu
MONTANT-LIGNE: montant à payer inhérent à la livraison d'un produit

étape 3: Mise en évidence des nouveaux objets à définir

définition formelle:

lib: LIBELLE-PRODUIT
qté: QUANTITE-LIVREE
p-u: PRIX-UNITAIRE
mont: MONTANT-LIGNE

définition informelle:

lib: libellé du produit livré
qté: quantité de produit facturé
p-u: prix unitaire du produit facturé
mont: montant à payer pour le produit facturé

étape 4: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

lib = Recherche-libellé-produit (Num-prod(:ligne-livr), doc-prod)
qté = Quantité-livrée (:ligne-livr)
p-u = Recherche-prix-produit (Num-prod(:ligne-livr), doc-prod)
mont = Calcul-montant-ligne (Num-prod(ligne-livr), Quantité-livrée(:ligne-livr), doc-prod)

définition informelle:

Recherche-libellé-produit: recherche du libellé
Recherche-prix-produit: Recherche du prix unitaire associé au numéro d'un produit livré
Calcul-montant-ligne: calcul du montant facturé pour un produit livré

étape 5: Définition des intermédiaires apparus

définition formelle:

LIGNE-LIVRAISON = PC [Num-prod: NUMERO-PRODUIT, Quantité-livrée: QUANTITE-LIVREE]
NUMERO-PRODUIT: ENTIER (0 < num-prod ≤ 99999, num-prod: NUMERO-PRODUIT)

définition informelle:

NUMERO-PRODUIT: numéro du produit
DOC-PRODUITS: documentation caractéristiques des produits vendus par la société
doc-prod: documentation concernant les produits

étape 6:

Spécif-rel (Recherche-libellé-produit) [cf. page A.54]

étape 7:

Spécif-rel (Recherche-prix-produit) [cf. page A.56]

étape 8:

Spécif-rel (Recherche-montant-ligne) [cf. page A.59]

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.53]

[Remarque: doc-prod n'est pas défini par une relation intermédiaire (règle du avec) car il s'agit d'un argument du problème]

[Retour à la page A-48]

Spécification du problème: Def-ligne-fact

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Définition d'une ligne de facture **Données** ligne-fact: ligne de facturation ligne-livr: ligne de livraison doc-prod: documentation concernant les produits lib: libellé du produit vendu qté: quantité du produit facturé p-u: prix-unitaire de produit facturé mont: montant à payer pour le produit facturé **Relations** Recherche-libellé-produit: Recherche du libellé associé au numéro d'un produit livré Recherche-prix-produit: Recherche du prix unitaire associé au numéro d'un produit livré Calcul-montant-ligne: Calcul du montant facturé pour un produit livré	**Types** ligne-fact: LIGNE-FACTURE ligne-livr: LIGNE-LIVRAISON doc-prod: DOC-PRODUITS lib: LIBELLE-PRODUIT qté: QUANTITE-LIVREE p-u: PRIX-UNITAIRE mont: MONTANT-LIGNE **Relations** Def-ligne-fact: LIGNE-FACTURE <-- LIGNE-LIVRAISON, DOC-PRODUITS Recherche-libellé-produit: LIBELLE-PRODUIT <-- LIGNE-LIVRAISON, DOC-PRODUITS Recherche-prix-produit: PRIX-UNITAIRE <-- LIGNE-LIVRAISON, DOC-PRODUITS Calcul-montant-ligne: MONTANT-LIGNE <-- LIGNE-LIVRAISON, DOC-PRODUITS	**résultats:** ligne-fact **arguments:** ligne-livr, doc-prod **relation résultats- arguments** ligne-fact = (lib, qté, p-u, mont) lib = Recherche-libellé-produit (Num-prod:ligne-livr), doc-prod) et qté = Quantité-livrée (: ligne-livr) et p-u = Recherche-prix-produit (Num-prod:ligne-livr), doc-prod) et mont = Calcul-montant-ligne (Num-prod:ligne-livr), Quantité-livrée:ligne-livr), doc-prod)

RECHERCHE-LIBELLE-PRODUIT

lib = Recherche-libelle-produit (num-prod, doc-prod)

lib: LIBELLE-PRODUIT

num-prod: NUMERO-PRODUIT

doc-prod: DOC-PRODUITS

étape 1: Choix de la stratégie

Choix de la stratégie de réutilisation des connaissances

étape 2: Définition du type de la donnée.

définition formelle:

DOC-PRODUITS = TABLE [NUMERO-PRODUIT —› CARACTERISTIQUES-

-PRODUITS = PC[Lib-prod:LIBELLE-PRODUIT

Prix-prod: PRIX-PRODUIT]]

définition informelle:

CARACTERISTIQUES-PRODUIT: Caractéristiques d'un produit (à l'exception de son numéro)

étape 3: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

lib = Lib-prod(: Accès(doc-prod, num-prod))

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.55]

[Retour à la page A-51]

Spécification du problème: Recherche-libelle-produit

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Recherche du libellé associé au numéro d'un produit livré **Données** lib: libellé du produit facturé num-prod: numéro de produit livré doc-prod: documentation concernant les produits **Relations**	**Types** lib: LIBELLE-PRODUIT num-prod: NUMERO-PRODUIT doc-prod: DOC-PRODUITS **Relations**	résultats: lib arguments: num-prod, doc-prod relation résultats- arguments lib = Lib-prod (: Accès (doc-prod, num-prod))

RECHERCHE-PRIX-PRODUIT

p-u = Recherche-prix-produit (num-prod, doc-prod)

p-u: PRIX-UNITAIRE

num-prod: NUMERO-PRODUIT

doc-prod: DOC-PRODUITS

étape 1: Choix de la stratégie

Choix de la stratégie de réutilisation des connaissances

étape 2: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

p-u = Prix-prod (: Accès(doc-prod, num-prod))

[La spécification du niveau de traitement à ce stade se présente comme indiquée à la page A.57]

[Retour à la page A-52]

Spécification du problème: Recherche-prix-produit

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Recherche du prix-unitaire associé au numéro d'un produit livré Données p-u: prix unitaire du produit facturé num-prod: numéro du produit livré doc-prod: documentation concernant les produits Relations	Types p-u: PRIX-UNITAIRE num-prod: NUMERO-PRODUIT doc-prod: DOC-PRODUITS Relations	résultats: p-u arguments: num-prod, doc-prod relation résultats-arguments p-u = Prix-prod (: Accès (doc-prod, num-prod))

CALCUL-MONTANT-LIGNE

mont = Calcul-montant-ligne (num-prod, qté-livr, doc-prod)

num-prod: NUMERO-PRODUIT

qté-livr: QUANTITE-LIVREE

doc-prod: DOC-PRODUITS

mont: MONTANT-LIGNE

étape 1: Choix de la stratégie

Choix de la stratégie de réutilisation des connaissances (réutilisation de "Recherche-prix-produit")

étape 2: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

mont = qté-livr * Recherche-prix-produit (doc-prod, num-prod)

qté-livr: QUANTITE-LIVREE

définition informelle:

qté-livr: quantité livrée d'un produit

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.59]

[Retour à la page A-52]

Spécification du problème: Calcul-montant-ligne

Objectif
Calcul du montant facturé pour un produit livré

Données
mont: montant à payer pour la produit facturé
num-prod: numéro du produit livré
qté-livr: quantité livrée d'un produit commandé
doc-prod: documentation concernant les produits

Relations
Recherche-prix-produit: Recherche du prix-unitaire associé au numéro d'un produit livré

Types
mont: MONTANT-LIGNE
qté-livr: QUANTITE-LIVREE
num-prod: NUMERO-PRODUIT
doc-prod: DOC-PRODUITS

Relations
Recherche-prix-produit: PRIX-UNITAIRE
(— NUMERO-PRODUIT, DOC-PRODUITS

résultats: mont
arguments: num-prod, qté-livr, doc-prod

relation résultats= arguments

mont =
qté-livr * Recherche-prix-produit
(doc-prod, num-prod)

LEXIQUE | PROFIL | DESCRIPTION FORMELLE

DEF-TOTAL-A-PAYER

```
tot = Def-total-à-payer (corps-prem-livr, doc-prod)
          tot: TOTAL-A-PAYER
          corps-prem-livr: CORPS-PREMIERE-LIVRAISON
          doc-prod: DOC-PRODUITS
```

étape 1: Choix de la stratégie

Choix de la stratégie de réutilisation des connaissances (réutilisation de "Calcul-montant-ligne")

étape 2: Expression de la relation définissant la construction de chaque résultat.

définition formelle:

```
corps-prem-livr = ligne-livr*
tot = 40 +
     Total(Simple(ligne-livr*,
               Calcul-montant-ligne(Num-prod(:ligne-livr),
                                    Quantité-livrée(:ligne-livr)
                                    doc-prod)
          +)
```

[La spécification du pavé de traitement à ce stade se présente comme indiquée à la page A.61]

[Retour à la page A-44]

Spécialisation du problème: Def-total-à-payer

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Calcul du montant total à payer **Données** tot: total facturé corps-prem-livr: corps d'une première livraison doc-prod: documentation concernant les produits ligne-livr: ligne livraison **Relations**	**Types** tot: TOTAL-A-PAYER corps-prem-livr: CORPS-PREMIERE-LIVRAISON doc-prod: DOC-PRODUITS ligne-livr: LIGNE-LIVRAISON **Relations** Def-total-à-payer: TOTAL-A-PAYER (-- CORPS-PREMIERE-LIVRAISON, DOC-PRODUITS	**résultats:** tot **arguments:** corps-prem-livr, doc-prod **relation résultats- arguments** corps-prem-livr = ligne-livr* tot = 40 + Total (Simple (ligne-livr*, Calcul-montant-ligne (Num-prod(ligne-livr), Quantité-livrée(ligne-livr), doc-prod) +)

LIVRAISON-DIFFEREE

bool = Livraison-différée (corps-livr)

bool: BOOLEEN

corps-livr: CORPS-LIVRAISON

étape 1: Choix de la stratégie

Choix de la stratégie inductive basée sur la structure de l'argument

étape 2: Expression de la relation définissant le résultat.

Définition formelle:

Corps-livraison-différée-type (corps-livr)

[La spécification finale se présente comme indiquée à la page A.63]

[Remarque: On aurait également pu utiliser une stratégie de réutilisation de connaissance (en l'occurrence de "Première-livraison", cf. A.41): Livraison-différée (corps-livr)

= non Première-livraison (corps-livr)]

[Retour à la page A-38]

Spécification du problème: Livraison-différée

LEXIQUE	PROFIL	DESCRIPTION FORMELLE
Objectif Prédicat indiquant qu'il s'agit d'une livraison différée Données corps-livr: corps d'une livraison Relations	Types bool: BOOLEEN corps-livr: CORPS-LIVRAISON Relations Livraison-différée: BOOLEEN (-- CORPS-LIVRAISON	résultats: bool arguments: corps-livr relation résultats- arguments Corps-livraison-différée-type (corps-livr)

DEF-FACT-DIFFEREE

corps-fact-diff = Def-fact-différée (corps-livr-diff, doc-prod)

corps-fact-diff: CORPS-FACTURE-DIFFEREE

corps-livr-diff: CORPS-LIVRAISON-DIFFEREE

doc-prod: DOC-PRODUITS

étape 1: Choix de la stratégie

Choix de la stratégie de réutilisation des connaissances (réutilisation de "Def-ligne-fact", cf. A.50)

étape 2: Définition du type de la donnée

définition formelle:

CORPS-FACTURE-DIFFEREE = LIGNE-FACTURE

CORPS-LIVRAISON-DIFFEREE = LIGNE-LIVRAISON

étape 3: Expression de la relation définissant la construction du résultat

définition formelle

Def-ligne-fact (corps-livr-diff)

[La spécification finale se présente comme indiquée à la page A.65]

[Retour à la page A-38]

Spécification du problème:

Objectif
Définition du corps de la facture lorsqu'on envois en différé les produits suite à la commande

Données
corps-fact-diff: corps d'une facture différée
corps-livr-diff: corps d'une livraison différée
doc-prod: documentation concernant les produits

Relations
Def-ligne-fact: Définition d'une ligne de facture

LEXIQUE

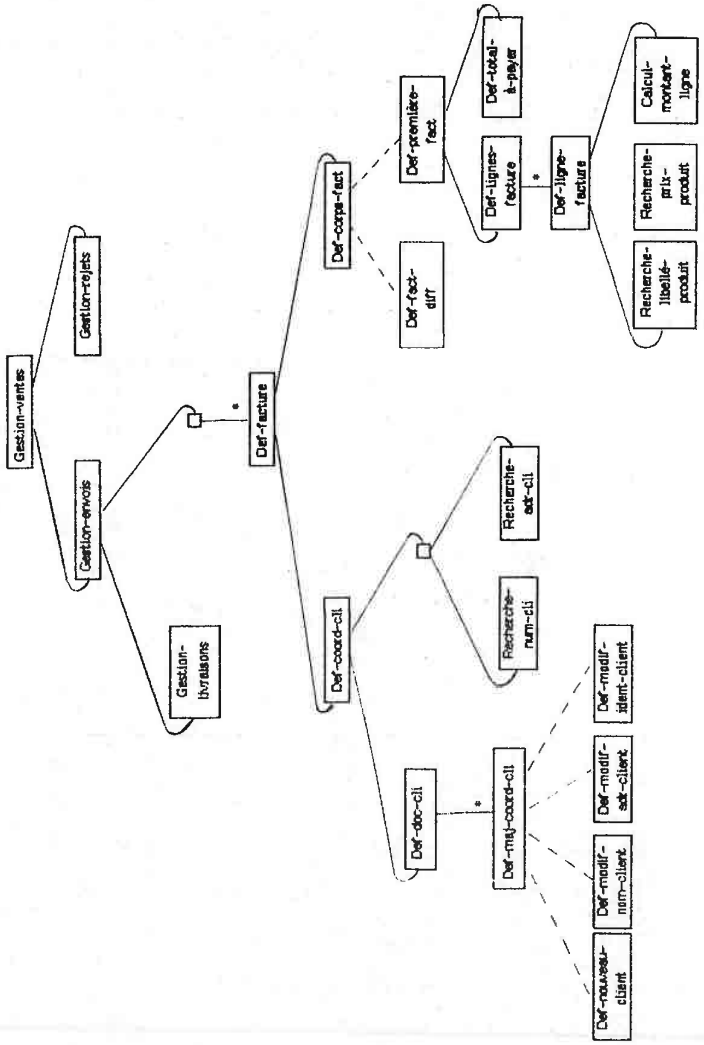
Types
corps-fact-diff: CORPS-FACTURE-DIFFEREE
= LIGNE-FACTURE
corps-livr-diff: CORPS-LIVRAISON-DIFFEREE
= LIGNE-LIVRAISON
doc-prod: DOC-PRODUITS

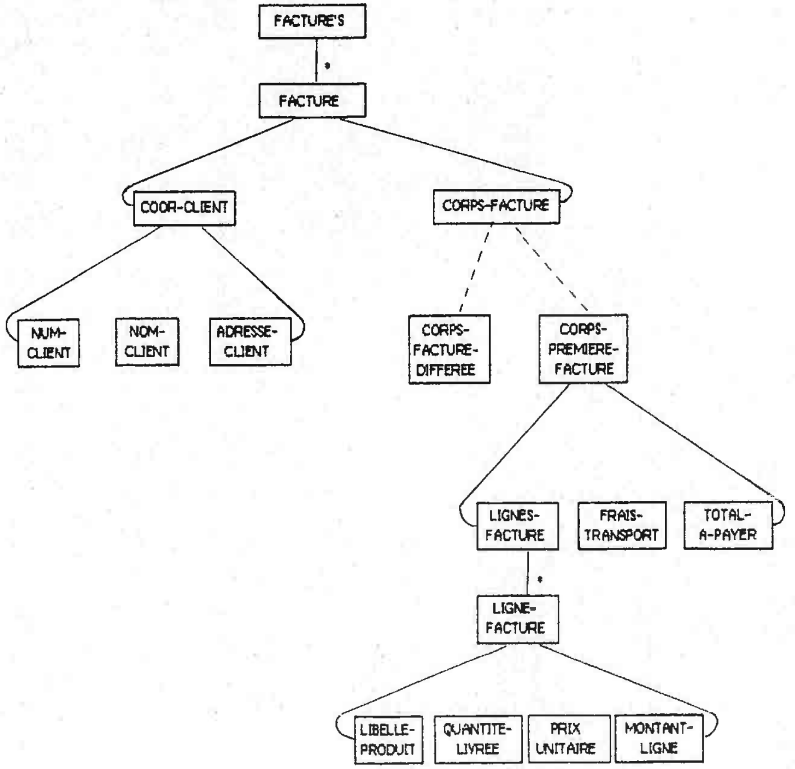
Relations
Def-fact-différée:
CORPS-FACTURE-DIFFEREE <--
CORPS-LIVRAISON-DIFFEREE,
DOC-PRODUITS
Def-ligne-fact:
LIGNE-FACTURE <--
LIGNE-LIVRAISON, DOC-PRODUITS

PROFIL

résultats: corps-fact-diff
arguments: corps-livr-diff, doc-prod
relation résultats- arguments
corps-fact-diff =
Def-ligne-fact (corps-livr-diff)

DESCRIPTION FORMELLE


Gestion-ventes
Gestion-rejets
Gestion-envois
Gestion-livraisons
Def-facture
Def-corps-fact
Def-premiere-fact
Def-total-a-payer
Def-lignes-facture
Def-ligne-facture
Calcul-montant-ligne
Recherche-prix-produit
Recherche-libelle-produit
Def-fact-diff
Def-coord-cli
Recherche-adr-cli
Recherche-num-cli
Def-doc-cli
Def-maj-coord-cli
Def-modif-ident-client
Def-modif-adr-client
Def-modif-nom-client
Def-nouveau-client


FACTURE'S
FACTURE
COOR-CLIENT
CORPS-FACTURE
NUM-CLIENT
NOM-CLIENT
ADRESSE-CLIENT
CORPS-FACTURE-DIFFEREE
CORPS-PREMIERE-FACTURE
LIGNES-FACTURE
FRAIS-TRANSPORT
TOTAL-A-PAYER
LIGNE-FACTURE
LIBELLE-PRODUIT
QUANTITE-LIVREE
PRIX UNITAIRE
MONTANT-LIGNE

Spécification du type: DOC-COOR-CLIENT

LEXIQUE

Types

DOC-COOR-CLIENT: il s'agit d'un répertoire contenant les caractéristiques d'identification des clients
RENS-CLIENT: renseignements caractérisant un client à l'exception de son numéro
COMMANDE'S: commandes émanant des clients de la société
NOM-CLIENT: nom du client
ADRESSE-CLIENT: adresse du client
COMMANDE: commande émanant d'un client de la société
NUM-CLIENT: numéro identifiant un client
MAJ-COORD-CLI: mise à jour des coordonnées d'un client à répercuter dans la documentation

Relations

Def-doc-cli: définition de la documentation concernant les identifications des clients
Recherche-nom-cli: recherche du nom du client dans la documentation "clients"
Recherche-adr-cli: recherche de l'adresse du client dans la documentation "clients"
Def-maj-coord-cli: Définition d'une modification des coordonnées du client à répercuter dans la documentation

PROFIL

Types

doc-cli, doc-cli: DOC-COOR-CLIENT
cmde*: COMMANDE'S
cmde: COMMANDE
num-cli-livr: NUM-CLIENT
nom-cli-fact: NOM-CLIENT
adr-cli-fact: ADRESSE-CLIENT

Relations

Def-doc-cli: DOC-COOR-CLIENT <-- COMMANDE'S

Def-maj-coord-cli: MAJ-COORD-CLI <-- COMMANDE, DOC-COOR-CLIENT

Recherche-nom-cli: NOM-CLIENT <-- DOC-COOR-CLIENT, NUM-CLIENT

Recherche-adr-cli: ADRESSE-CLIENT <-- DOC-COOR-CLIENT, NUM-CLIENT

DESCRIPTION FORMELLE

DOC-COOR-CLIENT =
TABLE [NUM-CLIENT -->
RENS-CLIENT =
PC [Nom-client: NOM-CLIENT
Adresse-client: ADRESSE-CLIENT]]

Invariant

Opérations

Création et mise à jour.

Def-doc-cli (cmde*) doc-cli
tq
Tabulation(Simple(cmde*, Def-maj-coord-cli(cmde,doc-cli)))

Consultation.

Recherche-nom-cli (doc-cli,num-cli-livr) nom-cli-fact
tq
Nom-client (: Accès (doc-cli, num-cli-livr))

Recherche-adr-cli (doc-cli,num-cli-livr) adr-cli-fact
tq
Adresse-client (: Accès (doc-cli, num-cli-livr))

BIBLIOGRAPHIE

BIBLIOGRAPHIE.

[ABR, 74] ABRIAL, J.R., *Data Semantics*, In: "Data Base Management", North-Holland, 1974.

[ABR, 78] ABRIAL, J.R., *Z: A Specification Language*, IFIP, Tokyo, 1978.

[ABR, 80] ABRIAL, J.R., *The Specification Language Z: Syntax and Semantics*, Programming Research Group, Oxford Univ., 1980.

[ADJ, 78] GOGUEN, J.A., THATCHER, J.W. et E.G. WAGNER, *An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types*, Current Trends in Programming Methodology, vol. 4 (Ed. Yeh), Prentice Hall, pp. 80-144.

[ADJ, 82] THATCHER, J.W., WAGNER, E.G. et J.B. WRIGHT, *Data Type Specification: Parametrization and the Power of Specification Techniques*, ACM TODS, vol.4, 4, 1982, pp. 711-732.

[ALF, 77] ALFORD, M.W., *A Requirement Engineering Methodology for Real Time Processing Environments*, IEEE Trans. Soft. Eng., vol. SE-3, 1977, pp. 60-69.

[ANS, 75] ANSI/X3/SPARC, *Study Group on Data Base Management Systems*, Interim Report, 1975.

[ATZ, 82] ATZENI, P., BATINI, C., DE ANTONELLIS, V., LENZERINI, M., VILANELLI, F. et B. ZONTA, *A Computer Aided Tool for Conceptual Data Base Design*, In: Automated Tools for Information Systems Design, H.J. Schneider et A.I. Wasserman (Ed.), North-Holland, 1982, pp. 85-105.

[BAL, 77] BALZER, R., GOLDMAN, N. et D. WILE, *Informality in Program Specifications*, Proc. Fith. Int. J. Conf. Artif. Intel., Cambridge, Mass., 1977, pp. 389-397.

[BAT, 83] BATINI, C. et M. LENZERINI, *A Conceptual Foundation for View Integration*, IFIP TC-2 "System Description Methodologies", Kecskemet, Hongrie, 1983, pp. 109-140.

[BAU, 78] BAUER, F.L., PEPPER, P. et H. WÖSSNER, *Notes on the Project CIP: Outline of a Transformation System*, In "Program Construction", F.L. Bauer et M. Broy (éd.), Lectures Notes in Computer Science 69, Springer Verlag, 1978.

[BAU, 79] BAUER, F.L. et H. WÖSSNER, *Algorithmic Language and Program Development*, London, Prentice Hall, 1979.

[BAU, 82] BAUER, F.L., *From Specifications to Machine Code: Program Construction through Formal Reasoning*, Sixth International Conference on Software Engineering, Tokyo, 1982, pp. 84-91.

[BEN, 76] BENCI, G., BODART, F., BOGAERT, H. et A. CABANES, *Concepts for the Design of a Conceptual Scheme*, Proc. IFIP TC2 WC Freundenstadt, 1976.

[BID, 81] BIDOIT, M., *Une méthode de présentation des types abstraits:*

applications, Thèse de 3ième cycle, Paris-Sud, 1981.

[BOD, 79] BODART, F. et Y. PIGNEUR, *A Model and a Language for Functional Specifications and Evaluation of Information System Dynamics*, IFIP, TC-8, "Formal Models and Practical Tools for Information System Design", North-Holland, 1979.

[BOD, 83] BODART, F. et Y. PIGNEUR, *Conception assistée des applications informatiques. Première partie: Etude d'opportunité et analyse conceptuelle*, Masson (ed.), Paris, 1983.

[BOE, 76] BOEHM, B.W., *Software Engineering*, IEEE Transactions on Computer, vol. 25, 12, 1976, pp. 126-143.

[BOL, 83] BOLEY, H., *Artificial Intelligence Languages and Machines*, Technique et Science Informatique, vol. 2, 3, 1983, pp. 145--166.

[BRO, 81] BRODIE, M.L., *On Modelling Behavorial Semantics of Databases*, Proc. 81 International Conference on Very Large Databases, Cannes, France, September, 1982.

[BRO, 82] BRODIE, M.L. et E. SILVA, *Active and Passive Component Modelling: ACM/PCM*, In: Information Systems Design Methodologies: A Comparative Review, T.W. Olle, H.G. Sol et A.A. Verrijn-Stuart (Ed.), North-Holland, 1982, pp. 41-91.

[CAI, 75] CAINE, S.H., *FOL: A Tool for Software Design*, National Computer Conference, 1975, pp. 271-276.

[CHA, 77] CHABANNE, F., *Adaptation des concepts de la programmation structurée à un projet d'entreprise*, Thèse de troisième cycle, UPS, Toulouse, 1977.

[CHA, 82] CHABRIER, J-J., *Spécification et Construction de systèmes orientés bases de données: Techniques et langages basés sur le concept de type abstrait*, Thèse d'Etat, Nancy-1, 1982.

[CHE, 76] CHEN, P.P., *The Entity-Relationship Model: Toward a Unified View of Data*, ACM TODS 1, 1, 1976, pp. 9-36.

[COD, 70] CODD, E.F., *A Relationnal Model of Data for Large Shared Data Banks*, CACM, vol.13, 6, 1970.

[COD, 71] CODD, E.F., *A Data Base Sublanguage formed on the Relationnal Calculus*, Proc. ACMSIGDIDET Workshop on Data Description, access and control, 1971.

[COL, 83] COLMERAUER, A., KANOUI, H. et M. VAN CANEGHEM, *Prolog, bases théoriques et développements actuels*, Technique et Science de l'Informatique, vol. 2,4, 1983, pp. 271-311.

[DAH, 72] DAHL, O.J., DIJKSTRA, E.W. et C.A.R. HOARE, *Structured Programming*, Academic Press, 1972.

[DAT, 77] DATE, C.J., *An Introduction to Data Base Systems*, Second Edition, Addison-Wesley, 1977.

[DAR, 79] DARLINGTON, J. et M. FEATHER, *A Tranformational Approach to Modification*, Dept. of Computing and Control, Imperial College, London, Feb., 1979.

[DEJ, 75] DE JONG, S.P. et M. ZLOOF, *Application Design within the System for Business Automation (SBA)*, Proc. 12th Design Automation Conference, ACM-IEEE, Boston, 1975, pp. 69-76.

[DEL, 80] DELOBEL, C., *An Overview of the Relationnal Data Theory*, Proc. of the IFIP Conference Tokyo, 1980.

[DEL, 82] DELISLE, N.M., MENICOSY, D.E. et N.L. KERTH, *Tools for Supporting Structured Analysis*, In: "Automated Tools for Information Systems Design", H.J. Schneider et A.I. Wasserman (eds), North-holland, 1982, pp. 11-20.

[DEM, 80] DEMUYNCK, M., CHENUT, S. et J-M. NERSON, *Système Zaide d'aide à la spécification*, Actes du congrès Afcet: Théorie et Techniques de l'Informatique, 1980, pp. 179-188.

[DER, 79] DERNIAME, J-C. et J-P. FINANCE, *Types abstraits de données: Spécification, Utilisation et Réalisation*, Cours de l'Ecole d'Eté de l'AFCET, Monastir, 1979.

[DIJ, 68] DIJKSTRA, E.W., *The Structure of the "THE"- Multiprogramming System*, Comm. Ass. Comput. Mach., vol. 11, 1968, pp. 341-346.

[DIJ, 76] DIJKSTRA, E.W., *A Discipline of Programming*, Prentice-Hall, 1976.

[DON, 80] DONZEAU-GOUGE, V., HUET, G., KAHN, G. et B. LANG, *Programming Environment based on Structured Editors*, Research Report no. 26, INRIA, Rocquencourt, 1980.

[DUB, 81] DUBOIS, E., *Une approche déductive à la spécification et au développment d'une application de gestion en temps réel*, Mémoire de fin d'Etudes, Institut d'Informatique, Namur, 1981.

[DUB, 82] DUBOIS, E., FINANCE, J-P. et A. VAN LAMSWEERDE, *Towards a Deductive Approach to Information System Specification and Design*, In: International Symposium of Current Issues of Requirements Engineering Environments, Y. Ohno (Ed.), North-Holland, 1982, pp. 23-31.

[DUB, 83] DUBOIS, E., FINANCE, J-P., LEVY, N. et A. VAN LAMSWEERDE, *Specification Techniques for Large Information Systems*, Symposium IBM-FNRS, Bruxelles, 1983, pp. 9-19.

[DUF, 80] DUFOURD, J-F., *Maquettes pour évaluer les systèmes d'information dans les organisations*, Thèse d'Etat, Nancy-1, 1980.

[FEA, 80] FEATHER, M.S., *Formal Specification of Real System*, USC/ Information Sciences Institute, 1980.

[FIN, 79] FINANCE, J-P., *Etude de la construction des programmes: méthodes et langages de spécification et de résolution de problèmes*, Thèse d'Etat, Université de Nancy-1, 1979.

[FIN, 83] FINANCE, J-P., GRANDBASTIEN, M., LEVY, N., QUERE, A. et J. SOUQUIERES, *SPES: Un système pour spécifier et transformer*, Centre de Recherche en Informatique de Nancy, Rapport 83-R-079, 1983.

[FLO, 67] FLOYD, R.W., *Assigning Meanings to Programs*, Proceedings of

the Symposium in Applied Mathematics, Mathematical Aspect of Computer Science, J.T. Schwartz (ed), American Mathematical Society, 1967, pp. 19-32.

[FOI, 81] FOISSEAU, J., JACQUART, R., LEMAITRE, M. et G. ZANON, *SFRAC: a Computer Assisted Software Development System*, In: "Tools and Notions for Program Construction", INRIA-CCE-CREST, Nice, 1981.

[FOI, 82] FOISSEAU, J., *Assistance à la spécification des fonctions et des types dans SFRAC*, Actes des journées BIGRE, Grenoble, 1982.

[FOU, 82] FOUCAUT, O., *Modèle et outil pour la conception des systèmes d'information dans les organisations: Projet REMORA*, Thèse d'Etat, Université de Nancy-1, 1982.

[GAL, 73] GALAN, I., *Les tables de décision et leurs applications*, Dunod, Paris, 1973.

[GAU, 80] GAUDEL, M-C., *Génération et preuve de compilateurs basées sur une sémantique formelle des langages de programmation*, Thèse d'Etat, Paris 6, 1980.

[GOL, 83] GOLDBERG, A. et D. ROBSON, *SMALLTALK-80: The Language and its implementation*, Addison-Wesley, 1983.

[GRE, 76] GREEN, C., *The Design of the PSI: Program Synthesis System*, Second International Conference on Software Engineering, Long Beach, Ca., 1976, pp. 4-48.

[GRE, 82] GREESPAN, S.J. et D. MYLOPOULOS, *Capturing More World Knowledge in the Requirements Specification*, Sixth International Conference on Software Engineering, Tokyo, 1982, pp. 225-234.

[GUI, 80] GUIHO, G., GRESSE C. et M. BIDOIT, *Conception et Certification de Programmes à partir d'une Décomposition par les données*, RAIRO Informatique, vol.14, 4, 1980, pp. 319-351.

[GUT, 77] GUTTAG, J.V., *Abstract Data Types and the Development of Data Structures*, CACM, vol. 20, 6, 1977, pp. 396-404.

[HAM, 76] HAMILTON, M. et S. ZELDIN, *High Order Software: A Methodology for Design Software*, IEEE Trans. Soft. Eng., vol. 2, 1, 1976, pp. 9-32.

[HAM, 77] HAMMER, M., HOWE, W.G., KRUSKAL, V.J. et I. WLADAWSKY, *A Very High Level Programming Language for Data Processing Applications*, CACM, vol. 20, 11, 1977, pp. 832-840.

[HEN, 80] HENINGER, K.L., *Specifying Software Requirements for Complex Systems: Techniques and their Application*, IEEE Trans. Soft. Eng., vol. SE-6, 1, 1980, pp. 2-13.

[HOA, 69] HOARE, C.A.R., *An Axiomatic Basis for Computer Programming*, Comm. ACM, vol.12, 10, 1969, pp. 576-583.

[HOW, 75] HOWE, W.G., KRUSKAL, V.J., LEAVENWORTH, B.H., LEWIS, C. et I. WLADAWSKY, *The Preliminary Definition of the Document Flow Component of the Business Definition Language*, IBM Res.

Rep. RC5204, Yorktown Heigts, N.Y., Jan, 1975.

[HOW, 75b] HOWE, W.G., KRUSKAL, V.J. and I. WLADAWSKY, *A New Approach for Customizing Business Application*, IBM Res. Rep. RC5744, Yorktown Heights, N.Y., 1975.

[ISO, 81] ISO TC 97/SC5/WG3, *Preliminary Reports Concepts and Terminology for the Conceptual Schema*, Ed. JJ Van Griethuysen, 1981.

[JAC, 75] JACKSON, M.A., *Principles of Program Design*, Academic Press, 1975.

[JAC, 83] JACKSON, M.A., *System Development*, Prentice-Hall, International Series on Computer Science, 1983.

[KAT, 76] KATZEN, H., *System Design and Documentation: An Introduction to the HIPO Method*, Van Nostrand, Rheinhold, N.Y., 1976.

[LAM, 82] VAN LAMSWEERDE, A., *Les outils d'aide au développement de logiciels: un aperçu des tendances actuelles*, 15ième Journées Internationales de l'Informatique et de l'Automatisme, Paris, Juin, 1982.

[LAM, 83] VAN LAMSWEERDE, A., *Automatisation de la production de logiciels d'application: quelques approches - deuxième partie*, Technique et Science Informatique, vol. 2, 1, 1983, pp. 5-20.

[LEM, 76] LEMOIGNE, J-L., *Les systèmes d'information des organisations*, éd. PUF, 1976.

[LER, 82] LEROY, F. et P. ROELLY, *GISTEC: Rapport Final, Rapport Interne Elf Aquitaine*, Août 1982.

[LEV, 84] LEVY, N., Thèse de Troisième Cycle en Informatique, A paraître, Université de Nancy-1, 1984.

[LIS, 75] LISKOV, B.H. et S.N. ZILLES, *Specification Techniques for Data Abstraction*, I.E.E.E. Trans. Soft. Eng., SE-1, 1975, pp. 1-9.

[LIS, 83] LISSANDRE M., LAGIER, P. et A. SKALLI, *SAS: A Specification Support System*, IFIP TC-2 "System Description Methodologies", Kecskemet, Hongrie, 1983, pp. 141-166.

[LIV, 78] LIVERCY, C., *Théorie des Programmes*, Dunod (éd), 1978.

[LUN, 83] LUNDBERG, B., *On Relative Strengh of Information Model*, IFIP TC-2 "System Description Methodologies", Kecskemet, Hongrie, 1983, pp. 5-20.

[MAL, 71] MALLET, R.A., *La Méthode Informatique*, Hermann, 1971.

[MAL, 75] MALHOTRA, A., *Design Criteria for a Knowledge-based English Language System for Management: an Experimental Analysis*, Report MAC TR-146, MIT, Laboratory for Computer Science, Cambridge, Mass., 1975.

[MAR, 79] DE MARCO, T., *Structured Analysis and System Specification*, A Yourdon Book, Prentice Hall, Englewood Cliffs, 1979.

[MAS, 82] MASON, R.E.A. et T.T. CAREY, *An Approach to Prototyping*

Interactive Information Systems, Proc. 3rd International Conference on Information Systems, Ann Arbor, Michigan, December, 1982.

[MED, 81] MEDINA-MORA, R. et P. FEILER, *An Incremental Programming Environment*, IEEE Trans. Soft. Eng., vol. SE-7, 1981, pp. 472-481.

[MEY, 78] MEYER, B. et C. BAUDOIN, *Méthodes de Programmation*, EYROLLES, Collection DER-EDF, 1978.

[MEY, 80] MEYER, B., *Sur le formalisme des spécifications*, Globule, Bulletin du Groupe de Travail "Génie Logiciel" de l'AFCET, 1980.

[MIL, 56] MILLER, G.A., *The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information*, Psychol. Rev, vol. 63, 1956, pp. 81-97.

[MIT, 82] MITTERMEIR, R.T., HSIA, P. et R. YEH, *Alternatives to Overcome the Communications Problem of Formal Requirements Analysis*, In: International Symposium of Current Issues of Requirements Software Environments, Y. Ohno (Ed.), North-Holland, 1982, pp. 163-169.

[NIL, 80] NILSON, N., *Principles of Artificial Intelligence*, Tioga Pub. Co., 1980.

[PAI, 79] PAIR, C., *La Construction des Programmes*, RAIRO Informatique, vol. 13, 2, 1979, pp. 113-118.

[PAR, 74] PARNAS, D.L., *On a Buzzword: Hierarchical Structure*, Proc. IFIP, North-Holland Publ., 1974, pp. 336-339.

[PET, 76] PETRI, C.A., *General Net Theory*, Computing System Design, Proc. of the Joint IBM-University of Newcastle-upon-Tyne Seminar, September, 1976.

[PET, 77] PETERSON, J.L., *Petri Nets*, Computing Surveys, vol. 9, 3, September, 1977, pp. 223-252.

[POL, 71] POLLACK, I., HICKS, H. et W. HARRISON, *Decision Tables: Theory and Practice*, Wiley, New-York, 1971.

[PRY, 77] PRYWES, N.S., *Automatic Generation of Computer Programs*, NCC 1977, Afips Press, Montvale, N.J., pp. 679-689.

[ROL, 82] ROLLAND, C. et C. RICHARD, *The RENORA Methodology for Information Systems Design and Management*, IFIP TC8 Conference on Comparative Review of Information Systems Methodologies, Amsterdam, 1982.

[ROS, 77] ROSS, D.T. et K.G. SCHOMAN, *Structured Analysis for Requirements Definition*, I.E.E.E. Trans. Soft. Eng., SE-3 (1), 1977, pp. 1-65.

[RUT, 81] RUTH, G., ALTER, S. et W. MARTIN, *A Very High Level Language for Business Data Processing*, Report MIT/LCS/TR-254, Mass. Inst. Techn., 1981.

[SCH, 79] SCHOLL, P.C., *Vers une programmation systématique: étude de*

quelques méthodes, techniques et outils, Thèse d'Etat, USMG et INPG, Grenoble, 1979.

[SCH, 83] SCHERLIS, L.W. et D. SCOTT, *First Steps Towards Inferential Programming*, Ifip, Mason (Ed.), 1983, pp. 199-211.

[SEN, 75] SENKO, M.E., *Data Description Language in the Context of a Multilevel Structured Description: Diam 2 with Foral*, Data Base Description, Proceedings of the IFIP TC-2 Special Working Conference on Data Description, Nijssen (ed.), North-Holland, 1975, pp. 239-258.

[SIM, 74] SIMON, H.A., *La Science des Systèmes: Science de l'artificiel* EPI, 1974.

[SIN, 79] SINTZOFF, M., *Composing and Specifying Program Design Decisions*, Working Notes (Communication Privée), 1979.

[SMI, 77] SMITH, J-M. et D.C.P. SMITH, *Database Abstractions: Agregation and Generalization*, Trans. on Database Systems, vol. 2, 2, 1977, pp. 105-133.

[STE, 74] STEVENS, W., MYERS, G. et L. CONSTANTINE, *Structured Design*, IBM System Journal, vol. 13, 2, 1974, pp. 115-139.

[SUF, 81] SUFRIN, B., *Formal System Specifications: Notation and Examples*, In: "Tools and Notion for Program Construction", Lectures Notes, INRIA-CEE-CREST, Nice, 1981.

[SUF, 82] SUFRIN, B., *Formal Specification of a Display Oriented Text Editor*, Science of Computer Programming, Vol. 1,3, 1982, pp. 157-202.

[SWA, 82] SWARTOUT, W. et R. BALZER, *On the Inevitable Intertwinning of Specification and Implementation*, Comm. ACM, vol. 25,7, 1982, pp. 438-440.

[TAB, 81] TABOURIER, Y. et D. NANCI, *The Occurrences Structure Concept*, GAMMA INTERNATIONAL-CECIMA, 1981.

[TAR, 79] TARDIEU, H., NANCI, D. et D. PASCOT, *Conception d'un système d'information*, Les éditions d'organisation, 1979.

[TAR, 83] TARDIEU, H., ROCHFELD, A. et R. COLETTI, *La Méthode MERISE: Principes et Outils*, Les éditions d'Organisation, 1983.

[TEI, 77] TEICHROEW, D. et E.A. HERSHEY, *PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems*, I.E.E.E. Trans. Soft. Eng., SE-3 (1), 1977, pp. 41-48.

[WAR, 75] WARNIER, J.D., *Les procédures de traitement et leurs données*, Editions d'Organisation, 1975.

[WAS, 82] WASSERMANN, A.I., *Automated Tools in the Information System Development*, In: "Automated Tools for Information Systems Design", H.J. Schneider et A.I. Wassermann (eds), North-Holland, 1982, pp. 1-10.

[WEI, 78] WEINBERG, V., *Structured Analysis*, Yourdon Press, 1978.

[WIL, 82] WILE, D.S., *Program Developments: Formal Explanations of*

Implementations, USC/Information Science Institute, Report ISI/RR-82-99, 1982.

[WIR, 73] WIRTH, N., *Systematic Programming: an Introduction*, Prentice-Hall Series in Automatic Computation, 1973.

[WUL, 75] WULF, W.A., LONDON, R.L. et M. SHAW, *Abstraction and Verification in ALPHARD*, In: New Directions in Algorithmic Languages, S.A. SCHUMAN (ed), IRIA, 1975.

[ZLO, 77] ZLOOF, M.M. et S.P. DE JONG, *The System for Business Automation (SBA): Programming Language*, CACM, vol. 20, 6, 1977, pp. 385-396.